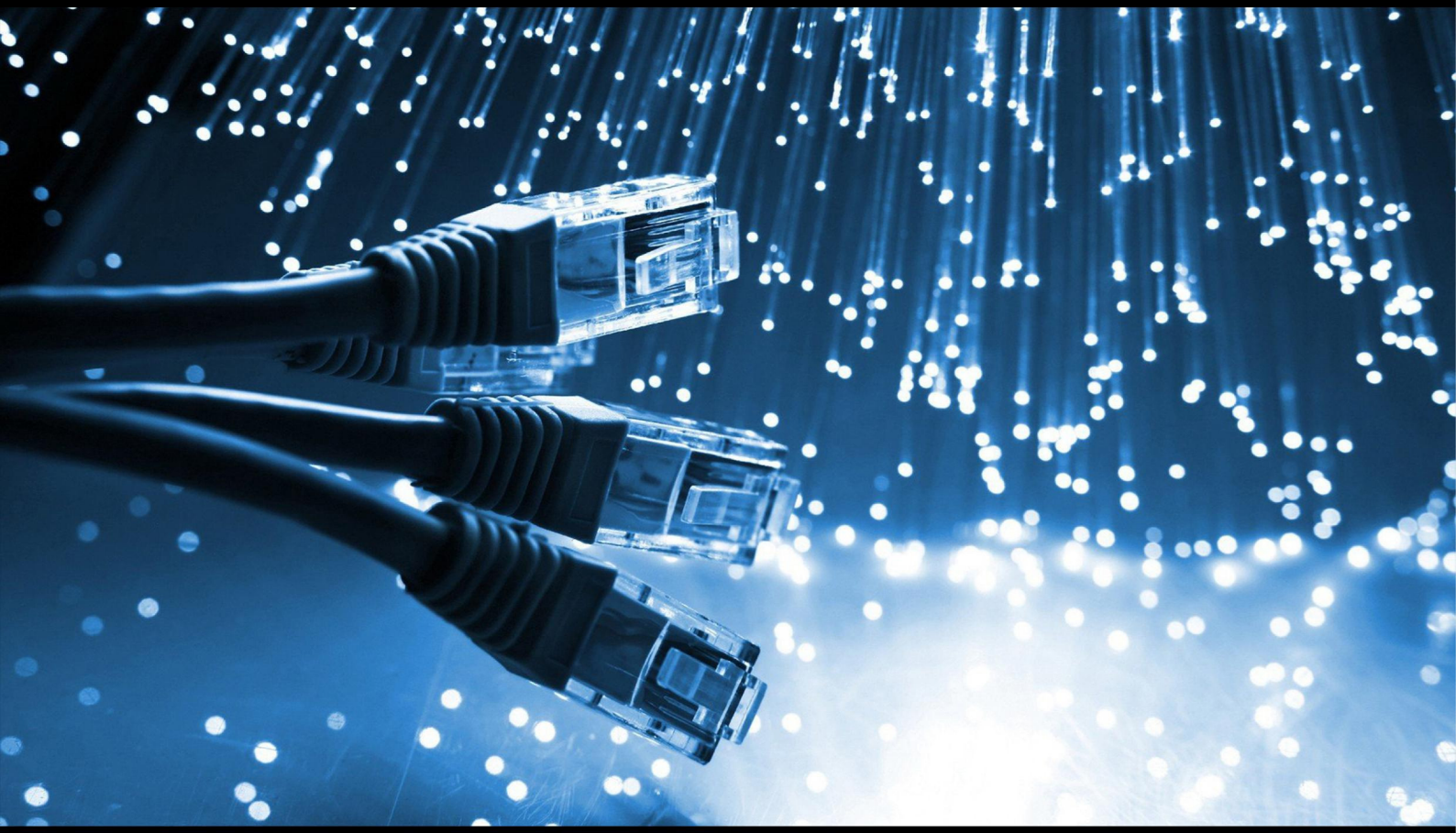
JCSI

Journal of Computer Sciences Institute

Volume 7/2018



Institute of Computer Science
Lublin University of Technology

jcsi.pollub.pl

ISSN: 2544-0764

Redakcja JCSI
e-mail: jcsi@pollub.pl
www: jcsi.pollub.pl
Instytut 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 Mariusz Dzieńkowski
dr inż. Tomasz Szymczyk
dr Edyta Łukasik
dr inż. Marek Miłosz
dr inż. Maria Skublewska-Paszkowska
dr inż. Dariusz Gutek
dr inż. Elżbieta Miłosz
dr inż. Jakub Smołka
dr inż. Maciej Pańczyk
dr Beata Pańczyk
dr hab. inż. Dariusz Czerwiński, prof. PL

Skład komputerowy:
Piotr Misztal
e-mail: p.misztal@pollub.pl

Projekt okładki:
Marta Zbańska

JCSI Editorial
e-mail: jcsi@pollub.pl
www: jcsi.pollub.pl
Institute 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:
Mariusz Dzieńkowski
Tomasz Szymczyk
Edyta Łukasik
Marek Miłosz
Maria Skublewska-Paszkowska
Dariusz Gutek
Elżbieta Miłosz
Jakub Smołka
Maciej Pańczyk
Beata Pańczyk
Dariusz Czerwiński

Computer typesetting:
Piotr Misztal
e-mail: p.misztal@pollub.pl

Cover design:
Marta Zbańska

ISSN 2544-0764

Spis treści

Contents

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 102-107
Wysłane: 2017-11-17
Przyjęte: 2017-11-21

Tworzenie aplikacji internetowych w technologii ASP.NET MVC i JavaSerwer Faces

Mariia Radutina*, Beata Pańczyk

Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono wyniki analizy porównawczej dwóch konkurencyjnych technologii tworzenia aplikacji internetowych: ASP.NET MVC firmy Microsoft oraz JavaServer Faces (JSF) wspieranej przez Oracle. Badania zostały zrealizowane poprzez implementację dwóch aplikacji o takiej samej funkcjonalności, korzystającej z tej samej bazy danych MySQL. Do pracy z danymi wykorzystano najczęściej stosowane narzędzia typu ORM: Hibernate dla JSF i Entity Framework dla ASP.NET MVC. Przy porównaniu brano pod uwagę strukturę aplikacji, łatwość implementacji, wsparcie środowiska programistycznego, wsparcie społecznościowe, komponenty interfejsu graficznego oraz efektywność pracy z bazą danych.

Słowa kluczowe: ASP.NET MVC; JavaServer Faces; aplikacje internetowe

*Autor do korespondencji.

Adres e-mail: mariaradutina@gmail.com

Web application development using ASP.NET MVC and JavaServer Faces

Mariia Radutina*, Beata Pańczyk

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

Abstract. The paper presents the results of comparative analysis of two competing web application technologies: ASP.NET MVC from Microsoft and JavaServer Faces (JSF) supported by Oracle. The research was done by implementing two applications with the same functionality using the same MySQL database. The most commonly used ORM tools are Hibernate for JSF and Entity Framework for ASP.NET MVC. The research was done by comparison the application structure, ease of implementation, support of the development environment, community support, graphical interface components, and database performance.

Keywords: ASP.NET MVC; JavaServer Faces; web application development

*Corresponding author.

E-mail address: mariaradutina@gmail.com

1. Wstęp

W warunkach szybkiego rozwoju technologii informacyjnych i szybkiej ewolucji systemów zarządzania informacją, najbardziej aktualne z zadań, które nieustannie występuję przed twórcami aplikacji, to wybór podejścia do tworzenia interfejsu. Podczas tworzenia aplikacji, deweloperzy często projektują interfejs, za pomocą specjalnych narzędzi do projektowania, przeciągając zestaw składników lub elementów sterowania na powierzchnię projektową [1].

Ze względu na olbrzymią konkurencję pomiędzy aplikacjami o podobnym zakresie funkcjonalności, użytkownik wybierze tę, która będzie bardziej intuicyjna i przyjazna a programista w prostszy sposób będzie w stanie zmienić interfejs oraz wspierać aplikację.

Do wyboru istniej dużo szkieletów programistycznych, bibliotek i narzędzi przeznaczonych dla różnych języków programowania. Ich nowe wersje pojawiają się na rynku nawet kilka razy w ciągu roku.

W niniejszym artykule do porównania wybrano dwie konkurencyjne technologie. JavaServer Faces (JSF) dla języka programowania Java oraz ASP.NET MVC dla języka programowania C#. Obie technologie mają na celu wsparcie procesu implementacji aplikacji webowych.

ASP.NET MVC, firmy Microsoft, łączy w sobie skuteczność i dokładność architektury model-widok-kontroler (MVC), najnowsze pomysły i wszystko, co najlepsze z istniejącej platformy ASP.NET [2].

JavaServer Faces (JSF) zmienił sposób pisania aplikacji internetowych w Javie. JSF proponuje eleganckie rozwiązania kluczowych problemów, związanych z opracowaniem komercyjnych zastosowań webowych [3]. JavaServer Faces opiera się również na architekturze MVC i jest standardem dla aplikacji Java EE. Technologię aktywnie rozwijają Oracle i IBM, a poza tym utworzono wiele bibliotek, które z nim współpracują. pozwalając wykorzystać niestandardowe komponenty UI, oparte na przykład na jQuery [4].

Obie technologie umożliwiają pracę z bazami danych za pomocą dedykowanych narzędzi ORM (ang. Object Relational Mapping) [5]. Dwa najbardziej popularne rozwiązania typu ORM to Hibernate dla języka Java i Entity Framework dla języka C#.

W artykule przedstawiono wyniki analizy porównawczej opartej na dwóch, identycznych co do funkcjonalności aplikacjach testowych współpracujących z tą samą bazą danych MySQL.

Przyjęto założenie, że przystępując do tworzenia aplikacji webowej, programista zna już podstawy programowania w języku C# i Java.

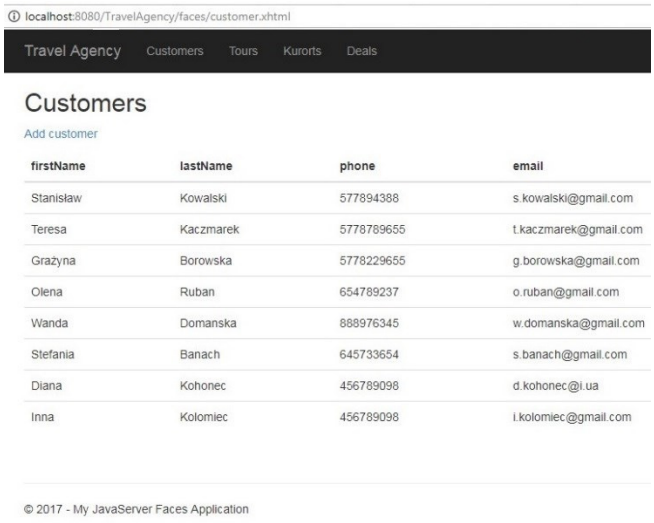
Postawiono następujące tezy:

T1: Obie technologie umożliwiają stworzenie aplikacji współpracującej z bazą danych MySQL o tej samej funkcjonalności.

T2: Technologia ASP.NET MVC jest bardziej przyjazna dla początkującego programisty aplikacji webowych w stosunku do technologii JSF.

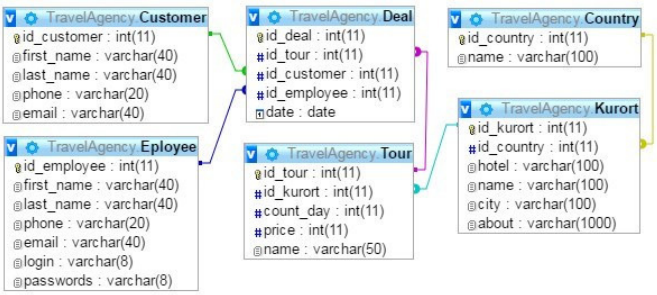
2. Aplikacje testowe

Prosta aplikacja testowa do obsługi biura podróży pozwala na realizację podstawowych funkcjonalności typu CRUD (ang. Create, Read, Update, Delete). Obie aplikacje posiadają jednakowy, prosty interfejs przedstawiony na rysunku 1. Schemat bazy danych przedstawia rysunek 2.



Rys. 1. Strona aplikacji z widokiem klientów biura podróży

Schemat bazy danych przedstawiono na Rys. 2.



Rys. 2. Schemat bazy danych dla aplikacji testowych

3. Ogólne cechy obu platform

Tabela 1 przedstawia podstawowe właściwości analizowanych technologii.

Z danych w tabeli 1 widać przewagę technologii ASP.NET MVC nad technologią JavaServer Faces, w kwestii związanej z rozwojem i ciągłym unowocześnianiem frameworka. Technologia ASP.NET jest na bieżąco aktualizowana. Niemalże każdego roku wychodzą jej nowe wersje, które mają nowe właściwości oraz możliwości. Najpopularniejsza wersja technologii JSF 2.2, z której korzysta największa liczba osób, została wydana w 2013 roku, a ostatnia wersja JSF 2.3 pojawiła się dopiero w roku 2017, po 4 letniej przerwie. Developerzy frameworka ASP.NET MVC dodają nowe funkcje i możliwości, zwiększając tym samym jego konkurencyjność.[6].

