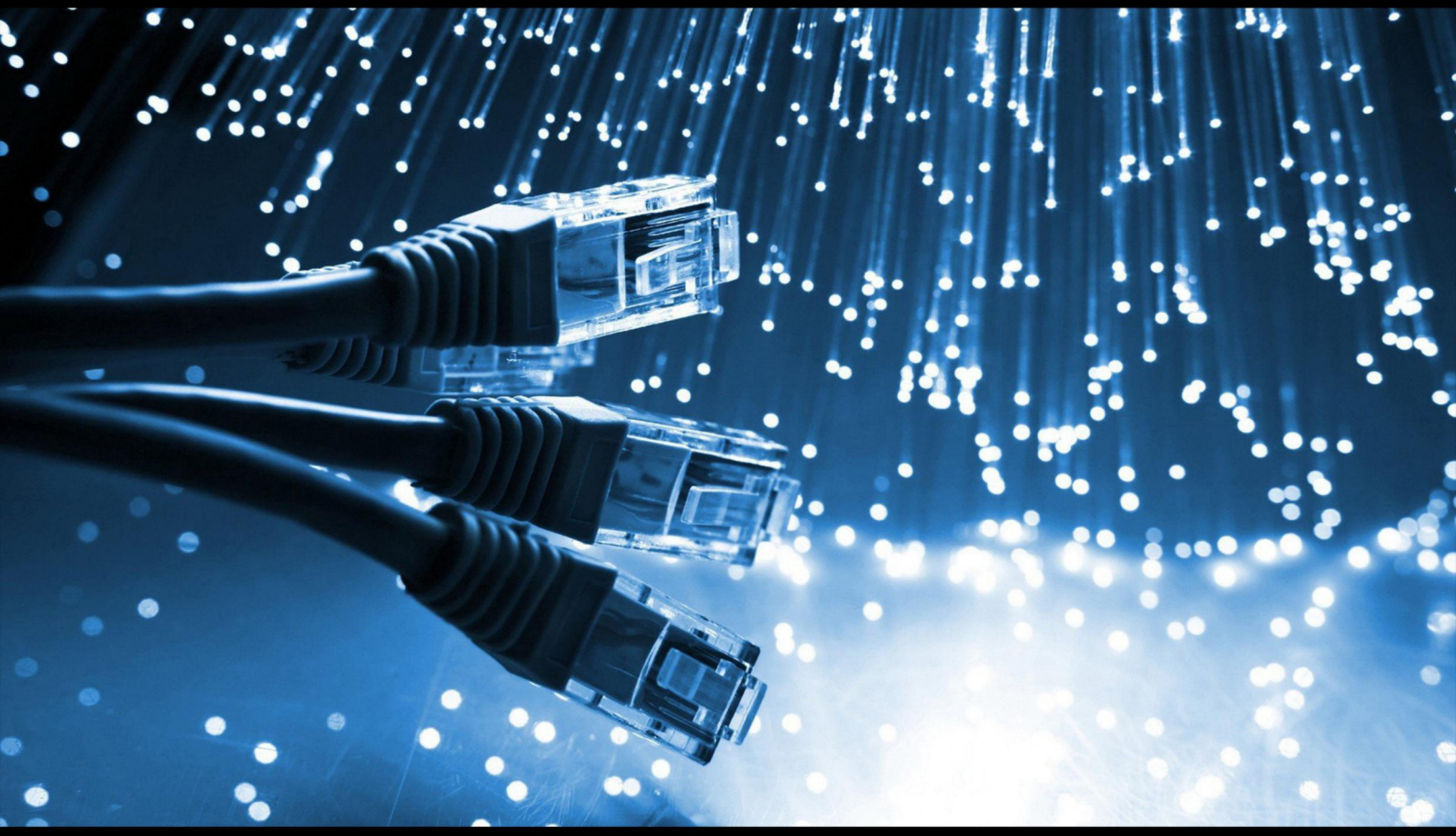
JCSI

Journal of Computer Sciences Institute

Volume 29/2023



Department of Computer Science
Lublin University of Technology

jcsi.pollub.pl

ISSN: 2544-0764

Redakcja JCSI
e-mail: jcsi@pollub.pl
www: jcsi.pollub.pl
Katedra Informatyki
Wydział Elektrotechniki i Informatyki

Politechnika Lubelska
ul. Nadbystrzycka 36 b
20-618 Lublin

Redaktor naczelny:
Tomasz Zientarski
e-mail: t.zientarski@pollub.pl

Redaktor techniczny:
Beata Pańczyk,
e-mail: b.panczyk@pollub.pl

Recenzenci numeru:

dr inż. Maria Skublewska-Paszkowska
dr inż. Marek Miłosz, prof. uczelni
dr inż. Jakub Smołka
dr inż. Marcin Badurowicz
dr inż. Jacek Kęsik
dr inż. Piotr Kopniak
dr inż. Grzegorz Kozieł
dr Paweł Powroźnik
dr inż. Sławomir Przyłucki
dr inż. Maciej Pańczyk

Skład komputerowy:
Anna Sałamacha
e-mail: a.salamacha@pollub.pl

Projekt okładki:
Marta Zbańska

JCSI Editorial
e-mail: jcsi@pollub.pl
www: jcsi.pollub.pl
Department of Computer Science
Faculty of Electrical Engineering and Computer Science
Lublin University of Technology
ul. Nadbystrzycka 36 b
20-618 Lublin, Poland

Editor in Chief:
Tomasz Zientarski
e-mail: t.zientarski@pollub.pl

Assistant editor:
Beata Pańczyk,
e-mail: b.panczyk@pollub.pl

Reviewers:

Maria Skublewska-Paszkowska
Marek Miłosz
Jakub Smołka
Marcin Badurowicz
Jacek Kęsik
Piotr Kopniak
Grzegorz Kozieł
Paweł Powroźnik
Sławomir Przyłucki
Maciej Pańczyk

Computer typesetting:
Anna Sałamacha
e-mail: a.salamacha@pollub.pl

Cover design:
Marta Zbańska

ISSN 2544-0764

Spis treści

Contents

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 304–311
Received: 21 June 2023
Accepted: 29 August 2023

Performance analysis of web applications created in the Spring and Laravel frameworks

Analiza wydajnościowa aplikacji internetowych utworzonych w szkieletach Spring i Laravel

Jakub Suchanowski*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The article compares the performance of two test applications created using the Spring Boot and Laravel application frameworks. The aim of the study is to find an answer to the research question: which framework offers better time performance. Twelve test scenarios were created for the analysis, and they were used to conduct performance tests using the Apache JMeter tool. Additionally, application metrics and community support for both frameworks were compared. The research confirmed that the application built on the Spring Boot framework has better performance.

Keywords: spring boot; laravel; web applications; application performance

Streszczenie
W artykule porównano wydajność dwóch aplikacji testowych, które stworzono używając szkieletów aplikacji Spring Boot i Laravel. Celem pracy jest uzyskanie odpowiedzi na pytanie badawcze: który szkielet oferuje lepszą wydajność czasową. Do analizy stworzono 12 scenariuszy testowych, a następnie na ich podstawie przeprowadzono testy używając narzędzia Apache JMeter. Dodatkowo porównane zostały metryki aplikacji oraz wsparcie społeczności dla obu szkieletów. Badania potwierdziły, że aplikacja zbudowana na szkielecie Spring Boot ma lepszą wydajność.

Słowa kluczowe: spring boot; laravel; aplikacje internetowe; wydajność aplikacji

*Corresponding author
Email address: jakub.suchanowski@pollub.edu.pl (J. Suchanowski)

1. Wstęp

W dzisiejszych czasach aplikacje internetowe pełnią ważną rolę w różnych dziedzinach, takich jak media społecznościowe, bankowość, dostęp do informacji, handel internetowy. Wraz z dynamicznym rozwojem technologii internetowych i rosnącymi oczekiwaniami użytkowników, tworzenie wydajnych aplikacji internetowych stało się niezwykle istotne. Zapewnienie odpowiedniej wydajności w dużej mierze wpływa na satysfakcję użytkowników i tym samym na pozytywny odbiór przekazywanych treści [1]. Z tego powodu programiści tworzący aplikacje internetowe muszą odpowiednio dobrać używane technologie i zadbać o zoptymalizowanie ich pod kątem wydajności.

Istnieje wiele artykułów porównujących wydajności aplikacji internetowych zbudowanych na poszczególnych szkieletach aplikacji (ang. framework). Najczęściej autorzy skupiali się na porównaniu aplikacji stworzonych za pomocą jednego języka programowania. W tym artykule porównane zostały aplikacje na bazie jednych z najpopularniejszych szkieletów z dwóch różnych języków programowania. Podobne porównanie stworzył Jakub Radomski [2]. Autor przedstawił wyniki wydajności dwóch aplikacji internetowych stworzonych na bazie Javy i Vaadin oraz PHP i Laravel. Dla obu aplikacji przeprowadził testy wydajności czasowej tworząc 13 scenariuszy, w których za pomocą narzędzia Chrome DevTools oraz ApacheBench mierzył czasy trwania operacji CRUD (Create, Read, Update i Delete) na zbiorach danych. Badania pokazały, że aplikacja na bazie szkieletu Vaadin jest znacznie wydajniejsza, niż aplikacja zbudowana za pomocą platformy Laravel. Największe różnice widoczne były przy zapisie i odczycie większej liczby rekordów oraz przy symulacji pobierania zasobów dla większej liczby użytkowników. W pracy pt. „Porównanie wytwarzania oprogramowania internetowego z wykorzystaniem różnych technologii" [3] porównano dwie aplikacje internetowe stworzone na bazie PHP i Laravel oraz Javy i Spring. Artykuł przedstawia porównanie tych technologii pod względem pracy z bazą danych. We wnioskach autor stwierdził, że przy dodawaniu nowego rekordu do bazy danych szybszy jest Spring. Przy odczytywaniu pojedynczych rekordów z bazy danych oraz obliczaniu n-tego Ciągu Fibonacciego również lepiej spisuje się Spring. Jedynie przy usuwaniu pojedynczego rekordu z bazy danych aplikacja stworzona na bazie szkieletu Laravel osiąga mniejszy czas. W artykule [4] autorzy stworzyli dwie aplikacje internetowe używając języka Java oraz PHP. Miały one na celu automatyzację manualnego systemu oceny nauczycieli na ich uniwersytecie. Aplikacje zawierały pytania, na które odpowiadali studenci, następnie po przesłaniu odpowiedzi na serwer, generowano raporty. Głównym celem pracy był pomiar wydajności aplikacji. Po wykonaniu pomiarów parametrów wydajności stwierdzono, że aplikacja zbudowana w języku Java reaguje szybciej niż aplikacja zbudowana w języku PHP oraz zawiera niższe wartości odchylenia standar-

dowego. Z kolei w badaniu pt. „A Performance Comparison of RESTful Applications Implemented in Spring Boot Java and MS.NET Core" [5] autor skupił się na porównaniu oraz ocenie wydajności aplikacji internetowych opartych na RESTful. Aplikacje zaimplementowano używając szkieletów Spring Boot oraz MS.NET Core. Aplikacje testowe obsługiwały podstawowe operacje Create, Read, Update i Delete (CRUD). Do analizy wydajności użyto narzędzia Apache JMeter. Przeprowadzone badania pozwoliły autorowi stwierdzić, że MS.NET Core zapewnia szybszy czas odpowiedzi oraz zużywa mniej zasobów w porównaniu do szkieletu Spring Boot we wszystkich metodach http. Wpływ języka na wydajność REST Api zbadano również w artykule [6]. Autorzy porównali trzy języki programowania: JavaScript, Java oraz Python. Na podstawie wyników, które uzyskali, stwierdzili, że dla małych i średnich aplikacji najlepszą wydajność osiągnął język JavaScript, natomiast dla aplikacji z dużą ilością danych najlepszym wyborem okazał się język Java. W kolejnej pracy [7] autorzy porównywali szkielety aplikacji Laravel oraz Yii2 języka PHP. Badanie przeprowadzono pod względem czasu wykonania, użycia pamięci oraz wydajności. Do testów posłużono się dużą bazą danych, aby sprawdzić jak badane szkielety aplikacji zachowują się przy pracy z nimi. Po opracowaniu wyników autorzy stwierdzili, że Laravel dominował pod względem czasu wykonania oraz wydajności, gorzej spisał się jedynie w zakresie użycia pamięci. Autorzy badania [8] porównującego trzy szkielety aplikacji PHP (Laravel, Symfony, CodeIgniter) uznali, że aplikacja zbudowana za pomocą szkieletu Laravel ma najlepszą wydajność. Podobne wnioski można zauważyć w kolejnym artykule [9], w którym porównano cztery szkielety aplikacji języka PHP. W artykule [10], gdzie porównano wydajność szkieletu Laravel ze szkieletem CodeIginiter, wyniki były różne pod względem czasu i szybkości działania. CodeIgniter osiągnął dłuższy czas działania niż Laravel oraz wyższy wynik w badaniu średniej szybkości. Natomiast w porównaniu [11] wydajności aplikacji na bazie szkieletów Laravel i Slim ten drugi wypadł znacznie lepiej. W przypadku porównania [12] szkieletu aplikacji Spring do szkieletu aplikacji Play, wyniki wydajności obu szkieletów były na podobnym poziomie. W literaturze można również znaleźć inne prace porównujące zarówno Spring Boot jak i Laravel do szkieletów aplikacji języka C# i innych języków programowania, gdzie autorzy pokazują mocne i słabe strony badanych struktur [13][14][15].

2. Cel i zakres pracy

Celem pracy jest porównanie wydajności dwóch aplikacji stworzonych za pomocą najpopularniejszych szkieletów aplikacji języka Javy i PHP. Celem badania jest odpowiedź na następujące pytanie: za pomocą której technologii można budować aplikacje o lepszej wydajności. Na potrzeby badania powstały dwie proste aplikacje testowe o identycznych funkcjonalnościach. Następnie stworzone aplikacje zostały porównane pod względem czasu ładowania zasobów oraz obciążenia aplikacji.

Teza: Aplikacja internetowa stworzona przy użyciu szkieletu Spring Boot wykazuje lepszą wydajność niż analogiczna aplikacja zbudowana przy użyciu szkieletu Laravel.

Szczegółowe pytania badawcze:

1. Czy aplikacja stworzona w szkielecie Spring Boot wyświetla szybciej zasoby strony niż aplikacja w szkielecie Laravel?
2. Czy aplikacja stworzona przy użyciu Spring Boot ma mniejszy rozmiar od aplikacji stworzonej używając platformy Laravel?
3. Czy szkielet aplikacji Spring Boot ma większe wsparcie społeczności niż szkielet aplikacji Laravel?

3. Wykorzystane narzędzia i technologie

3.1. Spring Boot

Spring Boot to popularny framework, który umożliwia tworzenie aplikacji w języku Java. Spring Boot oparty jest na Spring Framework. Wykorzystuje najlepsze techniki swojego poprzednika, a dodatkowo zapewnia domyślne ustawienia konfiguracji dla wielu komponentów i bibliotek, minimalizując potrzebę manualnej konfiguracji oraz zapewnia kontener, który pozwala w prosty sposób uruchomić aplikację bez konieczności wdrażania i ustawiania zewnętrznych kontenerów. Framework posiada system zarządzania zależnościami oparty na Maven lub Gradle. Dzięki temu w łatwy sposób można zarządzać bibliotekami i ich wersjami, co ułatwia utrzymanie aplikacji oraz aktualizacje jej zależności. Spring Boot najczęściej wykorzystywany jest do tworzenia aplikacji internetowych i mikroserwisów.

3.2. Laravel

Laravel, framework języka PHP, został wydany w 2011 roku i szybko stał się jednym z popularniejszych szkieletów aplikacji dla tego języka. Zbudowany jest w oparciu o wzorzec MVC (Model-View-Controller), co pomaga w rozdzieleniu logiki biznesowej, warstwy prezentacji oraz warstwy danych i tym samym ułatwia zarządzanie kodem. Laravel zawiera w sobie wiele funkcjonalności, które są często używane w aplikacjach internetowych, takich jak uwierzytelnianie, routing, sesje czy buforowanie. Funkcjonalności te umożliwiają programiście pominięcie pisania części kodu, co wpływa na skrócenie czasu pracy nad projektami. Dodatkowo Laravel oferuje narzędzia usprawniające obsługę baz danych. Język zapytań Eloquent pozwala w prosty sposób tworzyć zapytania do bazy danych, definiować relacje między tabelami oraz wykonywać operacje CRUD (Create, Read, Update, Delete). Do tworzenia widoków framework wykorzystuje mechanizm szablonów Blade, który wykorzystuje składnie podobną do języka PHP.

3.3. JMeter

Apache JMeter to darmowe narzędzie do testowania wydajności aplikacji internetowych. Oprogramowanie

zostało stworzone w 1999 roku przez Apache Software Foundation. W ciągu kilku lat stało się jednym z najpotężniejszych narzędzi, które pozwala programistom i testerom wykonywać testy obciążeniowe, wydajnościowe i inne rodzaje testów aplikacji. Obecnie program JMeter używany jest na całym świecie przez największe firmy branży IT. Apache JMeter posiada interfejs graficzny, który pozwala na łatwe konfigurowanie i wykonywanie testów. Narzędzie umożliwia tworzenie scenariuszy testowych, w których definiuje się zestawy akcji, takich jak żądania http, symulowanie interakcji użytkowników, reakcji programu na poszczególne dane oraz sposób prezentacji wyników. Dodatkowo JMeter obsługuje wielowątkowość, co pomaga w badaniu wykonywania akcji przez wielu użytkowników jednocześnie.

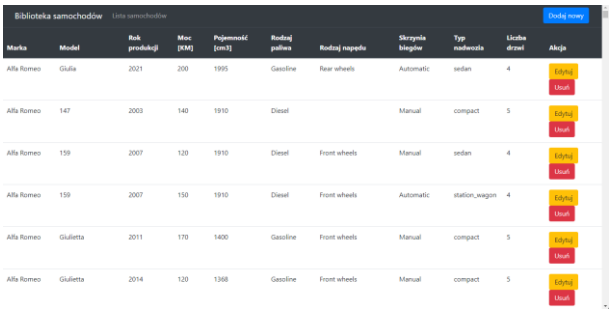
4. Metoda badań

4.1. Plan badań

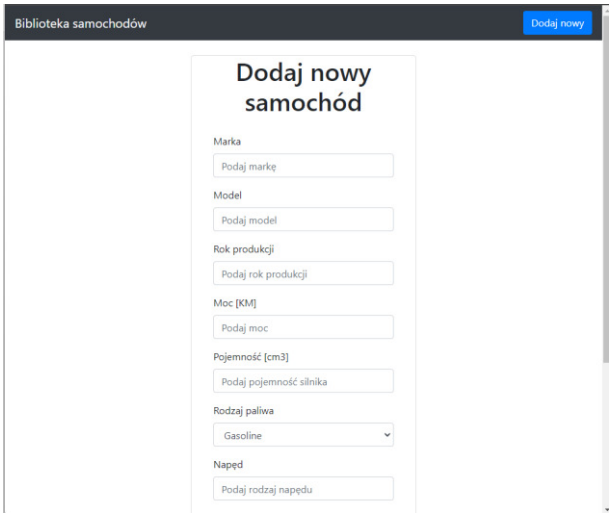
Analizę wydajnościową aplikacji utworzonych na bazie szkieletów programistycznych Laravel i Spring Boot zrealizowano na podstawie następującego planu:

1. Utworzenie dwóch identycznych aplikacji na podstawie dwóch różnych szkieletów programistycznych:
   a) jedna używając szkieletu Laravel,
   b) druga używając szkieletu Spring Boot.
2. Przeprowadzenie testów wydajności aplikacji:
   a) użyte narzędzie: Apache JMeter,
   b) kryteria badania: obsługa bazy danych (zapis, pobranie, edycja, usuwanie danych) dla różnych zbiorów danych,
3. Porównanie oraz opracowanie wyników.
4. Porównanie rozmiaru poszczególnych aplikacji:
   a) ilość zajętego miejsca na dysku,
   b) liczba plików,
   c) metryki kodu.
5. Analiza wsparcia społeczności poprzez zbadanie liczby zapytań o dane szkielety na platformie stackoverflow.com [16].
6. Przedstawienie wniosków.

W celu przeprowadzania badań wykonane zostały dwie identyczne aplikacje testowe, które pozwalają użytkownikowi na wyświetlanie, dodawanie, aktualizację oraz usuwanie danych o samochodach. Jedna aplikacja została stworzona używając szkieletu Spring Boot, do drugiej użyto szkieletu Laravel. Do automatyzacji prac nad ich implementacją posłużyły narzędzia Maven w przypadku aplikacji tworzonej w języku Java oraz Composer w przypadku języka PHP. Obie aplikacje połączono z tą samą bazą danych obsługiwaną za pomocą oprogramowania MySQL. Wygląd interfejsu użytkownika stworzono przy użyciu silnika szablonów Thymeleaf dla aplikacji na bazie szkieletu Spring Boot oraz silnika szablonów Blade dla aplikacji na bazie szkieletu Laravel. Widok listy samochodów oraz formularza dodawania nowego samochodu w aplikacjach przedstawiony został na Rysunku 1 oraz Rysunku 2.



Rysunek 1: Wygląd strony wyświetlającej dane.



Rysunek 2: Wygląd strony z formularzem dodawania samochodu.

4.2. Środowisko testowe

Testy wydajności obu aplikacji przeprowadzone zostały w jednakowych warunkach, aby zapewnić jak najbardziej wiarygodne wyniki. Parametry środowiska testowego przedstawia Tabela 1.

Tabela 1: Parametry środowiska testowego

Sprzęt	
Procesor	Intel Core i5-8250
System operacyjny	Windows 11 Home 64-bit
Pamięć RAM	12 GB
Technologie	
PHP	8.2.0
Java	17.0.6
Laravel	10.10.0
Spring Boot	3.0.5
Apache JMeter	5.5

4.3. Scenariusze testowe

Do przeprowadzenia badania wydajności aplikacji zostały opracowane scenariusze testowe, które przedsta-

wia Tabela 2. Scenariusze dotyczą operacji wyświetlania strony, dodania, edycji oraz usunięcia danych.

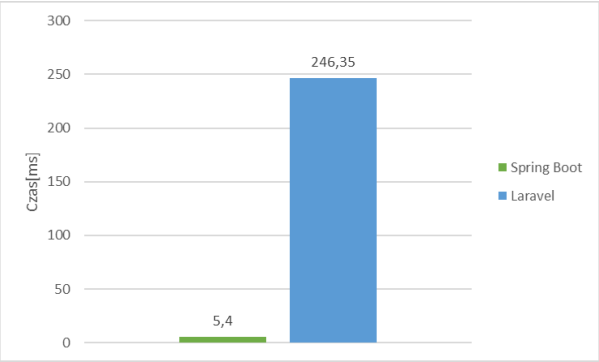
Tabela 2: Scenariusze testowe do zbadania wydajności aplikacji

Scenariusz	Opis
1	Wyświetlenie strony bez danych z bazy
2	Wyświetlenie strony z 10 rekordami przez 1 użytkownika
3	Wyświetlenie strony z 100 rekordami przez 1 użytkownika
4	Wyświetlenie strony z 1000 rekordów przez 1 użytkownika
5	Wyświetlenie strony z 10000 rekordów przez 1 użytkownika
6	Wyświetlenie strony z 10 rekordami gdy 10 użytkowników wyświetla ją jednocześnie
7	Wyświetlenie strony z 1000 rekordami gdy 10 użytkowników wyświetla ją jednocześnie
8	Dodanie 1 rekordu przez 1 użytkownika
9	Dodanie 1 rekordu gdy 10 użytkowników dodaje go jednocześnie
10	Aktualizacja 1 rekordu przez 1 użytkownika
11	Aktualizacja po 1 rekordzie gdy 10 użytkowników aktualizuje go jednocześnie
12	Usunięcie rekordu

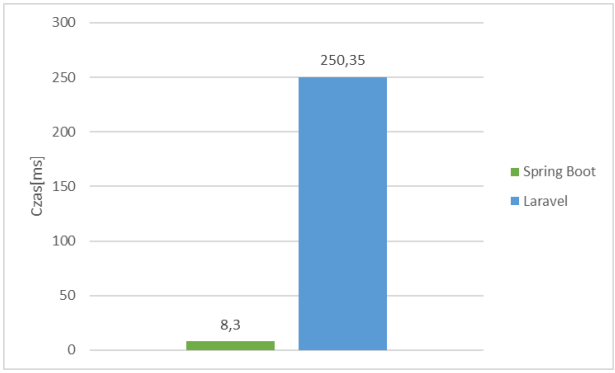
5. Prezentacja rezultatów badań

5.1. Analiza wydajnościowa

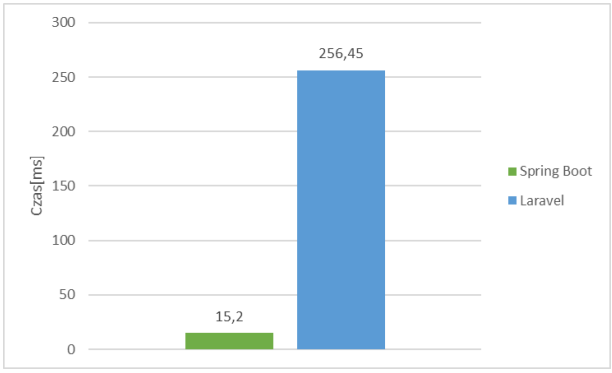
Wydajność aplikacji testowych zbadano za pomocą narzędzia Apache JMeter bazując na wcześniej przygotowanych scenariuszach. Każdy scenariusz został wykonany 20 razy dla każdej aplikacji. Następnie policzone zostały średnie czasy dla każdego scenariusza. Na Rysunkach 3-14 przedstawione są wyniki badań dla poszczególnych scenariuszy.



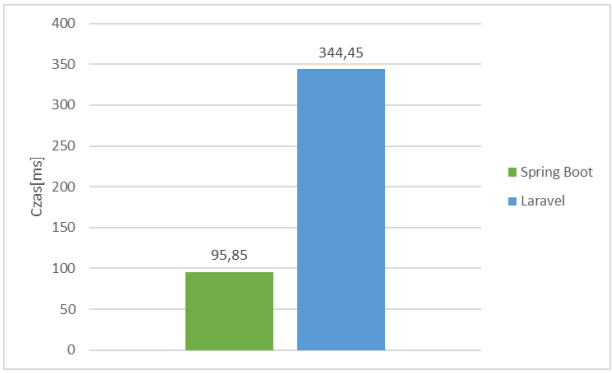
Rysunek 3: Średnie czasy operacji dla scenariusza 1 – wyświetlanie strony bez danych z bazy.



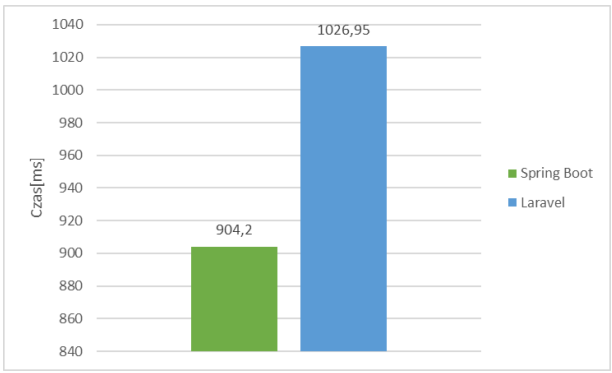
Rysunek 4: Średnie czasy operacji dla scenariusza 2 – wyświetlanie strony z 10 rekordami przez 1 użytkownika.



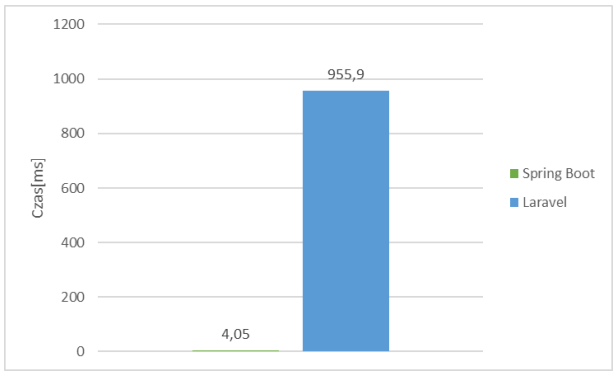
Rysunek 5: Średnie czasy operacji dla scenariusza 3 - wyświetlenie strony z 100 rekordami przez 1 użytkownika.



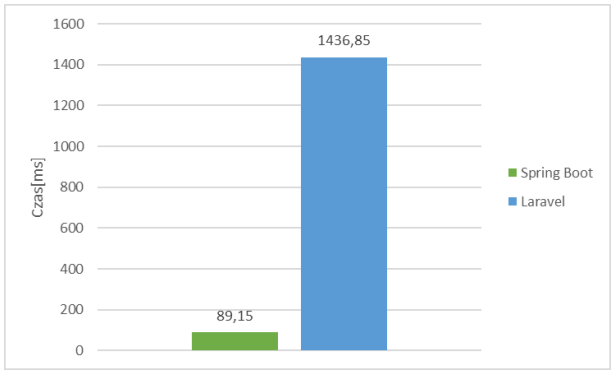
Rysunek 6: Średnie czasy operacji dla scenariusza 4 - wyświetlenie strony z 1000 rekordów przez 1 użytkownika.



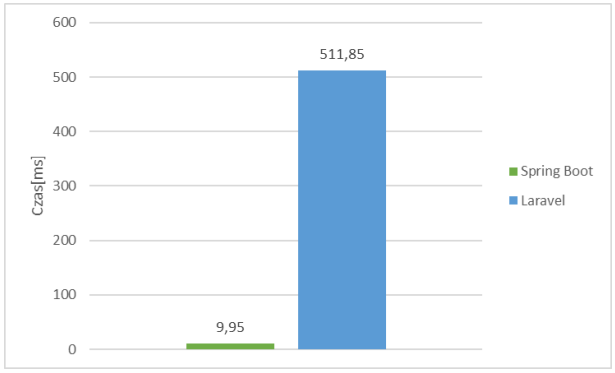
Rysunek 7: Średnie czasy operacji dla scenariusza 5 - wyświetlenie strony z 10000 rekordów przez 1 użytkownika.



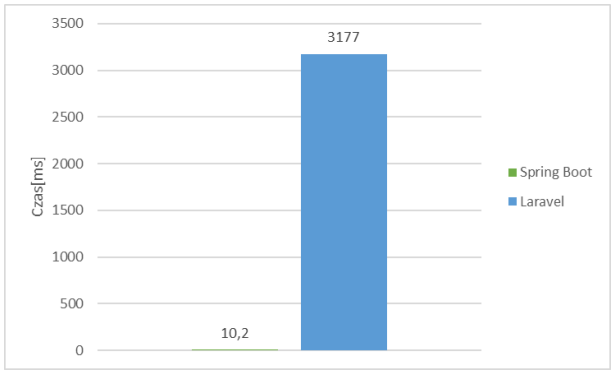
Rysunek 8: Średnie czasy operacji dla scenariusza 6 - wyświetlenie strony z 10 rekordami gdy 10 użytkowników wyświetla ją jednocześnie.



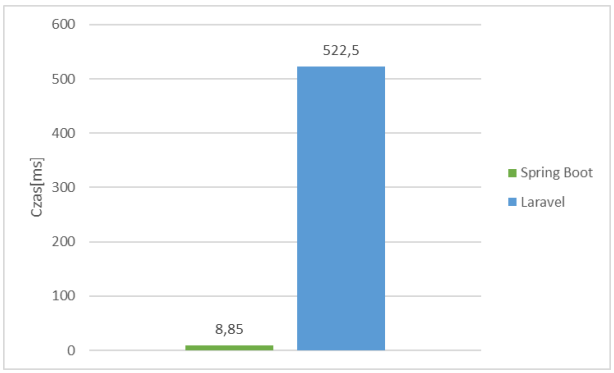
Rysunek 9: Średnie czasy operacji dla scenariusza 7 - wyświetlenie strony z 1000 rekordami gdy 10 użytkowników wyświetla ją jednocześnie.



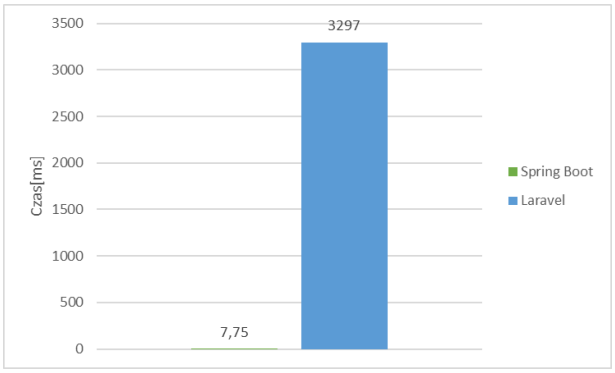
Rysunek 10: Średnie czasy operacji dla scenariusza 8 - dodanie 1 rekordu przez 1 użytkownika.



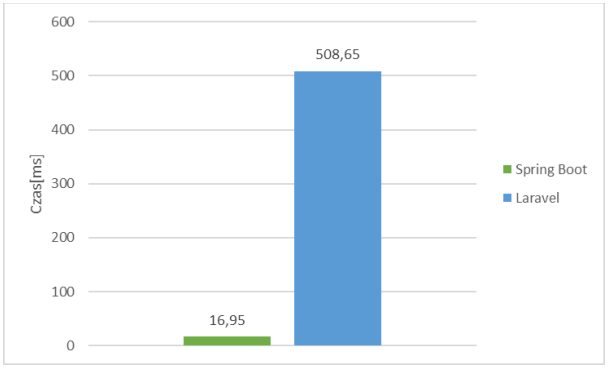
Rysunek 11: Średnie czasy operacji dla scenariusza 9 - dodanie 1 rekordu gdy 10 użytkowników dodaje go jednocześnie.



Rysunek 12: Średnie czasy operacji dla scenariusza 10 - aktualizacja 1 rekordu przez 1 użytkownika.



Rysunek 13: Średnie czasy operacji dla scenariusza 11 - aktualizacja po 1 rekordzie gdy 10 użytkowników aktualizuje go jednocześnie.



Rysunek 14: Średnie czasy operacji dla scenariusza 12 - usunięcie rekordu.

W scenariuszu 1, badano czasy wyświetlenia strony bez danych z bazy. Dla aplikacji napisanej w Spring Boot, najkrótszy czas wyniósł 5 ms, a najdłuższy 6 ms, przy odchyleniu standardowym 0,5026. Natomiast dla aplikacji napisanej w Laravel, najkrótszy czas wyniósł 234 ms, najdłuższy 294 ms, a odchylenie standardowe wyniosło 14,0611. Różnica między średnimi czasami obu aplikacji wyniosła około 4462%. W scenariuszu 2, badano czas wyświetlenia strony z 10 rekordami przez 1 użytkownika. Dla aplikacji napisanej w Spring Boot, najkrótszy zmierzony czas wyniósł 7 ms, a najdłuższy 13 ms, przy odchyleniu standardowym 1,6255. Dla aplikacji napisanej w Laravel, najmniejszy czas wyniósł 234 ms, a największy 264 ms, przy odchyleniu standardowym 9,9857. Różnica w średnich czasach wynosi około 2887%. W scenariuszu 3, zbadano czas wyświetlenia strony z 100 rekordami przez 1 użytkownika. Dla

aplikacji napisanej w Spring Boot, najkrótszy czas wyniósł 14 ms, a najdłuższy 18 ms, przy odchyleniu standardowym 1,0563. Czasy dla aplikacji napisanej w Laravel mieściły się w przedziale 242 ms do 291 ms, przy odchyleniu standardowym 13,2961. Różnica między średnimi czasami wyniosła około 1587%. W scenariuszu 4, zbadano czas wyświetlenia strony z 1000 rekordami przez 1 użytkownika. Dla aplikacji stworzonej w Spring Boot, najkrótszy czas wyniósł 92 ms, najdłuższy 108 ms, a odchylenie standardowe 3,3916. Dla aplikacji stworzonej w Laravel, najkrótszy czas wyniósł 311 ms, najdłuższy 423 ms, a odchylenie standardowe 26,6705. Różnica między średnimi wyniosła około 259%. W kolejnym scenariuszu, dotyczącym czasów wyświetlania strony z 10000 rekordami, dla aplikacji stworzonej w Spring Boot, najkrótszy zbadany czas wyniósł 885 ms, a najdłuższy 964 ms, przy odchyleniu standardowym 25,2662. Dla aplikacji napisanej w Laravel, najkrótszy czas wyniósł 1009 ms, najdłuższy 1080 ms, a odchylenie standardowe 15,4936. Różnica między średnimi wynosi około 14%. W scenariuszu 6, badano czasy wyświetlenia strony z 10 rekordami, gdy 10 użytkowników wyświetla ją jednocześnie. Dla aplikacji stworzonej w Spring Boot, najkrótszy czas wyniósł 4 ms, najdłuższy 5 ms, a odchylenie standardowe wyniosło 0,2236. Dla aplikacji stworzonej w Laravel, najkrótszy czas wyniósł 914 ms, najdłuższy 1017 ms, a odchylenie standardowe to 29,0171. Różnica między średnimi wyniosła około 23502%. W scenariuszu 7, badano czasy wyświetlania strony z 1000 rekordami, gdy 10 użytkowników wyświetla ją jednocześnie. Dla aplikacji napisanej w Spring Boot, minimalny czas wyniósł 87 ms, a maksymalny 94 ms, przy odchyleniu standardowym 1,8715. Dla aplikacji napisanej w Laravel, czasy mieszczą się w przedziale 1326 ms do 1646 ms, przy odchyleniu standardowym 107,0378. Różnica między średnimi wynosi około 1512%. W scenariuszu 8, badano czasy dodania rekordu przez 1 użytkownika. Dla aplikacji napisanej w Spring Boot, najkrótszy czas wyniósł 7 ms, najdłuższy 13 ms, a odchylenie standardowe 1,1459. Dla aplikacji napisanej w Laravel, najkrótszy czas wyniósł 493 ms, najdłuższy 577 ms, a odchylenie standardowe 20,8030. Różnica między średnimi czasami aplikacji wynosi 5044%. W scenariuszu 9, badano czasy dodania rekordu przez 10 użytkowników jednocześnie. Dla aplikacji stworzonej w Spring Boot, czasy mieszczą się w przedziale 10 ms do 11 ms, przy odchyleniu standardowym 0,4104. Dla aplikacji stworzonej w Laravel, najmniejszy czas wyniósł 3073 ms, a największy 3295 ms, przy odchyleniu standardowym 69,9165. Różnica między średnimi czasami wynosi 31047%. W scenariuszu 10, badano czasy aktualizacji 1 rekordu przez 1 użytkownika. Dla aplikacji napisanej w Spring Boot, czasy wynosiły od 8 ms do 13 ms, a odchylenie standardowe wyniosło 1,1367. Dla aplikacji napisanej w Laravel, czasy znajdowały się w przedziale od 493 ms do 560 ms, przy odchyleniu standardowym 17,2977. Różnica między średnimi czasami dla obu aplikacji wynosi 5804%. W scenariuszu 11, badano czas aktualizacji po 1 rekordzie, gdy 10 użytkowników aktualizuje go jednocześnie. Dla aplikacji napisanej w Spring Boot, czasy mieszczą się w przedziale od 7 ms do 8 ms, a odchylenie standardowe wynosi 0,4443. Dla aplikacji napisanej w Laravel, czasy wynoszą od 3035 ms do 3665 ms, przy odchyleniu standardowym 193,9305. Różnica między średnimi czasami aplikacji wynosi 42442%. W scenariuszu 12, badano czasy usunięcia rekordu. Dla aplikacji napisanej w Spring Boot, najkrótszy czas wynosi 14 ms, najdłuższy 22 ms, a odchylenie standardowe wyniosło 1,9861. Dla aplikacji napisanej w Laravel, najkrótszy czas wynosi 483 ms, najdłuższy 564 ms, a odchylenie standardowe to 19,5024. Różnica między średnimi czasami wynosi 2901%. W każdym z 12 scenariuszy Spring Boot osiągnął krótsze czasy. Najmniej różnic można zaobserwować w scenariuszu 5, gdzie jeden użytkownik wyświetlał stronę z dużą ilością danych z bazy. W tym przypadku średnie czasy różniły się o około 122 ms. Natomiast największe rozbieżności odnotowano w scenariuszu 11, w którym mierzono czasy aktualizacji rekordu przez 10 użytkowników jednocześnie, gdzie średnie czasy różniły się o około 3289 ms. Podczas wyświetlania strony z danymi przez jednego użytkownika wraz ze wzrostem liczby danych malała różnica między badanymi szkieletami. Scenariusze dotyczące średnich czasów dodawania rekordu przez 1 użytkownika oraz 10 użytkowników jednocześnie pokazują, że czasy dla aplikacji zbudowanej w szkielecie Spring Boot nie różnią się znacząco, natomiast czasy dla aplikacji stworzonej w szkielecie Laravel wzrosły ponad sześciokrotnie. Podobne wyniki można zaobserwować dla średnich czasów aktualizacji rekordu przez 1 użytkownika oraz 10 użytkowników jednocześnie.

5.2. Metryki aplikacji

Zestawienie metryk aplikacji przedstawia Tabela 3. Porównano w niej ogólny rozmiar projektu dla obu aplikacji, liczbę plików w obu projektach, liczbę linii kodu języka Java i PHP, które napisał autor aplikacji oraz liczbę linii kodu dla silników szablonów Thymeleaf oraz Blade, które również napisał autor aplikacji. Przed rozpoczęciem pomiarów aplikacje zostały uszczuplone o foldery z plikami konfiguracyjnymi, które mogłyby znacznie zaburzyć wyniki. W przypadku aplikacji zbudowanej przy pomocy szkieletu Spring Boot odrzucono foldery takie jak „target", natomiast w przypadku aplikacji budowanej na bazie frameworka Laravel usunięty został między innymi folder „vendor". Do pomiarów użyto wtyczki Statistic, która jest dostępna zarówno w Intellij IDEA, jak i PhpStorm.

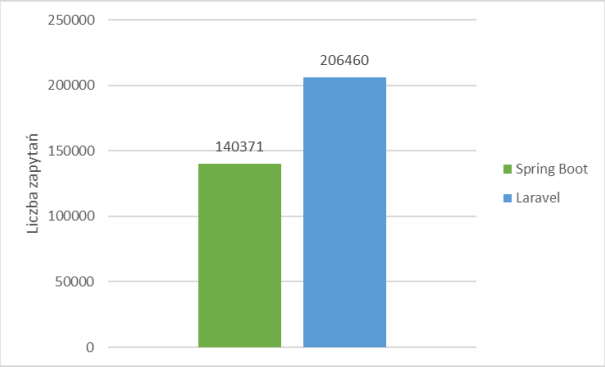
Tabela 3: Metryki kodu aplikacji testowych

	Spring Boot	Laravel
Rozmiar projektu na dysku [KB]	136	540
Liczba plików w projekcie	24	82
Liczba linii kodu Java/PHP napisanych przez autora	150	119
Liczba linii kodu Thymelaeaf/Blade	241	312

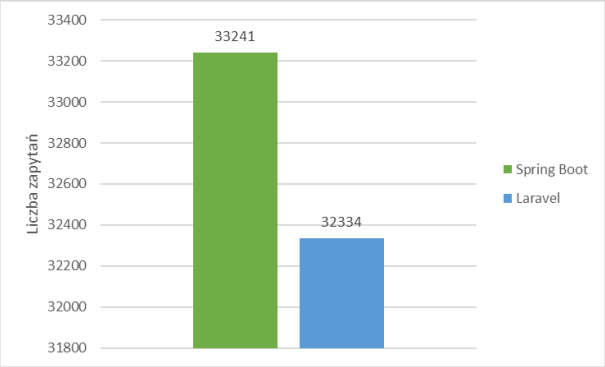
Projekt zbudowany na szkielecie Spring Boot zajął prawie czterokrotnie mniej miejsca na dysku od projektu, do którego wykorzystano szkielet Laravel. Liczba plików w projekcie oraz liczba linii kodu Thymeleaf/Blade również są mniejsze w przypadku szkieletu Spring Boot o odpowiednio 58 oraz 71. Jedynie w przypadku liczby linii kodu Java/PHP napisanych przez autora szkielet Laravel osiągnął wynik mniejszy o 31.

5.3. Wsparcie społeczności

Badanie polegało na porównaniu liczby zapytań dla obu szkieletów w serwisie społecznościowym Stack Overflow, który przeznaczony jest do zadawania pytań dotyczących wytwarzania programowania. Wyniki wszystkich zapytań o dany szkielet zostały przedstawione na Rysunku 15. Na Rysunku 16 przedstawiono wyniki dla zapytań o szkielet, które nie miały żadnej odpowiedzi.



Rysunek 15: Liczba zapytań w serwisie stackoverflow.com.



Rysunek 16: Liczba zapytań w serwisie stackoverflow.com bez odpowiedzi.

Użytkownicy serwisu stackoverflow.com częściej pytali o szkielet aplikacji Laravel. Różnica w liczbie zapytań wyniosła 66089. Zapytania bez odpowiedzi stanowiły 24% wszystkich zapytań w przypadku szkieletu Spring i 16% w przypadku szkieletu aplikacji Laravel.

6. Wnioski

Przeprowadzone badania pokazują, że aplikacja zbudowana na szkielecie Spring Boot ma znacznie lepszą wydajność od tej zbudowanej na szkielecie Laravel. Tym samym postawiona teza: *Aplikacja internetowa stworzona przy użyciu szkieletu Spring Boot wykazuje lepszą wydajność niż analogiczna aplikacja zbudowana przy użyciu szkieletu Laravel.*

Pierwsze szczegółowe pytanie badawcze brzmiało: *Czy aplikacja stworzona w szkielecie Spring Boot wyświetla szybciej zasoby strony niż aplikacja w szkielecie Laravel?* Odpowiedź uzyskano na postawie przeprowadzonych badań wydajności, gdzie Spring Boot osiągnął krótsze czasy wyświetlania danych dla każdego scenariusza przedstawiającego tę operację. Wyniki pokazują również, że przy większym zestawie danych wyświetlanych przez jednego użytkownika różnica w czasie wyświetlenia zasobów była mniejsza.

Drugim szczegółowym pytaniem badawczym było: *Czy aplikacja stworzona przy użyciu Spring Boot ma mniejszy rozmiar od aplikacji stworzonej używając platformy Laravel?* Analiza metryk kodu wykazała, że aplikacja zbudowana z pomocą szkieletu Spring Boot zajmuje prawie czterokrotnie mniej miejsca na dysku, niż identyczna aplikacja napisana w szkielecie Laravel. Również w przypadku liczby plików w projekcie oraz liczba linii kodu Thymeleaf/Blade aplikacja stworzona w szkielecie Spring Boot osiąga mniejsze wyniki. Natomiast aplikacja stworzona w szkielecie Laravel osiągnęła mniejszy wynik dla liczby linii kodu Java/PHP napisanych przez autora.

Trzecia hipoteza badawcza brzmiała: *Czy szkielet aplikacji Spring Boot ma większe wsparcie społeczności niż szkielet aplikacji Laravel?* Analiza została przeprowadzona w oparciu o zapytania w serwisie stackoverflow.com. Wyniki wskazały, że użytkownicy częściej pytali o szkielet aplikacji Laravel, a także chętniej odpowiadali na pytania dotyczące tego szkieletu aplikacji.

Przeprowadzone badania oraz otrzymane wyniki mogą być pomocne dla programistów w wyborze technologii do tworzenia aplikacji internetowych.

Literatura

[1] C. L. Hsu, K. C. Chang, M. C. Chen, The impact of website quality on customer satisfaction and purchase intention: perceived playfulness and perceived flow as mediators, Information Systems and e-Business Management 10 (2012) 549-570.

[2] J. Radomski, Porównanie wydajności aplikacji internetowych na przykładzie szkieletów programistycznych Laravel i Vaadin, Journal of Computer Sciences Institute 22 (2022) 35-39.

[3] Paweł Rzeńca, Porównanie wytwarzania oprogramowania internetowego z wykorzystaniem różnych technologii, Repozytorium PJATK, Warszawa, 2022.

[4] S. Memon, R.B. Palh, M. Memon, H.S. Memon, Performance comparison of QEC network based JAVA application and web based PHP application, International Journal of Advanced Computer Science and Applications 9 (2018) 555-564.

[5] H. K. Dhalla, A Performance Comparison of RESTful Applications Implemented in Spring Boot Java and MS. NET Core, In Journal of Physics: Conference Series, IOP Publishing (2021) 2-7.

[6] L. R. Abbade, M. A. da Cruz, J. J. Rodrigues, P. Lorenz, R. A. Rabelo, J. Al-Muhtadi, Performance comparison of programming languages for Internet of Things

middleware, Transactions on Emerging Telecommunications Technologies 31 (2020) 522-532.

[7] U. Latif, T. Kusumasari, Comparison Between Yii Frameworks and Laravel in 3 Different Version for Viewing Large Data of Shipyard Industry in Indonesia, International Journal of Innovation in Enterprise System 2 (2018) 13-18.

[8] M. Laaziri, K. Benmoussa, S. Khoulji, M. L. Kerkeb, A Comparative study of PHP frameworks performance, Procedia Manufacturing 32 (2019) 864-871.

[9] P. R. Chavan, S. Pawar, Comparison Study Between Performance of Laravel and Other PHP Frameworks, International Journal of Research in Engineering, Science and Management 4 (2021) 27-29.

[10] O. W. Purbo, A Systematic Analysis: Website Development using Codeigniter and Laravel Framework, Enrichment: Journal of Management 12 (2021) 1008-1014.

[11] A. Sunardi, MVC architecture: A comparative study between laravel framework and slim framework in freelancer project monitoring system web based, Procedia Computer Science 157 (2019) 134-141.

[12] M. Gajewski, W. Zabierowski, Analysis and comparison of the Spring framework and play framework performance, used to create web applications in Java, IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (2019) 170-173.

[13] A. Poudel, A comparative study of project management system web applications built on ASP. Net core and laravel MVC frameworks, The Repository @ St. Cloud State, Minnesota, 2018.

[14] M. Kaluža, M. Kalanj, A comparison of back-end frameworks for web application development, Computer Science 7 (2019) 317-332.

[15] J. Muittari, Modern web back-end, Oulu University of Applied Sciences Information Technology, Oulu, 2020.

[16] Stack Overflow – serwis społecznościowy, https://stackoverflow.com, [10.06.2023].

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 312–316
Received: 21 June 2023
Accepted: 17 September 2023

Comparative Analysis of Selected Game Engines

Analiza porównawcza wybranych silników graficznych

Bartłomiej Szabat* , Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
This article presents a comparative analysis of two popular graphic engines, namely Unity and Unreal Engine 5. The objective of the study was to gather insights into the performance and quality of applications developed using these engines. The research involved creating two games, followed by conducting performance tests and collecting user feedback on the quality and performance of these games. The results indicated that the application developed in Unity exhibited superior performance, while the application developed in Unreal Engine 5 received higher user ratings in terms of visual aspects.

Keywords: game engines; game development; comparative analysis

Streszczenie
W niniejszym artykule przeprowadzono analizę porównawczą dwóch popularnych silników graficznych, Unity i Unreal Engine 5. Głównym celem badania było uzyskanie informacji dotyczących wydajności obu silników oraz jakości aplikacji tworzonych przy ich użyciu. Badanie zostało przeprowadzone poprzez stworzenie dwóch gier, a następnie przeprowadzenie testów wydajnościowych oraz zebranie opinii od użytkowników tych gier na temat jakości i wydajności. Wyniki badań wskazały, że aplikacja stworzona w Unity charakteryzuje się lepszą wydajnością, podczas gdy aplikacja stworzona w Unreal Engine 5 otrzymuje wyższe oceny od użytkowników pod względem wizualnym.

Słowa kluczowe: silniki gier komputerowych; produkcja gier; analiza porównawcza

*Corresponding author
Email address: bartlomiej.szabat@pollub.edu.pl (B. Szabat)

1. Wstęp

Rynek gier komputerowych z roku na rok staję się co raz większy przez co coraz więcej osób decyduje się na rozpoczęcie kariery w dziedzinie tworzenia gier komputerowych. Proces tworzenia gier komputerowych jest niezwykle skomplikowany, ponieważ wymaga zastosowania zaawansowanych technologii, a także integracji takich elementów jak grafika, dźwięk, animacja, mechanika rozgrywki, połączenie wielu graczy czy sztuczna inteligencja. Aby ułatwić ten proces tworzone są silniki graficzne, które pomagają twórcom w szybszy i bardziej efektywny sposób tworzyć gry na różne platformy, zapewniając im dostęp do narzędzi wspomagających połączenie ze sobą wcześniej wymienionych elementów gier komputerowych. Ten artykuł skupia się na porównaniu dwóch popularnych silników graficznych Unity oraz Unreal Engine. Powstało wiele badań porównujących te silniki, jednym z nich jest praca inżynierska A. Šmída [1], który stworzył dwie identyczne gry 3D oparte na grze Pac-Man, następnie porównał je na podstawie własnych odczuć co do procesu tworzenia oraz wydajności obu gier na kilku platformach. W innym przypadku, w badaniu przeprowadzonym przez R. Salama oraz M. Elsayeda [2] silniki zostały porównane pod względem ich specyfikacji technicznej, historii obu silników oraz prostotę ich użytkowania. Można zauważyć, że podobny temat został poruszony w kilku innych pracach [3-6], jednak każda z nich skupiała się na innych aspektach silników lub przeprowadzała porównanie z innymi kryteriami. Na przykład, w jednym z artykułów autor skupił się na tym, który silnik jest lepszy w pomaganiu rozwijania umiejętności programistycznych [5], a w innym badano wykorzystywane techniki oświetlenia otoczenia w czasie rzeczywistym [6]. Istnieją również artykuły, które oprócz silników Unity oraz Unreal Engine skupiają się także na innych silnikach tak jak w artykule A. Barczaka i H. Woźniaka [7] gdzie oprócz Unity i Unreal Engine porównano także silnik Cry Engine 3 lub w badaniach [8, 9] gdzie na podstawie danych technicznych oraz powszechnie dostępnych informacji analizowano możliwości technologiczne nawet 14 silników graficznych. Pod innym kątem porównanie przeprowadzono w “Comparison between famous game engines and eminent games” [10]. W artykule tym analizowano wydajność nie tyle całych silników, co gier powstałych z pięciu najpopularniejszych silników. Przeprowadzono także liczne badania sprawdzające możliwości wykorzystania silników graficznych do celów naukowych. Informacje o takich badaniach możemy przykładowo znaleźć w artykułach takich jak:

- „XR game development as a tool for authentic, experiential, and collaborative learning” [11], którego celem badania było wykorzystanie silnika Unity jako narzędzia do pomocy w nauczaniu biologii w czasie zajęć zdalnych,
- „Comparative analysis of Unity and Unreal Engine efficiency in creating virtual exhibitions of 3D scanned models” [12], w którym autorzy pracy spraw-

dzają możliwości silników Unity i Unreal Engine do tworzenia cyfrowych wystaw dzieł sztuki,
- „Comparison of game engines for serious games" [13], który skupia się na możliwości wykorzystania tych silników do tworzenia gier edukacyjnych.

2. Cel i zakres pracy

Celem niniejszej pracy jest porównanie dwóch gier stworzonych za pomocą silnika Unity oraz Unreal Engine 5 w celu uzyskania odpowiedzi na pytanie: który z tych silników pozwala na budowę aplikacji, które lepiej odpowiadają użytkownikom oraz charakteryzują się wyższą wydajnością. Aby wyniki były jak najdokładniejsze stworzono identyczne gry pod względem funkcjonalności
a stworzone gry posiadają niewielką złożoność co oznacza, że nie posiadają napisanych przez użytkownika skomplikowanych skryptów, sztucznej inteligencji czy też wielu scen. Następnie stworzone aplikacje porównano na podstawie ich wydajności oraz ankiety użytkowników.
Teza: Gra stworzona w Unreal Engine 5 wykazuje wyższą wydajność oraz lepsze oceny użytkowników niż gra stworzona w Unity.

Szczegółowe pytania badawcze:
1. Czy gra o niedużej złożoności stworzona na silniku Unreal Engine 5 wykorzystuje mniej zasobów od identycznej funkcjonalnie gry stworzonej w Unity?
2. Czy gra o niedużej złożoności stworzona przy użyciu Unreal Engine 5 osiąga większe średnie liczby klatek na sekundę niż podobna gra stworzona w Unity?
3. Czy aplikacja stworzona za pomocą Unreal Engine 5 uzyskała wyższe oceny od użytkowników?

3. Plan Badań

Plan eksperymentu badawczego składa się z następujących etapów:
1. Stworzenie gier na podstawie, których przeprowadzone będzie badanie
   a) Stworzenie konceptu gry
   b) Przygotowanie potrzebnych modeli
   c) Stworzenie aplikacji w silniku Unity
   d) Stworzenie aplikacji w silniku Unreal Engine
2. Stworzenie badania wydajności aplikacji
   a) Przygotowanie testu, który zmierzy średnią ilość klatek na sekundę oraz wykorzystanie zasobów
3. Stworzenie badania ankietowego dla użytkowników aplikacji na około 10 pytań
   a) Przygotowanie pytań dotyczących wydajności gry
   b) Przygotowanie pytań dotyczących wyglądu gry
   c) Przygotowanie pytań dotyczących opinii graczy na temat gry
   d) Grupa badawcza: Osoby grające na co dzień w gry komputerowe
   e) Przedział wiekowy: 18-26 lat
   f) Minimalna liczba badanych osób: 5
4. Przeprowadzenie badania wydajności na komputerach użytkowników
   a) Przesłanie użytkownikom badanych aplikacji
   b) wykonanie testu oraz przesłanie wyników
5. Przeprowadzenie ankiety badawczej wśród użytkowników
6. Analiza wyników testu wydajności oraz ankiety użytkowników
7. Wnioski.

3.1. Stworzone aplikacje

Do przeprowadzenia badań stworzono dwie aplikacje w poszczególnych silnikach przedstawiające tę samą scenę. Scena przedstawia górzysty las z małym jeziorem oraz rozpalonym ogniskiem, po której gracz może swobodnie się poruszać. Aplikacje wykonano korzystając z podstawowych funkcji dostępnych w obu silnikach co oznacza, że ograniczono je do funkcji dostępnych bez dodatkowych opłat. W aplikacji stworzonej w Unreal Engine 5 użyto modeli dostępnych w „Quixel Bridge" gdzie znajdziemy całą bibliotekę wysokiej rozdzielczości skanów elementów krajobrazu takich jak drzewa, skały czy trawy. W Unity natomiast modele ograniczone były do „Asset Store", w którym użytkownicy mogą udostępniać stworzone przez siebie modele oraz skrypty. Aby powstałe aplikacje były do siebie jak najbardziej zbliżone, wykorzystano modele 3D z teksturami o jednakowych rozdzielczościach (2048x2048 pikseli) i zbliżonej złożoności.

Rysunek 1: Zrzut ekranu aplikacji stworzonej w Unreal Engine 5.

Rysunek 2: Zrzut ekranu aplikacji stworzonej w Unity.

W narzędziu „Quixel Bridge" istnieje możliwość wyboru określonej jakości dla wybranych modeli, co nie jest możliwe w przypadku „Asset Store". Dlatego początkowo skupiono się na modelach przeznaczonych do silnika Unity, a następnie wybrano podobne modele dostępne dla silnika Unreal Engine 5. W rezultacie uzyskano obiekty 3D, z których najbardziej złożony

został wykorzystany do stworzenia góry w aplikacji zrealizowanej w Unreal Engine 5. Ten obiekt zawiera 5184 wierzchołki i 9158 trójkątów, natomiast w aplikacji stworzonej w Unity użyto obiektu o 6038 wierzchołkach i 8071 trójkątach. Po pobraniu i zaimportowaniu obiektów, nie dokonywano żadnych modyfikacji, z wyjątkiem dostosowania ich do sceny poprzez skalowanie lub obrót. Otrzymane obiekty rozłożono na scenach tak, aby wyglądały one jak najbardziej podobnie do siebie. Do oświetlenia sceny w obu silnikach użyto światła kierunkowego imitującego słońce oraz świateł punktowych przy tworzeniu ogniska, samo ognisko było stworzone za pomocą „Niagara VFX System" w Unreal Engine 5 oraz wbudowanego systemu cząsteczek w Unity. Finalnie otrzymane sceny posiadały około 6 milionów trójkątów.

3.2. Badanie wydajności oraz ankietowe

Test wydajnościowy został stworzony na podstawie monitorowania liczby klatek na sekundę (FPS) poprzez zewnętrzny program „RTSS Rivatuner Statistics Server". Test przeprowadzono na różnych urządzeniach z różnymi konfiguracjami sprzętowymi, podczas testu użytkownicy mogli swobodnie poruszać się po aplikacji. Następnie, po 5 minutach, użytkownika proszono o zrestartowanie aplikacji i przejście określonej ścieżki, podczas której sprawdzano i zapisywano średnią liczbę uzyskanych klatek na sekundę, oraz informacje dotyczące zużycia procesora i pamięci RAM. Dodatkowo, przeprowadzono badanie ankietowe wśród użytkowników, którzy używali porównywanych aplikacji. Ankieta została opracowana w celu zebrania opinii na temat różnych aspektów aplikacji, które podzielono na cztery kategorie

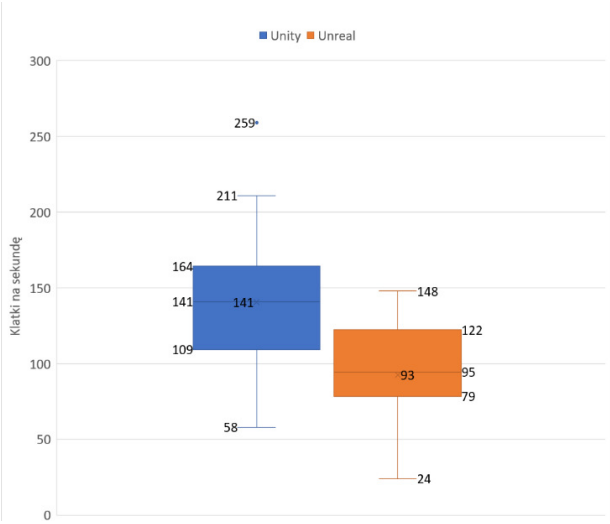
- Animacja: Uczestnicy byli pytani o swoje spostrzeżenia dotyczące animacji w obu aplikacjach. Oceniali płynność, realizm i ogólną jakość animacji,
- Grafika: Użytkownicy byli proszeni o ocenę grafiki w obu aplikacjach. Badano aspekty takie jak jakość tekstur, oświetlenie, modele postaci i otoczenia oraz ogólną estetykę wizualną gier,
- Wydajność: Respondenci zostali zapytani o wydajność aplikacji stworzonych w Unity i Unreal Engine 5. Oceniali płynność rozgrywki, szybkość ładowania, stabilność i ogólną wydajność gier na swoich komputerach,
- Ogólne wrażenia: Uczestnicy mieli okazję wyrazić swoje ogólne wrażenia i opinie na temat obu aplikacji. Badano ich preferencje, subiektywne odczucia, satysfakcję z gry oraz ogólną ocenę jakości użytkowej.

Ankieta została udostępniona online na platformie „Google Forms", a respondenci mieli możliwość udzielenia szczegółowych odpowiedzi i wyrażenia swoich preferencji.

4. Wyniki

4.1. Wyniki testu wydajnościowego

Badanie przeprowadzono na 15 komputerach osobistych, każdy z nich posiadał inną konfigurację sprzętową przez co uzyskane wyniki mają dużą rozpiętość. Na podstawie zebranych danych stworzono wykres pudełkowy (ang. *Box plot*) (Rysunek 3) wygenerowanego w programie „Microsoft Excel". Wykres ten składa się z pudełka, którego dolna krawędź reprezentuje pierwszy kwartyl, górna krawędź trzeci kwartyl, a linia w środku pudełka to drugi kwartyl czyli mediana. Na wykresie pudełkowym, dolna i górna linia wąsów reprezentują minimalną i maksymalną wartość w zbiorze danych. W przypadku pudełka przedstawiającego dane o aplikacji stworzonej w Unity, występuje również punkt, który jest wartością odstającą od głównego zakresu danych. Takie wykresy pozwalają na zilustrowanie dużych ilości danych oraz wyodrębnić wartości odstające od zbioru. Na rysunku poniżej (Rysunek 3) możemy zauważyć, że aplikacja stworzona w Unity wykazywała średnio o 49% większe średnie klatek na sekundę od aplikacji wykonanej w Unreal Engine 5, dodatkowo w najbardziej ekstremalnym przypadku na tym samym komputerze osobistym średnia klatek na sekundę w Unreal Engine 5 wyniosła 24, natomiast w przypadku Unity 58 klatek na sekundę co oznacza, że aplikacja w Unity posiadała aż o 141% większą liczbę klatek na sekundę. Dodatkowo podczas testów zmierzono zużycie zasobów komputera przez aplikację (Tabela 1), w celu uzyskania jak największej liczby klatek na sekundę obie aplikację zużywały blisko 100% procesora graficznego (GPU), zużycie procesora (CPU) wyniosło 14,3% dla aplikacji w Unity oraz 22,5% dla aplikacji stworzonej w Unreal Engine 5, a zużycie pamięci RAM wyniosło 276 MB dla aplikacji wykonanej w Unity oraz 603 MB dla aplikacji wykonanej w Unreal Engine 5.



Rysunek 3: Wykres otrzymanej liczby klatek na sekundę.

Tabela 1: Odchylenie standardowe pomiaru odległości

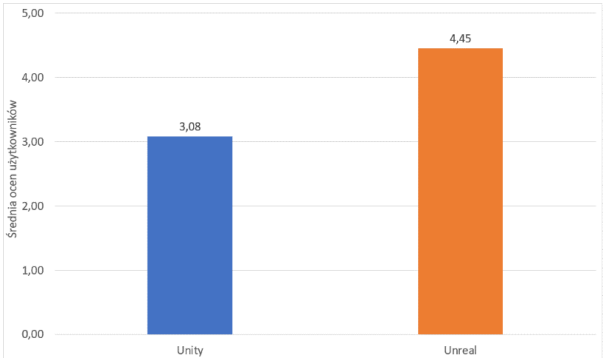
Silnik	Średnie zużycie procesora (%)	Zużycie pamięci RAM (MB)
Unity	14,3	276
Unreal Engine 5	22,5	603

4.2. Wyniki badania ankietowego

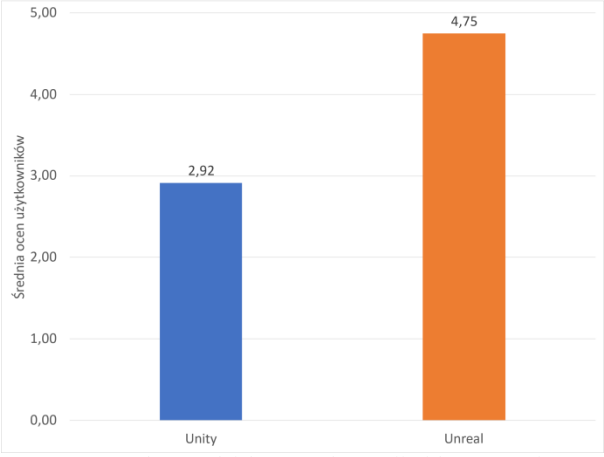
Badaniu ankietowemu poddało się piętnastu użytkowników w wieku od 18 do 26 lat, którzy regularnie korzystają z komputerów do grania w gry. W badaniu zadano pytania w czterech kategoriach

- ogólne wrażenia wizualne,
- efekty otoczenia oraz jakość grafiki,
- animacje postaci oraz otoczenia,
- płynność działania.

W każdej z tych kategorii ankietowani mieli możliwość wyrażenia swojej opinii na podstawie skali od 1 do 5, gdzie 1 oznaczała najniższą jakość lub najgorsze wrażenie, a 5 oznaczała najwyższą jakość lub najlepsze wrażenie. Na podstawie wykresu średnich ocen w kategorii "ogólne wrażenia wizualne" (Rysunek 4) można stwierdzić, że użytkownicy przyznawali znacznie wyższe oceny dla aplikacji stworzonej w Unreal Engine 5. Podobne wnioski można wyciągnąć z wykresu średnich ocen w kategorii "efekty otoczenia oraz jakość grafiki" (Rysunek 5), gdzie aplikacja stworzona w Unity otrzymała średnią ocenę 2,92, co jest jednym z najniższych wyników uzyskanych we wszystkich kategoriach. Natomiast aplikacja stworzona w Unreal Engine 5 otrzymała w tej kategorii najwyższą średnią ocenę jaką jest 4,75.

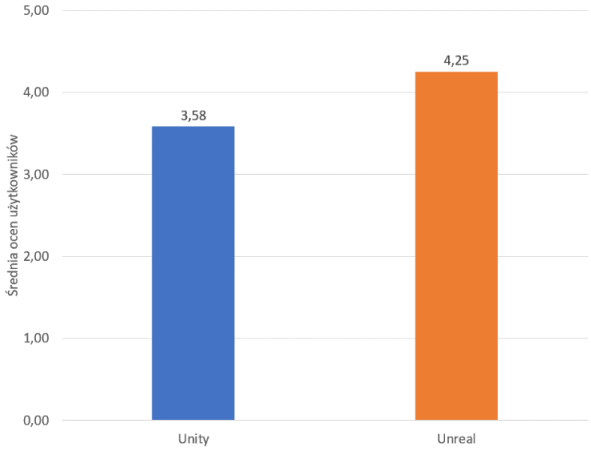


Rysunek 4: Wykres średnich ocen w kategorii ogólne wrażenia wizualne.



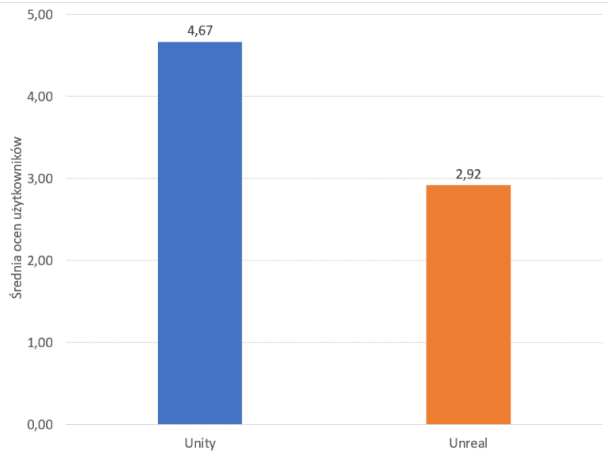
Rysunek 5: Wykres średnich ocen w kategorii efekty otoczenia oraz jakość grafiki.

W ramach kategorii "animacje postaci oraz otoczenia" (Rysunek 6) wyniki analizy wykazują, że zarówno Unity, jak i Unreal Engine 5 osiągnęły wysokie oceny w tej kategorii. Chociaż wyniki były do siebie najbardziej zbliżone, to Unreal Engine 5 nadal uzyskał wyższą ocenę. Unity otrzymało wynik 3,58, wskazując na solidną jakość animacji, podczas gdy Unreal Engine 5 zyskał jeszcze wyższą ocenę 4,25, co świadczy o jeszcze lepszej jakości animacji w tym silniku.



Rysunek 6: Wykres średnich ocen w kategorii animacje postaci oraz otoczenia.

Wśród wszystkich analizowanych kategorii, szczególną uwagę zwraca kategoria „płynność działania" (Rysunek 7). W której jako jedynej Unity otrzymało wyższy wynik niż Unity z oceną 4,67, a Unreal Engine 5 otrzymał jedyną ocenę poniżej progu 4,0. To wskazuje na pewne niedoskonałości w obszarze optymalizacji silnika Unreal Engine 5, które prowadzą do niezadowolenia użytkowników z utworzonej aplikacji.



Rysunek 7: Wykres średnich ocen w kategorii płynność działania.

5. Wnioski

Głównym celem artykułu było wykazanie podstawowej tezy „Gra stworzona w Unreal Engine 5 wykazuje wyższą wydajność oraz lepsze oceny użytkowników niż gra stworzona w Unity." W celu weryfikacji tezy, zostały sformułowane szczegółowe hipotezy badawcze. Analizując otrzymane wyniki z testu wydajnościowego można zaobserwować, że aplikacja stworzona w silniku Unreal Engine 5 zużywa nawet dwukrotnie więcej zasobów niż aplikacja stworzona w silniku Unity. Widać to w przypadku zużycia procesora gdzie aplikacja

w Unity zużywała średnio 14,3% procesora (CPU) a aplikacja w Unreal 22,5% oraz zużycia pamięci RAM odpowiednio 276 MB oraz 603 MB, co odpowiada przecząco na pytanie badawcze „Czy gra o niedużej złożoności stworzona na silniku Unreal Engine 5 wykorzystuje mniej zasobów od identycznie funkcjonalnie gry stworzonej w Unity?" Z testu można również wywnioskować, że aplikacja stworzona w Unity osiąga znacznie wyższą średnią liczbę klatek na sekundę, wynoszącą 141, w porównaniu do aplikacji opartej na silniku Unreal Engine 5, która uzyskała średnią wartość 94,5 odpowiadając tym samym przecząco na pytanie badawcze „Czy gra o niedużej złożoności stworzona przy użyciu Unreal Engine 5 osiąga większe średnie liczby klatek na sekundę niż podobna gra stworzona w Unity?" Odpowiadając na ostatnie pytanie badawcze „Czy aplikacja stworzona za pomocą Unreal Engine 5 uzyskała wyższe oceny od użytkowników?" Badanie ankietowe wykazało, że aplikacja stworzona w Unreal Engine 5 otrzymała wyższe oceny w niemal wszystkich kategoriach, z wyjątkiem "płynności działania", gdzie aplikacja wykonana w Unity była lepiej oceniana, odpowiadając twierdząco na ostatnie pytanie badawcze. Na podstawie przeprowadzonych badań oraz uzyskanych wyników można stwierdzić, że oba silniki, Unreal Engine 5 i Unity, umożliwiają tworzenie gier o solidnej grafice i wydajności. Jednakże, w porównaniu do Unity, Unreal Engine 5 zapewnia łatwiejszy dostęp do modeli oraz narzędzi umożliwiających tworzenie gier z wysokiej jakości efektami wizualnymi kosztem wydajności. Z drugiej strony, Unity pozwala na tworzenie gier o podobnych funkcjach oraz większej wydajności, ale gorszymi efektami wizualnymi. Warto jednak uwzględnić potencjalny efekt, jaki może wywołać rosnąca złożoność gry na modyfikacje w jej wydajności, co mogłoby prowadzić do zmiany uzyskanych wyników na korzyść drugiej ze stron.

Literatura

[1] A. Šmíd, Comparison of unity and unreal engine, Czech Technical University in Prague (2017) 41-61.

[2] R. Salama, M. Elsayed, A live comparison between Unity and Unreal game engines, Global Journal of Information Technology: Emerging Technologies 11 (2021) 1-7.

[3] H. Al Lawati, The Path of unity or the Path of unreal? A Comparative Study on Suitability for Game Development, Fourth Middle East College Student Research Conference, Muscat, Sultanate of Oman, Journal of Student Research (2019) 791-797.

[4] P. Skop, Comparison of performance of game engines across various platforms, Journal of Computer Sciences Institute 7 (2018) 116-119.

[5] I. Pachoulakis, G. Pontikakis, Combining features of the Unreal and Unity Game Engines to hone development skills, Computing Research Repository (2015) 64-68, https://doi.org/10.48550/arXiv.1511.03640.

[6] C. Lambru, Comparative Analysis of Real-Time Global Illumination Techniques in Current Game Engines, IEEE Access 9 (2021) 125158-125183.

[7] A. M. Barczak, H. Woźniak, Comparative study on game engines. Studia Informatica. Systems and Information Technology, Systemy i Technologie Informacyjne 1-2 (2019) 5-24.

[8] E. Christopoulou, S. Xinogalos Overview and comparative analysis of game engines for desktop and mobile devices. International Journal of Serious Games 4 (2017) 21-36.

[9] A. Andrade, Game engines: A survey. EAI Endorsed Trans. Serious Games 2 (2015) 1-8.

[10] P. Mishra, U. Shrawankar, Comparison between famous game engines and eminent games, International Journal of Interactive Multimedia and Artificial Intelligence 4 (2016) 69-77.

[11] J. Cook XR game development as a tool for authentic, experiential, and collaborative learning, Biochemistry and Molecular Biology Education 49 (2021) 846-847.

[12] A. Ciekanowska, A. Kiszczak-Gliński, K. Dziedzic, Comparative analysis of Unity and Unreal Engine efficiency in creating virtual exhibitions of 3D scanned model, Journal of Computer Sciences Institute 20 (2021) 247-253.

[13] S. Pavkov, I. Franković, N. Hoić-Božić, Comparison of game engines for serious games, In: 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), IEEE (2017) 728-733.

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 317–324
Received: 21 June 2023
Accepted: 29 June 2023

Video game performance analysis on selected operating systems

Analiza wydajności gier komputerowych na wybranych systemach operacyjnych

Agata Wrześniewska*, Maria Skublewska-Paszkowska

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The video game industry is currently one of the most dominant in IT. Unfortunately, developers rarely focus on maintaining older games, which often leads to the inability to launch them on newer systems. The aim of the paper is the video game performance analysis on selected operating systems. The analysis was performed on the first three installments of The Sims series, published in the first decade of the 21st century, on a computer with Windows XP as the operating system and another with Windows 10 as the operating system. For the performance analysis three hardware monitoring programs were used: Open Hardware Monitor, MSI Afterburner and Windows Performance Monitor. In addition, all tested games were compared visually in order to determine whether their appearance and available graphic options are the same on both systems. Results have shown, that despite lower system load in all games on the computer with Windows 10 there are some graphical anomalies not present on the older operating system.

Keywords: video games; operating system; performance analysis; The Sims

Streszczenie

Rynek gier komputerowych jest obecnie jednym z najbardziej dominujących w całym przemyśle informatycznym. Niestety nie zawsze producenci skupiają się na konserwacji starszych produkcji, co często uniemożliwia ich uruchomienie na nowszych systemach. Celem artykułu jest analiza wydajności gier komputerowych na wybranych systemach operacyjnych. Analizie poddano wydajność trzech pierwszych gier serii The Sims, wydanych w pierwszej dekadzie XXI wieku, na konfiguracji sprzętowej z systemem Windows XP oraz na drugiej z systemem Windows 10. Do wykonania badań wykorzystano trzy programy pozwalające na monitorowanie wydajności systemu oraz stanu jego podzespołów: Open Hardware Monitor, MSI Afterburner oraz Monitor Wydajności systemu Windows. Dodatkowo badane gry zostały również porównane pod kątem graficznym w celu stwierdzenia, czy ich wygląd oraz dostępne opcje dostosowania grafiki pozostaną niezmienne w obu konfiguracjach. Analiza wyników wykazała, iż pomimo mniejszego obciążenia systemu nowszego w przypadku każdej z gier, pojawiały się w nich pewnie anomalia graficzne nieobecne na starszym systemie.

Słowa kluczowe: gry komputerowe; system operacyjny; analiza wydajności; The Sims

*Corresponding author

Email address: agata.wrzesniewska@pollub.edu.pl (A. Wrześniewska)

1. Wstęp

Rynek gier komputerowych jest obecnie jednym z najbardziej dominujących w całym przemyśle informatycznym. Dynamiczny rozwój technologii, który można było zaobserwować przez ostatnie dwadzieścia lat, przyczynił się do umocnienia pozycji gier nie tylko jako formy rozrywki, ale również nieodłącznej części kultury. Można jednak zauważyć, że o ile gry często są traktowane jako forma sztuki [1], wielu ich producentów nie przykłada się do konserwacji starszych produkcji, często uniemożliwiając ich uruchomienie na nowszych systemach. Dostępne na rynku starsze gry mogą być również wprowadzane do sprzedaży bez ich wcześniejszego dostosowania do nowszych komputerów, powodując przez to błędy graficzne oraz spadki wydajności.

Różnice generacyjne między komputerami z pierwszej dekady XXI wieku a dzisiejszymi są znaczne. Maksymalna ilość obsługiwanej pamięci RAM w 32-bitowych systemach Windows XP Home Edition wynosiła tylko 4GB, w przypadku systemu Windows 10 Home jest to już 128GB [2]. Zwiększyła się ilość rdzeni w procesorach, a także liczba pamięci wideo w kartach graficznych. Wszystkie te aspekty przyczyniły się do powstawania bardziej wymagających gier komputerowych oraz generowania bardziej złożonych obrazów 3D. Pojawiać się mogą jednak problemy związane z kompatybilnością. Nowe karty graficzne mogą być nierozpoznawane, zbyt duża ilość pamięci RAM wychodzi poza wartości akceptowalne, co może przyczyniać się do znacznego pogorszenia działania danych gier.

Celem niniejszej pracy jest analiza wydajności wybranych gier z pierwszej dekady XXI wieku na konfiguracjach systemowych dedykowanych dla badanych gier oraz współczesnych, a także ich porównanie pod względem graficznym.

2. Przegląd literatury

Pełne przejście do środowisk 64-bitowych niosło ze sobą wiele problemów w związku z czym pierwszym

wprowadzonym do powszechnego użytku systemem 64-bitowym był dopiero Windows XP Professional z 2005 roku [3]. Kod, który na jednej platformie był jak najbardziej bezpieczny, na drugiej może stać się wrażliwy (np. przez zmiany długości liczb całkowitych) [4].

Oprócz zmian w platformach systemowych rozwój technologiczny dotyczył również podzespołów komputerowych, w tym istotnych do badań kart graficznych. Pomimo przewidywań, że dynamiczny rozwój kart graficznych zwolni w drugiej dekadzie XXI wieku oraz że większość z urządzeń będzie korzystać ze zintegrowanych rozwiązań do przetwarzania grafiki [5], obecnie karty graficzne są wykorzystywane w wielu dziedzinach, nie tylko naukowych. Powszechnie stosowane jest przetwarzanie danych za pomocą GPU (ang. GPU computing). Pojęcie to przybliżają autorzy artykułu zatytułowanego „GPU Computing" [6]. Ciągły rozwój kart graficznych jest również często motywowany dążeniem do osiągnięcia realizmu wizualnego w np. grach komputerowych, których rynek jest kluczowy w procesie rozwoju GPU [7].

Badania dotyczące gier komputerowych obecnie są coraz liczniejsze. Zainteresowanie pojęciem chociażby interakcji między graczem a grą zaczęło wzrastać w pierwszej dekadzie XXI wieku [8]. W jednym ze znalezionych artykułów autorzy przeprowadzili internetową ankietę dotyczącą nawyków, preferencji oraz doświadczeń użytkowników podczas grania w gry komputerowe [9]. Wyniki ankiety pozwoliły autorom stwierdzić, że na czas spędzany przy grach wpływają głównie doświadczenia graczy z daną grą. W kolejnym artykule autorzy skupili się na pojęciu interakcji człowiek-komputer (HCI, ang. Human-computer interaction) w kontekście gier komputerowych i jak różni się ono od standardowych interakcji ze zwykłym oprogramowaniem [10]. Autorzy podkreślają, że gry komputerowe są tworzone z predefiniowanymi aktywnościami, które mają za zadanie wykonywać użytkownicy, w przeciwieństwie do programów tworzonych z myślą o zadaniach definiowanych przez użytkowników.

Gry komputerowe jako medium wymagają odmiennego podejścia do testów niż to stosowane powszechnie przy innym oprogramowaniu. W jednym z artykułów wspomniano o tym, że profesjonaliści są często niechętni do używania np. testów automatycznych w grach ze względu na ciągłe zmiany w procesie ich wytwarzania w kwestii wymagań oraz projektowania [11]. Podobne wnioski wyciągnęli autorzy artykułu zatytułowanego „Towards Automated Video Game Testing: Still a Long Way to Go" [12]. Ze znalezionych artykułów wynika, że testy gier powinny łączyć ze sobą aspekty techniczne, kreatywne oraz artystyczne [13]. W kolejnym artykule z wywiadów z siedmioma zespołami wytwarzającymi gry komputerowe wynikło, że głównymi elementami, na których twórcy skupiają się przy testach, jest zawartość gry oraz doświadczenie użytkownika (ang. user experience), a nie wydajność czy niezawodność [14]. Błędy graficzne oraz wydajnościowe mogą jednak również negatywnie wpływać na doświadczenie użytkownika, szczególnie jeśli bezpośrednio szkodzą płynności rozgrywki.

Rodzaje błędów występujących w grach komputerowych przybliża artykuł „What went wrong: A taxonomy of video game bugs" [15]. Kategorie błędów dotyczyły m.in. błędnie wczytywanych zasobów graficznych. Jednym z przykładowych rozwiązań służących do odnajdywania tego typu błędów jest debugowanie gier komputerowych w trakcie ich działania (ang. runtime monitoring) [16]. Każda gra oparta jest na tzw. „pętli gry" (ang. „game loop"), którą można użyć w procesie testowania, edytując, przy pomocy kodu, jej parametry podczas trwania rozgrywki.

Porównania wydajnościowe oraz obciążeniowe gier komputerowych pojawiły się w kilku znalezionych artykułach. W jednym z nich autorzy skupili się głównie na charakterystykach obciążeniowych kilku wybranych przez nich gier 3D [17]. Zastosowali do tego testy obciążeniowe, istotne w procesie wytwarzania oprogramowania [18]. W kolejnym artykule wykonano podobne porównanie, tym razem pod kątem wydajności silników gier wideo Unity oraz CryEngine [19]. Jeden z artykułów przedstawiał również przykład użycia programu 3DMark podczas testowania wydajności gier 3D [20]. Autorzy artykułu sprawdzili, w jaki sposób będą zmieniać się wartości przy różnej liczbie rdzeni procesora, różnej częstotliwości taktowania procesora, różnej liczbie GPU oraz różnej liczbie wspieranych wątków. Otrzymane wyniki pozwoliły na wywnioskowanie, że gry 3D zyskują na wydajności przy większej liczbie rdzeni procesora w przeciwieństwie do jedynie większego taktowania przy np. jednym rdzeniu.

Do mierzenia wydajności oraz obciążenia systemu wykorzystywane jest zazwyczaj oprogramowanie monitorujące stan podzespołów komputera. Przykład ich wykorzystania opisany został w artykule „The basics of performance-monitoring hardware" [21]. Autor pisze o dwóch podejściach do profilowania wydajności: opartej na czasie (time-based profiles) oraz na zdarzeniach (event-based profiles). Pierwsza z nich przerywa działanie aplikacji w wyznaczonych odstępach czasu oraz zapisuje dane z jej działania. Druga przerywa działanie programu po wykonaniu określonej liczby zdarzeń. W niniejszej pracy wykorzystano podejście podobne do opisanego przez autora profilowania wydajności opartego na czasie. Pomiary parametrów podzespołów komputera były również wykonywane w określonych odstępach czasu z tą różnicą, że działanie gier w trakcie wykonywania pomiaru nie było przerywane.

Przegląd artykułów wykazał, że do tej pory nie przeprowadzano badań dotyczących analizy wydajności gier komputerowych na różnych systemach operacyjnych, dlatego też zdecydowano się poddać tę tematykę analizie. Sformułowano następujące hipotezy badawcze:

H1: Obciążenie starszego systemu podczas uruchomienia gry jest większe niż obciążenie nowego systemu.

H2: Badane gry są bardziej podatne na wystąpienie błędów graficznych na nowszych systemach.

3. Metodyka badań

3.1. Narzędzia

Do analizy wydajności i obciążenia systemu podczas uruchomienia gry oraz w stanie bezczynnym zostaną wykorzystane trzy narzędzia:

1. Open Hardware Monitor - otwartoźródłowy program do monitorowania stanu podzespołów komputerowych,
2. Monitor Wydajności systemu Windows – wbudowane narzędzie systemu Windows, pozwalające na monitorowanie stanu systemu,
3. MSI Afterburner – narzędzie firmy MSI, głównie wykorzystywane do monitorowania stanu karty graficznej, umożliwia dodatkowo wyświetlanie nakładki z aktualnymi parametrami podzespołów komputera podczas uruchomienia gry.

Wszystkie trzy z wymienionych narzędzi posiadają wersję na systemy 32- i 64-bitowe oraz oferują opcję zapisu danych w ustalonych odstępach czasowych do pliku .csv bądź pliku o podobnym formacie danych (np. .hml).

3.2. Badana seria gier

The Sims jest serią gier z gatunku symulatorów życia [22]. Główna rozgrywka polega na zarządzaniu parcelami, zamieszkałymi przez „simów", czyli wirtualnych ludzi. Każda z gier posiada przynajmniej jedno otoczenie z domyślnym zestawem zamieszkałych bądź pustych parcel, który można rozszerzać. Na parcelach gracz ma do wyboru szereg obiektów, z którymi „simowie" są w stanie wchodzić w interakcję. Głównym celem gry jest spełnianie potrzeb mieszkańców parcel oraz rozwój ich umiejętności.

3.3. Pomiary

Pomiary były przeprowadzane przez pół godziny w odstępach 30 sekund dla większego obciążenia (podczas wykonywania czynności w grze) oraz minuty dla mniejszego obciążenia (gra w stanie bezczynnym oraz komputer w stanie bezczynnym bez uruchomionej gry). Do wykonania pomiarów użyto narzędzi opisanych w sekcji 3.1, każde z nich zostało skonfigurowane, aby zapisywać do pliku .csv (bądź .hml w przypadku narzędzia MSI Afterburner, plik .hml ma strukturę analogiczną do pliku .csv) wartość danego parametru komputera w ustalonych wcześniej odstępach. Mierzone będą:

- temperatura procesora,
- temperatura karty graficznej,
- obciążenie procesora,
- zużycie pamięci RAM,
- zużycie pamięci VRAM.

Tabela 1: Specyfikacja komputera starszego

Procesor	Intel Core 2 Duo E8200 @2,66GHz
Karta graficzna	NVIDIA GeForce 8600 GT 512MB VRAM
Pamięć RAM	3,25GB
System operacyjny	Windows XP Home Edition 32-bit (Service Pack 3)
Wersja DirectX	9.0
Wersja .NET	4.0

Analizie poddane będą trzy pierwsze części serii The Sims (wydane odpowiednio w 2000, 2004 i 2009 roku), uruchamiane na systemach Windows 10 (64-bitowy) na nowszej konfiguracji sprzętowej oraz Windows XP (32-bitowy) na starszej konfiguracji sprzętowej. W tabelach 1 i 2 przedstawiono specyfikację każdej z konfiguracji sprzętowych.

Tabela 2: Specyfikacja komputera nowszego

Procesor	Intel Core i5 7600K @3,80GHz
Karta graficzna	NVIDIA GeForce GTX 1050Ti 4GB VRAM
Pamięć RAM	16GB
System operacyjny	Windows 10 Home 64-bit (22H2)
Wersja DirectX	12.0
Wersja .NET	4.8

4. Wyniki

4.1. Porównanie wydajnościowe

Za stan bezczynny w grze przyjęto pozostawienie uruchomionej gry bez wykonywania żadnych czynności po załadowaniu głównego miejsca rozgrywki – w przypadku badanej serii było to załadowanie otoczenia, z którego można było przejść do stanu aktywnego. Za stan aktywny przyjęto wykonywanie czynności w grze na zajętych parcelach. Pomiary wykazały, że stan aktywny i bezczynny w grach obciążają system na podobnym poziomie, dlatego w dalszej części pracy analizie poddane zostaną jedynie wyniki dla stanu aktywnego.

4.1.1.Stan bezczynny

W tabeli 3 przedstawiono średnie wartości mierzonych parametrów dla odpowiednio komputera w starszej konfiguracji sprzętowej z systemem Windows XP oraz dla komputera w nowszej konfiguracji sprzętowej z systemem Windows 10.

Tabela 3: Średnia oraz odchylenie standardowe. poszczególnych parametrów w stanie bezczynnym komputera z Windowsem XP oraz Windowsem 10

Parametr	Win XP		Win 10	
	$\bar{x}$	σ	$\bar{x}$	σ
Temperatura procesora [°C]	42,00	1,39	30,59	1,56
Temperatura karty graficznej [°C]	49,59	0,61	26,97	0,18
Obciążenie procesora [%]	0,05	0,07	3,76	2,07
Zużycie pamięci RAM [MB]	533,72	1,30	3831,56	55,20
Zużycie pamięci VRAM [MB]	1,34	0,00	380,55	1,96

Temperatury procesora oraz karty graficznej są w przypadku Windowsa XP prawie dwukrotnie wyższe. Windows 10 natomiast posiada bardziej obciążający

interfejs graficzny zużywając ok. 9% całkowitej pamięci VRAM karty w porównaniu do ok. 0,3% pamięci zużywanej przez system Windows XP. Obciążenie procesora w przypadku systemu nowszego było nieznacznie większe.

4.1.2.Starsza konfiguracja sprzętowa

W tabeli 4 przedstawiono wyniki pomiarów dla gry The Sims w stanie aktywnym.

Tabela 4: Windows XP stan aktywny – The Sims

Parametr	Średnia	Odchylenie std.
Temperatura procesora [°C]	63,43	1,35
Temperatura karty graficznej [°C]	49,00	0,00
Obciążenie procesora [%]	50,43	0,13
Zużycie pamięci RAM [MB]	857,95	0,68
Zużycie pamięci VRAM [MB]	2,84	0,00

Zużycie pamięci VRAM w tym przypadku było stałe i wynosiło mniej niż 1% całkowitej pamięci VRAM karty graficznej, dlatego stwierdzono, że jest to gra głównie obciążająca procesor. Wynika to z faktu, iż jest to gra korzystająca z modeli 3D jedynie przy postaciach, otoczenie oraz obiekty natomiast nie są renderowane w czasie rzeczywistym przez procesor/kartę graficzną, a tworzone techniką tzw. pre-renderingu (zasoby tworzone są wcześniej, natomiast podczas gry są odtwarzane jak np. plik wideo).

W tabeli 5 przedstawiono wyniki pomiarów dla gry The Sims 2. Można zauważyć znaczny wzrost wykorzystanej pamięci VRAM, w tym przypadku jest to już ponad 20%. W odróżnieniu od The Sims jest to gra, która korzysta z grafiki 3D w pełni również do renderowania obiektów oraz otoczenia, stąd do przetwarzania grafiki w znacznej części wykorzystuje kartę graficzną. Wzrosła również ilość wykorzystywanej pamięci RAM (różnica ok. 500MB). Obciążenie procesora natomiast pozostawało na podobnym poziomie ok. 50%.

Tabela 5: Windows XP stan aktywny – The Sims 2

Parametr	Średnia	Odchylenie std.
Temperatura procesora [°C]	60,07	4,92
Temperatura karty graficznej [°C]	51,44	2,05
Obciążenie procesora [%]	50,50	8,89
Zużycie pamięci RAM [MB]	1320,66	171,40
Zużycie pamięci VRAM [MB]	146,95	45,01

Znaczne różnice w obciążeniu systemu można zauważyć porównując dwie pierwsze gry z trzecią, której wyniki pomiarów przedstawiono w tabeli 6. W tym przypadku wykorzystywana pamięć VRAM jest o ok. 200MB wyższa od wartości osiąganej w przypadku gry The Sims 2, a obciążenie procesora wynosi aż 70%. Wynika to z faktu, iż w grze The Sims 3 po raz pierwszy pojawia się otwarte otoczenie (wszystkie parcele oraz obiekty renderowane są jednocześnie w czasie rzeczywistym bez ekranów ładowania) w przeciwieństwie do poprzednich części, gdzie jednocześnie w pełni renderowana była tylko jedna parcela.

Tabela 6: Windows XP stan aktywny – The Sims 3

Parametr	Średnia	Odchylenie std.
Temperatura procesora [°C]	66,11	1,82
Temperatura karty graficznej [°C]	65,72	0,86
Obciążenie procesora [%]	70,67	4,66
Zużycie pamięci RAM [MB]	1607,49	122,01
Zużycie pamięci VRAM [MB]	266,12	47,27

4.1.3.Nowsza konfiguracja sprzętowa

W tabeli 7 przedstawiono wyniki pomiarów dla gry The Sims w stanie aktywnym. Zużycie pamięci VRAM w tym przypadku wynosi ok. 7% całkowitej pamięci VRAM karty graficznej, co oznacza, że nastąpił spadek zużycia pamięci VRAM od stanu bezczynnego (różnica ok. 92MB). Porównując ze sobą część procentową całkowitej dostępnej pamięci RAM jaką stanowił przyrost pamięci od stanu bezczynnego komputera do stanu aktywnego gry (czyli faktyczna ilość pamięci RAM, jaką zużywała każda z gier) można zauważyć, że w przypadku nowszego systemu są to wartości mniejsze niż w przypadku systemu starszego. Oznacza to, że system Windows 10 jest obciążany przez badane gry zdecydowanie mniej niż system Windows XP mimo większego przyrostu zużycia pamięci RAM (1161MB na nowszym systemie w porównaniu do 324MB na starszym). Dla starszego systemu ilość wykorzystywanej pamięci RAM wzrosła o ok. 9%, w przypadku nowszego było to ok. 7% wartości całkowitej pamięci RAM komputera.

Tabela 7: Windows 10 stan aktywny – The Sims

Parametr	Średnia	Odchylenie std.
Temperatura procesora [°C]	38,51	2,51
Temperatura karty graficznej [°C]	28,97	0,58
Obciążenie procesora [%]	16,61	2,39
Zużycie pamięci RAM [MB]	4993,10	18,85
Zużycie pamięci VRAM [MB]	288,26	0,00

W tabeli 8 przedstawiono wyniki pomiarów dla gry The Sims 2. Można zauważyć, że w tym przypadku obciążenie procesora nie pozostaje już na tym samym poziomie, co przy grze The Sims, ale wzrasta z 16,61%

do 27,71%. Pomimo wartościowo większego zużycia pamięci VRAM niż na starszym systemie, przyrost w porównaniu do stanu bezczynnego stanowi mniejszą część całkowitej pamięci VRAM niż w przypadku systemu Windows XP (poprzednio było to ok. 29% z wartością 145MB, tym razem jest to tylko ok. 5% z wartością 209MB). Wzrost pamięci RAM również stanowi mniejszą część całkowitej pamięci RAM (ok. 16% z wartością 2662MB w porównaniu do 24% przy systemie Windows XP z wartością 786MB).

Tabela 8: Windows 10 stan aktywny – The Sims 2

Parametr	Średnia	Odchylenie std.
Temperatura procesora [°C]	41,84	3,60
Temperatura karty graficznej [°C]	31,87	0,97
Obciążenie procesora [%]	27,71	5,21
Zużycie pamięci RAM [MB]	6494,41	98,79
Zużycie pamięci VRAM [MB]	589,95	32,48

W tabeli 9 przedstawiono wyniki dla gry The Sims 3. Można zauważyć, że pomimo większego obciążenia procesora oraz wyższych temperatur, The Sims 3 na systemie Windows 10 zużywa średnio mniej zasobów pamięciowych niż The Sims 2. Jest to więc sytuacja odwrotna niż w przypadku systemu Windows XP, gdzie wszystkie wartości wzrastały w każdej kolejnej grze. Tak jak poprzednio, przyrost zużycia pamięci RAM oraz VRAM jest procentowo mniejszy niż w przypadku analogicznym na starszej konfiguracji sprzętowej. W tym wypadku przyrost pamięci RAM stanowi ok. 13% całkowitej pamięci RAM komputera (2175MB) a w przypadku komputera starszego było to 33% (1073MB). Przyrost pamięci VRAM natomiast oprócz stanowienia procentowo mniejszej części całkowitej pamięci VRAM karty graficznej (ok. 3% w porównaniu do 51% na systemie Windows XP) jest również wartością mniejszą niż w przypadku starszego systemu (różnica ok. 130MB, dla systemu Windows 10 było to średnio 137MB, dla systemu Windows XP – 264MB).

Tabela 9: Windows 10 stan aktywny – The Sims 3

Parametr	Średnia	Odchylenie std.
Temperatura procesora [°C]	45,46	1,51
Temperatura karty graficznej [°C]	38,46	0,83
Obciążenie procesora [%]	51,49	1,64
Zużycie pamięci RAM [MB]	6007,02	68,26
Zużycie pamięci VRAM [MB]	518,31	22,81

4.2. Analiza graficzna

Każda z badanych gier testowana była z domyślnymi ustawieniami graficznymi dla danego komputera. Na Rysunku 1 przedstawiony został wygląd oraz ustawienia domyślne gry The Sims 2 na komputerze z systemem Windows XP. Wszystkie opcje graficzne są dostępne do modyfikacji, gra oferuje np. kilka możliwych do wyboru rozdzielczości, ustawienia cieni bądź antyaliasingu. Przy większości z opcji domyślne ustawienia były najwyższe z możliwych.

Na Rysunku 2 przedstawiony został wygląd oraz ustawienia domyślne gry The Sims 2 na komputerze z systemem Windows 10. Czarne prostokąty na ziemi, widoczne na rysunku, to błędnie renderowane cienie. Jedyną opcją graficzną, która naprawiła problem było całkowite wyłączenie cieni. Domyślnie ustawiona została rozdzielczość 800x600 bez możliwości jej dostosowania. Pomimo tego, że podobnie jak na komputerze z systemem Windows XP większość opcji została ustawiona na najwyższą wartość można zauważyć, że oprócz braku dodatkowych rozdzielczości nie jest możliwe również sterowanie antyaliasingiem.

Rysunek 1: Poprawny wygląd cieni w grze The Sims 2 na systemie Windows XP.

Rysunek 2: Uszkodzone cienie w grze The Sims 2 na systemie Windows 10.

W przypadku gry The Sims grafika uszkadza się w specyficznych warunkach. Na Rysunku 3 przedstawiony został poprawny, domyślny wygląd fragmentu otoczenia gry na komputerze z systemem Windows 10.

Rysunek 3: Grafika przed minimalizacją gry The Sims na komputerze z systemem Windows 10.

Po minimalizacji okna z grą następuje deformacja m.in. czcionki oraz palety kolorów wyświetlonego okna, widoczna na Rysunku 4.

Rysunek 4: Grafika po minimalizacji gry The Sims na komputerze z systemem Windows 10.

Oprócz widocznych błędów w wyświetlaniu grafiki gracz nie jest w stanie wykonać żadnej czynności, przestaje działać przycisk wyjścia z gry, a po wejściu w menadżer zadań gra sama się wyłącza.

Rysunek 5: Komunikat o nierozpoznanej karcie graficznej w grze The Sims 3 na komputerze z systemem Windows 10.

Gra The Sims 3 okazała się być najmniej problematyczną pod względem graficznym ze wszystkich badanych gier. Pomimo nielicznych błędów (np. komunikat o nierozpoznanej karcie graficznej, widoczny na Rysunku 5) gracz jest w stanie dostosować wszystkie oferowane przez grę opcje graficzne na wartości maksymalne, nawet jeśli domyślnie, w związku z niewykrywaną kartą graficzną, niektóre z opcji są ustawione na najniższe.

5. Dyskusja

Porównując ze sobą wyniki uzyskane w obu konfiguracjach sprzętowych można zauważyć pewne zależności. Jedną z nich jest fakt, iż na komputerze z systemem Windows XP im nowsza była testowana gra, tym więcej zasobów zużywała i bardziej obciążała procesor. W przypadku komputera z systemem Windows 10 zależność ta występuje jedynie częściowo, ponieważ dla gry The Sims 2 pomimo mniejszego procentowego obciążenia procesora niż przy grze nowszej i bardziej wymagającej graficznie, zużywa ona więcej pamięci VRAM oraz RAM niż gra The Sims 3. Może to świadczyć o występowaniu pewnych problemów w zakresie optymalizacji gry na nowszych systemach operacyjnych.

Jak wspomniano w sekcji 4, wartości temperatury w konfiguracji sprzętowej z systemem Windows 10 były w każdym z testowanych przypadków o połowę mniejsze niż przy systemie Windows XP. W obu przypadkach, według strony producenta [23], były to jednak temperatury mieszczące się w granicach temperatur dopuszczalnych dla danego procesora.

W przypadku obciążenia procesora można zauważyć, że dla systemu Windows 10 wartości procentowego obciążenia były mniejsze niż dla systemu Windows XP. Patrząc na zużycie zasobów pamięci VRAM oraz RAM widać, że gry uruchamiane na nowszym systemie zużywają więcej zasobów niż na starszym. Porównując jednak przyrost tych wartości od stanu bezczynnego komputera, można zauważyć, że przyrost ten stanowi w każdym z przypadków mniejszą część całkowitej pamięci RAM oraz VRAM niż dla systemu starszego.

Porównanie pod względem graficznym badanych gier wykazało szereg problemów pojawiających się jedynie na konfiguracji sprzętowej z systemem Windows 10. Najgorzej z testowanych gier wypadła gra The Sims 2. Oprócz błędnie renderowanych cieni, zablokowane były również niektóre opcje graficzne w ustawieniach jak chociażby sterowanie antyaliasingiem oraz zmiana rozdzielczości. Dla gry The Sims błędne wyświetlanie interfejsu pojawiało się po minimalizacji gry. Gra przestawała być wtedy również responsywna. W grze The Sims 3 pojawiał się komunikat o nierozpoznawanej karcie graficznej, fakt ten powodował ustawienie niższych domyślnych opcji graficznych niż w przypadku konfiguracji sprzętowej z systemem Windows XP. W przeciwieństwie jednak do gry The Sims 2 żadna z opcji graficznych nie była zablokowana i mogły być one przestawione na wyższe wartości.

Na podstawie uzyskanych wyników (Tabele 3-9) można stwierdzić, iż pierwsza hipoteza dotycząca więk-

szego obciążenia systemu starszego została potwierdzona. Jak wspomniano wcześniej, pomimo zużywania większych zasobów pamięciowych przyrost tych wartości w porównaniu do stanu bezczynnego stanowi mniejszą część całkowitej pamięci RAM/VRAM niż w przypadku systemu Windows XP. Procentowe obciążenie procesora jest zdecydowanie mniejsze, a osiągane przez procesor oraz kartę graficzną temperatury są niższe.

Odnosząc się do sekcji 4.2 można zauważyć, że na systemie Windows 10 pojawiały się błędy graficzne nieobecne na systemie Windows XP. Porównując ze sobą rysunki 1 i 2 dotyczące gry The Sims 2 wyraźnie widać, że na nowszym systemie oprócz zablokowanych opcji sterowania grafiką pojawiają się również uszkodzone cienie. Problemy związane z grafiką pojawiają się również w przypadku gry The Sims (Rysunek 4), w przypadku gry The Sims 3 dotyczą głównie nierozpoznania karty graficznej (Rysunek 5). Na podstawie wyżej wymienionych obserwacji można stwierdzić, że hipoteza druga dotycząca większej podatności nowszych systemów na błędy graficzne została również potwierdzona.

6. Podsumowanie

Celem niniejszej pracy było zbadanie oraz analiza wydajności wybranych gier z początkowej dekady XXI wieku na konfiguracjach systemowych dedykowanych (system Windows XP) oraz współczesnych (system Windows 10), a także ich porównanie graficzne. Do realizacji założonego celu wykorzystano trzy programy monitorujące stan podzespołów komputera oraz jego obciążenie, a badaniom poddano trzy pierwsze części serii gier The Sims.

Otrzymane wyniki pozwoliły na stwierdzenie, że pomimo większego obciążenia systemu w starszej konfiguracji systemowej, w konfiguracji współczesnej pojawiają się błędy graficzne, które mogą powodować utrudnienia w rozgrywce a nawet jej przerwanie. Może to wynikać z braku optymalizacji starszych gier pod kątem systemów z późniejszych generacji, co ostatecznie może doprowadzić do braku możliwości ich uruchomienia na coraz nowszych systemach.

W pracy skupiono się głównie na analizie wydajności przy domyślnych ustawieniach graficznych, więc analiza wpływu zmiany ustawień graficznych na wydajność oraz działanie gier mogłaby być przedmiotem dalszych badań. Dodatkowo analizie mogłyby być poddane gry z innych gatunków niż użyte na potrzeby artykułu symulatory życia. Ciekawą mogłaby okazać się analiza gier z gatunku roguelike, w którym każde nowe pomieszczenie w grze jest generowane losowo.

Literatura

[1] J. R. Parker, Games are art: Video games as theatrical performance, IEEE Consumer Electronics Society's International Games Innovations Conference (2013) 203-208, https://doi.org/10.1109/igic.2013.6659148.

[2] Limity pamięci systemów operacyjnych Windows i Windows Server, https://learn.microsoft.com/en-us/windows/win32/memory/memory-limits-for-windows-releases, [05.05.2023].

[3] J. R. Mashey, The long road to 64 bits, ACM Queue 4(8) (2006) 24-35, https://doi.org/10.1145/1165754.1165766.

[4] C. Wressnegger, F. Yamaguchi, A. Maier, K. Rieck, Twice the bits, twice the trouble: Vulnerabilities induced by migrating to 64-bit platforms, Proceedings of the ACM Conference on Computer and Communications Security (2016) 541-552, https://doi.org/10.1145/2976749.2978403.

[5] M. Doggett, Texture Caches, IEEE Micro 32(3) (2012) 136-141, https://doi.org/10.1109/mm.2012.44.

[6] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, J. C. Phillips, GPU computing, Proceedings of the IEEE 96(5) (2008) 879-899, https://doi.org/10.1109/jproc.2008.917757.

[7] D. Blythe, Rise of the Graphics Processor, Proceedings of the IEEE 96(5)(2008) 761–778, https://doi.org/10.1109/jproc.2008.917718.

[8] L. Caroux, K. Isbister, L. L. Bigot, N. Vibert, Player-video game interaction: A systematic review of current concepts, Computers in Human Behavior 48 (2015) 366-381, https://doi.org/10.1016/j.chb.2015.01.066.

[9] D. Johnson, J. Gardner, P. Sweetser, Motivations for videogame play: Predictors of time spent playing, Computers in Human Behavior 63 (2016) 805-812, https://doi.org/10.1016/j.chb.2016.06.028.

[10] P. Barr, J. Noble, R. Biddle, Video game values: Human-computer interaction and games, Interacting with Computers 19(2) (2007) 180-195, https://doi.org/10.1016/j.intcom.2006.08.008.

[11] R. E. S. Santos, C. V. C. Magalhes, L. F. Capretz, J. S. Correia-Neto, F. Q. B. Da Silva, A. Saher, Computer games are serious business and so is their quality: Particularities of software testing in game development from the perspective of practitioners, Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (2018) 1-10, https://doi.org/10.1145/3239235.3268923.

[12] C. Politowski, Y. G. Guéhéneuc, F. Petrillo, Towards automated video game testing, Proceedings of the 6th International ICSE Workshop on Games and Software Engineering: Engineering Fun, Inspiration, and Motivation (2022) 37-43, https://doi.org/10.1145/3524494.3527627.

[13] F. T. Tschang, Videogames as Interactive Experiential Products and their Manner of Development, International Journal of Innovation Management 09(01) (2005) 103–131, https://doi.org/10.1142/s1363919605001198.

[14] J. Kasurinen, K. Smolander, What do game developers test in their products? Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14 (2014) 1-10, https://doi.org/10.1145/2652524.2652525.

[15] C. Lewis, J. Whitehead, N. Wardrip-Fruin, What went wrong: A taxonomy of video game bugs, Proceedings of the 5th International Conference on the Foundations of Digital Games (2010) 108-115, https://doi.org/10.1145/1822348.1822363.

[16] S. Varvaressos, K. L. Lavoie, S. Gaboury, S. Hallé, Automated Bug Finding in Video Games: A case study

for runtime monitoring, Computers in Entertainment 15(1) (2017) 1–28, https://doi.org/10.1145/2700529.

[17] J. Roca, V. Moya, C. Gonzalez, C. Solis, A. Fernandez, R. Espasa, Workload Characterization of 3D Games, IEEE International Symposium on Workload Characterization (2006) 17-26, https://doi.org/10.1109/iiswc.2006.302726.

[18] H. AlGhamdi, C. Bezemer, W. Shang, A. E. Hassan, P. Flora, Towards reducing the time needed for load testing, Journal of Software 35(3) (2020) 1-17, https://doi.org/10.1002/smr.2276.

[19] H. Żukowski, Comparison of 3D games' efficiency with use of CRYENGINE and Unity game engines, Journal of Computer Sciences Institute 13 (2019) 345–348, https://doi.org/10.35784/jcsi.1330.

[20] F. N. Sibai, 3D graphics performance scaling and workload decomposition and analysis, 6th IEEE/ACIS International Conference on Computer and Information Science ICIS; 1st IEEE/ACIS International Workshop on e-Activity IWEA (2007) 604-609, https://doi.org/10.1109/icis.2007.3.

[21] B. Sprunt, The basics of performance-monitoring hardware, IEEE Micro 22(4) (2002) 64–71, https://doi.org/10.1109/mm.2002.1028477.

[22] Seria gier komputerowych The Sims, https://en.wikipedia.org/wiki/The_Sims, [18.06.2023].

[23] Informacje na temat temperatur procesorów Intel, https://www.intel.pl/content/www/pl/pl/support/articles/000005597/processors.html, [03.06.2023].

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 325–332
Received: 22 June 2023
Accepted: 10 July 2023

Analysis of the ergonomics of interfaces of popular e-marketing tools

Analiza ergonomii interfejsów popularnych narzędzi e-marketingu

Weronika Studzińska*

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

This article analyses the ergonomics of the interfaces of selected e-marketing tools. The main aim of the study was to verify whether all the examined tools have a similar user interface quality. The research was conducted using the cognitive walkthrough method, a modified LUT list with WUP scores, and a questionnaire for assessing tool usability based on Nielsen heuristics. Three tools designed for Internet monitoring were analysed - Brand24, Mention and Awario. The experiment showed that, in terms of the ergonomics of the solutions, not all tools have a similar interface quality. For two of the analysed tools, similar results and ratings were obtained, while one of them presented a noticeably lower level.

Keywords: ergonomics of interfaces; e-marketing; cognitive walkthrough

Streszczenie

W artykule dokonano analizy ergonomii interfejsów wybranych narzędzi e-marketingu. Głównym celem pracy było zweryfikowanie, czy wszystkie badane narzędzia mają podobną jakość interfejsu użytkownika. Badania zostały przeprowadzone przy wykorzystaniu metody wędrówki poznawczej, zmodyfikowanej listy LUT wraz z punktami WUP oraz ankiety pozwalającej na ocenę użyteczności narzędzi w oparciu o heurystyki Nielsena. Analizie poddano trzy narzędzia przeznaczone do monitoring Internetu – Brand24, Mention i Awario. Eksperyment wykazał, że pod względem ergonomii rozwiązań nie wszystkie narzędzia mają podobną do siebie jakość interfejsu. Dla dwóch analizowanych narzędzi uzyskiwano zbliżone wyniki i oceny, jedno z nich prezentowało zaś zauważalnie niższy poziom.

Słowa kluczowe: ergonomia interfejsów; e-marketing; wędrówka poznawcza

*Corresponding author

Email address: weronika.studzinska@pollub.edu.pl (W. Studzińska)

1. Wstęp

Ciągły postęp technologii i automatyzacja społeczeństwa wpłynęły na znaczący wzrost popularności marketingu internetowego. Z biegiem czasu staje się on coraz bardziej efektywną formą dotarcia do potencjalnego klienta i kontaktu z nim. Jest szczególnie przydatny do pozyskania konsumentów reprezentujących młodsze pokolenia [1]. Opracowanie właściwej strategii marketingowej oraz wykorzystanie narzędzi e-marketingu może znacząco wpłynąć na rozwój i sukces biznesu. Ważne staje się więc korzystanie z różnego rodzaju narzędzi ułatwiających jego prowadzenie. Narzędzia te powinny cechować się ergonomią, efektywnością oraz łatwością obsługi i odbioru interfejsu także w przypadku użytkowników, którzy nie mają doświadczenia w korzystaniu z tego typu programów. W związku z tym ważne jest by podczas projektowania i tworzenia oprogramowania kierować się istniejącymi kryteriami i regułami projektowania interfejsów użytkownika [2], które zapewnią pozytywny odbiór systemu przez potencjalnego użytkownika [3, 4].

Istnieje wiele artykułów odnoszących się do pojęcia e-marketingu, stosowanych w nim metod oraz narzędzi. W artykule [5] przedstawiono charakter środowiska cyfrowego. Zaprezentowano podstawowe narzędzia marketingu jakimi są strona internetowa, reklamy online, e-mail i komunikatory internetowe, media społecznościowe, narzędzia do monitorowania i pomiaru. Omówiono również wykorzystanie tych narzędzi. Artykuł dowodzi, że w zależności od sposobu ich zastosowania, narzędzia marketingu cyfrowego mogą być wykorzystywane do różnych celów. Mogą być sposobem do podkreślenia obecności firmy na rynku, spotkań z klientami czy obsługi klientów online.

Artykuł [6], w oparciu o analizę trendów biznesowych z 2017 roku, przedstawia główne powody stosowania strategii marketingowej. Dokonano w nim zestawienia ponad 50 narzędzi i platform stosowanych w marketingu internetowym. Opisano najlepsze sposoby opracowywania strategii marketingowej. Zaprezentowano również kryteria wyboru narzędzi do marketingu internetowego oraz wskazówki dotyczące ich efektywnego zastosowania. W artykule przedstawiono pojęcie benchmarkingu oraz zaprezentowano, w jaki sposób może on poprawić konkurencyjność firmy.

W literaturze można znaleźć wiele publikacji dotyczących oceny interfejsów użytkownika [7] oraz metod stosowanych podczas dokonywania ewaluacji [8, 9]. W zależności od celu przeprowadzenia oceny, wielkości systemu oraz ogólnych warunków do przeprowadzenia badania, eksperci stosują różne metody analizy, często wykorzystywane jest również połączenie różnych metod. W artykule [10] przedstawiono definicje użyteczności i dostępności interfejsu użytkownika oraz omówiono metody ich badania i oceniania. Dokonano klasyfikacji metod oceny użyteczności a następnie omówiono szczegółowo wybrane metody – prototypowanie, anali-

zę heurystyczną, wędrówkę poznawczą oraz testy użyteczności. W pracy [11] dokonano porównania jakości interfejsu użytkownika aplikacji mobilnych przeznaczonych dla klientów wybranych banków, do którego wykorzystano cztery metody - ankiety, testy korytarzowe, testy eksperckie oraz eye-tracking. Autorzy pracy [12] skupili się na wykorzystaniu metody KLM (Keystroke Level Model) do udowodnienia, że mysz komputerowa może nie zawsze być najodpowiedniejszym przyrządem dla każdego użytkownika. Przeanalizowano drogę kursora, czas przebycia drogi oraz prędkość kursora. Jedną z popularniejszych metod oceny interfejsu jest metoda wędrówki poznawczej. Metodę tą zastosowali autorzy artykułu [13] do przeprowadzenia badania oceny użyteczności systemu informacji pielęgniarskiej, podczas której na podstawie pięciu scenariuszy zidentyfikowano problemy interfejsu i przyporządkowano je do poszczególnych atrybutów użyteczności.

W artykule [14] autorzy omówili proces poprawy jakości GUI aplikacji wchodzącej w skład systemu ERP. Do przeprowadzenia badań wykorzystano dwie metody – analizę ekspercką oraz wędrówkę poznawczą. Dla metody wędrówki poznawczej opracowana została lista LUT zawierająca pytania i kryteria oceny odnoszące się do poszczególnych obszarów interfejsu, analizowanych w tej metodzie. Wyznaczona została również ocena całego interfejsu. Po przeprowadzeniu badań okazało się, że ogólna ocena jest dość niska i zgłoszonych zostało wiele uwag. Badania dowiodły, że zastosowana metoda ekspercka umożliwiła ocenę interfejsu użytkownika aplikacji oraz zdefiniowanie jego słabych punktów, co pozwoliło na wyeliminowanie błędów.

Autor pracy [15] poszukiwał najodpowiedniejszego narzędzia do zarządzania projektami. W tym celu przeprowadzono analizę porównawczą kilku przeznaczonych do tego narzędzi. Omówione zostały najpopularniejsze metody przeznaczone do oceny jakości interfejsów użytkownika. Do przeprowadzenia badań wykorzystano ankietę oraz metodę wędrówki poznawczej rozszerzoną o dodatkową ocenę interfejsu w oparciu o przygotowaną listę LUT. Ankieta była skierowana do osób mających doświadczenie w pracy z narzędziami do zarządzania projektami i polegała na ocenie interfejsu wybranego przez respondenta narzędzia. W badaniu metodą wędrówki poznawczej, celem było wykonanie kilku zadań w każdym spośród określonych narzędzi oraz ocenienie za pomocą 5-cio stopniowej skali stopnia trudności wykonania poszczególnych zadań. Uczestnikami badania w metodzie wędrówki poznawczej były osoby, które nie miały doświadczenia z programami do zarządzania projektami. W badaniu oceniana była szybkość wykonywania określonych zadań oraz liczba błędów uczestników. Badanie pozwoliło wskazać program, który ma najbardziej przyjazny interfejs dla nowego użytkownika oraz umożliwiło zidentyfikowanie problemów związanych z użytecznością interfejsów. Badanie ankietowe pozwoliło na wskazanie najpopularniejszych narzędzi do zarządzania projektami.

Artykuł [16] przedstawia badanie i analizę doświadczeń użytkownika korzystającego z biblioteki cyfrowej. Do przeprowadzenia badania wykorzystano metodę eye-trackingu oraz dwa kwestionariusze. W badaniu wzięło udział 30 uczestników. Analizie poddano dwie biblioteki cyfrowe. Uczestnicy mieli do wykonania określone zadania, po czym za pomocą kwestionariusza ocenili swoje doświadczenia z korzystania z systemu. Kwestionariusze wskazały, że względem jednej z bibliotek uczestnicy mieli bardziej pozytywne odczucia. Metoda eye-trackingu wskazała różne wzorce spoglądania na interfejs przez użytkowników w obu bibliotekach cyfrowych, zwłaszcza w polach wyszukiwania. Analiza rezultatów obu metod pozwoliła stwierdzić, że duży wpływ na odczucia użytkownika mają usytuowanie pola wyszukiwania oraz intuicyjność strony głównej.

W artykule [17] omówiono badania dotyczące użyteczności aplikacji mobilnej przeznaczonej do sterowania dostępem do pomieszczeń. Badanie polegało na ocenie za pomocą testów realizowanych przez użytkowników najważniejszych funkcjonalności aplikacji. W trakcie badania, za pomocą eye-trackera zbierane były dane dotyczące aktywności użytkowników w aplikacji w trakcie wykonywania określonych zadań. Do oceny użyteczności wykorzystano ankiety oraz pogłębiony wywiad. Użytkownicy za pomocą pięciostopniowej skali oceniali stopień zadowolenia z użytkowania aplikacji. Badanie pozwoliło wskazać i zhierarchizować problemy dotyczące użyteczności interfejsu. Przedstawiono również wskazówki dotyczące sposobu usunięcia błędów. Pomimo małych rozmiarów aplikacji oraz przeprowadzenia badania na niewielkiej próbie badawczej, wskazana została duża liczba błędów, co dowodzi, że łączenie różnych metod służących do oceny użyteczności jest bardzo skuteczne.

Autorzy artykułu [18] zbadali jak eye-tracking i analiza ruchu gałek ocznych mogą posłużyć do oceny interfejsów użytkownika. Przeanalizowano również w jaki sposób można dokonać jej automatyzacji. Wykonany eksperyment polegał na dokonaniu przez uczestnika rezerwacji biletów lotniczych. Każdemu z uczestników zaprezentowano trzy interfejsy o tych samych funkcjonalnościach, ale innej formie wyświetlania treści. Otrzymali oni do zrealizowania w nich kilka scenariuszy użyteczności. Badanie pozwoliło na stworzenie automatycznej metody służącej ewaluacji użyteczności interfejsów w oparciu o dane eye-trackingowe. Rezultaty eksperymentu potwierdziły, że eye-tracking, umożliwiając wskazanie słabych i mocnych cech i punktów interfejsu, może być wykorzystywany do jego projektowania i tworzenia.

Artykuł [19] przedstawia badania polegające na ocenie interfejsu chatbota w systemie rezerwacji, poprzez porównanie go z tradycyjnym sposobem wprowadzania danych za pomocą formularzy. Ocena systemu polegała na zbadaniu czasu wykonywania zadań przez użytkownika przy pomocy metody KLM (Keystroke Level Model). Badania wykazały, że w systemie z interfejsem chatbota do wykonania rezerwacji pakietu wycieczkowego potrzebna była mniejsza liczba wykonywanych operacji. Ponadto wykonanie rezerwacji było szybsze o kilkanaście sekund w porównaniu z dokona-

niem rezerwacji przy pomocy tradycyjnych formularzy. Stwierdzono zatem, że nowy system rezerwacji jest efektywniejszy od starego systemu.

Autorzy artykułu [20] dokonali porównania dwóch metod eksperckich oceny użyteczności systemu – oceny heurystycznej oraz wędrówki poznawczej. W badaniu wzięło udział pięciu ekspertów, których zadaniem było poddanie ocenie systemu zarządzania gabinetem medycznym. Autorzy porównali obie metody w oparciu o liczbę wykrytych problemów oraz ich znaczenie. Oceniając atrybut „satysfakcja" użyteczności, więcej problemów zostało zidentyfikowanych przy użyciu oceny heurystycznej. W przypadku atrybutu „łatwość uczenia się" skuteczniejsza w wykrywaniu błędów była metoda wędrówki poznawczej. Skuteczność obu metod w wykrywaniu ogólnej liczby problemów dotyczących użyteczności była podobna. Otrzymane wyniki mogą świadczyć o tym, że do badania systemów skierowanych dla początkujących użytkowników, lepszą metodą może być metoda wędrówki poznawcza. W przypadku systemów, z których korzystają doświadczeni użytkownicy skuteczniejsza wydaje się być ocena heurystyczna.

Wykonany przegląd literatury wskazał, że przeprowadzano już podobne badania jakości interfejsów aplikacji, ale w żadnym z nich, w przeciwieństwie do badania opisanego w tym artykule, nie dokonano oceny interfejsów narzędzi stosowanych w e-marketingu. Podczas analizowania badań opisywanych w publikacjach zauważono, że w wielu przypadkach do oceny wykorzystywano kilka różnych metod badawczych.

2. Cel i zakres badań

Celem pracy było dokonanie analizy ergonomii interfejsów popularnych narzędzi e-marketingu. Badanie pozwoliło ocenić jakość wybranych narzędzi, zbadać ich zgodność z powszechnie znanymi zasadami i normami projektowania interfejsów oraz ocenić ich przystępność dla potencjalnych użytkowników, w szczególności tych bez doświadczenia w korzystaniu z tego typu oprogramowania. Analiza miała wskazać najlepsze narzędzie do marketingu internetowego.

Zakres: Zakres badań obejmował dokonanie analizy interfejsu trzech wybranych narzędzi e-marketingu. Każdy z nich został oceniony za pomocą trzech metod oceny interfejsów użytkownika.

Teza: Wszystkie analizowane narzędzia mają podobną jakość ergonomii interfejsu.

Pytania badawcze:

1. Czy metoda wędrówki poznawczej pozwoli wskazać najlepsze narzędzie e-marketingu?
2. Które narzędzie ma największą liczbę problemów i błędów interfejsu użytkownika?
3. Jakie narzędzie e-marketingu jest najlepszym rozwiązaniem dla użytkowników?

3. Plan badań

Plan eksperymentu obejmuje następujące etapy:

1. Przeprowadzenie badań przy wykorzystaniu metody wędrówki poznawczej.
   a) Przygotowanie scenariuszy z zadaniami do wykonania.
   b) Przygotowanie listy LUT umożliwiającej dodatkową ocenę interfejsu.
   c) Określenie grupy badawczej (wykształcenie, wiek, liczba osób).
   d) Wybór osób do grupy badawczej.
   e) Przeprowadzenie badania z udziałem użytkowników.
2. Analiza wyników badań.
   a) Analiza czasu wykonania poszczególnych zadań.
   b) Analiza trasy myszy podczas wykonywania poszczególnych zadań.
   c) Analiza liczby kliknięć myszą podczas wykonywania poszczególnych zadań.
   d) Analiza liczby wciśnięć klawiszy na klawiaturze podczas wykonywania poszczególnych zadań.
   e) Analiza błędów popełnianych podczas wykonywania poszczególnych zadań.
   f) Analiza komentarzy użytkowników.
3. Przeprowadzenie ankiety.
   a) Sformułowanie pytań opartych na heurystykach Nielsena, analizujących użyteczność narzędzi będących obiektami badania.
   b) Sformułowanie pytań umożliwiających wyrażenie opinii uczestnika badania o narzędziu.
   c) Określenie grupy badawczej (wykształcenie, wiek, liczba osób).
   d) Wybór osób do grupy badawczej.
   e) Udostępnienie ankiety uczestnikom badania.
4. Analiza danych z przeprowadzonej ankiety.
   a) Analiza ocen narzędzi dla poszczególnych heurystyk Nielsena.
   b) Analiza opinii i preferencji uczestników badania.
5. Wnioski.

4. Metody badawcze

Do przeprowadzenia analizy wybrano trzy narzędzia przeznaczone głównie do monitoringu Internetu i analizy danych. Narzędzia te to Brand24 [21], Mention [22] i Awario [23]. Aby możliwe było jak najdokładniejsze porównanie, wybrane zostały narzędzia, które posiadają podobne funkcjonalności i sposoby ich realizacji. Do badania wykorzystano wersje internetowe programów.

4.1. Badanie metodą wędrówki poznawczej

W grupie badawczej znalazło się pięć osób o wykształceniu wyższym, w przedziale wiekowym 23-26 lat. Osoby te nie korzystały wcześniej z narzędzi do e-marketingu.

Ze względu na różnice w sposobie realizacji niektórych funkcjonalności, opracowano trzy osobne scenariusze zadań, po jednym dla każdego narzędzia. Każdy z nich zawierał polecenia do realizacji tych samych funkcjonalności. Pojedynczy scenariusz zawierał sześć zadań, a każde z nich składało się dodatkowo z dwóch lub trzech podzadań. Na Rysunku 1 przedstawiono scenariusz dla jednego z narzędzi. Każdemu uczestnikowi badania przydzielono inną kolejność narzędzi w jakiej realizowano badanie.

Awario

Zadanie 1.

1. Wejdź na stronę narzędzia Awario (awario.com).
2. Zaloguj się

Zadanie 2.

1. Utwórz nowy alert typu **standard alert**, dla słowa kluczowego „Netflix", wybierz język wyszukiwania **angielski**, pozostałe ustawienia pozostaw bez zmian.
2. Po znalezieniu wzmianek odszukaj możliwość filtrowania i wybierz tylko wzmianki **z ostatnich 30 dni**, **neutralne** (o neutralnym sentymencie).
3. Usuń dwie pierwsze wzmianki.

Zadanie 3.

1. Znajdź w menu i wyświetl panel ze statystykami dla wyszukanej frazy (opcja Mention Statistics), pokaż dane tylko dla **ostatnich 7 dni** (opcja last 7 days).
2. Odszukaj i podaj **liczbę** oraz **procent** wzmianek pochodzących **ze źródła News/Blog .**

Zadanie 4.

1. Odszukaj i wyświetl ranking Infuencerów,.
2. Podaj nazwę osoby/konta/firmy na pierwszym miejscu rankingu.

Zadanie 5.

1. Dodaj nowy alert dla frazy „HBO" (standard alert, język angielski).
2. Wyszukaj opcję umożliwiającą porównanie wyszukiwań dla alertów „Netflix" i „HBO".
3. Dla której frazy wyszukano więcej zmianek („Netflix" czy „HBO")?
   a. Podaj nazwę tej frazy.
   b. Podaj tą liczbę wzmianek.

Zadanie 6.

1. Usuń oba alerty.
2. Wyloguj się.

Rysunek 1: Scenariusz zadań dla narzędzia Awario.

Podczas realizowania scenariuszy mierzony był ogólny czas wykonania zadania oraz czas realizacji poleceń do momentu zakończenia wykonywania czynności przez badanego. Oprócz tego dokonano pomiarów umożliwiających analizę drogi kursora w systemie oraz zbadanie wykorzystania klawiatury. Zmierzono przebyty kursorem dystans, liczbę kliknięć lewym przyciskiem myszy oraz liczbę wciśnięć klawiszy na klawiaturze. Do pomiarów wykorzystano program *Mousotron* [24]. Dodatkowo sprawdzono poprawność wykonania wszystkich zadań i zebrano informacje o liczbie popełnionych błędów. Uczestnicy badania zostali poproszeni o wyrażenie ewentualnych uwag i opinii o narzędziach na głos. Całe badanie było rejestrowane poprzez nagrywanie dźwięku oraz ekranu komputera.

4.2. Badanie przy wykorzystaniu listy LUT

Lista LUT wypełniana była przez osoby z grupy badawczej badania metodą wędrówki poznawczej, składającej się z pięciu osób o wykształceniu wyższym, w przedziale wiekowym 23-26 lat. Oceny dla każdego narzędzia dokonywano po wykonaniu zadań ze scenariusza wędrówki. Na każde pytanie udzielana była odpowiedź w skali od 1 do 5, gdzie 5 oznaczało najwyższą możliwą do uzyskania ocenę, świadcząca o braku problemów związanych z użytecznością narzędzia.

Po zebraniu ocen od wszystkich badanych, wyliczone zostały średnie oceny pytań i poszczególnych obszarów w danym narzędziu. Na podstawie tych średnich, dla każdego z nich wyznaczono liczbę punktów WUP (ang. *Web Usability Points*). W przypadku tej punktacji wyższa ocena oznacza wyższą jakość narzędzia.

4.3. Ankieta do oceny użyteczności interfejsu

W grupie badawczej badania ankietowego znalazły się cztery spośród pięciu osób biorących udział w badaniu metodą wędrówki poznawczej, w przedziale wiekowym 23-26 lat, posiadające wyższe wykształcenie. Badani dokonywali oceny danego narzędzia po zrealizowaniu zadań ze scenariusza wędrówki i wypełnieniu listy LUT.

Pytania ankiety oparto na 10 heurystykach Nielsena, analizujących poszczególne aspekty użyteczności oprogramowania. Dla każdej z nich zdefiniowano jedno lub dwa pytania odnoszące się do poruszanego przez nią zagadnienia.

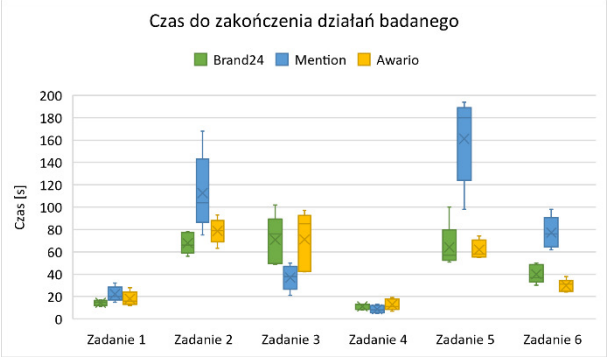
Dodatkowo do ankiety dołączone zostały cztery pytania, pozwalające na pozyskanie ogólnej opinii badanych osób o komforcie pracy z narzędziem. Łącznie ankieta składała się więc z 15 pytań.

5. Wyniki badań

Po zakończeniu badań zebrano wyniki dokonywanych pomiarów oraz ocen przyznawanych w poszczególnych metodach badawczych. Następnie dane te zostały przedstawione głównie w postaci tabel oraz wykresów pudełkowych, dzięki którym dostrzeżono różnice występujące dla poszczególnych narzędzi.

5.1. Wyniki badania metodą wędrówki poznawczej

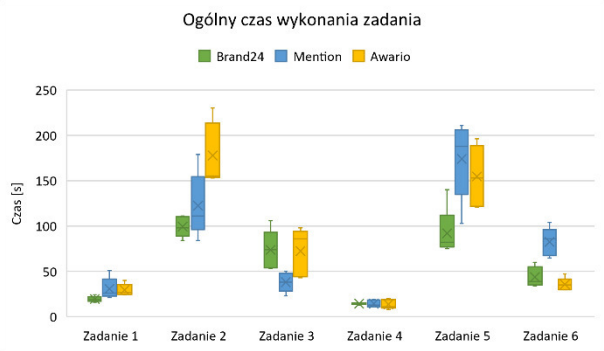
Pierwszą wartością zmierzoną podczas badania metodą wędrówki poznawczej był czas wykonywania zadania. Pomiaru dokonano na dwa sposoby. Pierwszy z nich kończył się, gdy badany wykonał wszystkie czynności niezbędne do zrealizowania polecenia (Rysunek 2).



Rysunek 2: Rozkład czasu do zakończenia działań badanego.

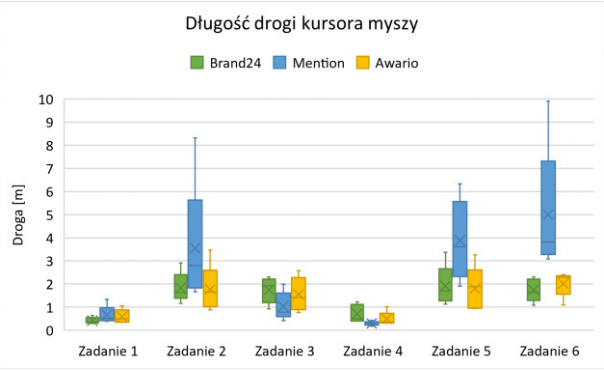
Analizując otrzymane dane można zauważyć, że pod względem czasu wykonywania działań przez użytkownika najgorzej wypadło narzędzie Mention. Fakt ten zauważalny jest w szczególności w zadaniu 2, którego częścią było utworzenie alertu umożliwiającego wyszukiwanie wzmianek. Podczas realizacji polecenia, konieczne było przejście przez kilka widoków ekranu oraz przeanalizowanie licznych opcji dostosowujących wyszukiwanie. Zadanie 5, którego częścią ponownie było zdefiniowanie alertu a także utworzenie i przeanalizowanie raportu, wymagało od badanych wykonania większej liczby kroków niż w narzędziach Brand24 i Awario. Oprócz tego badani mieli w nim problemy z odczytaniem wartości z raportu. Największą trudność w zadaniu 6 sprawiło badanym nietypowe umiejscowienie przycisku umożliwiającego wylogowanie się. Dla narzędzi Brand24 i Awario uzyskiwano niższe oraz bardziej zbliżone do siebie i skoncentrowane czasy.

Drugi pomiar czasu jest ogólnym czasem realizacji zadań, wraz z czasem potrzebnym na sfinalizowanie akcji przez system (Rysunek 3). Porównując wyniki z czasami dla pierwszego pomiaru, dostrzegalna jest różnica w otrzymanych wynikach. Widoczne jest to w zadaniu 2, gdzie najgorsze wyniki uzyskano w programie Awario. Bezpośredni wpływ miał na to bardzo długi czas (w porównaniu z dwoma pozostałymi narzędziami) wyszukiwania wzmianek. Wpłynął on również na znaczne wydłużenie czasów w zadaniu 5. Najniższe czasy oraz najbardziej skoncentrowane wyniki uzyskano w narzędziu Brand24.



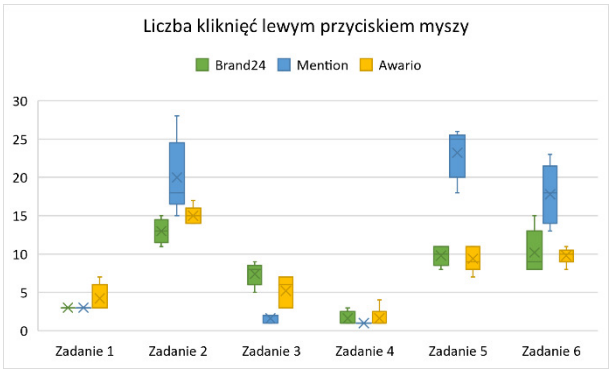
Rysunek 3: Rozkład ogólnego czasu.

Kolejną zmierzoną wartością była długość drogi przebytej kursorem myszy (Rysunek 4). Podobnie jak w przypadku pomiaru czasu, największe różnice w wynikach uzyskano dla bardziej skomplikowanych zadań (drugie, piąte i szóste). Ponownie najgorsze wyniki uzyskano w nich dla narzędzia Mention, dla którego zaobserwowano największe średnie długości drogi kursora, największe minimalne i maksymalne wartości długości oraz największy rozrzut wyników. W zadaniu pierwszym i czwartym wszystkie trzy narzędzia uzyskały zbliżone wyniki. Dla narzędzi Brand24 i Awario otrzymywano niższe, zbliżone do siebie dystanse.



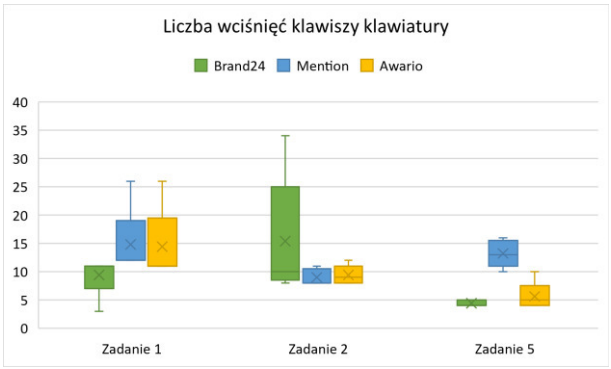
Rysunek 4: Rozkład długości drogi kursora myszy.

Poza dystansem kursora, zmierzono również liczbę kliknięć lewym przyciskiem myszy (Rysunek 5). W przypadku krótkich i prostych zadań, na których wykonanie nie składało się dużo czynności (podczas uruchamiania strony i logowania się do serwisu w zadaniu 1 oraz wyświetlania i odczytywania rankingu influencerów w zadaniu 4) we wszystkich narzędziach uzyskiwano praktycznie te same lub bardzo zbliżone liczby kliknięć przyciskiem myszy. Analizując wyniki dla wszystkich zadań widać, że ponownie najgorsze okazało się narzędzie Mention, w którym liczby kliknięć myszą w trudniejszych zadaniach znacząco odstawały od liczby kliknięć w pozostałych narzędziach oraz charakteryzowały się największym rozrzutem danych. Dla programów Brand24 i Awario odnotowywano zbliżone wartości.



Rysunek 5: Rozkład liczby kliknięć lewym przyciskiem myszy.

Ostatnią mierzoną wartością była liczba wciśnięć klawiszy na klawiaturze (Rysunek 6). Do realizacji zadania 3, 4 i 6 uczestnikom badania wystarczyła mysz komputerowa, w związku z tym nie odnotowywano w nich raczej wciśnięć klawiszy. Uruchomienie strony oraz zalogowanie się na konto użytkownika (zadanie 1) generowało zazwyczaj podobne wartości w poszczególnych narzędziach. Pojedyncze niższe wyniki oznaczały skorzystanie z podpowiedzi adresu strony, a wyższe związane były z błędnym wpisaniem adresu. Dla przeanalizowania tej charakterystyki narzędzi, najważniejsze są zadanie 2 oraz zadanie 5, gdzie uczestnicy badania używali klawiatury podczas tworzenia alertów do wyszukiwania wzmianek oraz generowania raportów. W zadaniu drugim najwięcej wciśnięć klawiszy klawiatury odnotowano w narzędziu Brand24. W Mention i Awario uzyskano niższe, zbliżone do siebie wyniki. W przypadku zadania 5 najwięcej kliknięć potrzeba było w narzędziu Mention (od 10 do nawet 16 wciśnięć klawiszy). W pozostałych narzędziach większość badanych potrzebowała 4-5 kliknięć.



Rysunek 6: Rozkład liczby wciśnięć klawiszy dla wybranych zadań.

Poza wykonaniem wyżej omówionych pomiarów, po zakończeniu badań zweryfikowano liczbę popełnionych błędów. Zostały one podzielone na dwie kategorie.

Pierwsza z nich to błędne wykonanie, oznaczające, że uczestnik nie zrealizował części polecenia lub popełnił błąd, który negatywnie wpłynął na poprawność wykonania. Drugi typ błędu to pomyłka podczas realizacji. Do tego rodzaju błędów zaliczone są akcje, które nie miały wpływu na ogólną poprawność rozwiązania, ale sprawiły, że zwiększył się czas wykonywania scenariusza, dystans kursora myszy lub liczba kliknięć myszą czy klawiaturą. W Tabeli 1 zestawiono błędy obu typów dla poszczególnych narzędzi.

Tabela 1: Zestawienie liczby błędów i pomyłek w narzędziach

Narzędzie	Liczba błędów	Liczba pomyłek
Brand24	3	9
Mention	2	19
Awario	2	12

Analizując otrzymane wyniki można zauważyć, że liczba błędów skutkujących niepoprawnym wykonaniem zadania była prawie jednakowa dla wszystkich narzędzi – w Mention i Awario popełniono 2 błędy, natomiast w Brand24 wystąpiły 3 błędy. Większość z nich dotyczyła nieodczytania wartości ze statystyk czy raportu lub ich błędnego odczytania. Błędy te prawdopodobnie wynikały z braku skupienia lub pośpiechu niż z niewłaściwie zaprojektowanego interfejsu. Pomyłki we wszystkich narzędziach dotyczyły przede wszystkim wyboru złych lub niepotrzebnych opcji czy wybierania niewłaściwych zakładek w menu. Najmniej pomyłek zaobserwowano w narzędziu Brand24 – łącznie 9. Najgorzej wypadło narzędzie Mention – odnotowano 19 pomyłek uczestników badania. Ich liczba oraz pochodzenie mogą świadczyć o problemach związanych z interfejsem narzędzia.

Uczestnicy badania zostali poproszeni o wyrażenie swojej opinii o narzędziach i ich interfejsach. Najlepsze wrażenie na uczestnikach badania zrobiło narzędzie Awario. Zauważono, że posiada ono prostszy, bardziej intuicyjny i przyjemniejszy w obsłudze interfejs. Stwierdzono również, że menu programu jest lepiej opisane niż menu w narzędziu Mention, dzięki czemu łatwiej jest się też w nim poruszać. Prostsze było również odczytywanie statystyk. Największą wadą narzędzia okazał się bardzo długi (w porównaniu z dwoma pozostałymi programami) czas wyszukiwania wzmianek. Problematyczna była również nieintuicyjna kolejność ułożenia filtrów sentymentu. Pozytywnie ocenione zostało także narzędzie Brand24. Badani uważali, że najłatwiej było w nim znaleźć niektóre funkcje i opcje a poszczególne widoki były łatwiejsze w obsłudze i nawigacji niż w pozostałych narzędziach. Najwięcej krytyki skierowano do narzędzia Mention. Badani uważali, że choć interfejs narzędzia był spójny to jednak nie był on dla nich w pełni zrozumiały. Dużo trudności sprawiła obsługa i nawigacja w menu. Dodatkowo uczestnikom badania nie spodobało się również rozmieszczenie opcji umożliwiającej wylogowanie się z konta użytkownika.

5.2. Wyniki oceny przy pomocy listy LUT

Po uzyskaniu ocen dla każdego analizowanego obszaru wyliczona została jego średnia ocena (Tabela 2). Dla nawigacji i struktury oraz obszaru dotyczącego treści podstron najlepsze oceny uzyskały Brand24 oraz Awario. Pod względem zastosowania odpowiednich komunikatów, informacji zwrotnych i pomocy dla użytkownika najlepszy okazał się program Brand24. Ocena obszaru dotyczącego interfejsu aplikacji wskazała, że dla uczestników badania najlepszy layout oraz dobór barw posiada narzędzie Awario, które uzyskało maksymalną ocenę. Dla wprowadzania danych najlepsze, zbliżone do siebie wyniki uzyskano dla narzędzi Brand24 oraz Awario.

We wszystkich pięciu obszarach, najsłabiej oceniono narzędzie Mention. Najniższą ocenę spośród wszystkich obszarów uzyskało ono dla nawigacji i struktury (ocena 4,37). Związane jest to z problemami jakie mieli niektórzy uczestnicy badania z odnalezieniem się i wyszukaniem określonych opcji w menu narzędzia.

Wszystkie uzyskane oceny przekraczały wartość 4,00. Oznacza to, że dostrzeżono pojedyncze drobne problemy odnoszące się do użyteczności aplikacji, mogące wpłynąć nieznacznie na komfort pracy z danym narzędziem. Nie stwierdzono jednak wystąpienia poważnych problemów uniemożliwiających wykonanie określonych funkcjonalności lub mogących całkowicie uniemożliwić korzystanie z narzędzia.

Tabela 2: Uśrednione oceny obszarów dla narzędzi

Obszar	Brand24	Mention	Awario
Nawigacja i struktura	4,86	4,37	4,86
Komunikaty, *feedback*, pomoc dla użytkownika	4,75	4,45	4,60
Interfejs aplikacji	4,90	4,80	5,00
Treść podstron	4,92	4,68	4,92
Wprowadzanie danych	4,96	4,80	4,92

Uzyskane średnie oceny obszarów posłużyły do wyliczenia punktów WUP dla każdego z badanych narzędzi (Tabela 3).

Tabela 3: Punkty WUP wyliczone dla narzędzi

Narzędzie	Punkty WUP
Brand24	4,88
Mention	4,62
Awario	4,86

Najwyższą liczbę punktów WUP uzyskało narzędzie Brand24. Drugi wynik, z bardzo zbliżoną liczbą otrzymanych punktów uzyskało narzędzie Awario. Na ostatnim miejscu uplasowało się narzędzie Mention. Otrzymane wyniki pokrywają się z pomiarami wartości wykonywanymi podczas badania metodą wędrówki poznawczej oraz z uwagami uczestników badania.