Tabela 1. Właściwości technologii ASP.NET MVC i JavaServer Faces

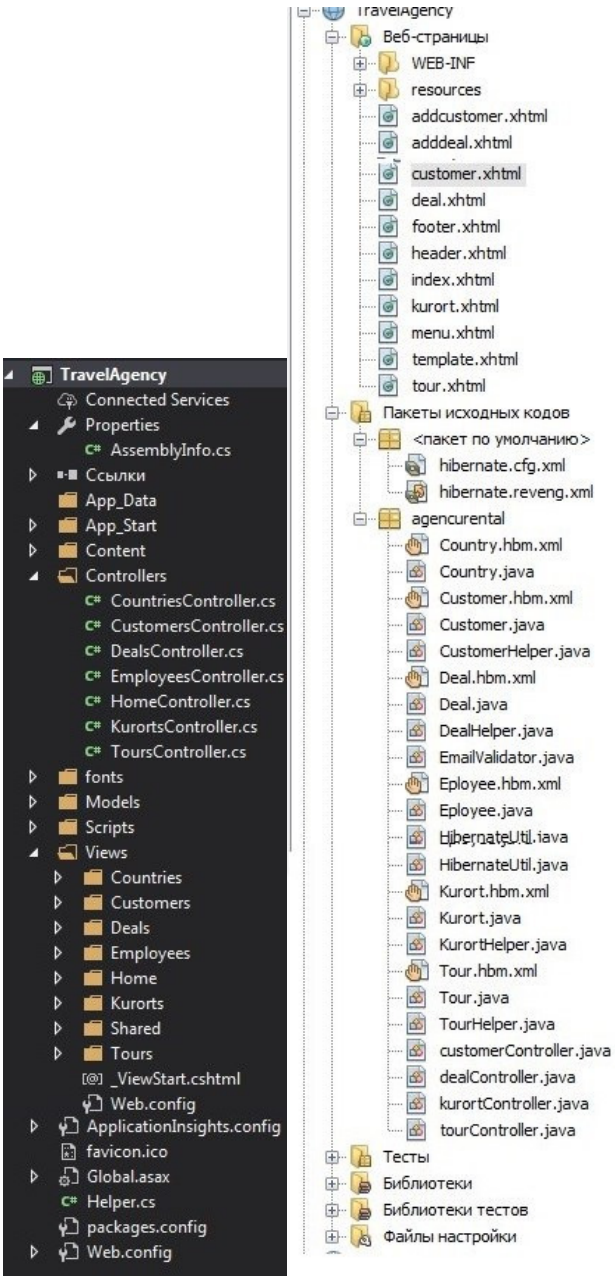
Właściwości	ASP.NET MVC 5	JavaServer Faces 2.2
Data przedstawienia	10 grudnia 2007	1 lipca 2009
Data ostatniej wersji	ASP.NET MVC 6 — 1.0.0 (2016)	JSF 2.3 (2017)
Twórca	Microsoft	Oracle Mojarra, Apache MyFaces
Język programowania	C#	Java
Open Source	Komercyjna	Tak
Środowisko programistyczne	Visual Studio	NetBeans, Eclipse, IntelliJ IDEA
Hosting	IIS	Nie posiada własnego hostingu
Technologia widoku	Razor	Facelet lub JSP
Stylizacja widoku	Wbudowany Bootstrap	Nie posiada wbudowanych narzędzi
Standard widoku	HTML5	XHTML 1.1
Generacja widoku wspomagana przez IDE	Jest możliwość generacji widoku z kontrolera dla jego wybranej metody	Nie posiada takiej możliwości
Tutoriale	Wiele oficjalnych informacji, wiele blogów, przykładów	Dokumentacja na stronie Oracle, mało informacji
Dedykowany serwer	IIS	Tomcat, GlassFish
Współpraca z MySQL i inną bazą danych	Jest możliwość współpracy	Jest możliwość współpracy
ORM wspierające pracę z bazą danych	Entity Framework, NHibernate, DataObjects.NET, ADO.NET Entity Framework	Hibernate, SpringORM
Instalacja bibliotek	Przez silnik NuGet	Dodawanie ręcznie pobranych bibliotek z oficjalnej strony
System operacyjny	Windows	Windows, Linux itp.

4. Struktura aplikacji

Aplikacje wytworzone w obu technologiach posiadają podobne struktury (Rys. 3), podzielone na trzy warstwy: model, widok, kontroler oraz zasoby uzupełniające w postaci plików CSS, skrypty JS.

W ASP.NET MVC w środowisku programistycznym Visual Studio automatycznie są tworzone foldery dla modeli, kontrolerów, widoków, plików stylu itp.

W JSF występuje obowiązek samodzielnego tworzenia odpowiedniej struktury katalogowej. Standardem jest podział na pliki Java oraz pliki warstwy web. Pliki w folderze Java zawierają klasy modeli i kontrolerów, a pliki w folderze web - widoki, pliki stylów CSS oraz skrypty JS.



Rys. 3. Struktura testowej aplikacji ASP.NET MVC i JSF

Obie technologie zawierają również pliki konfiguracyjne, ASP.NET MVC w folderze głównym a JavaServer Faces w folderze *web* oraz pliki konfiguracji Hibernate w pakietach Java i plik *Manifest* w folderze konfiguracji.

Przykład 1 przedstawia fragment kodu klasy modelu *Customer* reprezentującej klienta biura podróży dla JSF. Przykład 2 prezentuje analogiczny kod dla ASP.NET MVC.

Z przykładów 1-2 widać, że sposób definiowania podstawowych metod typu get/set (wykorzystywanych w klasach modeli) w C# sprowadzony jest do jednego wiersza, co generalnie wpływa na zmniejszenie liczby wierszy w klasach modeli.

Przykład 1. Fragment modelu Customer dla JSF

```
public String getPhone()
{
    return this.phone;
}
public void setPhone(String phone)
{
    this.phone = phone;
}
```

Przykład 2. Fragment modeluCustomer dla ASP.NET MVC

```
[Required]
[StringLength(20)]
[Display(Name = "Phone")]
public string phone { get; set; }
```

5. Interfejs graficzny

W aplikacji JSF intefejs graficzny został zbudowany za pomocą technologii facelet, natomiast ASP.NET MVC korzysta z silnika widoków Razor. Interfejs obu programów jest jednakowy (Rys. 1), drobna różnica występuje w nagłówku.

Przykład 3 przedstawia kod dodania bibliotek specjalnych, stylów i skryptów JavaScript dla JSF (13 linijek kodu). W celu podłączenia tych plików wykorzystywany jest specjalny znacznik biblioteki JSF, generujący nagłówek dokumentu HTML *<h:head>* oraz znacznik wskazujący źródło dołączanych zasobów *<h:outputStylesheet>*. W JSF występuje konieczność dodatkowego podłączenia bibliotek dla znaczników JSF a każdy plik stylów oraz skryptów dołączany jest oddzielnie.

Przykład 3. Nagłówek pliku widoku dla JSF

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head>
        <meta name="viewport" content="width=device-width,
              initial-scale=1"></meta>
        <h:outputStylesheet library="css" name="Site.css"/>
        <h:outputStylesheet library="css" name="bootstrap.css"/>
        <h:outputStylesheet library="js" name="modernizr-2.6.2.js"/>
        <title>Tour agency</title>
    </h:head>
```

Przykład 4 przedstawia sposób dołączenia stylów dla ASP.NET MVC za pomocą funkcji silnika Razor. W tym wypadku nagłówek zajmuje 9 linii kodu.

Przykład 4. Nagłówek pliku widoku dla ASP.NET MVC

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
          initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
```

W ASP.NET wszystkie style podłączane są za pomocą jednej linii kodu. Ten framework bazuje na silniku Razor i czystym kodzie HTML, nie trzeba korzystać z dodatkowych bibliotek znaczników. Razor oferuje natomiast tzw.

‘pomocników’ (ang. helper) do generowania kodu HTML z poziomu pliku widoku.

Przykład 5 i 6 przedstawiają odpowiednio dla JSF i ASP.NET MVC, fragmenty plików widoku kolumny tabeli z łączem do strony usunięcia elementu *Customer*.

Przykład 5. Fragment widoku pojedynczej kolumny z hiperłączem dla JSF

```
<h:column>
  <h:commandLink
      action="#{customerController.deleteCustomer
                (item.idCustomer)}"
                value="Delete"/>;
</h:column>
```

Przykład 6. Fragment widokupojedynczej kolumny z hiperłączem dla ASP.NET MVC

```
<td>
    @Html.ActionLink("Delete",
   "Delete", new { id=item.id_customer })
</td>
```

JSF do generowania widoków korzysta z technologii *facelet* i wykorzystuje specjalne znaczniki (np. *<h:column>* - komórka tabeli z zawartością, *<h:commandLink>* - hiperłącze z możliwością wywołania akcji *Delete*). ASP.NET MVC bazuje na technologii widoków Razor i korzysta z prostych znaczników tabeli HTML (np. *<td>*) a hiperłącza umożliwiające wywołanie akcji *Delete,* wstawiane jest również za pomocą specjalnych „helperów” Razora (np. *@Html.ActionLink*).

Porównując oba pliki widoków - ponownie widać, że zastosowanie silnika Razor skraca i upraszcza kod.

W opinii autorki, napisanie interfejsu w technologii ASP.NET MVC jest łatwiejsze. Środowisko programistyczne Visual Studio pozwala wygenerować widok do wybranej funkcji kontrolera, ale można też dowolnie zmienić stylizację lub wszystkie pliki tworzyć ręcznie. Udostepnienie gotowego szablon (opartego o Bootstrap), zdecydowanie ułatwia i przede wszystkim przyspiesza proces projektowania layoutu strony W przypadku JSF udostępniane są bardzo podstawowe szablony i cały layout strony należy projektować manualnie.

Jeśli chodzi o komponenty GUI to również ASP,NET MVC oferuje więcej kontrolek. Natomiast JSF wymaga tu wsparcia dodatkowych bibliotek np. Prime Faces [7], RichFaces [8].

6. Współpraca z bazą danych

Dla aplikacji JSF wymagane jest pobranie jednej biblioteki MySQL Connector Java z oficjalnej strony Oracla oraz dodanie jej do projektu.

Dla aplikacji ASP.NET MVC dodawanie bibliotek MySQL jest łatwiejsze, ale potrzebne są dwie biblioteki: *MySql.Data*, *MySql.Data.Entity* Wystarczy w środowisku programistycznym otworzyć pakiet NuGet oraz wybrać do zainstalowania te biblioteki.

Przykład 3 przedstawia konfigurację połączenia z baza danych przy pomocy Entity Framework dla ASP.NET MVC. W pliku *Web.config* w znaczniku *connectionString* umieszczono parametry połączenia z serwerem bazy danych: *nazwa użytkownika, nazwa serwera, parametry bazy danych.*

Przykład 3. Konfiguracja połączenia z bazą danych w Entity Framework

```
<connectionStrings>
  <add name="Master"
       connectionString="user id=root;
       password=;
       server=localhost;
       database=TravelAgency;
       Convert Zero
       Datetime=True;
       persistsecurityinfo=True"
providerName="MySql.Data.MySqlClient" />
</connectionStrings>
```

Przykład 4 przedstawia konfigurację połączenia z baza danych przy pomocy Hibernate. Plik *hibernate.cfg* jest tworzony automatycznie przy tworzeniu pustej aplikacji. W tagach *property* jest wskazana konfiguracja połączenia do bazy danych: typ dialektu, link do serwera, użytkownik oraz pliki mapowanych tabel z bazy danych.

Przykład 4. Konfiguracja połączenia z bazą danych przy pomocy Hibernate

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">
        org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">
        com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">
        jdbc:mysql://localhost:3306/TravelAgency?
        zeroDateTimeBehavior=convertToNull</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.current_session_context_class">
        thread</property>
    <mapping resource="agencurental/Eployee.hbm.xml"/>
    <mapping resource="agencurental/Country.hbm.xml"/>
    <mapping resource="agencurental/Deal.hbm.xml"/>
    <mapping resource="agencurental/Kurort.hbm.xml"/>
    <mapping resource="agencurental/Customer.hbm.xml"/>
    <mapping resource="agencurental/Tour.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

W kwestii podłączenia MySQL, można stwierdzić, że z jednej strony lepsza jest technologia JSF (wymagana jest dodatkowo jedna biblioteka, która zajmuje 2 razy mniej pamięci). A z drugiej strony w technologii ASP.NET MVC łatwiejszy jest proces instalacji bibliotek, które jednak zajmują więcej pamięci.

W obu aplikacjach przez narzędzia ORM (Hibernate dla JSF, Entity Framework dla ASP.NET MVC), automatycznie mapowane są tabele bazy danych na odpowiednie klasy modeli.

W obu technologiach mapowanie jest niemal jednakowe. Hibernate tworzy dodatkowe dwa pliki konfiguracyjne. Po mapowaniu tworzą się klasy z polami tabel bazy danych, z funkcjami dostępowymi get/set. Oba frameworki automatycznie generują odpowiednie kody w plikach konfiguracyjnych, ustawiając parametry połączenia do bazy danych.

Przykład 5 przedstawia metody tworzenia listy danych z tabeli *Deals* (korzystając z modelu Deals i języka zapytań LINQ) przy pomocy Entity Framework.

Przykład 5. Formowanie listy Deals przy pomocy Entity Framework

```
var deals = db.Deals.Include(d => d.customer)
    .Include(d => d.eployee).Include(d => d.tour);
return View(deals.ToList());
```

Przykład 6 przedstawia metody tworzenia listy danych z tabeli *Deals* przy pomocy Hibernate.

Przykład 6. Tworzenie listy Deals przy pomocy Hibernate

```
List<Deal> dealList = null;
org.hibernate.Transaction tx = session.beginTransaction();
Query q = session.createQuery ("from Deal ");
dealList = (List<Deal>) q.list();
```

Tworzenie listy umów z klientami w ASP.NET MVC bazuje na odpowiednim zapytaniu LINQ wykonanym za pomocą jednej instrukcji. W JSF najpierw należy zadeklarować listę umów, następnie, uruchomić transakcję Hibernate i dopiero zapytanie do baz danych

Przykład 7 przedstawia kod dodawania nowego klienta do tabeli *Customer* w ASP.NET MVC za pomocą Entity Framework. Przykład 8 prezentuje analogiczny kod dla Hibernate i JSF.