5.3. Wyniki ankiety

Na podstawie ocen przydzielonych przez osoby biorące udział w badaniu, wyliczone zostały średnie oceny dla każdego z pytań odnoszącego się do danej heurystyki Nielsena. Następnie na podstawie otrzymanych wartości wyliczono średnie oceny dla narzędzi (Tabela 4). Podczas oceny tą metodą ponownie najlepsze wyniki uzyskano dla narzędzia Brand24 i Awario, które otrzymało nieco wyższą średnią ocenę i uzyskało maksymalny rezultat w pięciu pytaniach. Dotyczyły one widoczności statusu systemu, dopasowania elementów do świata rzeczywistego, kontroli i wolności jaką ma użytkownik, konsekwencji i dostosowania się do standardów oraz elastyczności i efektywności używania. Najniższą ocenę o wartości 4,25 uzyskało narzędzie Mention, które dla większości pytań otrzymało najniższe średnie oceny.

Tabela 4: Średnie oceny w ankiecie oceny użyteczności

Heurystyka	Średnia ocena		
	Brand24	**Mention**	**Awario**
Widoczność statusu systemu	5,00	4,25	5,00
Dopasowanie systemu do świata rzeczywistego	4,75	4,00	4,75
	4,75	4,50	5,00
Kontrola i wolność użytkownika	5,00	4,50	5,00
Konsekwencja i standardy	4,75	4,00	5,00
Zapobieganie błędom	4,50	4,50	4,75
Rozpoznawanie a nie przypominanie	5,00	4,25	4,75
Elastyczność i efektywność użytkowania	4,75	4,50	5,00
Estetyczny i minimalistyczny design	4,75	4,00	4,75
Pomaganie użytkownikom w rozpoznawaniu, diagnozowaniu oraz usuwaniu błędów	4,25	4,25	4,25
Pomoc i dokumentacja	4,50	4,00	4,50
Średnia ocena	**4,73**	**4,25**	**4,80**

Oprócz ocen dla pytań opartych na heurystykach, wyliczono również średnie oceny w pytaniach dodatkowych, pozwalających wyrazić ogólną opinię o narzędziu (Tabela 5). Uczestnicy badania najlepiej ocenili program Brand24. Według nich zapewniał on najwyższy komfort pracy i posiadał najbardziej czytelny interfejs, co wpłynęło na najwyższy poziom zadowolenia z narzędzia. Drugą najlepszą ocenę uzyskało narzędzie Awario, którego nawigacja została oceniona na równi z tą znajdującą się w programie Brand24. Oceny przyznane narzędziu Mention potwierdziły wcześniejsze uwagi i problemy związane z nawigacją narzędzia oraz jego czytelnością. Program uzyskał najniższą średnią ocenę.

Tabela 5: Średnie oceny w dodatkowych pytaniach ankiety

Pytanie	Brand24	Mention	Awario
Jak ocenia Pan/Pani nawigację narzędzia?	4,75	4,00	4,75
Jak ocenia Pan/Pani komfort pracy z narzędziem?	4,75	3,75	4,50
Jak ocenia Pan/Pani czytelność interfejsu narzędzia?	5,00	4,00	4,75
Proszę ocenić ogólny poziom zadowolenia z narzędzia	4,75	4,00	4,50
Średnia ocena	**4,81**	**3,94**	**4,63**

6. Wnioski

Przeprowadzony eksperyment oraz postawione wcześniej pytania badawcze pozwoliły zweryfikować prawdziwość postawionej tezy, w której założono, że wszystkie analizowane narzędzia mają podobną jakość ergonomii interfejsu.

W pierwszym pytaniu badawczym zastanawiano się, czy metoda wędrówki poznawczej pozwoli wskazać najlepsze narzędzie e-marketingu. Na podstawie wyników badań tą metodą nie można jednak wskazać jednego najlepszego narzędzia, ponieważ programy Brand24 i Awario uzyskiwały podobne rezultaty. Dla badanych wartości takich jak czas wykonania zadań, długość dystansu kursora, liczba kliknięć lewym przyciskiem myszy, wciśnięć przycisków klawiatury czy liczba błędów, narzędzia naprzemiennie okazywały się od siebie niewiele lepsze ale uzyskiwane wyniki były do siebie zbliżone. Podczas dodatkowej oceny za pomocą listy LUT również uzyskiwano podobne oceny. Analogiczna sytuacja dotyczyła punktów WUP, gdzie narzędzie Brand24 uzyskało ocenę 4,88 a narzędzie Awario osiągnęło wynik 4,86. Wędrówka poznawcza pomogła jednak wskazać najsłabsze narzędzie spośród analizowanych, którym okazało się być Mention. W większości pomiarów uzyskiwało ono najsłabsze wyniki, cechowało się też największą liczbą zaobserwowanych pomyłek. Dodatkowo uzyskiwało ono również najgorsze oceny dla listy LUT i najmniejszą liczbę punktów WUP (wynik 4,62).

Drugie postawione pytanie miało wykazać, które narzędzie ma największą liczbę problemów i błędów interfejsu użytkownika. Na podstawie uzyskanych wyników można stwierdzić, że największą liczbę problemów zaobserwowano w narzędziu Mention. Odnotowano w nim aż 19 pomyłek uczestników badania, które pomimo, że nie miały bezpośredniego wpływu na poprawność rozwiązywania zadań to znacząco wpływały na inne mierzone wartości, takie jak czas realizacji poleceń, droga kursora czy liczba kliknięć myszą. Najwięcej problemów sprawiała badanym nawigacja w menu oraz wylogowanie się z konta użytkownika. Pod względem poważnych błędów, wpływających na poprawność rozwiązania we wszystkich narzędziach zaobserwowano ich zbliżoną liczbę (2 lub 3).

W trzecim pytaniu badawczym próbowano określić jakie narzędzie e-marketingu jest najlepszym rozwiązaniem dla użytkowników. Na podstawie uzyskanych wyników, można stwierdzić, że w analizowanej kategorii narzędzi do monitoringu Internetu najlepsze okazały się narzędzia Brand24 i Awario. Uzyskiwały one najlepsze wyniki w pomiarach realizowanych podczas metody wędrówki poznawczej, otrzymywały najwyższe oceny w liście LUT i najwyższą liczbę punktów WUP. Dodatkowo programy te zostały najlepiej ocenione pod względem użyteczności, badanej w oparciu o heurystyki Nielsena. Uczestnicy badania pozytywnie komentowali te narzędzia i nie zgłaszali poważnych problemów podczas ich wykorzystania. Brand24 i Awario uzyskały również wysokie oceny w pytaniu o ogólny poziom zadowolenia z narzędzia.

Na podstawie otrzymanych wyników oraz odpowiedzi uzyskanych na pytania badawcze, możliwe jest obalenie postawionej tezy. Nie wszystkie badane narzędzia mają podobną jakość ergonomii interfejsów. Programy Brand24 i Awario uzyskiwały lepsze wyniki niż narzędzie Mention. Zostały one również lepiej ocenione przez uczestników badania.

Literatura

[1] C. Giunta, Digital marketing platform tools, generation Z, and cultural considerations, Journal of Marketing Development and Competitiveness 14(2) (2020) 63-75.

[2] F. K. Mazumder, U. K. Das, Usability guidelines for usable user interface, International Journal of Research in Engineering and Technology 3(9) (2014) 79-82.

[3] K. S. Park, C. H. Lim, A structured methodology for comparative evaluation of user interface designs using usability criteria and measures, International Journal of Industrial Ergonomics 23(5-6) (1999) 379-389.

[4] Ł. Kozioł, M. Żabińska, J. Majewski, Dobre praktyki projektowania interfejsu użytkownika, Zeszyt Naukowy Wyższej Szkoły Zarządzania i Bankowości w Krakowie 57 (2020) 1-14.

[5] H. Taiminen, One gets what one orders: Utilisation of digital marketing tools, The Marketing Review 16(4) (2016) 389-404.

[6] P. Ganev, How to Choose the Appropriate Digital Marketing Tool, Global Business & Economics Anthology 1(3) (2018) 16-25.

[7] N. A. Zaini, S. F. M. Noor, T. S. M. T. Wook, Evaluation of api interface design by applying cognitive walkthrough, International Journal of Advanced Computer Science and Applications 10(2) (2019) 306-315.

[8] M. Jusiak, M. Miłosz, Analiza jakości interfejsu aplikacji internetowej z wykorzystaniem eye-trackingu – studium przypadku, Journal of Computer Sciences Institute, 10 (2019) 62-66.

[9] A. Miklosik, M. Kuchta, N. Evans, S. Zak, Towards the adoption of machine learning-based analytical tools in digital marketing, IEEE 7 (2019) 85705-85718.

[10] M. Borys, M. Plechawska-Wójcik, Badanie użyteczności oraz dostępności interfejsu w aplikacjach mobilnych, Nierówności Społeczne a Wzrost Gospodarczy 35 (2013) 63-78.

[11] M. Kurzyna, D. Matysiak, Analiza porównawcza jakości interfejsów mobilnego dostępu do usług wybranych banków, Praca magisterska, Politechnika Lubelska, Lublin, 2017.

[12] M. Cioczek, T. Czarnota, Analiza współczesnych interfejsów człowiek-komputer, Praca magisterska, Politechnika Lubelska, Lublin, 2021.

[13] M. Farzandipour, E. Nabovati, H. Tadayon, M. S. Jabali, Usability evaluation of a nursing information system by applying cognitive walkthrough method, International Journal of Medical Informatics 152 (2021) 1-7, https://doi.org/10.1016/j.ijmedinf.2021.104459.

[14] M. Miłosz, M. Plechawska-Wójcik, M. Borys, M. Laskowski, Quality improvement of ERP system GUI using expert method: A case study, 2013 6th International Conference on Human System Interactions (HSI), IEEE (2013) 145-152.

[15] P. Pawłowski, Analiza porównawcza narzędzi dedykowanych zarządzaniu projektami, Praca magisterska, Politechnika Lubelska, Lublin, 2022.

[16] M. Kuhar, T. Merčun, Exploring user experience in digital libraries through questionnaire and eye-tracking data, Library & Information Science Research 44(3) (2022) 1-11, https://doi.org/10.1016/j.lisr.2022.101175.

[17] M. Borys, M. Miłosz, Mobile application usability testing in quasi-real conditions-the synergy of using different methods, 2018 11th International Conference on Human System Interaction (HSI), IEEE (2018) 362-368.

[18] K. Harezlak, J. Rzeszutek, P. Kasprowski, The eye tracking methods in user interfaces assessment, International Conference on Intelligent Decision Technologies, Springer (2015) 325-335.

[19] A. Prahara, T. S. Saputro, Keystroke-level model to evaluate chatbot interface for reservation system, 2019 6th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), IEEE (2019) 241-246.

[20] R. Khajouei, M. Zahiri Esfahani, Y. Jahani, Comparison of heuristic and cognitive walkthrough usability evaluation methods for evaluating health information systems, Journal of the American Medical Informatics Association 24 (2017) 55-60.

[21] Brand24 – narzędzie do monitoringu Internetu i analizy danych, https://brand24.pl/, [20.03.2023].

[22] Mention – narzędzie do monitoringu Internetu i analizy danych, https://mention.com/en/, [20.03.2023].

[23] Awario – narzędzie do monitoringu Internetu i analizy danych, https://awario.com/, [20.03.2023].

[24] Mousotron – narzędzie do kontroli myszy i klawiatury, https://www.blacksunsoftware.com/mousotron.html, [21.03.2023].

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 333–340
Received: 22 June 2023
Accepted: 4 July 2023

Research on user experience during interactions with mobile applications for diabetics

Badania dotyczące doświadczeń użytkowników podczas interakcji z aplikacjami mobilnymi dla diabetyków

Przemysław Bajda* Rafał Baliński*, Mariusz Dzieńkowski

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The purpose of this article was to assess in terms of performance and user experience three mobile applications that support the daily functioning of people with diabetes. Two data collection methods were used in the study: eye tracking and the System Usability Scale questionnaire. Eye tracking provided information on eye movement and fixation while using the application, whereas the SUS questionnaire helped evaluate the overall usability level. Three applications were examined, *mySugr*, *Diabetes diary control* from Google Play, and one proprietary application implemented for the purposes of the study. It was found that the proprietary *SugarCare* application characterized by a simple, modern, and readable interface achieved the highest results in terms of task completion times and SUS questionnaire scores. The *mySugr* application, which also has a clear and simple interface, took the second place. The *Diabetes diary control* application performed the worst, as it had a complex and less user-friendly interface.

Keywords: user experience; mobile application for diabetics; eye tracking; SUS

Streszczenie
Celem artykułu jest ocena pod względem wydajności i doświadczeń użytkownika trzech aplikacji mobilnych, które wspomagają codzienne funkcjonowanie osób chorych na cukrzycę. W badaniu wykorzystano dwie metody zbierania danych: eyetracking oraz ankietę SUS (System Usability Scale). Eyetracking dostarczył informacji dotyczących ruchu oczu i fiksacji podczas korzystania z aplikacji, natomiast ankieta SUS pomogła ocenić ogólny poziom użyteczności. Przetestowano trzy aplikacje: *mySugr* i *Diabetes diary control* pochodzące ze sklepu Google Play oraz jedną aplikację zaimplementowaną na potrzeby badań. Stwierdzono, że autorska aplikacja *SugarCare*, charakteryzująca się prostym, nowoczesnym i czytelnym interfejsem, osiągnęła najlepsze wyniki pod względem czasów wykonywania zadań oraz wskaźnika SUS. Aplikacja *mySugr*, która również miała przejrzysty i prosty interfejs zajęła drugie miejsce. Najgorzej wypadło oprogramowanie *Diabetes diary control*, posiadające złożony i mało czytelny interfejs użytkownika.

Słowa kluczowe: doświadczenia użytkownika; aplikacje mobilne dla cukrzyków; eyetracking; SUS

*Corresponding authors
Email address: **przemyslaw.bajda@pollub.edu.pl** (P. Bajda), **rafal.balinski@pollub.edu.pl** (R. Baliński)

1. Wstęp

We współczesnym świecie cukrzyca określana jest mianem choroby cywilizacyjnej, na którą zapada coraz więcej osób, szczególnie tych w podeszłym wieku. W roku 2013 liczba chorych wynosiła 381,8 miliona, a według prognoz spodziewa się, że do roku 2035 ta liczba wzrośnie o prawie 210 milionów [1]. Oznacza to prawie dwukrotny wzrost zachorowań na przestrzeni kilkunastu lat.

Coraz częściej wskazuje się na rolę technologii, a szczególnie technologii mobilnych w procesie wspomagania leczenia i dbania o zdrowie pacjentów. Mimo tego, że coraz więcej ludzi korzysta z urządzeń mobilnych, wciąż niewielu chorych na cukrzycę korzysta z oprogramowania dla nich dedykowanych, ułatwiających im codzienne funkcjonowanie. Zmiana tego stanu rzeczy nastąpi, gdy większa uwaga będzie poświęcona intuicyjności struktury interfejsów poprzez ich orientowanie się na użytkownika, jego możliwości i ograniczenia, a także potrzeby.

Oprogramowanie wytwarzane na urządzenia mobilne powinno uwzględniać ograniczenia, jakie narzuca im środowisko, między innymi małe wymiary wyświetlacza oraz niskie zasoby wydajnościowe. Ponadto aplikacje przeznaczone dla diabetyków powinny być wysoce niezawodne i bezpieczne, aby nie narazić użytkowników na powikłania zdrowotne związane z ich użytkowaniem.

W raporcie research2guidance [2] obejmującym okres do końca 2018 roku jedynie 7,8% osób chorych na cukrzycę korzystało z aplikacji mobilnej przeznaczonej dla diabetyków. Jednym z głównych problemów, który może wpływać na takich stan rzeczy jest skomplikowany i nieprzejrzysty interfejs. Ten problem jest szczególnie istotny dla osób starszych w niskim stopniu obeznanych z najnowszymi technologiami. Warto nadmienić, że prawdopodobieństwo zachorowania na cukrzycę wśród seniorów jest 10 krotnie większe aniżeli wśród młodzieży.

W celu przyszłej poprawy współczynnika chorych korzystających z oprogramowania dla diabetyków,

należałoby przeprowadzić testy interfejsów i zbadać doświadczenia użytkowników korzystających z tego typu aplikacji. Do realizacji tego planu można wykorzystać metody z udziałem lub bez udziału użytkowników. Zbieranie opinii najczęściej odbywa się za pomocą ankiety. Ostatnio coraz częściej do badań interfejsów wykorzystuje się metodę, która angażuje technikę eye-trackingową. Metoda ta polega na analizie działań użytkownika w trakcie pracy z oprogramowaniem poprzez śledzenie kolejnych punktów skupienia uwagi na wyświetlanym materiale. W ten sposób powstają charakterystyczne ścieżki składające się z punktów zatrzymania w postaci kół (fiksacji) oraz linii łączących kolejne fiksacje obrazujące ruch (sakady).

Niniejsze badanie dotyczy analizy interfejsów aplikacji mobilnych wspomagających leczenie cukrzycy. W ramach pracy przygotowano i przeprowadzono eksperyment składający się z dwóch części: kwestionariuszowej oraz z użyciem eyetrackera. Celem tych badań była ocena wpływu użyteczności interfejsów aplikacji mobilnych dla cukrzyków na wydajność oraz doświadczenie użytkowników, na które składają się użyteczność i zapamiętywalność. W efekcie tych badań wskazano problemy dotyczące użyteczności powstające na etapie projektowania oraz przedstawiono dobre praktyki w opracowywaniu intuicyjnych, estetycznych, prostych w użyciu i efektywnych interfejsów użytkownika.

2. Przegląd literatury

Ciągły wzrost zachorowań na cukrzycę oraz powiększający się rynek urządzeń mobilnych sprzyja powstawaniu aplikacji przeznaczonych dla diabetyków [1], które w swoich założeniach mają wspomagać codzienne funkcjonowanie chorych poprzez dokładniejsze kontrolowanie właściwego poziomu cukru, a co z tym się wiąże utrzymywaniu prawidłowej kondycji zdrowotnej [2]. W artykule *Mobile Applications for Control and Self Management of Diabetes: A Systematic Review* [3] opisano badania mające na celu analizę 65 darmowych aplikacji mobilnych dla diabetyków dostępnych na systemy Android, iOS oraz Windows Phone. Wyniki wykazały, że jedynie 9 aplikacji spełnia określone kryteria takie jak: monitorowanie poziomu glukozy we krwi, monitorowanie dawkowania insuliny oraz innych leków, a także możliwość śledzenia aktywności fizycznej, wagi ciała oraz odżywiania, co w opinii autorów jest niezbędne do pełnej funkcjonalności programu w profilaktyce związanej z cukrzycą. Kolejnym krokiem była szczegółowa analiza aplikacji w oparciu o wymienione wyżej funkcje. W wyniku badania stwierdzono, że w celu poprawienia doświadczenia użytkowników należałoby rozważyć dodanie funkcjonalności takich jak: przypomnienia i alerty, poradniki instruujące jak korzystać z aplikacji, połączenia z social-mediami czy rozpoznawanie głosu do wprowadzania danych.

Do podobnych wniosków doszli autorzy artykułu [4], twierdząc, że przy obecnym szybkim rozwoju i zwiększonym wykorzystaniu nowoczesnych technologii, w tym aplikacji mobilnych dla diabetyków, istniejące rozwiązania nie zapewniają kompleksowej obsługi i pomocy cukrzykom. Autorzy swoje badania rozpoczęli od wyróżnienia czterech kluczowych elementów, w których program wspomógłby chorego: monitorowanie poziomu glukozy we krwi, skuteczne leczenie, właściwe nawyki żywieniowe oraz aktywność fizyczna. Następnie opracowane zostały 3 prototypy nowych aplikacji mobilnych, które zostały ocenione poprzez ankietę na grupie 30 uczestników. Respondenci jednogłośnie stwierdzili, że wyżej wymienione kluczowe funkcje są niezbędne, jednakże niewystarczające dla aplikacji wspomagających diabetyków w codziennym funkcjonowaniu. Wszyscy badani twierdzili, że niezbędna byłaby także funkcjonalność synchronizacji programu z jednym z zewnętrznych urządzeń do pomiaru poziomu glukozy we krwi

W pracy [5] autorzy dokonali systematycznego przeglądu badań nad wpływem aplikacji mobilnych na życie cukrzyków. Najpierw przeanalizowali 804 artykuły, a następnie wybrali 17 i ocenili ich jakość metodologiczną. Spośród wyselekcjonowanych w przypadku siedmiu poziom metodologiczny został określony jako wysoki lub umiarkowany. Zidentyfikowano 23 różne aplikacje mobilne, które były związane z odżywianiem i aktywnością fizyczną. Wyniki przeglądu ujawniły, że aplikacje mobilne dla diabetyków poprawiają w krótkim okresie kontrolę glikemii, ale w długim horyzoncie wyniki nie były jednoznaczne.

W artykule [6] dokonano przeglądu artykułów w okresie od 2011 do 2017 roku związanych z aplikacjami mobilnymi dla diabetyków. Na podstawie analizy 20 publikacji opisujących użyteczność i zadowolenie z użytkowania, stwierdzono, że wiele programów jest nieczytelnych dla odbiorców, co skutkuje utrudnionym dostępem do danych na temat stanu zdrowia. Zauważono, że problemy z użytecznością wynikają z błędów w interfejsach użytkownika, co jest rezultatem braku zgodności z heurystykami Nielsena. Wieloetapowe zadania, ograniczone funkcjonalności oraz trudności związane z interakcją z systemem nawigacyjnym są głównymi problemami, które negatywnie rzutują na poziom użyteczności aplikacji. Oprócz błędów projektowych na postrzeganie aplikacji ma wpływ wiek użytkowników. Stwierdzono, że osoby poniżej 55 roku życia potrzebowały mniej czasu by w sposób efektywny korzystać z aplikacji. Zaobserwowano również, że na poprawę kontroli pacjenta nad cukrzycą mają wpływ funkcjonalności, w jakie wyposażona jest aplikacja. Dodanie powiadomień czy interaktywnych wiadomości zwrotnych zwiększa zaangażowanie użytkowników.

Metoda eyetrackingu staje się coraz popularniejsza w obszarze User Experience (UX), w związku z tym, że daje obiektywne wyniki, niezależne od opinii użytkowników. Na ich podstawie można określić miejsca i elementy interfejsów, które przyciągają uwagę uczestników badań. W pracy pt. *A study of Eye Tracking Technology and its applications* [7] autorzy udowadniają, że

technika eyetrackingowa jest bardzo pomocna w testowaniu interfejsów stron internetowych, a także aplikacji mobilnych. Pozwala ona zdobyć informacje na temat kierunku ruchu gałek ocznych, miejsc skupienia wzroku, miejsc ignorowanych przez wzrok oraz reakcji oczu na różne bodźce.

W artykule *Influence of Design Elements in Mobile Applications on User Experience of Elderly People* [8] skupiono się na badaniu doświadczeń użytkowników z grupy wiekowej 50+ podczas korzystania z aplikacji dla diabetyków Glucosia w wersji oryginalnej i prototypowej. Badanie przeprowadzono na grupie 6 osób. Każdy z badanych miał wykonać 10 zadań oraz odpowiedzieć na zestaw pytań dotyczących doświadczeń użytkownika: po każdym z zadań oraz na końcu badania. Na podstawie odpowiedzi z ankiet porównano atrybuty UX takie jak: efektywność, użyteczność, łatwość użycia i dostępność. Autorzy wykazali, że dla wersji prototypowej aplikacji, w której zmieniono rozmiar czcionki oraz uproszczono interfejs wszystkie oceniane wskaźniki UX były wyższe niż dla aplikacji oryginalnej. Oznacza to, że odpowiednie zaprojektowanie interfejsu użytkownika może poprawić doświadczenia użytkownika, a także zmniejszyć niechęć do korzystania z tego typu aplikacji mobilnych wśród osób starszych.

W artykule *User Experience Design Based on Eye-Tracking Technology: A Case Study on Smartphone APPs* [9] autorzy oceniali doświadczenie użytkownika podczas interakcji z aplikacjami mobilnymi za pomocą dwóch metod: kwestionariuszowej i eyetrackingowej. Przeprowadzono badanie ośmiu mobilnych komunikatorów tekstowych. Uczestnicy badań mieli do wykonania dwa zadania na każdej z aplikacji, w losowej kolejności. Po realizacji części eyetrackingowej eksperymentu, badani wypełniali ankietę. Autorzy stwierdzili, że stosując techniki mieszane, tzn. obiektywny pomiar aktywności ocznej za pomocą eyetrackera i subiektywne zbieranie opinii przy pomocy ankiety w rezultacie zmniejsza się subiektywność oceny doświadczenia użytkownika.

3. Cel i zakres pracy

Celem artykułu jest dokonanie oceny pod względem doświadczeń użytkownika trzech aplikacji mobilnych, które wspomagają codzienne funkcjonowanie osobom chorym na cukrzycę. Dwie aplikacje zostały pobrane z platformy Google Play, natomiast trzecia została utworzona na potrzeby badań. Za główny cel badania obrano analizę wydajności i doświadczeń użytkowników podczas interakcji z aplikacjami, wykorzystując do tego celu technikę eyetrackingową oraz ankietę SUS.

W ramach pracy sformułowana została hipoteza, mówiąca o tym, że „prosty i łatwy w obsłudze, użyteczny interfejs aplikacji dla diabetyków przyspiesza proces wykonywania zadań oraz wpływa pozytywnie na doświadczenia użytkownika".

4. Metoda badawcza

Do przeprowadzenia badania zdecydowano się wykorzystać technikę eyetrackingową oraz kwestionariusz SUS, złożony z 10 pytań/stwierdzeń. Procedura badawcza składała się z pięciu etapów:

1. Przygotowanie badań: wybór aplikacji do badań, implementacja aplikacji prototypowej, opracowanie scenariuszy badawczych, dobór i zorganizowanie grupy badawczej, przygotowanie stanowiska eyetrackingowego, badanie pilotażowe.
2. Dwuetapowy eksperyment składający się z badania eyetrackingowego oraz badania kwestionariuszowego SUS.
3. Weryfikacja poprawności wykonania zadań, identyfikacja i zliczenie błędów oraz pomiar czasów realizacji zadań, opracowanie miar eyetrackingowych i wskaźników SUS.
4. Analiza ilościowa i jakościowa uzyskanych wyników oraz sformułowanie wniosków.

4.1. Obiekty badań

Przed wyborem aplikacji dla diabetyków ustalono kilka kryteriów. Pierwszym z nich była dostępność dla użytkowników smartfonów z systemem Android, kolejnym darmowy i łatwy dostęp do aplikacji. Podczas selekcji ważne było także, aby system posiadał możliwość monitorowania poziomu glukozy we krwi, zapisywania wyników pomiarów, możliwość zmiany jednostki w jakiej prowadzone są pomiary oraz generowania wykresów, raportów i kopii zapasowych. Kolejnymi ważnymi kryteriami były duża popularność aplikacji oraz dobre opinie wyrażane w sklepie Google Play. Do badań ostatecznie wybrano dwie aplikacje dla diabetyków: *mySugr* oraz *Diabetes diary control*, które cieszyły się dużą popularnością i pozytywnymi opiniami użytkowników [10, 11].

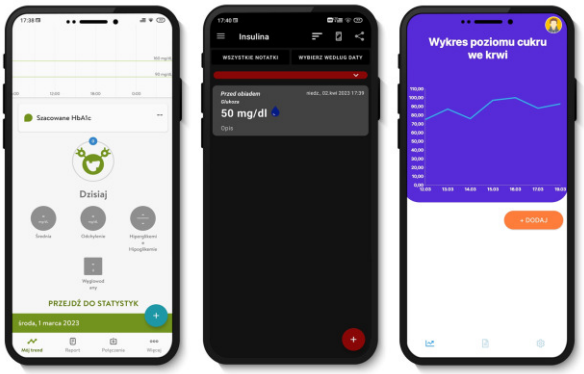
Do celów badawczych utworzono własną aplikację dla diabetyków [12], która miała spełniać wszystkie wymagania i kryteria ustalone przed procesem selekcji. Prototypowa aplikacja miała zapewniać użytkownikom łatwy i intuicyjny sposób monitorowania poziomu glukozy we krwi oraz możliwość dostępu do wykresów i raportów w czasie rzeczywistym. Aplikacja została stworzona z myślą o osobach starszych, zapewniając prosty i czytelny interfejs o jasnej kolorystyce, z dużymi i czytelnymi ikonami oraz umożliwiając łatwość personalizacji. Dzięki takiemu podejściu użytkownicy mogliby skutecznie kontrolować swoją cukrzycę i w pełni korzystać z zalet technologii w codziennym życiu.

Aplikacja *mySugr* to narzędzie, które umożliwia osobom z cukrzycą monitorowanie swojego stanu zdrowia i kontrolowanie poziomu cukru we krwi. System daje możliwość zapisywania niezbędnych informacji, takich jak spożyte posiłki, ilość węglowodanów, przyjmowane leki oraz poziom cukru we krwi. Aplikacja oferuje prosty w obsłudze, spersonalizowany interfejs, czytelne wykresy poziomu cukru we krwi, a także raporty dzienne, tygodniowe i miesięczne, które można udostępniać bezpośrednio lekarzowi. Interfejs jest jednak znacznie bardziej rozbudowany, posiada więcej przycisków przenoszących użytkownika do innych widoków, a także widok ustawień, w którym podkategorie są pogrupowane w kartach (ang. tabs). System

mySugr jest kompatybilny z różnymi urządzeniami, takimi jak glukometr, krokomierz, ciśnieniomierz, waga i inne.

Diabetes diary control jest także aplikacją wspomagającą diabetyków w ich codziennym funkcjonowaniu. System umożliwia rejestrowanie poziomu cukru we krwi, dodawanie komentarzy, a także rodzaju zjedzonego posiłku. Oprogramowanie pozwala na wydruk oraz eksport raportów, diagramów i kopii zapasowych w formacie csv oraz xlsx. Interfejs charakteryzuje się czarną kolorystyką, czerwonymi elementami i białym kolorem czcionki. W aplikacji rolę menu pełni wysuwany pasek boczny zawierający dużą liczbę przycisków. Każdy z nich posiada ikonę, jednak nie zawsze jest ona jednoznaczna, co czasami powoduje dezorientację wśród użytkowników. Przykładem takiego stanu rzeczy może być przycisk do robienia zrzutu ekranu, który reprezentuje ikona smartfona bez podpisu sugerującego jego działanie.

Głównymi motywami wyboru wyżej opisanych aplikacji, których widoki przedstawiono na Rysunku 1, były bardzo dobre oceny w serwisie Sklep Play (minimum 4 punkty w 5 punktowej skali) oraz ich wysoka popularność (minimum 100 tys. pobrań). Aplikacje pod względem posiadanych funkcjonalności są do siebie zbliżone, wszystkie mają możliwość prowadzenia statystyki poziomu glukozy, konfiguracji oraz generowania raportów i kopii zapasowych.



Rysunek 1: Aplikacje wybrane do badań: mySugr, Diabetes diary control oraz SugarCare.

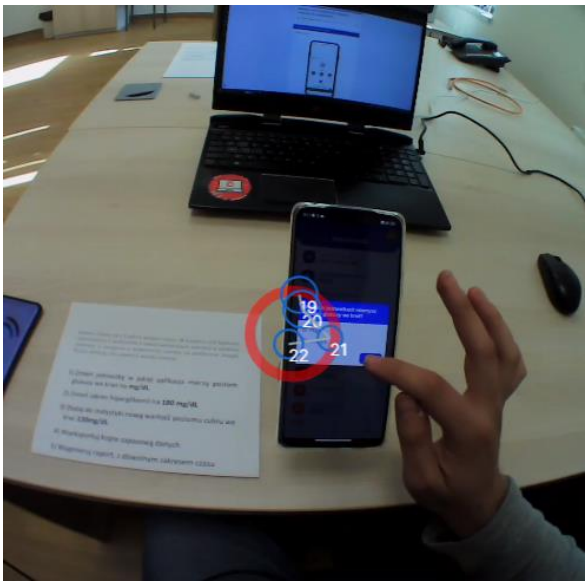
4.2. Grupa badawcza

W badaniu wzięło udział 17 osób, w tym 2 kobiety i 15 mężczyzn w przedziale wiekowym 20-24 lat. Grupa w całości składała się ze studentów Wydziału Elektrotechniki i Informatyki Politechniki Lubelskiej studiujących na kierunku Informatyka. Wszyscy badani pewnie posługiwali się smartfonami, natomiast żaden z nich nie jest diabetykiem i wcześniej nie korzystał z tego typu aplikacji.

4.3. Stanowisko badawcze

Eksperyment został przeprowadzony w laboratorium należącym do Katedry Informatyki i znajdującym się w Centrum Innowacji i Zaawansowanych Technologii Politechniki Lubelskiej. Podczas badania zapewniono badanym optymalne warunki oświetleniowe, możliwość regulacji wysokości siedzenia oraz odpowiednią odległość od ekranu smartfona. Urządzenie mobilne było zamocowane w specjalnym, stabilnym uchwycie samochodowym, przytwierdzonym do blatu biurka, aby uniemożliwić jego niepożądane ruchy. Za pomocą uchwytu można było także ustawić właściwą pozycję smartfona, tak, aby był on skierowany prostopadle do kierunku patrzenia osoby badanej. Badania były prowadzone pod kierunkiem moderatora.
Urządzeniem, na którym zostały zainstalowane wszystkie aplikacje oraz na którym badani wykonywali zadania był smartfon Xiaomi Redmi Note 10 Pro z 6,67 calowym wyświetlaczem Super Amoled o rozdzielczości 1080x2400 px, procesorem Snapdragon 732G, zawierającym 6 GB pamięci RAM.

Do śledzenia aktywności wzrokowej badanych osób (Rysunek 2), wykorzystano eyetracker mobilny Pupil Invisible, którego parametry zostały przedstawione w Tabeli 1 [13]. Eyetracker ten ma postać okularów z wbudowaną kamerą wideo do rejestracji sceny przed uczestnikiem badań. Okulary są podłączone za pomocą kabla z modułem, który rejestruje obraz sceny oraz obrazy oczu. Moduł rejestracyjny jest zwykłym smartfonem z zainstalowanym dedykowanym oprogramowaniem Pupil Invisible Companion, który oprócz nagrywania, również automatycznie wysyła pliki do chmury Pupil Cloud, skąd można je pobrać i poddać dalszej analizie.



Rysunek 2: Stanowisko eyetrackingowe podczas badań.

Tabela 1: Parametry techniczne eyetrackera mobilnego Pupil Insivible [13]

Częstotliwość	200Hz w Pupil Cloud; 120Hz na urządzeniu towarzyszącym
Kamera do obserwacji oczu	200Hz; 192x192px z oświetlaczem podczerwieni

Kamera sceny	30Hz; rozdzielczość 1088x1080; zakres widzenia 82x82°; z możliwością odłączenia
Nagrywanie dźwięku	Mikrofon wbudowany w kamerę sceny
Soczewki	Wykonane z materiału CR 39, zawierają filtr UV, powłokę antyrefleksyjną Anti-Scratch, powłokę chroniąca przed kurzem i wodą
Waga	46,7 g
Oprogramowanie	Pupil Invisible Companion App
Kompatybilność z urządzeniami	OnePlus 8 oraz OnePlus 6
Możliwość wysyłania danych do dedykowanej chmury	Tak

4.4. Opis eksperymentu

Do zrealizowania celów badań opracowano eksperyment, składający się z dwóch części. Podczas pierwszej części – eyetrackingowej, uczestnicy realizowali 3 razy ten sam scenariusz na każdej testowanej aplikacji. Scenariusz składał się z 5 zadań (Tabela 2). Osoby badane testowały aplikacje w kolejności losowej.

Każda sesja badawcza przebiegła według ustalonego schematu i była powtarzana 3 razy dla każdej testowanej aplikacji:

1. Przedstawienie uczestnikowi celu badania, jego przebiegu oraz udzielenie instrukcji dotyczących odpowiedniego postępowania podczas badania.
2. Wyrażenie przez uczestnika zgody na udział w badaniu.
3. Przygotowanie uczestnika do badań: zajęcie odpowiedniej pozycji, nałożenie okularów i wykonanie kalibracji.
4. Nagranie sesji badawczej podczas realizacji zadań w jednej z trzech aplikacji mobilnych.
5. Transfer danych do chmury.
6. Zebranie danych metrycznych, tj. wiek czy płeć.
7. Wypełnienie ankiety SUS dotyczącej doświadczeń użytkownika podczas interakcji z daną aplikacją.
8. Powtórzenie punktów 3-7 dla pozostałych aplikacji.

Tabela 2: Treść zadań do wykonania w scenariuszu badawczym

Lp.	Treść zadania
1	Ustaw jednostkę pomiaru poziomu glukozy we krwi na mg/dL
2	Ustaw zakres hiperglikemii na 180 mg/dL
3	Dodaj do statystyki nową wartość poziomu cukru we krwi
4	Wyeksportuj kopię zapasową danych
5	Wygeneruj raport z dowolnym zakresem czasu

Ocena użyteczności i możliwości skutecznego zapamiętania sposobu działania aplikacji została przeprowadzona za pomocą kwestionariusza SUS. Ankieta SUS składa się z 10 pytań/stwierdzeń (Tabela 3), na które badany udzielał odpowiedzi w klasycznej, pięciostopniowej skali Likerta z zakresu od “Zdecydowanie się nie zgadzam” do “Zdecydowanie się zgadzam”. Zależnie od pytania każdej z opcji przydzielona była wartość punktowa w skali od 1 do 5. Otrzymane oceny zostały przeliczone za pomocą specjalnej formuły, dającej wynik będący wskaźnikiem SUS, który przyjmuje wartość z zakresu 0 - 100. Interpretacja tego wskaźnika jest bardzo prosta: im wyższa jego wartość, tym wyższy poziom użyteczności systemu.

5. Wyniki badań

Do analizy porównawczej wykorzystano następujące metryki:

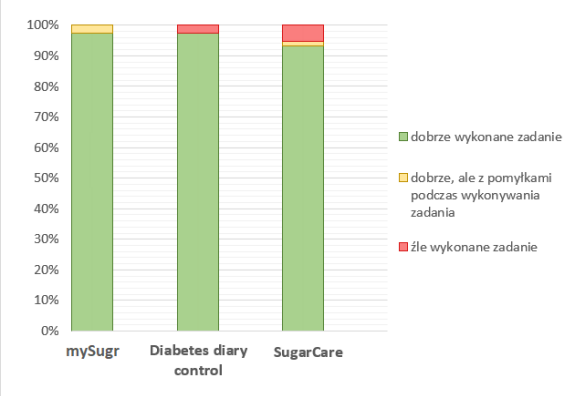
- wskaźnik SUS – ocena subiektywnego poziomu satysfakcji użytkowników oprogramowania;
- czas realizacji zadania – pomiar długości czasu potrzebnego do wykonania zadania (miara wydajności);
- stopa błędów – liczba błędów popełnionych podczas realizacji zadań;
- liczba fiksacji – miara negatywnie skorelowana z efektywnością poszukiwania wzrokowego; dobra organizacja interfejsu powinna zmniejszyć liczbę fiksacji; duża liczba fiksacji może świadczyć o poziomie skomplikowania interfejsu.

5.1. Identyfikacja poprawnych i błędnych realizacji zadań

Pomiar poziomu błędów (stopy błędów) pozwala zidentyfikować liczbę pomyłek popełnianych podczas wykonywania zadania. W ramach badań określono procentową poprawność wykonanych zadań. Wyniki wskazały, że najlepiej wypadła aplikacja *mySugr*, w której 97,3% zadań zostało wykonanych prawidłowo, a 2,7% z pewnymi pomyłkami. Na drugim miejscu uplasowała się aplikacja *Diabetes diary control,* osiągając wynik: 97,5% zadań poprawnie zrealizowanych. Najgorzej w zestawieniu wypadła aplikacja *SugarCare*. Badani poprawnie wykonali za jej pomocą 93,3% poleceń. Do najczęściej popełnianych błędów należały: omyłkowe ustawienie wartości hipoglikemii, zamiast wymaganej hiperglikemii oraz niewykonanie potwierdzenia podczas dodawania nowego pomiaru, co skutkowało brakiem jego zapisu w statystyce.

Aplikacje *mySugr* oraz *Diabetes diary control* uzyskały niemal identyczny poziom błędów. W tej analizie autorskie oprogramowanie *SugarCare* osiągnęło wynik o 4% gorszy. Ze względu na wysoką skuteczność tej

aplikacji i niewielką różnicę w poziomach błędów, trudno jest określić, biorąc pod uwagę tylko tą miarę, który z porównywanych interfejsów ma lepszą użyteczność (Rysunek 3).

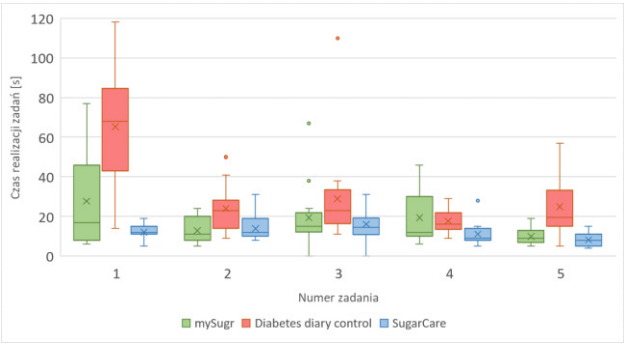


Rysunek 3: Wykres procentowej poprawności wykonywanych zadań.

Z badania wykluczone zostało 5 realizacji zadań z całej puli liczącej 51. Trzy z nich dotyczyły badanego, z wadą wzroku, a pozostałe dwa uległy uszkodzeniu podczas wysyłania danych do chmury Pupil Cloud.

5.2. Pomiar czasów realizacji zadań

Analizie poddano czasy wykonywania poszczególnych zadań dla kolejnych aplikacji. Wyniki przedstawione zostały na wykresie skrzynkowym (Rysunek 4). Pudełko na tym wykresie przedstawia zakres między pierwszym a trzecim kwartylem (Q1 i Q3) danych. Kreska znajdująca się wewnątrz pudełka oznacza medianę wyników, natomiast symbol "x" wartość średnią. Sama szerokość pudełka wskazuje na rozproszenie danych. Wąsy reprezentują dane, które nie są wartościami odstającymi i wskazują na wynik minimalny i maksymalny. Natomiast pojawiające się miejscami kropki oznaczają, że dana wartość w sposób znaczny odstępuje od reszty zbioru danych.

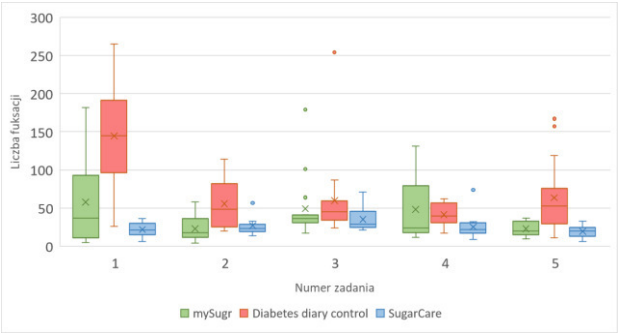


Rysunek 4: Wykres czasów wykonywania poszczególnych zadań.

Na wykresie widać, że największą różnicę między aplikacjami da się zaobserwować przy pierwszym zadaniu. Mediana czasu wykonania zadania w sekundach dla aplikacji m*ySugr* wyniosła 16,7, dla *SugarCare* 12,5, natomiast dla *Diabetes diary control* było to aż 67,9. Obrazuje to jak kluczowe znaczenie ma interfejs przy pierwszym kontakcie z aplikacją. Przy następnych zadaniach czasy są już bardziej zbliżone do siebie, ale wciąż *Diabetes diary control* uzyskuje najgorsze wyniki spośród analizowanych aplikacji. Wyjątkiem może tu być zadanie czwarte, gdzie dla m*ySugr* mediana jest niższa od mediany *Diabetes diary control*, lecz większa liczba badanych osiągnęła dłuższy czas wykonania zadania. Wynikać to może z braku komunikatu potwierdzenia dotyczącego utworzenia kopii zapasowej. Ta przyczyna mogła zdezorientować niektórych użytkowników, którzy kilkakrotnie próbowali naciskać na przycisk odpowiedzialny za utworzenie kopii zapasowej i tym samym wydłużało to czas wykonania zadania.

5.3. Średnia liczba fiksacji

Analizie poddano również metrykę eyetrackingową – średnią liczbę fiksacji podczas wykonywania zadań na poszczególnych aplikacjach. Wyniki dla tego wskaźnika przedstawione zostały na Rysunku 5.



Rysunek 5: Wykres liczby fiksacji na telefon.

Analizując wykres (Rysunek 5) można zauważyć, iż mediana liczby fiksacji dla aplikacji *Diabetes diary control* charakteryzuje się wartością od 1,5 do 7 razy większą w stosunku do aplikacji *SugarCare* oraz m*ySugr*. Oznacza to, że użytkownicy o wiele dłużej i intensywniej wodzili wzrokiem po ekranie, na którym uruchomiona była ta aplikacja, w poszukiwaniu odpowiednich elementów interfejsu. Z tego wynika, że interfejs tej aplikacji jest skomplikowany, nieintuicyjny oraz trudny w użyciu.

Analiza wykresów z Rysunków 4 i 5 pokazuje, że czas wykonywania zadania oraz liczba fiksacji we wszystkich przypadkach są ze sobą skorelowane. To oznacza, że im dłużej wykonywane było zadanie, tym większa liczba fiksacji miała miejsce. Wystąpił tylko jeden wyjątek od tej reguły. W przypadku zadania 2 i aplikacji *Diabetes diary control* można było zauważyć zwiększoną liczbę fiksacji. Taka sytuacja może oznaczać, że osoby badane napotkały na trudności z interfejsem użytkownika podczas wykonywania tego zadania.

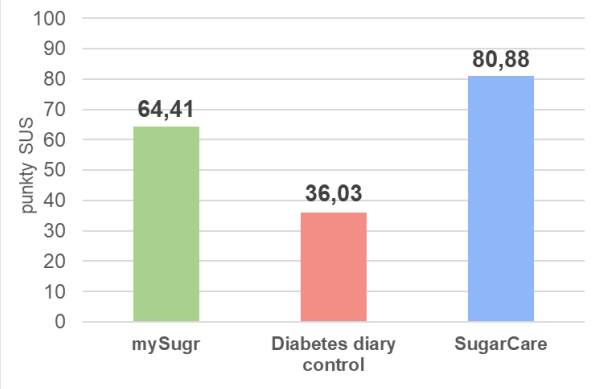
5.4. Test użyteczności SUS

Na podstawie danych zebranych od uczestników eksperymentu za pomocą ankiety SUS, w badaniu eyetrackingowym, obliczono wskaźnik SUS. Wykorzystano do tego celu poniższą formułę:

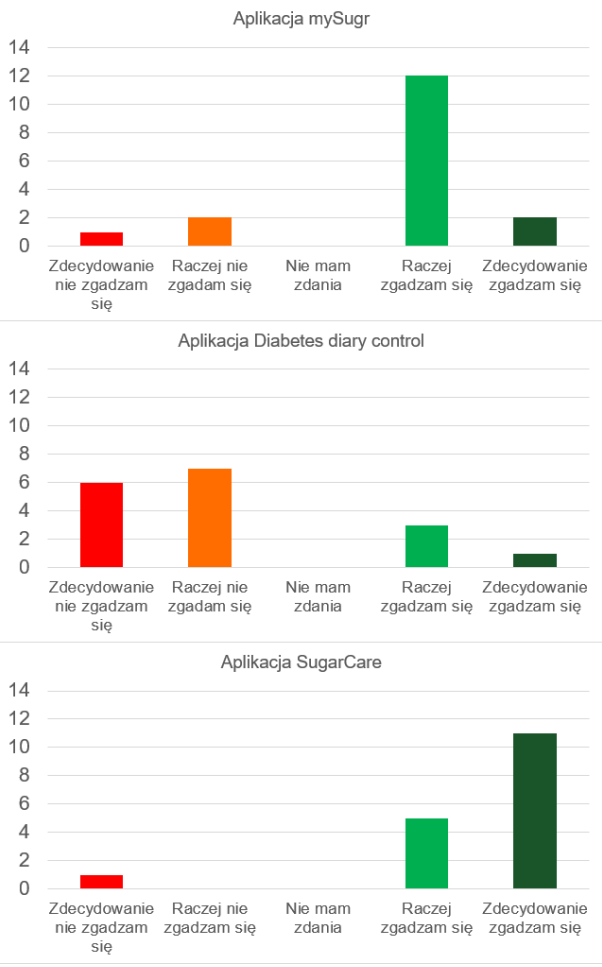
$$SUS = (\sum_{i=1,3,5,7,9} (S_i - 1) + \sum_{i=2,4,6,8,10} (5 - S_i)) * 2,5$$

w której *SUS* jest miarą oceny użyteczności systemu interfejsu, natomiast S_i jest oceną udzieloną przez i-tego użytkownika na poszczególne pytania. Wskaźnik SUS podawany jest w sali od 0 do 100 punktów, gdzie wartości powyżej 68 oznacza wynik dobry [14].

Po zebraniu danych dla każdej aplikacji od wszystkich 17 badanych, ostatecznie uśredniono wyniki i przedstawiono je na Rysunku 6.



Rysunek 6: Wykres wskaźnika SUS dla każdej aplikacji.



Rysunek 7: Rozkład odpowiedzi na pytanie "Większość osób będzie w stanie opanować aplikację bardzo szybko".

Aplikacja m*ySurg* osiągnęła wynik 64,41 punktów, co określane jest jako wynik słaby, *Diabetes diary control* uzyskała 36,02 punktów, co oznacza wynik nie akceptowalny, natomiast autorska aplikacja *SugarCare* zdobyła 80,88 punktów, co jest wynikiem bardzo dobrym (Rysunek 6) [14].

Rysunek 7 przedstawia odpowiedzi udzielone przez użytkowników na stwierdzenie *„Większość osób będzie w stanie opanować aplikacje bardzo szybko"*. Wyniki dla tego pytania charakteryzowały się największą rozbieżnością. W przypadku aplikacji *mySugr* było to 14 na 17 pozytywnych odpowiedzi. Aplikacja *Diabetes diary control* otrzymała tylko 4 pozytywne odpowiedzi, natomiast autorska aplikacja *SugarCare* uzyskała 16 dobrych odpowiedzi. Wyniki dla tego aspektu ankiety SUS są istotne w kontekście wykorzystania tych aplikacji przez osoby starsze. W odczuciu badanych, młodych osób, które są dobrze zaznajomione z nowymi technologiami aplikacja *Diabetes diary control* była trudna do opanowania, ze względu na to, że większość osób wskazała problem z intuicyjnością jej interfejsu. Można więc z dużym prawdopodobieństwem przypuszczać, że obsługa tej aplikacji będzie również na tyle kłopotliwa dla osób starszych chorych na cukrzycę, że korzystanie z aplikacji w tej grupie nie będzie w ogóle możliwe.

6. Wnioski

Celem pracy była obiektywna i rzetelna ocena interfejsów trzech aplikacji mobilnych dla diabetyków, w tym jednej autorskiej. Badanie przeprowadzono za pomocą techniki eyetrackingowej oraz ankiety Skali Użyteczności Systemu. Zebrane dane poddano analizie, by następnie odpowiedzieć na postawioną wcześniej tezę czy „prosty i łatwy w obsłudze, użyteczny interfejs aplikacji dla diabetyków przyspiesza proces wykonywania zadań oraz wpływa pozytywnie na doświadczenia użytkownika". Na podstawie analizy czasów realizacji zadań, średniej liczby fiksacji oraz wskaźnika SUS zaprezentowanych na rysunkach 4, 5 i 6 można stwierdzić, że postawiona hipoteza została potwierdzona.

Wyniki eksperymentu wskazują, iż niewłaściwe dobranie kolorystyki interfejsu, a także za duża liczba opcji zagnieżdżona w kolejnych widokach oraz towarzysząca im nadmierna ilość tekstu może negatywnie wpływać na doświadczenia użytkownika z aplikacją. Wszystkie te elementy zawarte są w aplikacji *Diabetes diary control*, co znacząco wpływało na wydłużenie czasu wykonywania zadań, szczególnie przy pierwszym kontakcie z oprogramowaniem (Rysunek 5 - zadanie 1). Aplikacja ta uzyskała najsłabszy wynik tzn. 36,02 punktów w skali SUS, co wskazuje na jej negatywny odbiór wśród badanych osób. Z analizy nagrań eyetrackingowych wynika, że użytkownicy wielokrotnie wodzili wzrokiem po ekranie, nie mogąc znaleźć odpowiedniego przycisku lub przechodzili do kolejnych zagnieżdżonych stron, próbując znaleźć żądaną opcję.

Zupełnie inaczej sytuacja wyglądała podczas realizowania przez badanych zadań na aplikacjach *mySugr* oraz *SugarCare*. Oba te systemy posiadają nowoczesne, minimalistyczne interfejsy, które zdaniem uczestników

badania były przejrzyste i intuicyjne w obsłudze. Liczba opcji w obu tych aplikacjach była odpowiednio skoncentrowana i nie przekraczała granicy, kiedy interfejs stawał się przytłaczający. Zgodnie z wynikami ankiet zarówno *mySugr*, jak i *SugarCare* otrzymały wysokie oceny użytkowalności, odpowiednio 64,41 i 80,88 punktów w Skali Użyteczności Systemu. Po analizie nagrań oraz ankiet można stwierdzić, iż prostota interfejsu oraz czytelność tekstu znacznie przyspieszały proces wykonywania zadań i sprawiały, że w odczuciu uczestników badań korzystanie z tych aplikacji było bezproblemowe i przyjemne.

Podsumowując, badanie wykazało, że istotne czynniki, takie jak dobranie kolorystyki, liczba i rozmieszczenie dostępnych opcji oraz tekstu mają istotny wpływ na doświadczenia użytkowników aplikacji mobilnych dla diabetyków. Przejrzysty interfejs, odpowiednio wyważona liczba funkcjonalności i klarownie przedstawione informacje przyczyniają się do skutecznego i przyjemnego korzystania z aplikacji. W przypadku aplikacji *Diabetes diary control*, nadmiar opcji i zbyt duża ilość tekstu negatywnie wpływały na efektywność i satysfakcję użytkowników.

Literatura

[1] T. Chomutare, L. Fernandez-Luque, E. Årsand, G. Hartvigsen, Features of Mobile Diabetes Applications: Review of the Literature and Analysis of Current Applications Compared Against Evidence-Based Guidelines, Journal of Medical Internet Research 13(3) (2011) 65-76.

[2] E. Årsand, D. H. Frøisland, S. O. Skrøvseth, T. Chomutare, N. Tatara, G. Hartvigsen, J. T. Tufano, Mobile Health Applications to Assist Patients with Diabetes: Lessons Learned and Design Implications, Journal of Diabetes Science and Technology 6(5) (2012) 1197 – 1206.

[3] P. P. Brzan, E. Rotman, M. Pajnkihar, P. Klanjsek, Mobile Applications for Control and Self Management of Diabetes: A Systematic Review, Journal of Medical Systems 40 (2016) 210-219.

[4] J. Pavlas, O. Krejcar, P. Maresova, A. Selamat, Prototypes of User Interfaces for Mobile Applications for Patients with Diabetes, Computers 8(1) (2019) 1-14, https://doi.org/10.3390/computers8010001

[5] F. J. Represas-Carrera, Á. A. Martínez-Quesb, A. Claveríaa, Effectiveness of mobile applications in diabetic patients' healthy lifestyles: A review of systematic reviews, Primary Care Diabetes 15(5) (2021) 751-760.

[6] H. Fu, S. K. McMahon, C. R. Gross, T. J. Adam, J. F. Wyman, Usability and clinical efficacy of diabetes mobile applications for adults with type 2 diabetes: A systematic review, DiabetesResearch and Clinical Practice 131 (2017) 70-81.

[7] P. A. Punde, M. E. Jadhav, R. R. Manza, A study of Eye Tracking Technology and its applications, 1st International Conference on Intelligent Systems and Information Management (2017) 86-90.

[8] K. Kalimullah, D. Sushmitha, Influence of Design Elements in Mobile Applications on User Experience of Elderly People, Procedia Computer Science 113 (2017) 352-359.

[9] Q. Qu, L. Zhang, W. Chao, V. Duffy, User Experience Design Based on Eye-Tracking Technology: A Case Study on Smartphone APPs, AHFE 2016 International Conference on Digital Human Modeling and Simulation (2016) 303-315.

[10] mySugr - Dzienniczek diabetyka, https://play.google.com/store/apps/details?id=com.mysugr.android.companion&pli=1, [15.05.2023].

[11] Diabetes diary control, https://play.google.com/store/apps/details?id=com.insulindiary.glucosenotes, [15.05.2023].

[12] B. Tulu, D. Strong, L. Wang, Q. He, E. Agu, P. Pedersen, S. Djamasbi, Design Implications of User Experience Studies: The Case of a Diabetes Wellness App, 49th Hawaii International Conference on System Sciences (2016) 3472-3482.

[13] Pupil Labs, Pupil Invisible. Technical Specs & Performance, https://pupil-labs.com/products/invisible/tech-specs/, [15.05.2023].

[14] J. Saouro, 5 Ways to Interpret a SUS Score, https://measuringu.com/interpret-sus-score/, [23.10.2021].

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 341–345
Received: 23 June 2023
Accepted: 4 July 2023

Performance analysis of React v. 18.1.0 and Angular v. 11.0.2 development frameworks

Analiza wydajności szkieletów programistycznych React v. 18.1.0 i Angular v. 11.0.2

Albert Poniedziałek*, Beata Pańczyk

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
This paper presents the results of the comparative analysis of the two JavaScript frameworks. The research was carried out using test applications with the same functionalities, implemented in both environments. The number of bytes occupied by both projects, the number of lines of code, the average RAM consumption and the time efficiency were used as criteria for comparison. Automatic load tests and technical analysis of both environments were performed. The results showed that React is able to better handle a large amount of data (around 100000 records), while for small data sets oscillating around 1000 records, no significant differences in both application performance were demonstrated. For a medium number of records (10000), React proved to be more efficient.

Keywords: React; Angular; performance analysis

Streszczenie
W artykule porównano dwa szkielety programistyczne we wskazanych wersjach. Badania przeprowadzono za pomocą aplikacji testowych o takich samych funkcjonalnościach, zaimplementowanych w obu środowiskach. Za kryterium porównawcze przyjęto liczbę bajtów zajmowanych przez oba projekty, liczbę wierszy z kodem, koniecznych do implementacji, średnie zużycie pamięci RAM oraz wydajność czasową. Wykonano automatyczne testy obciążeniowe oraz analizę techniczną obu środowisk. W wyniku badań stwierdzono, że React lepiej radzi sobie z obsługą dużej liczby danych (około 100000 rekordów), natomiast w przypadku małych zbiorów danych oscylujących w okolicach 1000 rekordów nie wykazano istotnych różnic w działaniu obu aplikacji. W przypadku umiarkowanej liczby rekordów (10000) bardziej wydajny okazał się React.

Słowa kluczowe: React; Angular; analiza wydajnościowa

*Corresponding author
Email address: **albert.poniedzialek@pollub.edu.pl** (A. Poniedziałek)

1. Wstęp

Obecnym standardem w tworzeniu aplikacji internetowych są aplikacje typu SPA (ang. Single Page Application). Dostępnych jest wiele różnych szkieletów programistycznych, które przyspieszają proces generowania kodu JavaScript. Najbardziej popularne z nich to Angular, React.js, Vue.js, Ember.js czy Backbone.js [1]. Są to rozwiązania, które oferują bardzo zaawansowane funkcje i gotowe do użycia komponenty. Problem stanowi fakt, że każde środowisko oferuje nieco inny zestaw funkcjonalności i nierzadko początkujący programista ma duży problem z wybraniem odpowiedniego narzędzia do pracy.

2. Cel i zakres badań

W artykule porównano dwa najbardziej popularne szkielety programistyczne (ReactJS v.18.1.0 oraz AngularJS v11.02) w celu wskazania, który z nich jest wydajniejszy. W momencie podjęcia badań (rok 2022) wskazane wersje były najnowsze.

W pracy zwrócono uwagę na podobieństwa oraz różnice pomiędzy obydwoma narzędziami, zbadano efektywność czasową aplikacji przy wykonywaniu operacji na różnej ilości danych.

Postawiono dwie tezy:

T1. React jest bardziej wydajny czasowo przy obsłudze dużej ilości danych w porównaniu do środowiska Angular.

T2. Angular jest bardziej wydajny czasowo przy pracy z mniejszą ilością danych.

3. Analiza literatury

Na temat tworzenia aplikacji internetowych istnieje wiele publikacji, od artykułów poświęconych analizie składni poszczególnych szkieletów programistycznych, po analizę cech danych środowisk i porównania wydajności w pracy z danymi.

Stanowią one dobre źródła wiedzy do nauki i są także punktem wyjścia do badań, prowadzonych dla kolejnych wersji szkieletów programistycznych, które wprowadzają coraz to nowe udogodnienia i jednocześnie zwiększają efektywność działania aplikacji internetowych.

W artykule R. Nowackiego z roku 2016 [2] myślą przewodnią jest analiza porównawcza trzech szkieletów programistycznych języka JavaScript (React, Angular i EmberJS). Podano tam cztery kryteria porównawcze:

trudność nauki biblioteki, dodatkowe narzędzia charakterystyczne dla danej platformy, wydajność na urządzeniach stacjonarnych oraz wydajność na urządzeniach mobilnych. Konkluzją badań było wskazanie środowiska React jako najlepszego narzędzia, między innymi z powodu dużej społeczności zbudowanej wokół tego środowiska.

W pracy W. Stępniaka i Z. Nowaka z roku 2017 [3] przeanalizowano skuteczność metod przyspieszających proces ładowania aplikacji typu SPA, w tym mechanizmów oferowanych przez protokół HTTP/2. Zbadano wydajność gotowych komponentów dostarczanych ze szkieletem programistycznym Angular.

Za pomocą niewielkich aplikacji implementujących tę samą funkcjonalność w środowisku Angular w wersji 2.0, porównano również wydajność różnych konstrukcji oferowanych przez Angular. W badaniu wykorzystano serwer Apache, przeglądarkę Chrome dla systemu Linux i Android jak również narzędzia Browser-perf i h2load. Wnioskami było wskazanie środowiska jako dobre oraz wydajne podczas tworzenia aplikacji typu SPA.

W artykule A. Kumara i R. K. Singha z roku 2016 [4] analizie poddano także platformy React i Angular. Cechy obu szkieletów przedstawiono w postaci tabeli. Oceniano m.in. takie subiektywne cechy jak krzywą uczenia. We wnioskach stwierdzono, że oba środowiska są porównywalne a podstawowym kryterium wyboru powinny być potrzeby samego programisty i jego poziom doświadczenia.

Artykuł M. Kaluża i K. Troskot z roku 2018 [5] wyjaśnia różnice w modelu aplikacji MPA (ang. Multi Page Application) i SPA. Zdefiniowano tu zagadnienia wpływające na rozwój aplikacji MPA i SPA. Autorzy poddali analizie również inne szkielety programistyczne, takie jak Angular, Vue.js i React-js, poprzez oceny jakościowe każdego z zagadnień wpływających na rozwój aplikacji MPA i SPA. W podsumowaniu wskazano VueJS, jako najlepszy wybór.

Artykuł K. Kowalczyka z roku 2022 [6] zawiera porównanie wydajności bibliotek Angular oraz React. Badanie zostało zrealizowane poprzez implementację aplikacji w obu środowiskach i zbadaniu ich wydajności. W artykule zestawiono jedynie zalety i wady omawianych szkieletów.

Reasumując, literatura dotycząca porównania biblioteki React i Angular nie daje jednoznacznej odpowiedzi na pytanie, jak również często opisuje starsze wersje obu środowisk, przez co nie możliwe jest wskazanie, które ze środowisk jest lepszym wyborem dla programisty. Studiując dostępną literaturę czytelnik jest w stanie wywnioskować jedynie kilka faktów:

React posiada dużą społeczność zebraną wokół siebie jak również dobrze radzi sobie z dużymi zbiorami danych, zaś Angular jest wydajny przy małych zbiorach danych oraz posiada wiele funkcjonalności wspomagających tworzenie aplikacji. Dodatkowo czytelnik może odnieść słuszne wrażenie, że wybór pomiędzy środowiskami React czy Angular wcale nie jest trywialny i wymagane są dalsze badania w tym kontekście.

W niniejszym artykule podjęto kolejną próbę porównania obu szkieletów w wersjach (rok 2022), React 18.1.0 i Angular 11.0.1.

4. Charakterystyka badanych technologii

W tym rozdziale zestawiono najważniejsze cechy wyróżniające oba badane narzędzia.

4.1. React

React jest biblioteką utworzoną dla języka JavaScript, jego głównym przeznaczeniem jest tworzenie interfejsu graficznego w aplikacjach internetowych. Jest prosty w instalacji i użyciu, pozwala na korzystanie z języka JavaScript, jak również z TypeScript.

Cechą najbardziej wyróżniającą React jest wirtualny element DOM (ang. Document Object Model). Umożliwia on realnemu elementowi DOM wykonanie jak najmniejszej liczby operacji podczas ponownego renderowania treści strony, aktualizując jedynie te fragmenty, które uległy zmianie. Proces ten działa na zasadzie porównania wirtualnego elementu DOM do wyświetlonego DOM. Dzięki takiemu podejściu polepsza się wydajność działania strony.

Kolejną cechą szczególną React jest język JSX utworzony specjalnie na potrzeby tej biblioteki. Umożliwia on stosowanie kodu HTML i XML oraz samych komponentów React wewnątrz funkcji JavaScript. JSX umożliwia również przekazywania tak przygotowanych elementów do kolejnych funkcji bez potrzeby tworzenia ciągów znakowych, jak w przypadku innych bibliotek.

Ponadto React bazuje na komponentach posiadających własne stany, dzięki czemu nie wpływają nawzajem na siebie i nie wymuszają aktualizacji innego komponentu, gdy same są aktualizowane (co zapewnia wirtualny DOM). Dodatkowo rozdzielenie stanów bazujące na *React Hooks* pozwala na śledzenie zmian w aplikacji i wykonanie zdefiniowanych czynności w momencie aktualizacji stanu komponentu.

React wspiera również renderowanie po stronie serwera, dając możliwość wysyłania gotowych już komponentów z serwera do aplikacji klienckiej, tym samym zwiększając płynność działania i poprawiając ogólne wrażenie podczas korzystania z witryny.

Dużym atutem biblioteki React jest możliwość konwersji projektu w React Native, który jest wieloplatformowy w przeciwieństwie do podstawowej wersji biblioteki i jest wykorzystywany również do tworzenia aplikacji dla systemów Android, Android TV, iOS, macOS, tvOS i Windows.

4.2. Angular

Angular jest szkieletem programistycznym o otwartym kodzie, jest platformą m.in. do tworzenia aplikacji typu SPA. Został napisany w języku TypeScript, właścicielem licencji i głównym deweloperem jest firma Google.

Angular, podobnie jak React, wspiera renderowanie po stronie serwera, posiada moduły i jest zorientowany na komponenty. Ma dobre wsparcie dla przeglądarek mobilnych w przeciwieństwie do React, który musi być specjalnie do tego celu przestawiony do React Native.

Ponadto Angular posiada narzędzia konsolowe wspomagające wytwarzanie aplikacji, na przykład łatwe debugowanie. Posiada on również wsparcie dla wersjonowania semantycznego (ang. Semantic Versioning).

Domyślnym językiem dla Angular jest TypeScript, który jest bardziej skomplikowanym językiem od JavaScript. Angular daje możliwość zmienienia natywnego języka na JavaScript, może to jednak generować wiele problemów dla niedoświadczonego programisty.

Podstawowe cechy obu rozwiązań zostały przedstawione w tabeli 1.

Tabela 1: Podstawowe cechy React i Angular

Atrybut	React	Angular
Autor	Jordan Walke (pracownik Facebook)	Google
Data powstania	marzec 2013	maj 2016
Aktualnie rozwijane przez	Facebook	Google
Typ licencji	licencja MIT	licencja MIT
Możliwość tworzenia aplikacji mobilnych	React Native	Ionic Framework
Struktura projektu	Elastyczna, oparta na komponentach	Stała i złożona struktura platformy, oparta na komponentach
Domyślny używany język	JavaScript z możliwością użycia składni JSX	TypeScript
struktura DOM	wirtualny DOM	rzeczywisty DOM
Krzywa uczenia	relatywnie niska	Może być stroma dla początkujących
Dynamiczne wiązanie interfejsu użytkownika	Bezpośrednie połączenie stanów z interfejsem użytkownika	Wiązanie interfejsu użytkownika na poziomie zwykłego obiektu lub właściwości

5. Metoda badań

Na potrzeby badań opracowane zostały dwie proste aplikacje typu SPA. Aplikacje, oferują te same funkcjonalności i umożliwiają generowanie i wysłanie danych do bazy, pobrania i wyświetlenie danych, sortowanie i usuwanie elementów struktury DOM przechowujących dane. Aplikacje wykonują operacje na relacyjnej bazie danych MySQL, przechowującej rekordy z informacjami o użytkownikach.

Oba projekty zostały utworzone na tym samym urządzeniu i poddane jedynie drobnej modyfikacji plików package.json, tak aby uzyskać odpowiednie wersje środowiska. Po instalacji niezbędnych modułów, dodano dodatkowe elementy, które umożliwiają wykonywanie testów obciążeniowych.

Wygląd aplikacji testowych jest identyczny, dzięki czemu całkowicie został wyeliminowany wpływ kodu CSS na działanie aplikacji testowych.

Obsługiwane przez aplikacje dane mają postać tablicy z ośmioma polami zawierającymi dane wygenerowane za pomocą biblioteki fakerjs [7], są nimi m.in. imię, nazwisko, płeć, kraj zamieszkania. Z aplikacji klienckich dane przesyłane są do aplikacji serwerowej, zaimplementowanej na platformie NodeJS i Express.

Porównanie wydajności aplikacji polegało na wyznaczaniu czasu wykonywania operacji na danych (pobieranie, wstawianie, sortowania i usuwanie różnej liczby rekordów z bazy danych).

Programy uruchomione zostały na lokalnym sprzęcie i były wystawione na lokalnych portach.

Analiza porównawcza została przeprowadzona z zastosowaniem testów manualnych i automatycznych. Na potrzeby badań zdefiniowano 5 różnych scenariuszy, w 3 wariantach (liczba rekordów: 1000, 10000 i 100000). Każdy scenariusz został wykonany po 20 razy. W skrajnych przypadkach jeden test trwał nawet 30 minut. Parametry sprzętu, na którym zostały wykonane badania zostały przedstawione w tabeli 2.

Tabela 2: Parametry sprzętu do testów

Parametr	Wartość
Procesor	AMD Ryzen 7 5800H
Ram	16 GB
Dysk	1000 GB SSD
Karta graficzna	NVIDIA GeForce RTX 3070
System operacyjny	Windows 10 Home

Dla testów automatycznych zrealizowano 5 scenariuszy badawczych:
S1. Generowanie danych testowych,
S2. Wysłanie i zapis w bazie wygenerowanych danych.
S3. Pobranie danych z bazy i ich wyświetlenie,
S4. Sortowanie wyświetlonych danych,
S5. Usunięcie elementów przechowujących dane.

Po każdym teście przeglądarka była resetowana co wyeliminowało wzajemną interferencję między kolejnymi testami.

6. Analiza wyników badań

Wyniki badań zestawiono w kolejnych podrozdziałach.

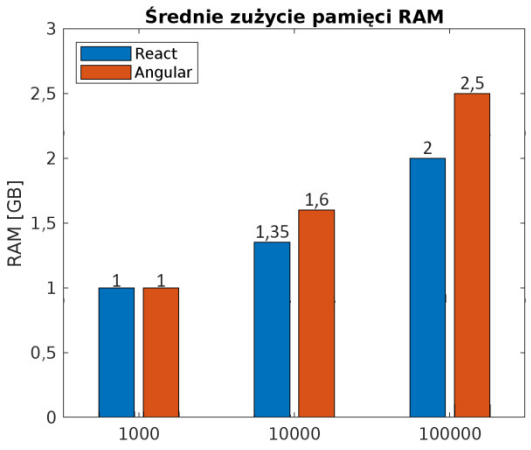
6.1. Analiza testów manualnych

Dane z testów dotyczących wykorzystania sprzętu przedstawione zostały w postaci wykresów słupkowych na Rysunkach 1 i 2.

W przypadku pobierania oraz wysyłania 1000 rekordów (Rysunek 1) miało miejsce niewielkie zużycie pamięci, około 1GB w trakcje całej akcji. Operacja wysyłania danych trwała relatywnie krótko, średnio około 15.5s dla React oraz 17s dla Angular. Operacja pobierania danych z bazy trwała średnio około 6s dla obu środowisk.

Przy umiarkowanym obciążeniu dla 10000 rekordów (Rys.1) można zauważyć większą różnicę w wykorzystaniu pamięci RAM przez oba środowiska. W przypadku biblioteki React jest to wartość powyżej 1GB a momentami nawet 1.5GB. Dla Angular, zużycie pamięci RAM ciągle rosło aż do około 1.7GB bez żadnego zwolnienia pamięci.

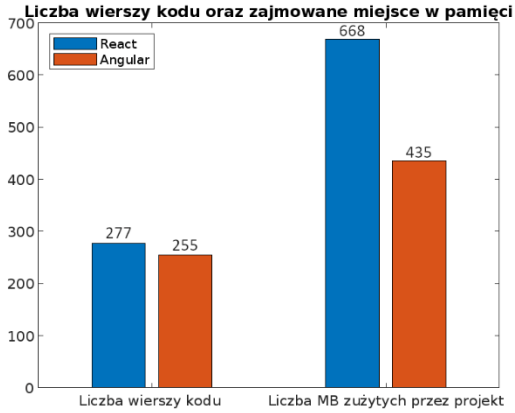
Najciekawsze rezultaty zaobserwowane dla wariantu 100000 rekordów (Rysunek 1). Podczas tego testu zużycie RAM osiągało nawet 5GB dla React pod koniec wykonywania testu. W przypadku Angular, zużycie RAM dochodziło nawet do 6GB.



Rysunek 1: Średnie zużycie pamięci RAM dla różnej liczby rekordów.

Podsumowując system zwalniania pamięci działał cały czas i zmniejszał na bieżąco zużycie jednak odbywało się to zbyt wolno, wskutek czego zużycie pamięci potrafiło nadal narastać.