Przykład 7. Dodawanie nowego klienta w ASP.NET MVC

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include =
    "id_customer,first_name," +
    "last_name,phone,email")] Customer customer)
{
    if (ModelState.IsValid)
    {
        db.Customers.Add(customer);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    return View(customer);
}
```

Przykład 8. Dodawanie nowego klienta w technologii JSF

```
public void addCustomer() throws IOException
{
        Customer customer = new Customer();
         customer.setFirstName(first_name);
         customer.setLastName(last_name);
         customer.setEmail(email);
         customer.setPhone(phone);
         CustomerHelper customersClass = new CustomerHelper();
         customersClass.insertCustomer(customer);
         HttpServletResponse response =
         (HttpServletResponse) FacesContext.getCurrentInstance().
                getExternalContext().getResponse();
         response.sendRedirect("/TravelAgency/faces/customer.xhtml");
}
```

Dodawanie nowego obiektu w ASP.NET MVC zajmuje mniej linii kodu. W JSF należy uzupełnić pola obiektu, korzystając z funkcji *set.* W ASP.NET MVC jest to realizowane automatycznie za pomocą mechanizmu Data Model Binding i Entity Framework - dlatego wystarczy jedna instrukcja do pobrania danych klienta z formularza, utworzenie nowego obiektu *Customer* i dodania go do bazy danych.

Przykład 9 przedstawia metody usuwanie klienta z tabeli *Customer* w ASP.NET MVC przy pomocy Entity Framework.

Przykład 9. Usuwanie klienta w ASP.NET MVC za pomocą Entity Framework

```
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(int id)
{
    Customer customer = db.Customers.Find(id);
    db.Customers.Remove(customer);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

Przykład 10 przedstawia metody usuwanie klienta z tabeli *Customer* w JSF przy pomocy frameworka Hibernate.

Przykład 10. Usuwanie klienta w JSF za pomocą Hibernate

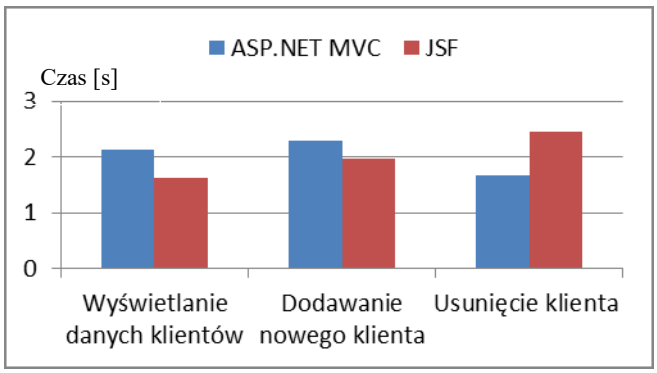
```
public void deleteCustomer(int id)
{
    SessionFactory sessionFactory;
    sessionFactory =
            new AnnotationConfiguration().
                    configure().buildSessionFactory();
    Session session = sessionFactory.openSession();
    session.beginTransaction();
    Query q = session.createQuery("from Customer where "
            + "id_customer = :id_customer ");
    q.setParameter("id_customer", id);
    Customer stock = (Customer)q.list().get(0);
    session.delete(stock);
    session.getTransaction().commit();
    session.close();
}
```

Usuwanie obiektów w obu technologiach jest proste i w zasadzie podobnie kodowane.

Ważnym aspektem pracy z bazą danych jest czas wykonywania operacji bazodanowych. Tabela 2 i rysunek 4 przedstawiają czas wykonywania podstawowych operacji na tabeli *Customer* w obu technologiach.

Tabela 2. Czas wykonywania operacji na bazie danych

	ASP.NET MVC	JSF
Pierwsze uruchamianie (sekundy)	34,15	74,27
Wyświetlanie danych klienta (sekundy)	2,13	1,63
Dodawanie nowegoklienta (sekundy)	2,3	1,98
Usuwanie klienta (sekundy)	1,67	2,46



Rys. 4. Wydajność operacji bazodanowych [s]

7. Metryki kodu aplikacji

Ostatnim analizowanym kryterium porównania obu aplikacji były podstawowe metryki kodu, które zostały zestawione w tabeli 3.

Aplikacja w technologii ASP.NET MVC zajmuje do 5 razy więcej pamięci, i wykorzystuje więcej bibliotek.

Porównując fizyczne linie kodu, można stwierdzić, że aplikacja JavaServer Faces zajmuje więcej linii kodu. Tylko dla widoku w obu technologiach liczba linii kodu jest jednakowa.

Dla pliku modelu JavaServer Faces zajmuje półtora razy więcej linii kodu. Wynika to z faktu, że w ASP.NET MVC metody get oraz set są pisane zwykle w jednej linii.

Tabela 3. Wybrane metryki kodu obu aplikacji

	JavaServer Faces	**ASP.NET MVC**
Liczba plików aplikacji	164	384
Pliki konfiguracyjne	5	5
Liczba bibliotek	18	26
Waga bibliotek(MB)	11.5	68.7
Kontroler - liczba linii kodu	180	115
Widok - liczba linii kodu	50	50
Model - liczba linii kodu	69	43
Waga projektu (MB)	22.3	103

8. Wnioski

W artykule przedstawiono najważniejsze aspekty analizy porównawczej technologii ASP.NET MVC oraz JavaSerwer Faces. Stworzone aplikacje testowe potwierdziły tezę 1, że obie technologie umożliwiają stworzenie aplikacji współpracującej z bazą danych MySQL o tej samej funkcjonalności.

Na podstawie doświadczenia nabytego w trakcie tworzenia aplikacji w obu środowiskach i przedstawionych wyników, można wnioskować, że to ASP.NET MVC jest dla początkującego programisty łatwiejszy i bardziej przyjazny. Oferuje możliwość automatycznego generowania layoutu, tworzenia widoków do wybranego modelu danych, czy też do wskazanych akcji kontrolerów. Dodatkowo istnieje wiele materiałów pomocniczych udostępnianych na stronach Microsoft, blogach i forach dyskusyjnych. Nie wymaga też instalowania dodatkowych bibliotek z komponentami GUI.

Technologia JavaServer Faces jest silnym konkurentem dla ASP.NET MVC, jednak oferuje zdecydowanie mniejsze wsparcie, jeśli chodzi o generowanie kodu dla aplikacji webowej. Jej niezaprzeczalną zaletą jest dużo mniejsza waga gotowej aplikacji.

Literatura

[1] Chris Sutton Programming ASP.NET MVC; Москва, 2008.

[2] http://smarly.net/pro-asp-net-mvc-4/introducing-asp-net-mvc-4/what-is-the-big-idea/key-benefits-of-asp-net-mvc [03.11.2017]

[3] https://professorweb.ru [12.08.2017]

[4] Kito D. Mann: JavaServer Faces in Action, Manning Publications Company, 2005.

[5] Дэвид Гири, Кей С. Хорстманн. Core JavaServer Faces. Вильямс, 2011.

[6] David Geary, Cay Horstmann: Core JavaServer Faces, Second Edition, Prentice-Hall, 2007.

[7] http://showcase.richfaces.org/, [18.11.2017]

[8] https://www.primefaces.org/showcase/, [18.11.2017]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 108-115
Wysłane: 2017-11-19
Przyjęte: 2017-11-24

Analiza wydajnościowa i możliwościowa narzędzia Laravel do tworzenia nowoczesnych aplikacji webowych

Przemysław Mincewicz*, Małgorzata Plechawska-Wójcik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Tematyką badaną w artykule jest sprawdzenie możliwości narzędzia do tworzenia aplikacji webowych Laravel. Przedstawiono przegląd literatury traktującej o Laravel, jego najpopularniejsze moduły oraz dodatkowe możliwości. Na podstawie stworzonej aplikacji testowej napisanej przy pomocy trzech narzędzi – Laravel, Symfony oraz Zend porównano je ze sobą pod względem wydajności oraz możliwości. Aplikacją testową była baza albumów muzycznych o trzech zestawach danych – z jednym, dwoma oraz pięcioma tysiącami albumów. Badanie wydajności narzędzia Laravel na tle Zend Framework i Symfony sporządzono na podstawie między innymi czasu ładowania danych z bazy oraz ich wyszukiwania w kolekcji danych. Podsumowanie informacji zawartych w artykule stwierdza, że Laravel do narzędzie nastawione na komfort pisania aplikacji, odstające jednak na tle innych narzędzi wydajnością.

Słowa kluczowe: framework Laravel; język PHP; MVC; tworzenie aplikacji w narzędziu Laravel; porównanie narzędzi języka PHP

*Autor do korespondencji.

Adres e-mail: p.mincewicz@gmail.com

Performance and possibility analysis of Laravel tool dedicated to create modern web applications

Przemysław Mincewicz*, Małgorzata Plechawska-Wójcik
[a] Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The aim of this paper is verification of possibilities of Laravel PHP framework. An overview of literature is presented, which shows Laravel as a research tool in various scientific fields and compares it with other PHP language tools. The project of a test application was presented, which is a database of music albums with three datasets - one, two and five thousand albums. Laravel's Zend Framework and Symfony performance survey was based on, among other things, database loading time, database searches, and the time to generate a single album view with relationships. Summary of the information contained in the paper states that Laravel is an easy-to-use tool, with a number of app creation aids, but with performance outweighed by the frameworks compared to it, the most efficient of which is Zend Framework.

Keywords: Laravel framework; PHP programming language; MVC; Laravel application development; PHP frameworks comparison

*Corresponding author.

E-mail address: p.mincewicz@gmail.com

1. Wprowadzenie

Aplikacje internetowe są w dzisiejszych czasach postrzegane w taki sam sposób jak aplikacje desktopowe. Wymaga się od nich jednakowych funkcjonalności, przystępności dla użytkownika, aktualizacji rozwojowych oraz ciągłego wsparcia technicznego. Zaletą wyróżniającą aplikacje webowe jest dostępność oraz uniwersalność – do uruchomienia ich wystarczy jedynie przeglądarka internetowa.

Dla deweloperów obecny rynek narzędzi do tworzenia internetowych aplikacji jest niezwykle bogaty. Wiele języków programowania pozwala na tworzenie w pełni funcjonalnych i rozbudowanych aplikacji, a każdy z nich zawiera co najmniej kilka narzędzi wspomagających ich pisanie. W artykule zwrócono szczególną uwagę na nowe i dynamicznie rozwijające się narzędzie Laravel, które oparte jest w głównym stopniu o język PHP. Bardzo szybko zyskał on wielkie grono zwolenników i stał się jednym z najpopularniejszych narzędzi do tworzenia nowoczesnych aplikacji internetowych.

1.1. Charakterystyka narzędzia Laravel

Wraz z rozwojem internetu oraz powstawaniem nowych urządzeń, które mogły z niego korzystać, w procesie tworzenia aplikacji internetowych większą wagę przykładano do skalowalności oraz większej uniwersalności programowania wspomagającej multiplatformowość. Język PHP, będącym początkowo jedynie narzędziem tworzenia spersonalizowanych stron internetowych [8], wkomponował się idealnie w zaistniałą sytuację i stał się jednym z najważniejszych języków programowania nowoczesnych aplikacji internetowych.

Wraz z rozwojem narzędzi do tworzenia aplikacji w języku PHP, takich jak CodeIgniter, CakePHP, Symfony czy Laravel zmieniał się sposób implementacji odpowiednich warstw w kodzie poprzez zróżnicowane technologie obsługi baz danych czy tworzenia widoków dla użytkownika. Dużą uwagę zwrócono na bezpieczeństwo użytkownika. W okresie od połowy 2006 do 2010 roku aplikacje PHP zwiększyły swoje bezpieczeństwo o blisko 9% [1] poprzez zmiany wyłącznie

w strukturze tworzenia aplikacji i zmianom w CORE (podstawie) języka.

1.2. Przegląd literatury

Laravel jako narzędzie do tworzenia aplikacji internetowych w bardzo krótkim czasie zyskał wielu sympatyków. Złożyło się na to wiele czynników, między innymi prosta implementacja, bogata społeczność deweloperska oraz ciągłe wsparcie najnowszych rozwiązań języka PHP. Badania przeprowadzone przez portal sitepoint ukazują, że Laravel w ciągu 4 lat stał się najpopularniejszym narzędziem języka PHP, dwukrotnie częściej wykorzystywanym niż CodeIgniter i Symfony.

W niewielkim okresie czasu od powstania Laravel był porównywany z najpopularniejszymi narzędziami tworzenia aplikacji modelem MVC (ang. Model-View-Controller), takich jak Symfony, Yii czy CakePHP. W badaniach przeprowadzonych przez pana Adriana Zurkiewicza oraz Marka Miłosza, którzy stworzyli w pełni funkcjonalną platformę edukacyjną w wyżej wymienionych narzędziach, nie okazuje się on jednak narzędziem idealnym [2]. Wśród aspektów branych pod uwagę podczas analizy, z których wyróżnić należy obsługę baz danych, techniki programowania skupiające się na najbardziej efektywnym pisaniu kodu oraz realnej pracy programisty, Laravel okazał się liderem jedynie we wsparciu dla technologii. Ukazało to jego otwartość i wiele możliwości rozwiązywania problemów, jednak odsłoniło braki, z których szczególnie poważnym jest bardzo mała efektywność działania. Zestawienie wszystkich analiz wykazało, że Laravel nie jest najlepszym rozwiązaniem dla projektów rozbudowanych aplikacji MVC umożliwiających wielopoziomową komunikację między jej użytkownikam.

Ze względu na wysoką dostępność Laravel jest bardzo popularnym narzędziem w aplikacjach badawczych. Jedną z nich jest system sterowania domowymi urządzeniami, monitorujący jednocześnie zużycie energii [3]. Głównym jego założeniem jest obniżenie całkowitego zużycia energii urządzeń poprzez dynamiczne i automatyczne wyłączanie niewykorzystywanych. Dzięki takiemu projektowi użytkownik ma możliwość własnoręcznego zarządzania wszystkimi urządzeniami zsynchronizowanymi z systemem, wglądu w raporty odczytów oraz dostosowanie opcji wspierających oszczędzanie energii do własnych potrzeb. Po przeprowadzeniu badań i analizie raportów zmniejszono zużycie energii ze wszystkich branych pod uwagę urządzeń o blisko 59%.

Inną bardzo ciekawą aplikacją badawczą jest InverPep [9], traktującą o tematyce biologicznej. Stworzona przez Universidad Nacional de Colombia jest bazą peptydów antybakteryjnych (AMP), czyli podstawowych mechanizmów obronnych wytwarzanych przez rośliny i zwierzęta [4]. Narzędzie Laravel, poza utworzeniem bazy oraz strony wizualnej, posiada zaimplementowany algorytm obliczania właściwości fizykochemicznych wielu peptydów jednocześnie - CALCAMPI [5]. Każdy AMP jest opisany źródłem, właściwościami fizykochemicznymi, strukturą oraz aktywnością biologiczną.

Poza aplikacjami typowo badawczymi, gdzie narzędzie w gruncie rzeczy pełni drugorzędną rolę, Laravel wykorzystywany jest w aplikacjach testujących działanie innowacyjnych rozwiązań algorytmicznych oraz programistycznych. Przykładem takiej aplikacji jest zaproponowane przez He Ren Yu wdrożenie narzędzia Laravel bez wykorzystywania tradycyjnego wzorca MVC - programista skupia się jedynie na warstwie logiki [6]. Ma to na celu ujednolicenie procesu rozwoju aplikacji, lepszej implementacji założeń klienta oraz łatwiejszym zarządzaniem danymi, które wprowadzanie są do projektu poprzez plik XML. Takie rozwiązanie tworzenia aplikacji za pomocą Laravel zestawiono z tradycyjnym MVC z wykorzystaniem narzędzia CodeIgniter. Wynikiem okazało się dużo efektywniejsze tworzenie aplikacji za pomocą Laravel, co może świadczyć jednocześnie o łatwiejszym programowaniem w tym narzędziu jak i skuteczności prezentowanego przez autora rozwiązania.

Laravel nie jest narzędziem idealnym dla wszystkich aplikacji badawczych. Przykładem bazującej na wzorcu MVC jest system monitorujący i badający wpływ czynników naturalnych na wydajność technologii informacyjnych [7]. Głównym celem jej działania jest badanie parametrów środowiskowych takich jak temperatura i wilgotność powietrza pobranych przez przez sieć Sentinel i generowanie na ich podstawie szczegółowych raportów. Laravel jako narzędzie nie posiada bogatego wsparcia dla europejskiej sieci naukowej, brak w nim wielu przydatnych funkcji, między innymi uwierzytelniania z siecią. Przeprowadzone badania aplikacji na wielu przeglądarkach ukazało, że Laravel w jednakowy sposób jest przez każdą z nich interpretowany. Nie zmienia to faktu, że tak rozbudowana i niezwykle istotna technologia jaką jest sieć Sentinel nie dostała jeszcze wsparcia ze strony narzędzia Laravel.

1.3. Przedmiot i cel badań

Badania przeprowadzono w celu sprawdzenia wydajności i możliwości aplikacji napisanych w narzędziu Laravel. Jako punkty odniesienia posłużyły dwa popularne frameworki PHP – Symfony, czyli jeden z najpopularniejszych oraz Zend – w wielu badaniach uznawany za najwydajniejsze narzędzie języka PHP. Dadzą one pogląd na to co odpowiada za popularność Laravel - wydajność czy możliwości.

Przedmiotem badań jest jest aplikacja testowa. Badania zostaną przeprowadzone jako porównanie jej wersji w trzech narzędziach pod względem:

- wydajności:
  - czas dostępu i odczyt danych z serwera dla różnych zbiorów danych – wykorzystane zostaną 3 bazy, odpowiednio z 1 tys. albumów, 2 tys. albumów oraz 5 tys.
  - czas generowania strony przez narzędzie,
  - obsługa plików graficznych, czy występuje cachowanie,
- możliwości, czyli wykorzystanie wbudowanych w narzędzie mechanizmów tworzenia formularzy, widoków list obsługi bazy danych.
  - czas dostępu i odczyt danych z serwera dla różnych zbiorów danych – wykorzystane zostaną 3 bazy, odpowiednio z 1 tys. albumów, 2 tys. albumów oraz 5 tys.
  - czas generowania strony przez narzędzie,
  - obsługa plików graficznych, czy występuje cachowanie,
  - obsługa kolekcji obiektów pobieranych z bazy danych,

Aplikacje różnią się od siebie jedynie narzędziem, w którym zostały stworzone – bazy danych oraz ogólne funkcjonalności są takie same. Wyniki badań pozwolą przedstawić, które z wykorzystanych w pracy narzędzi jest najbardziej efektywne w jednakowych warunkach działania oraz poziom złożoności tworzenia jednakowej aplikacji.

2. Projekt aplikacji testowej

W celu analizy wydajności oraz możliwości narzędzia Laravel na tle innych narzędzi języka PHP stworzone zostaną trzy jednakowe aplikacje internetowe – każda z wykorzystaniem innego narzędzia – Laravel 5, Symfony 3 oraz Zend 3. Oparte będą o wzorzec MVC oraz indywidualne dla każdego z nich mechanizmy obsługi baz danych MySQL. Tematem aplikacji będzie zbiór informacji o albumach muzycznych, ich wykonawcach, wydawcach oraz zawartych w nich utworach.

Aplikacja będzie posiadać podstawowe funkcjonalności aplikacji internetowej wzbogacone o dodatkowe możliwości, takie jak:

- dodawanie albumów, wykonawców i wydawców,
- dodawanie okładek do albumów,
- powiązanie albumu z wykonawcą i wydawcą,
- dodawanie utworów do albumów,
- edycję albumów, wykonawców, wydawców i utworów,
- usuwanie albumów, wykonawców, wydawców i utworów,
- wyświetlanie listy albumów, wykonawców i wydawców,
- wyświetlanie szczegółowych informacji o albumach, w tym listę utworów, wykonawcach i wydawcach,
- filtrowanie wybranych pól na liście albumów, wykonawców oraz wydawców ,
- wyszukiwanie pozycji na liście albumów, wykonawców i wydawców w wybranym kryterium wpisanym w pole tekstowe,
- rejestracja nowego użytkownika i system logowania do aplikacji.

Aplikacja zapewnia także bezpieczeństwo użytkowania, poprzez obsługę sesji i ciasteczek, jest responsywna (przystosowana do wielu typów urządzeń) Umożliwia użytkownikowi filtrowanie oraz wyszukiwanie informacji o albumach muzycznych, ich wykonawcach oraz wydawcach, wymusza od użytkownika zalogowanie do aplikacji – w przypadku braku danych do logowania wymagana jest rejestracja.

Dodawanie nowego albumu muzycznego wymaga wcześniej definicji artystów i wydawców – relacje między tymi elementami są wymagane i widoczne w szczegółach płyty. Utwory z kolei dodawane są wyłącznie z poziomu szczegółów albumu – nie posiadają tym samym własnego, wyodrębnionego widoku, stale przypisane do konkretnej kompozycji.

2.1. Kryteria analizy aplikacji

Badania podzielone zostaną na dwa etapy:

1) Analizę możliwości narzędzi, które przeprowadzono podczas tworzenia aplikacji. W skład badania wchodzą:
   a. mechanizm tworzenia kontrolerów,
   b. mechanizm tworzenia widoków,
   c. mechanizm tworzenia formularzy,
   d. mechanizm obsługi bazy danych,
   e. mechanizm obsługi kolekcji obiektów (ORM),
   f. mechanizm migracji bazy danych,
   g. łatwość budowy aplikacji - liczbę linii kodu, jakie należy przygotować do uzyskania określonego efektu.
2) Analizę ogólnej wydajności działania narzędzi, które przeprowadzono na gotowych aplikacjach. W skład badania wchodzą:
   a. czas dostępu do bazy danych z 1 tys. albumów,
   b. czas odczytu z bazy danych z 1 tys. albumów,
   c. czas wyszukiwania pojedynczego albumu w bazie danych z 1 tys. albumów,
   d. czas wyszukiwania odpowiednich elementów w pobranej kolekcji danych w bazie z 1 tysiącem albumów,
   e. czas filtrowania albumów w bazie danych z 1 tys. albumów,
   f. czas filtrowania elementów w pobranej kolekcji danych w bazie z 1 tys. albumów,
   g. czas generowania kolekcji danych złożonych z wielu encji w bazie z 1 tys. albumów,
   h. czas dostępu do bazy danych z 2 tys. albumów,
   i. czas odczytu z bazy danych z 2 tys. albumów,
   j. czas wyszukiwania pojedynczego albumu w bazie danych z 2 tys. albumów,
   k. czas wyszukiwania elementów w pobranej kolekcji danych w bazie z 2 tysiącami albumów,
   l. czas filtrowania albumów w bazie danych z 2 tys. albumów,
   m. czas filtrowania odpowiednich elementów w pobranej kolekcji danych w bazie z 2 tys. albumów,
   n. czas generowania kolekcji danych złożonych z wielu encji w bazie z 2 tys. albumów,
   o. czas dostępu do bazy danych z 5 tys. albumów,
   p. czas odczytu z bazy danych z 5 tys. albumów,
   q. czas wyszukiwania pojedynczego albumu w bazie danych z 5 tys. albumów,
   r. czas wyszukiwania pojedynczego albumu w bazie danych z 5 tys. albumów,
   s. czas filtrowania albumów w bazie danych z 5 tys. albumów,
   t. czas filtrowania elementów w pobranej kolekcji danych w bazie z 5 tys. albumów,
   u. czas generowania kolekcji danych złożonych z wielu encji w bazie z 5 tys. albumów,
   v. czas generowania strony,
   w. czas ładowania plików graficznych.

3. Najpopularniejsze biblioteki

Architektura narzędzia Laravel została oparta o moduły. Początkowo spotkało się to z dozą krytyki, jednak w ogromnym stopniu przyczyniło do powstania ogromnej społeczności deweloperów wokół środowiska. Mają oni w łatwy sposób możliwość dzielenia się swoimi rozwiązaniami wielu problemów związanych z programowaniem, dzięki czemu każdy może wykorzystać przygotowany przez nich dodatek. Instalacja takiego dodatku przebiega poprzez narzędzie Composer.

W dalszej części artykułu przedstawiono cztery najczęściej wykorzystywane dodatki w aplikacjach Laravel.

3.1. Laravel Collectives – Forms & HTML

Najpopularniejszym i najczęściej wykorzystywanym dodatkiem narzędzia Laravel jest Form & HTML. Automatyzuje i znacznie ułatwia on tworzenie widoków w aplikacji oraz formularzy, które są nieodzowną jej częścią - każde wysłanie żądania dla zapisu czy odczytu danych jest przeprowadzane poprzez formularz. Dodatek ten wydaje się idealnym dla każdego dewelopera - początkującego, nie chcącego tracić czasu bądź nie potrafiącego tworzyć rozbudowanych formularzy, czy też doświadczonego, który na bazie dodatku zbuduje własne, bardziej spersonalizowane formularze [10].

3.2. Laravel Debugbar

Dodatek Laravel Debugbar służy do integracji bardzo przydatnego deweloperom PHP narzędzia do debugowania - PHP Debug Bar z aplikacją Laravel. W dużym stopniu ułatwia on implementację narzędzia w aplikacji, która sprowadza się do standardowej instalacji dodatku, bez konieczności dodatkowych "ingerencji" w kod [11]. Dodatkowo narzędzie Debug Bar zostało wzbogacone o obsługę przekierowania JQuery i żądania Ajax oraz niestandardowych kolekcji danych, specyficznych dla aplikacji Laravel.

3.3. Laravel Excel

Dodatek Laravel Excel pozwala aplikacji na obsługę arkuszy kalkulacyjnych w formacie .xls oraz .csv. Pozwala na odczytywanie danych z plików automatycznie przekształcając je w postać zbioru obiektów jak i również na zapisywanie zbiorów do plików jako wykonywalne w arkuszu kalkulacyjnym [12].

3.4. Agent

Dodatek Agent przystosowany dla narzędzia Laravel służy do wykrywania platformy, z jakiej korzysta użytkownik aplikacji. Za jego pomocą można stwierdzić z jakiego urządzenia korzysta (komputer, smartfon, tablet), systemu operacyjnego oraz przeglądarki internetowej [13].

4. Dodatkowe możliwości narzędzia Laravel

W dalszej części artykułu przedstawiono dodatkowe możliwości oferowane przez narzędzie Laravel. Żadna z nich nie została wykorzystana podczas tworzenia aplikacji testowej – wyłączając REST API są to aspekty wyróżniające Laravel na tle innych frameworków, zatem założenie stworzenia identycznej aplikacji w trzech narzędziach byłoby niemożliwe.

REST API jest możliwością, którą oferują zarówno Laravel, Symfony oraz Zend, dlatego też służyła jako dodatkowy aspekt porównawczy.

4.1. REST API

Laravel, podobnie jak Symfony i Zend, pozwala na stworzenie komunikacji pomiędzy aplikacją klienta a danymi bez konieczności implementacji do niej pełnej funkcjonalności za nią odpowiadającą. Służy do tego REST API, czyli interfejs komunikacji pomiędzy klientem a serwerem.