Z tego wynika, że odbieranie oraz wysyłanie rekordów w sposób rekurencyjny nie jest najlepszym rozwiązaniem, gdyż środowiska oraz przeglądarki nie mają wbudowanych dostatecznie dobrych systemów zarządzania pamięcią RAM. Przy większej liczbie rekordów zużycie nadal będzie rosło.



Rysunek 2: Liczba linii kodu programistycznego i miejsce w pamięci zajmowane przez projekty.

Dla obu projektów sprawdzono ile miejsca zajmują w pamięci (Rysunek 2). Biblioteki oraz pliki utworzone przez środowisko React zajmują 668 MB pamięci, podczas gdy Angular potrzebuje tylko 435MB, co stanowi około 65% tego, co potrzebuje React.

W przypadku liczby linii kodu (Rysunek 2) w Angular należało napisać o 22 linie kodu mniej (8% mniej) niż w React.

6.2. Testy automatyczne

Dane z testów automatycznych przedstawione zostały w postaci wykresów pudełkowych.

Wykresy tego typu przedstawiają dane na temat mediany oznaczonej jako czerwona linia w obrębie pudełka. Prezentują też dane na temat 25-tego oraz 75-tego centyla (oznaczone jako dolne i górne części pudełka). „Wąsy" pokazują najmniejsze i największe wartości, które nie zostały oznaczone jako odbiegające od reszty. Wartości znacznie odbiegające od pozostałych są oznaczone symbolem czerwonego plusa.

Rysunki 3-5 prezentują wyniki testów w zależności od liczby rekordów.

6.2.1. Wariant obsługi 1000 rekordów

Dla 1000 rekordów (Rysunek 3) oba środowiska uzyskały zbliżone wyniki. React radzi sobie lepiej dla scenariuszy wysyłania (12% szybciej), pobierania (11% szybciej) i usuwania (13% szybciej). Angular szybciej wykonuje operacje sortowania (5% szybciej) i generowania danych (4% szybciej).

Z testów wynika zatem, teza o lepszej obsłudze małej liczby danych nie może zostać ani potwierdzona, ani zaprzeczona, ponieważ wyniki pomiarów nie wskazują jednoznacznie na żadne ze środowisk.

6.2.2. Wariant obsługi 10000 rekordów

Dla 10000 rekordów (Rysunek 4) bardziej jednoznacznie widać różnice w szybkości wykonywania operacji na danych. React szybciej wykonał cztery operacje: sortowanie (15% szybciej), usuwanie (25% szybciej), wysyłanie (12% szybciej) i pobieranie (31% szybciej).

Jedynie w przypadku generowania 10000 rekordów Angular wypadł lepiej o około 3%.

6.2.3. Wariant obsługi 100000 rekordów

W przypadku obciążenia największą badaną liczbą rekordów (Rysunek 5) React także wykonał zdecydowanie szybciej operacje sortowania (55% szybciej), usuwania (15% szybciej) i pobierania (5% szybciej). W przypadku wysyłania średnie czasy różnią się o około 3% na korzyść środowiska React, jednak z analizy wykresów (Rysunek 5) wynika, że dla React wyniki są bardziej „rozrzucone", zaś w przypadku Angular wyniki są bardziej zwarte. Tylko dla przypadku generowania danych nie można jednoznacznie wskazać żadnego ze środowisk, ponieważ na wykresie pudełkowym (Rysunek 5) są one niemal takie same a średnie czasy są prawie identyczne.

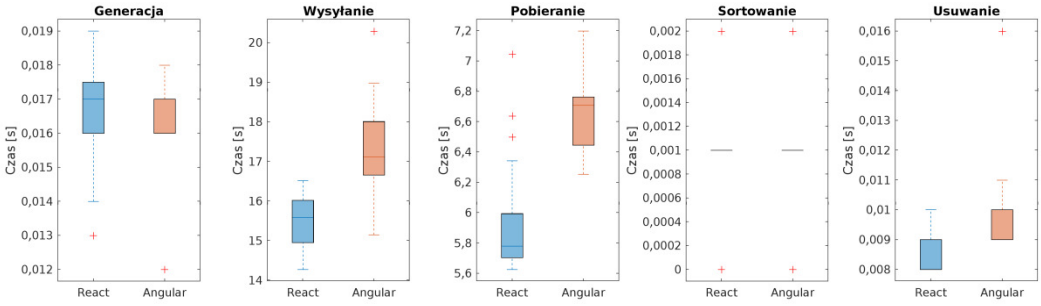
W wyniku otrzymanych rezultatów można potwierdzić tezę, że React szybciej wykonuje operacje na dużej liczbie danych.

7. Wnioski

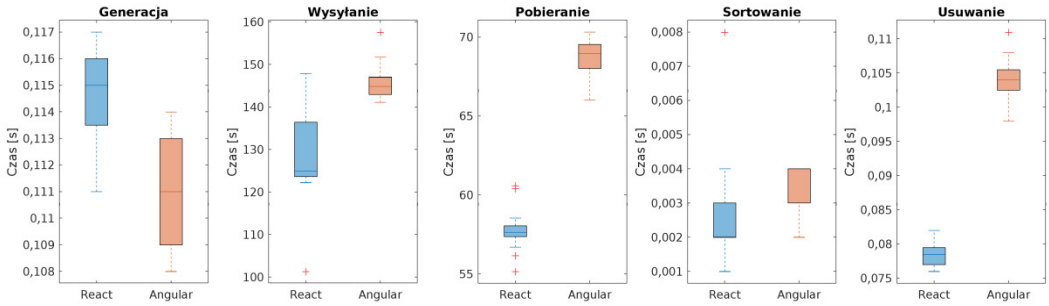
Z analizy wyników przedstawionej w rozdziale 6 można potwierdzić tezę T1 dotyczącą lepszej wydajności środowiska React podczas obsługi dużej liczby danych (100000 rekordów). Wykonane badania wskazały większą szybkość operacji wykonywanych na danych przez aplikację React na poziomie od 15% do nawet 55% w scenariuszu z sortowaniem 100000 rekordów.

Teza T2 dotycząca lepszej wydajności środowiska Angular podczas obsługi małej liczby danych (1000 rekordów) nie może zostać ani potwierdzona, ani obalona, ponieważ uzyskane rezultaty nie wskazują jednoznacznie na żadne ze środowisk. W tym wariancie

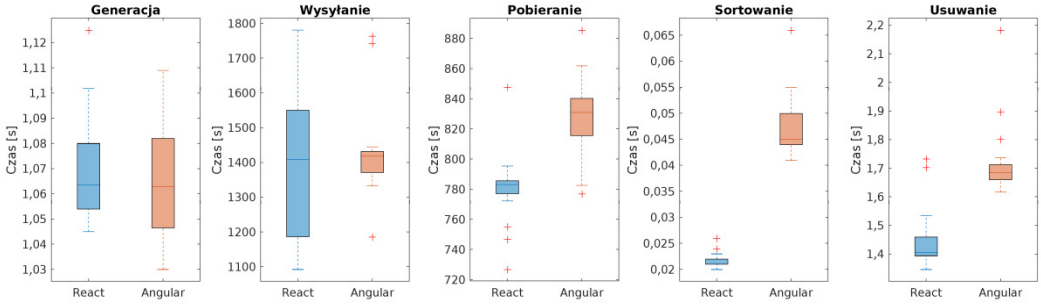
3 scenariusze wykazały lepsze działanie biblioteki React zaś 2 pozostałe wskazały, że szybszy jest Angular.



Rysunek 3: Wyniki testów dla obu środowisk dla wariantu obsługi 1000 rekordów.



Rysunek 4: Wyniki testów dla obu środowisk dla wariantu obsługi 10000 rekordów.



Rysunek 5: Wyniki testów dla obu środowisk dla wariantu obsługi 100000 rekordów.

Podsumowując oba szkielety programistyczne poradziły sobie dobrze z zadanym obciążeniem i często różnice czasowe były bardzo małe.

Literatura

[1] Statystyki popularności JavaScript 2022, https://2022.stateofjs.com/en-US/, [20.05.2023].

[2] R. Nowacki, M. Plechawska-Wójcik, Analiza porównawcza narzędzi do budowania aplikacji Single Page Application – AngularJS, ReactJS, Ember.js, Journal of Computer Sciences Institute 2 (2016) 98-103, https://doi.org/10.35784/jcsi.122.

[3] W. Stępniak, Z. Nowak, Performance analysis of SPA web systems, Advances in Intelligent Systems and Computing 521 (2017) 235-247, https://doi.org/10.1007/978-3-319-46583-8_19.

[4] A. Kumar, R. K. Singh, Comparative Analysis of AngularJS and ReactJS, International Journal of Latest Trends in Engineering and Technology 7 (2016) 225-227, https://doi.org/10.21172/1.74.030.

[5] M. Kaluža, K. Troskot, B. Vukelić, Comparison of front-end frameworks for web applications development, Zbornik Veleučilišta u Rijeci 6 (2018) 261-282, https://doi.org/10.31784/zvr.6.1.19.

[6] R. Ollila, N. Mäkitalo, T. Mikkonen, Modern Web Frameworks: A Comparison of Rendering Performance, Journal of Web Engineering 21(03) (2022) 789–814, https://doi.org/10.13052/jwe1540-9589.21311.

[7] Strona główna biblioteki fakerjs, https://fakerjs.dev/, [08.06.2023].

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 346–351
Received: 27 June 2023
Accepted: 5 September 2023

A comparative analysis of resource use in the Flutter and React Native frameworks

Analiza porównawcza wykorzystania zasobów w szkieletach programistycznych Flutter oraz React Native

Mateusz Markowski*, Jakub Smołka

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The article describes a comparative analysis of the resource use efficiency in mobile applications developed using Flutter and React Native frameworks. The study consisted in creating two mobile apps with the same functionalities and then comparing usage of the following resources: Virtual Memory (VIRT), Shared Memory (SHR), Central Processing Unit (CPU), Resident Set Size (RES) and Memory (MEM). The functionality of the application consisted of actions on the GUI. The test was conducted on a Huawei P20 Lite smartphone, using the Android Debug Bridge (ADB) tool and custom scripts in the Bash shell language. The results do not clearly indicate which technology is more efficient.

Keywords: Flutter; React Native; efficiency; mobile applications

Streszczenie

Artykuł opisuje analizę porównawczą efektywności wykorzystania zasobów aplikacji mobilnych stworzonych za pomocą szkieletów programistycznych Flutter oraz React Native. Badanie polegało na stworzeniu dwóch aplikacji mobilnych z tymi samymi funkcjonalnościami, a następnie porównaniu wykorzystania następujących zasobów: Virtual Memory (VIRT), Shared Memory (SHR), Central Processing Unit (CPU), Resident Set Size (RES) oraz Memory (MEM). Na funkcjonalność aplikacji składały się działania na interfejsie graficznym użytkownika. Badanie zostało przeprowadzone na telefonie Huawei P20 Lite, za pomocą narzędzia Android Debug Bridge (ADB) oraz własnych skryptów w języku powłoki Bash. Wyniki nie wskazują jednoznacznie, która technologia jest wydajniejsza.

Słowa kluczowe: Flutter; React Native; wydajność; aplikacje mobilne

*Corresponding author
Email address: mateusz.markowski@pollub.edu.pl (M. Markowski)

1. Wstęp

Od wielu lat zaobserwować można trend wzrostowy na rynku urządzeń mobilnych. Dostawcy prześcigają się w dostarczaniu użytkownikom coraz bardziej funkcjonalnych smartfonów, tabletów, smartbandów czy innych tego typu urządzeń. Dominującym systemem operacyjnym jest Android jednak coraz więcej osób posiada także urządzenia z systemem iOS. Standardowo aplikacje dla systemu Android tworzone są w językach Java/Kotlin w środowisku Android Studio natomiast dla iOS z wykorzystaniem języka Swift oraz środowiska programistycznego XCode. Tworzenie aplikacji dla dwóch różnych platform wiąże się ze zwiększonymi kosztami, co wywołuje konieczność powołania dwóch zespołów programistów, którzy pisali tą samą aplikację w dwóch różnych technologiach. W związku z dużymi kosztami tworzenia aplikacji mobilnych firmy takie jak Meta czy Google zaczęły pracę nad własnymi technologiami umożliwiającymi programistom pisanie jednocześnie na obydwa systemy mobilne. Tak powstały dwa szkielety programistyczne Flutter oraz React Native, które pojawiły się na rynku odpowiednio w 2017 roku oraz 2015 roku.

Celem niniejszego artykułu jest porównanie wydajności aplikacji mobilnych stworzonych z wykorzystaniem szkieletów programistycznych React Native i Flutter. W niniejszym artykule skupiono się na porównaniu zużycia zasobów przez aplikacje stworzone z użyciem frameworków wieloplatformowych działających w systemie Android. Motywacją do zbadania tych parametrów był fakt, iż w dotychczasowych badaniach obu technologii skupiano się na zupełnie innych charakterystykach wydajności, takich jak czas renderowania klatek. Postawiono następującą hipotezę badawczą „Aplikacje mobilne tworzone z wykorzystaniem technologii Flutter są wydajniejsze od aplikacji mobilnych tworzonych z wykorzystaniem technologii React Native".

2. Przegląd literatury

W celu przeprowadzenia rzetelnych badań został wykonany przegląd literatury. W artykule [1] przeprowadzono badanie porównawcze wydajności aplikacji mobilnych z wykorzystaniem technologii natywnych Flutter, React Native, iOS oraz Android. Celem artykułu było zbadanie czy aplikacje między-platformowe są wydajniejsze od aplikacji natywnych. Na podstawie otrzymanych wyników autorzy określili, że technologie natywne są szybsze od technologii dedykowanych konkretnym systemom mobilnym. Na podstawie otrzymanych wyni-

ków dotyczących wydajności aplikacji autorzy nie byli w stanie stwierdzić, które technologie są wydajniejsze.

Artykuł [2] opisuje porównanie wydajności dwóch aplikacji między platformowych. Autor tego badania skupił się na zmierzeniu liczby porzuconych ramek w określonym czasie przez szkielety programistyczne React Native oraz Flutter. Celem badania było sprawdzenie czy istnieje znacząca różnica pomiędzy wyżej wymienionymi technologiami. Na podstawie otrzymanych wyników stwierdzono, że szkielet programistyczny Flutter wypadł lepiej od szkieletu programistycznego React Native, pod względem liczby porzucanych ramek w aplikacjach mobilnych zawierających bardzo duże listy.

Wydajność aplikacji mobilnych można zmierzyć również z wykorzystaniem takich parametrów jak: zużycie pamięci, wykorzystanie CPU, zużycie baterii oraz FPS. W tym celu wykorzystano technologię Flutter oraz Native Mobile App [3]. Celem tego badania było odpowiedzenie na następujące pytanie: „Czy lepiej tworzyć jedną aplikację, która będzie kompatybilna z wieloma platformami, czy też wydajniej jest stworzyć wiele aplikacji dla różnych systemów?". Po przeprowadzeniu badań Autorzy stwierdzili, że rozmiar aplikacji natywnych jest niewiele mniejszy od rozmiaru aplikacji mobilnej stworzonej w technologii Flutter. Zużycie baterii jest na podobnym poziomie w obydwu technologiach. Jednak pod względem liczby wyświetlanych klatek na sekundę lepiej wypadł szkielet programistyczny Flutter.

Ważnym aspektem tworzenia aplikacji mobilnych jest również proces ich tworzenia oraz umiejętność korzystania z dokumentacji dostarczanej przez twórców. Na takie badanie zdecydowano się w artykule [4]. Gdzie Autorzy porównując technologię React Native oraz Flutter zdecydowali się przebadać obydwa szkielety wśród programistów. Dodatkowo porównano ich składnię, strukturę, wykorzystanie zmiennych, przepływ kontroli i danych oraz architekturę, na której są oparte wyżej wymienione technologie. W tym celu również zdecydowano się na przeszukanie dokumentacji każdego szkieletu programistycznego.

Po przeprowadzeniu badań autorzy uznali, że obie technologie Flutter oraz React Native posiadają wiele użytecznych aspektów. Technologia React Native była bardziej zaawansowana w porównaniu do technologii Flutter w obszarach: przygotowania środowiska programistycznego, rozwoju aplikacji internetowych, różnorodności dla komponentów gotowych do użycia oraz dostępnych zasobów. Szkielet programistyczny Flutter posiada trzystopniowy system testowy, który był bardziej przydatny pod względem rozmiaru aplikacji, zajmując mniej miejsca. Dodatkowo był szybszy od szkieletu programistycznego React Native w momencie pierwszego otwierania aplikacji. Podsumowując autorzy nie znaleźli żadnej wyróżniającej charakterystyki sugerującej, która technologia jest wydajniejsza.

W artykule [5] przedstawiono badanie dotyczące porównania aplikacji mobilnych stworzonych w szkielecie programistycznym Flutter z aplikacjami natywnymi. Badanie polegało na stworzeniu podobnych aplikacji o takich samych funkcjonalnościach w technologiach: Kotlin, Flutter oraz Swift. Celami tego badania było porównanie aplikacji pod względem wydajności procesora oraz sprawdzenie ilości kodu potrzebnego do wytworzenia dwóch identycznie działających aplikacji mobilnych. Z otrzymanych wyników wydajności procesora wynika, że nie ma prawie żadnej różnicy pomiędzy szkieletem programistycznym Flutter oraz technologiami iOS i Kotlin. Autor zaznacza, że w celu zweryfikowania otrzymanych wyników należy przeprowadzić dalsze testy, które potwierdzą otrzymane wyniki. Po porównaniu ilości kodu potrzebnego do stworzenia trzech podobnych aplikacji Autor zauważył, że w technologii Flutter wystarczył napisać 125 linii kodu. W technologii Swift 363 linie kodu. Natomiast w technologii Kotlin 217 linii kodu.

Artykuł [6] przedstawia porównanie zalet oraz wad aplikacji między-platformowych oraz natywnych. Badanie polegało na porównaniu wielkości projektów, w których wykorzystywane są technologie: React Native, Java, Kotlin, Flutter. Zbadano popularność technologii, do tego celu posłużono się danymi z Google Trends. Przedstawiono także liczbę ofert pracy dla technologii: Android, iOS, Flutter oraz React Native. Celem pracy było przeanalizowanie zalet i wad natywnych aplikacji mobilnych oraz aplikacji między-platformowych. Po przeanalizowaniu wszystkich dostępnych danych Autorzy uzyskali wyniki, z których wynika, że nie da się udzielić jednoznacznej odpowiedzi, które technologie są lepsze natywne czy między-platformowe. Wynika to z faktu, iż każda z wyżej wymienionych technologii posiada własne wymagania. Dodatkowo każdy programista posiada inny poziom wiedzy oraz preferencji dotyczących szkieletów programistycznych.

3. Metoda badawcza

W celu zbadania wydajności aplikacji opracowano scenariusz badawczy, którego zadaniem było zbadanie następujących parametrów aplikacji mobilnej: Virtual Memory (VIRT), Shared Memory (SHR), Central Processing Unit (CPU), Resident Set Size (RES), Memory (MEM). Parametr VIRT określa całkowitą ilość pamięci wirtualnej, która jest używana przez aplikację. Jest to pamięć, która dostępna jest dla procesu. Może obejmować zarówno pamięć fizyczną jak i pamięć dostępną na dysku. Wartość tego parametru mierzona jest w gigabajtach. Parametr SHR określa ilość pamięci współdzielonej przez procesy. Może być ona używana przez różne wątki i procesy w tym samym czasie. Wartość tego parametru mierzona jest w megabajtach. CPU to parametr odnoszący się do wykorzystania procesora przez aplikację. Wyrazić ją można jako procentowy udział czasu procesora przeznaczonego na wykonanie kodu aplikacji. RES określa ilość obecnie zajmowanej pamięci RAM przez aplikację, mierzona jest w megabajtach. Parametr MEM odnosi się do ogólnego wykorzystania pamięci przez aplikację. Może obejmować pamięć wirtualną jak i pamięć fizyczną. Mierzony jest w procentach [7].

Tabela 1 przedstawia scenariusz badawczy, który opracowano w celu przeprowadzenia badań wydajności aplikacji mobilnych. Grupa badawcza składała się z 15 studentów. Przed wykonaniem badania scenariusz został wyjaśniony osobom badanym. Następnie badani zostali poproszeni o wykonanie opisanych czynności. Maksymalny czas badania ustalono na 10 minut.

Do przeprowadzenia badania wykorzystano telefon marki Huawei P20 Lite ANE-LX1 z pamięcią wbudowaną 64 GB oraz pamięcią RAM 4 GB. System operacyjny to Android 9. Telefon wyposażony był w procesor Hisilicon Kirin 659, posiadający 8 rdzeni z taktowanie zegara 2,36 GHz.

Wydajność aplikacji została zmierzona za pomocą narzędzia Android Debug Bridge (ADB) oraz własnych skryptów stworzonych w języku programowania Python oraz języku powłoki Bash.

W momencie gdy badany był gotowy do wykonania scenariusza badawczego uruchamiany był jeden ze specjalnych skryptów powłoki Bash stworzonych odpowiednio dla szkieletu programistycznego React Native oraz Flutter. Skrypt (Listing 1) działał w następujący.

1. Sprawdzał czy od momentu uruchomienia upłynął czas 10 minut.
2. Jeśli warunek był fałszywy oczekiwał, aż użytkownik kliknie w dowolne miejsce na ekranie aplikacji.
3. W momencie wykrycia dotyku zbierał informacje o parametrach VIRT, SHR, RES, CPU, MEM oraz TIME.
4. Zebrane dane wysyłał do pliku z rozszerzeniem .csv.
5. Powtarzanie punktów 2 - 4 do momentu zakończenia badania, czyli zamknięcia aplikacji lub przekroczenia 10 minut od rozpoczęcia badania.
6. Wyliczenie średniej wartości zebranych parametrów oraz zapisanie wyników do pliku z rozszerzeniem .txt.
7. Uruchomienie skryptu napisanego w języku programowania Python.

Tabela 1: Scenariusz badawczy stworzony dla testowanych aplikacji

Lp.	Opis czynności	Oczekiwany wynik
1.	Wpisanie tekstu o długości 255 znaków.	Wpisano ciąg o długości 255 znaków.
2.	Skasowanie ciągu znaków z wykorzystaniem wirtualnej klawiatury.	Usunięto ciągu o długości 255 znaków.
3.	Wpisanie ciągu o długości 255 znaków.	Wpisano ciąg o długości 255 znaków.
4.	Skasowanie ciągu o długości 255 znaków z wykorzystaniem przycisku.	Usunięto ciąg o długości 255 znaków.
5.	25- krotne wciśnięcie przycisku “+”/	Stan licznika zmienił wartość z 0 na 25.
6.	25-krotne wciśnięcie przycisku “-”.	Stan licznika zmienił wartość z 25 na 0.
7.	Wciśnięcie przycisku przekierowującego do drugiego ekranu	Wyświetlany jest ekran z przewijaną listą elementów.
8.	Przewinięcie listy na sam dół.	Wyświetlany jest ostatni element z listy.
9.	Przewinięcie listy na samą górę.	Wyświetlany jest pierwszy element z listy.
10.	Wciśnięcie przycisku sortowanie.	Wyświetlana jest posortowana lista.
11.	Przewinięcie listy na sam dół.	Wyświetlany jest ostatni element z listy.
12.	Przewinięcie listy na samą górę.	Wyświetlany jest pierwszy element z listy.
13.	Wciśnięcie drugiego przycisku sortowania.	Wyświetlana jest posortowana lista w innej kolejności niż poprzednim razem.
14.	Przewinięcie listy na sam dół.	Wyświetlany jest ostatni element z posortowanej listy.
15.	Przewinięcie listy na samą górę	Wyświetlany jest pierwszy element z posortowanej listy.

Na Listingu 2 przedstawiono fragment kodu odpowiedzialnego za uruchomienie skryptu w języku programowania Python. Skrypt ten działał w następujący sposób, pobierał dane z pliku csv, w którym zapisywane były dane dla badanych parametrów: SHR, VIRT, RES, MEM oraz CPU. Następnie na ich podstawie tworzył wykres zależność ich wykorzystania w trakcie konkretnego badania. Fragment skryptu w języku Python przedstawiono na Listingu 3. Przedstawiono na nim ścieżkę z jakiej skrypt pobierał dane (w tym wypadku dla technologii Flutter) oraz w jaki sposób pobierał dane wykorzystywane na wykresie.

Listing 1: Fragment kodu skryptu w języku powłoki Bash sprawdzający kiedy należy zakończyć badanie wraz ze zbieraniem wartości

```
current_time=$(date +%s)

if [ $((current_time - start_time)) -ge 500 ] then
  break

fi

if  adb shell dumpsys window windows | grep -E
'mCurrentFocus|mFocusedApp' | grep
com.example.mgr_app_flutter >/dev/null  then

result=$(adb shell top -n 1 -d 1 | grep
com.example.mgr+ | awk '{
printf"%s,%s,%s,%s,%.2f,%s,%s,%s,%s,%.2f,%s,%s\
n",

$1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12 }')

if [ -n "$result" ]; then

echo $result >>
~/Desktop/wyniki/wyniki_flutter/dane_usrednione/dane
_szczegolowe_$1.txt

counter=$((counter+1))
```

Listing 2: Fragment kodu skryptu w języku powłoki Bash wywołujący w tle skrypt w języku programowania Python

```
#Uruchomienie skryptu w Pythonie w tle

nohup  python flutter.py user_$1.csv &

echo "Badanie zostało zakończone!!!!!!!!"
```

Listing 3: Fragmentu kodu w języku Python odpowiedzialny za wczytanie danych z pliku

```
path = '~/Desktop/wyniki/wyniki_flutter/dane_python/'

filename = path + sys.argv[1]

values = ["VIRT", "RES", "SHR", "CPU", "MEM",
"TIME"]

df = pd.read_csv(filename, header=None, names=values)

fig1, ax1 = plt.subplots()
```

4. Wyniki

Na Rysunku 5 oraz Rysunku 6 przedstawiono zrzuty ekranu prezentujące stworzoną aplikację z wykorzystaniem technologii Flutter oraz języka programowania Dart. Rysunek 7 oraz Rysunek 8 przedstawiają aplikację stworzoną w technologii React Native stworzoną przy użyciu języka programowania Java Script. W Tabeli 2 oraz w Tabeli 3 przedstawiono wyniki, które zebrano podczas prowadzonych badań.

W Tabeli 2 zaprezentowane dane dotyczą aplikacji mobilnej stworzonej z wykorzystaniem szkieletu programistycznego Flutter. Natomiast w Tabeli 3 znajdują się dane dotyczące aplikacji mobilnej stworzonej z wykorzystaniem szkieletu programistycznego React Native.

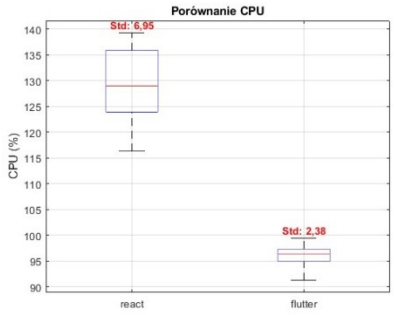
Każdy z wierszy przedstawia średnią wartość parametrów: VIRT, RES, SHR, CPU, MEM oraz czas trwania pojedynczego badania, który został zarejestrowany dla pojedynczego uczestnika badania. Na Rysunku 1 przedstawiono porównanie wykorzystania CPU dla obu technologii. Zaznaczono na nim także odchylenie standardowe dla badanego parametru, które wynosi odpowiednio około 7 % dla technologii React Native oraz około 2,4 % dla technologii Flutter. Na Rysunku 2 przedstawiono porównanie wykorzystania pamięci wirtualnej (parametr VIRT) dla technologii React Native oraz Flutter. Zmierzone odchylenie standardowe dla tego parametru jest bardzo bliskie 0 GB. Na Rysunku 3 zaprezentowano porównanie wykorzystania pamięci RAM (parametr RES) oraz pamięci współdzielonej (parametr SHR) dla technologii React Native i Flutter. Odchylenie standardowe dla parametru RES wynosi odpowiednio około 19 MB dla technologii React Native oraz około 35,5 MB dla technologii Flutter. Z kolei odchylenie standardowe dla parametru SHR dla technologii React Native wynosi około 9,6 MB, dla technologii Flutter jest to około 19,3 MB. Na Rysunku 4 przedstawiono porównanie całkowitego wykorzystania pamięci (parametr MEM). Odchylenie standardowe dla technologii React Native wynosi około 0,5 %. Natomiast dla technologii Flutter odchylenie standardowe wynosi około 0,9 %. Pomiędzy obiema badanymi technologiami największa różnica dotyczy wykorzystania przez system Android wirtualnej pamięci opisanej w tabelach jako parametr VIRT. W wartościach liczbowych to około 13 GB na korzyść technologii Flutter. Dane dotyczące wykorzystania pamięci fizycznej przez system Android prezentują się na zbliżonym poziomie i wynoszą około 5 %. Przy porównaniu wykorzystania czasu procesora przez aplikację korzystniej wypadła technologia Flutter. Średnia wartość zużycia wynosiła około 97 %. Technologia React Native wykorzystywała około 130 % czasu procesora (oznacza to zajmowanie więcej niż jednego rdzenia). Całkowity rozmiar pamięci współdzielonej przez aplikację dla szkieletu programistycznego Flutter oscylował w granicach od 50 MB do 120 MB. Dla aplikacji stworzonej z wykorzystaniem szkieletu programistycznego React Native wartość tej pamięci oscylowała w okolicy 75 MB. Wartość pamięci fizycznej określonej przez parametr RES dla aplikacji mobilnej stworzonej w technologii Flutter nie przekracza wartości 300 MB. Natomiast ten sam parametr dla technologii React Native nie przekracza wartości 170 MB.

Tabela 2: Wyniki wykorzystania zasobów VIRT, RES, SHR, CPU oraz MEM dla aplikacji mobilnej stworzonej w technologii Flutter

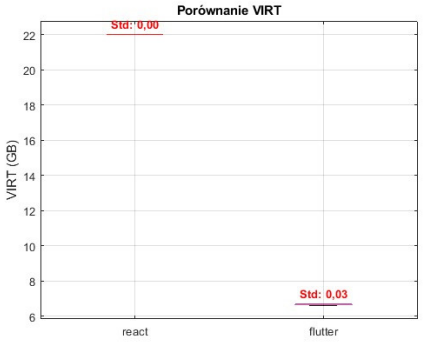
Numer badania	Średnia wartość VIRT [GB]	Średnia wartość RES [MB]	Średnia wartość SHR [MB]	Średnia wartość CPU [%]	Średnia wartość MEM [%]
1.	6,63	251	110	96,43	6,60
2.	6,62	268	113	96,65	7,04
3.	6,63	162	56	91,75	4,25
4.	6,65	159	56	95,19	4,15
5.	6,64	175	59	98,53	4,57
6.	6,67	157	58	97,49	4,12
7.	6,70	160	56	91,30	4,18
8.	6,70	160	56	96,29	4,18
9.	6,70	157	56	99,47	4,10
10.	6,70	179	58	97,42	4,68
11.	6,70	181	60	97,20	4,74
12.	6,70	185	58	94,82	4,84
13.	6,70	152	54	95,91	3,97
14.	6,70	148	55	96,50	3,87
15.	6,70	157	57	92,43	4,10

Tabela 3: Wyniki wykorzystania zasobów VIRT, RES, SHR, CPU oraz MEM dla aplikacji mobilnej stworzonej w technologii React Native

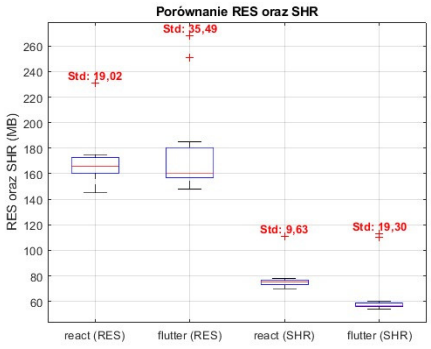
Numer badania	Średnia wartość VIRT [GB]	Średnia wartość RES [MB]	Średnia wartość SHR [MB]	Średnia wartość CPU [%]	Średnia wartość MEM [%]
1.	6,63	251	110	96,43	6,60
2.	6,62	268	113	96,65	7,04
3.	6,63	162	56	91,75	4,25
4.	6,65	159	56	95,19	4,15
5.	6,64	175	59	98,53	4,57
6.	6,67	157	58	97,49	4,12
7.	6,70	160	56	91,30	4,18
8.	6,70	160	56	96,29	4,18
9.	6,70	157	56	99,47	4,10
10.	6,70	179	58	97,42	4,68
11.	6,70	181	60	97,20	4,74
12.	6,70	185	58	94,82	4,84
13.	6,70	152	54	95,91	3,97
14.	6,70	148	55	96,50	3,87
15.	6,70	157	57	92,43	4,10



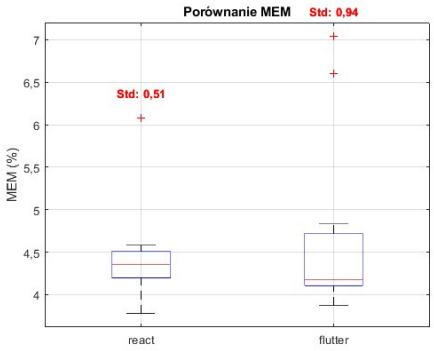
Rysunek 1: Porównanie wykorzystania parametru CPU dla technologii React Native oraz Flutter.



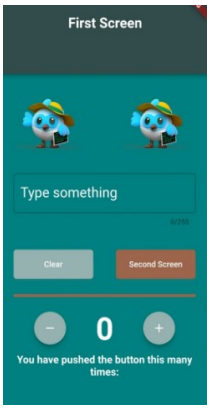
Rysunek 2: Porównanie wykorzystania pamięci wirtualnej (parametr VIRT) dla technologii React Native oraz Flutter.



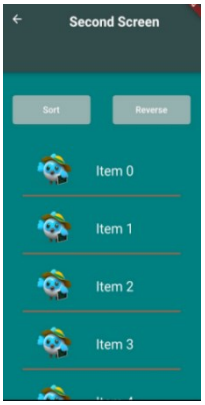
Rysunek 3: Porównanie wykorzystania pamięci RAM (parametr RES) oraz pamięci współdzielonej (parametr SHR) dla technologii React Native oraz Flutter.



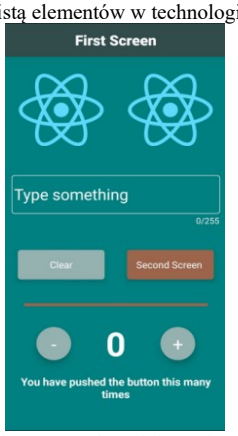
Rysunek 4: Porównanie całkowitego wykorzystania pamięci (parametr MEM) dla technologii React Native oraz Flutter.



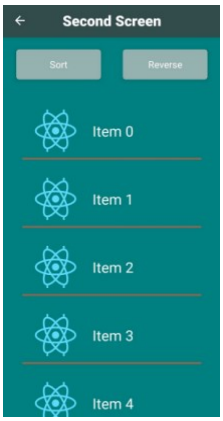
Rysunek 5: Zrzut ekranu przedstawiający główny ekran aplikacji w technologii Flutter.



Rysunek 6: Zrzut ekranu przedstawiający ekran aplikacji z przewijaną listą elementów w technologii Flutter.



Rysunek 7: Zrzut ekranu przedstawiający ekran główny aplikacji stworzony w technologii React Native.



Rysunek 8: Zrzut ekranu przedstawiający ekran aplikacji z przewijaną listą elementów w technologii React Native act Native.

5. Wnioski

Celem artykułu było porównanie wydajności aplikacji mobilnych tworzonych z wykorzystaniem technologii React Native oraz Flutter. W postawionej hipotezie badawczej założono, że efektywniejsze aplikacje mobilne tworzone są w technologii Flutter niż w technologii React Native. Po przeprowadzeniu badań i porównaniu badanych parametrów wyniki prezentują się następująco. Dla parametru MEM, dotyczącego ogólnego wykorzystania pamięci przez aplikacje obie technologie wypadły podobnie. Średnie zużycie wynosi około 4,5 %. Dla parametru VIRT, który odnosi się do całkowitej ilości pamięci wirtualnej przez aplikację lepiej wypadła technologia Flutter. Wykorzystuje ona tylko około 7 GB pamięci wirtualnej telefonu. Natomiast technologia React Native wykorzystuje 22 GB tej pamięci. Porównując procentowy udział czasu procesora przeznaczony na wykonanie kodu aplikacji również lepiej wypadał technologia Flutter. Po porównaniu wyników dotyczących ilości pamięci współdzielonej przez aplikację korzystniej wypadła technologia React Native. Również dla parametru określającego rozmiar pamięci RAM zajmowany przez aplikację lepiej wypadł szkielet programistyczny React Native. Jednak różnice między dwoma wyżej wymienionymi parametrami RES oraz SHR nie są duże i wynoszą około 20 MB. Dla obu parametrów SHR i RES najprawdopodobniej związane jest to z większą aktywnością innej aplikacji zainstalowanej na telefonie w momencie przeprowadzania badania z technologią Flutter. Jednak aby dokładniej potwierdzić ten scenariusz należałoby zwiększyć liczbę pobranych próbek badawczych.

Podsumowując po porównaniu pięciu parametrów: VIRT, SHR, RES, MEM oraz CPU nie da się jednoznacznie potwierdzić postawionej hipotezy badawczej świadczącej o lepszej wydajności technologii Flutter niż technologii React Native. W tym celu należałoby wykonać bardziej szczegółowe badania. Przykładowo można porównać inne czynniki m.in. interakcję technologię z zewnętrzną bazą danych, czas przetwarzań żądań HTTP czy wiele więcej inny danych.

Literatura

[1] L.P. Barros, F. Medeiros, E. Moraes, A. F. Júnior, Analyzing the Performance of Apps Developed by using Cross-Platform and Native Technologies, SEKE (2020) 186-191.

[2] J. Jagiełło, Performance comparison between React Native and Flutter, Bachelor thesis Umeå University, Umeå, 2019.

[3] H. Hussain, K. Khan, F. Farooqui, Q. A. Arain, I. F. Siddiqui, Comparative Study of Android Native and Flutter App Development, Memory 47 (2021) 36-37.

[4] E. Gucuoglu, A. B. Ustun, N. Seyhan, Comparison of Flutter and React Native Platforms, Journal of Internet Applications and Management 12 (2) (2021) 129-143, https://doi.org/10.34231/iuyd.888243.

[5] M. Olsson, A comparison of Performance and Looks Between Flutter and Native Applications: When to prefer Flutter over native in mobile application development, Bachelor thesis Blekinge Institute of Technology, Karlskrona, 2020.

[6] N. A. Shevtsiv, A. M. Striuk, Cross platform development vs native development, CEUR Workshop Proceedings, 2021.

[7] Opis komendy top w systemie operacyjnym Linux, https://www.man7.org/linux/manpages/man1/top.1.html, [01.06.2022].

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 352–359
Received: 27 June 2023
Accepted: 5 July 2023

Performance analysis of REST API technologies using Spring and Express.js examples

Analiza wydajności technologii tworzenia REST API na przykładzie Spring i Express.js

Maciej Wicha*, Beata Pańczyk

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The purpose of this article is a comparative analysis of two technologies for building applications in REST architecture. A Java-based development framework - Spring, and a framework designed for JavaScript language and Node environment - Express.js were analyzed. The test application was designed and implemented in both studied technologies. Using the Apache JMeter tool, HTTP request processing times were measured by operating on simple text data. The experiment was based on 5 scenarios repeated for a different number of users in the range of 10 to 100, with a constant number of executed requests to the server. The analysis conducted showed that the application implemented in Express.js handles HTTP requests up to 249% more efficiently than its counterpart in Spring.

Keywords: Spring; Express.js; REST API; performance benchmarking

Streszczenie

Tematem niniejszego artykułu jest analiza porównawcza dwóch technologii do budowania aplikacji w architekturze REST. Badania dotyczą opartego na języku Java szkieletu programistycznego - Spring oraz szkieletu przeznaczonego dla języka JavaScript i środowiska Node – Express.js. Aplikację testową zaimplementowano w obu badanych technologiach. Przy wykorzystaniu narzędzia Apache JMeter dokonano pomiaru czasów przetwarzania żądań HTTP operując na prostych danych tekstowych. Eksperyment opierał się na 5 scenariuszach powtórzonych dla różnej liczby użytkowni (od 10 do 100), przy stałej częstotliwości wykonywanych zapytań do serwera. Przeprowadzone analizy pozwoliły określić, że aplikacja zaimplementowana w Express.js obsługuje żądania HTTP nawet o 249% sprawniej niż jej odpowiednik w Spring.

Słowa kluczowe: Spring; Express.js; REST API; analiza wydajności

*Corresponding author

Email address: **maciejwicha8@gmail.com** (M. Wicha)

1. Wstęp

Pojawienie się smartfonów łączących funkcje telefonu komórkowego i komputera zrewolucjonizowały wiele aspektów naszego życia. Standardem stało się projektowanie trzech wariantów aplikacji – dostępnej dla urządzeń z systemem Android, iOS oraz w klasycznej postaci WEB.

W przypadku aplikacji działających w trybie online, pojawia się problem pobierania danych, tak aby na wszystkich trzech platformach dostępne były te same treści. Naturalnym zabiegiem zdaje się być zaimplementowanie jednej centralnej aplikacji udostępniającej dane do wszystkich aplikacji klienckich.

Nieodłącznym zjawiskiem w wytwarzaniu różnego rodzaju aplikacji jest integracja z istniejącymi już systemami lub uwzględnienie jej w przyszłości. Twórcy portali społecznościowych udostępniają API swoich aplikacji, aby inni mogli z niego korzystać poszerzając standardowe funkcjonalności macierzystej aplikacji. Odpowiednie usługi pozwalające np. udostępniać posty za pomocą odpowiednich punktów końcowych udostępniają autorzy zarówno Facebooka [1] jak i Twittera [2].

Aplikacje serwerowe w technologii REST rozwiązują wszystkie powyższe kwestie. Jedna aplikacja serwerowa jest w stanie obsłużyć wiele aplikacji klienckich, bez względu na przeznaczenie, środowisko czy funkcjonalności.

Wśród wielu dostępnych technologii, istnieje jeszcze więcej szkieletów programistycznych ułatwiających pracę nad tego typu aplikacjami. Ich wybór nie należy do najprostszych i uzależniony jest od wielu czynników, takich jak stopień skomplikowania projektu czy jego przeznaczenie. Oprócz szeregu funkcji, rozbudowanej społeczności dzielącej się rozwiązaniami i wspólnie pomagającym wyeliminować pojawiające się błędy, każda technologia posiada również swoje wady.

Celem niniejszej pracy jest analiza porównawcza dwóch popularnych technologii tworzenia aplikacji internetowych w architekturze REST – Spring i Express, pod kątem popularności oraz czasów przetwarzania żądań HTTP odpowiadającym podstawowym operacjom CRUD.

W artykule postawiono hipotezę badawczą, że technologia Express zapewnia wydajniejsze rozwiązanie do tworzenia REST API niż Spring.

2. Przegląd literatury

Marcin Grudniak wraz z Mariuszem Dzieńkowskim w artykule *Porównanie wydajności aplikacji internetowych REST API opartych na szkieletach programistycznych JavaScript* opublikowanym na łamach JCSI poddali analizie porównawczej dwa szkielety programistyczne Express.js i Hapi [3]. Autorzy przygotowali aplikację testową, a następnie opracowali 4 scenariusze testowe, w ramach których poddano badaniom 4 podstawowe typy operacji HTTP (POST, PUT, GET, DELETE). Każdy ze scenariuszy operował na innym typie danych, w eksperymencie uwzględniono łańcuchy znaków, tablicę 100 elementów zawierających 100 znakowe losowe ciągi znaków, obiekt o 100 atrybutach oraz 50 elementowej tablicy zawierającej 50 atrybutów każdy po 50 losowych znaków. Aplikacja utrwalała dane w nierelacyjnej bazie danych MongoDB. Czasy przetwarzania żądań zostały zebrane za pomocą modułu wewnątrz-serwerowego oraz za pomocą aplikacji zewnętrznej rejestrującej czas odpowiedzi. Na podstawie przeprowadzonych badań autorzy doszli do wniosku, że aplikacja zbudowana w oparciu o szkielet Express.js osiągnęła lepsze wyniki niż jej odpowiednik w Hapi. Jedynie dla operacji na tablicy obiektów, czasy obu technologii były do siebie zbliżone.

Celem artykułu *A performance comparision of RESTful Applications Implemented in Spring Boot Java and MS.NET Core* autorstwa Hardeep Kaur Dhalla jest porównanie wydajności aplikacji typu REST przy użyciu dwóch różnych szkieletów programistycznych opierających się na języku Java i C# [4]. Obie aplikacje były implementacją tych samych procesów biznesowych. Do zebrania wyników autor wykorzystał narzędzie Apache JMeter w wersji 5.2.1. W trakcie przeprowadzonego przeglądu literatury autor natknął się na badania wskazujące lepszą wydajność aplikacji opartej o Java EE Struts niż VB.NET. Jako kryterium oceny wydajności aplikacji autor obrał średni czas odpowiedzi oraz procentowy błąd przy obsłudze żądania. Dodatkowo monitorowano również zużycia zasobów takich jak CPU czy pamięć RAM. Scenariusze testowe zakładały wzrost liczby aktywnych użytkowników od 1000 do 6400. Uzyskane wyniki pozwoliły stwierdzić, że aplikacja zaimplementowana w MS.NET Core zapewnia krótsze czasy odpowiedzi, oraz zużywa mniej zasobów takich jak CPU czy RAM w stosunku do aplikacji utworzonej przy użyciu Java Spring Boot.

W artykule *Performance Comparision of Java EE and ASP.NET Core Technologies for Web API Development* opublikowanej w Applied Computer Systems przez Kristians Kronis i Marina Uhanova poruszono analizę porównawczą szkieletów ASP.NET Core i Java EE w wersji 7 [5]. Autorzy zwrócili uwagę na istotność niezmienności środowiska testowego przez cały okres przeprowadzania eksperymentu, dlatego aplikacje zostały wdrożone na serwer VPS (ang. Virtual Private Server), który przez cały okres prowadzenia pomiarów utrzymywał ten sam stan (aktualizacje, wersja oprogramowania, zainstalowane programy). Jako miejsce przechowywania danych przez aplikacje testowe wybrano bazę danych MySQL, a dane przesyłane za pomocą API miały format JSON. Scenariusze badawcze dotyczyły m.in. wysyłania i odbierania krótkich ciągów tekstowych, odbierania danych o zadanym rozmiarze i generowaniu liczb pseudolosowych, operacji arytmetycznych na 1000 losowo wybranych liczbach oraz operacji na rozbudowanych plikach JSON. Analiza uzyskanych wyników pozwoliła określić, że o ile ASP.NET Core uzyskał krótsze czasy przetwarzania żądań to sam serwer Kestrel potrzebował więcej czasu na wysłanie odpowiedzi niż Tomcat.

Performance Comparision and Evaluation of Web Development Technologies in PHP, Python and Node.js autorstwa Kai Lei, Yinng Ma oraz Zhi Tan w kompleksowy sposób porównuje trzy różne technologie tworzenia aplikacji internetowych [6]. Jako narzędzie pomiarowe wykorzystali oni dwa programy ApacheBench oraz LoadRunner dla testów obciążeniowych. Testy przeprowadzono dla 10, 100, 200, 500 i 1 000 użytkowników. Materiałem badawczym były trzy proste aplikacje: klasyczny program wyświetlający „Hello World!", program wyznaczający *n*-ty wyraz ciągu Fibbonacciego (autorzy wybrali 10, 20 i 30 wyraz) oraz program dokonujący operacji *select* na bazie danych MySQL. Uzyskane wyniki wskazały, że aplikacja zaimplementowana za pomocą PHP i Python obsługuje znacznie mniej żądań niż Node.js w określonym czasie. Dodatkowo autorzy pokusili się o wskazanie przeznaczenia każdej z technologii, dla aplikacji napisanej w języku Python zaproponowano duże rozbudowane aplikacje biznesowe, podczas gdy dla języka PHP małe i średnie przedsiębiorstwa. Technologię Node.js polecają natomiast dla stron wykorzystujących intensywnie operacje wejścia-wyjścia.

3. Badane technologie

Analizie porównawczej poddano dwa popularne szkielety programistyczne Java Spring (z konfiguracją Spring Boot) oraz Express.js.

3.1. Szkielet programistyczny Spring

Spring framework to od lat jeden z najpopularniejszych szkieletów programistycznych do tworzenia aplikacji w języku Java [7].

Jednym z popularnych modułów Spring jest Spring Boot, który w prosty sposób pozwala skonfigurować i wdrożyć aplikację. Spring Initializr pozwala wygenerować początkowy szablon aplikacji Spring wraz z dodatkowymi zależnościami.

Wśród wielu cech i zalet szkieletu Spring wymienić można między innymi:

- Wstrzykiwanie zależności (ang. Dependency Injection – DI) – jest to mechanizm wyręczający programistę w ręcznym tworzeniu obiektów. Szkielet automatycznie wstrzykuje odpowiednie zależności.
- Model MVC (ang. Model-View-Controler) – obsługa wzorca projektowego pozwalającego na budowę aplikacji internetowych, zapewniającego gotowe komponenty do obsługi żądań HTTP, obsługę for-

mularzy, zarządzanie sesją użytkownika i generowanie widoków.
- Modularność – aplikacje Spring można zintegrować z wieloma narzędziami i bibliotekami, pozwalającymi na szybsze i sprawniejsze programowanie, np. Spring Security (konfiguracja zabezpieczeń aplikacji, autoryzacji), Spring Data (zarządzanie danymi w aplikacji), Thymeleaf (silnik szablonów HTML) [8].

3.2. Szkielet programistyczny Express.js

Express.js to minimalistyczny szkielet programistyczny języka JavaScript przeznaczony na platformę Node.js zyskujący popularność w ostatnich latach [9]. W swoim założeniu jest prostym i elastycznym narzędziem do tworzenia aplikacji internetowych, zapewniając prostszy sposób obsługi żądań HTTP niż sam Node.js. Podobnie jak Spring, udostępnia narzędzie do generowania podstawowego szablonu aplikacji. Razem z Angular.js, Node.js oraz MongoDB należy do tzw. MEAN Software Stack, czyli zestawu technologii i narzędzi napisanych w języku JavaScript pozwalających na kompleksowe pisanie aplikacji internetowych [10].

Wśród zalet wykorzystania szkieletu Express.js wymienić można:
- Trasowanie (ang. routing) – intuicyjny system routingu, pozwalający programistom prosty sposób definiować ścieżki URL i ich obsługę poprzez specjalne funkcje (tzw. middleweare).
- Funkcje pośredniczące (ang. middleware) – funkcje wywoływane sekwencyjnie podczas przetwarzania żądań HTTP przesyłanych do serwera aplikacji. Odpowiadają za obsługę żądań, wysyłania odpowiednich wiadomości zwrotnych w postaci danych lub zakończenia przetwarzania żądań.
- Model MVC (ang. Model-View-Controler, model-widok-kontroler) – podobnie jak Spring, zapewnia obsługę wzorca projektowego MVC ułatwiającego rozwój aplikacji i zarządzanie kodem aplikacji.
- Prostota i elastyczność – w przeciwieństwie do Spring, aplikacja zaimplementowana z wykorzystaniem szkieletu Express nie posiada narzuconej struktury projektu. Pomimo, że w dokumentacji znajduje się zalecana struktura projektu to cały kod aplikacji może znajdować się w jednym pliku – twórcy nie narzucają w tej kwestii żadnych ograniczeń, a większość standardów wyznacza społeczność [10].

3.3. Popularność technologii w latach 2019-2023

Zarówno Express.js, jak i Spring zdobyły przez lata grono swoich zwolenników i przeciwników. Wyniki corocznej ankiety „Developer Survey" przeprowadzanej przez Stack Overflow widoczne w Tabeli 1. wskazują, że popularność obu technologii jest do siebie zbliżona. W latach 2019-2023 Express.js jest nieznacznie bardziej popularny wśród profesjonalnych programistów pracujących na pełnym etacie niż jego odpowiednik Spring.

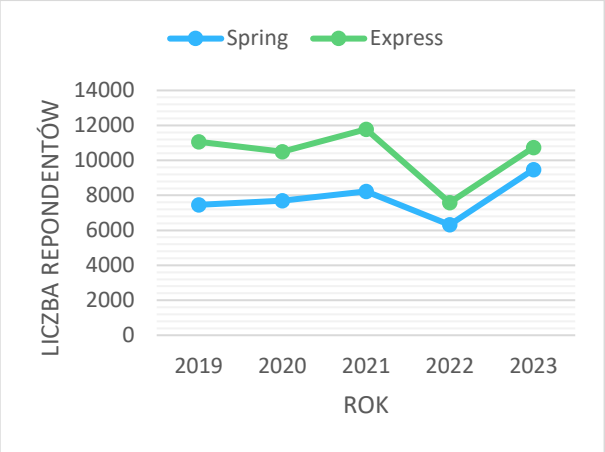
W przypadku Spring zauważyć można, że w latach 2019-2021 liczba respondentów rosła (Rysunek 1), gdy popularność Express zachowywała się niemonotonicznie – w roku 2020 spadła z 11 070 głosów na 10 504, by w kolejnym roku zanotować wzrost o 12%.

Tabela 1: Popularność Spring i Express - wyniki ankiety przeprowadzonej przez Stack Overflow w latach 2019-2023 [7-9, 11-13]

ROK	Spring	Express
2019	7 463	11 070
2020	7 695	10 504
2021	8 226	11 780
2022	6 315	7 585
2023	9 474	10 740

W roku 2022 obie technologie straciły na popularności osiągając najniższe wyniki w badanym okresie (Spring - 6 315 głosów, Express – 7 585). Dla Express był to spadek o 36% w stosunku do roku poprzedniego.

Pomimo tego, obie technologie nadal znajdują się w czołówce popularności szkieletów programistycznych przeznaczonych dla aplikacji internetowych.



Rysunek 1: Popularność badanych szkieletów programistycznych w latach 2019-2023 na podstawie ankiety przeprowadzonej przez Stack Overflow [7-9, 11-13].

4. Metoda badań

Zebranie materiału badawczego polegało na przeprowadzeniu eksperymentu, w ramach którego testowano odpowiedniki operacji CRUD dla protokołu HTTP (POST, GET, PUT, DELETE) dla zaimplementowanych aplikacji. Następnie wykorzystując statystykę opisową przeanalizowano zebrane wyniki.

4.1. Scenariusze badawcze

Przy użyciu narzędzia Apache JMeter zdefiniowano 5 scenariuszy testowych, które pozwoliły zbadać czasy odpowiedzi aplikacji i przetwarzania żądań HTTP.

Ogólnymi założeniami dla wszystkich testów były:
- stała liczba zapytań wysyłana do serwera REST (1000 zapytań na minutę);
- każdy scenariusz powtarzano 10 krotnie, dla różnej liczby użytkowników (od 10 do 100);
- każdy z użytkowników próbuje jednocześnie odczytać swoje dane;
- użytkownik jest zautoryzowany i ma wygenerowany token JWT;

- użytkownik próbuje odczytać tylko dane, do których ma dostęp.

Poszczególne scenariusze zostały zdefiniowane następująco:

1. *Scenariusz I – metoda POST* – polegający na tworzeniu przez użytkownika 100 rekordów, przekazując pełen obiekt zadania. Przy każdym żądaniu zmianie ulegała data ukończenia zadania ustawiana na aktualną datę.
2. *Scenariusz II – metoda GET dla pojedynczego elementu* – użytkownik pobierał pojedyncze zadanie z bazy danych, operację powtarzał dla każdego ze 100 zadań.
3. *Scenariusz III – metoda GET dla 100 elementów* – użytkownik pobierał wszystkie dodane zadania 100 krotnie.
4. *Scenariusz IV – metoda PUT* – użytkownik aktualizował wartość kolumny „isCompleted" z wartości false na true. Operację wykonywał dla każdego ze 100 zadań. W ciele zapytania przekazywano tylko aktualizowaną wartość.
5. *Scenariusz V – metoda DELETE* – użytkownik usuwał każde ze 100 zadań.

4.2. Platforma testowa

W Tabeli 2. przedstawiono konfigurację środowiska testowego, na którym dokonywane były pomiary, oraz na których uruchomione były wszystkie 3 usługi – aplikacja kliencka (scenariusz w narzędziu Apache JMeter), aplikacja serwerowa (aplikacja „to-do" w technologii Express lub Spring) oraz baza danych (MariaDB za pośrednictwem narzędzia XAMPP).

Tabela 2: Środowisko testowe

NPM	9.5.1
JDK	17
Procesor	Intel Core i7-9750H (2.60 GHz, 6 rdzeni, 12 wątków)
RAM	2x16GB (DDR4, 2667 MHz)
Dysk #1	SSD 1TB 2.5" SATA
Dysk #2	SSD 512 GB M.2 PCIe
System operacyjny	Windows 10 Pro

4.3. Narzędzie Apache JMeter

Apache JMeter jest rozbudowanym narzędziem typu open-source wyprodukowanym przez Apache Software Foundation, pozwalającym na wykonywanie testów wydajności różnych usług, w tym aplikacji internetowych. Do jego zalet zaliczyć można:

- symulowanie dostępu do aplikacji przez wielu użytkowników jednocześnie;
- pomiar dynamicznych i statycznych parametrów (np. czas odpowiedzi serwera, jego obciążenie lub przepustowość sieci, itp.);
- testowanie wielu protokołów i usług (m.in. HTTP, HTTPS, REST, SOAP, SMTP, itp.);
- samodzielne budowanie scenariuszy testowych przez użytkownika;
- tworzenie scenariuszy za pomocą interfejsu graficznego lub poprzez specjalny plik;
- dostęp do wielu wtyczek (oficjalnych i tworzonych przez społeczność) rozszerzających podstawowe funkcjonalności narzędzia.

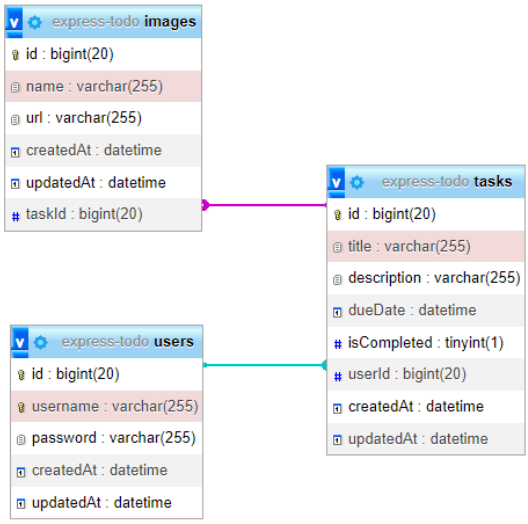
5. Materiał badawczy

Materiałem badawczym była prosta aplikacja w architekturze REST zaimplementowana w dwóch wybranych technologiach w następujących wersjach – Spring 3.0.4 oraz Express 4.16.1.

5.1. Aplikacja testowa

Zaimplementowano aplikację „to-do" o strukturze bazy danych widocznej na Rysunku 2. Dla każdej z testowanej aplikacji baza danych tworzona była przez badany szkielet programistyczny przy użyciu dodatkowych modułów (np. dla Express.js – Sequalizer, dla Spring – JPA).

Aplikacja w swoim założeniu pozwala na rejestrację i logowanie użytkownika, oraz dodawanie i zarządzanie zadaniami. Odpowiednie punkty końcowe wymagają autoryzacji. Wszystkie wymagania funkcjonalne i niefunkcjonalne zawarte zostały w kolejnych podrozdziałach.



Rysunek 2: Struktura bazy danych wygenerowana przez moduł Sequelizer (Express.js).

5.2. Wymagania funkcjonalne

Określenie wymagań funkcjonalnych było istotnym elementem planowania eksperymentu w celu zachowania spójności pomiędzy obiema wersjami aplikacji testowej.

1. Użytkownik
   - 1.1. Logowanie
     - 1.1.1. Weryfikacja poprawności danych
     - 1.1.2. Wygenerowanie JWT
   - 1.2. Rejestracja
     - 1.2.1. Weryfikacja poprawności danych
     - 1.2.2. Weryfikacja unikalności loginu
     - 1.2.3. Rejestracja użytkownika

2. Zadanie
   2.1. Dodanie zadania
      2.1.1. Uzupełnienie tytułu
      2.1.2. Uzupełnienie opisu
      2.1.3. Uzupełnienie daty wykonania zadania
      2.1.4. Uzupełnienie statusu zadania
      2.1.5. Utrwalenie zadania w bazie danych
   2.2. Edycja zadania
      2.2.1. Weryfikacja identyfikatora zadania
      2.2.2. Zmiana tytułu
      2.2.3. Zmiana opisu
      2.2.4. Zmiana daty wykonania zadania
      2.2.5. Zmiana statusu zadania
      2.2.6. Utrwalenie zmian w bazie danych
   2.3. Pobranie zadania
      2.3.1. Wskazanie zadania do wyświetlenie
      2.3.2. Weryfikacja identyfikatora zadania
      2.3.3. Zwrócenie danych w postaci JSON
   2.4. Pobranie wszystkich zadań
      2.4.1. Zwrócenie danych w postaci JSON
   2.5. Usunięcie zadania
      2.5.1. Wskazanie zadania do usunięcia
      2.5.2. Weryfikacja identyfikatora zadania
      2.5.3. Zwrócenie komunikatu o statusie

5.3. Wymagania niefunkcjonalne

Oprócz funkcjonalności zdefiniowano również ograniczenia aplikacji określające architekturę i format przesyłanych danych.

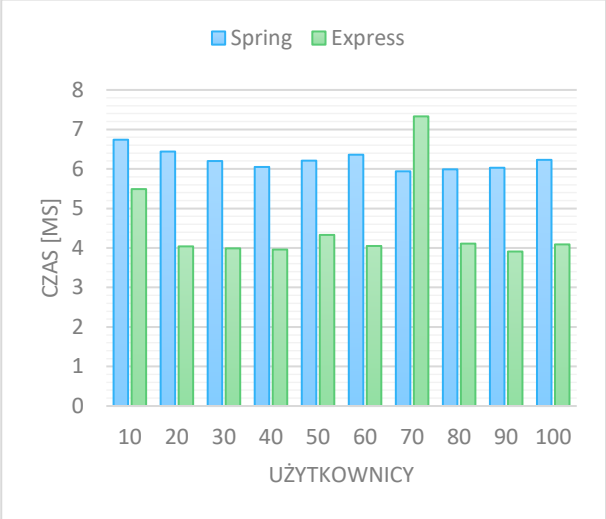
Dostępność systemu:

a) serwis dostępny jest jako aplikacja REST,
b) każdej funkcjonalności odpowiadają odpowiednie punkty końcowe (ang. endpoints),
c) komunikaty zwracane przez aplikację mają postać zgodną z formatem JSON.

6. Wyniki badań

6.1. Scenariusz I – metoda POST

Na Rysunku 3 przedstawiono wyniki, będące średnią arytmetyczną uzyskanych wyników dla poszczególnych iteracji testu odpowiednio od 10 do 100 użytkowników.



Rysunek 3: Średnie czasy przetwarzania żądania POST.

W przeważającej większości Express.js uzyskuje niższe czasy przetwarzania żądania POST odpowiedzialnego za utworzenie nowego rekordu w bazie danych. Jedynie dla 70 użytkowników potrzebuje 23% więcej czasu niż Spring.

Tabela 3 przedstawia szczegółowe informacje na temat wybranych parametrów dla 50 użytkowników. Pomimo, że maksymalny czas, który Express.js potrzebuje na przetworzenie żądania wynosi 43,00 ms, a Spring jedynie 25,00 to zarówno średnia jak i mediana jest niższa dla Express i wynosi odpowiednio 4,33 ms i 4,00 ms.

Tabela 3: Wartości parametrów dla metody POST – 50 użytkowników

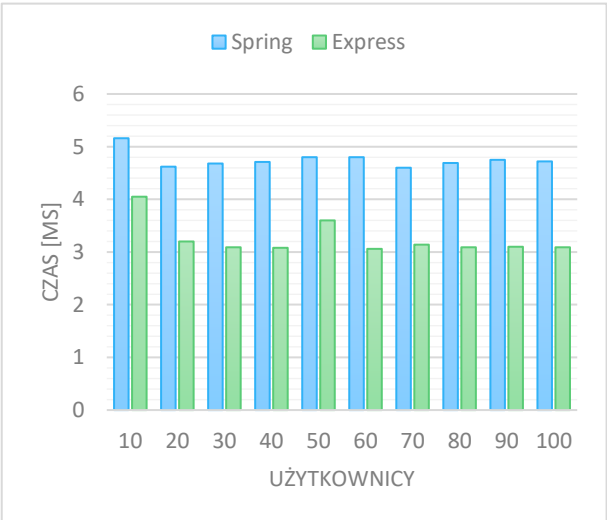
Parametr	Spring	Express
Średnia (ms)	6,21	4,33
Wartość maksymalna (ms)	25,00	43,00
Wartość minimalna (ms)	4,00	3,00
Odchylenie standardowe (ms)	1,04	1,55
Mediana (ms)	6,00	4,00

6.2. Scenariusz II – metoda GET dla pojedynczego obiektu

Średnie czasy przetwarzania żądania z metodą GET dla pojedynczego obiektu wskazanego poprzez parametr adresu URL przekazywanego do aplikacji testowej widoczne są na Rysunku 4.

Podobnie jak w przypadku tworzenia nowego rekordu, również i przy odczycie danych Spring radzi sobie gorzej od Express. Obie technologie potrzebują jednak najwięcej czasu w przypadku obsługi 10 użytkowników (Spring – 5,16 ms, Express – 4,05 ms).

Najkrótszy średni czas obsługi żądania dla Express występował dla 60 użytkowników (3,06 ms) oraz 70 użytkowników dla Spring (4,6 ms).



Rysunek 4: Średnie czasy przetwarzania żądania GET dla pojedynczego elementu.

W Tabeli 4 porównano statystyki dla 50 użytkowników. W tym przypadku dla obu technologii mediana jest sobie równa i wynosi 5,00 ms. Podobnie jak w przypadku poprzedniego scenariusza najdłuższy zarejestrowany czas przetwarzania żądania należy do Express (19 ms). Średni czas odpowiedzi wynosi 4,80 ms dla Spring oraz 3,60 dla Express.

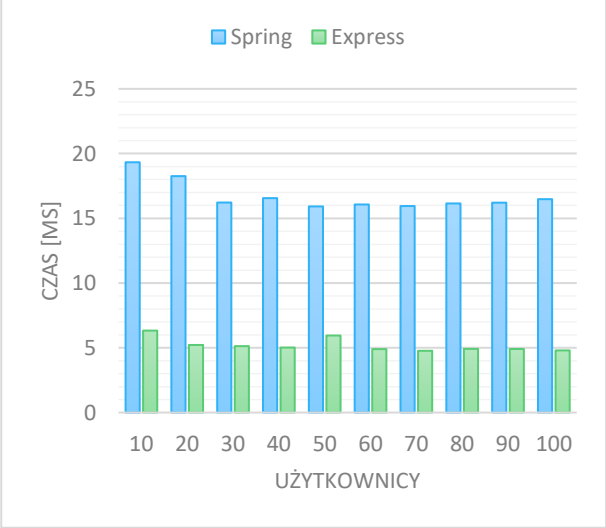
Tabela 4: Wartości parametrów dla metody GET dla pojedynczego elementu – 50 użytkowników

Parametr	Spring	Express
Średnia (ms)	4,80	3,60
Wartość maksymalna (ms)	9,00	19,00
Wartość minimalna (ms)	3,00	2,00
Odchylenie standardowe (ms)	0,74	1,23
Mediana (ms)	5,00	5,00

6.3. Scenariusz III – metoda GET dla 100 obiektów

Znaczącą przewagę w czasie odpowiedzi można zauważyć podczas analizy wyników trzeciego scenariusza, pobierającego 100 obiektów dodanych przez zalogowanego użytkownika (Rysunek 5).

Spring potrzebuje ponad 15 ms na odesłanie danych w czasie gdy Express około 5 ms.



Rysunek 5: Średnie czasy przetwarzania żądania GET dla 100 elementów.

Podobnie jak w dwóch poprzednich scenariuszach, tak i w tym przypadku najdłuższe czasy osiągane są dla 10 użytkowników (Spring – 19,33 ms, Express – 6,33 ms). Analizując czasy dla szkieletu JavaScript mają one tendencję malejącą do 50 użytkowników gdzie widoczny jest wzrost o 18% w porównaniu do 40 użytkowników. Następnie dla 60. i 70. użytkowników czas ponownie maleje. W przypadku Spring, czasy maleją do 30 użytkowników, następnie dla 40 użytkowników czas wzrasta o 2% i znowu maleje. Dla kolejnych użytkowników widać niewielką tendencję wzrostową.

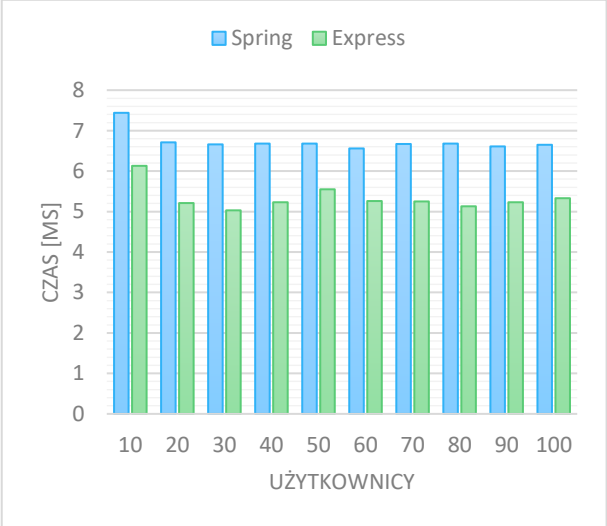
Tabela 5: Wartości parametrów dla metody GET dla 100 elementów – 50 użytkowników

Parametr	Spring	Express
Średnia (ms)	15,92	5,95
Wartość maksymalna (ms)	29,00	25,00
Wartość minimalna (ms)	11,00	4,00
Odchylenie standardowe (ms)	1,69	2,04
Mediana (ms)	16,00	3,00

Zarówno średnia, wartość maksymalna, minimalna oraz mediana największe są dla aplikacji zaimplementowanej w Spring. Potrzebuje ona co najmniej 11 ms na przetworzenie pojedynczego żądania, podczas gdy Express tylko 4 ms (Tabela 5).

6.4. Scenariusz IV – metoda PUT

Średnie czasy uzyskane w trakcie czwartego scenariusza zostały zwizualizowane na Rysunku 6. Najszybciej zostały przetworzone żądania w przypadku 30 użytkowników dla szkieletu Express (5,03 ms) oraz 60 użytkowników dla aplikacji w Spring (6,56 ms). Najdłuższe czasy uzyskane zostały dla 10 użytkowników (Spring – 7,44 ms, Express – 6,13 ms).



Rysunek 6: Średnie czasy przetwarzania żądania PUT.

W zakresie od 20 do 100 użytkowników, czasy uzyskane w aplikacji utworzonej przy użyciu Spring oscylują w okolicy 6,65 ms, osiągając wyniki delikatnie poniżej lub powyżej tej wartości (w granicach około 1%).

W przypadku Express widoczny jest wzrost czasu potrzebnego na przetworzenie żądania w zakresie od 20 do 50 użytkowników. W kolejnych iteracjach następuje skrócenie czasu z 5,55 ms (dla 50 użytkowników) do 5,13 ms (dla 80 użytkowników). Dla dwóch ostatnich powtórzeń scenariusza czas wzrasta odpowiednio do 5,23 ms i 5,33 ms.

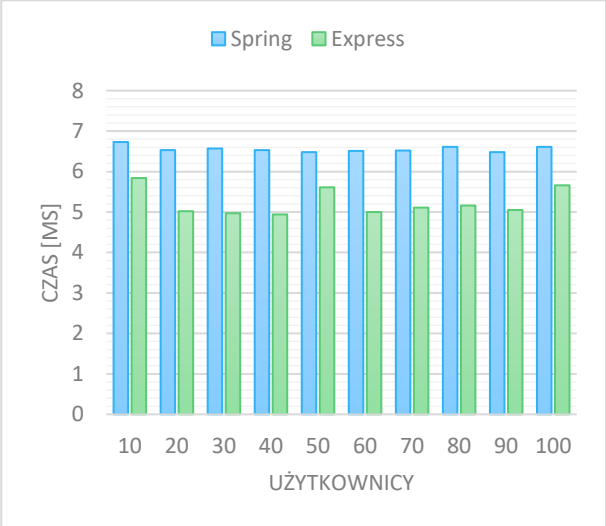
Najdłuższy jednostkowy czas potrzebny na przetworzenie żądania wynosił 22,00 ms, najszybciej zaś odpowiedzi udzielił Express i potrzebował na to 2,0 ms. Mediana w obu przypadkach jest zbliżona do średniej i wynosi – 7,00 ms i 6,68 ms dla Spring oraz 5,00 ms i 5,55 ms dla drugiego szkieletu (Tabela 6).

Tabela 6: Parametry dla metody PUT – 50 użytkowników

Parametr	Spring	Express
Średnia (ms)	6,68	5,55
Wartość maksymalna (ms)	22,00	19,00
Wartość minimalna (ms)	4,00	2,00
Odchylenie standardowe (ms)	1,04	1,50
Mediana (ms)	7,00	5,00

6.5. Scenariusz V – metoda DELETE

Rysunek 7 przedstawia wizualizację czasów przetwarzania żądania HTTP dla usuwania rekordów z bazy danych. Podobnie jak w poprzednich scenariuszach iteracja dla 10 aktywnych użytkowników potrzebuje średnio najwięcej czasu na odpowiedź (Spring – 6,73 ms, Express – 5,84 ms).



Rysunek 7: Średnie czasy przetwarzania żądania DELETE.

Czasy dla Spring oscylują w okolicach 6,50 ms i osiągają wartość ± 1%, podczas gdy zmiany wartości dla Express są wyraźnie widoczne w 3 punktach – dla 10, 50 i 100 użytkowników. Wynoszą one odpowiednio 5,84 ms, 5,61 ms oraz 5,66 ms. Dla pozostałych przebiegów testów wartości te są bliskie 5 ms (w zakresie od 4,97 ms dla 30 użytkowników, do 5,16 ms dla 80 użytkowników.

Szczegółowe statystyki dla 50 użytkowników wykonujących operacje usunięcia widoczne są w Tabeli 7. Najdłuższy czas potrzebny na usunięcie pojedynczego rekordu wynosi 17,00 ms dla Spring i 15,00 ms dla Express. Najkrótszy czas jest taki sam dla obu technologii i wynosi 4,00 ms.