Komunikacja aplikacji z API w dużym skrócie przebiega poprzez wysyłanie odpowiednich ścieżek, które odpowiednio interpretowane zwracają żądane dane lub wykonują odpowiednie akcje. Dane zwracane są w najczęściej w formacie .json, interpretowanym przez każdy język programowania.

Tworzenie REST API w narzędziu Laravel nie odbiega znacząco od standardowego, webowego projektu. Jedyna istotną różnicą jest specyficzne kierowanie zapytań (ang. routing) do odpowiednich metod w kodzie aplikacji.

4.2. Wykorzystanie chmury Amazon S3

Amazon Simple Storage Service (w skrócie Amazon S3) jest to internetowa przestrzeń gromadzenia, przechowywania oraz analizy danych. Dane przechowywane w taki sposób mogą być w jednakowy sposób wykorzystywane przez różne systemu i aplikacje – zarówno internetowe jak i mobilne oraz desktopowe.

Narzędzie Laravel, ze względu na rosnącą popularność oraz skuteczność wykorzystywania Amazon S3 posiada wbudowany mechanizm pozwalający w łatwy sposób wykorzystywać przestrzeń chmurową Amazon.

Wykorzystywanie takiej przestrzeni w aplikacjach internetowych wydaje się kolejnym krokiem w rozwoju języka PHP. Zmniejsza to ilość miejsca wykorzystywaną fizycznie przez aplikację oraz pozwala wykorzystywać te same dane w kilku aplikacjach, działających również równolegle – dane mogą być na bieżąco i w tym samym momencie nadpisywane/odczytywane.

4.3. Wykorzystanie bazy GeoIP

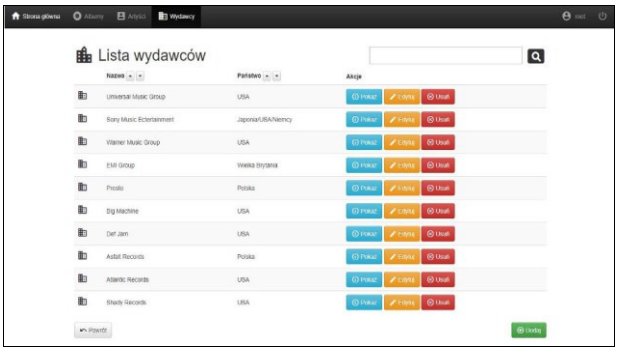
Określenie lokalizacji użytkownika korzystającego z aplikacji jest obecnie bardzo powszechne. Baza GeoIP jest to ogólnodostępny zbiór lokalizacji rozwijany przez firmę MaxMind. Dzięki takiej bazie mamy możliwość w bardzo dokładny sposób wyznaczyć lokalizację użytkownika poprzez jego adres IP.

Narzędzie Laravel posiada przystosowany dla siebie moduł korzystający z bazy GeoIP, rozwijany przez firmę Lyften. Jego implementacja w aplikacji nie odbiega niczym od instalacji standardowego modułu, istnieje także możliwość wykorzystania API prosto od firmy MaxMind lub ich bazy danych.

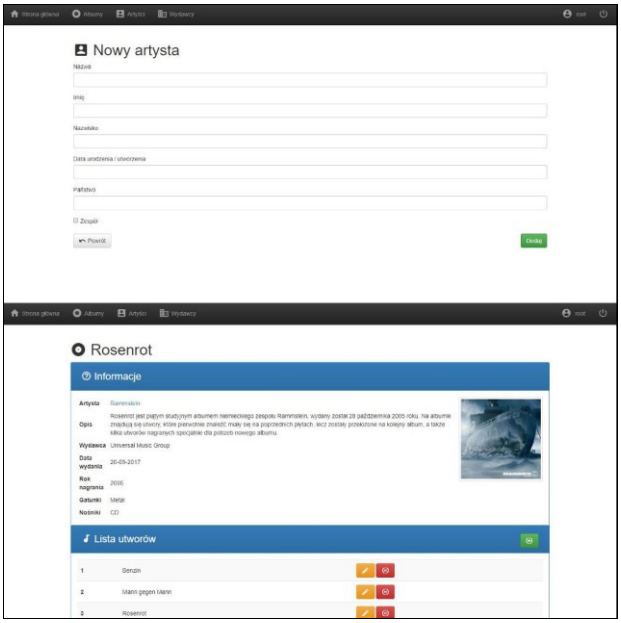
Bazę GeoIP wykorzystuje wiele rozbudowanych aplikacji na czele z Facebook.com. Stworzenie specjalnego modułu dla języka Laravel nie było na pewno konieczne, gdyż API firmy MaxMind sprawnie działa z tym narzędziem, to daje ono większy wybór sposobów i stopnia zaawansowania, w jakim deweloper chce z tej funkcjonalności korzystać – API daje dużo więcej możliwości.

5. Prezentacja rezultatów badań

Wygląd aplikacji napisanej w narzędziu Laravel (analogiczny w każdym z frameworków) przedstawiają Rys. 1 oraz Rys. 2.



Rys.1. Wygląd listy wydawców w aplikacji testowej.



Rys.2. Wygląd formularza dodawania nowego artysty oraz szczegółów albumu w aplikacji testowej

Porównanie stworzonych aplikacji zawiera analizę ich możliwości oraz wydajności. Badane aspekty zostały przedstawione w postaci tabeli, gdzie widać zestawienie trzech wykorzystanych w badaniu frameworków.

5.1. Analiza możliwościowa

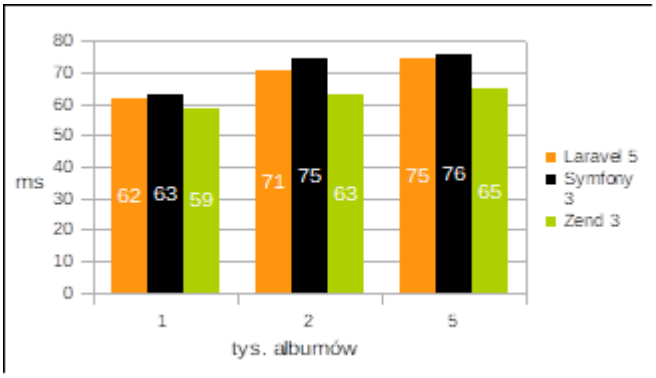
Analizę możliwościową przeprowadzono podczas tworzenia aplikacji testowych. Zaobserwowane aspekty w pracy z poszczególnymi narzędziami zebrano w postaci tabeli i przedstawiono w Tabeli 1. Ukazuje ona analizę czysto programistyczną, skupiając się na wspólnych czynnikach opisywanych frameworków i odpowiednio je opisując.

Tabela 1. Zestawienie aspektów w pracy z poszczególnymi narzędziami języka PHP.

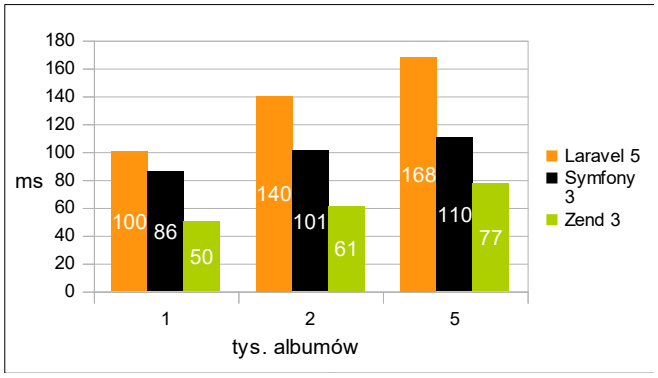
		Laravel 5	Symfony 3	Zend 3
Kontroler	Routing akcji	w oddzielnym pliku	w oddzielnym pliku, w kontrolerze jako adnotacje	w oddzielnym pliku
	Gotowy mechanizm tworzenia kontrolera	rozbudowany, z gotowymi metodami, dla wybranego modelu danych	rozbudowany z wieloma opcjami dla każdego aspektu, szybki z domyślnymi opcjami	nie
Widok	Format	.blade	.twig	.phtml
	Gotowy mechanizm tworzenia widoków	nie	tak	nie
	Zagnież-dżanie widoków	tak	tak	tak
	Dziedziczenie widoków	tak	tak	tak
Kolekcje danych	ORM	Eloquent	Doctrine	nie
	Gotowy mechanizm obsługi relacji pomiędzy tabelami	tak, lecz bez wspomagania relacji wiele do wielu, które należy definiować własnoręcznie w modelu	tak, tworzony jest schemat encji wraz z relacjami a na jego podstawie tabela i model dla obiektów w aplikacji	nie
Baza danych	Obsługiwane systemy baz danych	MSBI, MongoDB, ,MySQL, MariaDB, PgSQL, Redis,SQLite	MS BI MongoDB MySQL Oracle PgSQL MariaDB	MariaDB MySQL MSSQL Oracle PgSQL SQLite FireBird
	Gotowe mechanizmy migracyjne	tak	tak	nie
Formularze	Gotowy mechanizm generowania formularzy	tak, dostępny dopiero po instalacji modułu	tak	tak
	Implementacja z poziomu kontrolera	tak	nie	nie
	Implementacja z poziomu widoku	tak	tak	tak
Poziom trudności tworzenia aplikacji	Znajomość języka PHP	znajomość programowania obiektowego, podstawy pracy z obiektami	znajomość programowania obiektowego, podstawy pracy z obiektami	znajomość programowania obiektowego, podstawy pracy z obiektami
	Ilość gotowych mechanizmów	duża, wszystkie od razu w pełni gotowe do użycia	bardzo duża, jednak wiele jest jedynie rozszerzeniem tych najpopu-larniej-szych	mała
	Instalowanie nowych modułów	wprowadzenie zmian w dwóch plikach konfiguracyjnych	wprowadzenie zmian w dwóch plikach konfiguracyjnych	wprowadzanie zmian w wielu plikach konfiguracyjnych
	Szybkość tworzenia aplikacji w porównaniu z pozostałymi narzędziami	najszybszy	mozolna praca z plikami konfiguracyjnymi, duża swoboda w tworzeniu aplikacji ale kosztem czasu	najwolniejsza, wiele rzeczy trzeba tworzyć samemu, brak gotowych komponentów
	Mechanizm serwerowy do testowania aplikacji	tak	tak	tak

5.2. Analiza wydajnościowa

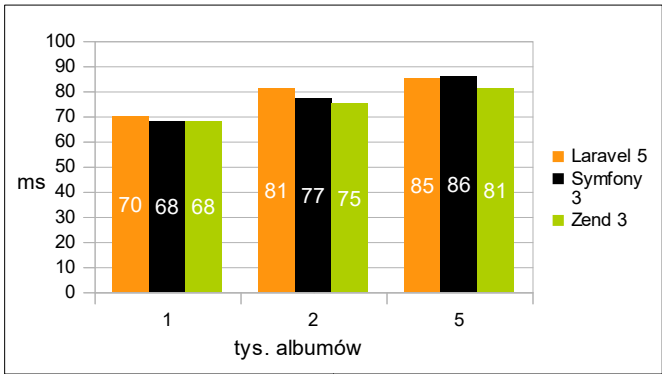
Analizę wydajnościową przedstawiono w postaci wykresów opisujących najważniejsze badane aspekty. Każdy z nich zawiera zestawienie wszystkich badanych narzędzi na tle trzech zestawów danych. W badaniach zwrócono uwagę na czas, opisany na wykresach w milisekundach (ms).



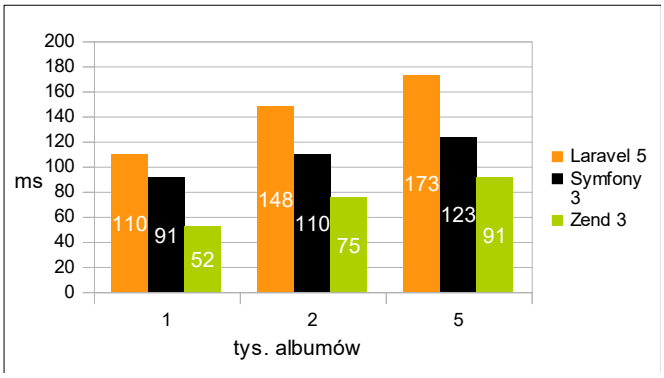
Rys.3. Czas dostępu do bazy danych.



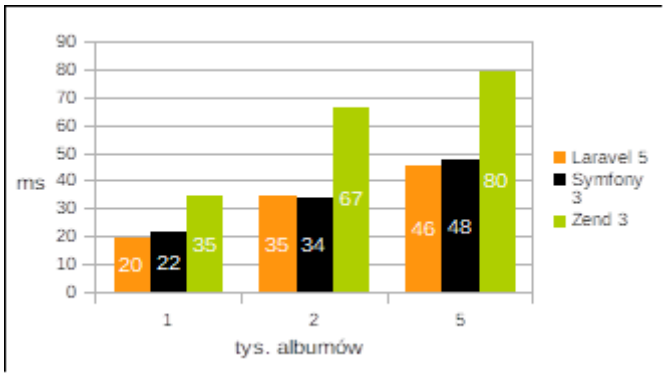
Rys.4. Czas odczytu z bazy danych.



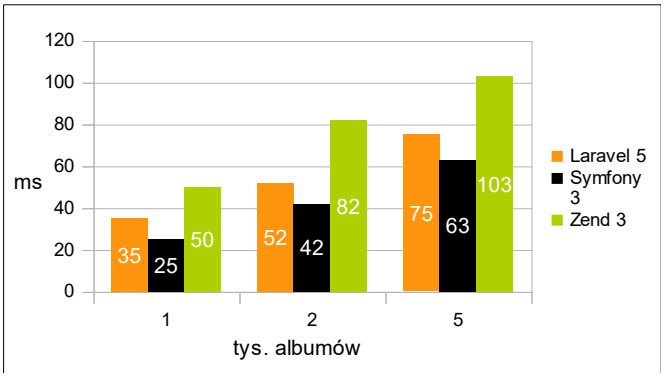
Rys.5. Czas wyszukiwania pojedynczego albumu w bazie danych.



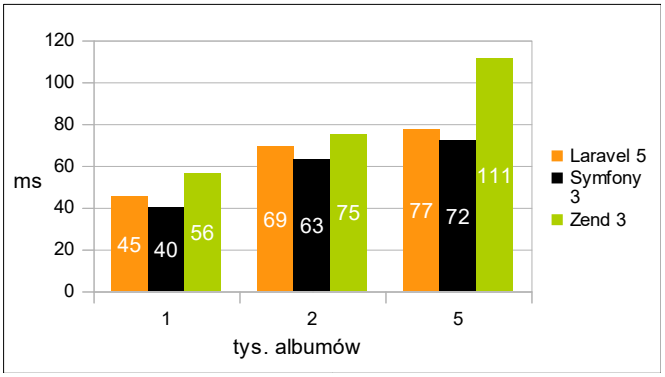
Rys.6. Czas filtrowania albumów w bazie danych.



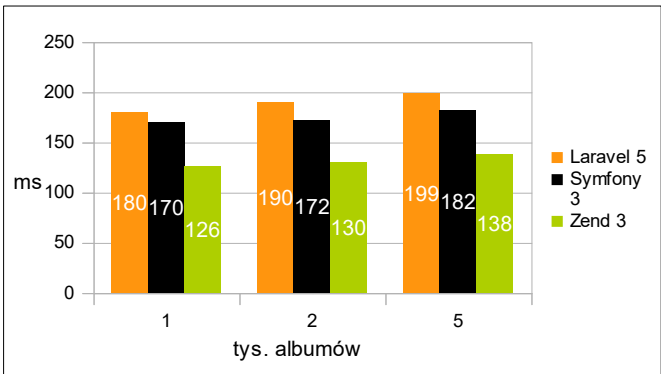
Rys.7. Czas wyszukiwania odpowiednich elementów w pobranej kolekcji danych w bazie.



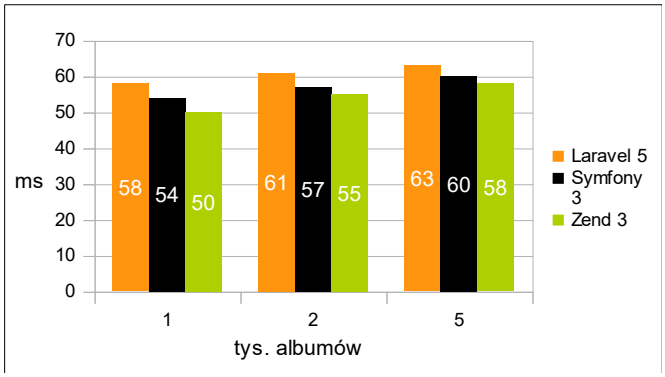
Rys.8. Czas filtrowania odpowiednich elementów w pobranej kolekcji danych w bazie.



Rys.9. Czas generowania kolekcji danych złożonych z wielu encji.



Rys.10. Czas generowania strony.



Rys.11. Czas ładowania plików graficznych.

5.3. Podsumowanie

Na podstawie powyższych analiz można stwierdzić, że Laravel jest narzędziem zdecydowanie najbardziej przystępnym dla programisty. Zarówno Symfony jak i Zend wymagają dużego doświadczenia w tworzeniu aplikacji PHP oraz skupiają się w szczególności na plikach konfiguracyjnych co dla mniej wymagającego dewelopera może okazać się zbyt przytłaczające. Najmłodsze z powyższych narzędzi ogranicza korzystanie z plików konfiguracyjnych do minimum, wymagając jedynie podawania nazw modułów przy ich instalacji czy też podania konfiguracji bazy danych w specjalnym pliku – wszystko inne narzędzie wykona samodzielnie. Także routing okazuje się bardzo intuicyjny i, w przeciwieństwie do Zend, łatwy w definicji. Mechanizm ORM Eloquent pozwala programiście zdefiniować schemat bazy danych i na jego podstawie tworzy encje oraz wspiera późniejsze ich migracje – jedynym mankamentem w tym przypadku są trudności w aktualizacji bazy, co w ORM Doctrine przebiega bezproblemowo. Format .blade, na którym oparte są widoki w Laravel jest zbliżony do .twig używanego w Symfony, posiada jednak bardziej zbliżone do natywnego języka PHP odwołania do obiektów i instrukcji.

Wydajność to w wielu przypadkach najistotniejsza kwestia, przez pryzmat której deweloper wybiera narzędzie to tworzenia internetowej aplikacji. W tym aspekcie Laravel wypadł zdecydowanie najsłabiej ze wszystkich porównywanych frameworków – w większości badań osiągał najdłuższe czasy ładowania, co pokazuje, że system obsługi kolekcji danych Eloquent oraz mechanizm widoków .blade to w porównaniu chociażby z Symfonowym Doctrine oraz .twig mechanizmy przyjazne w obsłudze kosztem szybkości działania aplikacji. Czasy w powyższych badaniach oscylowały w granicach 50-200 milisekund, więc nie są to znacząco odczuwalne przez użytkownika różnice. W bardziej rozbudowanych aplikacjach, posiadających zdecydowanie więcej niż 5 tysięcy wpisów w jednej tabeli i wielu relacji do niej, dysproporcja czasowa byłaby jednak dużo bardziej widoczna i w znaczącym stopniu ograniczała wybór Laravel jako narzędzie ich pisania.

6. Wnioski

Laravel to bardzo dynamicznie rozwijające się narzędzie. Dzięki swojej przystępności szybko zyskał popularność i zbudował społeczność, która wspiera twórców w ciągłym ulepszaniu ich produktu, między innymi poprzez moduły, które każdy użytkownik Laravel może używać na mocy otwartej licencji frameworka i wszystkich jego komponentów.

Jako narzędzie do tworzenia aplikacji internetowych wspiera wiele technologii, pozostawiając programistom pełną swobodę w ich wyborze. Sugerowane przez twórców technologie są natomiast bogato udokumentowane i w większości przypadków wystarczające dla bardziej wymagających deweloperów. Można to zaobserwować na przykładzie przeglądu literatury, gdzie przedstawiono kilka innowacyjnych pomysłów bazujących na Laravel.

Framework posiada wiele dodatkowych funkcji. W łatwy sposób pozwala na tworzenie API i integrację z innymi narzędziami, dodatkowo posiada swoisty miniframework zaprojektowany wyłącznie do API - LUMEN. Wsparcie firmy Amazon modułem bazy GeoIP daje przewagę nad innymi narzędziami PHP – Laravel jako jedyny posiada dedykowany mechanizm dający większy wybór sposobów i stopnia zaawansowania, w jakim programista chce z tej funkcjonalności korzystać.

W badaniu wydajności Laravel niestety nie wypadł najlepiej. Dwa pozostałe badane narzędzia – Symfony i Zend w większości aspektów były wydajniejsze, w tym Zend blisko dwukrotnie. Przyjazny w wykorzystaniu ORM Eloquent okazał się znacznie wolniejszy nich Doctrine zarówno w operacjach na bazie danych jak i lokalnych kolekcjach obiektów. Ogranicza to w znacznym stopniu wybór Laravel jako narzędzie do tworzenia zaawansowanych aplikacji internetowych takich jak Facebook czy YouTube. Nie zmienia to faktu, że jest to bardzo dobre narzędzie, dynamicznie rozwijane przez bogatą społeczność. Pomimo najsłabszej spośród badanych narzędzi wydajności wydaje się odpowiedni dla większości mało wymagających aplikacji, w głównej mierzę dzięki przystępności i bogatej dokumentacji.

Literatura

[1] M. Doyle, J. Walden, "An Empirical Study of the Evolution of PHP Web Application Security", Department of Computer Science Northern Kentucky University Highland Heights, 2011, s. 4.

[2] A. Zurkiewicz, M. Miłosz, "SELECTING A PHP FRAMEWORK FOR A WEB APPLICATION PROJECT - THE METHOD AND CASE STUDY", Conference: 9th international Technology, Education and Development Conference, s. 1712

[3] L. Putra, Michael, Yudishtira, "Design and Implementation of Web Based Home Electrical Appliance Monitoring, Diagnosing, and Controlling System", Procedia Computer Science volume 59, 2015, s. 36

[4] M. Żyłowska, A. Wyszyńska, E. K. Jaguszyn-Krynicka, "Defensyny - peptydy. O aktywności przeciwbakteryjnej", Instytut Mikrobiologii Uniwersytetu Warszawskiego, Zakład Genetyki Bakterii, marzec 2011, s. 223

[5] E. A. Gomez, P. Giraldo, S. Orduz, "InverPep: A database of invertebrate antimicrobial peptides", Journal of Global Antimicrobial Resistance volume 8, marzec 2017, s.13 - 17

[6] H. R. Yu, "Design and implementation of web based on Laravel framework",ACSR-Advances in Comptuer Science Research valume 6, 2015, s. 302

[7] A. Lathifah, T. Aris, „Sentinel Web: Implementation of Laravel Framework in Web Based Temperature and Humidity

Monitoring System", 2nd Int. Conference on Information Technology, Computer and Electrical Engineering, 2015, s.48

[8] http://php.net/manual/pl/preface.php [26.10.2017]

[9] http://ciencias.medellin.unal.edu.co/gruposdeinvestigacion/prospeccionydisenobiomoleculas/InverPep/public/home_en [27.10.2017]

[10] https://laravelcollective.com [26.10.2017]

[11] http://github.com/barryvdh/laravel-debugbar [26.10.2017]

[12] http://maatwebsite.nl/laravel-excel [26.10.2017]

[13] https://packalyst.com/packages/package/jenssegers/agent [26.10.2017]

[14] http://lyften.com/projects/laravel-geoip/doc/ [26.10.2017]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 116-119
Wysłane: 2017-11-20
Przyjęte: 2017-11-24

Porównanie wydajności silników gier na wybranych platformach

Paweł Skop*
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono wyniki porównania badania wydajności dwóch silników do produkcji gier Unreal Engine oraz Unity. Analiza porównawcza silników została przeprowadzona na podstawie pomiaru wybranych kryteriów dla dwóch identycznych pod względem funkcjonalności i zasobów gier, stworzonych kolejno w obu wybranych silnikach.

Słowa kluczowe: Unreal Engine; Unity; produkcja gier; wydajność silników gier

*Autor do korespondencji.

Adres e-mail: skop.pawel1@gmail.com

Comparison of performance of game engines across various platforms

Paweł Skop*
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents the results of performance study of two selected game development engines, Unreal Engine and Unity. The comparative analysis of engines was performed based on measuring selected criteria for two identical, in terms of functionality and assets, games made in selected engines.

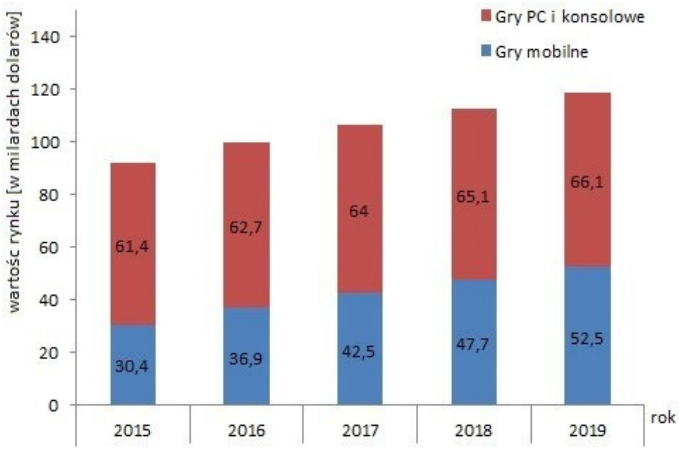
Keywords: Unreal Engine; Unity; game production; performance of game engines

*Corresponding author.

E-mail address: skop.pawel1@gmail.com

1. Wstęp

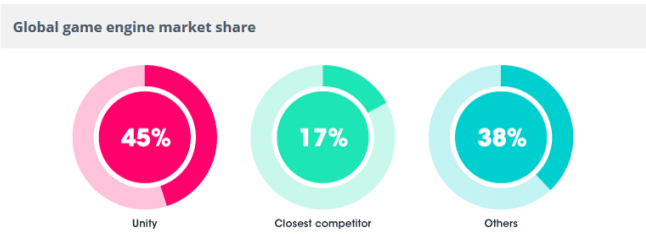
Gry wideo są jednym z najbardziej popularnych rodzajów oprogramowania. Z uwagi na ten fakt rynek gier wideo jest obszerny i został wyceniony na 99.6 miliarda dolarów w roku 2016 (Rys. 1)[1]. Wartość rynku oraz prognozowane tendencje jej wzrostu powodują powstawanie nowych studiów deweloperskich.



Rys. 1. Wartość rynku gier w poszczególnych latach, wraz z prognozą wzrostu[1].