Tabela 7: Wartości parametrów dla metody DELETE – 50 użytkowników

Parametr	Spring	Express
Średnia (ms)	6,48	5,61
Wartość maksymalna (ms)	17,00	15,00
Wartość minimalna (ms)	4,00	4,00
Odchylenie standardowe (ms)	0,83	1,37
Mediana (ms)	6,00	5,00

7. Wnioski

W niniejszym artykule położono nacisk na pomiary wydajności obsługi żądań HTTP – GET, POST, PUT, DELETE dla aplikacji wykonanych w technologii Spring i Express.

Biorąc pod uwagę całość przeprowadzonych badań w 5 scenariuszach testowych aplikacja zaimplementowana z wykorzystaniem mikro-szkieletu programistycznego Express.js radziła sobie lepiej z przetwarzaniem żądań uzyskując krótsze czasy niż odpowiednik aplikacji w szkielecie dla Javy. Przewaga Express nad Spring widoczna jest szczególnie w scenariuszu trzecim. W najlepszym przypadku czas oczekiwania jest dla Spring dłuższy o 167%, a w najgorszym przypadku 249% dla tej samej liczby użytkowników w aplikacji opartej o Express. Jedynym miejscem w którym Express.js przegrywa ze Spring jest pierwszy scenariusz dodawania nowych rekordów dla 70 użytkowników – średni czas jest tutaj najwyższy ze wszystkich.

Ponadto podczas analizy uzyskanych czasów dla szkieletu opartego o język JavaScript i środowisko Node.js można zauważyć tendencję spadkową w zakresie 10 – 40 użytkowników. W scenariuszach 2 – 5 czas odpowiedzi chwilowo rośnie dla 50 użytkowników. W porównaniu do wyników uzyskanych przez drugą aplikację czasy również maleją w stosunku do pierwszej iteracji testów, jednak utrzymują się na zbliżonym poziomie bez względu na liczbę aktywnych użytkowników.

Z podstawowych operacji CRUD najdłużej zajęły operacje aktualizacji i usuwania wskazanych rekordów.

W pracy porównano również popularność obu technologii na przestrzeni kilku lat, wskazując że Express.js jest aktualnie popularniejszym szkieletem niż Spring wśród zawodowych programistów.

We wszystkich przeprowadzonych scenariuszach czasy uzyskane dla 10 użytkowników były największe. W tych przypadkach mechanizmy zarządzające wielowątkowością w Express i Spring nie przydzieliły większej liczby zasobów.

Na podstawie uzyskanych wyników i przeprowadzonych wniosków można stwierdzić, że postawiona we wstępie hipoteza badawcza była słuszna wskazując, że

technologia Express zapewnia wydajniejsze rozwiązanie do budowy aplikacji w architekturze REST.

Zakres przeprowadzonych badań skupiał się wyłącznie na wydajności czasowej aplikacji. Nie jest to jednak jedyne kryterium doboru technologii przy planowaniu projektu. W dalszej pracy należałoby uwzględnić parametry takie jak dostępna dokumentacja, sposób implementacji kluczowych fragmentów kodu, struktura projektu, obciążenie procesora i pamięci RAM, wydajność z innymi bazami danych, czy zachowanie szkieletów dla zmiennej liczby jednoczesnych żądań HTTP dochodzących do serwera. Te i wiele innych kryteriów stanowić będą obszar dalszych badań autorów.

Literatura

[1] Dokumentacja API - Meta for Developers, https://developers.facebook.com/docs/pages/publishing/, [18.06.2023].

[2] Dokumentacja API - Twitter for Developers, https://developer.twitter.com/en/docs/twitter-api/tweets/manage-tweets/api-reference/post-tweets, [18.06.2023].

[3] M. Grudniak, M. Dzieńkowski, REST API performance comparison of web applications based on JavaScript programming frameworks, Journal of Computer Sciences Institute, 19 (2021) 121-125, https://doi.org/10.35784/jcsi.2620.

[4] K. K. Dhall, A performance comparison of restful applications implemented in Spring Boot Java and MS.NET Core, Journal of Physics: Conference Series, 1933 (2021) 12-41, https://doi.org/10.1088/1742-6596/1933/1/012041.

[5] K. Kronis, M. Uhanova, Performance comparison of Java EE and ASP.NET Core Technologies for web API development, Applied Computer Systems 23 (2018) 37-44, https://doi.org/10.2478/acss-2018-0005.

[6] K. Lei, Y. Ma, Z. Tan, Performance comparison and evaluation of Web Development Technologies in PHP, python, and node.js, 2014 IEEE 17th International Conference on Computational Science and Engineering, (2014) 661-668, https://doi.org/10.1109/cse.2014.142.

[7] Stack Overflow Developer Survey 2023, https://survey.stackoverflow.co/2023#most-popular-technologies-webframe-prof, [26.06.2023].

[8] Spring - prostota i uniwersalność najpopularniejszego frameworku Java, https://boringowl.io/tag/spring, [26.06.2023].

[9] Stack Overflow Developer Survey 2022, https://survey.stackoverflow.co/2022#most-popular-technologies-webframe-prof, [26.06.2023].

[10] Express.js - MVC Framework Node.js, https://boringowl.io/tag/express-js, [19.06.2023].

[11] Stack Overflow Developer Survey 2018, https://insights.stackoverflow.com/survey/2018#most-popular-technologies, [26.06.2023].

[12] Stack Overflow Developer Survey 2020, https://insights.stackoverflow.com/survey/2020#technology-web-frameworks-professional-developers2, [26.06.2023].

[13] Stack Overflow Developer Survey 2021, https://insights.stackoverflow.com/survey/2021#most-popular-technologies-webframe-prof, [26.06.2023].

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 360–365
Received: 29 June 2023
Accepted: 24 September 2023

A performance analysis of a cloud database on mobile devices

Badanie wydajności chmurowej bazy danych na urządzeniach mobilnych

Sylwester Kot*, Jakub Smołka

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

The article presents a performance analysis of Firebase cloud database. Two services, namely Realtime Database and Cloud Firestore, are examined, and their query speed are compared to those of the local SQLite database. Basic CRUD operations were examined, taking into account the number of records in the database, the size of individual records and the complexity of the database structure. Upon completion of the research, it was concluded that Realtime Database outperforms Cloud Firestore and cloud databases are slower than the local database when it comes to operations on a single record. However, when working with a larger volume of data, cloud database can achieve better results than SQLite. The accuracy of the outcome is also influenced by the stability of the network connection and the distance from the cloud server.

Keywords: performance; cloud database; Firebase; mobile device

Streszczenie

Artykuł dotyczy badania wydajności chmurowej bazy danych Firebase. Badane są dwie usługi: Realtime Database oraz Cloud Firestore, których prędkość zapytań jest porównywana do prędkości zapytań lokalnej bazy danych SQLite. Zbadane zostały podstawowe operacje CRUD z uwzględnieniem ilości rekordów w bazie danych, rozmiaru pojedynczego rekordu oraz rozbudowaniem struktury bazy. Po zakończeniu badań stwierdzono, że baza Realtime Database jest wydajniejsza od bazy Cloud Firestore oraz chmurowe bazy danych są wolniejsze od lokalnej bazy w przypadku operacji na pojedynczym rekordzie. Jednocześnie przy pracy na większej ilości danych chmurowe bazy danych potrafią osiągać lepsze rezultaty niż SQLite. Wpływ na dokładny wynik ma też stabilność łącza oraz odległość od serwera chmury.

Słowa kluczowe: wydajność; chmurowa baza danych; Firebase; urządzenie mobilne

*Corresponding author

Email address: sylwester.kot@pollub.edu.pl (S. Kot)

1. Introduction

With the development of the market for mobile business applications and applications designed for private clients, the issue of data storage arises. Application developers need to consider scalability and the type of data storage in their databases. The accessibility and storage method are also important factors. Thanks to the ubiquity of the Internet, one of the solutions that has emerged is the cloud-based database. It eliminates the need for local memory for storing information and potential hardware limitations. However, it requires continuous internet access, which may limit the speed of database operations.

Initially, all mobile applications utilized the only available database, SQLite. However, due to technological advancements and the efforts of tech giants such as Amazon and Microsoft, there are now numerous database offerings designed for mobile devices on the market. These databases come in both relational and NoSQL forms, differing in schema structure. Currently, one of the most popular cloud solutions for mobile devices is Firebase, a service created by Google. It offers two database solutions: Realtime Database and Firestore Database. This article presents a performance evaluation of these tools based on the speed of database operations, taking into account the number of records, database schema complexity, and individual record size. The results are then compared with a corresponding local database created in SQLite.

2. Literature review

The article "Cloud database as a service" by W. Al. Shehri [1] discusses cloud-based databases as the future standard for information storage, highlighting their scalability and hardware fault tolerance. It also presents parameters to consider when selecting an appropriate cloud database service, such as data size, portability, transaction capability, availability, and security.

In the article "Comparison of NoSQL and SQL Databases in the Cloud" by D. Hammes, H. Medero, and H. Mitchell [2], relational and non-relational databases are compared as cloud services. The CAP theorem is introduced as a means of identifying the main weaknesses of any database system. According to this theorem, any database implementation must choose two out of three properties: consistency, availability, and partition tolerance. Relational databases prioritize availability and consistency, while NoSQL databases lean towards consistency and partition tolerance. Performance tests were also conducted using Postgres and MongoDB databases, with the results favoring

Postgres as the more performant database. However, further research is deemed necessary to confirm these findings.

The article "A performance comparison of SQL and NoSQL Databases in the Cloud" [3] focuses on comparing a larger number of non-relational databases and examining their performance for key-value data. The results indicate that while non-relational databases are optimized for key-value data, not all services outperform the reference relational database. The results vary depending on the type of database operation and the number of operations performed. The authors determined that Couchbase and MongoDB are the fastest in read, write, and delete operations.

The article "The Comparison Firebase Realtime Database and MySQL Database Performance using Wilcoxon Signed-Rank Test" [4] compares the Firebase database with MySQL using the Wilcoxon signed-rank test for paired observations to determine the optimal database for a mobile application intended for daily nutritional needs for young children. The tests were conducted for all CRUD operations and using the database structure used in the application. The results indicate that Firebase is a more efficient database.

The article "A comparison of NoSQL and SQL Databases over the Hadoop and Spark Cloud Platforms using Machine Learning Algorithms" [5] utilizes machine learning algorithms to create a NoSQL database from a relational database. The authors then evaluate the performance of the algorithms on both databases using the k-means method and random forest. The results indicate the superiority of the non-relational database in terms of operation speed, ranging from 26% to 54%.

The author of the article "A Performance Comparison of SQLite and Firebase Databases from a Practical Perspective" [6] compares two officially supported types of databases on the Android system: SQLite and Firebase, citing results from previous articles. The comparison involves basic database operations such as data insertion, retrieval of data and specific records, updating, and deletion. The study was conducted on a simple database model containing one entity with two properties: ID and text. The obtained results favor SQLite as the more efficient database in every operation except data deletion. It is also noted that Firebase performs better when sharing database resources with a larger number of users or when limited by local disk space.

In the article “On the Performance of Cloud-Based mHealth Applications: A Methodology on Measuring Service Response Time and a Case Study” [7] the Firebase database is being used in a performance test based on a prototype medical application. The study is being conducted on both Android and iOS platforms. Results show that the average response time for Android devices is slightly higher than that for iOS devices, which may be influenced by buffering and differences in the Firebase API design. It is worth noting that the response time is not dependent on the smartphone's battery mode. Additionally, in the case of a Wi-Fi connection, the average response time is lower by at least a factor of two as in the case of an LTE connection, indicating Wi-Fi as a more efficient connection for retrieving large data chunks.

The authors of the article “Monitoring the performance of cloud real-time databases: A firebase case study” [8] are investigating the performance of the Firebase database using Firebase Console and Google Cloud Monitoring tools. It was observed that Firebase Console provides detailed information regarding the database, while in cases of availability or latency issues, consideration should be given to using Google Cloud Monitoring, which automatically collects data on Firebase services. Additionally, it was determined that for the free version of Firebase, the maximum number of concurrent connections to the database is limited to 100.

3. Research method

Two forms of data storage provided by the Firebase service were subjected to the study: Realtime Database and Cloud Firestore, along with a local SQLite database implemented using the Room library. Three different database schemas were used: a simple key-value type, complex database with multiple objects in relationships, and a database with a single record of size 1KB. The complex database with relationships is an order model consisting of a product list and details related to payment and customer. In the SQLite database, the relationships were established using foreign keys, while in the Realtime Database, a reference to object IDs was added, and in Cloud Firestore, three collections were created: payment, order and customer.

For the purpose of this research two mobile apps were made: one for cloud databases and one for local database. To achieve optimal performance for the Cloud Firestore database batched writes were used for higher amount of data. For the Realtime Database transactions were not increasing the performance of operations so standard methods were used.

Cloud servers used in the research were located in Belgium and Western Europe. A Wi-Fi connection with a bandwidth of 30Mb/15Mb was used to connect to the servers.

The study was conducted using two smartphones: Xiaomi Redmi Note 8 Pro and Samsung A8 with the specifications shown in Table 1.

Table 1: Specifications of devices used in research

Parameter	Xiaomi Redmi Note 8 Pro	Samsung A8 (2018)
CPU	Mediatek Hello G90T 8x2.05 GHz	Samsung Exynos 7885 2.2 GHz
RAM	6 GB	4 GB
internal memory	64 GB	32 GB
operating system	Android 11	Android 9

3.1. Research scenarios

All test scenarios were conducted on 10, 100, 500, 1000, 2000 and 10000 records as well as on the three different database models mentioned before. Each test

was performed 10 times, and the average execution time was calculated. After each test, the database and application memory were cleared and generated again for the next test. Prior to the research, the smartphone cache was cleared and all background applications were shut down.

The following operations were examined:

- creation of a database and populating it with data,
- inserting single record,
- editing single record,
- reading single record and entire database,
- deleting single record and entire database.

Additionally, the impact of database size on the performed operations was investigated.

4. Results

Due to the limits of free tier for Cloud Firestore, which is 20000 write and delete operations per day, scenarios using data sample of 10000 records were performed in the following days and tests for other databases were repeated to get the accurate results.

4.1. Inserting data

First scenario tested was about generating database and populating it with data.

The results (Table 2) indicate that for simple and complex data models SQLite is more efficient and has better average execution time regardless of number of records inserted into database. However, when the size of single record is about 1KB, cloud databases are better as more records are being added. Realtime Database manages to beat local database from 100 records onwards and Cloud Firestore manages to get faster time for 1000 records.

Table 2: Execution time of database generation

		Realtime Database	Cloud Firestore	SQLite
Number of records	Database model	Average execution time (ms)		
10	Key-value	63	1302	14.5
	Complex	98.5	1232	36
	Large	506.5	908	203
100	Key-value	111.5	3059.5	20
	Complex	370.5	9851	84.5
	Large	2145.5	5819	3515.5
500	Key-value	155.5	969.5	35
	Complex	1067	4551.4	307
	Large	9932.5	29235.5	21680
1000	Key-value	304.5	1775	43.5
	Complex	1624	9297.5	478
	Large	21574.5	36126	43436.5
2000	Key-value	420.5	3266.5	69.5
	Complex	3086	46962	941.5
	Large	14422	-	86111
10000	Key-value	1405	15018.5	400
	Complex	14852	-	3189
	Large	-	-	493641

Unfortunately, for record count above 2000 in case of Cloud Firestore and 10000 in case of Realtime Database there was an OutOfMemory exception thrown by the server side of database, which prevented getting the results for these conditions.

Next examined operation was about inserting single record to already existing database. This led to the following results (Table 3).

Table 3: Execution time of inserting single record

Database model	Database	Average execution time (ms)
Key-value	Realtime Database	54
	Cloud Firestore	98
	SQLite	5
Complex	Realtime Database	53
	Cloud Firestore	95
	SQLite	11
Large size	Realtime Database	103
	Cloud Firestore	181
	SQLite	20

Realtime Database manages to get faster execution time than Cloud Firestore for inserting single record in each database model scenario. The difference between them is nearly twofold. It can also be observed that the complex model does not cause a decrease in performance while a larger object size results in twice the execution time. On the other hand, the local database experiences a twofold increase in time for complex model and a fourfold increase for a 1KB-sized record.

It is worth noting that the number of records in database does not affect the execution time and the average execution time is composed of every tested case.

4.2. Editing data

The second scenario involves editing a single record in already existing database. In case of a key-value model, a value is changed. In a complex model the order status is changed and for the large set the big size field was changed. The results are as follows (Table 4).

Table 4: Execution time of editing data

Database model	Database	Average execution time (ms)
Key-value	Realtime Database	52
	Cloud Firestore	93
	SQLite	3
Complex	Realtime Database	53
	Cloud Firestore	77
	SQLite	5
Large size	Realtime Database	98
	Cloud Firestore	128
	SQLite	3

When it comes to editing data, execution times are similar to the inserting single record time. Realtime Database achieves differences at maximum of 5 milliseconds for the large record, while other models stay at 53-54 milliseconds. Cloud Firestore manages to get faster execution time in every model and the biggest difference is seen in the 1KB-sized record where the difference is over 50 milliseconds. However, both of the cloud databases have worse performance compared to local database, which has execution time below 5 milliseconds.

Just as with previous scenario, the number of records in database does not affect the execution time of editing a single record.

4.3. Reading data

The next scenario of the study examines database read operations. Both reading a single record and retrieving the entire database are evaluated.

When it comes to reading a single record (Table 5), Realtime Database stays at similar execution time no matter the database model. Even on the large-size model the difference between the best time is less than 10 milliseconds. Cloud Firestore performance is worse than the other cloud database, especially when it comes to simple key-value model where the average time exceeds 100 milliseconds. However, as with previous single-record operations, both databases have worse performance than local database.

Table 5: Execution time of reading single record

Database model	Database	Average execution time (ms)
Key-value	Realtime Database	56
	Cloud Firestore	122
	SQLite	4
Complex	Realtime Database	53
	Cloud Firestore	92
	SQLite	4
Large size	Realtime Database	62
	Cloud Firestore	94
	SQLite	4

Table 6: Execution time of reading whole database

		Realtime Database	Cloud Firestore	SQLite
Number of records	Database model	Average execution time (ms)		
10	Key-value	48.5	196	4.5
	Complex	69.5	169	28
	Large	158	496	3
100	Key-value	86	302	7.5
	Complex	160	149.5	39
	Large	680	2048	7
500	Key-value	101.5	276.5	8.5
	Complex	386.5	106.5	192
	Large	3077.5	10619.5	8.5
1000	Key-value	116	614.5	9.5
	Complex	556.5	171	236.5
	Large	5961	-	11.5
2000	Key-value	161.5	930.5	15
	Complex	1193.5	404	494.5
	Large	5223	-	19.5
10000	Key-value	416	5684.5	185
	Complex	3953.5	186.5	2290.5
	Large	-	-	84.5

Results of reading the entire database (Table 6) presents that Cloud Firestore excels in reading many data from the complex structure. The average execution time for all evaluated database size samples is less than 500 milliseconds. This is especially good for the larger number of records, where execution time for both Realtime Database and SQLite exceeds 2 seconds, whereas Cloud Firestore manages to get similar performance regardless of number of records to read. However, when it comes to simple key-value model and for the large-sized records Cloud Firestore performs the worst out of the three. Realtime Database shows that it can compete with local database on the complex and key-value model at larger set of data, however it falls short on 1KB-sized records. It can also be seen that SQLite struggles with complex database model as the execution time is much higher than for other database models.

Just as in the database generation scenario, for records above 1000 for Cloud Firestore and above 10000 for Realtime Database there was an OutOfMemory exception which made it impossible to gather the time of the operation.

It is worth mentioning that the average execution time for cloud databases decreased every time the read operation was repeated on the same database. This is caused by the device caching the data in memory so it doesn't have to load whole data from the cloud server. The decrease is significant, however in this scenario only the first read from database was measured.

4.4. Deleting data

The last performed test scenario relates to single record and whole database deletion. Results of the tests can be seen below (Table 7, Table 8):

Table 7: Execution time of deleting single record

Database model	Database	Average execution time (ms)
Key-value	Realtime Database	56
	Cloud Firestore	90
	SQLite	4
Complex	Realtime Database	48
	Cloud Firestore	78
	SQLite	10
Large size	Realtime Database	52
	Cloud Firestore	99
	SQLite	3

Results for deleting single record shows that Realtime Database deletes data at around the same time no matter the database model. Cloud Firestore performs better at deleting a complex model record, which is about 20 milliseconds faster than other models. However, as in all previous one-record operations, local database performs much better providing nearly-instant execution time.

As for the database deletion operation the results vary depending on the database (Table 8). Realtime Database seems to be unaffected by both number of records and size of single record. The average execution time for key-value model increases slightly with number of records, on the other hand it decreases for large-sized record the more records are in the database. The performance is slightly worse for complex database model but all tests indicate that execution time is between 63-118 milliseconds, which is better than Cloud Firestore. The latter database has execution time of over a second for all of the database models above 500 records. The best performing model is key-value, following large-sized records at small quantity, but

being outperformed by complex model at records above 500. What is worth mentioning, Realtime Database has better performance in deleting database consisting of larger size data than SQLite for higher amount of records deleted. It is best seen on the 10000 record test as the execution time for local database reaches over 10 minutes compared to only 63 milliseconds for Realtime Database.

Table 8: Execution time of deleting whole database

		Realtime Database	Cloud Firestore	SQLite
Number of records	Database model	Average execution time (ms)		
10	Key-value	70	541	2.5
	Complex	98.5	1354	26
	Large	59.5	495.5	15.5
100	Key-value	73.5	3060.5	3
	Complex	97.5	11253.5	17.5
	Large	118	3937	411.5
500	Key-value	76	905.9	5
	Complex	107.5	4713.4	12
	Large	71.5	6343.1	7649
1000	Key-value	90	1480	6
	Complex	103	8406.5	13.5
	Large	65.5	13397.5	20235
2000	Key-value	82.5	2702	6
	Complex	118.5	20833	19.5
	Large	78	-	148684
10000	Key-value	92.5	11755.5	13
	Complex	118.5	-	107.5
	Large	63	-	>100000

Looking at stability of operations, Realtime Database managed to successfully complete all delete operations regardless of number of records in database. Cloud Firestore however failed to delete the data for large records in number above 2000 and for the complex data model in the 10000 record test due to OutOfMemory exception which happened for the most of the multiple-records tests.

5. Conclusions

The article presents performance tests of selected cloud databases in CRUD operations.

The results obtained in all conducted tests indicate that Realtime Database outperforms Cloud Firestore in terms of efficiency. The only area where Cloud Firestore is favored is reading from a complex database with multiple records. In all other cases, the differences in execution time of operations are more than twice as large.

As for single-record operations, the performance of cloud databases is similar for all of the conducted operations. In both Realtime Database and Cloud Firestore there is slight increase in execution time for large-sized record database model, expect for reading a single record. However, compared to local database, the difference is visible as SQLite performs single-record operations nearly instantly, when cloud databases have respond time within 50-150 milliseconds. When comparing the results of cloud databases with the local one, it is important to consider the fact that in cloud databases, the execution time of operations includes both the data processing time and the server response time, which is doubled due to the query and response time. On the other hand, local databases do not have such limitations, which allows them to execute certain queries instantly, especially for smaller amounts of data.

When it comes to working with large number of data cloud databases manage to outperform local database in generating and deleting database for objects of 1KB size. The result depends on the database, as Realtime Database performs better in simple database model, whereas Cloud Firestore is optimized for complex model and performs exceptionally well in reading database for that type of data. However, it works worse with a simple key-value data model. Another thing to consider is the size limit of operations, as in the research, especially for number of records exceeding 2000, server happened to throw the OutOfMemory exception which make it unable to process big amount of data at the same time, which is not happening in SQLite. For the Cloud Firebase the maximum number of operations for a single transaction can't exceed 500 [9] and for the Realtime Database the limits are only for maximum size of a single response, which is 256MB and write rate at 1000 writes/second [10].

When assessing the performance of cloud databases, it is essential to take into account the conditions that exist in mobile devices. Significant variations in operation times can be attributed to factors such as network stability, connection quality, distance from the cloud server, or server latency. Moreover, when operating with a larger number of records, there is a chance of reaching the concurrent data transmission limit, resulting in exceptions within the application. However, when utilizing SQLite database with the Room library and the MVVC model implementation, such issues do not occur, and queries for a larger number of records can take over 10 minutes. Additionally, Firebase services impose limitations on data size, as observed in Realtime Database, or daily limits on read, write, and delete operations, as seen in Cloud Firestore. Utilizing paid tiers would allow examining how operation times change with a larger number of records, a scenario commonly encountered in advanced IT systems where data volume can reach several million entries.

Summarizing all the points presented above, Realtime Database performs better in basic operations than Cloud Firestore, however Cloud Firestore is more optimized for complex data structure. Both of the cloud databases perform worse than local one in single-record operations but can outperform it if working at large number of data. The user has to bear in mind the limits of a single call according to the documentation.

References

[1] W. Al Shehri, Cloud Database Database as a Service, International Journal of Database Management Systems 5(2) (2013) 1-12.

[2] D. Hammes, H. Medero, H. Mitchell, Comparison of NoSQL and SQL Databases in the Cloud, SAIS 2014 Proceedings (2014) 12-20.

[3] Y. Li, S. Manoharan, A performance comparison of SQL and NoSQL databases, IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM, Victoria, BC, Canada, August 27-August 29, 2013, IEEE (2013) 15-19.

[4] M. Ohyver, J. V. Moniaga, I. Sungkawa, B. E. Subagyo, I. A. Chandra, The Comparison Firebase Realtime Database and MySQL Database Performance using Wilcoxon Signed-Rank Test, Procedia Computer Science 157 (2019) 396-405.

[5] C. H. Lee, Z. W. Shih, A Comparison of NoSQL and SQL Databases over the Hadoop and Spark Cloud Platforms using Machine Learning Algorithms, 2018 IEEE International Conference on Consumer Electronics-Taiwan, ICCE-TW, Taichung, Taiwan, May 19-May 21, 2018, IEEE (2018) 1-2.

[6] A. T. KABAKUŞ, A Performance Comparison of SQLite and Firebase Databases from A Practical Perspective, Düzce Üniversitesi Bilim ve Teknoloji Dergisi 7(1) (2019) 314-325.

[7] D. Inupakutika, G. Rodriguez, D. Akopian, P. Lama, P. Chalela, A. G. Ramirez, On the Performance of Cloud-Based mHealth Applications: A Methodology on Measuring Service Response Time and a Case Study, IEEE Access 10 (2022) 53208-53224.

[8] M. F. Younis, Z. S. Alwan, Monitoring the performance of cloud real-time databases: A firebase case study, 2023 Al-Sadiq International Conference on Communication and Information Technology, AICCIT, Al-Muthana, Iraq, July 4-July 6, 2023, IEEE (2023) 240-245.

[9] Usage and limits | Firestore | Firebase, https://firebase.google.com/docs/firestore/quotas?hl=en [15.06.2023]

[10] Realtime Database Limits | Firebase Realtime Database, https://firebase.google.com/docs/database/usage/limits?hl=en [15.06.2023]

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 366–373
Received: 30 June 2023
Accepted: 28 September 2023

Face Recognition using Deep Learning and TensorFlow framework

Rozpoznawanie twarzy przy użyciu głębokiego uczenia i frameworka TensorFlow

Makrem Beldi*

The National Engineering School of Tunis or ENIT, Tunisia

Abstract
The detection and recognition of human faces, crucial for a wide range of applications, has made progress thanks to precise machine learning techniques. But the complexity can be daunting for newcomers. Our project focuses on building a Python-based framework for face recognition, with the aim of democratising access and fostering innovation. Harnessing the power of TensorFlow and Python, we painstakingly refined a CNN model using AT&T dataset. The results were striking, a remarkable in accuracy. With the strategic addition of layers, the accuracy of our model increased. While recognising the crucial role of accuracy, the importance of deployment time can't be overlooked. Our discussions also highlight the interplay between accuracy, operational efficiency and resource allocation.

Keywords: Face Recognition; Face Detection; CNN; TensorFlow

Streszczenie
Wykrywanie i rozpoznawanie ludzkich twarzy, kluczowe dla szerokiego zakresu zastosowań, poczyniło postępy dzięki precyzyjnym technikom uczenia maszynowego. Jednak ich złożoność może być zniechęcająca dla nowicjuszy. Nasz projekt koncentruje się na budowie platformy opartej na języku Python do rozpoznawania twarzy. Wykorzystując TensorFlow i Pythona, starannie dopracowaliśmy model CNN przy użyciu zbioru danych AT&T. Wyniki były uderzające, z niezwykłą dokładnością. Dzięki strategicznemu dodawaniu warstw, dokładność naszego modelu wzrosła. Nasze dyskusje podkreślają również wzajemne korelacje pomiędzy dokładnością, wydajnością operacyjną i alokacją zasobów.

Słowa kluczowe: rozpoznawanie twarzy; wykrywanie twarzy; CNN; TensorFlow

*Corresponding author

Email address: makrem.beldi@enit.utm.tn (M. Beldi)

1. Introduction

Facial recognition technology, which relies on the ability to distinguish individuals based on their facial features, plays a central role in security and law enforcement [1-5]. Despite its effectiveness, this technology faces a number of multifaceted challenges [9,10]. Concerns about invasion of privacy, algorithmic bias and security vulnerabilities underscore the urgent need for reliable deployment [11,12].

The ethical and competent use of facial recognition requires a comprehensive approach to ensure not only accuracy, but also adherence to quality standards that encompass functional meaning, reliability, performance, security and feasibility. This article embarks on a journey towards these goals and provides valuable insights into the field of facial recognition refinement, enriched by the capabilities of artificial intelligence and deep learning.

This article is organized as follows: Segment 2 digs into the scene of facial recognition, featuring its advantages for security and framing existing techniques. Area 3 presents our proposed technique, covering information assortment, pre-handling, extraction and the advancement of our FaceDetect framework. Sections 4 to 6 dig into the center of our examination, clarifying fundamental layers, introducing the engineering of our proposed CNN model, and thoroughly assessing its presentation. At long last, Segment 7 epitomizes our excursion, summing up our discoveries and preparing for future advances in facial recognition innovation.

2. State of the art

2.1. Benefits of facial recognition for security and use

Without a doubt, facial recognition innovation offers a few benefits in security and access control frameworks [7,13]. Its exactness in recognizing people in light of remarkable facial highlights makes it a solid and secure option in contrast to customary distinguishing proof techniques. The innovation likewise increments effectiveness by rapidly checking and recognizing people progressively [14-16], particularly in high-traffic regions like air terminals, arenas or transportation center points. By recognizing expected dangers, facial recognition innovation can further develop security and diminish the gamble of damage to individuals and property [17,18].

Facial recognition innovation can likewise assist with diminishing the expenses related with security and access control frameworks [12]. Customary distinguishing proof techniques, like actual identifications or passwords, are defenseless against misfortune or burglary and require incessant substitution and refreshing. Be that as it may, facial recognition innovation dispenses

with these expenses by utilizing an individual's novel facial highlights to recognize them.

Generally, facial recognition innovation offers a few advantages in security and access control frameworks, further developing productivity, security and wellbeing while at the same time lessening costs.

2.2. Existing face recognition methods

An assortment of face recognition procedures are accessible, traversing traditional supervised and automated methods, as well as those utilizing the capacities of deep learning techniques.

2.2.1.Traditional methods

In face recognition, feature (descriptor) extraction plays a very important role in overall system performance. In the literature, two feature extraction environments exist: local features describing the dynamics of an image area, and global features entered by the attributes of the entire image.

2.2.2.Local features

Local descriptors, rooted in local features, focus on characterizing specific image regions [49]. Each descriptor captures sectorial details, requiring integration with other descriptors to form a holistic image representation. Elements such as intensity, color, and texture are considered local descriptors. These methods often rely on prior knowledge of facial structural morphology, involving the identification or extraction of local facial features. Local techniques offer the advantage of effectively addressing changes in pose, lighting, and facial expression.

Local Binary Patterns (LBP): LBP stands as a facial recognition approach centered on extracting texture features by representing faces as binary pattern histograms [27]. Unlike a global approach, LBP accentuates local texture nuances within facial images [28]. By sidestepping certain information losses linked to gradients, LBPs also capture diverse local structures [29]. This method's versatility is showcased in evaluating facial recognition performance for low-quality images [45].

Gabor Wavelets: Gabor filters, renowned for frequency localization and orientation selectivity, hold significance in image processing [50]. In the context of facial feature extraction, Gabor filters offer the advantage of capturing information across various orientations and scales. Although computationally intensive, Gabor methods find widespread use in image analysis and head pose estimation.

Oriented Gradient Histograms (HoG): Oriented Gradient Histograms, introduced by Dalal and Triggs, describe local and object appearances based on gradient intensity distribution [51]. This method operates by dividing images into cells, calculating gradient direction histograms for each cell, and combining these histograms to form the HoG parameter. It maintains invariance for geometric and photometric transformations, rendering it suitable for individual detection.

Scale Invariant Feature Transform (SIFT): Lowe's SIFT descriptor [52] identifies, characterizes, and describes interest areas within images, facilitating subsequent recognition in different images. This descriptor's value-based representation enables efficient matching of points of interest.

Convolutional Neural Networks (CNNs): In facial recognition, CNNs are adept at extracting local features [30]. By applying convolutional filters, CNNs unveil contours, textures, and patterns, generating local feature representations [31]. These local details amalgamate to form a comprehensive global representation of facial images.

2.2.3.Global features

The core concept is to extract a set of attributes computed over the entire image. This approach aims to transform the initial vectorised input face image into a lower dimensional space. This projection is carefully designed to highlight crucial and discriminative features that are essential for distinguishing between individuals. Many of these techniques delve into face subspace analysis, recognising that the face class occupies a subspace within the input image space (facespace). By focusing only on the relevant features - essentially the facial aspects - the dimensionality of these images can be reduced.

Early efforts in face recognition focused mainly on global features implicitly extracted by subspace decomposition methods. In particular, Eigenfaces and Fisher Faces embody this by projecting the entire face into a linear subspace, thereby capturing the range of facial variations.

Eigenfaces: This technique uses Principal Component Analysis (PCA) to extract facial features and represent them as Eigenfaces [19-21]. Eigenfaces are mathematical vectors that symbolize common facial feature variations in a dataset. During recognition, an input image is compared to the Eigenfaces to find the closest matches, facilitating individual identification. Known for its efficiency, Eigenfaces adeptly handles lighting, expressions and pose variations. Its real-world applications include access control, security systems, and surveillance [22].

Fisherfaces: Fisherfaces relies on the extraction of facial features using Linear Discriminant Analysis (LDA) [23-25]. LDA identifies features with substantial inter-individual differences while minimising within-person variation. This method identifies the most distinctive facial features for accurate identification. Fisherfaces is gaining ground in scenarios with limited samples per individual, extending its application to law enforcement, access control and surveillance [22,26].

2.3. Deep learning-based methods

Deep learning, an artificial intelligence methodology, operates through intricate neural networks and excels in learning from voluminous datasets. With prowess in tackling intricate tasks like image recognition and prediction, deep learning offers a potent avenue. Within deep neural networks, multiple interconnected layers of

neurons facilitate the acquisition of hierarchical data representations. Deep learning's capacity to automatically extract pertinent local features contributes to remarkable performance across diverse domains. Convolutional neural networks (CNNs), particularly, hold promise, often outperforming conventional machine learning techniques [55]. This research spotlights deep learning's potential in bolstering agriculture and plant health, as it continues to reshape computing and various industries, continually redefining the boundaries of machine learning [54].

Convolutional Neural Networks (CNN): For facial recognition, a deep neural network model leverages specialized architectures like CNNs. In this structure, interconnected layers of neurons decipher distinctive features from images to identify individuals [30-33].

Siamese Networks: Siamese networks feature in facial recognition to gauge facial similarities. Employing identical segments, these networks extract local features from pairs of facial images. By comparing extracted features using a similarity metric, Siamese networks ascertain distances between faces of the same or different individuals. Their robustness shines, transcending lighting, expression, or pose variations. Training on extensive facial image pairs enables them to discern discriminative features that set individuals apart [30,34-36].

Triplet Networks: Triplet networks extend recognition by assessing groups of three faces to determine pairs belonging to the same individual [30,37]. This advanced approach crafts more discriminating facial representations, projecting faces into a space where faces of the same person converge while those of different people diverge [56].

Selecting the most fitting method hinges on factors such as accuracy, speed, and computational resources, as each technique possesses its unique advantages and limitations.

2.4. Advantages and limitations of learning-based

Learning-based approaches, such as deep neural networks, have become increasingly popular for facial recognition due to their ability to automatically extract relevant features and learn discriminative representations. However, they also have certain advantages and limitations that should be considered .

2.4.1.Advantages

- High accuracy: Learning-based approaches have been shown to achieve state-of-the-art performance in facial recognition tasks, often outperforming traditional methods [30,31,38].
- Robustness to variation: Learning-based approaches can handle variations in lighting, pose, and facial expression, making them more robust and reliable in real-world scenarios [31,38].
- Adaptability: Learning-based approaches can be trained on large datasets, making them highly adaptable to different face recognition tasks and applications[31,38].

2.4.2.Limitations

- Data Requirements: Learning-based approaches require large amounts of labeled data for training, which can be time-consuming and expensive to obtain [39,40].
- Computational Resources: Learning-based approaches can be computationally intensive, requiring powerful hardware and significant training time [41].
- Vulnerability to Adversarial Attacks: Learning-based approaches are vulnerable to adversarial attacks, where malicious actors can intentionally manipulate or modify input images to deceive the system [42-44].

Overall, learning-based approaches have shown great promise in facial recognition, but their advantages and limitations should be carefully considered when selecting a method for a specific application.

3. Methodology

The primary goal of face recognition is to detect and identify faces of different sizes, shapes and orientations. Within the field of computer vision, challenges in face detection and recognition remain, including issues such as illumination variations, occlusions and object orientations. Existing research highlights that conventional approaches using traditional features in supervised and global feature extraction systems have the advantage of preserving the holistic information, including texture, colour, position, lighting and shape, which are all critical for distinguishing faces within a set. These methods often represent faces as high-dimensional vectors proportional to the original image size. By merging or concatenating pixel values, these vectors are projected into a novel, lower-dimensional space, optimising the variance of the data. In particular, a face image is transformed into a linear combination of eigenvectors, which underpins the k-nearest neighbour (KNN) classification method [57].

However, there are inherent challenges due to uncontrollable factors such as pose, facial expression, lighting and occlusion, which ultimately affect recognition accuracy and lead to suboptimal performance [57,58]. In response, the deep learning paradigm [59] has emerged as a compelling alternative. Influenced by this development, we adopt the deep learning methodology, specifically using a convolutional neural network (CNN) powered by TensorFlow [60-62], an open source deep learning framework. This CNN-driven solution significantly improves face recognition by providing a robust basis for face detection.

In this context, the use of convolutional neural networks within deep learning manifests as a potent tool for increasing accuracy and precision in face detection and recognition tasks. Our proposed solution introduces a breakthrough approach that enables developers to effortlessly detect facial features and identify faces in

images. Our architecture is a testament to this, culminating in a set of features that streamline and enhance the process.

The overall goal of our application is to increase the effectiveness of facial recognition, in line with the principles embedded in our code. Through iterative improvements, our mission focuses on improving critical aspects such as time efficiency, accuracy and memory usage. Through a series of improvements, we aim to pave the way for seamless future deployments of the application. We aim to achieve unparalleled performance, ensuring fast and accurate face recognition while optimising memory resources.

3.1. Data collection

The process of data acquisition is of paramount importance in the development of a Convolutional Neural Network (CNN) model. In this endeavour, the choice of dataset has a profound impact on the training and evaluation results. In this respect, the AT&T dataset [63] proves to be a central resource. This dataset plays a central role in both training and evaluating the effectiveness of machine learning models.

The AT&T face dataset, which consists of a set of facial images from 40 different individuals, is a comprehensive repository. Each individual is encapsulated by a collection of ten different facial images, which add up to a cumulative total of 400 images. With dimensions of 92x112 pixels, these images adhere to a greyscale color scheme using 256 levels per pixel. This dataset serves as a cornerstone that allows us to refine the capabilities of our CNN model, paving the way for refined and effective face recognition capabilities.

3.2. Optimization of face recognition system

The Convolutional Neural Network (CNN) architecture used in this work is a specialised deep learning framework tailored for image recognition and computer vision applications, with a focus on face recognition. Comprising a sequence of distinct layers, the CNN systematically extracts complex features from input facial images, enabling nuanced discrimination between individuals. The process begins with a convolutional layer, which convolves learnable filters over the input image to reveal elementary features such as edges and textures. Subsequent max-pooling layers downsample the extracted features to improve translation invariance. The flattening layer is used to reshape the transformed data, making it suitable for processing by fully connected layers. These densely connected layers of the neural network skilfully merge the extracted features, progressively constructing higher-level abstractions. Ultimately, the network's output layer provides probabilities corresponding to the potential identities, effectively achieving face classification.

This comprehensive design, using the TensorFlow library in Python as the programming framework, facilitated the exploration and development of the CNN model. In addition, the AT&T dataset, consisting of facial images of 40 individuals, was instrumental in training and fine-tuning the CNN model, resulting in a robust and accurate face recognition system.

In our study, we experimented with various parameters, such as the quantity and nature of layers, as well as the number of epochs, in order to conduct a comprehensive performance evaluation. We assessed the performance using the AT&T dataset, as shown in this Table 1.

Table 1: A brief description of the datasets AT&T

	Class	Training	Testning	Dimension
AT&T	40	400	40	(112*92)

4. Essential layers to develop CNN

In my experiments, I began by constructing a fundamental CNN model that incorporated four essential layers: Conv2D, MaxPooling2D, Flatten, and Dense. This architecture is detailed in Table 2. Convolutional Neural Networks (CNNs) incorporate various filters or kernels, each equipped with trainable parameters. These filters convolve spatially over an input image, detecting features such as edges and shapes. The abundance of these filters effectively learns to capture spatial characteristics, driven by the weights acquired through backpropagation. Stacked filter layers further enable the identification of intricate spatial patterns, progressively transforming the image into a highly abstracted representation conducive to predictive tasks.

Table 2: Architecture of essential CNN model

Layers	Output Shape	Parameters
Conv2D	(110, 90, 32)	320
MaxPooling2D	(55, 45, 32)	0
Flatten	(79200)	0
Dense	40	3168040
Total parameters		3168360

The fondamental model architecture consists of quatre layers designed to process and extract complex features from the input data. Starting with a Conv2D layer. A MaxPooling2D layer then reduces the output dimensions. The Flatten layer then reshapes the data, allowing efficient data manipulation. Finally, the Dense layer with 40 output nodes or class, contributing to the complexity and ability of the model to learn and predict patterns in the data.

5. Architecture of proposed CNN

The proposed model in Table 3 introduces a number of improvements that contribute to its improved performance and efficiency. By introducing batch normalisation after each Conv2D layer, the model's ability to generalise and learn from the data is enhanced. This ensures that the network's intermediate outputs are normalised, mitigating potential problems associated with vanishing gradients and accelerating convergence. In addition, the inclusion of MaxPooling2D layers aids

spatial reduction, allowing the model to focus on essential features and patterns while minimising computational load.

Taking into account the optimisation of the architecture, the number of parameters is significantly reduced compared to the original model. These design choices improve the model's ability to capture complex features while maintaining a more streamlined number of parameters. This translates into improved training speed and reduced risk of overfitting, ensuring the model's effectiveness and adaptability across different applications.

Table 3: Architecture of proposed CNN model

Layers	Output Shape	Parameters
Conv2D	(110,90, 32)	320
BatchNormalization	(110, 90, 32)	128
Max_pooling2D	(55, 45, 32)	0
Conv2D	(53, 43, 64)	18496
BatchNormalization	(53, 43, 64)	256
MaxPooling2D	(26, 21, 64)	0
Conv2D	(24, 19, 128)	73856
Flatten	(58368)	0
Dense	(40)	2334760
Total parameters		2427816

6. Evaluation of the performance of essential layers to develop CNN model

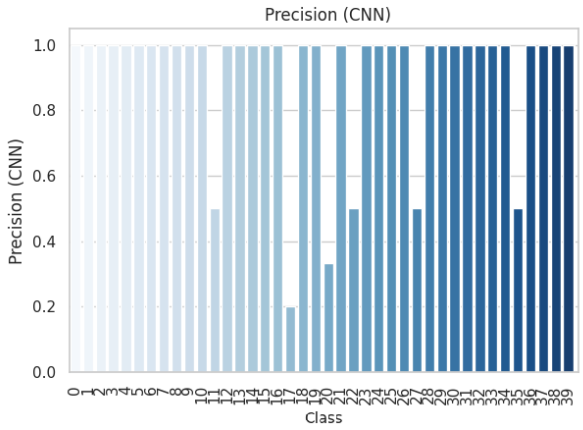


Figure 1: Precision Distribution Before Enhancement.

In Figure 1 shows the distribution of accuracy values for the different classes. The basic architecture of a CNN for facial recognition clearly shows that it is preferable to test several modifications before arriving at a high-performance, reliable architecture. The variation in recognition accuracy between the different classes is evident. Some classes show higher accuracy, while six classes achieve accurate recognition. This analysis provides an overview of the initial performance of the fundamental model, highlighting areas of strength and areas for improvement. In the following experiment in section 7 we try to improve the quality of the system in terms of functional relevance and performance indicators of a calculation.

Table 4: Global metrics essential layers to develop CNN model

Parameterization	Values
CNN Layers	4
Epochs	10
Accuracy	90.8%
Training time	21.26 (s)
Testing time	0.15 (s)
Total memory	6576 bytes

Table 4 present the underlying setup and the outcomes got for a fundamental design of the CNN model.The CNN model, made out of an information convolutional layer with 32 channels followed by a maximum pooling layer, accomplished a precision of 0.908 on the acknowledgment task. The preparation time was 21.27 seconds, while the testing time was 0.15 seconds, for a presentation concentrate on as far as execution time in time space. The memory usage of the model during execution was 48 bytes, and the memory usage for putting away the expectation probabilities was 6528 bytes for an exhibition concentrate on as far as memory space and assets consumed.

This description sums up the key execution measurements and asset use parts of your CNN model in a reasonable and succinct way.

7. Evaluation of the performance of the proposed model

We rolled out a few improvements to the design of our CNN model, which prompted a huge improvement in recognition execution. We expanded the intricacy of the model by adding additional layers. The design currently comprises of three successive convolutional layers, each followed by a clump normalization layer to upgrade stability and convergence during preparing. We likewise presented a new convolutional layer with 256 filters. To forestall overfitting, we incorporated a dropout layer with a pace of 0.5.

Table 5: Global metrics of the proposed model

Parameterization	Test 1 (Values)	Test 2 (Values)
CNN Layers	9	9
Epochs	10	20
Accuracy	85%	96.3%
Training time	204.49 (s)	203.13(s)
Testing time	0.61 (s)	0.82 (s)
Total memory	6576 bytes	6576 bytes

Table 5 shows the modifications be made to the basic model and the variation in parameters such as the number of layers, the number of epochs and the results. We extended the training duration to 20 epochs to allow for better convergence and feature learning. As a result of these improvements, our CNN model exhibited a noticeable boost in precision, achieving a value of 0.963. The training process took approximately 287.5

seconds, while the testing time was around 0.88 seconds.

In Figure 2 the chart showcases the change in precision distribution across classes after implementing model enhancements. The disparity among class-specific precision values has notably reduced. The refinement efforts, aimed at enhancing overall accuracy, have yielded a more balanced distribution. This analysis emphasizes the positive impact of strategic modifications on recognition accuracy across a range of facial features and expressions. The chart underscores how our adjustments have led to improved precision for a wider classes.

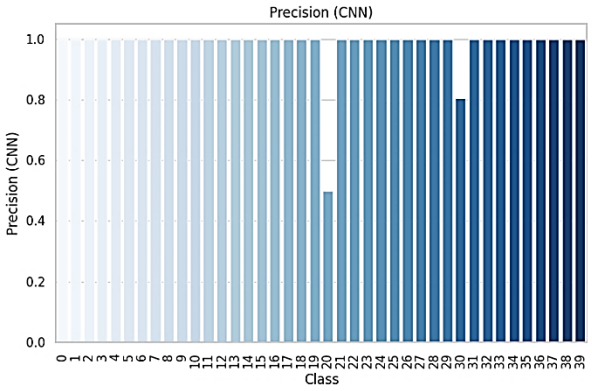


Figure 2: Precision Enhancement Impact.

Throughout execution, the model's memory usage remained consistent at 48 bytes. The memory used for storing prediction probabilities remained at 6528 bytes. The overall memory utilization during execution was unchanged at 6576 bytes.

8. Conclusions

Our journey to enhance the CNN model's architecture led us to substantial improvements in recognition accuracy, empowered by the combined forces of TensorFlow and Python. The essence of these enhancements is most vividly captured in the precision rates 0.90 initial and post-enhancements 0.96 signifying our model's prowess in facial recognition accurately.

While acknowledging the importance of recognition precision, it's equally vital to consider execution efficiency. The discussions around training time and memory space shed light on the practical implications of deploying such models. With the CNN model dynamic architecture of facial recognition constantly evolving, our findings serve as a valuable compass, guiding the optimization of recognition accuracy and resource utilization alike.

References

[1] R. Rajeev et al. A fast and accurate system for face detection, identification, and verification. IEEE Transactions on Biometrics, Behavior, and Identity Science 1(2) (2019) 82-96.

[2] H. Wang, J. Hu, W. Deng, Face feature extraction: a complete review, IEEE Access 6 (2017) 6001-6039.

[3] M. Sharif, M.Y Javed, Javed, M. Sajjad, Face recognition based on facial features. Research Journal of Applied Sciences, Engineering and Technology 4(17) (2012) 2879-2886.

[4] L. Shen, L. Bai, A review on Gabor wavelets for face recognition, Pattern analysis and applications 9 (2006) 273-292.

[5] S. Anwarul, D. Susheela, A comprehensive review on face recognition methods and factors affecting facial recognition accuracy, Proceedings of ICRIC 2019: Recent Innovations in Computing (2020) 495-514.

[6] M. Owayjan, A. Dergham, G. Haber, N. Fakih, A. Hamoush, E. Abdo, New trends in networking, computing, E-learning, systems sciences, and engineering, Springer International Publishing, 2015.

[7] J.H. Li, Cyber security meets artificial intelligence: a survey, Frontiers of Information Technology & Electronic Engineering 19(12) (2018) 1462-1474.

[8] B. Gupta, D.P. Agrawal, S. Yamaguchi, Handbook of research on modern cryptographic solutions for computer and cyber security, IGI global, 2016.

[9] P.M. Kumar, U. Gandhi, R. Varatharajan, G. Manogaran, T. Vadivel, Intelligent face recognition and navigation system using neural learning for smart security in Internet of Things, Cluster Computing 22 (2019) 7733-7744.

[10] M.S. Obaidat, I. Traore, I. Woungang, Biometric-based physical and cybersecurity systems. Cham: Springer International Publishing, 2019.

[11] Z. Ma et al., Lightweight privacy-preserving ensemble classification for face recognition, IEEE Internet of Things Journal 6(3) (2019) 5778-5790.

[12] Z. Zhang et al., Artificial intelligence in cyber security: research advances, challenges, and opportunities, Artificial Intelligence Review (2022) 1-25.

[13] R. Chellappa, P. Sinha, P.J. Phillips, Face recognition by computers and humans, Computer 43(2) (2010) 46-55.

[14] M. Sahani et al., Web-based online embedded door access control and home security system based on face recognition, 2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]. IEEE, 2015.

[15] M. Owayjan et al., Face recognition security system. New trends in networking, computing, E-learning, systems sciences, and engineering, Springer International Publishing, 2015.

[16] S.A. Radzi et al., IoT based facial recognition door access control home security system using raspberry pi, International Journal of Power Electronics and Drive Systems 11(1) (2020) 417-424.

[17] S. Trivedi, P. Nikhil, Virtual Employee Monitoring: A Review on Tools, Opportunities, Challenges, and Decision Factors, Empirical Quests for Management Essences 1(1) (2021) 86-99.

[18] H. Ai, X, Cheng. Research on embedded access control security system and face recognition system, Measurement 123 (2018) 309-322.

[19] L. Sirovich, M. Kirby, Low-dimensional procedure for the characterization of human faces, Josa a 4(3) (1987) 519-524.

[20] A. K. Jain, R. C. Dubes, Algorithms for Clustering Data. New Jersey: Prentice-Hall, 1988.

[21] K. Fukunaga, Introduction to Statistical Pattern Recognition, second ed. Boston, MA: Academic Press, 1990.

[22] M. Ringnér, What is principal component analysis?. Nature biotechnology 26(3) (2008) 303-304.

[23] J. Lu, K.N. Plataniotis, A.N. Venetsanopoulos, Face recognition using LDA-based algorithms, IEEE Transactions on Neural networks 14(1) (2003) 195-200.

[24] J. Lu, K.N. Plataniotis, A.N. Venetsanopoulos, Regularized D-LDA for face recognition. 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).Vol. 3. IEEE, 2003.

[25] G.L. Marcialis, R. Fabio, Fusion of LDA and PCA for Face Recognition. Department of Electrical and Electronic Engineering, University of Cagliari, Piazza díArmi (2002).

[26] A. Bansal, M. Kapil, A. Sahil, Face recognition using PCA and LDA algorithm. 2012 second international conference on Advanced Computing & Communication Technologies. IEEE, 2012.

[27] T. Ojala, P. Matti, H. David, A comparative study of texture measures with classification based on featured distributions, Pattern recognition 29(1) (1996) 51-59.

[28] T. Ahonen, M Pietikäinen, Soft histograms for local binary patterns, Proceedings of the Finnish signal processing symposium, FINSIG 5(9) 2007.

[29] R. Hadsell, C. Sumit, L. Yann, Dimensionality reduction by learning an invariant mapping, 2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06) 2. IEEE, 2006.

[30] F. Schroff, K. Dmitry, P. James, Facenet: A unified embedding for face recognition and clustering, Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[31] R. E. Saragih, Q. H. To, A survey of face recognition based on convolutional neural network, Indonesian Journal of Information Systems 4(2) (2022).

[32] S. Almabdy, E. Lamiaa, Deep convolutional neural network-based approaches for face recognition, Applied Sciences 9(20) (2019) 4397.

[33] R. Chauhan, K.K. Ghanshala, R.C. Joshi, Convolutional neural network (CNN) for image detection and recognition, 2018 first international conference on secure cyber computing and communication (ICSCCC). IEEE, 2018.

[34] X. Zhou, et al., Siamese neural network based few-shot learning for anomaly detection in industrial cyber-physical systems, IEEE Transactions on Industrial Informatics 17.8 (2020) 5790-5798.

[35] S. Bell, K. Bala, Learning visual similarity for product design with convolutional neural networks, ACM transactions on graphics (TOG) 34(4) (2015) 1-10.

[36] D. Chicco, Siamese neural networks: An overview, Artificial neural networks (2021) 73-94.

[37] Y. Feng, et al., Triplet distillation for deep face recognition, 2020 IEEE International Conference on Image Processing (ICIP). IEEE, 2020.

[38] R. Jafri, H. R. Arabnia, A survey of face recognition techniques, Journal of information processing systems 5(2) (2009) 41-68.

[39] R. Cabeza, et al., The prototype effect in face recognition: Extension and limits, Memory & cognition 27 (1999) 139-151.

[40] J. Galbally, M. Sébastien, F. Julian, Biometric antispoofing methods: A survey in face recognition, IEEE Access 2 (2014) 1530-1552.

[41] H. J. Hsu, K. T. Chen, Face recognition on drones: Issues and limitations, Proceedings of the first workshop on micro aerial vehicle networks, systems, and applications for civilian use. 2015.

[42] Y. Dong, et al., Efficient decision-based black-box adversarial attacks on face recognition, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.

[43] Y. Zhong, D. Weihong, Towards transferable adversarial attack against deep face recognition, IEEE Transactions on Information Forensics and Security 16 (2020) 1452-1466.

[44] N. Mani, M. Melody, T.S. Moh, Defending deep learning models against adversarial attacks, International Journal of Software Science and Computational Intelligence (IJSSCI) 13(1) (2021) 72-89.

[45] J. Gozdur, B. Wiśniewski, P. Kopniak, Comparison of the effectiveness of selected face recognition algorithms for poor quality photos, Journal of Computer Sciences Institute 10 (2019) 67-70.

[46] A. Cellerino, B. Davide, S. Ferdinando, Sex differences in face gender recognition in humans, Brain research bulletin 63(6) (2004) 443-449.

[47] T. Baltrusaitis, P. Robinson, L. P. Morency, Constrained local neural fields for robust facial landmark detection in the wild, Proceedings of the IEEE international conference on computer vision workshops. 2013.

[48] S. W. Arachchilage, E. Izquierdo, A framework for real-time face-recognition, 2019 IEEE Visual Communications and Image Processing (VCIP). IEEE, 2019.

[49] T. Tuytelaars, K. Mikolajczyk, Local invariant feature detectors: a survey, Foundations and trends in computer graphics and vision 3(3) (2008) 177-280.

[50] L. Shen, L. Bai, A review on Gabor wavelets for face recognition, Pattern analysis and applications 9 (2006) 273-292.

[51] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05). Vol. 1. IEEE, 2005.

[52] K. Mikolajczyk, C. Schmid, A performance evaluation of local descriptors, IEEE transactions on pattern analysis and machine intelligence 27(10) (2005) 1615-1630.

[53] D. G. Lowe, Distinctive image features from scale-invariant keypoints, International journal of computer vision 60 (2004) 91-110.

[54] A. Naeem, et al., Malignant melanoma classification using deep learning: datasets, performance measurements, challenges and opportunities, IEEE access 8 (2020) 110575-110597.

[55] R. Sujatha, et al., Performance of deep learning vs machine learning in plant leaf disease detection, Microprocessors and Microsystems 80 (2021) 103615.

[56] E. Hoffer, N. Ailon, Deep metric learning using triplet network. Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3. Springer International Publishing, 2015.

[57] P. Parveen, B. Thuraisingham, Face recognition using multiple classifiers, 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06). IEEE, 2006.

[58] J.R. Beveridge, et al., A nonparametric statistical comparison of principal component and linear discriminant subspaces for face recognition, Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001. Vol. 1. IEEE, 2001.

[59] P. Kamencay, et al., A new method for face recognition using convolutional neural network, Advances in Electrical and Electronic Engineering 15(4) (2017) 663-672.

[60] M. Abadi, TensorFlow: learning functions at scale, Proceedings of the 21st ACM SIGPLAN international conference on functional programming. 2016.

[61] J. V. Dillon, et al., Tensorflow distributions, arXiv preprint arXiv:1711.10604 (2017).

[62] B. Pang, E. Nijkamp, Y N Wu, Deep learning with tensorflow: A review, Journal of Educational and Behavioral Statistics 45(2) (2020) 227-248.

[63] F.S Samaria, A.C Harter, Parameterisation of a stochastic model for human face identification, In Proceedings of 1994 IEEE workshop on applications of computer vision. IEEE, (1994) 138-142.

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 374–382
Received: 30 June 2023
Accepted: 29 August 2023

Comparison of tools for creating and conducting automated tests

Porównanie narzędzi do tworzenia i przeprowadzania testów automatycznych

Grzegorz Wojciech Bielesza*, Mariusz Dzieńkowski

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The aim of the article was to compare three tools for preparing and conducting automated tests - JUnit5, TestNG and Spock. The study involved the execution of a series of three experiments that consisted of performance tests based on five test scenarios, functional tests checking the capabilities of each tool, and popularity and support tests. The performance experiment consisted in setting up the test environment, preparing tests based on developed scenarios, executing them multiple times, and then verifying and analysing the obtained time results. The comparison of tools capabilities was based on the extracting of a set of key functionalities of the software used to develop tests. The last analysis was based on a comparison of the popularity of the tested platforms, and was performed on the basis of three popular websites providing information on user activity within the test tool.

Keywords: automated testing; JUnit5; TestNG; Spock

Streszczenie
Celem artykułu było porównanie trzech narzędzi do przygotowywania i przeprowadzania testów automatycznych – JUnit5, TestNG oraz Spock. Zrealizowane badania obejmowały wykonanie serii trzech eksperymentów, na które się składały testy wydajnościowe, oparte na pięciu scenariuszach testowych, testy funkcjonalne sprawdzające możliwości każdego z narzędzi oraz testy popularności i wsparcia. Eksperyment wydajnościowy polegał na skonfigurowaniu środowiska testowego, przygotowaniu testów bazujących na opracowanych scenariuszach, wielokrotnym ich wykonaniu, a następnie weryfikacji i analizie uzyskanych wyników czasowych. Porównanie możliwości narzędzi opierało się na wyodrębnieniu zestawu kluczowych funkcjonalności oprogramowania służącego do opracowywania testów. Ostatnia analiza polegała na porównaniu popularności badanych platform testowych i zrealizowana została na podstawie trzech popularnych serwisów internetowych udostępniających informacje o aktywności użytkowników w obrębie danego narzędzia testowego.

Słowa kluczowe: testowanie automatyczne; JUnit5; TestNG; Spock

*Corresponding author
Email address: grzegorz.bielesza@pollub.edu.pl (G. W. Bielesza)

1. Wstęp

Występowanie błędów w systemach informatycznych spowodowanych niepoprawną implementacją może wiązać się z różnego rodzaju następstwami. Wykrycie niewielkiej usterki w środowisku deweloperskim, może skutkować szybką eliminacją nieprawidłowości, niosącą za sobą żadnych odczuwalnych, negatywnych konsekwencji dla użytkowników. Istnieją jednak przypadki, w których wystąpienie luk w oprogramowaniu w środowisku produkcyjnym spowodowało ograniczenie dostępności aplikacji, a niektóre problemy z bezpieczeństwem czy też stabilnością systemu były przyczyną poważnych kłopotów finansowych czy prawnych, a nawet doprowadzały do wycofania systemu z rynku.

Testy to niezbędny element każdego procesu wytwarzania oprogramowania. Bez odpowiedniego przetestowania napisanego kodu, praktycznie nieosiągalne jest poprawne działanie programu. Dotyczy to zarówno małych, jak i dużych systemów. Im większy system, tym większy potencjał na występowanie luk w kodzie, które docelowo mogą prowadzić do poważnych problemów podczas fazy utrzymania oprogramowania.

Wobec powyższego, większość profesjonalnych zespołów produkujących oprogramowanie stawia duży nacisk na testowanie wprowadzanych zmian w kodzie. Bywają przypadki, w których testami zajmują się deweloperzy oraz takie, w których odpowiedzialność za odpowiednie przetestowanie funkcjonalności spoczywa na wykwalifikowanych w tym aspekcie testerach.

Wykrycie wszystkich błędów w oprogramowaniu jest niewykonalne, zwłaszcza gdy badany jest duży system, jednakże odpowiednio przeprowadzone testy programu, zdecydowanie zmniejszają ryzyko wystąpienia nieprzewidzianych skutków ubocznych związanych z implementacją, jednocześnie pozwalając na doprowadzenie jej do satysfakcjonującego poziomu poprawności, szczególnie dla kluczowych funkcjonalności systemu.

Na rynku istnieje wiele narzędzi do przygotowania i realizacji różnego typu automatycznych testów wykonywanych na oprogramowaniu. Istotnym problemem jest wybór optymalnego narzędzia dostosowanego do potrzeb. W związku z tym, celem niniejszego artykułu jest analiza porównawcza trzech popularnych platform do tworzenia i przeprowadzania testów automatycz-

nych, oparta na trzech kryteriach: funkcjonalności, wydajności oraz perspektyw rozwoju oprogramowania. W badaniach zostały uwzględnione testy jednostkowe, integracyjne oraz testy typu end-to-end. Porównywane były narzędzia umożliwiające testowanie aplikacji napisanych w języku Java.

W oparciu o powyższy cel określono następujące hipotezy badawcze:
H1: JUnit5 może umożliwić uzyskanie najlepszego czasu dla testów jednostkowych o niskim poziomie skomplikowania funkcjonalnego.
H2: Zastosowanie parametryzacji testów w przypadku dużych zbiorów danych może poskutkować najlepszym wynikiem wydajnościowym dla narzędzia TestNG w stosunku do wyników uzyskanych przez pozostałe badane narzędzia.
H3: Najpopularniejszym oprogramowaniem do przygotowywania i przeprowadzania testów spośród badanych narzędzi jest JUnit5.

2. Przegląd literatury

Wdrożenie oprogramowania powinno wiązać się z dostarczeniem sprawdzonego produktu, który przechodząc w fazę utrzymania powinien cechować się stabilnością i wysoką niezawodnością. W celu spełnienia tych dwóch kryteriów rozwijane systemy są wieloaspektowo testowane. Wykorzystywane są różne techniki, narzędzia i rodzaje testów, a mnogość rozwiązań jest częstym powodem dyskusji występującym w wielu publikacjach zajmujących się tą tematyką. Przykładem pracy, poruszającej te zagadnienia jest artykuł pt. „Towards Automated Software Testing: Techniques, Classifications and Frameworks", w której autor przedstawia różne rozwiązania dla testów automatycznych [1].

Rozwijanie systemu można oprzeć na wielu różnych technikach, w których testy odgrywają istotną rolę. Publikacja pt. „Combination of test-driven development and behavior-driven development for improving backend testing performance" porusza tematy metodyk Test-Driven-Development oraz Behaviour-Driven-Development. Badania przeprowadzone przez autorów artykułu skupiają się na analizie połączenia obu tych technik w tak zwane T-BDD. Badacze z powodzeniem pokazują, że kombinacja obu tych metodyk pozwala na osiągnięcie wysokiego procentu pokrycia testami oraz elastyczności testów na zmiany parametrów metod [2].

Istnieje wiele klasyfikacji testów takich jak chociażby podział na testy manualne polegające na ręcznym sprawdzaniu poprawności zaimplementowanych funkcjonalności oraz testy automatyczne wykonywane przez specjalistyczne oprogramowanie. W pracy „Web based Automation Testing and Tools", autor w jednym z rozdziałów dokonuje porównania testów manualnych i automatycznych. Analizy przeprowadza na podstawie badań szybkości wykonania się testów, ich precyzji oraz opłacalności. Dodatkowo publikacja ta zawiera zestawienia innych rodzajów testów. Porównywane są testy białej i czarnej skrzynki oraz testy statyczne i dynamiczne [3].

Problematyka testowania automatycznego poruszana jest w bardzo wielu dyskusjach naukowych. Przykładem pracy dotyczącej tej tematyki jest artykuł pt. „A Study of Automated Software Testing: Automation Tools and Frameworks", w której autor zwraca uwagę na przewagę testów automatycznych w aspekcie szybkiego dostarczania wysokiej jakości oprogramowania [4].

Odnosząc się ponownie do kwestii testów automatycznych, istnieje inna klasyfikacja, dzieląca testy w zależności od obszaru i sposobu testowania. W związku z tym można wyróżnić izolowane testy jednostkowe, testy integracyjne sprawdzające zależności pomiędzy oddzielnymi komponentami aplikacji oraz testy end-to-end realizujące scenariusze testowe. Temat testów integracyjnych został poruszony w ramach pracy pt. „Integration testing of object-oriented software", która dotyczy tematu testowania systemów obiektowych ze szczególnym uwzględnieniem zagadnień związanych z testami integracyjnymi. W pracy tej zaprezentowano różne poziomy testowania integracji oraz przedstawiono sposoby adaptacji tradycyjnych strategii integracyjnych [5].

Współcześnie, testowanie oprogramowania najczęściej wiąże się z korzystaniem z zaawansowanych narzędzi lub szkieletów usprawniających pracę. W artykule o nazwie „Analiza i porównanie szkieletów programistycznych używanych do testów automatycznych" została przeprowadzona analiza porównawcza zestawiająca narzędzia JUnit oraz TestNG, która polegała na porównaniu składni obu frameworków oraz testach wydajnościowych. W wyniku analizy stwierdzono, że czasy uruchomienia poszczególnych przypadków testowych są podobne [6].

W przypadku testowania kodu napisanego w języku Java, najpopularniejszym narzędziem do przeprowadzania testów jest JUnit – narzędzie, które przeanalizowano w pracy pod tytułem „Automated Java Testing: JUnit versus AspectJ". W artykule tym autorzy porównują dwa popularne narzędzia do testowania automatycznego JUnit i AspectJ. Podczas analizy, badający dokonali zestawienia podobieństw obu systemów, a następnie szczegółowo porównali je pod kątem takich kryteriów jak liczba linii kodu testowego czy krzywa uczenia się. Ostatecznym wynikiem analizy było stwierdzenie, że w przypadku programowania zorientowanego aspektowo (AOP), AspectJ jest skuteczniejszy niż JUnit [7].

Kolejna analiza dotyczyła porównania frameworków TestNG oraz WebdriverIO. Została ona opisana w artykule pt. „Analiza porównawcza frameworków do automatyzacji testowania aplikacji webowych na przykładzie TestNG i WebdriverIO". W ramach tej pracy określono kryteria porównawcze odnoszące się do dwóch kwestii: przebiegu tworzenia testów oraz ich wydajności. Na podstawie uzyskanych wyników z badań wydajność w przeprowadzaniu testów narzędziem WebdriverIO okazała się być wyższa niż wydajność przeprowadzania testów we frameworku TestNG. Jeśli chodzi o kryteria ogólne, WebdriverIO okazał się być łatwiejszy we wdrożeniu [8].

W pracy „Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete" zostały przedstawione i porównane trzy frameworki wykorzystywane do tworzenia i uruchamiania testów automatycznych. Zestawione narzędzia to Selenium, Quick Test Professional oraz TestComplet. Analiza porównawcza wykonana w ramach artykułu opierała się na kryterium funkcjonalności. Według autorów publikacji, najlepszą platformą testową okazał się Quick Test Professional będąc narzędziem wszechstronnym i przystosowanym dla krytycznych i bardziej ryzykownych aplikacji [9].

3. Metody badawcze

Badania przeprowadzone zostały w oparciu o aplikację testową napisaną w języku Java. Zbiór narzędzi, które wykorzystano w badaniach obejmuje:

- szkielet programistyczny Spring Boot;
- środowisko programistyczne Intellij IDEA;
- narzędzia testowe: JUnit5, Spock, TestNG.

Środowisko testowe służące do pomiarów szybkości wykonywania testów zostało zaprezentowane w Tabeli 1.

Tabela 1: Parametry środowiska testowego

Procesor	AMD Ryzen 5 2600
Płyta główna	Gigabyte B450m DS3H
Pamięć RAM	Patriot 16GB 3200MHz CL16 Viper 4
Dysk	ADATA 512GB M.2 PCIe NVMe XPG GAMMIX S11 Pro
Karta graficzna	ASUS Radeon RX 580 Dual OC 4GB GDDR5

Aplikacja użyta na potrzeby eksperymentu posłużyła jako środowisko bazowe, dostarczające metod umożliwiających porównanie narzędzi testowych pod względem wydajności przeprowadzania testów. Każde z narzędzi zostało wykorzystane do przeprowadzenia testów na tych samych testowalnych metodach realizujących proste algorytmy oraz korzystające z dużych zasobów komputera. Takie podejście miało na celu sprawdzenie zarówno prędkości uruchamiania testu, jak i przeprowadzenia bardziej skomplikowanych operacji, przy wykorzystaniu mechanizmów dostarczanych przez dane narzędzie. Zestawienie narzędzi zostało przedstawione w postaci tabelarycznej oraz opisowej wyjaśniającej otrzymane wyniki. Testy zostały napisane przy użyciu analogicznych mechanizmów, w celu analizy wydajności poszczególnych funkcjonalności narzędzi. Każdy scenariusz został powtórzony pięciokrotnie w celu wyeliminowania błędu pomiarowego, związanego z buforem lub chwilowym opóźnieniem systemu. Na podstawie uzyskanych wyników z pięciu prób zostały obliczone: średnia, mediana i odchylenie standardowe.

Scenariusz badawczy składał się z następujących kroków:

1. instalacja oraz konfiguracja frameworka w aplikacji testowej;
2. stworzenie zbioru przykładowych testów w oparciu o dokumentację techniczną badanego narzędzia;
3. przeprowadzenie testów automatycznych z użyciem wybranego środowiska testowego;
4. zapisanie wyników czasowych wszystkich wykonanych prób;
5. analiza otrzymanych wyników: ocena skuteczności oraz szybkości na tle innych rozwiązań;
6. podsumowanie i wyciągnięcie wniosków.

Analiza funkcjonalna została zrealizowana w formie zestawienia możliwości narzędzi testowych i porównania ich, podczas którego zwrócono uwagę na wpływ oferowanych funkcjonalności na produktywność podczas procesu testowania kodu. Dodatkowo porównana została składnia oraz czytelność kodu testowego, a także ergonomia przygotowywania testów. Porównanie funkcjonalności zostało zrealizowane w odniesieniu do przygotowywania kluczowych rodzajów testów. Ze względu na brak możliwości porównania pomiarowego na rzecz zestawienia funkcjonalnego, został sporządzony szczegółowy opis i argumentacja przewagi jednego narzędzia nad innym pod względem funkcjonalnym. Całość analizy narzędzia została również przedstawiona w postaci tabelarycznej prezentującej kluczowe możliwości oferowane przez oprogramowanie (Tabela 13). Sporządzono scenariusz testowy badania funkcjonalnego składający się z następujących kroków:

1. wybór opisywanej funkcjonalności;
2. zapoznanie się z oficjalną dokumentacją środowisk testowych w kwestii możliwości oferowanych w zakresie wybranej funkcjonalności;
3. utworzenie przykładowych testów z wykorzystaniem metod oferowanych przez każde z narzędzi;
4. wykonanie symulacji testu;
5. podsumowanie i analiza wsparcia dla funkcjonalności w odniesieniu do każdego środowiska.

Porównanie popularności środowisk do testowania aplikacji zostało wykonane w formacie opisowym. W odniesieniu do popularności wyciągnięto również wnioski na temat wsparcia technicznego badanego narzędzia, aktualności oraz perspektyw rozwoju. Poniżej przedstawiono kryteria porównawcze odnoszące się do wybranych trzech serwisów internetowych, na podstawie których przygotowano analizę porównawczą.

- Github – porównanie liczby zgłoszeń, gwiazdek oraz forków, dotyczących danego frameworka;
- Stack Overflow – porównanie liczby pytań i odpowiedzi w tematach odnoszących się do badanego narzędzia;
- Google Trends – zestawienie popularności w czasie dla każdego środowiska.

4. Przebieg badań

4.1. Testy wydajnościowe

Do przeprowadzenia testów wydajnościowych została wykorzystana autorska aplikacja testowa pozwalająca na instalację narzędzi testowych za pomocą oprogramowania Maven. Program w połączeniu ze środowiskiem programistycznym umożliwiał uruchamianie

testów z poziomu kodu. Ze względu na eksperyment związany z testami typu end-to-end, w aplikacji zostały umieszczone klasy testowe: ”Room”, ”Service” oraz ”Equipment”. Posiadały one konstruktory, metody modyfikujące stan obiektów oraz umożliwiające zapis do bazy danych. Kod aplikacji testowej został użyty tylko w testach end-to-end. Pozostałe testy były autonomiczne i badały obszary, w których dane narzędzie mogło wykazywać znaczne różnice w wydajności w stosunku do innych, porównywanych platform testowych. Do celów badań opracowano pięć przypadków testowych, które zostały opisane w Tabeli 2.

Tabela 2: Scenariusze testów wydajnościowych

Przypadek testowy	**Opis**
Test jednostkowy odwracający ciąg znaków	Test został utworzony w oparciu o prostą metodę umożliwiającą generowanie palindromu z podanego tekstu. Test ma na celu sprawdzenie różnic w czasach uruchomienia narzędzi testowych. Czas samego przebiegu testu można uznać za pomijalny.
Jednostkowy test jednowątkowy wykorzystujący dużą ilość zasobów	Test sprawdzający czas wykonania operacji mocno obciążającej zasoby komputera, która wykorzystuje jeden wątek. Tymi operacjami są dwa rodzaje sortowania (bąbelkowe oraz binarne), które zostały wykonane po sobie. Algorytmy te działają na dużym zbiorze liczb. Test ma na celu sprawdzenie czy wykonanie operacji mocno obciążającej zasoby komputera jest spowalniane przez framework.
Test parametryzowany	Badanie sprawdzające czas wykonania testu z dołączoną do niego pulą danych. Ma ono na celu przeanalizowanie prędkości pobierania oraz wykorzystywania danych w ramach testu. Kod wykorzystywany na potrzeby testu polega na odwróceniu znaków w tekście zawierającym 10000 wygenerowanych wyrazów, użytych jako dane wejściowe.
Test wykorzystujący wielowątkowość	Test sprawdzający czas uruchomienia wielu testów jednocześnie z wykorzystaniem wbudowanego mechanizmu uruchamiania wielowątkowego. Test ma na celu sprawdzenie, które z porównywanych narzędzi testowych posiada najlepszą wydajność podczas jednoczesnego przeprowadzania wielu testów.
Test end-to-end	Test oparty o realną infrastrukturę sprawdzający czas wykonania identycznego scenariusza testowego dla każdego porównywanego narzędzia. Test wykorzystuje JPA, przeprowadzając operacje na bazie danych MySQL. W ramach testu tworzone są obiekty, a następnie zmieniany jest ich stan za pomocą metod, klas, których obiekty są reprezentacją. Ostatecznie obiekty zapisywane są do bazy danych.

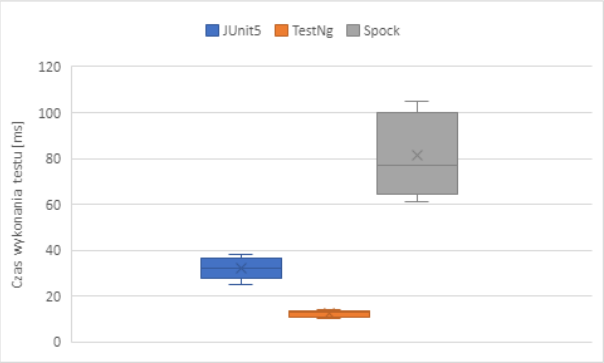
Pięciokrotny pomiar czasu wykonania testu dla każdego narzędzia, zwiększył wiarygodność przeprowadzonego badania i pozwolił na obliczenie średniej arytmetycznej, mediany oraz odchylenia standardowego. Dla każdego z przypadków testowych został przygotowany wykres pudełkowy przedstawiający rozrzut czasów wykonania, a także opis słowny otrzymanych wyników (Rysunki 1-5).

Tabela 3: Pomiary czasów dla testu odwracającego ciąg znaków

	T1 [ms]	T2 [ms]	T3 [ms]	T4 [ms]	T5 [ms]
JUnit5	25	32	35	38	30
TestNG	13	10	12	13	14
Spock	77	95	105	68	61

Tabela 4: Analiza statystyczna wyników testu odwracającego ciąg znaków

	Średnia [ms]	Mediana [ms]	SD [ms]
JUnit5	32	32	4
TestNG	12	13	1
Spock	81	77	15



Rysunek 1: Wykres pudełkowy czasów wykonania pierwszego przypadku testowego.

W pierwszym przypadku testowym, w którym porównywano czasy uruchomienia narzędzia testowego najlepiej wypadło narzędzie TestNG, a najgorzej Spock. Z wyników zawartych w Tabelach 3 i 4 oraz na Rysunku 1 można wywnioskować, że pomiędzy czasem uruchomienia JUnit5, a TestNG jest niewielka różnica, natomiast Spock uruchamia się znacznie wolniej. Średnie czasy uruchamiania narzędzia TestNG są trzy razy krótsze niż narzędzia Spock. Dla pojedynczego testu, różnica wyrażona w milisekundach może być nieodczuwalna, natomiast przy realizacji kilku tysięcy testów utworzonych podczas rozwijania dużego projektu programistycznego, różnice mogą być znaczące.

Wyniki uzyskane po zrealizowaniu drugiego scenariusza testowego (Tabela 5 i 6, Rysunek 2), w którym wykonywano operacje jednowątkowe wymagające dużej ilości zasobów, okazały się do siebie zbliżone. Różnice czasowe były w granicach błędu pomiarowego.

Oznacza to, że żadne z analizowanych narzędzi nie spowalnia wykonywania czystego kodu napisanego w języku programowania.

Tabela 5: Pomiary czasów dla testu jednowątkowego

	T1 [ms]	T2 [ms]	T3 [ms]	T4 [ms]	T5 [ms]
JUnit5	65	63	62	63	63
TestNG	65	65	64	63	63
Spock	63	61	59	62	62

Tabela 6: Analiza statystyczna wyników testu jednowątkowego

	Średnia [ms]	Mediana [ms]	SD [ms]
JUnit5	63	63	1
TestNG	64	64	1
Spock	61	62	1

Rysunek 2: Wykres pudełkowy czasów wykonania drugiego przypadku testowego.

Tabela 7: Pomiar czasów wykonania testu wykorzystującego wielowątkowość

	T1 [s]	T2 [s]	T3 [s]	T4 [s]	T5 [s]
JUnit5	19,136	20,543	15,577	15,554	18,237
TestNG	26,274	23,681	17,670	17,811	19,268
Spock	21,451	14,935	18,483	14,453	11,371

Tabela 8: Analiza statystyczna wyników testu wykorzystującego wielowątkowość

	Średnia [s]	Mediana [s]	SD [s]
JUnit5	17,809	16,907	1,711
TestNG	20,941	19,268	2,967
Spock	16,139	14,935	2,976

Rysunek 3: Wykres pudełkowy czasów wykonania trzeciego przypadku testowego.

Do osiągnięcia wielowątkowości dla JUnit5 skorzystano z pliku konfiguracyjnego umożliwiającego ustawienie trybu wielowątkowego oraz adnotacji "@Execution(ExecutionMode.CONCURRENT)". Także dla narzędzia Spock przygotowano konfigurację umożliwiającą włączenie wielowątkowości. Natomiast dla oprogramowania TestNg wykorzystano parametry "threadPoolSize" oraz "invocationCount" dla adnotacji @Test.

W przypadku badania czasów wykonania podczas uruchomienia wielu testów jednocześnie okazało się, że TestNG osiąga najgorsze wyniki. Średni czas przeprowadzenia testów dla narzędzia Spock jest około 23% krótszy względem TestNG oraz około 15% krótszy od JUnit5 (Tabele 7 i 8, Rysunek 3). Pomimo automatycznego zarządzania wątkami, ze względu na dużą liczbę zmiennych podczas przeprowadzenia tego badania widoczny jest duży rozrzut wyników.

Tabela 9: Wyniki testu parametryzowanego

	T1 [ms]	T2 [ms]	T3 [ms]	T4 [ms]	T5 [ms]
JUnit5	693	639	709	585	689
TestNG	729	670	732	792	670
Spock	324	350	313	345	314

Tabela 10: Analiza statystyczna wyników testu parametryzowanego

	Średnia [ms]	Mediana [ms]	SD [ms]
JUnit5	666	664	39
TestNG	719	729	39
Spock	329	324	13

Rysunek 4: Wykres pudełkowy czasów wykonania czwartego przypadku testowego.

W odniesieniu do testów parametryzowanych wykorzystujących kod odwracający ciąg znaków najlepiej wypadła platforma Spock, osiągając znacznie krótszy średni czas niż pozostałe dwie platformy testowe (Tabele 9 i 10, Rysunek 4). Oznacza to, że wczytywanie danych testowych oparte na parametryzacji testów jest najbardziej wydajne dla tego właśnie frameworka. Najgorzej pod względem czasowym wypadł TestNG, który jest średnio o około połowę wolniejszy od szkieletu Spock oraz nieznacznie wolniejszy od JUnit5. W przypadku JUnit5 użyto adnotacji "@ParametrizedTest" w połączeniu z "@MethodSource", dla narzędzia Spock wykorzystano adnotację "@Unroll", natomiast w dla

TestNG skorzystano charakterystycznej dla tego frameworka adnotacji "@DataProvider".

Czasy wykonania testów typu end-to-end przeprowadzonych na jednym z modułów aplikacji testowej były najkrótsze dla platform TestNG oraz JUnit5. Oba narzędzia osiągnęły wyniki na podobnym poziomie (Tabele 11 i 12, rysunek 5) i w stosunku do platformy Spock czasy były o około 120 ms krótsze. Może to wynikać z dłuższego uruchamiania się środowiska testowego i nieznacznie wolniejszego przetwarzania operacji opartych na języku Java.

Tabela 11: Wyniki testu end-to-end

	T1 [ms]	T2 [ms]	T3 [ms]	T4 [ms]	T5 [ms]
JUnit5	344	361	380	385	353
TestNG	338	391	350	342	356
Spock	494	492	466	484	499

Tabela 12: Analiza statystyczna wyników testu parametryzowanego

	Średnia [ms]	Mediana [ms]	SD [ms]
JUnit5	365	361	13
TestNG	355	350	16
Spock	487	492	10

Rysunek 5: Wykres pudełkowy czasów wykonania piątego przypadku testowego.

4.2. Testy funkcjonalne

Analiza możliwości oferowanych przez dany framework została zrealizowana w oparciu o wyspecyfikowane kryteria. Wyniki tej analizy przedstawia Tabela 13. Dodatkowo dokonano scharakteryzowania poszczególnych narzędzi, biorąc pod uwagę takie kwestie jak składnia, elastyczność oraz prostota sporządzania testów. Poza tym określono istotne wady i zalety każdego z narzędzi.

Tabela 13: Zestawienie funkcjonalności

Funkcjonalność	**JUnit5**	**TestNG**	**Spock**
Testy jednostkowe	Tak	Tak	Tak
Testy integracyjne	Tak	Tak	Tak
Asercje	Bogate wsparcie	Bogate wsparcie	Bogate wsparcie
Testy dynamiczne	Tak	Tak	Tak
Testy parametryzowane	Tak	Tak	Tak
Obsługa wyjątków	Tak	Tak	Tak
Testy równoległe	Adnotacja, plik konfiguracyjny	Adnotacja	Plik konfiguracyjny
Generowanie raportów wyników testów	Tak	Tak	Nie
Wsparcie dla tworzenia rozszerzeń	Tak	Tak	Tak
Możliwość integracji z narzędziami zewnętrznymi	Tak	Tak	Tak
Testy asynchroniczne	Tak	Tak	Nie
Możliwość zarządzania stanem testów	Tak	Tak	Tak
Możliwość wykonywania testów w kolejności	Tak	Tak	Tak
Sposób dostarczania danych testowych	Fabryki, metody, pliki	Fabryki, metody, pliki	Tabele, zbiory, metody
Wsparcie dla testów wykorzystujących interfejsy	Nie	Tak	Nie
Wsparcie dla testów wykorzystujących słuchaczy	Tak	Tak	Nie
Wsparcie dla testów wykorzystujących grupy	Tak	Tak	Tak
Wsparcie dla testów wykorzystujących wyrażenia regularne	Nie	Nie	Tak

Z Tabeli 12 wynika, że podstawowe funkcjonalności takie jak wsparcie dla testów jednostkowych, integracyjnych, parametryzowanych czy obsługa wyjątków są oferowane przez wszystkie narzędzia. Każde oprogramowanie posiada szeroki zbiór asercji. Pewne różnice w zakresie oferowanych możliwości widoczne są w przypadku narzędzia Spock. Oprogramowanie to nie wspiera testów opartych o interfejsy, testów asynchronicznych, a także nie pozwala na generowania raportów z wynikami testów. Natomiast jako jedyne spośród badanych narzędzi umożliwia przygotowywanie testów wykorzystujących wyrażenia regularne, za pomocą adnotacji. Spock nie posiada również wbudowanej adnotacji umożliwiającej dostarczanie danych z pliku, co zapewniają pozostałe dwa środowiska. Testy równoległe zarówno dla narzędzia Spock, jak i JUnit5 muszą być skonfigurowane w specjalnym pliku konfiguracyjnym. W przypadku JUnit5 możliwe jest określenie za pomocą adnotacji, które testy powinny zostać wykonane równolegle. Spock nie posiada takiej możliwości. Dodatkowo autorzy szkieletu opartego na języku Groovy określają testy równoległe jako funkcjonalność eksperymentalną dla tego narzędzia. TestNG oraz JUnit5 posiadają wsparcie dla testów asynchronicznych, natomiast tylko TestNG umożliwia tworzenie testów w oparciu o interfejsy. Dodatkowo TestNG, pozwala na zarządzanie stanem testów w największym zakresie

spośród analizowanych narzędzi. Pozwala on na wykonywanie operacji przed testami określonymi w grupie lub konkretnym pakiecie. Według tabeli 13 TestNG posiada największy zakres funkcjonalności. To narzędzie nie obsługuje jedynie testów wykorzystujących wyrażenia regularne (tzw. regexy).

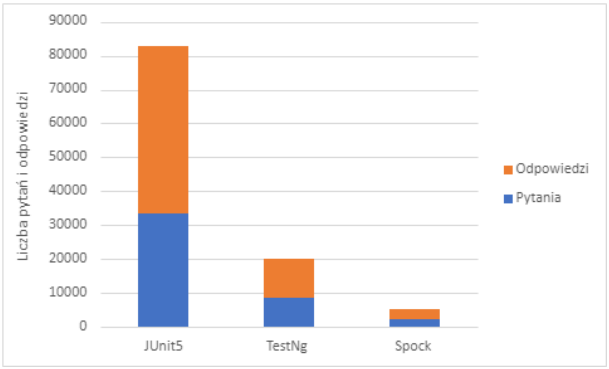
Odnośnie składni i prostoty tworzenia testów szkielety JUnit5 oraz TestNG nie posiadają ściśle zdefiniowanych ram. Komponowanie testów jest elastyczne i wymaga jedynie użycia odpowiedniej adnotacji. Przygotowywanie testów we frameworku Spock wymaga zdefiniowania klauzul given, when oraz then, a także umieszczenia, opisów obok tych klauzul zazwyczaj odnoszących się do stanu testu. Narzucenie konieczności jawnego określenia granic testów, może mieć konsekwencje w postaci podwyższenia standardu testów obecnych w aplikacji, a także może ułatwić utrzymanie kodu testowego i może być pomocne w jego zrozumieniu. Specyfika testów opracowanych w szkielecie Spock ułatwia projektowanie testów typu end-to-end ze względu na konieczność zdefiniowania poszczególnych kroków testów. W kwestii komponowania testów, wszystkie analizowane narzędzia mają podobny poziom skomplikowania.

Odnosząc się do ergonomii przygotowywania testów, JUnit5 oraz TestNG wykorzystują język Java. Ze względu na charakter języka, na którym są oparte, błędy popełnione podczas opracowywania testów są na bieżąco przetwarzane, a oprogramowanie nie pozwala na skompilowanie kodu odpowiedzialnego za realizację testu. Szkielet Spock używa języka Groovy i dlatego wyświetla błędy dopiero po zakończeniu działania testowanego kodu. Wydłuża to proces przygotowywania testów, szczególnie podczas testowania bardziej wymagających funkcjonalności aplikacji, ze względu na długi czas wykonywania kodu i wykorzystanie dużej ilość zasobów.

4.3. Testy popularności

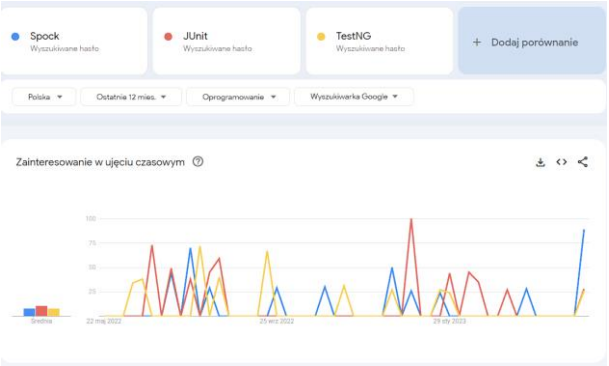
Popularność porównywanych narzędzi została przeanalizowana na podstawie aktualnych wyników zamieszczanych w trzech serwisach internetowych GitHub, StackOverflow oraz Google Trends. Analiza w serwisie GitHub, opiera się na statystykach oficjalnych projektów odpowiedzialnych za rozwój testowanego oprogramowania. Badanie na stronie StackOverflow, zostało przeprowadzone za pomocą narzędzia StackExchange umożliwiającego formułowanie zapytań SQL. Wykorzystano zapytania SQL sprawdzające liczbę zapytań i odpowiedzi w oparciu o tag reprezentujący dany framework. Analiza popularności w serwisie Google Trends została zrealizowana na podstawie liczby wyszukiwań porównywanych narzędzi w sekcji oprogramowanie oraz ich średnich wyników uwzględniających dane z ostatnich 12 miesięcy.

Według serwisu GitHub pierwsze miejsce w zestawieniu zajmuje JUnit5, osiągając wynik ponad 1300 forków, około 5700 gwiazdek oraz 110 zgłoszeń. Wybór najbardziej popularnego frameworka spośród pozostałych nie jest jednoznaczny. Kod TestNG został pobrany około 1000 razy, na jego podstawie utworzono 252 zgłoszenia, a sam framework otrzymał około 1800 gwiazdek. Szkielet Spock choć uzyskał dużo mniejszą liczbę forków, tzn. 459, to jego repozytorium zgromadziło ponad 3400 gwiazdek i 212 zgłoszeń. Te wyniki świadczą o podobnym poziomie popularności narzędzi TestNG oraz Spock w serwisie GitHub.



Rysunek 6: Wykres porównujący popularność narzędzi na platformie StackOverflow.

Przyglądając się wykorzystaniu tagów na platformie StackOverflow (Rysunek 6) można zauważyć, że sprawdzając liczbę zapytań i odpowiedzi dla tagu “junit” oraz tagu “junit5” uzyskamy zupełnie różne wyniki. Dla tagu bez podanej wersji frameworka wyniki wynoszą aż 33601 pytań oraz 49170 odpowiedzi, natomiast dla narzędzia z podaną wersją jest to zaledwie 3499 pytań oraz 4125 odpowiedzi. Spowodowane jest to tym, że tag “junit” używany jest dla wątków dotyczących wcześniejszych wersji szkieletu JUnit i taki wysoki wynik tagu “junit” ma przełożenie na wybór szkieletu JUnit5 jako najbardziej popularnego na platformie StackOverflow. Drugie miejsce zajmuje TestNG z wynikiem 8676 pytań oraz 11540 odpowiedzi, a na ostatniej pozycji plasuje się framework Spock z wynikiem 2376 pytań oraz 2900 odpowiedziami.



Rysunek 7: Wykres porównujący popularność narzędzi w aplikacji Google Trends.

Analizując wyniki przedstawione przez serwis Google Trends (Rysunek 7) jako odpowiedź na żądanie porównania liczby wyszukiwań nazw analizowanych trzech narzędzi można stwierdzić, że i w tym przypadku ponownie JUnit5 jest najpopularniejszym narzędziem, a średnia popularności dwóch pozostałych narzędzi

w okresie ostatnich 12 miesięcy była na podobnym poziomie.

Biorąc pod uwagę powyższe analizy, można stwierdzić, że spośród badanych narzędzi JUnit5 jest najpopularniejszym szkieletem programistycznym do przeprowadzania testów automatycznych. Na drugim miejscu znajduje się TestNG, natomiast ostatnie miejsce przypada frameworkowi Spock. Uwzględniając oficjalną dokumentacje techniczną, liczbę zgłoszeń wraz z odpowiedziami oraz regularne aktualizacje, można stwierdzić, że każde z wyżej wymienionych narzędzi posiada bogate wsparcie techniczne oraz ma na wysokim poziomie prowadzoną dokumentację.

5. Wnioski

Analiza porównawcza narzędzi do przeprowadzania testów automatycznych wykazała wady i zalety każdego z nich. W kwestii wydajności szkieletów do testów trudno określić, który z nich jest najlepszy. Przeprowadzone rozważania wykazały, że najszybciej uruchamia się TestNG, natomiast najwolniej Spock. Problemem TestNG okazały się testy wielowątkowe, które wypadły gorzej na tle pozostałych dwóch narzędzi, a w przypadku Spock testy end-to-end wykonywały się średnio o około 30 procent dłużej. Natomiast testy parametryzowane oparte o dostarczanie danych za pomocą mechanizmów wbudowanych we framework Spock wykonywały się w czasie o połowę krótszym niż w przypadku narzędzi JUnit5 i TestNG. Należy zwrócić uwagę, że możliwe jest uzyskanie lepszych czasów realizacji testów poprzez zastosowanie bardziej zaawansowanych technik, aczkolwiek założeniem autorów pracy były badania wykorzystujące wyłącznie podstawowe mechanizmy dostarczane przez każde badane oprogramowanie.

Porównanie funkcjonalne, pozwoliło stwierdzić, że TestNG posiada największy zasób wybranych funkcjonalności spośród testowanych narzędzi. Narzędzie to wspiera prawie wszystkie rodzaje testów wymienionych w Tabeli 13 oprócz testów opartych na wyrażeniach regularnych. Framework ten posiada również bogate wsparcie dla dostarczania danych, a także najbogatszy zasób adnotacji umożliwiających zarządzanie stanem testów. Wypada on również bardzo dobrze w przypadku ergonomii wytwarzania testów. Ze względu na wykorzystywanie języka Java, jest on bardziej praktyczny w przypadku przygotowywania dużej ilości testów lub testów odwołujących się do kodu, który do wykonania wymaga dużej ilości zasobów. Jeśli chodzi o wykorzystywaną strukturę i składnię podczas przygotowywania oraz utrzymania testów, najlepiej wypadł szkielet Spock, który narzuca sztywne ramy, które wytwarzane testy muszą spełniać, aby mogły się wykonać. Każde z porównywanych narzędzi charakteryzuje się prostotą przygotowywania testów. Uwzględniając wady oraz zalety każdego z powyższych frameworków najbardziej funkcjonalnym oprogramowaniem okazał się TestNG, który oferuje najwięcej możliwości bez dodatkowego doinstalowania (ang. out of the box).

Analizując wyniki popularności wśród trzech wybranych znanych serwisów internetowych można jednoznacznie stwierdzić, że JUnit5 jest najpopularniejszym narzędziem do przygotowywania i przeprowadzania testów. W każdym z badanych serwisów osiągnął najwyższy wynik. Przekłada się to na ilość poruszonych problemów programistycznych, co wiąże się z szybkością znajdowania rozwiązań w internecie. W konsekwencji może to przyspieszyć proces wytwarzania testów, ze względu na duże wsparcie, mnogość różnych wątków i szybszy dostęp do wiedzy na temat specyficznych zagadnień.

Wyniki uzyskane podczas przeprowadzonych badań pozwoliły na weryfikację postawionych na wstępie hipotez. Okazało się, że narzędzie JUnit5 nie jest najwydajniejszym frameworkiem do przeprowadzania prostych testów jednostkowych. Użycie TestNG może przynieść lepsze czasy wykonywania prostych testów ze względu na czas uruchomienia narzędzia. Oznacza to, że pierwsza hipoteza nie została potwierdzona. Także druga hipoteza nie została spełniona, ponieważ wyniki testów wydajnościowych pokazały, że TestNG nie jest najszybszym szkieletem wykonującym testy parametryzowane. Natomiast ostatnia hipoteza została potwierdzona - popularność frameworka JUnit5 okazała się jednoznacznie największa.

Wybór konkretnego narzędzia zazwyczaj podyktowany jest jego zastosowaniem oraz umiejętnościami osób przygotowujących testy. Wydajność może mieć znaczenie w odniesieniu do projektów programistycznych, na które składają się setki lub tysiące testów. Wówczas użycie nieoptymalnego rozwiązania może spowolnić proces wytwarzania i dalszego rozwoju oprogramowania. Dla większości systemów wybór technologii opiera się na przeanalizowaniu możliwości funkcjonalnych. W przypadku zapotrzebowania na utrzymanie określonej struktury testów najlepszym rozwiązaniem będzie Spock. Z kolei, kiedy ważniejsza będzie ergonomia przygotowywania dużej ilości testów, to ze względu na natychmiastowe wychwytywanie błędów, proces testowania może przyśpieszyć któryś z frameworków opartych na języku Java.

Literatura

[1] R. Torkar, Towards Automated Software Testing: Techniques, Classifications and Frameworks (PhD dissertation, Blekinge Institute of Technology) (2006).

[2] I. B. K. Manuaba, Combination of test-driven development and behavior-driven development for improving backend testing performance. Procedia Computer Science 157 (2019) 79-86.

[3] M. Sharma, R. Angmo, Web based automation testing and tools. International Journal of Computer Science and Information Technologies 5(1) (2014) 908-912.

[4] M. A. Umar, C. Zhanfang, A study of automated software testing: Automation tools and frameworks. International Journal of Computer Science Engineering (IJCSE) 6 (2019) 217-225.

[5] A. Orso, Integration testing of object-oriented software. Dottorato di Ricerca in Ingegneria Informatica e Automatica, Politecnico di Milano, 1998.

[6] D. Gromek, D. Gutek, Analysis and comparison of programming frameworks used for automated tests. Journal of Computer Sciences Institute 17 (2020) 339-344.

[7] M. Jain, D. Gopalani, Automated Java testing: JUnit versus AspectJ. International Journal of Computer and Information Engineering 11(11) (2017) 1215-1220.

[8] A. Shtokal, J. Smołka, Comparative analysis of frameworks used in automated testing on example of TestNG and WebdriverIO. Journal of Computer Sciences Institute 19 (2021) 100-106.

[9] H. Kaur, G. Gupta, Comparative study of automated testing tools: selenium, quick test professional and testcomplete. Int. Journal of Engineering Research and Applications 3(5) (2013) 1739-1743.

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 383–390
Received: 3 July 2023
Accepted: 17 September 2023

Comparison of application container orchestration platforms

Porównanie platform do orkiestracji kontenerów aplikacji

Adam Pankowski*, Paweł Powroźnik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

This article presents a comparative analysis of three well-known container orchestration platforms: Docker Swarm, Kubernetes and Apache Mesos, focusing on the deployment of a test application and measuring parameters such as deployment time, CPU, memory and disk utilization, application response time and the time to restore a replica of the application using an auto-recovery mechanism. The aim of the research is to verify the performance and efficiency of the analyzed platforms, facilitating informed decisions while choosing an orchestrator for containerized applications. Two research hypotheses have been stated. The first one assumes that the time required to launch an application using the Docker Swarm tool is the shortest among the analyzed platforms. The second hypothesis is that Kubernetes provides the most efficient results in terms of load scheduling and application scaling. The analysis performed on the Jenkins application showed the superiority of the Docker Swarm platform over the other studied tools in terms of performance.

Keywords: contenerization; Docker Swarm; Kubernetes; Apache Mesos

Streszczenie

Niniejszy artykuł przedstawia analizę porównawczą trzech znanych platform orkiestracji kontenerów: Docker Swarm, Kubernetes i Apache Mesos, koncentrując się na wdrażaniu aplikacji testowej oraz pomiaru takich parametrów jak: czas wdrożenia, obciążenie procesora, wykorzystanie pamięci i dysku, czas odpowiedzi aplikacji oraz czas przywrócenia repliki aplikacji przy użyciu mechanizmu autoregeneracji. Celem badań jest weryfikacja wydajności i efektywności analizowanych platform, ułatwiając podjęcie świadomych decyzji przy wyborze orkiestratora dla skonteneryzowanych aplikacji. Zostały postawione dwie hipotezy badawcze. Pierwsza z nich zakłada, że czas potrzebny na uruchomienie aplikacji przy użyciu narzędzia Docker Swarm jest najkrótszy spośród analizowanych platform. Druga hipoteza zakłada, że Kubernetes zapewnia najwydajniejsze wyniki pod względem planowania obciążenia i skalowania aplikacji. Analiza przeprowadzona na podstawie wykonanych badań na obrazie aplikacji Jenkins wykazała przewagę platformy Docker Swarm nad pozostałymi badanymi narzędziami pod względem wydajnościowym.

Słowa kluczowe: konteneryzacja; Docker Swarm; Kubernetes; Apache Mesos

*Corresponding author

Email address: adam.pankowski@pollub.edu.pl (A. Pankowski)

1. Wstęp

Coraz więcej firm i organizacji decyduje się na zastosowanie systemu złożonego z mikroserwisów zamiast tradycyjnej architektury monolitycznej [1]. Budowę takiego systemu można zdefiniować w kontekście architektury zorientowanej na usługi jako kompozycję małych, rozproszonych i luźno powiązanych ze sobą bloków konstrukcyjnych, stanowiących komponenty oprogramowania [2]. Struktura ta polega na budowaniu autonomicznych komponentów, które używane są jako bloki do konstruowania złożonych systemów. Każda aplikacja składa się z niezależnych usług, które współdziałają ze sobą poprzez interfejsy programowania aplikacji (API). Wynika to z szybkiego tempa rozwoju technologii we współczesnym świecie. Z drugiej strony, wirtualizacja umożliwia budowanie maszyn wirtualnych, co pozwala na uruchamianie wielu systemów operacyjnych na jednym fizycznym hoście. Konteneryzacja to nie tylko technologie ułatwiające pracę administratorom i programistom, ale także sposób na zwiększenie produktywności i oszczędności dla firm. Rozwiązania te ułatwiają migrację aplikacji pomiędzy środowiskami oraz umożliwiają szybsze skalowanie i uruchamianie aplikacji, ponieważ konteneryzacja pozwala na oddzielenie aplikacji od zależności systemu operacyjnego. Konteneryzacja odgrywa znaczącą rolę w świecie komercyjnym. Skonteneryzowane aplikacje mogą być skalowane w górę lub w dół w zależności od potrzeb. Dzięki elastyczności i krótkiego czasu reakcji na potrzeby rynku kontenery zrewolucjonizowały sposób wdrażania i zarządzania poprzez redukcję złożoności aplikacji i jej zależności, jak również możliwości zbudowania na jej podstawie pojedynczego obrazu kontenera. Dzięki czemu programiści mogą w łatwy sposób dystrybuować swoje programy i uruchamiać je w dowolnym środowisku. Podejmowanie decyzji o wyborze narzędzi i technologii do wykorzystania w różnych kontekstach wymaga ciągłej oceny i analizy możliwości, które są do dyspozycji. Celem niniejszego artykułu jest zbadanie trzech znanych i stosowanych rozwiązań konteneryzacji: Kubernetes, Docker Swarm i Apache Mesos. Analiza dotyczy takich parametrów jak: czas wdrożenia, wykorzystanie pamięci, procesora i dysku, czasu odpowiedzi aplikacji oraz czasu przywrócenia repliki aplikacji przy użyciu mechanizmu autoregeneracji.

W pracy postawiono następujące hipotezy badawcze:

H1. Czas potrzebny na uruchomienie aplikacji za pomocą narzędzia Docker Swarm jest najkrótszy.

H2. Kubernetes zapewnia najlepsze wyniki pod względem planowania obciążenia aplikacji oraz skalowania.

2. Przegląd literatury

Metody konteneryzacji stale się rozwijają, szukając bardziej wydajnych rozwiązań. Dokonano analizy artykułów skupiających się na porównaniu wybranych narzędzi pod względem wdrażania oraz zarządzania kontenerami, a także te, prezentujące innowacyjne rozwiązania tym zakresie.

W artykule [3] badano orkiestratory Docker Swarm, Kubernetes, Apache Mesos oraz Cattle pod względem czasu dostarczenia aplikacji o różniej złożoności oraz skalowalności kontenerów. Do badań wybrane zostały aplikacje: Jenkins na jednym kontenerze, WordPress z dwoma kontenerami i GitLab składający się z czterech kontenerów. Eksperyment wykonano 33 razy. Zebrane wyniki eksperymentalne pokazują, że Kubernetes przewyższa swoje odpowiedniki w przypadku bardzo złożonych wdrożeń aplikacji, podczas gdy inne orkiestratory mogą być lepszym wyborem dla prostszych rozwiązań.

Artykuł [4] przedstawił analizę Kubernetes, Docker Swarm, Mesos i Redhat OpenShift pod kątem różnych parametrów bezpieczeństwa, wdrożenia, stabilności, skalowalności, instalacji klastra oraz czasu nauki obsługi. Zaobserwowano, że Kubernetes ma najlepsze funkcje planowania, podczas gdy Docker Swarm jest łatwy w użyciu. Stwierdzono również dobrą skalowalność orkiestratora Mesos, podczas gdy OpenShift jest wysoce bezpiecznym narzędziem orkiestracji.

W kolejnym artykule [5] skupiono się na ocenie kosztów wydajności za pomocą znanych narzędzi porównawczych. Podjęto szereg badań z wykorzystaniem dobrze znanych narzędzi takich jak Phoronix Test Suite i LiDAR Data Benchmarks, służących do testów porównawczych. Przeprowadzono analizę wydajności narzędzi do orkiestracji kontenerów biorąc pod uwagę ich wady i zalety. Wyniki pokazują, że wydajność Kubernetes jest nieco gorsza niż Docker w trybie Swarm. Jednak Docker w trybie Swarm nie jest tak elastyczny i wydajny jak Kubernetes w bardziej złożonych środowiskach.

Artykuł [6] zawiera analizę porównawczą implementacji „Container Storage Interface" (CSI) typu „bare-metal", która pokazuje zalety i wady istniejących implementacji CSI, na podstawie, której uważa się, że podejście „bare-metal" jest najbardziej wydajne. Posiada ono jednak wiele błędów, które wymagają dalszej pracy nad rozwiązaniem. Istnieje więc potrzeba rozszerzenia CSI na „bare-metal storage provisioning", aby uniknąć kosztów wirtualizacji i chmury oraz zminimalizować ręczne operacje zarządzania pamięcią masową.

W artykule [7] autorzy przedstawili, że obecnie Docker Swarm posiada trzy podstawowe strategie harmonogramowania (spread, binpack i random), każda z nich uruchamia kontener z ustaloną liczbą zasobów. Nowość strategii prezentowanej w artykule polega jednak na wykorzystaniu klasy „Service Level Agreement" (SLA) użytkownika do dostarczenia zasobów kontenera, który musi wykonać usługę, na podstawie dynamicznego obliczenia liczby rdzeni CPU, które muszą być przydzielone kontenerowi zgodnie z klasą SLA użytkownika i obciążeniem maszyn równoległych w infrastrukturze. Testy nowej strategii zostały przeprowadzone, poprzez emulację, na różnych częściach ogólnego szkieletu i pokazują potencjał podejścia do dalszego rozwoju.

Technologia Apache Mesos jest bliżej przedstawiona w artykule [8]. Ma ona na celu zapewnienie wysokiego wykorzystania klastra poprzez ziarnisty podział i sprawiedliwość przyznawania zasobów wśród wielu użytkowników poprzez alokację opartą na „Dominant Resource Fairness" (DRF). Technologia ta bierze pod uwagę różne typy zasobów, procesor, pamięć czy dysk, wymagane przez każdą aplikację i określa udział każdego zasobu klastra, który może być przydzielony aplikacjom. Mesos przyjął dwupoziomową politykę szeregowania: DRF do przydzielania zasobów między konkurencyjnymi strukturami aplikacyjnymi oraz szeregowanie na poziomie zadań przez każdą strukturę aplikacyjną dla zasobów przydzielonych w poprzednim kroku.

W artykule [9] proponowany jest system pośrednictwa „Workload aware Energy Efficient Container" (WEEC), aby zaoszczędzić zużycie energii spowodowane przez uruchomione aplikacje kontenerowe, jednocześnie gwarantując akceptowalny poziom wydajności. Dodatkowo, przedstawiono heterogeniczność mocy i wydajność kilku serwerów kontenerowych Docker na podstawie wyników eksperymentalnych uzyskanych z urządzeń do pomiaru mocy. W ten sposób zidentyfikowano korzyści wynikające z wydajności proponowanego systemu.

Na podstawie przedstawionego przeglądu literatury można zauważyć, że brakuje szczegółowej analizy dotyczącej wydajności orkiestratorów uwzględniającej takie parametry jak: czas wdrożenia, obciążenie procesora, wykorzystanie pamięci, czas odpowiedzi aplikacji oraz czas przywrócenia repliki aplikacji przy użyciu mechanizmu autoregeneracji. Ponadto badania przeprowadzone w wymienionych artykułach zostały przeprowadzone na starszych wersjach oprogramowania.

3. Narzędzia orkiestracji kontenerów

Docker Swarm to pierwsze omawiane narzędzie do zarządzania kontenerami. Jedną z głównych cech wyróżniających tę technologię jest jej prostota konfiguracji i uruchamiania. Dzięki temu, że jest zintegrowana z Docker, nie wymaga dodatkowych instalacji ani zewnętrznych narzędzi. Tworzenie i zarządzanie klastrami za pośrednictwem Docker Swarm jest najprostsze spośród badanych orkiestratorów. Zapewnia zautomatyzowane równoważenie obciążenia w klastrach Docker, podczas gdy inne narzędzia do orkiestracji kontenerów wymagają działań manualnych [10]. Docker Swarm oferuje elastyczność w zależności od potrzeb użytkownika. Może działać zarówno w trybie jednego węzła,

gdzie wszystkie kontenery uruchamiane są na pojedynczej maszynie, jak i w trybie wielu węzłów, gdzie kontenery są rozproszone na różnych maszynach, tworząc klaster. Możliwe jest wtedy skalowanie aplikacji i dostosowywanie klastra do potrzeb organizacji. Narzędzie to jest kompatybilne z systemem Docker, co oznacza, że korzysta z tych samych narzędzi, interfejsów i obrazów kontenerowych. Dzięki temu użytkownicy pracujący z Dockerem mogą łatwo przenieść swoje aplikacje do klastra Swarm. Zapobiega on awariom dzięki współistnieniu wielu węzłów zarządzających w klastrze, aby odzyskać sprawność po awarii bez żadnych przestojów. Gdy istniejący lider nie działa lub nie jest dostępny, grupa wybierze nowego do prowadzenia zadań orkiestracji. Wiele firm, które korzystają z technologii kontenerowej, nadal z powodzeniem stosuje Docker Swarm do wdrażania i zarządzania swoimi aplikacjami [4]. Docker Swarm jest szeroko stosowany w małych i średnich przedsiębiorstwach oraz w zastosowaniach, gdzie prostota i łatwość konfiguracji są kluczowymi czynnikami, a ekosystem nie jest zaawansowany do tego stopnia, żeby potrzebne były bardziej wyspecjalizowane narzędzia.

Drugim analizowanym narzędziem jest Kubernetes będący jednym z najpopularniejszych i najdynamiczniej rozwijających się projektów w dziedzinie konteneryzacji i orkiestracji aplikacji [11]. Od momentu, gdy został udostępniony jako otwarte oprogramowanie zyskał ogromną popularność i zdobył wsparcie wielu wiodących firm technologicznych na całym świecie. Wielu gigantów branży technologicznej, takich jak Google, Microsoft, Amazon, IBM, Intel, Cisco czy Red Hat aktywnie wykorzystuje i rozwija tę technologię. Firmy te wnoszą znaczący wkład w rozwój projektu, udostępniając narzędzia, usługi chmurowe, integracje oraz innowacje. Kubernetes, znany również jako „K8s" jest systemem zarządzania kontenerami, który został opracowany przez Google na podstawie ich wewnętrznego systemu zarządzania kontenerami, znanego jako Borg. Jest to platforma do automatycznego wdrażania, skalowania i zarządzania aplikacjami kontenerowymi. Kubernetes integruje technologie i narzędzia, takie jak kontenery, automatyzację operacji i zarządzanie zasobami. Pozwala na scentralizowane i automatyczne wdrażanie aplikacji w kontenerach, skalowanie aplikacji w zależności od obciążenia oraz zapewnia jednolite środowisko uruchomieniowe niezależnie od infrastruktury. Podobnie jak Docker Swarm, Kubernetes również został opracowany w języku Go, a wdrażanie kontenerów również opiera się na wykorzystaniu języka serializacji YAML [12]. Współpraca wielu gigantów na rynku z Kubernetes świadczy o szerokim zainteresowaniu i uznaniu dla tej technologii, która stała się standardem przemysłowym w dziedzinie orkiestracji kontenerów. Dzięki temu Kubernetes nie tylko rozwija się intensywnie, ale także posiada bogaty ekosystem i wsparcie ze strony wiodących graczy na rynku technologicznym.

Apache Mesos to otwarte oprogramowanie, które ułatwia orkiestrację zasobów w środowiskach rozproszonych. Posiada system zarządzania, który może efektywnie wykorzystywać i udostępniać zasoby obliczeniowe w klastrze. Jedną z najważniejszych cech Mesos jest możliwość współdzielenia zasobów pomiędzy różnymi aplikacjami. Pozwala to na uruchamianie różnych typów aplikacji na tym samym klastrze, poprawiając wykorzystanie zasobów i wydajność. Mesos podobnie jak inne platformy do zarządzania kontenerami umożliwia efektywne wykorzystanie dostępnych zasobów i automatyczne skalowanie aplikacji w zależności od potrzeb. Działa on w modelu „master-service", w którym węzeł główny pełni rolę koordynatora, a węzły podrzędne zapewniają zasoby do wykonywania zadań. Może obsługiwać aplikacje kontenerowe, takie jak Docker, a także aplikacje na wirtualnych maszynach lub te działające bezpośrednio na systemie operacyjnym hosta. Rozwiązania z użyciem klastra Mesos są często wspomagane szkieletem zarządzającym Marathon. Marathon współdziała z komponentem głównym, zapewniając orkiestrację dla całego klastra Mesos. Tak więc, w przypadku błędu węzła podrzędnego, Marathon uruchamia nową instancję, aby zagwarantować tolerancję błędów [4]. Apache Mesos jest szeroko stosowany w dużych i złożonych środowiskach, w których konieczne jest zarządzanie dużymi zasobami obliczeniowymi. Jest używany przez wiele znanych firm i organizacji, takich jak Twitter, Airbnb, Apple i Netflix do skalowania i zarządzania ich infrastrukturą.

4. Metody badawcze

Celem badań było przeprowadzenie analizy porównawczej narzędzi do orkiestracji kontenerów aplikacji oraz zbadanie, który orkiestrator spełnia dane zadanie najlepiej. Badanie przeprowadzono dla narzędzi Docker Swarm, Kubernetes i Apache Mesos. Metryki obejmują czas potrzebny na uruchomienie aplikacji dla każdego z orkiestratorów. Następnie zbadano wykorzystanie zasobów sprzętowych z uruchomioną aplikacją testową, przepustowość dla nagłego wzrostu żądań oraz czas przywrócenia repliki po nieoczekiwanej awarii. Badania zostały przeprowadzone z użyciem oprogramowania o otwartym kodzie Jenkins.

4.1. Pomiar czasu uruchomienia aplikacji

Scenariusz badawczy dotyczący pomiaru czasu uruchomienia aplikacji zakłada monitorowanie procesu tworzenia kontenerów aplikacji oraz mierzenie czasu od rozpoczęcia tego procesu do momentu, kiedy aplikacja jest w pełni gotowa do działania. Badanie zostało przeprowadzone dla dwóch różnych przypadków, z uwzględnieniem jednej oraz pięciu replik aplikacji, aby zbadać różnicę w czasie uruchomienia w zależności od liczby replik. W celu dokładnego pomiaru, zastosowano analizę logów z poszczególnych kontenerów zawierających aplikację. Logi dostarczyły niezbędnych informacji dotyczących kolejnych etapów procesu uruchamiania, takich jak inicjalizacja kontenera, pobieranie obrazu aplikacji, konfiguracja środowiska, instalacja zależności, aż do momentu, gdy aplikacja jest w pełni gotowa do obsługi żądań. Poprzez analizę tych logów, możliwe było precyzyjne zarejestrowanie momentu, w

którym każda replika aplikacji osiągnęła stan gotowości. Czas uruchomienia został zdefiniowany jako różnica pomiędzy momentem rozpoczęcia tworzenia kontenera a momentem, kiedy aplikacja była w pełni gotowa do obsługi żądań. W celu zredukowania przypadkowości wyników, badanie powtórzono 33 razy.

4.2. Zbadanie obciążenia procesora i wykorzystanie pamięci.

Drugim scenariuszem badawczym było zbadanie obciążenia procesora i wykorzystanie pamięci podczas pracy orkiestratora. W celu dokładnego pomiaru, przeprowadzono badania dla dwóch różnych przypadków – z jedną i pięcioma replikami aplikacji. Aby przeprowadzić badania dla środowisk Docker Swarm Kubernetes oraz Apache Mesos, skorzystano z aplikacji Docker Desktop. Udostępnia ona metryki dotyczące wykorzystania pamięci, procesora i dysku bez konieczności dodatkowej instalacji lub konfiguracji narzędzi monitorujących. Dodatkowo, dla Apache Mesos, metryki dotyczące obciążenia procesora i wykorzystania pamięci zostały odczytane również z wbudowanego interfejsu Mesos Web UI [13]. W wyniku tego, można było monitorować obciążenie tych podzespołów w czasie rzeczywistym i zbierać dane do analizy. Badanie zostało wykonane 33 razy aby zmniejszyć losowość wyników. Aplikacja Jenkins z zainstalowanymi polecanymi wtyczkami zużywa znaczącą ilość pamięci podczas pracy. Podczas analizy wyników można łatwo zauważyć jak każdy orkiestrator radzi sobie z optymalizacją podczas replikacji aplikacji.

4.3. Zbadanie przepustowości i niezawodności.

Kolejnym scenariuszem badawczym było zbadanie przepustowości i niezawodności systemu poprzez testy wydajnościowe z użyciem pakietów danych. Metoda ta polegała na wysyłaniu dużej liczby pakietów danych przez system i mierzeniu ilości danych, które zostały prawidłowo przesłane. W tym celu użyto narzędzia Apache JMeter [14]. Umożliwia ono symulowanie dużej liczby użytkowników i generowanie intensywnego ruchu w celu przetestowania przepustowości systemu. W aplikacji JMeter zastosowano scenariusz testowy, który zawierał wejście na stronę powitalną badanej aplikacji oraz odczytanie tekstu z głównego widoku. W tym celu zastosowano 2000 hostów, które powtarzały operację 30 razy, co łącznie dawało 60000 operacji testowych. Liczba hostów została dobrana w celu symulacji dużego obciążenia systemu, często występującego w rzeczywistych warunkach użytkowania. Podczas przeprowadzania testów, zbierano dane dotyczące liczby prawidłowo przesłanych pakietów danych, jak i tych które nie powiodły się. Celem tego badania było zbadanie, jak system radzi sobie z dużą liczbą operacji i czy jest w stanie utrzymać odpowiednią przepustowość przy intensywnym ruchu.

4.4. Pomiar skuteczności mechanizmów autoregeneracji narzędzi.

Ostatnim scenariuszem badawczym jest pomiar skuteczności mechanizmów autoregeneracji orkiestratorów. Celem badania było sprawdzenie, jak każdy z orkiestratorów radzi sobie z błędem krytycznym, polegającym na zatrzymaniu jednej z replik aplikacji. Obserwowano, jak szybko orkiestrator podejmuje działania w celu przywrócenia usługi. Czas, który upłynął od momentu zatrzymania repliki do momentu, kiedy nowa replika została uruchomiona i aplikacja była ponownie w pełni dostępna został odczytany i zarejestrowany. Pomiar został wykonany za pomocą odczytania dokładnego czasu inicjalizacji nowej repliki w logach Docker, przeprowadzony został 33 razy w celu zredukowania błędów.

4.5. Środowisko testowe

Tabela 1: Środowisko testowe

Środowisko	
Procesor	Intel Core i7-13700KF
Ilość rdzeni	8
Pamięć RAM	32 GB
System	Ubuntu 20.04
Dysk	1 TB SSD
Karta sieciowa	Realtek GbE Network Adapter

Tabela 2: Narzędzia testowe

Narzędzie	wersja
Docker Swarm	23.0.5
Kubernetes	1.25.9
Apache Mesos	1.11.0
Apache JMeter	5.5
Docker Desktop	4.19.0

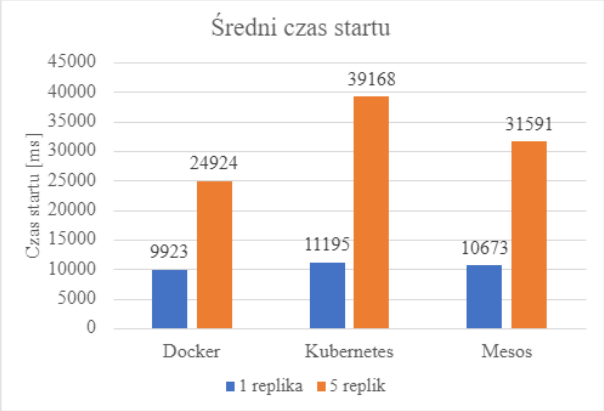
W tabelach 1 i 2 przedstawiono informacje dotyczące środowiska testowego, takie jak specyfikacje sprzętowe, wersje systemu operacyjnego oraz wykorzystane narzędzia. Wszystkie testowane narzędzia orkiestracji kontenerów: Docker Swarm, Kubernetes oraz Apache Mesos korzystały z demonów dockerd, które pełniły rolę silników kontenerów.

5. Wyniki badań

5.1. Pomiar czasu startu

Na rysunku 1 przedstawiono średnie czasy uruchomienia aplikacji dla każdego z orkiestratorów. Celem tego badania było zbadanie wydajności i efektywności trzech popularnych orkiestratorów: Docker Swarm, Kubernetes i Apache Mesos. Badanie przeprowadzono dla jednej oraz pięciu replik 33 razy. W celu prezentacji dokładnych pomiarów czas został podany w milisekundach (Rysunek 1). Na podstawie wyników z Rysunku 1 moż-

na zauważyć, że Docker osiągnął najkrótszy czas startu aplikacji w porównaniu do Kubernetes i Mesos dla pojedynczej repliki, co potwierdza hipotezę H1. Wraz ze wzrostem liczby replik, wszystkie trzy orkiestratory miały wydłużony czas startu aplikacji. Jednak nadal Kubernetes i Mesos wykazywały dłuższe czasy uruchamiania niż Docker. Spowodowane jest to natywną naturą tej technologii dla zastosowanego silnika konteneryzacji.



Rysunek 1: Średnie czasy startu aplikacji.

Z tabel 3 i 4 można odczytać mediany oraz odchylenia standardowe czasu startu aplikacji zarówno dla jednego jak i pięciu replik. Mediana każdego pomiaru jest zbliżona do wartości średniej co potwierdza stabilność narzędzi dla przeprowadzonych testów. Odchylenia standardowe z niskimi wartościami pokazują, że wyniki scenariuszy były do siebie zbliżone.

Tabela 3: Mediana i odchylenie standardowe czasu startu aplikacji dla jednej repliki

	Mediana [ms]	Odchylenie standardowe [ms]
Docker	9786,00	442,45
Kubernetes	11258,00	219,76
Mesos	10589,50	407,84

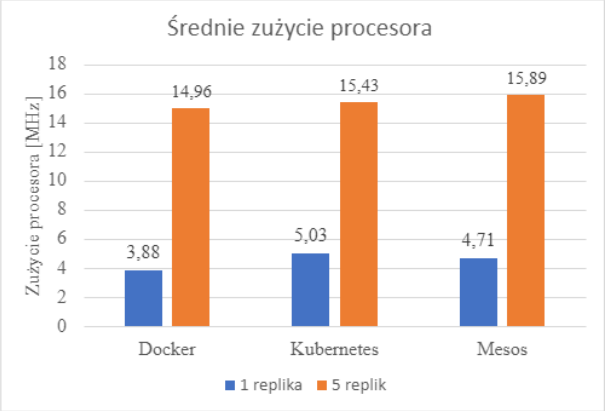
Tabela 4: Mediana i odchylenie standardowe czasu startu aplikacji dla pięciu replik

	Mediana [ms]	Odchylenie standardowe [ms]
Docker	22493,50	1367,21
Kubernetes	39721,00	1441,33
Mesos	30493,50	1167,21

5.2. Pomiar obciążenia podzespołów

Następne badanie skupiało się na pomiarze obciążenia podzespołów takich jak procesor i pamięć RAM przez każdy z orkiestratorów z uruchomioną aplikacją testową. Obciążenie procesora zostało podane w MHz, aby otrzymać precyzyjne dane na temat obciążenia podczas działania aplikacji testowej. Zużycie pamięci RAM ze względu na specyfikację testowanej aplikacji podano w GB. Na podstawie badania można było ocenić, jak dużo zasobów potrzebował zaalokować każdy orkiestrator, aby obsłużyć aplikację w sposób efektywny. Dzięki przeprowadzeniu testu 33 razy dla każdego scenariusza, możliwe było uzyskanie optymalnych wyników oraz zapewniło wiarygodność obserwacji.

Na rysunku 2 przedstawiono diagram średniego obciążenia procesora podczas pracy aplikacji. W przypadku jednej repliki podobnie jak czas startu aplikacji Docker Swarm uzyskał najmniejsze zużycie procesora dzięki bliskiej współpracy z silnikiem konteneryzacji Docker. Kubernetes i Mesos uzyskały wyniki zbliżone do siebie, lecz już nakład infrastruktury wymagany przez te platformy jest bardziej widoczny. W przypadku pięciu replik wszystkie trzy narzędzia uzyskały bardzo podobne wyniki. Biorąc pod uwagę przebieg badań jednej repliki można stwierdzić, że Kubernetes najlepiej poradził sobie ze skalowaniem replik aplikacji co potwierdza hipotezę H2. Niemniej jednak to Swarm po raz kolejny uzyskał najmniejszą wartość obciążenia procesora.



Rysunek 2: Średnie zużycie procesora.

Tabela 5 i 6 zawiera miarę median i odchylenia standardowe dla wykorzystania pamięci przez testowaną aplikację dla jednej i pięciu replik.

Tabela 5: Mediana i odchylenie standardowe obciążenia procesora dla jednej repliki

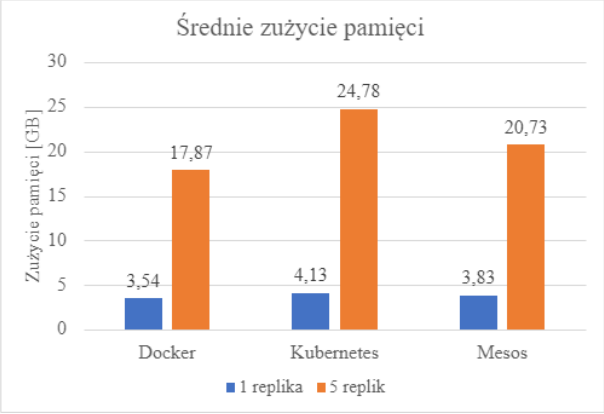
	Mediana [MHz]	Odchylenie standardowe [MHz]
Docker	3,40	1,16
Kubernetes	4,93	1,23
Mesos	4,38	0,98

Mediana posiada podobną wartość do średniego zużycia pokazanego na Rysunku 2 dla każdego narzędzia. Odchylenie standardowe oscyluje wokół 1 MHz dla jednej oraz 2,4 MHz dla pięciu replik. Są to stosunkowo niskie wartości, które wskazują na stabilne zużycie zasobów.

Tabela 6: Mediana i odchylenie standardowe obciążenia procesora dla pięciu replik

	Mediana [MHz]	Odchylenie standardowe [MHz]
Docker	14,28	2,62
Kubernetes	14,28	2,48
Mesos	15,98	2,01

Wyniki średniego wykorzystania pamięci RAM przedstawione na Rysunku 3 po raz kolejny wskazują na Docker Swarm jako najlepiej zoptymalizowane narzędzie dla małych systemów. Pomimo użycia tego samego obrazu aplikacji z identycznymi wtyczkami, poradził sobie najlepiej z optymalizacją aż o ponad 0,5GB dla jednej repliki w stosunku do narzędzia Kubernetes. Mesos prezentuje się pobodnie ze średnią wartością zajętości pamięci większą o około 0,3GB. W przypadku pięciu replik każdy orkiestrator uzyskał proporcjonalne wyniki około 5 razy większe od pojedynczej repliki. Jedynie Kubernetes uzyskał większą wartość na poziomie 6 krotności swojej pojedynczej repliki, co obala hipotezę H2.



Rysunek 3: Średnie zużycie pamięci RAM.

W tabelach 7 i 8 przedstawiono wartości mediany oraz odchylenia standardowego dla badań wykorzystania pamięci dla RAM jednej i pięciu replik aplikacji. Wartości mediany są po raz kolejny bardzo zbliżone do wartości średnich, co wskazuje na równomierność uzyskanych wyników. Dodatkowo, odchylenia standardowe są minimalne, co oznacza, że zmienność w zajętości pamięci jest niewielka. Gwarantuje to stabilny poziom zużycia pamięci RAM podczas przebiegu testów.

Tabela 7: Mediana i odchylenie standardowe wykorzystania pamięci RAM dla jednej repliki

	Mediana [GB]	Odchylenie standardowe [GB]
Docker	3,58	0,148
Kubernetes	4,12	0,073
Mesos	3,83	0,077

Tabela 8: Mediana i odchylenie standardowe wykorzystania pamięci RAM dla pięciu replik

	Mediana [GB]	Odchylenie standardowe [GB]
Docker	17,75	0,054
Kubernetes	24,78	0,011
Mesos	20,88	0,071

5.3. Pomiar przepustowości

Kolejny pomiarem jest badanie przepustowości i niezawodności orkiestratorów. Celem badania było zbadanie wydajności każdego orkiestratora w przypadku nagłego wzrostu obciążenia aplikacji. Zarządzanie ruchem dla każdego orkiestratora zostało przeprowadzone przez narzędzie równoważenia obciążenia. Dla Docker Swarm użyto wbudowany Ingress, Kubernetes korzystał z zadeklarowanego serwisu, w przypadku Mesos był to Nginx. Automatyczne skalowanie jest ważną cechą, która powinna być prawidłowo zaimplementowana w narzędziu.

Label	Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	Received KB/s	Sent KB/s
HomePage	60000	604	261	1659	2373	3746	3	9455	1.41%	2027.36	43280.67	226.43
TOTAL	60000	604	261	1659	2373	3746	3	9455	1.41%	2027.36	43280.67	226.43

Rysunek 4: Tabela raportu wygenerowanego dla testów obciążeniowych narzędzia JMeter dla Docker Swarm.

Label	Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	Received KB/s	Sent KB/s
HomePage	60000	892	78	3780	4367	5150	4	7441	0.41%	1420.7	31299.83	161.66
TOTAL	60000	892	78	3780	4367	5150	4	7441	0.41%	1420.7	31299.83	161.66

Rysunek 5: Tabela raportu wygenerowanego dla testów obciążeniowych narzędzia JMeter dla Kubernetes.

Label	Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	Received KB/s	Sent KB/s
HomePage	60000	742	143	2741	3439	4671	4	8647	0.71%	1824.65	38532.74	170.75
TOTAL	60000	742	143	2741	3439	4671	4	8647	0.71%	1824.65	38532.74	170.75

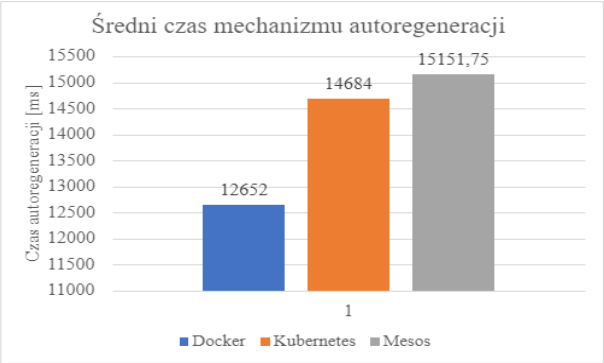
Rysunek 6: Tabela raportu wygenerowanego dla testów obciążeniowych narzędzia JMeter dla Apache Mesos.

Przeprowadzono badania, których celem było ocenienie wydajności różnych narzędzi w kontekście działania aplikacji na 2000 hostach. Testy opierały się na scenariuszu polegającym na uruchomieniu aplikacji i odczytaniu tekstu powitalnego "Welcome to Jenkins!". Całkowita liczba żądań wyniosła 60000, przy czym każdy host przeprowadzał test 30 razy, z 10-sekundowym czasem rozruchu. Wyniki testów zostały przedstawione na Rysunkach 4, 5 i 6, zawierają one informacje dotyczące sumy żądań, wartości minimalnej, maksymalnej, średniej i mediany czasu odpowiedzi, ilości wysłanych i przyjętych danych oraz poziomu przepustowości. Ważnym aspektem są także percentyle o wartościach 90%, 95% i 99%. Określają one wartości, poniżej których znajduje się odpowiedni procent wyników. Z analizy wynika, że Docker Swarm uzyskał najkrótszy i najdłuższy czas odpowiedzi, a także najlepszą wydajność przesyłu danych spośród badanych narzędzi. Jednocześnie, miał on również największą liczbę błędów na poziomie 1.41%, co oznacza, że 1.41% żądań nie powiodło się z powodu braku odpowiedzi. Najlepsze rezultaty w przypadku odporności na błędy zyskał Kubernetes, gdzie wskaźnik niepowodzeń wynosił około 0.40%. Pozostałe badane narzędzia wykazały podobne

wyniki pod względem czasu odpowiedzi i wydajności w obliczu nagłego wzrostu ruchu w aplikacji. Podsumowując, wyniki badań wskazują, że Docker Swarm wykazał się najlepszą ogólną wydajnością w zależności od mierzonej metryki. Pozostałe narzędzia radziły sobie podobnie w zakresie obsługi wzrostu ruchu, jednak Kubernetes osiągnął najniższy wskaźnik niepowodzeń.

5.4. Pomiar skuteczności mechanizmów autoregeneracji

Ostatnim z przeprowadzonych badań był pomiar mechanizmów autoregeneracji testowanych platform orkiestracji. Pomiar wykonano umyślnie wywołując usterkę, która zatrzyma działanie aplikacji oraz odczytano czas od zatrzymania do w pełni przywróconego kontenera z aplikacją. W tym celu zastosowano środowiska testowe zawierające pięć replik.



Rysunek 7: Średni czas regeneracji kontenera.

Na rysunku 7 przedstawiono średni czas, jaki potrzebował każdy z badanych orkiestratorów na przywrócenie repliki z aplikacją do stanu gotowości. Wynik został przedstawiony w milisekundach. Stanowczo najlepszy wynik, lepszy aż o 2 sekundy od Kuberntesa oraz o 3 sekundy od Mesosa zyskał Docker Swarm. Po raz kolejny narzędzie natywne do silnika konteneryzacji Docker radzi sobie najlepiej z małymi klastrami (do kilkudziesięciu podów). Kubernetes oraz Mesos uzyskały wynik zbliżony do siebie.

Tabela 9: Mediana i odchylenie standardowe czasu autoregeneracji kontenera

	Mediana [ms]	Odchylenie standardowe [ms]
Docker	12979	929,36
Kubernetes	14684	713,47
Mesos	15234	599,21

W Tabeli 9 przedstawione są mediana oraz odchylenie standardowe czasu autoregeneracji kontenera z aplikacją dla każdego narzędzia. Niskie wartości odchyleń standardowych potwierdzają stabilność przeprowadzonych badań. Mediana pokrywa się ze średnią wartością a odchylenie standardowe nie przekracza 1 sekundy. Z analizy uzyskanych danych wynika, że mechanizmy przywracania repliki po awarii działają przewidywalnie oraz bez większych zakłóceń.

6. Wnioski

Niniejszy artykuł przedstawia analizę narzędzi orkiestracji kontenerów, porównując ich funkcje i usługi. Opracowano kompleksowy zestaw wskaźników do oceny wydajności tych narzędzi pod względem planowania i zarządzania usługami. Wybrano trzy reprezentatywne orkiestratory, a mianowicie Docker Swarm, Kubernetes i Apache Mesos, i przeprowadzono szczegółowe badanie i ocenę każdego z nich. Pod względem wydajności, badanie wykazało, że Docker Swarm jest obecnie jednym z najwydajniejszych dostępnych orkiestratorów (Rysunek 1-7), co tłumaczy jego popularność wśród praktyków. Złożona architektura Kubernetes oraz Apache Mesos może czasami powodować znaczne obciążenie, co może wpływać na ich wydajność. Przodują one pod względem funkcjonalności. Wyniki tego badania dostarczyły cennych spostrzeżeń i służą jako podstawa do przyszłych badań. W celu głębszego zbadania narzędzi należy użyć rozbudowanego klastra z wieloma powiązanymi elementami. Przeprowadzenie badań na takim środowisku dostarczy szczegółowych danych na temat cech każdego orkiestratora. Wyniki przeprowadzonego badania (Rysunek 1) wyraźnie potwierdzają hipotezę H1. Docker Swarm potrzebował najmniej czasu na uruchomienie aplikacji testowej. Wyniki badania zużycia procesora potwierdzają hipotezę badawczą H2. Dla pięciu replik aplikacji platforma Kubernetes uzyskała najlepszą efektywność skalowania (Rysunek 2).

Przeprowadzone badania są zgodne z wynikami przytoczonych artykułów. Analiza skupiała się na małym środowisku kontenerowym oraz zbadaniu wydajności platform w zarządzaniu nim. Potwierdzono najlepszą pozycję orkiestratora Docker Swarm dla tego typu wdrożeń.

Literatura

[1] Mikrousługi a architektura monolityczna, https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith, [07.06.2023].

[2] J. Stubbs, W. Moreira, R. Dooley, Distributed systems of microservices using Docker and Serfnode, 7th International Workshop on Science Gateways (2015) 34–39, https://doi.org/10.1109/iwsg.2015.16.

[3] I. M. A. Jawarneh et al., Container Orchestration Engines: A Thorough Functional and Performance Comparison, ICC 2019 - 2019 IEEE International Conference on Communications (2019) 1-6, https://doi.org/10.1109/ICC.2019.8762053.

[4] A. Malviya, R. K. Dwivedi, A Comparative Analysis of Container Orchestration Tools in Cloud Computing, 9th International Conference on Computing for Sustainable Global Development (2022) 698-703, https://doi.org/10.23919/INDIACom54597.2022.9763171.

[5] Y. Pan, I. Chen, F. Brasileiro, G. Jayaputera, R. Sinnott, A Performance Comparison of Cloud-Based Container

Orchestration Tools, IEEE International Conference on Big Knowledge (2019) 191-198, https://doi.org/10.1109/ICBK.2019.00033.

[6] A. Shemyakinskaya, I. Nikiforov, Disk Space Management Automation with CSI and Kubernetes. Proceedings of Seventh International Congress on Information and Communication Technology. Lecture Notes in Networks and Systems 447 (2023) 171-179, https://doi.org/10.1007/978-981-19-1607-6_15.

[7] C. Cérin, T. Menouer, W. Saad, W. B. Abdallah, A New Docker Swarm Scheduling Strategy, IEEE 7th International Symposium on Cloud and Service Computing (2017) 112-117, https://doi.org/10.1109/SC2.2017.24.

[8] P. Saha, A. Beltre, M. Govindaraju, Exploring the Fairness and Resource Distribution in an Apache Mesos Environment, IEEE 11th International Conference on Cloud Computing (2018) 434-441, https://doi.org/10.1109/CLOUD.2018.00061.

[9] D. K. Kang, G. B. Choi, S. H. Kim, I. S. Hwang, C. H. Youn, Workload-aware resource management for energy efficient heterogeneous Docker containers, IEEE Region 10 Conference (2016) 2428-2431, https://doi.org/10.1109/TENCON.2016.7848467.

[10] Porównanie Docker Swarm i Kubernetes, https://circleci.com/blog/docker-swarm-vs-kubernetes/, [07.06.2023].

[11] Ankieta CNCF 2022, https://www.cncf.io/reports/cncf-annual-survey-2022/, [07.06.2023].

[12] Porównanie Kubernetes, Mesos oraz Docker Swarm, https://www.sumologic.com/insight/kubernetes-vs-mesos-vs-swarm/, [07.06.2023].

[13] Dokumentacja Apache Mesos, https://mesos.apache.org/documentation/lates, [07.06.2023].

[14] Dokumentacja Apache JMeter, https://jmeter.apache.org, [07.06.2023].

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 391–398
Received: 21 August 2023
Accepted: 1 October 2023

A study of the user experience while working with mobile applications cooperating with sports bands

Badanie doświadczeń użytkowników podczas pracy z aplikacjami mobilnymi współpracującymi z opaskami sportowymi

Szymon Czopek*, Mariusz Dzieńkowski

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The article analyses user experience while using two mobile applications, FitPro and Zepp Life that work with sports bands designed to monitor physical activity. As part of the study, the users were asked to perform tasks during which they used each app's features. During the same research scenario, initially on the first application and then on the second application, the users' eye activity was recorded by means of the eyetracker. Afterwards, each participant took part in a survey. Task completion time, eye-tracking indicators (the number of fixations, fixation duration, the number of blinks), users' evaluation and the level of diagnosed errors were used for the comparative analysis. All the results proved that the FitPro application, despite containing fewer features, was more intuitive, easier to use, more efficient, had a clear interface and, as a result, was better rated by the users.

Keywords: user experience study; mobile application; sports band; eye tracking

Streszczenie
Artykuł zawiera analizę doświadczeń użytkowników podczas korzystania z dwóch aplikacji mobilnych FitPro i Zepp Life, które współpracują z opaskami sportowymi przeznaczonymi do monitorowania aktywności fizycznej. W ramach badania, użytkownicy zostali poproszeni o wykonanie zadań, podczas których korzystali z poszczególnych funkcji aplikacji. Podczas realizacji tego samego scenariusza badawczego najpierw na pierwszej, a później na drugiej aplikacji, rejestrowano za pomocą eyetrackera aktywność oczną użytkowników. Następnie każdy uczestnik brał udział w badaniu ankietowym. Do analizy porównawczej wykorzystano czas realizacji zadań, wskaźniki eyetrackingowe (liczba fiksacji, czas trwania fiksacji, liczba mrugnięć), ocenę użytkowników oraz liczbę zdiagnozowanych błędów. Wszystkie wyniki wykazały, że aplikacja FitPro mimo, że zawierała mniej funkcji, była bardziej intuicyjna, łatwiejsza w obsłudze, wydajniejsza, miała przejrzysty interfejs i w efekcie została lepiej oceniona przez użytkowników.

Słowa kluczowe: badanie doświadczeń użytkownika; aplikacja mobilna; opaska sportowa; eyetracking

*Corresponding authors
Email address: szymon.czopek@pollub.edu.pl (S. Czopek)

1. Wstęp

Pomiar osiąganych wyników podczas aktywności fizycznej oraz ich ciągłe ulepszanie są nieodłącznymi cechami sportu. Jest to pewnego rodzaju element mobilizujący, dzięki któremu osoby aktywne ruchowo czują motywację do poprawienia osiągniętych rezultatów. Ma to również wpływ na własne zadowolenie oraz satysfakcję społeczną w momencie dzielenia się swoimi wynikami z innymi. Przed spopularyzowaniem cyfrowych urządzeń do pomiaru stanu fizjologicznego człowieka można było zmierzyć pewne parametry aktywności ruchowej, lecz były one tylko szacunkowe i nieprecyzyjne. Zebrane wyniki należało samodzielnie zapisywać i zliczać. Zmieniło się to dzięki rozwojowi i popularyzacji oraz miniaturyzacji technologii w formie gadżetów sportowych. Osiągane sukcesywnie wyniki nie muszą być już nigdzie zapisywane. Każda osoba uprawiająca sport ma dostęp do szczegółowych danych, które kiedyś były możliwe do zbadania tylko w specjalistycznych centrach sportowych.

W sporcie oraz medycynie wykorzystywanych jest wiele urządzeń mierzących parametry fizjologiczne człowieka. W obecnych czasach, najpopularniejszym, łatwo dostępnym narzędziem, które zawiera w sobie wiele czujników do pomiarów, jest opaska fitness. Gadżet ten zapewnia szeroki wachlarz funkcji. Standardem jest to, że zawiera w sobie GPS, żyroskop, a także ciśnieniomierz. Opaska komunikuje się z aplikacją zainstalowaną na urządzeniu mobilnym dzięki bezprzewodowemu połączeniu Bluetooth. W smartfonie zapisywane i przetwarzane są wszystkie interesujące użytkownika dane. Można je następnie analizować, porównywać i udostępniać innym użytkownikom.

Użyteczność aplikacji współpracujących z opaskami sportowymi to fundamentalna sprawa i jest ona istotna już w momencie wyboru modelu opaski. Aplikacja taka powinna być łatwa w opanowaniu i prosta w użytkowaniu. Realizacja czynności w aplikacji powinna zajmować mało czasu, ponieważ użytkownik zamierzający skorzystać z funkcji opaski sportowej może być w trakcie wykonywania aktywności fizycznej.

W badaniu użyteczności interfejsów użytkownika oprócz klasycznych sposobów, tzn. ankiet czy list kontrolnych, wykorzystuje się coraz częściej technikę eye-

trackingową. Polega ona na rejestrowaniu aktywności wzrokowej użytkownika w trakcie jego kontaktu z interfejsem. Dzięki rejestracji i odtwarzaniu utrwalonych stanów oczu, możliwe jest przeanalizowanie ruchów oczu oraz miejsc skupienia uwagi wzrokowej w trakcie korzystania z aplikacji.

W ramach pracy zostało przeprowadzone badanie eyetrackingowe oraz ankietowe. Obiektami badawczymi były dwie aplikacje mobilne, kompatybilne z opaskami sportowymi. Badanie polegało na ocenie jakości, wydajności oraz analizie użyteczności i satysfakcji użytkownika podczas korzystania z danego oprogramowania.