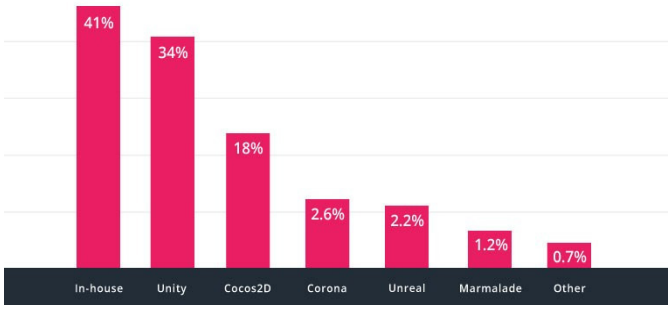
Stworzenie gry wymaga zazwyczaj wielu linii kodu. Aby ułatwić proces produkcji i odpowiedzieć na rosnące zapotrzebowanie rynku stworzono oprogramowanie ułatwiające tworzenie gier, czyli silniki gier. Współczesne silniki gier oferują przed wszystkim funkcje, takie jak:

- skryptowanie - opisywanie za pomocą kodu sposobu kontroli i interakcji obiektów w grze oraz dodatkowych zdarzeń;
- renderowanie – czyli generowanie obiektów sceny 2D lub 3D oraz innych efektów wizualnych;
- animacja – opis zmian ruchu lub deformacji obiektu w grze;
- sztuczna inteligencja – moduł odpowiedzialny za sterowanie zachowaniem obiektów w grze, takich jak ruch czy wykonywane akcje;
- silnik fizyki – udostępnia funkcje związane z opisem zjawisk fizycznych, reakcje obiektu na kolizje;
- silnik audio – odpowiedzialny za odtwarzanie efektów audio w grze;
- silnik sieciowy – umożliwia graczom interakcje w grze poprzez wymianę informacji w sieci komputerowej [2].



Rys. 2. Wykorzystanie silnika Unity w przemyśle gier w 2016 roku[3]

Gry są oprogramowaniem, które wymaga wielu zasobów sprzętowych z uwagi na ich charakter działania, czyli symulację w czasie rzeczywistym. Warto przy tym wspomnieć, że wielu twórców coraz częściej tworzy gry mając na celu działanie produktu z prędkością 60 klatek na sekundę(FPS) lub większą[4]. Aby dotrzeć do jak największego grona odbiorców warto korzystać z narzędzia, które umożliwi najbardziej optymalne wykorzystanie zasobów sprzętowych. W artykule przedstawione zostanie porównanie wydajności dwóch obecnie najpopularniejszych silników gier: Unity oraz Unreal Engine [5] (Rys. 2).



Rys. 3. Wykorzystanie silników gier na rynku gier mobilnych w 2016 roku, ranking dla 1000 najpopularniejszych darmowych gier[1]

Tabela 1. Najważniejsze cechy wybranych silników[1,6,7]

	Unity	**Unreal Engine**
Wspierany język	C#, JavaScript, Boo	C++, C#,
Platformy	Android, iOS, Windows Phone, Tizen, Windows, Windows Store Apps, Mac, Linux/Steam OS, WebGL, Playstation 4, Playstation Vita, Xbox One, Wii U, Nintendo 3DS, Oculus Rift, Google Cardboard, Steam VR, Playstation VR, Gear VR, Microsoft Hololens, Daydream, Android TV, Samsung Smart TV, tvOS, Nintendo Switch, Fire OS, Facebook Gameroom	Playstation 4, Xbox One, Nintendo Switch, Mac, Windows, Steam, Linux, HTML 5, iOS, Android, Playstation VR, Oculus Rift, Samsung Gear VR, VIVEPORT, Daydream
Silnik Fizyki	PhysX	PhysX
Silnik SI	RAIN	Kynapse

2. Cel, teza i metody badań

Celem przeprowadzonych badań było porównanie wydajności gier stworzonych w silnikach Unreal Engine i Unity.

Teza postawiona w artykule:

Unreal Engine jest wydajniejszym silnikiem gier niż Unity.

W celu potwierdzenia tej tezy przeprowadzono analizę porównawczą gier stworzonych w obu silnikach. W tym celu stworzono dwie identyczne pod względem funkcjonalności i użytych zasobów gry. Do stworzenia gier wykorzystano następujące technologie oraz narzędzia:

- Unity oraz Unreal Engine – silniki gier;
- Blender – narzędzie do modelowania 3D;
- GIMP – narzędzie do edycji grafiki.

3. Przegląd literatury

Przed realizacją zagadnień omawianych w niniejszym artykule, dokonano analizy dostępnych materiałów badawczych. Bazując na pracy I. Pachoulakis i G. Pontikakis [5] wiadomo, że jest możliwe wykonanie dwóch identycznych pod względem funkcjonalnym gier w silnikach Unity oraz Unreal Engine, co jest podstawą do przeprowadzenia badań. Na podstawie *Comparison between Famous Game Engines and Eminent Game* [6] ustalono najważniejsze do zbadania aspekty gotowych gier oraz metodę badawczą. Korzystając z pozycji [8,9,10] wiadomo, że jest możliwe wykonanie gry w wirtualnej rzeczywistości w danym silniku, jak również nakreślona zostaje w wymienionych pozycjach ogólna metoda przygotowania gry tego typu w poszczególnych silnikach.

4. Gry testowe

Na potrzeby badań wykonano proste gry w wirtualnej rzeczywistości, na podstawie pozycji [8,9,10]. Gra udostępnia możliwość rozglądania się w przygotowanej lokacji, składającej się z mapy wysokościowej, trzech modeli drzew, powierzchni wody oraz generatora cząsteczek tworzącego efekt spadającychpłatków.

Aby dodatkowo obciążyć silniki do gier dodano kwadratową mapę o wielkości 16 km^2 złożoną z 16 129 wielokątów. Generator cząstek ustawiony jest z parametrami zapewniającymi czas życia cząstek 20 sekund, maksymalną liczbę cząstek 2000. Tafla wody jest kwadratowym obiektem typu Plane o powierzchni 1 km^2. Model drzewa wykorzystany trzykrotnie został stworzony z 23 300 wielokątów.

Obie gry, wykonane kolejno w Unreal Engine i Unity, mają identyczne funkcjonalności i korzystają z takich samych zasobów. Gry umożliwiają obejrzenie zaprojektowanej lokacji w wirtualnej rzeczywistości (Rys. 4).

Rys. 4. Renderowanie sceny zaprojektowanej gry w wirtualnej rzeczywistości

5. Analiza porównawcza

W tabeli 2 przedstawiono najważniejsze parametry sprzętowe telefonu, na którym testowano gry.

Tabela 2. Parametry sprzętowe telefonu wykorzystanego w testach

Element	Stan/Wersja
System operacyjny	Android 5.1.1
Model telefonu	Huawei Y560-L01
Model procesora	Oualcom MSM8909
Liczba rdzeni	4
Częstotliwość procesora	1.1 GHz
Rozdzielczość	854*480
Pamięć Ram	1 GB

Do porównania obu gier, a tym samym wydajności wybranych silników wybrano następujące kryteria:

- liczba klatek na sekundę (FPS);
- ilość zajmowanej pamięci RAM;
- ilość zajmowanej pamięci Flash;
- obciążenie procesora.

6. Wydajność gier

W celu porównania wydajności zaprojektowanych gier, zmierzono parametry właściwe dla wybranych kryteriów. Do testów przygotowano dwa scenariusze:

- statyczna scena wirtualnej rzeczywistości;
- możliwość rozglądania się w scenie.

W celu zmierzenia wybranych parametrów skorzystano z aplikacji takich jak OS Monitor oraz FPS Meter. Wyniki pomiarów przedstawiono w tabelach 3 i 4.

Z uwagi na charakter pamięci Flash pomiaru dokonano tylko raz. Wyniki przedstawiono w tabeli 5.

Tabela 3. Pomiar dla sceny statycznej

Mierzona wartość	Unity	Unreal Engine
Liczba klatek na sekundę	43	36
Obciążenie procesora	20%	19%
Ilość zajmowanej pamięci Ram	107Mb	143Mb

Tabela 4. Pomiar podczas wykonywania ruchu

Mierzona wartość	Unity	Unreal Engine
Liczba klatek na sekundę	40	28
Obciążenie procesora	24%	30%
Ilość zajmowanej pamięci Ram	109Mb	142Mb

Tabela 5. Pomiar zajmowanej pamięci Flash

Mierzona wartość	Unity	Unreal Engine
Ilość zajmowanej pamięci Flash	49,2Mb	285Mb

7. Wnioski

Na podstawie przeprowadzonych badań można sformułować następujące wnioski.

- różnice wydajności pomiędzy badanymi silnikami gier, czyli Unity i Unreal Engine są nieznaczne przy uwzględnieniu możliwości współczesnych urządzeń mobilnych;
- unity osiąga lepsze wyniki dla liczby klatek na sekundę i ilości zajmowanej pamięci RAM przy jednoczesnym mniejszym użyciu zasobów procesora;
- projekt stworzony w Unity zajmuje mniej pamięci Flash, co może być powodem większej popularności silnika Unity na rynku mobilnym.

Wyniki badań zaprzeczają postawionej tezie, gra stworzona w Unreal Engine jest mniej wydajna na platformie Android pod względem aspektów związanych z płynnością działania gry, a więc tych, które są najbardziej widoczne dla przeciętnego użytkownika. Należy jednak zauważyć, że różnice w wydajności nie są znaczne. Dodatkowo warto wspomnieć, że silnik Unity dzięki wsparciu większej liczby platform może zapewniać dostęp do szerszego grona odbiorców.

Literatura

[1] https://unity3d.com/public-relations [04.05.2017]

[2] B. Cowan, B. Kapralos, A Survey of Frameworks and Game Engines for Serious Game Development, 2014 IEEE 14th International Conference on Advanced Learning Technologies.

[3] https://thenextweb.com/gaming/2016/03/24/engine-dominating-gaming-industry-right-now [04.05.2017]

[4] J. Rojas, Getting Started with Videogame Development, 2013 26th Conference on Graphics, Patterns and Images Tutorials.

[5] Pachoulakis, G. Pontikakis, Combining features of the Unreal and Unity Game Engines to hone development skills, 2015 9th International Conference on New Horizons in Industry, Business and Education.

[6] P. Mishra, U. Shrawankar, Comparison between Famous Game Engines and Eminent Games, 2016 International Journal of Interactive Multimedia and Artificial Intelligence.

[7] https://www.unrealengine.com/what-is-unreal-engine-4 [04.05.2017]

[8] X. Jing, Design and implementation of 3D Virtual Digital Campus – Based on Unity3D, 2016 Eighth International Conference on Measuring Technology and Mechatronics Automation.

[9] S. Wang, Z. Mao, Ch. Zeng, H. Gong, S. Li, B. Chen, A New Method of Virtual Reality Based on Unity3D, 2010 18th International Conference on Geoinformatics.

[10] J. Matej, Virual Reality and Vehicle Dynamics in Unreal Engine Enviroment, MM Science Journal, 2016.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 120-125
Wysłane: 2017-11-14
Przyjęte: 2017-11-20

Analiza porównawcza wybranych interfejsów człowiek-komputer

Kamil Bartosz Podkowiak*, Damian Burak*, Tomasz Szymczyk
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł dotyczy porównania dwóch technologii manipulowania obiektami 3D w kontekście intuicyjności, ergonomii i wygody użytkowania. Porównanie zostało przeprowadzone na podstawie czasów wykonania zadań testowych oraz ocen uzyskanych z ankiet przeprowadzonych na grupie 15 osób. Opracowano szczegółowe scenariusze badawczo-testowe a także dodatkowe pytania ankietowe. Każdy z respondentów przed wypełnieniem ankiety miał możliwość zapoznania się z urządzeniem poprzez wykonanie 3 prób w dwóch scenariuszach testowych (po jednym na urządzenie).

Słowa kluczowe: zSpace; SpaceNavigator; mysz 3D; Stylus; inferfejsy człowiek-komputer; grafika 3D

*Autor do korespondencji.

Adresy e-mail: kamil.podkowiak@pollub.edu.pl, damian.burak@pollub.edu.pl

Comparative analysis of selected human-computer interfaces

Kamil Bartosz Podkowiak*, Damian Burak*, Tomasz Szymczyk
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. This article deals with the comparison of two technologies for manipulating 3D objects in terms of ease of use, ergonomics and intuitiveness. The comparison was based on the results obtained from the outcome of test tasks and questionnaires performed on the group of 15 people. Detailed research and test scenarios and additional questionnaires have been made. Each of the respondents had the opportunity to familiarize themselves with the device by performing three tests in two test scenarios (one per device).

Keywords: zSpace; SpaceNavigator; 3D mouse; Stylus; human-computer interfaces, 3D graphics

*Corresponding author.

E-mail addresses: kamil.podkowiak@pollub.edu.pl, damian.burak@pollub.edu.pl

1. Wstęp

Interakcja człowiek - komputer (ang. Human - Computer Interaction) to nauka i obszar badawczy zajmujący się sposobem komunikacji pomiędzy użytkownikiem a komputerem, poprzez interfejs użytkownika UI (ang. User Interface). Od momentu pojawienia się pierwszych komputerów osobistych podstawowymi narzędziami do komunikacji z komputerem były klawiatura i myszka komputerowa. Pozwalały one na wprowadzanie tekstu, manipulowanie obiektami i jak dotąd w tych zastosowaniach nie mają sobie równych i wiodą prym wśród narzędzi HCI. Jednakże wraz z rozwojem technologii 3D, rozszerzonej i wirtualnej rzeczywistości, zaistniała realna potrzeba na narzędzie pozwalające manipulować obiektami w trójwymiarze, zapewniające jednocześnie precyzję i wygodę użytkowania [1], [2].