2. Przegląd literatury

Duża konkurencja wśród aplikacji mobilnych, wymusza na ich twórcach i wydawcach wysokiej jakości interfejsy, a w związku z tym także korzystanie z zaawansowanych technik ich tworzenia. Istnieje wiele metod projektowania oraz sposobów wytwarzania proponowanych przez specjalistów.

Pozycje książkowe [1, 2] są praktycznym wprowadzeniem do projektowania efektownych aplikacji mobilnych. Znajdują się w nich informacje o tym, jak poprawnie tworzyć interfejsy szczególnie w przypadku małych wyświetlaczy. Literatura ta dostarcza także wiedzy o typowych błędach, które obniżają doznania użytkowników w trakcie korzystania z oprogramowania. Autorzy wyjaśniają, czego należy unikać podczas projektowania oprogramowania. Ponadto przedstawione są również narzędzia do prototypowania interfejsu użytkownika.

Użyteczność i dostępność to pojęcia, które są ze sobą ściśle powiązane. Pierwsza cecha oprogramowania odnosi się do funkcjonalności oraz stopnia, w jakim aplikacja mobilna jest łatwa do nauczenia, zapamiętania i stosowania przez użytkowników. Termin *użyteczność* jest używany w odniesieniu do metod usprawniających intuicyjność korzystania z interfejsu użytkownika. Z kolei dostępność oprogramowania odnosi się do jakości systemu w sytuacji, gdy jest on wykorzystywany [3].

W ramach pracy [4] zaprojektowano interfejsy aplikacji mobilnych, zaprezentowano je ankietowanym do oceny pod kątem czy dany interfejs odpowiadał cechom, które są istotne w psychologii projektowania. Wzięto pod uwagę kluczowe elementy dotyczące projektowania interfejsów, które zwiększają atrakcyjność danego widoku. Badania wykonane w tej pracy zostały przeprowadzone za pomocą ankiety, w której respondent decydował o ważnych cechach dotyczących wyglądu interfejsu graficznego.

Z artykułu [5] można wywnioskować, że aplikacje jako narzędzia sportowe, mają pozytywny wpływ na poziom aktywności fizycznej w nowoczesnym społeczeństwie. Tego typu oprogramowanie pozwala na rejestrację osiągnięć, a następnie udostępnienie ich na wybranym portalu społecznościowym lub wysłanie do grona znajomych. W ten sposób włączony jest element grywalizacji - współzawodnictwa w poczynaniach sportowych. Te funkcje motywują do podejmowania wysiłku i stawiania sobie nowych celów.

W kolejnym artykule [6], jego autor podkreśla, że nie same funkcjonalności aplikacji wpływają na zainteresowanie aplikacją, ale także poziom trudności jej obsługi. Interfejs oprogramowania powinien być spójny i nie chodzi tylko o stronę graficzną, ale o spójne modele oraz odpowiednie ustawienie wyświetlanych elementów. Praca z interfejsem nie powinna sprawiać problemu nawet bez konieczności używania instrukcji. Autor twierdzi, że wielu programistów uważa się za genialnych artystów i w związku z tym często nie zwraca uwagi na standardy projektowania.

Artykuł [7] przedstawia przykład badania jakości interfejsu aplikacji internetowej z wykorzystaniem techniki eyetrackingowej. Obiektem badań był system wspomagający obsługę miejskich wyścigów rowerowych. W trakcie eksperymentu użytkownicy wykonywali typowe scenariusze obsługi systemu. Zachowania i poczynania badanych zostały zarejestrowane przez kamery eyetrackera i następnie poddane analizie. W eksperymencie wszyscy użytkownicy posiadali profil analogiczny do potencjalnych użytkowników aplikacji. Finalnie przedstawiono rezultaty badań ilościowych, wykryte błędy w interfejsie oraz wnioski ukierunkowane na poprawę jego jakości.

W obecnych czasach badania okulograficzne realizowane są na wysokim poziomie. W związku z tym istnieje wiele firm oferujących badania ergonomii interfejsów wykorzystujących tą technologię. Eyetracking pozwala zarejestrować i przeanalizować stany oczu, a przede wszystkim ruchy gałek ocznych badanego. Ruchy sakadowe (skokowe) to mimowolne ruchy oka, które wykonywane są podczas obserwowania obiektów. Informacje są pobierane podczas fiksacji, czyli w momentach, kiedy oczy są względnie nieruchome. Są to przerwy występujące pomiędzy sakadami i trwające powyżej 80 ms. Pozyskane dane mogą być przekształcone i przedstawione w postaci map cieplnych (ang. heatmaps) lub map fiksacji (ang. gazeplots). Mapa cieplna odwzorowuje pozycje, częstość skupiania wzroku oraz czas w lokalizacjach, które przyciągają uwagę. Mapa fiksacji przedstawia ruchy i zatrzymania punktu widzenia. Fiksacje są obrazowane w postaci kolorowych kółek, a sakady jako linie łączące fiksacje [8].

3. Cel i teza badawcza oraz zakres pracy

Celem pracy jest ocena doświadczeń użytkowników obejmująca użyteczność i poziom ich zadowolenia po interakcji z dwoma aplikacjami mobilnymi komunikującymi się z opaskami sportowymi, przeznaczonymi do monitorowania aktywności fizycznej. Badania zostały przeprowadzone z wykorzystaniem eyetrackera oraz ankiety. Dzięki eytrackerowi możliwe było zarejestrowanie pozycji skupień wzroku użytkownika w trakcie wykonywanych działań i utrwalenie ich w formie nagrania wideo. Eksperyment został przeprowadzony w laboratorium Katedry Informatyki Politechniki Lubelskiej. Przed jego realizacją wyselekcjonowano dwie

aplikacje mobilne współpracujące z opaskami sportowymi, dokonano doboru metod badawczych oraz opracowano scenariusz, który realizowali wszyscy uczestnicy badań.

W ramach pracy zostały sformułowane następujące hipotezy:

H1: *Aplikacja mobilna z większym zakresem funkcjonalności będzie wyżej oceniana przez użytkowników.*

H2: *Szybkość obsługi aplikacji i jej podatność na błędy użytkownika wpływa na jej ocenę wystawianą przez użytkowników.*

H3: *Złożoność interfejsu aplikacji zawierającej bogatszy zestaw funkcjonalności objawia się wysokim poziomem średniej czasu trwania fiksacji.*

H4: *Wysoka średnia liczba fiksacji podczas realizacji zadań oznacza wysoki poziom trudności obsługi aplikacji.*

4. Metoda badawcza

W eksperymencie porównano jakość interfejsów dwóch aplikacji, biorąc pod uwagę zarówno użyteczność, jak i zadowolenie użytkowników. Eksperyment badawczy obejmował następujące etapy:

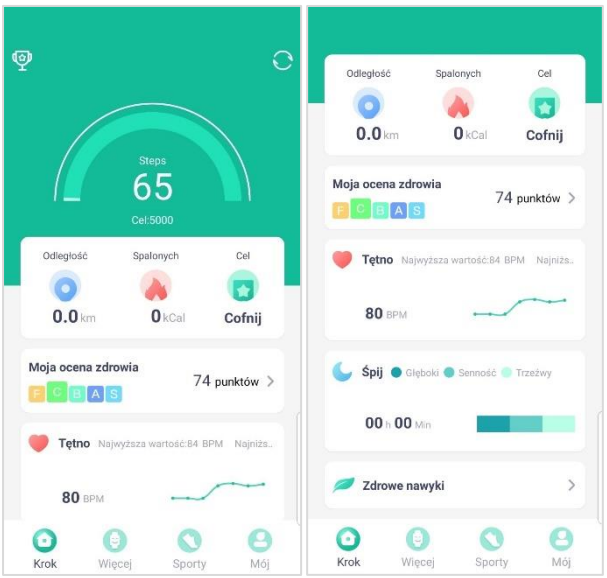
- realizacja przez użytkowników scenariuszy, składających się z zadań, których celem było wykonanie określonych czynności w aplikacji mobilnej;
- obserwacja działań i zachowań użytkowników w trakcie realizacji poleceń, w celu uchwycenia napotkanych problemów;
- po eksperymencie eyetrackingowym wypełnienie krótkiej ankiety diagnozującej zadowolenie badanych;
- analiza danych eyetrackingowych i wyników ankiet oraz ocena badanych aplikacji mobilnych.

Przy porównywaniu interfejsów aplikacji mobilnych uwzględniono następujące miary:

- czas wykonania poszczególnych scenariuszy;
- poziom satysfakcji uczestników badań;
- średnie wartości miar eyetrackingowych: czasu trwania fiksacji, liczby fiksacji oraz liczby mrugnięć;
- liczba problemów napotkanych przez użytkowników.

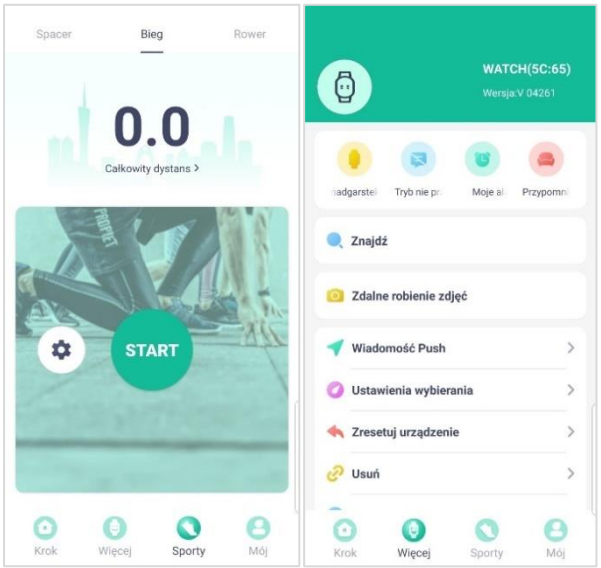
4.1. Obiekty badań

Obiektami badań były dwie aplikacje mobilne komunikujące się z opaskami sportowymi: FitPro oraz Zepp Life. Aplikacje te są darmowe i dostępne w sklepie Google Play oraz App Store. FitPro to narzędzie firmy Shenzhen Jusheng Intelligent Technology Ltd., które dzięki technologii Bluetooth jest w stanie nawiązać połączenie z wieloma modelami opasek fitness oraz smartwatchami (Rysunek 1). W menu głównym tej aplikacji widoczne są zarówno dane zarejestrowane przez opaskę sportową jak i automatycznie wygenerowane wyniki. W menu tym najbardziej wyeksponowanym widokiem jest interfejs dotyczący liczby pokonanych kroków, na podstawie której szacowana jest liczba spalonych kalorii oraz przebyta odległość. W tym miejscu wyświetlany jest ustalony przez użytkownika cel dotyczący liczby zamierzonych do przebycia kroków. Poniżej tych danych widnieje ocena zdrowia zaprezentowana w postaci litery i wyznaczona na podstawie wskaźnika BMI (ang. Body Mass Index). Obok oceny wyświetlany jest także procentowy poziom tkanki tłuszczowej. Kolejną informacją tego głównego interfejsu jest liczba uderzeń serca na minutę. Opaska sportowa mierzy również tętno, ciśnienie i natlenienie krwi. Ostatnią możliwością opaski jest monitorowanie snu. W tym przypadku dla kolejnych dni zapisywane są dane w interwałach czasowych dla poszczególnych faz snu. Są one wyznaczane na podstawie pomiarów bicia serca podczas snu.



Rysunek 1: Widoki menu głównego aplikacji FitPro.

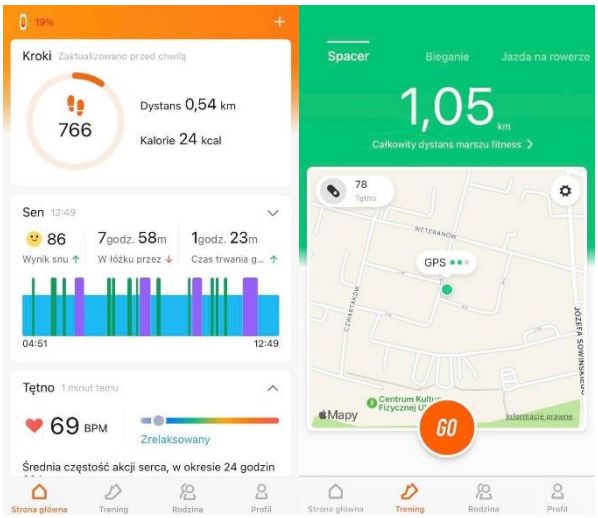
W kolejnym oknie aplikacji znajduje się narzędzie rejestrujące lokalizację sportowca i wyznaczające na tej podstawie trasę na mapie w czasie biegu, spaceru czy jazdy na rowerze. Odmierzany jest przy tym czas, a położenie rejestrowane jest za pomocą lokalizatora GPS telefonu.



Rysunek 2: Widoki zakładek *Sporty* oraz *Więcej* aplikacji FitPro.

Dodatkową funkcją aplikacji jest odbiór przez smartwatcha powiadomień z innych aplikacji obsługiwanych przez smartfon: informacji o nieodebranych połączeniach, pochodzących z mediów społecznościowych czy poczty elektronicznej. Kolejną funkcją programu jest zdalne robienie zdjęć aparatem telefonu za pośrednictwem opaski znajdującej się na ręce.

Zepp Life to aplikacja producenta Anhui Huami Information Technology. Służy do monitorowania i analizowania aktywności fizycznej. Aplikacja jest dostępna na różne platformy, takie jak iOS i Android. Pozwala ona na śledzenie swoich treningów, monitorowanie postępów i uzyskiwanie spersonalizowanych sugestii treningowych. Aplikacja Zepp Life łączy się z różnymi urządzeniami, jak na przykład smartwatche czy opaski fitness pozwalając na śledzenie parametrów takich jak liczba wykonanych kroków, przebyty dystans, spalone kalorie, jakość snu, itd. Poza tym aplikacja umożliwia pomiar tętna oraz poziomu stresu. Program oferuje także analizę wagi oraz bilans parametrów treningu.



Rysunek 3: Widoki menu głównego oraz zakładki *Trening* w Zepp Life.

Aplikacja ta zapewnia również funkcję monitorowania treningów specjalistycznych, takich jak bieganie czy jazda na rowerze, a także pozwala na ustawianie celów i monitorowanie postępów. ZeppLife oferuje również funkcję trenera, która dostarcza personalizowanych sugestii treningowych na podstawie analizy zebranych danych z urządzeń i wykonanych treningów.

Tabela 1: Zestawienie unikalnych funkcjonalności względem drugiej aplikacji

Aplikacja	Unikalne funkcjonalności względem drugiej aplikacji
FitPro	• Ocena zdrowia • Zdrowe nawyki
Zepp Life	• Zdrowie kobiety • Analiza wagi • Ocena sylwetki • Analiza stresu • Stan treningu • Bilans

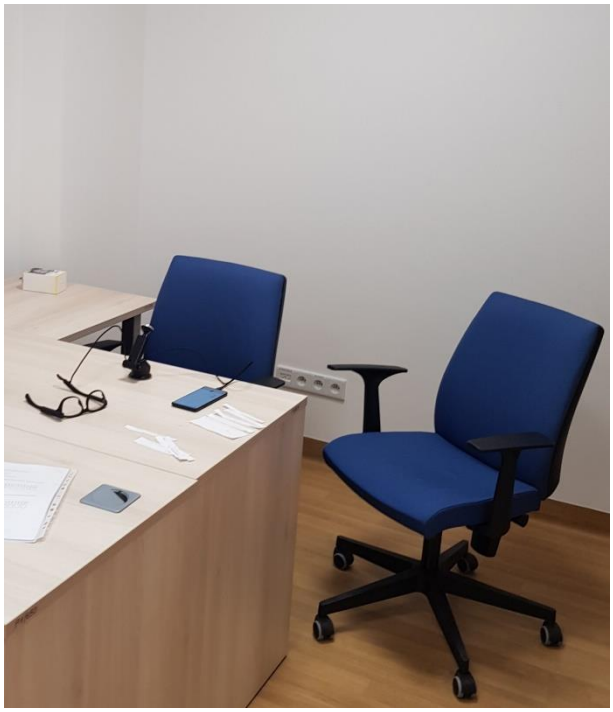
Aplikacja ta posiada również funkcję monitorowania i analizowania snu, pozwalając na lepsze zrozumienie tego, jak jakość snu wpływa na zdrowie i wydajność. Ogólnie rzecz biorąc, Zepp Life jest kompleksowym narzędziem do monitorowania i analizy aktywności fizycznej, które pomaga użytkownikom lepiej zrozumieć swoje treningi i osiągnąć swoje cele fitness.

4.2. Grupa badawcza

W badaniu wzięło udział dwanaście osób. Wiek badanych zawierał się w przedziale 21 – 25 lat. Uczestnicy badań byli studenci Politechniki Lubelskiej. Przeważająca liczba osób zadeklarowała bardzo dobrą wiedzę technologiczną i techniczną w temacie obsługi smartfonu. Połowa badanych nie korzystała wcześniej z aplikacji współpracujących ze smartwatchami lub opaskami fitness. Wśród badanych przeważał brak regularności w wykonywaniu ćwiczeń sportowych. Częste uprawianie sportu lub jego brak były najrzadszymi odpowiedziami.

4.3. Stanowisko badawcze

Eksperyment został przeprowadzony w specjalnym pomieszczeniu, które znajduje się w Centrum Innowacji i Transferu Technologii Politechniki Lubelskiej, należącym do Katedry Informatyki. Badania były przeprowadzane pod nadzorem moderatora, który zapisywał swoje obserwacje dotyczące wyników wykonanych scenariuszy.



Rysunek 4: Stanowisko badawcze.

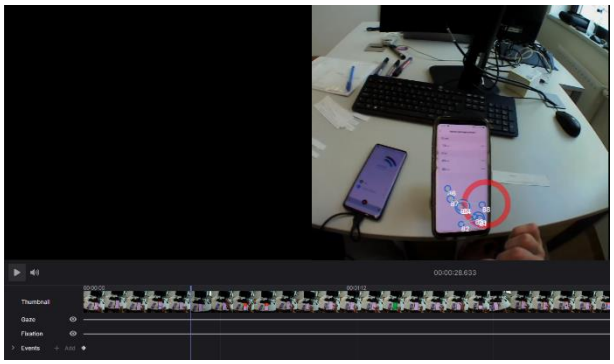
Do rejestracji danych wykorzystano zdalny eyetracker o nazwie Pupil Invisible [9], mający formę okularów z wbudowanymi kamerami oraz diodami podczerwieni (widoczny na Rysunku 4.). Okulary są połączone ze smartfonem, na którym zainstalowana jest odpo-

wiednia aplikacja do rejestrowania obrazu sceny przed użytkownikiem oraz ruchów i skupień oczu badanego. Szczegółowe parametry techniczne użytego eyetrackera przedstawiono w Tabeli 2.

Tabela 2: Parametry eyetrackera [9]

Częstotliwość próbkowania	200Hz
Detekcja stanów oka	fiksacje, sakady, zmiany średnicy źrenicy
Technika śledzenia	ciemna źrenica
Dokładność (w idealnych warunkach)	0,6° obuocznie
Wbudowana kamera sceny	rozdzielczość 1088 x 1080, zakres widzenia 82 x 82
Smartfon do rejestrowania danych	OnePlus 8 Android

Otrzymane dane są następnie wysyłane do chmury producenta oprogramowania Cloud Pupil Labs. Jest to internetowa platforma do przechowywania danych, ich wizualizacji oraz analizy, która wykorzystuje moc obliczeniową chmury. Oprócz nagrania wideo znajdują się tam także precyzyjne dane dotyczące pojedynczych fiksacji oraz mrugnięć.



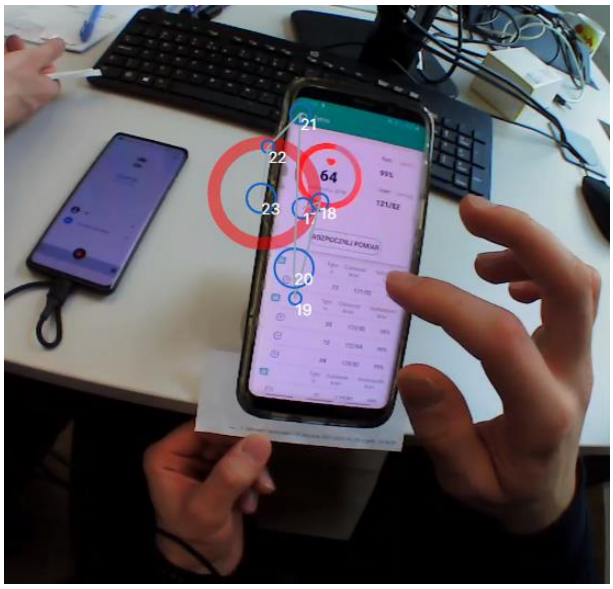
Rysunek 5: Widok okna aplikacji funkcjonującej na platformie Cloud Pupil Labs.

4.4. Eksperyment

Do zbadania doświadczenia użytkownika i użyteczności aplikacji przygotowano eksperyment, który obejmował pięć zadań, wykonywanych przez uczestników na obu testowanych aplikacjach - FitPro i Zepp Life (Tabela 3).

Tabela 3: Zadania wykonywane przez badanych

Zadanie	Treść zadania
Z1	Sprawdź tętno krwi z dnia 25 stycznia 2023 roku o godz. 14:38:40.
Z2	Jaką długość miał spacer odbyty w dniu 21 marca 2023 roku o godz. 14:27?
Z3	Zmień wagę (masę ciała) na dowolną.
Z4	Podaj średnią, przypadającą na jeden dzień liczbę kroków we wrześniu 2022 roku.
Z5	Ustaw cel kroków na 10000.



Rysunek 6: Eksperyment.

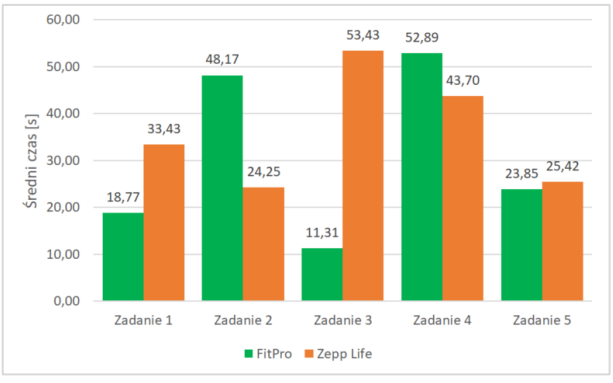
Przebieg każdej sesji badawczej składał się z następujących etapów:

- zaproszenie badanego;
- przedstawienie mu celu i przebiegu badań;
- aprobata uczestnictwa w eksperymencie przez badanego;
- kalibracja eyetrackera;
- nagrywanie sceny przed użytkownikiem wraz z aktywnością oczną podczas wykonywania zadań oraz obserwacja działań uczestników w celu wychwycenia błędów;
- wypełnienie ankiety przez badanego.

5. Wyniki badań

5.1. Pomiar czasu realizacji poszczególnych zadań

Wykres na Rysunku 7 prezentuje średni czas wykonywania poszczególnych zadań dla obu aplikacji – FitPro oraz Zepp Life przez uczestników eksperymentu.



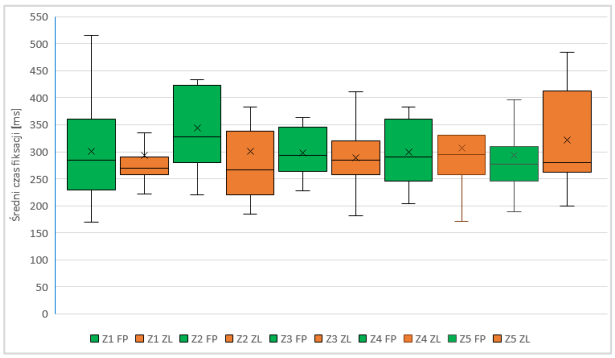
Rysunek 7: Średni czas wykonywania zadań dla poszczególnych aplikacji.

W aplikacji FitPro ukończenie większości zadań tj. Z1, Z3 i Z5, zajmowało użytkownikom mniej czasu. Średni czas wykonania zadania Z5, w którym badany miał zmienić cel kroków, był zbliżony dla obu aplikacji z lepszym wynikiem w przypadku FitPro. Znaczna

różnica w średnich czasach widoczna jest dla zadania Z3, w którym uczestnicy mieli zmienić swoją wagę ciała. Powodem tego stanu rzeczy jest to, że w aplikacji Zepp Life wykonanie tych operacji jest bardziej złożone. W FitPro wagę ustawia się intuicyjnie - w ustawieniach aplikacji, zaś w Zepp Life służy do tego oddzielna rozbudowana funkcjonalność - *Analiza wagi*.

5.2. Wyniki badań eyetrackingowych

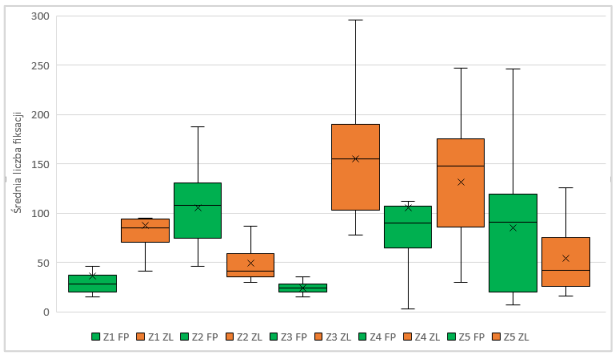
Na kolejnych wykresach pudełkowych zawartych na Rysunkach 8-10, zaprezentowano informacje statystyczne, obejmujące sześć podstawowych statystyk: wartość minimalną, pierwszy kwartyl, średnią arytmetyczną, medianę, trzeci kwartyl oraz wartość maksymalną. Wykresy te dotyczą miar eyetrackingowych takich jak: średni czas trwania fiksacji, liczba fiksacji oraz liczba mrugnięć.



Rysunek 8: Średnie czasy trwania fiksacji dla poszczególnych zadań realizowanych na dwóch testowanych aplikacjach.

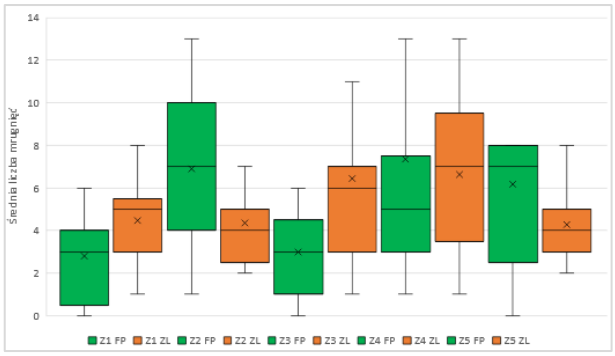
Przyjmuje się, że krótsze czasy fiksacji są związane z pobieraniem informacji z bardziej złożonych, zawierających więcej detali interfejsów [10]. Generalnie taka sytuacja powinna mieć miejsce w przypadku aplikacji Zepp Life zawierającej bogatszy zestaw funkcji. Tak też było podczas realizacji przez użytkowników zadań 1-3, gdy średnie czasy trwania fiksacji były dla tej aplikacji krótsze niż dla FitPro, dla której realizacja zadań wymagała głębszego przetwarzania danych sensorycznych i wiązała się z większym wysiłkiem poznawczym. Z kolei średnie czasy trwania fiksacji dla zadań 4 i 5 były nieznacznie krótsze dla aplikacji FitPro.

Wyniki przedstawione na Rysunku 9 dotyczą liczby fiksacji - wskaźnika, który m.in. może oznaczać poziom trudności w znalezieniu i rozpoznaniu umieszczonych w interfejsie elementów [10]. Dla zadań 1-3 liczba fiksacji jest proporcjonalna do czasu realizacji poleceń. Dla zadania 1 i aplikacji FitPro krótki czas wykonywania skutkował małą liczbą fiksacji. W przypadku zadania 4, pomimo dłuższego czasu wykonywania zadania, liczba fiksacji jest mniejsza dla aplikacji FitPro niż Zepp Life. Wpływ na taki stan rzeczy miały odstające wyniki niektórych użytkowników, pokazane graficznie na wykresie 9 w postaci długich wąsów. Odwrotna sytuacja miała miejsce w przypadku zadania 5, które mimo tego, że było szybciej wykonywane dla aplikacji FitPro, to z kolei charakteryzowało się większą liczbą fiksacji. Można więc wysnuć wniosek, że użytkownicy mieli problem ze znalezieniem/rozpoznaniem odpowiedniej funkcji i w efekcie z wykonaniem odpowiednich czynności.



Rysunek 9: Liczba fiksacji podczas realizacji poszczególnych zadań na dwóch testowanych aplikacjach.

Kolejna metryka - liczba mrugnięć jest powiązana z wysiłkiem kognitywnym i jest do niego odwrotnie proporcjonalna. Oznacza to, że im mniejszy wysiłek poznawczy, tym częstsze mrugnięcia [11]. Wykres z Rysunku 10 pokazuje niejednoznaczne wyniki średniej liczby mrugnięć dla poszczególnych zadań realizowanych na dwóch aplikacjach. Przy wykonywaniu zadań 1 i 3 na aplikacji Zepp Life użytkownicy wkładali mniejszy wysiłek poznawczy. Z kolei podczas realizacji zadań 2 i 5 uczestnicy wykonywali więcej mrugnięć, pracując z aplikacją FitPro, co wskazuje na mniejszy wysiłek kognitywny. Natomiast dla zadania 4 wyniki średnich liczb mrugnięć były zbliżone – wysiłek poznawczy był na tym samym poziomie, ale rozpatrując medianę, to mniejszy wysiłek objawiał się podczas interakcji z aplikacją Zepp Life.



Rysunek 10: Liczba mrugnięć podczas realizacji poszczególnych zadań na dwóch testowanych aplikacjach.

5.3. Wyniki ankiety

Dwunastu badanych wypełniło krótką ankietę składającą się z sześciu pytań. Odpowiedzi na pytania, miały formę liczbową w skali od 0 do 10 i oznaczały odczucia i zadowolenie uczestników po interakcji z interfejsami obu aplikacji. Wyniki zostały uśrednione i zaprezentowane w Tabeli 4.

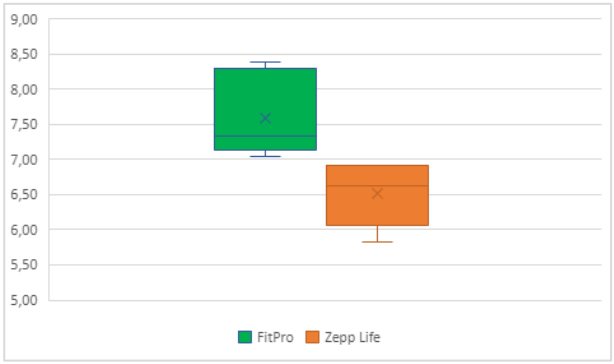
Zdecydowanie większym uznaniem cieszyła się aplikacja FitPro, której wszystkie oceny były wyższe od ocen Zepp Life. FitPro uzyskała średnią ocenę 7,59 (SD = ±0,59), a Zepp Life 6,51 (SD = ±0,44) w skali 10-cio

stopniowej. Wyniki te w formie graficznej przedstawia wykres na Rysunku 8.

Tabela 4: Wyniki ankiety dla aplikacji FitPro i Zepp Life

Lp.	Pytanie z ankiety	Średnia ocen	
		FitPro	Zepp Life
1	Czy aplikacja była łatwa w obsłudze?	7,22	6,13
2	W jakim stopniu aplikacja spełniła Twoje oczekiwania pod względem funkcjonalności?	7,04	6,61
3	Czy łatwo było Ci się zorientować w interfejsie aplikacji?	7,17	5,83
4	Czy byłeś zadowolony z jakości wyświetlanych informacji w aplikacji?	8,26	6,91
5	Czy aplikacja była atrakcyjna i przyjazna dla użytkownika pod względem wizualnym?	8,39	6,91
6	Jak całościowo oceniasz aplikację?	7,43	6,65

Oceny użytkowników wykazały znaczące różnice w trzech głównych aspektach: łatwości obsługi, wizualnej prezentacji interfejsu oraz dostępnych funkcjonalnościach. Badani w 91% odpowiadali, że poleciliby aplikacje FitPro, a w 83% Zepp Life. Według użytkowników FitPro nie miała żadnych problemów z płynnością obsługi, natomiast w Zepp Life występowały niewielkie niedogodności.



Rysunek 8: Wykresy pudełkowe ocen aplikacji według ankiety.

5.4. Identyfikacja i analiza błędów

W Tabelach 5 i 6 zaprezentowane zostały zaobserwowane błędy, jakie badani popełniali w takcie realizacji zadań wchodzących w skład scenariusza badawczego. Najczęstszym problemem było nieukończenie zadania 3 w aplikacji Zepp Life z powodu nieznalezienia funkcji "Waga".

Tabela 5. Błędy stwierdzone przez obserwatora w aplikacji FitPro

FitPro	
Z1	• Pominięcie funkcji, która znajduje się w głównym panelu na początku interfejsu.
Z2	-
Z3	-
Z4	• Trudność w odnalezieniu funkcji w interfejsie. • Trudność w korzystaniu z kalendarza.
Z5	• Szukanie rozwiązania w funkcji docelowej zamiast w ustawieniach aplikacji

Tabela 6. Błędy stwierdzone przez obserwatora w aplikacji Zepp Life

Zepp Life	
Z1	• Pominięcie funkcji, która znajduje się w głównym panelu na początku interfejsu. • Pomylenie funkcji tętna mierzonego automatycznie z ręcznym pomiarem tętna.
Z2	• Pomylenie funkcji spaceru z funkcją kroków.
Z3	• Trudność w odnalezieniu funkcji w interfejsie. • Nieukończenie zadania z powodu nieodnalezienia funkcji.
Z4	• Trudności w korzystaniu z kalendarza.
Z5	-

6. Wnioski

W przypadku aplikacji mobilnych, w których interakcja odbywa się za pośrednictwem niewielkiego interfejsu graficznego i ograniczonych zasobów przetwarzania danych, pozytywne doświadczenie użytkowników jest niezwykle istotne. Dlatego też, przed wprowadzeniem takiego oprogramowania na rynek, konieczne jest przeprowadzenie dokładnych i wszechstronnych testów. Najczęściej w takim przypadku stosuje się podejście mieszane polegające na wykorzystaniu dwóch metod. Przeprowadzone w ramach pracy badania zostały wykonane za pomocą dwóch narzędzi: eyetrackera oraz kwestionariusza ankiety.

Według ocen użytkowników zebranych za pomocą autorskiej ankiety, aplikacja FitPro została jednoznacznie wyżej oceniona niż aplikacja Zepp Life. Uczestnicy badań stwierdzili, że praca z aplikacją FitPro daje większą satysfakcję niż aplikacja Zepp Life. Na podstawie uzyskanych wyników należy uznać, że hipoteza H1 powinna zostać odrzucona, ponieważ aplikacja bogatsza w funkcjonalności została oceniona niżej niż aplikacja z ograniczoną liczbą funkcjonalności.

Dokonano również pomiarów czasu realizacji scenariuszy dla każdej aplikacji. Średni czas dla aplikacji FitPro wyniósł 31,0 s, a dla aplikacji Zepp Life 36,1 s. Z przeprowadzonych pomiarów można wyciągnąć wniosek, że obsługa pierwszej aplikacji była łatwiejsza. Także analiza błędów popełnianych przez użytkowników podczas obsługi obu aplikacji wykazała, że mniej tych błędów wystąpiło podczas operowania na aplikacji FitPro. Te dwie kwestie tzn. szybkość obsługi i podatność na błędy użytkownika wpłynęły na ocenę aplikacji przez użytkowników i w ten sposób potwierdziły hipotezę H2.

Aplikacja Zepp Life zawiera więcej funkcjonalności niż FitPro. Uzyskała ona niższy (302 ms) niż w przypadku drugiej aplikacji (307 ms) średni czas trwania fiksacji podczas realizacji scenariuszy badawczych. Wyniki te poświadczają większą złożoność oprogramowania Zepp Life i potwierdzają prawdziwość hipotezy H3.

Również wyniki średniej liczby fiksacji dla aplikacji FitPro i Zepp Life, wynoszące odpowiednio 72 i 96 fiksacji wskazują na wyższy poziom trudności tego drugiego narzędzia. Te obserwacje dają powód do potwierdzenia słuszności hipotezy H4.

Badania i otrzymane w ich efekcie wyniki pozwalają na sformułowanie wniosków oraz dostarczają dobrych praktyk w zakresie projektowania interfejsów aplikacji mobilnych. Opracowując interfejs, należy zachować umiar i równowagę. Liczba dostępnych obiektów i funkcji nie może być za duża, aby interfejs nie był zbyt przeładowany, złożony i skomplikowany. Istotne jest również przemyślane, optymalne rozmieszczenie elementów dających szybki dostęp do funkcji aplikacji. Interfejs powinien być prosty i wydajny. Kolejna kwestia to intuicyjność – użytkownicy powinni szybko uczyć się obsługi aplikacji. Nie bez znaczenia jest również estetyka i atrakcyjność graficzna. Wszystkie te elementy należy sprawdzać na etapie projektowania, bo to one w pierwszej kolejności mają wpływ na rynkowe powodzenie aplikacji.

Literatura

[1] P. Perea, P. Giner, UX Design. Projektowanie aplikacji dla urządzeń mobilnych, Helion, Gliwice, 2019.

[2] J. Nielsen, R. Budiu, Funkcjonalność aplikacji mobilnych. Nowoczesne standardy UX i UI, Helion, Gliwice, 2013.

[3] M. Borys, M. Plechawska-Wójcik, Badanie użyteczności oraz dostępności interfejsu w aplikacjach mobilnych. Wydawnictwo Uniwersytetu Rzeszowskiego, Nierówności społeczne a wzrost gospodarczy 35 (2013) 63-78.

[4] M. Oczoś, Projektowanie aplikacji mobilnych z wykorzystaniem preferencji użytkownika, praca magisterska, 2022.

[5] P. Łania, M. Pasławska, Wpływ mobilnych aplikacji sportowych na zainteresowanie aktywnością fizyczną wśród dorosłych Polaków. Zeszyty Naukowe. Turystyka i Rekreacja 2(16) (2015) 203-213.

[6] S. Ambler, User Interface Design: Tips and Techniques, Cambridge University Press, Toronto, 2000.

[7] A. Bojko, Eye tracking the user experience: A practical guide to research, Rosenfeld Media, Brooklyn, NY, 2013.

[8] J.T. Park, H.S. Hwang, I.Y. Moon, Study of Wearable Smart Band for a User Motion Recognition System, International Journal of Smart Home 8(5) (2014) 33-44, https://doi.org/10.14257/ijsh.2014.8.5.04.

[9] Pupil Labs, Pupil Invisible. Technical Specs & Performance, https://pupil-labs.com/products/invisible/tech-specs/, [20.06.2023].

[10] A. Stolińska, M. Andrzejewska, Metodologiczne aspekty stosowania techniki eye trackingowej w badaniach edukacyjnych, Przegląd Badań Edukacyjnych 24 (2017) 259-276, http://dx.doi.org/10.12775/PBE.2017.015.

[11] A. Andrychowicz-Trojanowska, Parametry okoruchowe ucznia szybko czytającego, Applied Linguistics Papers 25(3) (2018) 89-105.

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 399–404
Received: 24 August 2023
Accepted: 28 September 2023

Comparison of Machine Learning Performance on Classification of COVID-19 Cough Sounds Using MFCC Features

Porównanie wydajności uczenia maszynowego w zakresie klasyfikacji odgłosów kaszlu COVID-19 przy użyciu funkcji MFCC

Muhammad Thoriq Hidayat, Mohammad Reza Faisal*, Dwi Kartini, Fatma Indriani, Irwan Budiman, Triando Hamonangan Saragih

Department of Computer Science, Lambung Mangkurat University, Banjarbaru 70714, Indonesia

Abstract

Early detection for COVID-19 has now been widely developed. One of the methods used is cough audio detection. This research aims to classify cough audio. Audio feature extraction is performed using Mel Frequency Cepstral Coefficients (MFCC) to obtain numerical features. Feature classification uses SVM, Random Forest, and Naive Bayes methods. Evaluation is done to find the best classification method. The evaluation results in this study show that SVM Kernel RBF produces the best evaluation value with an AUC value of 0.657715.
Keywords: audio cough; SVM; Random Forest; Naive Bayes

Streszczenie

Wczesne wykrywanie COVID-19 zostało obecnie szeroko opracowane. Jedną z zastosowanych method jest wykrywanie dźwięku kaszlu. Badania te mają na celu klasyfikację dźwięku kaszlu. Ekstrakcję próbek audio wykonano przy użyciu Mel Frequency Cepstral Coefficients (MFCC) w celu uzyskania cech numerycznych. Klasyfikacja cech odbywa się przy użyciu metod SVM, Random Forest i Naive Bayes. Wyniki oceny w tym badaniu pokazują, że SVM Kernel RBF daje najlepszą wartość oceny z wartością AUC wynoszącą 0.657715.
Słowa kluczowe: dźwiękowy kaszel; SVM; losowy las; Naiwny Bayes

*Corresponding author
Email address: : reza.faisal@ulm.ac.id (M. R. Faisal)

1. Introduction

Techniques in detecting Covid-19 in patients have developed from rapid antigens and swab tests to thorax images with the Convolutional Neural Network (CNN) is method [1]. One of the new methods for Covid-19 detection can be done through cough audio. Classification is also applied to cough audio detection as a new method of Covid-19 detection. Cough audio is processed into images and classified with the CNN method [2].

Audio can be converted into other forms, including a spectrum image. Spectrum graphic images are generated from the audio frequency with color to represent the intensity of the signal carried [3]. The graph of the audio signal can be extracted into vectors to be used in classification. The extraction uses Mel Frequency Cepstral Coefficients (MFCC) so that important information related to the audio signal is still carried [4]. MFCC was compared with Zero Crossing Rate (ZCR) and Linear Predictive Coding (LPC) on voice samples, and it found that MFCC had better results [5].

Support vector machine (SVM) research with Linear and Radial Basic Function (RBF) kernels can classify MFCC extraction on Linear kernels with 92.5% accuracy while RBF kernels with 52.6% accuracy [6]. The accuracy value of 83% obtained from the classification results of the SVM method is better than Naive Bayes on unbalanced data. In contrast, the accuracy value of 87% on balanced data obtained from the classification results of the Naive Bayes method is better than SVM.[7]. Based on research [8], music genre detection was performed on music audio data with the random forest method and obtained 80.28% accuracy.

The results obtained from previous research still require further development in the early detection of Covid-19 so that a system is needed that is faster and easier to detect Covid-19. This dataset has never been used for related research before.

The purpose of this research is to determine the performance of machine learning on the classification of covid-19 cough sounds, namely:

1. What is the classification performance of SVM using a combination of parameters and kernels on MFCC extraction?
2. What is the classification performance using a combination of n parameters of the Random Forest estimator?
3. What is the classification performance of the Naïve Bayes model?

2. Literature overview

Random forest classification of music audio genres using music audio data on GitHub, which has 26 features and ten labels with a total of 1000 data, obtained 75% accuracy by determining the number of trees, 72% accuracy obtained from determining three different

parameters, 82% accuracy obtained from determining four different parameters [8].

Audio segmentation with MFCC feature extraction uses two types of data: balanced and unbalanced. The best accuracy value for unbalanced data is 0.83 using SVM, and the best value for balanced data is 0.87 using Naïve Bayes [7]. Audio cough detection of coswara and sarcos data amounted to 1079 healthy data and 92 positive data in the Coswara dataset, while 26 negative data and 18 positive data for the Sarcos dataset. The dataset is classified with several algorithms, including LR, SVM, KNN, MLP, CNN, LSTM, and Resnet 50, and then a comparison is made with the best results on Resnet 50 getting an AUC value of 0.96 while SVM is 0.815 [9].

Research by Matin and Valles [3] The accuracy value obtained is 77% on the recognition of autism children's and classified with SVM. Covid-19 detection using the DiCOVA dataset, data in the form of cough audio as many as 1040 records with 75 records of cough sounds of Covid-19 sufferers using SVM and Random Forest classification with Z-score normalization obtained AUC random forest value 82.15 and SVM AUC value 85.05 [10]. Detection of respiratory disorders in COVID patients collected by medical students using MFCC and spectrogram feature extraction obtained CNN accuracy increased from 87.04% to 96.38% [11].

3. Experiment Setting

3.1. Environment and Libraries

The hardware and supporting software specifications used in this research can be seen in Table 1.

Table 1: Specification of the computer

Hardware	
name	Version/description
CPU	Intel(R) Core (TM) i7-9750H
GPU	NVIDIA GeForce GTX 1650
RAM	24 GB DDR4 2666Hz
GPU Driver	NVIDIA 526.98
Software	
OS	Windows 11 pro
package	
matplotlib	3.5.3
scikit-learn	1.0.2
numpy	1.21.6
torch	1.13.0
pandas	1.3.5
librosa	0.10.0.post2

3.2. Classification used

This research will focus on processing cough audio data, extracted into images with Spectrogram and MFCC converts spectrogram images into numeric. Spectrograms images are segmented into frames, each representing the distribution of frequency energy. The frequency domain is transformed into the Mel domain, followed by applying the Discrete Cosine Transform (DCT) to the logarithm of the Mel filter bank, resulting in cepstral coefficients representing the audio signal. These MFCC coefficients serve as features for audio analysis and recognition. formula (1) can be used in convert spectrogram images to numeric with MFCC [12].

$$'\boldsymbol{FCC}(\boldsymbol{n},m) = \frac{1}{N}\sum_{k=0}^{N-1}\log(\sum_{k=0}^{N-1}X(k,n)e^{-j2\pi km/N})\,H_m(k) \qquad (1)$$

When *n* is audio frame, *m* is cepstral coefficient, *N* is number of frames used in the fast Fourier transform, *X(k,n)* is FFT value of frame *n* at frequency *k,* and $H_m(k)$ is filterbank Mel ke-m at frekuensi-k.

The extracted data will be classified using three classification methods, namely SVM, Random Forest, and Naive Bayes.

In SVM, hyperplane optimization can be done by optimizing parameters. One of the parameters that can be optimized is the parameter C [13]. SVM is a learning system to make predictions in classification cases. SVM can be used formula (2) in linear cases [6].

$$\boldsymbol{K}(x_i, x_j) = x.y \qquad (2)$$

When K is kernel used on SVM, *x* and *y* is points in the data that form a vector representing values in the classification.

The use of kernels in SVM can be used for data classification needs that cannot be solved in a linear way. One of these kernels is RBF. the following formula (2) is used in the RBF kernel SVM equation [14].

$$\boldsymbol{K}(x_i, x_j) = exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \qquad (3)$$

When K is kernel used on SVM, *x* and *y* is points in the data that form a vector representing values in the classification, and σ is parameters used in the RBF kernel.

Random Forest is a form of decision tree developed as a solution to overfitting in decision trees. Random forest is run by performing the regular random selection. The following formula (3) is used in Random Fores equation [8]. Random Forest is built to minimize bias, and in the construction of each tree, no pruning is done and done randomly [15].

$$Gini\ (S) = 1 - \sum pi2\ k\ i = 1 \qquad (4)$$

When *pi* is the probability of S belonging to class i

Naive Bayes is one of the classification methods. This method is considered simple and easy to use. In using Naive Bayes, not too much experimental data is needed. Formula (5) is used in Naive bayes equation [7].

$$(C|F_1, \dots, F_n) = \frac{p(C)p(F_1, \dots, F_n|C)}{p(F1, \dots, Fn)} \qquad (5)$$

When $p(C|F_1, \dots, F_n)$ is the posterior probability, $p(C)$ is the prior probability of class C, $p(F_1, \dots, F_n|C)$ is the probability likelihood, and

$p(F1, \ldots, Fn)$ is the prior probability of the instance $(F1, \ldots, Fn)$.

3.3. Used dataset

The dataset used in this research is the COVID-19 Cough Classification data from Alex Wer22ben's Kaggle. The number of cough sound datasets is 1926 audio with two labels, namely 1284 Healthy cough sounds and 642 Covid-19 cough sounds.

Figure 1 and 2 shows the shape of the cough audio data in the dataset used, , with the x-axis displaying the audio frequency value while the y-axis displays the audio time value. Figure 1 is audio with the label healthy, and Figure 2 is audio with the label covid-19.

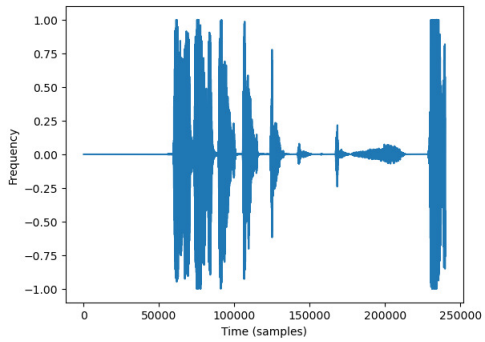


Figure 1: Audio data image healthy label.

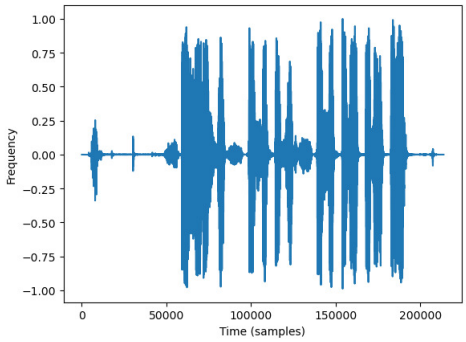


Figure 2: Audio data image covid label.

The audio data is equalized to 5 seconds in duration and then made into a spectrogram image. Figure 3 presents the spectrogram processing results of the audio data with the healthy class.

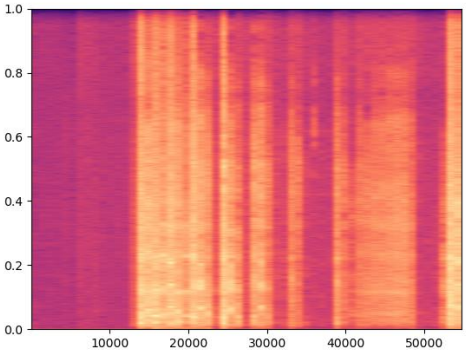


Figure 3: Spectrogram image of labelled healthy audio.

Figure 4 presents the results of processing the Spectrogram of audio data with the covid-19 class. The two-dimensional image produced in the Spectrogram is based on the frequency of the audio.

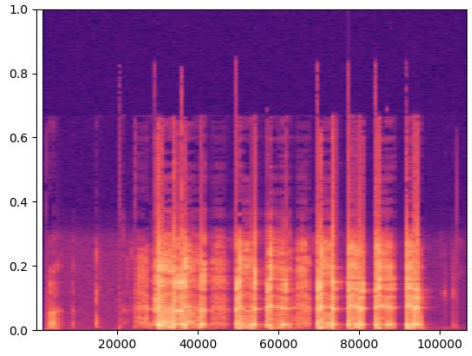


Figure 4: Spectrogram image of covid label.

The colors used in the image represent the altered audio frequency information. The results of the Spectrogram are then extracted into numerical data using MFCC. The resulting data includes 26 features.

3.4. Research Flow

Figure 5 shows the research flow used in this study as follows:

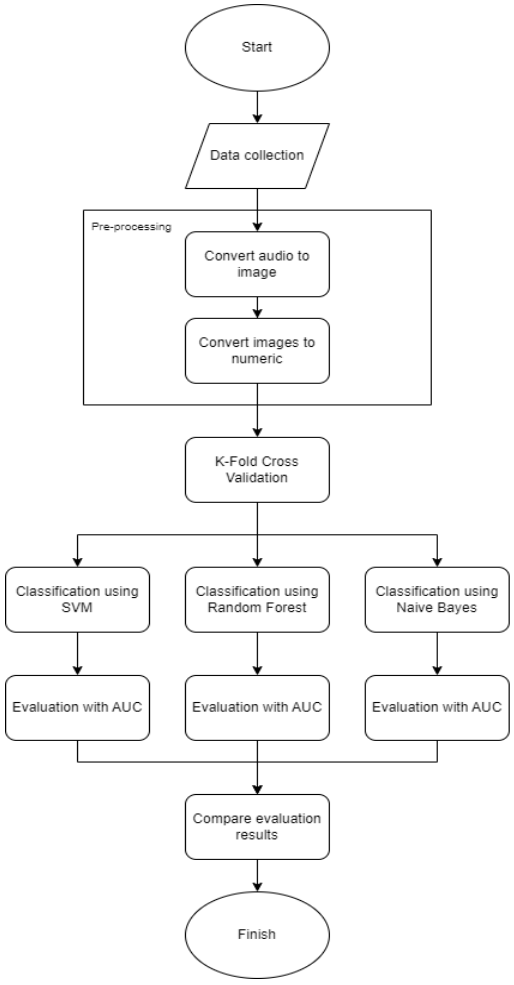


Figure 5: Research flow.

In the initial stage, data will be taken from Kaggle as audio. The data is preprocessed by cutting it with the same duration of five seconds. Once the data has the same duration, the audio data is divided into small segments that represent frequencies and measure their contribution. All segments are assembled into a 2D image with the x-axis representing time, the y-axis representing frequency, and the color or intensity representing the amplitude or power of the frequency component at each time. Darker areas indicate higher frequency energy [16]. The image represents time, the other axis represents frequency, and each point's color indicates the points' amplitude[17]. The image data is extracted again using MFCC to convert the voice data into structured numerical data with 26 extracted features.

Division of training and testing data of MFCC flare feature extraction results using K-fold Cross Validation to minimize overfitting [18]. The K value used is ten, dividing the data into ten partitions with the same number[19]. The classification methods used are SVM, Random Forest, and Naive Bayes.

The classification results are evaluated using the Confusion matrix because the data is not balanced, so the value used as a reference is AUC [20]. Classification evaluation is based on True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN) and is calculated according to the performance calculation used [21]. A comparison of AUC value evaluation is carried out to determine the best classification used in the early detection of covid-19 from cough audio.

4. Result

The results of this study were obtained from 4 classification methods and each classification was carried out 10 experiments. The following are the results obtained. Figure 6 presents the AUC results of linear SVM given changes in the value of C from 0.1 to 1.0, and the best results were found in SVM Linear C: 0.9 with an AUC of 0.60. the following confusion matrix results from the best parameters.

Figure 6: Linear SVM AUC Results.

Figure 7 presents the confusion matrix results of the SVM kernel linear. The following confusion matrix results are obtained from SVM kernel linear classification at the gamma scale parameter and C: 0.9.

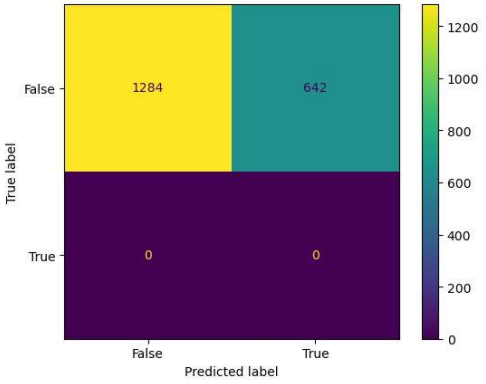


Figure 7: Confusion matrix SVM Linear.

Figure 8 presents the AUC results of SVM RBF given a change in the value of C from 0.1 to 1.0 and found the best results in SVM RBF C: 0.9 with an AUC result of 0.66. the following confusion matrix results from the best parameters.

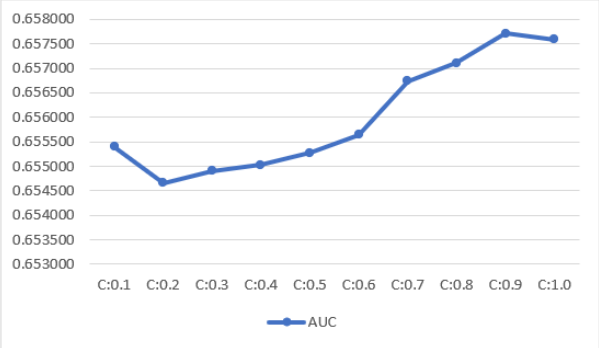


Figure 8: SVM RBF AUC Results.

Figure 9 presents the confusion matrix results of the RBF kernel SVM. The following confusion matrix results are obtained from SVM kernel RBF classification at the gamma scale parameter and C: 0.9.

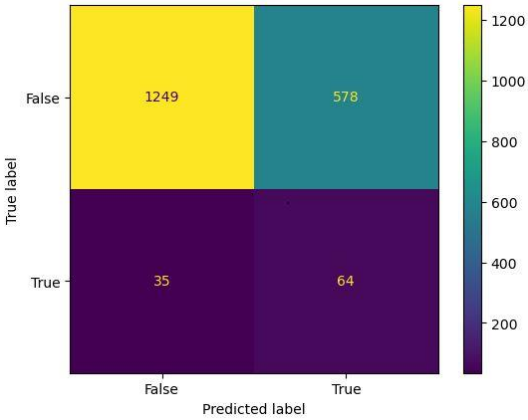


Figure 9: Confusion matrix SVM RBF.

Figure 10 presents the AUC results of Random Forest. The n estimator parameter is a parameter used to determine the number of trees to be made in performing random forest classification. Given changes in the value of the n estimator from 10 to 100, the best results were found in the random forest n estimator 100 with an

AUC result of 0.61. the following confusion matrix results from the best parameters.

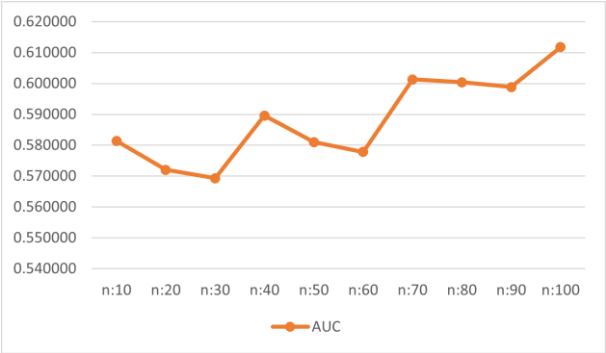


Figure 10: Random Forest AUC Results.

Figure 11 presents the confusion matrix results of Random Forest and those obtained from Random forest classification at parameter n estimator 100.

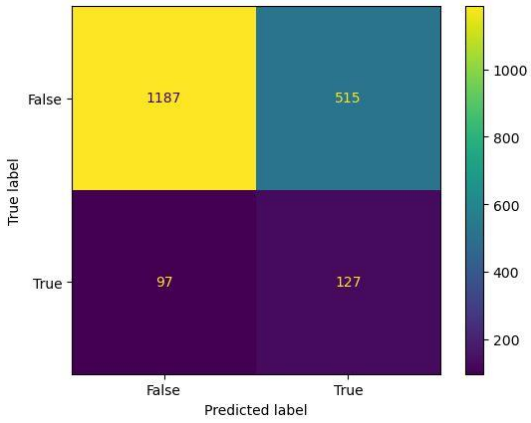


Figure 11: Confusion matrix Random Forest.

Figure 12 below is the confusion matrix result obtained from the Naive Bayes classification, which gets an AUC result of 0.64.

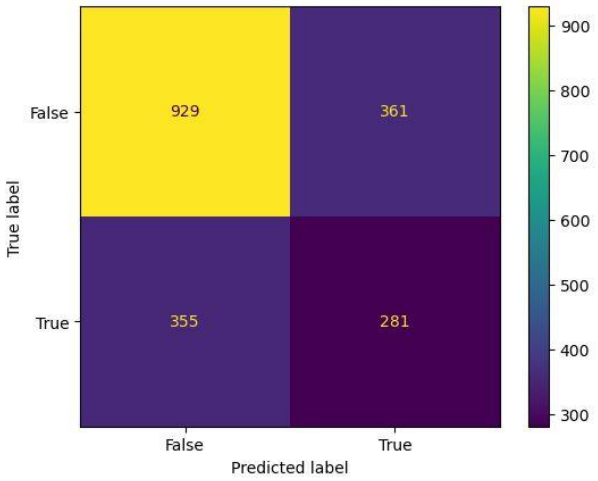


Figure 12: Confusion matrix Naive Bayes.

Table 2 shows the comparison results between the AUC values of each classification method used.

Table 2: Comparison of AUC values

Method	AUC
Support Vector Machine RBF	0.66
Support Vector Machine Linear	0.60
Random Forest	0.61
Naive Bayes	0.64

The order of AUC values in Table 2 is SVM kernel RBF 0.66, Naive Bayes with 0.64, Random Forest with 0.61, and SVM Linear with 0.60. Figure 13 displays a comparison chart of the AUC values of each classification method used.

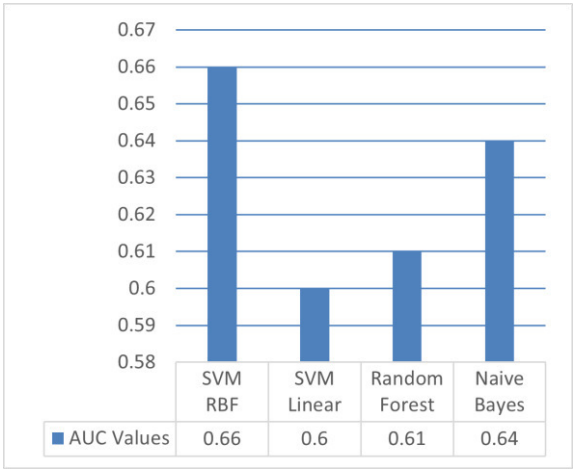


Figure 13: Chart Comparison of AUC values.

5. Conclusions

The conclusion is that using different classifications can produce different values in each AUC value obtained, and changes in parameters can provide changes to the results obtained. In SVM, the changed parameter value increases the value of the results obtained, while in Random Forest, several parameters increase and decrease.

This study found that SVM classification with RBF kernel is considered the best for early detection of covid-19 in cough audio, while for Linear SVM, there is still overfitting based on the confusion matrix results obtained.

For future research, it is recommended to try using deep learning, such as Convolutional Neural Networks (CNN). Further research can also add normalization to the data, using hyperparameters and adding extraction methods. Data can be normalized with min-max and Z-Score. As for hyperparameters, they can be used to get better results. That can be added to the extraction stage in the form of Chroma Constant-Q Transform, Chroma Energy Normalized Statistics, Chroma Variable-Q Transform, Mel Spectrogram, Spectral Contrast, Poly Features, Tonnetz and others.

Acknowledgements

The computation time of the computer system utilized in this study was bestowed by the Data Science Lab of the Computer Science Department, Faculty of Mathematics and Natural Sciences at Lambung Mangkurat

University. This endeavour was sustained by the Program Dosen Wajib Meneliti (PDWM) grant from PNBP Lambung Mangkurat University.

References

[1] S. Hassantabar, M. Ahmadi, A. Sharifi, Diagnosis and detection of infected tissue of COVID-19 patients based on lung X-ray image using convolutional neural network approaches, Chaos Solitons Fractals 140 (2020) 110-170, https://doi.org/10.1016/j.chaos.2020.110170.

[2] V. Bansal, G. Pahwa, N. Kannan, Cough Classification for COVID-19 based on audio mfcc features using Convolutional Neural Networks, in 2020 IEEE International Conference on Computing, Power and Communication Technologies (GUCON) IEEE (2020) 604–608, https://doi.org/10.1109/GUCON48875.2020.9231094.

[3] R. Matin, D. Valles, A Speech Emotion Recognition Solution-based on Support Vector Machine for Children with Autism Spectrum Disorder to Help Identify Human Emotions, in 2020 Intermountain Engineering, Technology and Computing (IETC) IEEE (2020) 1–6, https://doi.org/10.1109/IETC47856.2020.9249147.

[4] G. Karaca, Y. Kutlu, Turkish voice commands based chess game using gammatone cepstral coefficients, arXiv preprint arXiv:2101.08441 (2021) https://doi.org/10.48550/arXiv.2101.08441.

[5] N. Chauhan, T. Isshiki, D. Li, Speaker Recognition Using LPC, MFCC, ZCR Features with ANN and SVM Classifier for Large Input Database, in 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), IEEE (2019) 130–133, https://doi.org/10.1109/CCOMS.2019.8821751.

[6] S. R. Chaudhary, S. N. Kakarwal, J. V. Bagade, Feature selection and classification of indian musical string instruments using SVM, Indian Journal of Computer Science and Engineering 12(4) (2021) 859–867, https://doi.org/10.21817/indjcse/2021/v12i4/211204142.

[7] L. O. Iheme, Ş. Ozan, Multiclass digital audio segmentation with MFCC features using naive Bayes and SVM classifiers, in 2019 Innovations in Intelligent Systems and Applications Conference (ASYU) (2019) 1–5, https://doi.org/10.1109/ASYU48272.2019.8946441.

[8] N. Ndou, R. Ajoodha, A. Jadhav, Music genre classification: A review of deep-learning and traditional machine-learning approaches, in 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS) (2021) 1–6, https://doi.org/10.1109/IEMTRONICS52119.2021.9422487.

[9] M. Pahar, M. Klopper, R. Warren, T. Niesler, COVID-19 cough classification using machine learning and global smartphone recordings, Comput Biol. Med. 135 (2021) https://doi.org/10.1016/j.compbiomed.2021.104572.

[10] I. Södergren, M. P. Nodeh, P. C. Chhipa, K. Nikolaidou, G. Kovács, Detecting COVID-19 from Audio Recording of Coughs Using Random Forests and Support Vector Machines, in Interspeech 2021, ISCA (2021) 916–920, https://doi.org/10.21437/Interspeech.2021-2191.

[11] M. M. Gauy, M. Finger, Audio MFCC-gram Transformers for respiratory insufficiency detection in COVID-19, arXiv preprint arXiv:2210.14085 (2022) https://doi.org/10.48550/arXiv.2210.14085.

[12] M. B. Alsabek, I. Shahin, A. Hassan, Studying the Similarity of COVID-19 Sounds based on Correlation Analysis of MFCC, in 2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI), IEEE (2020) 1–5, https://doi.org/10.1109/CCCI49893.2020.9256700.

[13] A. S. Elkorany, M. Marey, K. M. Almustafa, Z. F. Elsharkawy, Breast Cancer Diagnosis Using Support Vector Machines Optimized by Whale Optimization and Dragonfly Algorithms, IEEE Access 10 (2022) 69688–69699, https://doi.org/10.1109/ACCESS.2022.3186021.

[14] A. Razaque, M. Ben Haj Frej, M. Almi'ani, M. Alotaibi, B. Alotaibi, Improved support vector machine enabled radial basis function and linear variants for remote sensing image classification, Sensors 21(13) (2021) 4431, doi: https://doi.org/10.3390/s21134431.

[15] E. Kubera, A. Wieczorkowska, A. Kuranc, and T. Słowik, Discovering Speed Changes of Vehicles from Audio Data, Sensors 19(14) (2019) 3067, https://doi.org/10.3390/s19143067.

[16] K. Palanisamy, D. Singhania, and A. Yao, Rethinking CNN models for audio classification, arXiv preprint arXiv:2007.11154, (2020) https://doi.org/10.48550/arXiv.2007.11154.

[17] Y. Zeng, H. Mao, D. Peng, Z. Yi, Spectrogram based multi-task audio classification, Multimed Tools Appl, 78(3) (2019) 3705–3722, https://doi.org/10.1007/s11042-017-5539-3.

[18] N. Fazakis, V. G. Kanas, C. K. Aridas, S. Karlos, S. Kotsiantis, Combination of active learning and semi-supervised learning under a self-training scheme, Entropy 21(10) (2019) 988, https://doi.org/10.3390/e21100988.

[19] D. Berrar, Cross-Validation, in Encyclopedia of Bioinformatics and Computational Biology, Tokyo: Elsevier (2019) 542–545, https://doi.org/10.1016/B978-0-12-809633-8.20349-X.

[20] V. Maeda-Gutiérrez *et al.*, Comparison of convolutional neural network architectures for classification of tomato plant diseases, Applied Sciences 10(4) (2020) 1245 https://doi.org/10.3390/app10041245.

[21] J. R. Maria Navin, R. Pankaja, Performance analysis of text classification algorithms using confusion matrix, International Journal of Engineering and Technical Research (IJETR) 6(4) (2016) 75–78.

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 405–412
Received: 7 September 2023
Accepted: 24 September 2023

Comparative analysis of package managers Flatpak and Snap used for open-source software distribution

Analiza porównawcza menedżerów pakietów Flatpak i Snap wykorzystywanych do dystrybucji oprogramowania o otwartym kodzie

Grzegorz Jan Cichocki*, Sławomir Wojciech Przyłucki

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
This article presents the result of a research of the Flatpak and Snap package managers used to distribute open-source software on Linux systems. Both package managers are characterised by their versatility and implementation of sandboxing. As part of the research, a test application was prepared, which was built in the Flatpak and Snap formats and published in the official software repositories, where for Flatpak it is Flathub and for Snap it is the Snap Store. The prepared application was first used to test and compare the implementation of sandboxing rules. This was followed by tests of RAM usage and start-up time by the application installed in both formats. The result of the research is an analysis of the measurement results and the drawing of conclusions.

Keywords: Flatpak; Snap; package manager; open-source; application performance

Streszczenie
Artykuł przedstawia wynik badań nad menedżerami pakietów Flatpak oraz Snap wykorzystywanych do dystrybucji oprogramowania o otwartym kodzie w systemach Linux. Obydwa menedżery pakietów cechują się uniwersalnością oraz implementacją systemu piaskownicy (ang. sandboxing). W ramach badań przygotowana została aplikacja testowa, którą zbudowano w formatach Flatpak i Snap, a także opublikowano w oficjalnych repozytoriach z oprogramowaniem, gdzie dla Flatpak jest to Flathub, a dla Snap – Snap Store. Przygotowaną aplikację najpierw wykorzystano do zbadania i porównania implementacji zasad piaskownicy. Następnie przeprowadzono testy użycia pamięci RAM oraz czasów uruchamiania przez aplikację zainstalowaną w obydwóch formatach. Wynikiem badań jest analiza uzyskanych pomiarów oraz wyciągnięcie wniosków.

Słowa kluczowe: Flatpak; Snap; menedżer pakietów; otwarty kod; wydajność aplikacji

*Corresponding author
Email address: grzegorz_cichocki@protonmail.com (G. J. Cichocki)

1. Wstęp

W dziedzinie dystrybucji oprogramowania o otwartym kodzie, menedżery pakietów odgrywają kluczową rolę w zapewnieniu efektywnego i bezpiecznego sposobu dostarczania aplikacji do użytkowników. W ostatnich latach dwie technologie, tj. Flatpak i Snap, wyłoniły się jako znaczące rozwiązania w tym obszarze. Obie te platformy przyniosły nowatorskie rozwiązania w zakresie pakowania i dystrybucji aplikacji, a ich popularność nadal rośnie wśród twórców oprogramowania oraz użytkowników systemów Linux.

Podczas badań skupiono się na kilku kluczowych zagadnieniach. Przeanalizowano implementację zasad piaskownicy (ang. sandboxing), która jest istotnym aspektem w kontekście bezpieczeństwa i izolacji aplikacji. Następnie przeprowadzono testy wydajności, w postaci pomiaru użycia pamięci RAM i czasu uruchamiania aplikacji w obu formatach. Wyniki tych badań dają szersze spojrzenie na Flatpaka i Snapa pod kątem efektywności i wydajności.

Niniejszy artykuł stanowi rezultat przeprowadzonych badań, których celem jest dogłębna analiza porównawcza menedżerów pakietów Flatpak i Snap w kontekście dystrybucji oprogramowania open-source. Skoncentrowano się na kluczowych aspektach technicznych, efektywności i bezpieczeństwa. Opracowana analiza opiera się na konkretnych badaniach, w trakcie których przygotowano aplikację testową o nazwie „Tabela", zbudowaną zarówno w formacie Flatpak, jak i Snap. Następnie opublikowano tę aplikację w oficjalnych repozytoriach, tj. Flathub dla Flatpaka [1] oraz Snap Store dla Snapa [2].

Aplikacja Tabela jest prostym narzędziem pozwalającym wyświetlić zawartość wybranego pliku CSV w formie graficznych wierszy i kolumn. Dodatkowo, zaznaczane są puste komórki, a także istnieje możliwość dodania kolejnych wierszy. Użytkownik może również zapisać dane do formatu CSV lub XML. Aplikacja została napisana z wykorzystaniem języka programowania Python oraz bibliotek graficznych Qt w wersji 6 (udostępnione dla języka Python jako biblioteka PySide6). Kod źródłowy aplikacji został udostępniony w repozytorium Codeberg [3]. W celu publikacji aplikacji w Snap Store oraz Flathub zostały przygotowane pliki manifestu. Repozytorium Github [4] zawiera plik manifestu dla sklepu z aplikacjami Flathub, a repozytorium Codeberg [5] plik manifestu napisany z myślą o sklepie z aplikacjami Snap Store.

1.1. Menedżer pakietów

Menedżer pakietów (ang. Package Manager) to oprogramowanie zazwyczaj wykorzystywane w systemach Linux do instalacji, aktualizacji i usuwania oprogramowania, na przykład aplikacji tekstowych, z interfejsem graficznym czy dodatkowych bibliotek (ang. libraries) oraz innych modułów. Instalacja oprogramowania z wykorzystaniem menedżera pakietów jest procesem koherentnym, to znaczy, że za każdym razem wygląda podobnie. Oprogramowanie jest pobierane z zaufanego serwera zwanego repozytorium i jest ono zazwyczaj już w wersji skompilowanej pod architekturę procesora komputera wykorzystywanego przez użytkownika. Wraz z plikami binarnymi potrzebnymi do instalacji programu, pobierane są inne metadane, które zawierają, między innymi, opis aplikacji, informacje o autorach, licencji i inne dane identyfikacyjne, a także dodatkowe, wymagane zależności oraz skrypty instalacyjne. Menedżery również często sprawdzają skróty kryptograficzne, na przykład sumy kontrolne SHA-256, aby upewnić się o autentyczności i zgodności ze źródłem pobranych pakietów. Dodatkowo, niektóre menedżery pakietów mogą również wykorzystywać podpisy cyfrowe do weryfikacji.

1.2. Flatpak

Flatpak to jeden z najpopularniejszych uniwersalnych menedżerów pakietów. Może być on wykorzystywany w wielu dystrybucjach i działa na warstwie systemowej i aplikacji. Menedżer pakietów wywodzi się z projektu „xdg-app" opracowywanego w 2014 roku. Flatpak bazuje na pracach wykonanych przez programistę Alexandera Larssona, a w projekt zainwestowało wiele znanych firm, w tym Red Hat, Endless Computers oraz Collabora.

Użytkownicy mogą zarządzać aplikacjami Flatpak na różne sposoby, zarówno za pomocą interfejsów graficznych, jak i tekstowych. Ważnym narzędziem jest aplikacja Flatseal pozwalająca zarządzać uprawnieniami. Aplikacje Flatpak są dystrybuowane za pomocą repozytoriów, gdzie Flathub jawi się jako największy katalog, zawierający ponad 2800 programów. Dostępne są zarówno aplikacje o otwartym kodzie, jak i własnościowe. Można także instalować aplikacje za pomocą plików .flatpak i .flatpakref.

Cechą wyróżniającą Flatpak jest pakowanie aplikacji w kontenery, które zawierają wszelkie niezbędne zależności. Podczas pracy, aplikacja znajduje się w izolowanym środowisku, tak zwanej piaskownicy, która zazwyczaj zawiera program oraz środowisko uruchomieniowe. Aplikacja działa nad warstwą systemu, dzięki czemu może być uruchamiana w wielu dystrybucjach.

Flatpak wykorzystuje mechanizm portali (ang. portals), co pozwala skonteneryzowanym aplikacjom komunikować się z innymi usługami systemowymi. Mowa tu, między innymi, o systemie plików, czy dostępie do mikrofonu. Kolejnym, ważnym składnikiem aplikacji Flatpak jest środowisko uruchomieniowe (ang. runtime). Każda aplikacja musi być zbudowana w oparciu o środowisko uruchomieniowe. Środowiska uruchomieniowe zapewniają również podstawowe składniki wykorzystywane przez programy. W jednym systemie można zainstalować wiele środowisk uruchomieniowych, w tym w wielu wersjach. Środowiska uruchomieniowe są niezależne od dystrybucji i poszczególnych wersji systemów operacyjnych.

Ze względu na to, że Flatpak nie wykorzystuje dostępnych w systemie bibliotek, menedżer pobiera wszelkie zależności ponownie. Aby ograniczyć zajętość dyskową, wprowadzono mechanizm deduplikacji środowisk uruchomieniowych i innych składników. Flatpak w pewnym stopniu jest w stanie współdzielić pewne moduły, aby obniżyć zajętość dyskową, a deduplikacja ma tym bardziej pozytywny wpływ, im więcej aplikacji Flatpak zainstalowano w systemie.

1.3. Snap

Snap to uniwersalny menedżer pakietów stworzony przez firmę Canonical, zaprojektowany dla systemów Linux. Pierwotnie, Snap był dostępny tylko w dedykowanej dystrybucji Ubuntu Core, jednak w 2016 roku został rozszerzony na systemy desktopowe.

Użytkownik może zainstalować oprogramowanie Snap na różne sposoby, w tym poprzez Snap Store dostępny w systemach Ubuntu. Oprogramowanie w Snap Store pochodzi z centralnego repozytorium, dostarczanego zarówno przez niezależnych twórców, jak i firmy komercyjne. W Snap Store dostępne jest oprogramowanie o otwartym kodzie oraz własnościowe.

Moduł dystrybucji oprogramowania w Snap jest kontrowersyjny, ponieważ choć sama technologia Snap jest otwarta, tak Snap Store pozostaje jedynym repozytorium zdalnym, nad którym kontrolę ma wyłącznie firma Canonical, a kod źródłowy repozytorium nie jest dostępny publicznie. Menedżer pakietów Snap został zoptymalizowany tylko pod kątem Snap Store oraz dystrybucji z wykorzystaniem lokalnych plików .snap.

Snap bazuje na systemie inicjalizacji Systemd dostępnym w większości dystrybucji. Jest to moduł potrzebny, ponieważ w systemie operacyjnym wykorzystującym menedżer Snap uruchamiany jest proces działający w tle (ang. deamon) o nazwie „snapd". Jego zadaniem jest ciągła i dynamiczna obsługa aplikacji oraz procesów zainstalowanych za pomocą tego menedżera pakietów. Aplikacje Snap są również automatycznie aktualizowane w tle.

Snap wykorzystuje mechanizmy izolacji. Zainstalowana aplikacja działa w specjalnym kontenerze, co znacznie zwiększa bezpieczeństwo i prywatność użytkownika. W celu dostępu do wybranych elementów, na przykład urządzeń USB, mikrofonu czy kamery wideo, aplikacje Snap wykorzystują moduł o nazwie „Interfejsy" (ang. Interfaces). Interfejsy Snap często wykorzystują domyślne interfejsy API systemów Linux oraz standaryzowane interfejsy XDG Desktop Portals. Użytkownik może dowolnie zarządzać uprawnieniami danych aplikacji do poszczególnych elementów. Modyfikacja uprawnień jest możliwa z wykorzystaniem aplikacji Snap Store oraz interfejsu tekstowego w konsoli. Programy Snap są dystrybuowane razem ze wszelkimi

zależnościami potrzebnymi do działania oprogramowania, tym samym nie są zależne od składników systemowych.

Snap wykorzystuje również system plików SquashFS. Jest to system plików tylko do odczytu (ang. read-only filesystem) stosowany w systemach operacyjnych, który został zaprojektowany do efektywnego kompresowania i przechowywania danych. Aplikacje Snap są zwykle opakowane w pojedyncze pliki SquashFS o rozszerzeniu „.snap", które zawierają wszystkie pliki, biblioteki i zależności wymagane do działania aplikacji. Metadane zawarte w pliku SquashFS są interpretowane przez snapd w celu skonfigurowania odpowiednio ukształtowanej bezpiecznej piaskownicy dla aplikacji. Po zainstalowaniu aplikacji Snap, w systemie użytkownika plik jest montowany i dekompresowany wtedy, kiedy to potrzebne. W systemie plików pliki Snap są widziane jako abstrakcyjne urządzenia w formacie „loop".

Unikatową zaletą uniwersalnego menedżera pakietów Snap jest możliwość dostarczania różnych typów aplikacji, a także całych stosów oprogramowania. W repozytorium Snap Store dostępne są nie tylko aplikacje z interfejsem graficznym czy narzędzia konsolowe, ale także pełne aplikacje serwerowe zawierające wiele składników, w tym serwer WWW, silnik bazodanowy, interfejs użytkownika oraz wykonywalny kod.

2. Przegląd literatury

2.1. Menedżery pakietów

Wynikiem prac nad rozwiązaniem, które pomogłyby w zarządzaniu pakietami, jak i aplikacjami oraz ich zależnościami, są menedżery pakietów. Jak zauważa autor publikacji [6] nie tylko instalacja oprogramowania stała się łatwiejsza, ale też ich aktualizacja oraz utrzymanie. O wiele łatwiej mają też deweloperzy, twórcy oprogramowania. Publikacja aplikacji jest prostsza, a użytkownik ma ułatwioną instalację – również poprzez oprogramowanie z interfejsem graficznym GUI (ang. Graphical User Interface). Wokół menedżerów pakietów zbudowano wiele aplikacji, które za pomocą kilku kliknięć myszą pozwalają na instalację określonego oprogramowania. Największą zaletą wykorzystywania menedżerów pakietów jest ponowne wykorzystywanie komponentów zewnętrznych (ang. third-party) bez ponoszenia konsekwencji w postaci przywołanego wcześniej zjawiska „Dependency Hell" oraz w formie lepszego wykrywania problemów z kompatybilnością jeszcze na etapie produkcji oprogramowania.

Autorzy pracy [7] również podkreślają znaczenie nowoczesnych menedżerów pakietów jako niezwykle ważnego składnika w procesie dystrybucji oprogramowania o otwartym kodzie, jednak także zauważają problemy wynikające z ich architektury. Jak zaznaczają, aktualnie wykorzystywane menedżery pakietów cechują się budową monolityczną i doraźnymi mechanizmami rozwiązywania problemów z zależnościami oraz niestandardowymi heurystykami. Zdaniem autorów, rozwiązaniem dostrzeżonych problemów mogą być modułowe menedżery pakietów.

W pracy [8] skupiono się na bezpieczeństwie menedżerów pakietów. W ramach badań przeprowadzono audyt bezpieczeństwa menedżera pakietów CPAN (ang. Comprehensive Perl Archive Network) dla języka Perl. Autorzy również zbadali popularne menedżery pakietów pod kątem szerokiej gamy wytycznych dotyczących bezpieczeństwa. Badanie integracji wykazało, że niektóre z menedżerów pakietów nie wykorzystują nawet tak prostej metody zabezpieczającej jak sprawdzanie sum kontrolnych danych udostępnionych oraz danych pobranych. Jak wynika z badań, menedżer cabal-install nie sprawdza sum kontrolnych, a EasyInstall tylko wtedy, kiedy takową sumę kontrolną dostarczy autor oprogramowania. Dla kontrastu, CPAN zawsze bada sumy kontrolne.

2.2. Uniwersalne menedżery pakietów

Standardowe menedżery pakietów w systemach Linux często są powiązane z konkretnymi dystrybucjami. Dystrybucja Debian i bazująca na niej dystrybucja Ubuntu wykorzystuje APT (ang. Advanced Packaging Tool), a dystrybucja Fedora oferuje DNF (ang. Dandified YUM). Autor pracy [9] przedstawia problematykę standardowych menedżerów pakietów. Rozważane są takie zagadnienia jak przywiązanie danego menedżera do jednej dystrybucji, co utrudnia dystrybucję oprogramowania, a także o utrudnionym dostępie do nowszych wersji aplikacji – często dana wersja wybranej aplikacji jest ściśle powiązana z wersją systemu operacyjnego.

W omawianym artykule wskazano, że uniwersalne menedżery pakietów są rozwiązaniem opisanych problemów. Autor skupia się na menedżerze Flatpak, który jest dostępny w wielu dystrybucjach rodziny systemów Linux, a więc jednocześnie można wykorzystywać go na wielu różnych systemach.

Uniwersalne menedżery pakietów nie zawsze są pozytywnie oceniane. Popularnym artykułem jest [10], gdzie autor wskazuje powody, przez które nie uważa menedżera pakietów Flatpak za dobre rozwiązanie dystrybucji aplikacji w systemach typu Linux. Wymienia, między innymi, potrzebę pobierania wielu megabajtów danych często dla prostych aplikacji. Za przykład podaje aplikację KCalc, gdzie sama aplikacja zajmuje 4,4 MB, jednakże wraz z nią użytkownik musi pobrać niemalże 900 MB innych modułów potrzebnych do uruchomienia. W publikacji wyszczególniono także problem z błędnie działającym systemem dzielenia środowisk wykonawczych (ang. runtime) dla aplikacji, a także zwiększone użycie pamięci operacyjnej RAM.

W odpowiedzi na artykuł [10] została przygotowana publikacja [11], w której skonfrontowano argumenty. Problem pobierania wielkich zasobów danych wytłumaczono potrzebą zachowania zgodności i pewności, że dane oprogramowanie będzie bezproblemowo działać. Użytkownicy uruchamiają dany program z takimi składnikami, które określił deweloper. Ma to również pozytywny wpływ na deweloperów, którzy nie będą musieli rozwiązywać wielu problemów zgłaszanych przez użytkowników, które wynikałyby właśnie z błęd-

nych wersji bibliotek czy środowisk wykonawczych (lub ich braku) u użytkownika.

Artykuł porusza także problem dużej zajętości dyskowej przez aplikacje i składniki zainstalowane poprzez menedżer Flatpak. Wskazano, że istnieją mechanizmy w tym menedżerze, których zadaniem jest zaoszczędzenie przestrzeni dyskowej, między wspomniana wcześniej deduplikacja środowisk wykonawczych. Zaznaczono również, że systemy z rodziny Linux, w których to Flatpak jest głównym menedżerem pakietów, często wykazują się mniejszą zajętością dyskową zaraz po instalacji, ponieważ nie zawierają innych modułów, które byłyby potrzebne do zarządzania, instalacji i uruchamiania aplikacji dostarczonych za pomocą tradycyjnych menedżerów pakietów.

2.3. Zasada piaskownicy

W publikacji [12] autorzy skupili się na badaniach zasad piaskownicy w systemach z rodziny Linux. Jako dwa prężnie rozwijane menedżery pakietów, które implementują funkcjonalność piaskownicy, wymieniono Flatpak oraz Snap. Autorzy zwracają również uwagę na fakt, że źródło takiej architektury bezpieczeństwa i prywatności ma korzenie w aplikacjach mobilnych, to znaczy aplikacji przeznaczonych na smartfony oraz tablety. Autorzy publikacji podkreślają, że transformacja zasad piaskownicy zależnych od użytkownika (lub konta użytkownika) na zasady oparte na samej aplikacji jest korzystne i wyraźnie widoczne w systemach typu Linux.

W ramach przeprowadzonych badań sprawdzono implementację zasad poprzez zbadanie 283 aplikacji dostępnych w obydwóch formatach, odpowiednio: Flatpak i Snap. Jak wynika z tych badań, aż 90,1% sprawdzonych aplikacji Snap implementuje zasady piaskownicy, natomiast w przypadku menedżera Flatpak jest to tylko 58,3%. Ponadto badacze zaznaczają, że autorzy oprogramowania niekiedy błędnie opisują zasady i uprawnienia swoich produktów. Bywa tak, że dokładnie ta sama aplikacja potrzebowała więcej uprawnień początkowych do działania w jednym menedżerze niż w przypadku drugiego menedżera, przy czym na obydwóch platformach działała ona poprawnie. Autorzy zaznaczają, że postęp w implementacji zasad piaskownicy jest coraz większy, ale nadal wymagane jest wiele usprawnień.

3. Eksperyment badawczy

3.1. Metodyka badań

Celem pracy jest przeprowadzenie analizy porównawczej dla dwóch uniwersalnych menedżerów pakietów w systemie Linux, czyli Flatpak i Snap. Porównanie skupia się na wydajności aplikacji okienkowej zainstalowane w oparciu o obie metody dystrybucji oprogramowania. Zostaną porównane także zasady piaskownicy dostępne w obu menedżerach pakietów w oparciu o aplikację testową. Badania zostaną podzielone na scenariusze badawcze opisane w dalszych podrozdziałach i zostaną przeprowadzone w sposób sekwencyjny z zachowaniem odpowiedniego czasu na ustabilizowanie się temperatury komputera przeznaczonego do wykonania testów. Otrzymane wyniki zostaną szczegółowo omówione i zaprezentowane w formie tabel, wykresów oraz z wykonaniem podstawowej analizy statystycznej.

3.2. Scenariusze badań

3.2.1. Scenariusz S1 – Porównanie zasad piaskownicy aplikacji na podstawie przygotowanej aplikacji badawczej o nazwie Tabela.

W tym scenariuszu zostanie przeprowadzona analiza porównawcza zasad piaskownicy aplikacji okienkowych w systemie Linux z wykorzystaniem menedżerów pakietów Flatpak i Snap. Do badania zostaną wykorzystane aplikacje: Flatseal (w przypadku menedżera Flatpak) oraz Snap Store (dla zbadania cech menedżera Snap), w których zostanie zliczona liczba zasad, którymi użytkownik programu może zarządzać. Porównanie zostanie przeprowadzone na podstawie oficjalnie wydanej aplikacji w oparciu o obydwie metody dystrybucji oprogramowania.

Aplikacja Flatseal to dedykowany program towarzyszący do obsługi aplikacji typu Flatpak. Za jego pomocą użytkownik jest w stanie zarządzać uprawnieniami programów. Aplikacja Snap Store pełni rolę przede wszystkim sklepu z oprogramowaniem dystrybuowanym za pomocą menedżera pakietów Snap. Jednakże ma ona też dodatkowe funkcje, w tym kontrolę uprawnień.

3.2.2. Scenariusz S2 – Zbadanie użycia pamięci operacyjnej RAM przez aplikację Tabela zainstalowaną za pomocą Flatpak i Snap

W tym scenariuszu badawczym zostanie zbadane użycie pamięci RAM aplikacji Tabela, która została zainstalowana za pomocą Flatpak oraz Snap. Użycie pamięci zostanie zmierzone wielokrotnie. Pierwsza seria badań uwzględni aplikację Tabela chwilę po uruchomieniu. Następnie zostanie wykonana druga seria badań użycia pamięci RAM przez aplikację Tabela z otworzonym przykładowym plikiem CSV.

Użycie pamięci RAM to jeden z najważniejszych czynników wpływających na wydajność aplikacji okienkowych. Badanie pozwoli wyłonić menedżer pakietów, którego sposób działania pozwala na mniejsze wykorzystanie pamięci operacyjnej. Badanie zostanie wykonane z użyciem narzędzia Sysstat.

3.2.3. Scenariusz S3 – Badanie czasu uruchomienia aplikacji Tabela zainstalowanej z wykorzystaniem metod dystrybucji Flatpak oraz Snap

W tym scenariuszu badawczym zostanie zbadany czas uruchomienia aplikacji Tabela, która została zainstalowana za pomocą Flatpak oraz Snap. Badanie zostanie przeprowadzone za pomocą narzędzia Hyperfine. Test przewiduje zbadanie czasu uruchomienia aplikacji po raz pierwszy od uruchomienia systemu, tak zwany zimny start (ang. cold start) oraz po wcześniejszym uruchomieniu i zamknięciu bez wyłączania systemu, co nazywane będzie gorącym startem (ang. warm start).

Czas uruchomienia to jeden z ważniejszych czynników wpływających na doświadczenia z korzystania z aplikacji okienkowych w systemie operacyjnym. Badając ten parametr można będzie wskazać ten menedżer pakietów, który został lepiej zoptymalizowany pod tym kątem.

3.3. Platforma testowa

Do wykonania badań został wykorzystany komputer typu laptop o podanej specyfikacji.

- Procesor: Intel Core i7-1165G7.
- Układ graficzny: Intel Iris Xe.
- Pamięć operacyjna RAM: 16 GB LPDDR4X.
- Dysk: Toshiba KXG60ZNV1T02 NVMe PCI-E 3.0 o pojemności 1 TB.
- System operacyjny: Ubuntu 23.04.

4. Wyniki badań

4.1. Wyniki dla scenariusza S1

Tabela 1 zawiera porównanie zasad piaskownicy menedżerów pakietów Flatpak i Snap na podstawie aplikacji Tabela. W tabeli wprowadzono podział na kategorie przedstawiony przez aplikację Flatseal. Są to kolejno: podsystemy udostępniane, gniazda piaskownicy, urządzenia, dodatkowe zezwolenia oraz system plików.

Tabela 1: Porównanie zasad piaskownicy menedżerów pakietów Flatpak i Snap na podstawie aplikacji Tabela

Flatpak	Snap
• Dostęp do sieci. • Umożliwienie komunikacji między procesami.	• Umożliwienie działania jako serwis sieciowy.
• Wykorzystanie systemu wyświetlania X11. • Wykorzystanie sytemu wyświetlania Wayland. • Możliwość wykorzystania systemu wyświetlania X11, jeśli istnieją problemy z Wayland. • Wykorzystanie serwera dźwięku Pulse-Audio • Dostęp do D-Bus sesji. • Dostęp do D-Bus systemu. • Dostęp do SSH. • Dostęp do karty mikroprocesorowej. • Dostęp do obsługi drukowania. • Dostęp do katalogu agenta GPG.	
• Możliwość wykorzystania GPU do przyspieszenia sprzętowego. • Dostęp do wirtualizacji KVM. • Dostęp do współdzielonej pamięci. • Dostęp do urządzeń komputera.	• Dostęp do stosu technologicznego OpenGL.
• Dostęp do programów innych architektur. • Dostęp do magistrali CAN. • Dostęp do pamięci współdzielonej dla aplikacji.	
• Możliwość precyzyjnego ustalenia dostępu do plików, w tym do konfiguracji systemowej, bibliotek systemowych, plików wykonywalnych lub danych statycznych, katalogu domowego użytkownika lub do ręcznie wybranego katalogu. • Dostęp do opcji działania w tle. • Możliwość wyświetlania powiadomień.	• Umożliwienie dostępu aplikacji do katalogu domowego użytkownika.

W przypadku Snap dodatkowe możliwości konfiguracji mogą zostać wyświetlone za pomocą polecenia „snap connections tabela”. Pomijając pozycje przedstawione w Tabeli 1, są to: desktop, desktop-legacy, wayland oraz x11.

Aplikacja Tabela w wersji Flatpak wykorzystuje uprawnienia:

- Umożliwienie komunikacji między procesami.
- Wykorzystanie systemu wyświetlania Wayland.
- Możliwość wykorzystania systemu wyświetlania X11, jeśli istnieją problemy z Wayland.
- Możliwość wykorzystania GPU do przyspieszenia sprzętowego.
- Dostęp do systemu plików (do katalogu „Dokumenty”).
- Możliwość działania w tle.

W przypadku menedżera pakietów Snap lista uprawnień aplikacji Tabela jest następująca:

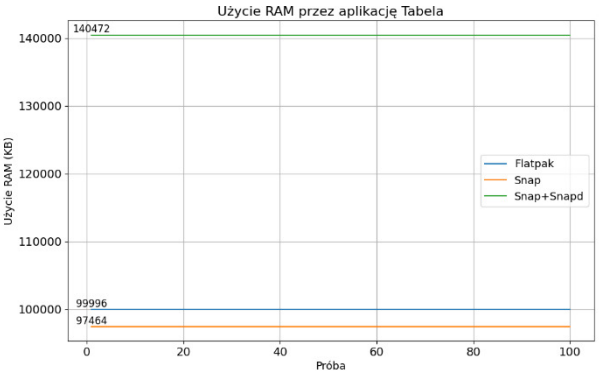
- Dostęp do systemu plików (katalog domowy użytkownika).
- Dostęp do systemu wyświetlania Wayland.
- Dostęp do systemu wyświetlania X11.
- Dostęp do zasobów środowiska graficznego.
- Dostęp do zasobów starszych środowisk graficznych.
- Dostęp do sposobu działania jako serwis sieciowy.

Pomimo dodatkowych uprawnień dostępnych z poziomu interfejsu tekstowego, to Flatpak z aplikacją Flatseal nadal umożliwia większą kontrolę nad piaskownicą w przypadku aplikacji testowej. Korzystając z aplikacji w wersji Flatpak użytkownik jest w stanie szczegółowo sprecyzować uprawnienia aplikacji dopasowując je do swoich potrzeb. Jest w stanie wybranej aplikacji wydzielić specjalny katalog, przełączyć się na serwer wyświetlania X11 (jeśli, na przykład, występują problemy z Wayland) czy wyłączyć wysyłanie wszelkich powiadomień. W przypadku menedżera pakietów Snap nie ma możliwości dostosowania większej liczby uprawnień niż wstępnie wymaga tego aplikacja. Jeśli program testowy byłby rozbudowany o dodatkowe

elementy wymagające dostępu do kolejnych modułów, na przykład kamery użytkownika lub mikrofonu, to lista uprawnień w Tabeli 1 byłaby dłuższa i konkretniejsza, tym samym nieco zmniejszając lukę pomiędzy Flatpak a Snap. Może to być szczególnie przydatne w przypadku aplikacji serwerowych, które właśnie mogą być udostępniane za pomocą menedżera pakietów Snap. Jednakże należy zaznaczyć, że dedykowane narzędzie graficzne pozwalające na kontrolę uprawnień Snap Store nie wyświetla pełnej listy uprawnień, która dopiero dostępna jest w interfejsie tekstowym. To niejako wymusza na użytkowniku zapoznanie się z niektórymi poleceniami terminalowymi, aby uzyskać jeszcze większą kontrolę nad piaskownicą.

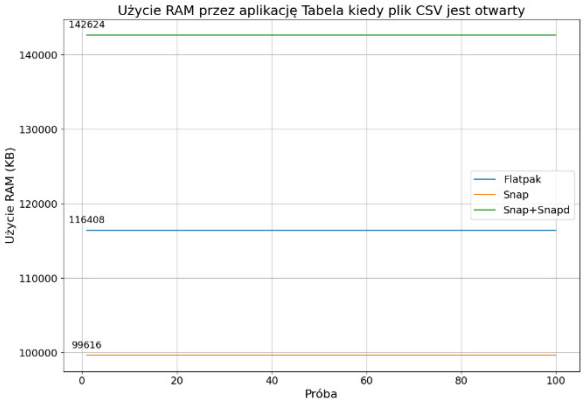
4.2. Wyniki dla scenariusza S2

W tym podrozdziale zostaną przedstawione wyniki dla scenariusza S2 polegającym na zbadaniu użycia pamięci RAM przez aplikację Tabela. Badanie podzielono na dwa podscenariusze. W pierwszym została uruchomiona sama aplikacja, natomiast w drugim podscenariuszu otworzono przykładowy plik CSV. Badanie wykonano za pomocą narzędzia Sysstat. Pomiary były wykonane 100 razy w odstępach 3 sekundowych pomiędzy każdym z pomiarów. Dla wszystkich wykonanych pomiarów, wynik pozostawał ten sam, dlatego wartość pierwszego badania jest zarazem wynikiem średniej statystycznej.



Rysunek 1: Wykres przedstawiający porównanie użycia pamięci RAM przez aplikację Tabela.

Na Rysunku 1 można dostrzec, że najlepszym wynikiem, to znaczy najmniejszym użyciem pamięci RAM, cechuje się aplikacja Tabela zainstalowana za pomocą menedżera pakietów Snap. Wykorzystanie pamięci w tym przypadku wyniosło 97464 KB. Aplikacja uruchomiona za pomocą Flatpak wykorzystuje nieco więcej pamięci operacyjnej, tj. 99996 KB. Należy zaznaczyć, że aplikacje Snap do działania potrzebują uruchomionego demona Snapd. Podczas badań zajętość pamięci RAM przez tego demona jest stała i wynosi 43008 KB. Łącząc wynik demona Snapd oraz aplikacji Tabela w wersji Snap, osiągany jest wynik 140472 KB.



Rysunek 2: Wykres przedstawiający porównanie użycia pamięci RAM przez aplikację Tabela z włączonym przykładowym plikiem CSV.

Rysunek 2 przedstawia porównanie użycia pamięci RAM przez aplikację Tabela z otworzonym przykładowym plikiem CSV. Podobnie jak w przypadku poprzedniego podscenariusza, tutaj ponownie najlepszym wynikiem wyróżnia się Snap. Co ważne, różnica pomiędzy Snap a Flatpak jest jeszcze większa. Aplikacja Tabela w wersji Snap wykorzystuje 99616 KB, kiedy to w wersji Flatpak 116408 KB. Jednakże aplikacja Snap do działania wymaga demona Snapd. Dodając użycie pamięci RAM przez demona wynik sumaryczny wynosi 142624 KB.

Aplikacja Tabela w wersji Snap cechuje się mniejszym użyciem pamięci operacyjnej RAM niż w wersji Flatpak i dotyczy to zarówno programu chwilę po uruchomieniu, jak i z otworzonym plikiem CSV. Jednakże aplikacje Snap do działania wymagają demona Snapd, który także wykorzystuje pamięć operacyjną. Sumaryczny wynik łączący zajętość aplikacji Snap i demona Snapd jest wyższy od wyniku Flatpak. Należy jednak zaznaczyć, że Snapd jest jednym demonem, który może zarządzać wieloma aplikacjami Snap. Dlatego też niekoniecznie należy go łączyć z zajętością pamięci przez samą aplikację.

4.3. Wyniki dla scenariusza S3

W badaniu porównano uruchamianie aplikacji Tabela zainstalowanej za pomocą Flatpak i Snap. Pierwsze uruchomienie odbyło się bezpośrednio po uruchomieniu systemu operacyjnego.

Rysunek 3: Wykres przedstawiający porównanie czasu uruchamiania aplikacji Tabela.

Wyniki badania ukazano na Rysunku 3. Uzyskane czasy uruchamiania są porównywalne i nie można zarejestrować większej różnicy. Wyjątkiem jest tutaj pierwsze uruchomienie aplikacji, gdzie w menedżerze pakietów Flatpak jest ono podobne jak w przypadku kolejnych prób, jednak w menedżerze Snap jest zauważalnie dłuższe.

Tabela 2: Dane statystyczne dotyczące czasów uruchamiania aplikacji

-	Flatpak	Snap
Średni czas uruchamiania [s]	1,0231	1,0304
Najkrótszy uzyskany czas uruchamiania [s]	1,0169	1,0165
Najdłuższy uzyskany czas uruchamiania [s]	1,0300	2,0266
Odchylenie standardowe [s]	0,0050	0,1007
Mediana [s]	1,0189	1,0177

Tabela 2 przedstawia podstawowe dane statystyczne dotyczące czasów uruchamiania aplikacji Tabela. Wartości średniej są do siebie zbliżone, choć niższą wartością cechuje się Flatpak. Ze względu na dużą liczbę prób uruchomienia, długi czas pierwszego włączenia aplikacji w wersji Snap nie wpłynął wyjątkowo zauważalnie na wynik średniej wartości. Jednakże odchylenie standardowe, a także najdłuższy czas uruchamiania aplikacji jasno wskazują na to, że wersja Snap aplikacji Tabela zapewnia mniej jednolite wyniki. Biorąc pod uwagę to, że podczas typowego korzystania z systemu operacyjnego, użytkownik uruchamia daną aplikację najczęściej raz lub kilka razy w trakcie trwania jednej sesji użytkowania, to zauważalnie dłuższy czas uruchamiania aplikacji pierwszy raz po włączeniu systemu w wersji Snap może być bardziej odczuwalny. Tym samym to właśnie wersja Flatpak zapewnia znacznie spójniejsze doświadczenia użytkowania.

Wynik badań udowadnia jedną z często wymienianych wad menedżera pakietów Snap. Pierwsze uruchomienie aplikacji po starcie systemu operacyjnego zawsze zajmuje więcej czasu. Jest to spowodowane wieloma elementami, choć najczęściej wymienia się potrzebę żądania pamięci podręcznej czcionek podczas uruchamiania. Problem został zaadresowany w 2019 roku i miał zostać rozwiązany w Snap 2.36.2 [13]. Niestety, badanie wykazuje, że problem dalej występuje.

5. Podsumowanie

Analizując wyniki uzyskane podczas badań można dostrzec, że pod względem wydajności Flatpak i Snap są podobnymi rozwiązaniami. W badaniu użycia pamięci RAM Snap uzyskiwał nieco lepsze wyniki, jednak są to niewielkie różnice, niemające większego wpływu na działanie komputera. W badaniu czasów uruchamiania obydwa rozwiązania uzyskiwały podobne wyniki, jednak wyjątkiem jest tutaj czas uruchomienia aplikacji Snap pierwszy raz. Jest to proces trwający zauważalnie dłużej, co w przypadku korzystania z komputera przez typowego użytkownika, który uruchamia dany program raz, maksymalnie kilka razy podczas jednej sesji, czas włączenia aplikacji ma spore znaczenie. W tym badaniu spójniejsze wyniki oferował Flatpak.

Jak wynika z badania S1, Flatpak wraz z narzędziem Flatseal oferują o wiele większe możliwości konfiguracji uprawnień. Użytkownik zyskuje sporą kontrolę nad tym, jak aplikacja działa oraz jakie ma możliwości. W przypadku menedżera pakietów Snap opcje konfiguracji są znacznie mniejsze.

Obydwa rozwiązania, tj. Flatpak i Snap, to nowoczesne podejście do dystrybucji aplikacji o otwartym kodzie w systemach Linux. Cechuje je uniwersalność, ponieważ mogą być wykorzystywane w różnych dystrybucjach i to niezależnie od wersji systemu. Wymienione menedżery pakietów zyskują na popularności i stają się domyślną formą dostarczania oprogramowania w coraz to większej liczbie dystrybucji. Wybierając aplikacje Flatpak lub Snap użytkownik zyskuje bardzo stabilną pracę, wygodę oraz nowoczesne formy dbania o bezpieczeństwo i prywatność. Użytkownik uzyskuje również sporą kontrolę nad oprogramowaniem dzięki systemowi uprawnień.

Flatpak i Snap to również spore ułatwienie dla deweloperów. Są to uniwersalne menedżery pakietów, więc przygotowując pakiet na jeden z nich (lub na oba) jednocześnie trafia się do ogromnej liczby dystrybucji, a nie tylko do jednej (lub jednej z rodzin dystrybucji). Deweloper ma również większą kontrolę nad środowiskiem wykonawczym. To powinno ograniczyć liczbę zgłaszanych błędów, które mogłyby wynikać z wykorzystywania niekompatybilnych składników systemowych, na przykład bibliotek.

Literatura

[1] Aplikacja Tabela w centralnym repozytorium Flathub, https://flathub.org/pl/apps/eu.cichy1173.tabela, [07.09.2023].

[2] Aplikacja Tabela w centralnym repozytorium Snap Store, https://snapcraft.io/tabela, [07.09.2023].

[3] Repozytorium zawierające kod źródłowy aplikacji Tabela, https://codeberg.org/cichy1173/tabela-flatpak, [16.09.2023].

[4] Repozytorium zawierające plik manifestu dla sklepu z aplikacjami Flathub, https://github.com/flathub/eu.cichy1173.tabela, [16.09.2023].

[5] Repozytorium zawierające plik manifestu dla sklepu z aplikacjami Snap Store, https://codeberg.org/cichy1173/Tabela, [16.09.2023].

[6] D. Spinellis, Package Management Systems, IEEE Computer Society 29 (2012) 84-86, https://doi.org/10.1109/MS.2012.38.

[7] P. Abate, R. Di Cosmo, R. Treinen, S. Zacchiroli, A modular package manager architecture, Information and Software Technology 55 (2012) 459-474, https://doi.org/10.1016/j.infsof.2012.09.002.

[8] A. Athalye, R. Hristov, T. Nguyen, Q. Nguyen, Package Manager Security, project report, Massachusetts Institute of Technology (2014).

[9] E. Kokot, Distribution-agnostic package management on Linux with Flatpak, thesis, University of Ljubljana (2022).

[10] N. Fraser, Flatpak is Not the Future, https://ludocode.com/blog/flatpak-is-not-the-future, [07.09.2023].

[11] H. Rana, Response to "Flatpak Is Not the Future", https://theevilskeleton.gitlab.io/2022/05/16/response-to-flatpak-is-not-the-future.html, [07.09.2023].

[12] T. Dunlap, W. Enck, B. Reaves, A Study of Application Sandbox Policies in Linux, Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies, (2022) 19-30.

[13] Uruchamianie aplikacji Snap, https://www.omgubuntu.co.uk/2019/03/the-cause-of-slow-snap-app-startup-times-has-been-identified, [07.09.2023].

JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 413–420
Received: 7 September 2023
Accepted: 24 September 2023

Analysis of the impact of using containerization techniques on application performance in Python

Analiza wpływu wykorzystania technik konteneryzacji na wydajność aplikacji w języku Python

Kacper Chołody*, Sławomir Wojciech Przyłucki

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

This article comprehensively evaluates the impact of two containerization environments, Docker and Podman, on the performance of Python applications. The paper characterizes the two tools and presents the differences in their architectures. The scope of the study covers three aspects. The first is a comparison of resource usage, such as CPU usage, RAM usage and execution time, during the calculation of the number π. The next step is to analyse the resource usage when sorting an ordered list. The final aspect of the research is a comparison of the start-up time of the container in both environments. The tests carried out allow the presence of a performance overhead in both containerization environments, with an average of 8%. In addition, it can be seen that there is better resource management in the case of the Podman tool and a more dynamic environment in the case of the Docker tool.
Keywords: containerization; performance comparison; Docker; Podman

Streszczenie

W niniejszym artykule dokonano kompleksowej oceny wpływu dwóch środowisk konteneryzacji, Dockera i Podmana, na wydajność aplikacji w języku Python. W pracy dokonano charakterystyki obu narzędzi oraz prezentacji różnic w ich architekturze. Zakres badań obejmuje trzy aspekty. Pierwszym z nich jest porównanie użycia zasobów, takich jak użycie procesora, pamięci RAM i czasu wykonania, w trakcie obliczeń liczby π. Kolejnym etapem jest analiza użycia zasobów podczas sortowania uporządkowanej listy. Ostatnim aspektem badań jest porównanie czasu startu kontenera w obu środowiskach. Przeprowadzone badania pozwalają na stwierdzenie występowania narzutu wydajności w obu środowiskach konteneryzacji, wynoszącego średnio 8%. Dodatkowo można zauważyć lepsze zarządzanie zasobami w przypadku narzędzia Podman oraz większą dynamikę środowiska w przypadku narzędzia Docker.

Słowa kluczowe: konteneryzacja; porównanie wydajności; Docker; Podman

*Corresponding author

Email address: kacper.cholody@pollub.edu.pl (K. Chołody)

1. Wstęp

Współczesna informatyka, zwłaszcza w kontekście wdrażania aplikacji i zarządzania nimi, znalazła się w epoce rewolucji. W miarę jak organizacje przechodzą od tradycyjnych, monolitycznych aplikacji w kierunku mikrousług i architektur opartych na kontenerach, konteneryzacja stała się jednym z kluczowych elementów tego przełomu. Jest to technologia, która pozwala na izolację aplikacji i ich zależności od infrastruktury, co umożliwia jednolite i niezawodne wdrażanie aplikacji na różnych platformach, niezależnie od systemu operacyjnego i środowiska.

Konteneryzacja umożliwia opakowanie aplikacji i wszystkich jej zależności, takich jak biblioteki, narzędzia i konfiguracje, w jednostkę zwaną kontenerem, która jest niezależna od środowiska, na którym jest uruchamiana. Dzięki temu możliwe jest uniknięcie problemów związanych z różnicami między środowiskami deweloperskimi, testowymi a produkcyjnymi, co jest częstym źródłem błędów podczas wdrażania aplikacji.

W tym kontekście narzędzia do konteneryzacji, takie jak Docker i Podman, odgrywają niezwykle ważną rolę. Są to platformy, które umożliwiają tworzenie, zarządzanie i uruchamianie kontenerów w sposób uproszczony. Dzięki nim programiści mogą pakować swoje aplikacje w kontenery i mieć pewność, że będą one działać tak samo na różnych maszynach i w różnych środowiskach. To ogromnie przyspiesza proces wdrażania aplikacji i pozwala zwiększyć niezawodność systemów.

Środowisko Docker, którego historia sięga 2013 roku, było jednym z pierwszych narzędzi do konteneryzacji, które zdobyło szeroką popularność. Jego prostota użycia i wsparcie dla kontenerów przyczyniły się do jego dominacji na rynku przez wiele lat.

Środowisko Podman natomiast, pomimo że młodsze, zdobyło znaczną popularność dzięki swojej elastyczności i filozofii "rootless", która pozwala uruchamiać kontenery bez konieczności uprawnień administratora. To narzędzie jest rozwijane jako część projektu OCI (ang. Open Container Initiative) i stanowi alternatywę dla narzędzia Docker, oferującą podobne możliwości, ale w bardziej modułowej i elastycznej formie.

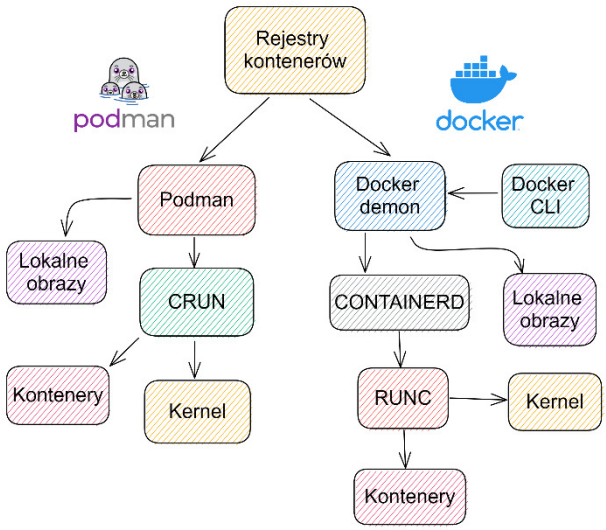
Wspólną cechą środowiska Docker i Podman jest to, że oba narzędzia mają ogromny wpływ na sposób, w jaki deweloperzy i administratorzy wdrażają

i zarządzają aplikacjami. Dzięki nim procesy te stają się bardziej skalowalne, niezawodne i bardziej zautomatyzowane, co ma kluczowe znaczenie w dzisiejszym dynamicznym świecie technologii informatycznych. Oba narzędzia również przyczyniły się do popularyzacji konteneryzacji jako standardowego podejścia do dostarczania i utrzymania oprogramowania, co stanowi kluczowy element transformacji cyfrowej organizacji na całym świecie.

2. Środowisko wirtualizacji Podman oraz Docker

Środowisko Docker jest jednym z najpopularniejszych narzędzi do konteneryzacji, ale przez ostatnie kilka lat mocno na popularności zyskały inne narzędzia, między innymi Podman. Podman jest to narzędzie do konteneryzacji i zarządzania kontenerami, które powstało jako alternatywa dla narzędzia Docker, oferując jednocześnie nowe podejście i szereg zalet w porównaniu do środowiska Docker, jednak jako młodsze nie jest aż tak rozbudowane, jak Docker.

Pomimo realizacji tych samych zadań, środowiska Docker i Podman różnią się znacząco pomiędzy sobą. Rysunek 1 przedstawia poglądową strukturę obu porównywanych środowisk konteneryzacji wraz z najważniejszymi wykorzystywanymi zależnościami.



Rysunek 1: Różnice w architekturze narzędzi Docker i Podman.

Architektura platform Docker i Podman różni się znacząco pod wieloma względami, wpływając na sposób, w jaki zarządzają i izolują kontenery oraz interakcje z nimi. Poniżej przedstawione są podstawowe różnice pomiędzy obydwiema architekturami:

- Architektura Docker [1]:
  - Demon i klient: Narzędzie Docker działa w oparciu o architekturę klient-serwer. Głównym komponentem jest Docker demon, który zarządza kontenerami, obrazami, sieciami itp. Klienci Docker komunikują się z demonem poprzez interfejs komend wiersza poleceń lub API REST.
  - Centralne zarządzanie: W środowisku Docker wszystkie operacje na kontenerach wykonywane są poprzez centralny demon. Wszystkie kontenery działające na danym hoście są zarządzane przez tego samego demona.
  - Jeden punkt awarii: Ponieważ Docker demon pełni kluczową funkcję w zarządzaniu kontenerami, awaria demona może wpłynąć na działanie wszystkich kontenerów na danym hoście.
- Architektura Podman [2]:
  - Bez demonów: W przeciwieństwie do środowiska Docker, Podman nie wymaga centralnego demona. Każdy kontener jest zarządzany przez oddzielny proces w przestrzeni użytkownika. Każdy kontener działa jako niezależny proces bez centralnego punktu zarządzania.
  - Zdecentralizowana architektura: Każdy kontener w środowisku Podman jest wykonywany jako osobny proces w przestrzeni użytkownika, co eliminuje potrzebę posiadania centralnego demona. To sprawia, że kontenery są bardziej izolowane
    i nie ma jednego punktu awarii dla wszystkich kontenerów.
  - Bez uprawnień administratora: W narzędziu Podman kontenery mogą działać w trybie "rootless", co oznacza, że nie wymagają uprawnień administratora. Każdy kontener jest wyizolowanym środowiskiem i może być zarządzany bez konieczności posiadania specjalnych uprawnień.
  - Mniejsza komunikacja: Bez potrzeby komunikacji z centralnym demonem, narzędzie Podman nie generuje dodatkowego ruchu sieciowego między klientem a demonem.
  - Większa izolacja: Ze względu na brak centralnego punktu zarządzania środowisko Podman dostarcza większej izolacji pomiędzy kontenerami.

3. Przegląd literatury

3.1. Porównanie wykorzystania procesora i pamięci RAM

W pracy [3] przedstawione jest porównanie wydajności dwóch konkurujących ze sobą rozwiązań dostarczających wirtualizację opartą o kontenery, to jest środowiska Docker i Podman. W tych badaniach porównanie wydajności opiera się na pomiarach użycia zasobów procesora, wykorzystania pamięci RAM oraz prędkości zapisu i odczytu danych z dysku. Do realizacji przeprowadzonych testów autor użył popularnych i dobrze znanych narzędzi do generowania syntetycznego obciążenia. W celu obciążenia procesora został wykorzystany Y-cruncher - program obliczający wartość liczby π przy użyciu wielu wątków. Na podstawie przedstawionych w pracy wyników można stwierdzić, że wykorzystanie konteneryzacji powoduje wymierny narzut na czas wykonania obliczeń dla liczby π, zarówno w przypadku narzędzia Docker, jak i Podman, a różnica czasu wykonania wynosi około 10%, w porównaniu do scenariusza testowego, w którym nie użyto rozwiązań wykorzystujących konteneryzację.

W badaniach przeprowadzonych przez autorów pracy [4] skupiono się między innymi na porównaniu stopnia wykorzystania procesora i pamięci RAM przez

systemy lekkiej wirtualizacji, jak i wykorzystujące nadzorców wirtualizacji (ang. hipervisor). Z przedstawionych wyników można wywnioskować, że biorąc za punkt odniesienia rozwiązanie niewykorzystujące wirtualizacji, środowiska lekkiej wirtualizacji opartej na kontenerach, takie jak Docker i Podman, oferują wydajność na bardzo zbliżonym poziomie. Różnice zawierają się w przedziale od 0,1% do 3% na korzyść lub niekorzyść tych rozwiązań w zależności od użytego benchmarku. Podman oferuje minimalnie lepszą wydajność w zastosowaniach, w których obciążenie jest wysokie. Różnica w tych zastosowaniach w porównaniu do narzędzia Docker wynosi około 2 – 3%.

W pracy [5] autor do pomiarów wydajności użył narzędzia „sysbench", w połączeniu z pomiarem czasu potrzebnego na odpowiedź. Otrzymane wyniki są zbieżne z wynikami badań opisanymi wcześniej i nie wskazują, aby wydajność rozwiązań wykorzystujących wirtualizację opartą o kontenery odbiegała w zauważalnym stopniu od innych sposobów uruchamiania aplikacji.

W [6] autorzy zbadali, jaki narzut na zasoby systemowe powoduje wykorzystanie środowiska Docker. Otrzymane przez nich wyniki stanowią ciekawy aspekt badań w omawianym obszarze, ponieważ wykazały one, że narzut rozwiązania Docker na wielkość wykorzystania zasobów jest tym większy, im mniejsze jest wykonywane zadanie. Narzut w przypadku zadań wymagających niewielkiej części z puli ogólnie dostępnych zasobów systemowych wynosi prawie 10% całkowitego użycia procesora i maleje do około 3% w przypadku zadań, które już wymagają dużo większej części z dostępnych zasobów.

Na podstawie wyników badań zaprezentowanych przez autorów pracy [7] wynika, że ogólna wydajność aplikacji uruchomionych w środowisku Docker jest dużo większa niż pełnej wirtualizacji w badaniu 7-zip polegającym na kompresji algorytmem LZMA pliku o rozmiarze 10 GB. Docker osiągnął wynik prawie dwukrotnie lepszy od maszyny wirtualnej, skracając czas kompresji z około 36 sekund do około 19 sekund. Dodatkowo liczba instrukcji na sekundę wykonywanych przez kontener uruchomiony w środowisku Docker jest średnio 4-krotnie większa, a w skrajnym przypadku niemal 8-krotnie większa od maszyny wirtualnej.

Podobne wyniki zostały zaprezentowane przez autorów pracy [8]. Jednak w tym przypadku uzyskane przez nich różnice w wydajności aplikacji uruchomionej w środowisku Docker oraz na maszynie wirtualnej są zdecydowanie bardziej zbliżone do siebie. Według opublikowanych wyników, w przypadku narzędzia Docker autorom udało się uzyskać wyniki wydajności lepsze o około 2%.

Wyniki testów przedstawionych w pracy [9], potwierdzają, że Docker ma tendencję do alokowania zbyt dużej ilości zasobów przy mniejszym obciążeniu. Autorzy w swojej pracy zbadali, że narzut rozwiązania Docker zmniejsza się wraz z liczbą uruchomionych kontenerów, przez co efektywne obciążenie procesora może praktycznie nie wzrosnąć przy większej liczbie kontenerów.

3.2. Czas uruchomienia kontenera

Czas potrzebny na uruchomienie kontenera jest jednym z istotnych czynników wpływających na ogólną ocenę wydajności danego rozwiązania lekkiej wirtualizacji. Temat ten został poruszony przez autora pracy [5], który wykorzystał zestaw kontenerów i wykonał pomiary czasu, który upłynął od wydania polecenia uruchomienia kontenera, do momentu otrzymania odpowiedzi z kontenera. Z otrzymanych wyników można wywnioskować, że czas potrzebny na uruchomienie kontenera przez narzędzia Docker i Podman jest przynajmniej 4 – 5 krotnie większy w porównaniu do innych rozwiązań.

Należy podkreślić, że w badaniach przeprowadzonych przez autorów pracy [10], w porównaniu do klasycznego podejścia z wykorzystaniem maszyn wirtualnych i pełnej wirtualizacji, czas potrzebny na uruchomienie aplikacji zbudowanej z wykorzystaniem kontenerów jest mniejszy o cały rząd wielkości i wynosi około 2 sekund, zamiast około 11 – 12 sekund w przypadku maszyn wirtualnych.

4. Metodyka i scenariusze badań

4.1. Metodyka badań

Celem pracy jest przeprowadzenie analizy porównawczej dla dwóch środowisk konteneryzacji, to jest Docker i Podman. Porównanie będzie dotyczyło takich parametrów jak czas wykonania operacji, średnie użycie procesora przez proces oraz średnie wykorzystanie pamięci operacyjnej komputera. Badania będą podzielone na scenariusze badawcze i będą wykonywane w sposób sekwencyjny z zachowaniem odpowiedniego czasu potrzebnego na ustabilizowanie temperatury komputera po poprzednim badaniu. W przypadku, gdy nie będzie można stwierdzić znaczących różnic między badanymi środowiskami dla konkretnych parametrów, dla których wystąpiło znaczne podobieństwo, badania zostaną powtórzone z pominięciem środowiska wirtualizacji, tak, aby zbadać ewentualny wpływ samej lekkiej wirtualizacji z pominięciem poszczególnych środowisk. Otrzymane wyniki zostaną szczegółowo omówione i zaprezentowane w formie tabel oraz wykresów wraz z wykonaniem podstawowej analizy statystycznej.

4.2. Scenariusze badań

4.2.1. Scenariusz S1 – testowanie obciążenia procesora przy wykorzystaniu obliczeń liczby π

W tym scenariuszu zostaną przeprowadzone badania z wykorzystaniem wielokrotnych obliczeń liczby π. przy wykorzystaniu formuły Leibniza. Formuła Leibniza [11] lub też szereg Leibniza może zostać wykorzystany do obliczeń przybliżonej wartości liczby π. Wzór Leibniza opiera się na szeregu nieskończonym, który może być sumowany do dowolnego stopnia dokładności. Wykorzystanie powyższych obliczeń może być skutecznym sposobem na generowanie stałego obciąże-

nia procesora, przy założeniu, że liczba wyrazów ciągu do obliczenia jest stosunkowo duża.

Parametrami, które będą mierzone w trakcie wykonywania obliczeń, będą:

- Czas wykonywania obliczeń.
- Średnie użycie procesora.
- Średnie użycie pamięci operacyjnej.

Do zmierzenia użycia procesora oraz wykorzystania pamięci RAM, zostały wykorzystany program *btop* oraz moduł *psutil* z języka Python, natomiast do pomiaru czasu wykorzystano moduły *time* oraz *pyperf*.

4.2.2. Scenariusz S2 – testowanie wykorzystania pamięci operacyjnej poprzez sortowanie uporządkowanej listy

W tym scenariuszu przeprowadzone zostaną badania sortowania już uporządkowanej listy, z wykorzystaniem standardowej implementacji funkcji sortującej dostępnej w bibliotekach języka Python [12]. Głównym zadaniem skryptu testującego będzie alokacja znacznej części systemowej pamięci operacyjnej tak, aby zbadać implementację funkcjonalności środowisk konteneryzacji odpowiedzialnych za zarządzanie pamięcią.

4.2.3. Scenariusz S3 – testowanie czasu uruchomienia kontenera

Ten scenariusz polega na pomiarze czasu wymaganego na wykonanie tak zwanego zimnego startu kontenera. Do testów wybrany został standardowy kontener języka Python w wersji 3.11.4. Czas potrzebny na uruchomienie został zdefiniowany jako czas od momentu wydania polecenia do uruchomienia kontenera do momentu otrzymania odpowiedzi na komendę *print("Hello world!")*.

5. Wyniki badań

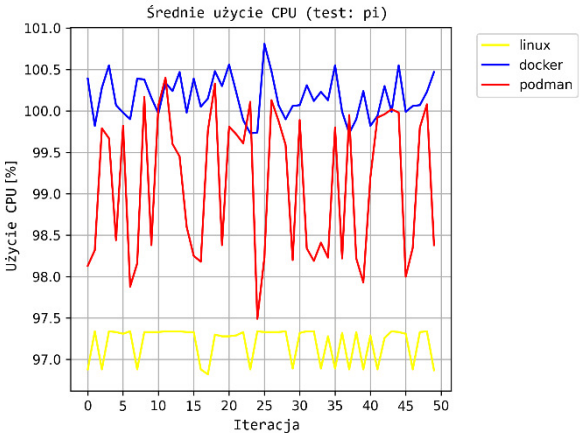
5.1. Wyniki dla scenariusza S1

Na Rysunku 2 przedstawiony został wykres średniego użycia procesora dla obliczeń liczby π. Wykres przedstawia pomiary z 50 uruchomień testów dla każdej z testowanych platform. Linia w kolorze żółtym oznacza procentowe zużycie procesora dla uruchomienia bez konteneryzacji, bezpośrednio w systemie Linux. Linia w kolorze czerwonym oznacza uruchomienie z wykorzystaniem platformy Podman. Linia w kolorze niebieskim oznacza uruchomienie na platformie Docker. Mniejsze użycie procesora oznacza lepszy wynik. Dokładniejsze wyniki oraz podstawowe wartości statystyczne zostały przedstawione w Tabeli 1.

Tabela 1: Wyniki zużycia CPU dla obliczeń liczby π

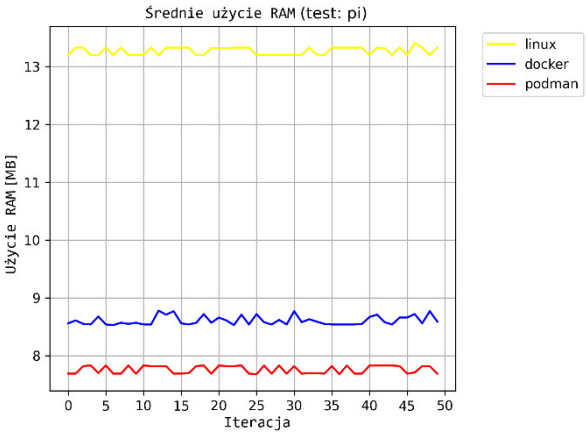
Platforma	Minimum (%)	Średnia (%)	Maksimum (%)	σ z próby
Linux	96,82	97,20	97,34	0,20
Docker	99,73	100,17	100,81	0,25
Podman	97,49	99,11	100,40	0,87

Na podstawie wykresu z Rysunku 6 oraz wyników przedstawionych w Tabeli 1 można stwierdzić, że najlepszy wynik w przeprowadzonym teście osiągnęło uruchomienie bez wykorzystania konteneryzacji, bezpośrednio w systemie Linux. Co warto zauważyć, w przypadku platformy Docker, generowane przez nią obciążenie przez znaczną część czasu wynosiło ponad 100%, oznacza to, że zadania w tym przypadku musiały czekać na wolny czas procesora. Należy jednocześnie podkreślić, że wyniki te dotyczą jednowątkowej aplikacji testowej.



Rysunek 2: Wykres przedstawiający średnie użycie procesora dla scenariusza S1.

Największe wahania wyników można zaobserwować w przypadku platformy Podman, a platformą generującą najbardziej jednorodne obciążenie był system Linux.



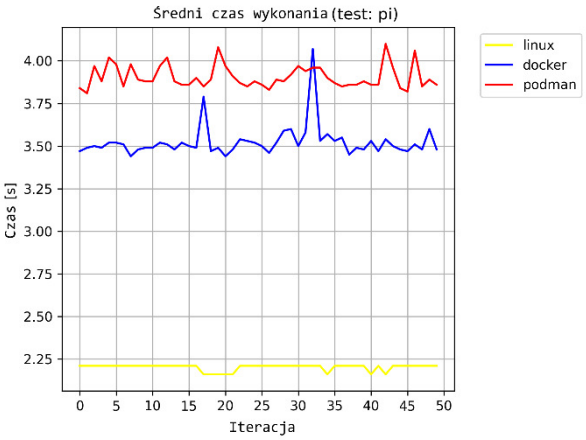
Rysunek 3: Wykres przedstawiający średnie użycie pamięci RAM dla scenariusza S1.

Tabela 2: Wyniki zużycia pamięci RAM dla obliczeń liczby π

Platforma	Minimum (MB)	Średnia (MB)	Maksimum (MB)	σ z próby
Linux	13,20	13,27	13,41	0,06
Docker	8,53	8,61	8,78	0,08
Podman	7,68	7,76	7,83	0,07

Na postawie wykresu z Rysunku 3 oraz danych przedstawionych w Tabeli 2 można stwierdzić, że najgorzej w trakcie przeprowadzonego testu wypadło uruchomienie bezpośrednio w systemie Linux, które zużywa o ponad 4 MB pamięci RAM więcej w porównaniu

do środowiska Docker oraz o ponad 5 MB więcej w porównaniu do platformy Podman. Różnica ta może być spowodowana, wykorzystaniem współdzielonych zasobów przez kontenery oraz lepszą optymalizacją współpracy z kernelem Linux. W przypadku tego testu najlepiej wypadło środowisko Podman.



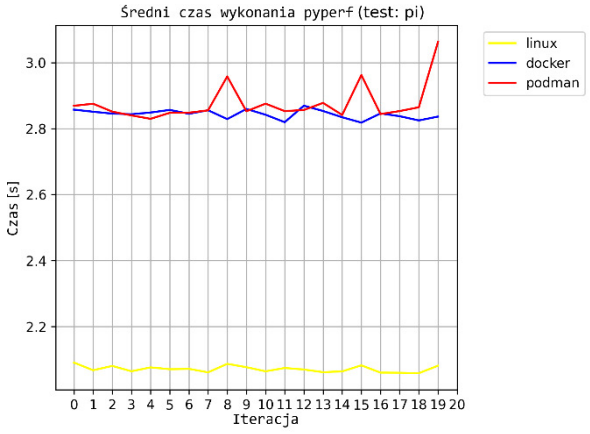
Rysunek 4: Wykres przedstawiający średni czas wykonania scenariusza S1.

Tabela 3: Wyniki dla czasu wykonania scenariusza S1

Platforma	Minimum (s)	Średnia (s)	Maksimum (s)	σ z próby
Linux	2,16	2,20	2,21	0,02
Docker	3,43	3,52	4,07	0,10
Podman	3,81	3,91	4,10	0,07

Analizując dane przedstawione na wykresie z Rysunku 4 oraz dane przedstawione w Tabeli 3 można zauważyć, że czas konieczny do wykonania scenariusza S1 jest znacząco dłuższy w przypadku rozwiązań wykorzystujących konteneryzację, niekiedy prawie dwukrotnie dłuższy niż czas realizacji tego samego zadania w systemie Linux. Najgorzej w tym teście wypada rozwiązanie Podman, może to być spowodowane tym, że jest to rozwiązanie niewykorzystujące stale działającego w tle demona, co pomimo zalet tego rozwiązania wymienionych w poprzednich rozdziałach, może wiązać się z dodatkowym czasem wymaganym do uruchomienia narzędzi konteneryzacji przy każdym starcie, zatrzymaniu
i modyfikacji kontenera.

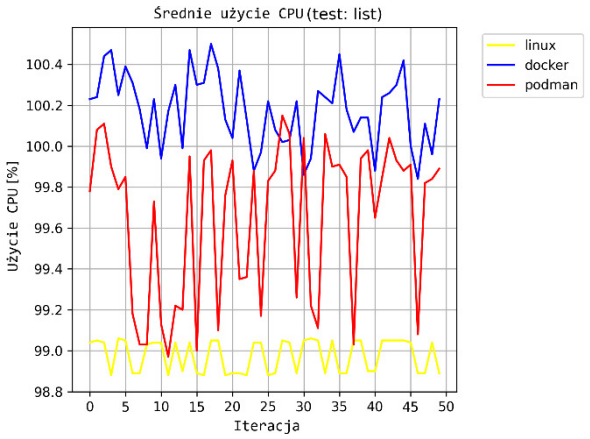
Wykres na Rysunku 5 przedstawia średni czas wykonania samych obliczeń liczby π, nie uwzględniając czasu potrzebnego na operacje związane z konteneryzacją. Pomiar czasu został wykonany przy pomocy modułu *pyperf* z języka Python. Po porównaniu wykresów z Rysunku 4 oraz Rysunku 5 można zauważyć, że czas potrzebny na zainicjowanie obliczeń w systemie Linux jest znacząco mniejszy niż czas potrzebny na uruchomienie oraz zatrzymanie kontenera zarówno w środowisku Docker, jak i Podman. W przybliżeniu czas potrzebny na operacje uruchomienia, zatrzymania i usunięcia kontenera w przypadku narzędzia Docker wynosi 0,68 sekundy, a przypadku narzędzia Podman 1,05 sekundy. Czas wykonywania samych obliczeń w przypadku rozwiązań Docker i Podman jest porównywalny.



Rysunek 5: Wykres przedstawiający średni czas wykonania obliczeń dla liczby π.

5.2. Wyniki dla scenariusza S2

W niniejszym podrozdziale przedstawione zostały wyniki badań testów zrealizowanych zgodnie ze scenariuszem S2.



Rysunek 6: Wykres przedstawiający średnie użycie procesora dla scenariusza S2.

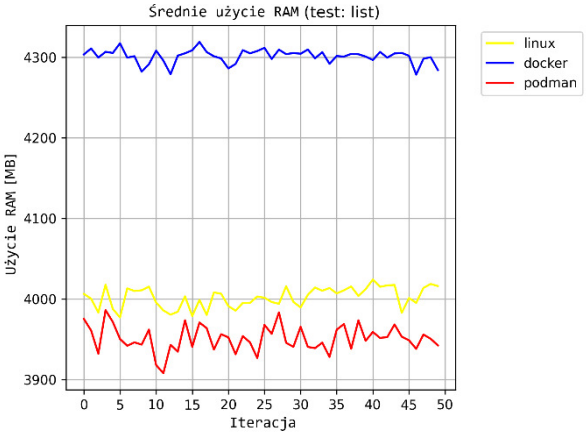
Na Rysunku 6 przedstawiony został wykres średniego obciążenia procesora dla obliczeń związanych z sortowaniem uporządkowanej listy.

Tabela 4: Wyniki zużycia CPU dla scenariusza S2

Platforma	Minimum (%)	Średnia (%)	Maksimum (%)	σ z próby
Linux	98,88	98,98	99,06	0,08
Docker	99,84	100,18	100,50	0,18
Podman	98,97	99,65	100,15	0,38

Analizując dane przedstawione na wykresie z Rysunku 6 oraz dane zgromadzone w Tabeli 4, można zauważyć podobieństwo do scenariusza S1. W tym przypadku również środowisko Docker okazało się tym, które bardziej obciąża procesor. Najbardziej zrównoważone obciążenie ponownie było generowane na platformie bez konteneryzacji wykorzystującej uruchomienie

bezpośrednio w systemie Linux. Największe wahania obciążenia dotyczą środowiska Podman.



Rysunek 7: Wykres przedstawiający średnie użycie pamięci RAM dla scenariusza S2.

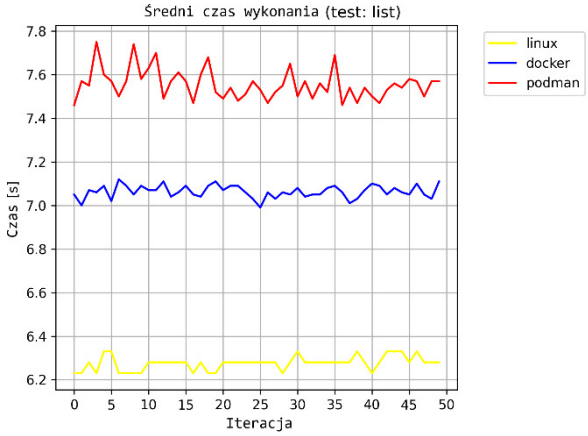
Tabela 5: Wyniki użycia pamięci RAM dla scenariusza S2

Platforma	Minimum (MB)	Średnia (MB)	Maksimum (MB)	σ z próby
Linux	3977,23	4002,14	4024,22	12,70
Docker	4278,47	4301,30	4318,95	8,61
Podman	3908,07	3951,17	3986,32	16,24

Dane przedstawione na wykresie z Rysunku 7 oraz dane zgromadzone w Tabeli 5 ukazują znaczne różnice w alokowaniu pamięci RAM przez narzędzia Docker i Podman. Środowisko Docker alokuje ponad 300 MB pamięci więcej w porównaniu do środowiska Podman przy wykonywaniu tego samego zadania. Prawdopodobnie ma to związek z problemem występującym w platformie Docker, polegającym na zbyt agresywnym alokowaniu zasobów przy stosunkowo małych zadaniach, zwłaszcza w warunkach małej liczby kontenerów i dostępności wolnych zasobów. Takie zachowanie zostało już zauważone przez autorów pracy [6] oraz [9].

Na podstawie danych przedstawionych na wykresie z Rysunku 8 oraz danych zgromadzonych w Tabeli 6 można zauważyć, że wszystkie 3 testowane platformy posiadają różne czasy wykonania dla tego samego zadania. Różnica między czasem wykonania dla środowiska Docker oraz środowiska Podman jest tutaj dużo wyraźniejsza niż w przypadku scenariusza S1 i wynosi w tym przypadku średnio prawie 0,5 sekundy, co stanowi wynik gorszy o około 6,5%. Sama powtarzalność czasu wykonania ma zauważalnie mniejsze wahania w przypadku kontenerów Docker, który oferuje porównywalną powtarzalność jak uruchomienie natywne w systemie Linux.

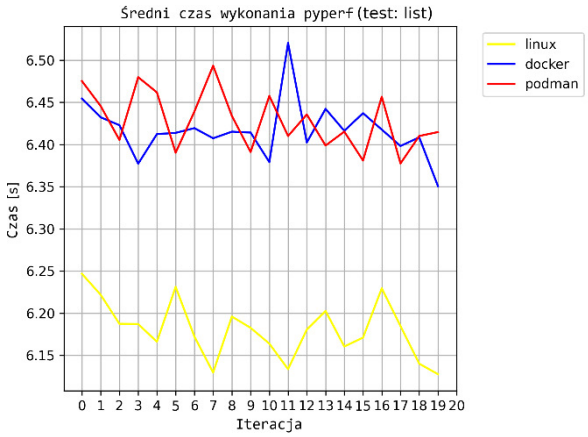
Po przeprowadzeniu badań czasu wykonania sortowania uporządkowanej listy za pomocą modułu *pyperf*, których wyniki zostały przedstawione na wykresie na Rysunku 9, można zauważyć, że sam czas trwania obliczeń dla kontenerów Docker i Podman nie różni się w sposób zauważalny oraz znaczący, jednak w porównaniu do uruchomienia natywnego można zaobserwować pewien spadek prędkości samych obliczeń, co świadczy o występowaniu pewnego rodzaju narzutu ze względu na konteneryzację.



Rysunek 8: Wykres przedstawiający średni czas wykonania scenariusza S2.

Tabela 6: Wyniki dla czasu wykonania scenariusza S2

Platforma	Minimum (s)	Średnia (s)	Maksimum (s)	σ z próby
Linux	6,23	6,28	6,33	0,03
Docker	7,00	7,06	7,11	0,03
Podman	7,46	7,55	7,75	0,07



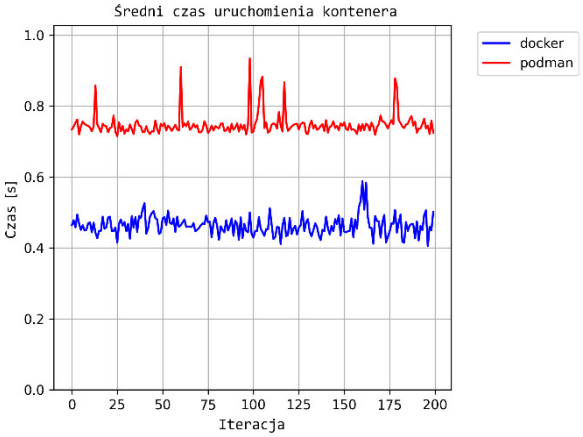
Rysunek 9: Wykres przedstawiający średni czas wykonania sortowania listy.

Narzut ten w przypadku czasu wykonywania obliczeń wewnątrz kontenera wynosi około 4% w porównaniu do uruchomienia natywnego w systemie Linux. Zauważa się także, że w przypadku kontenerów Podman występuje dodatkowy, wymierny narzut w postaci dodatkowego czasu potrzebnego na uruchomienie i zatrzymanie narzędzi konteneryzacji, które w przeciwieństwie do narzędzi ze środowiska Docker nie działają jako demon w tle.

5.3. Wyniki dla scenariusza S3

W niniejszym podrozdziale przedstawione zostały wyniki badań testów zrealizowanych zgodnie ze scenariuszem S3. Zestawiając ze sobą dane przedstawione na wykresie z Rysunku 10 oraz w Tabeli 7 z danymi zaprezentowanymi w scenariuszach S1 oraz S2 można zau-

ważyć, że narzędzie Podman posiada dodatkowy narzut do sumarycznego czasu wykonywania zadań w postaci czasu koniecznego na uruchomienie oraz zatrzymanie narzędzi konteneryzacji.



Rysunek 10: Wykres przedstawiający czas uruchomienia kontenera w scenariuszu S3.

Tabela 7: Wyniki dla czasu uruchomienia kontenera w scenariuszu S3

Platforma	Minimum (s)	Średnia (s)	Maksimum (s)	σ z populacji
Docker	0,405	0,463	0,588	0,026
Podman	0,715	0,747	0,934	0,031

Narzut ten wynosi od 0,3 sekundy do 0,7 sekundy w testowanych scenariuszach. W przypadku statycznych zadań, gdzie główną problematykę stanowi wykonywanie zadań w obrębie jednego kontenera, wpływ dodatkowego narzutu może być pominięty, ponieważ będzie stanowił niewielką część sumarycznego czasu działania kontenera, tak w przypadku dynamicznych środowisk, gdzie kontenery są uruchamiane i zatrzymywane z dużą częstotliwością, może stanowić przyczynę problemów z wydajnością i responsywnością, o ile nie zostanie w prawidłowy sposób rozwiązany.

6. Podsumowanie

Analizując wyniki zgromadzone w trakcie badań nad scenariuszami, można dostrzec pewne wzorce w działaniu obu porównywanych rozwiązań w dziedzinie konteneryzacji. Środowisko Podman w testowanych scenariuszach osiąga lepsze wyniki w kwestii ogólnie pojętego zarządzania zasobami. Użycie procesora oraz alokacja pamięci operacyjnej przez to rozwiązanie wypadają na ogół lepiej w porównaniu do narzędzia Docker. Różnica nie jest duża, ponieważ wynosi od 1% do maksymalnie 4% w przypadku wykorzystania CPU na korzyść środowiska Podman, jednak wynik ten jest powtarzalny i widoczny w wielu testach. Narzędzie Docker charakteryzuje się natomiast lepszą dynamiką działania, czas potrzebny na start i zatrzymanie kontenerów jest krótszy, co daje przewagę temu narzędziu od 0,3 sekundy do 0,7 sekundy według przeprowadzonych testów. Tutaj również różnica wyników jest niewielka, jednak może mieć znaczenie w pewnych zastosowaniach, wymagających wielokrotnego uruchamiania i zatrzymywania wielu kontenerów.

Pomimo występującego niewielkiego narzutu wydajności w zakresie 7 - 9%, wykorzystanie środowisk konteneryzacji, takich jak Docker i Podman, oferuje ogromne korzyści związane z wygodą, skalowalnością i zarządzaniem aplikacjami.

Warto również podkreślić, że różnice w wydajności na poziomie 7 - 9% są zazwyczaj akceptowalne w większości przypadków, zwłaszcza jeśli wziąć pod uwagę wszystkie korzyści związane z konteneryzacją. Ostatecznie, wybór między Dockerem a Podmanem może zależeć od indywidualnych potrzeb i preferencji, ale obie te platformy pozostają niezwykle przydatnymi narzędziami
w dziedzinie wdrażania i zarządzania aplikacjami.

Literatura

[1] S.Shah, N. Khandhar, Docker - The Future of Virtualization, International Journal of Research and Analytical Reviews (IJRAR) 6(2) (2019) 164 - 167.

[2] D.Walsh, Podman in Action, Manning Publications, (2023), ISBN: 9781633439689.

[3] M. Kjellstedt, Performance Evaluation of deploying microservices using Docker and Podman, thesis, Umeå University, (2020) 13 – 16.

[4] S. Giallorenzo, J. Mauro, M. G. Poulsen, F. Siroky, Virtualization Costs: Benchmarking Containers and Virtual Machines Against Bare-Metal, SN Computer Science 2(404) (2021) 11 – 15, https://doi.org/10.1007/s42979-021-00781-8.

[5] A. Subil, On the Use of Containers in High Performance Computing, 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), (2020) 284 - 293, https://doi.org/10.1109/CLOUD49709.2020.00048.

[6] E. Casalicchio, V. Percibali, Measuring Docker Performance: What a mess!!!, Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, (2017) 11 – 16, https://doi.org/10.1145/3053600.3053605.

[7] M. A. Potdar, G. D. Narayan, S. Kengond, M. M. Mulla, Performance Evaluation of Docker Container and Virtual Machine, Third International Conference on Computing and Network Communications (CoCoNet'19), (2019) 1419 – 1428, https://doi.org/10.1016/j.procs.2020.04.152.

[8] R. R. Yadav, G. T. E. Sousa, A. R. G. Callou, Performance Comparison Between Virtual Machines And Docker Containers, IEEE Latin America Transactions 16(8) (2018) 2282 – 2288, https://doi.org/10.1109/TLA.2018.8528247.

[9] C. MinSu, L. HwaMin, L. Kiyeol, A performance comparison of linux containers and virtual machines using Docker and KVM, Cluster Computing 22(1) (2019) 1765 – 1775, https://doi.org/10.1007/s10586-017-1511-2.

[10] B. B. Rad, J. H. Bhatti, M. Ahmadi, An Introduction to Docker and Analysis of its Performance, International Journal of Computer Science and Network Security (IJCSNS) 17(3) (2017) 228 – 234.

[11] Formuła Leibniza do obliczeń liczby π, https://en.wikipedia.org/wiki/Leibniz_formula_for_%CF%80, [06.09.2023].

[12] Sorting HOW TO - Python Documentation, https://docs.python.org/3/howto/sorting.html, [06.09.2023]