Każdy grafik 3D w swojej pracy zawodowej musi operować w trójwymiarowej przestrzeni i zajmować się modelami obiektów 3D. Ważną decyzją, którą musi podjąć jest wybór odpowiedniego narzędzia, które nie tylko będzie adekwatne do wykonywanej pracy lecz również znacznie ją usprawni i będzie wygodne w wielogodzinnej pracy. W niniejszym artykule porównano dwa nowoczesne narzędzia: mysz 3D - Spacenavigator i interfejs zSpace 200. Pierwsze pomimo, iż obecne na rynku od początku lat 90 ubiegłego wieku, z uwagi na cenę i specyfikę zastosowania nie jest często występujące na wyposażeniu komputerów przeciętnego użytkownika. Drugi interfejs zSpace swoją premierę miał w 2015 roku i jest jeszcze rzadziej spotykany.

2. zSpace 200

Pierwszą badaną technologią jest zestaw zSpace, interaktywne i przede wszystkim intuicyjne oraz proste w obsłudze urządzenie umożliwiające użytkownikom wręcz immersyjną interakcję z obiektami w trójwymiarowym środowisku. Taka forma interakcji użytkownika ze światem wirtualnym jest nie tylko bardziej naturalna od standardowego obrazu 2D ale przede wszystkim pozwala lepiej zrozumieć istotę obserwowanej scenerii. To w połączeniu z precyzyjnym kontrolerem Stylus daje szereg zastosowań dla monitora zSpace m.in. w takich branżach jak szeroko pojęta edukacja czy medycyna.

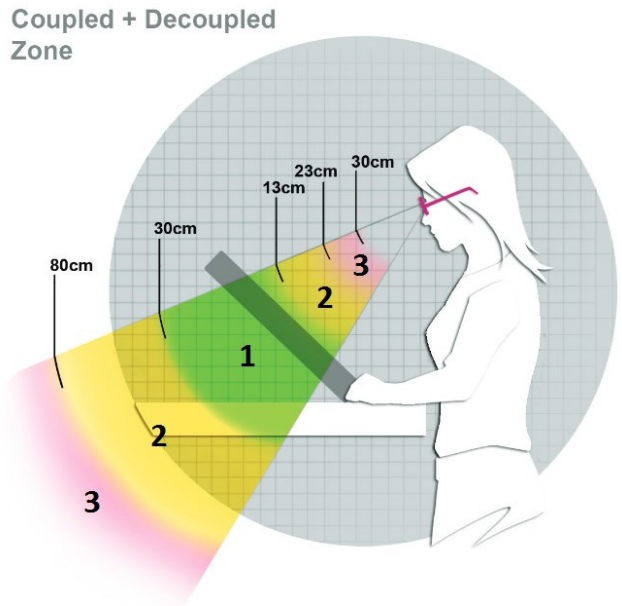
Rys. 1. Zestaw zSpace [3]

Zestaw zSpace składa się z trzech kluczowych elementów: interaktywnego monitora 3D (1), okularów (2) oraz kontrolera Stylus (3). Cały zestaw został zaprezentowany na rysunku 1. Monitor zSpace posiada 23,6” matową matrycę umożliwiającą wyświetlanie obrazu w pasywnym trybie 3D, 4 kamery śledzące ruch głowy użytkownika, znajdujące się w górnej części monitora, 1 port USB typu B, 4 porty USB typu A, 1 port DVI, 1 DisplayPort w wersji 1.2, port do podłączania kontrolera oraz slot do kabla zasilającego. Natywna rozdzielczość ekranu wynosi 1920x1080 Full HD. W skład zestawu wchodzą również okulary dla użytkownika (3D, sterujące sceną) i obserwatora (2D). Te przeznaczone dla użytkownika umożliwiają oglądanie obrazu w trybie 3D. Wyposażone są także w markery, które w interakcji z wbudowanymi kamerami monitora zSpace umożliwiają użytkownikowi manipulowanie wyświetlanym obrazem, czyli dowolne zmienianie kąta widzenia obrazu. Druga para przeznaczona dla obserwatorów umożliwia wyświetlanie obrazu 2D oraz nie posiada markerów. Na rysunku 2 przedstawiono umiejscowienie markerów na okularach użytkownika.

Rys. 2. Okulary 3D z markerami. Na czerwono zaznaczono znaczniki wykrywane przez kamery

Ostatnim kluczowym elementem zestawu jest manipulator Stylus, jest on połączony z monitorem za pomocą kabla, który nieznacznie, negatywnie wpływa na komfort korzystania. Kontroler wyposażony jest w 3 przyciski oraz diodę umieszczoną w centralnej części urządzenia. Dodatkowo Stylus zawiera opcję wibracji, która może posłużyć jako dodatkowe źródło informacji dla użytkownika. Właściwościami tych elementów można manipulować z poziomu kodu oprogramowania. Pozycja Stylusa określana jest w trzech płaszczyznach X, Y i Z, względem pozycji jego wierzchołka.

Rys. 3. “Stylus” - manipulator zestawu zSpace [4]



Rys. 4. Strefy wyświetlanego obrazu 3D [5]

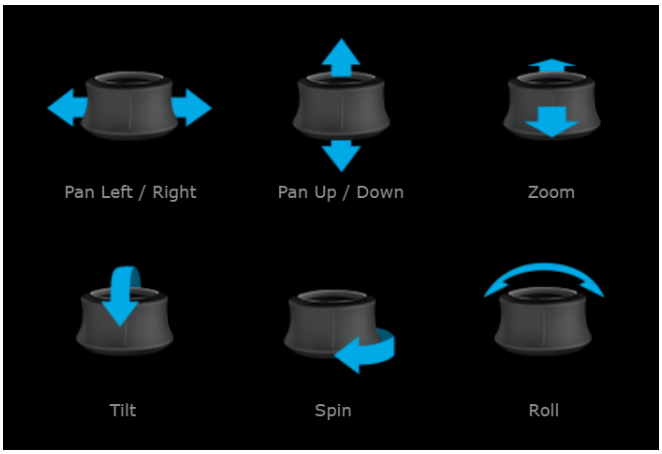
Wyświetlany obraz 3D można podzielić na trzy sektory rozszerzonej rzeczywistości, które różnią się komfortem użytkowania. Strefy zaznaczone na zielono (1) są domyślnie najlepszą opcją przy korzystaniu z monitora zSpace, obserwowanie obrazu 3D w tym obszarze nieznacznie wpływa na zmęczenie oczu. Obserwowanie obiektów znajdujących się w strefie pomarańczowej (2) przez dłuższy okres może powodować większy odczuwalny dyskomfort. Natomiast obserwowanie obiektów w strefie czerwonej (3) powinno być relatywnie krótkie gdyż sektor ten najmocniej obciąża oczy. Jak możemy zauważyć na rys. 4 monitor zSpace umożliwia obserwowania obiektów w „głębi monitora” jak i „przed monitorem” [5].

3. Mysz 3D

Badanym modelem myszy 3D jest SpaceNavigator, produkowany przez firmę 3Dconnexion, która dostarcza także inne, zbliżone rozwiązania. Według producenta SpaceNavigator zapewnia swobodę intuicyjnej i precyzyjnej nawigacji 3D.

Rys. 5. SpaceNavigator - mysz 3D [6]

Mysz składa się z metalowego obciążnika, czujnika wykorzystującego sześć stopni swobody ruchu pokrytego gumowym materiałem ułatwiającym chwyt oraz dwoma programowalnymi przyciskami. Do myszy dołączona jest płyta CD ROM, na której znajdują się sterowniki, wersje demonstracyjne i narzędzia. Wymiary myszy to 78 mm (długość) na 78 mm (szerokość) na 53 mm (wysokość), a jej waga to 479 gramów [6].



Rys. 6. SpaceNavigator – zakres ruchów [6]

Dzięki zastosowaniu opatentowanego przez 3Dconnexion czujnika istnieje możliwość precyzyjnego i swobodnego poruszania trójwymiarowym obiektem w 3 wymiarach bądź manipulacją kamerą na trójwymiarowej scenie (np. programu graficznego), wykonując przedstawione na rysunku 6 ruchy.

4. Zastosowanie badanych urządzeń

Monitor zSpace znajduje zastosowanie w specjalistycznych systemach pomocniczych w dziedzinie chirurgii i diagnostyki. W artykule [7] połączono korzyści płynące z wykorzystania ekranu interaktywnego i trójwymiarowych modeli narządów wykonanych za pomocą druku trójwymiarowego. zSpace w takiej konfiguracji stał się narzędziem umożliwiającym wygenerowanie dokładnych modeli 3D narządów pacjenta w wirtualnej przestrzeni, co w połączeniu z organem wydrukowanym za pomocą drukarki 3D umożliwiło dokładniejszy i co najważniejsze bezinwazyjny, opis stanu narządu pacjenta jeszcze przed zabiegiem.

Kolejnym przykładem zastosowania zSpace w medycynie jest projekt frameworku FAUST, który został dokładnie opisany w artykule [8]. Framework ten umożliwia manipulację wygenerowanym modelem 3D koła tętniczego mózgu oraz dodawaniem do niego adnotacji. W artykule wykazano kilka właściwości zestawu zSpace w kontekście wdrożenia go do placówek medycznych. W zestawieniu z odpowiednim oprogramowaniem umożliwia on dokładne i wygodne, ze względu na swoją intuicyjność formę analizy wyników medycznych. W kwestii wymaganego miejsca w pomieszczeniu zwrócono uwagę na fakt iż monitor zSpace działa również jako wyświetlacz 2D, więc może zastąpić monitory z których dotychczas korzystano. Cena zestawu zSpace jest relatywnie niska w porównaniu z ceną aparatury medycznej.

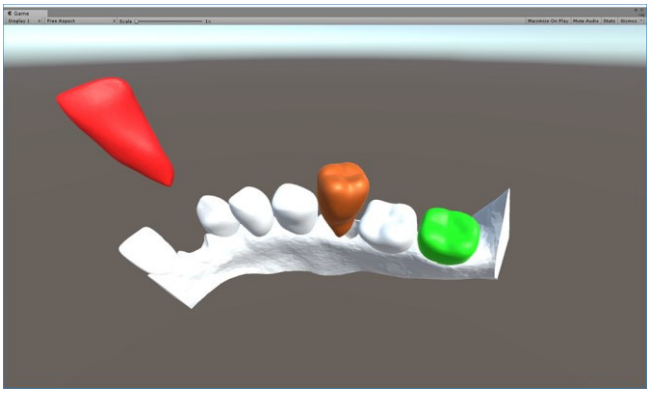
Jednym z przykładów zastosowania myszki 3D jest branża rozrywkowa, a konkretnie gry wideo. W artykule [9] wykazano iż dla osób posiadających motoryczne dysfunkcje wykorzystanie myszki 3D w grze „Second Life" uprościło rozgrywkę względem standardowego zestawu mysz 2D i klawiatura. Badania przeprowadzono na dziesięcioosobowej grupie, połowa osób przypisana była do myszki 3D, a druga połowa do myszki 2D. Następnie obie grupy do wykonania miały 13 scenariuszy, które dotyczyły aktywności w omawianej grze. Głównym wnioskiem przeprowadzonych badań było stwierdzenie iż 3 z 5 najtrudniejszych scenariuszy stało się łatwiejsze z wykorzystaniem myszki 3D. Ankietowani respondenci przychylniej wypowiadali się na temat myszy 3D.

W kolejnym przykładzie użyto myszy 3D SpaceNavigator jako manipulatora obiektów w grze polegającej na układaniu trójwymiarowych puzzli [10]. W ramach badań porównano czas, w którym badanym osobom udało się wykonać zadanie pomyślnie. Badanych podzielono, pierwsza grupa wcześniej przeszła szkolenie z obsługi myszy 3D, natomiast druga nie. Zgodnie z oczekiwaniami, ustalono, że osoby przeszkolone miały niższy średni czas ustawiania puzzla 3D, a także że osoby młodsze lepiej poradziły sobie z zadaniem. Stwierdzono, że interakcja z obiektami w przestrzeni trójwymiarowej jest trudnym zadaniem, ponieważ użytkownik traci perspektywę głębi podczas wykonywania zadań na płaskim, dwuwymiarowym ekranie.

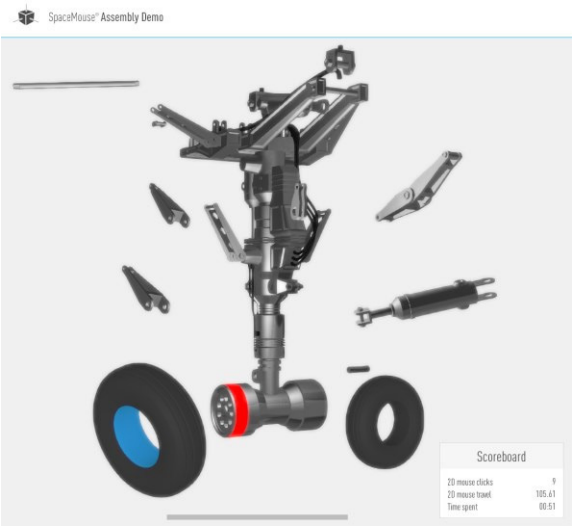
5. Procedura badawcza

Badania przeprowadzono na grupie piętnastu osób, respondenci do wykonania mieli dwa scenariusze, jeden dotyczący zSpace i drugi wykorzystujący mysz 3D. Na każdy ze scenariuszy przypadały trzy próby, po zakończonej procedurze osoby były ankietowane pod kątem intuicyjności i wygody użytkowania obu urządzeń. Na potrzeby badań stworzono autorską aplikację na technologię zSpace oraz wykorzystano demo zawarte w oprogramowaniu myszki 3D. Osobami ankietowanymi byli studenci kierunków technicznych w przedziale wiekowym 23-35 lat.

Scenariusz badawczy dotyczący zestawu zSpace opierał się na wykonaniu trzech prób ułożenia obiektów w przestrzeni trójwymiarowej z wykorzystaniem autorskiej aplikacji. Istotą zadania było dopasowanie trzech obiektów zębów do odpowiadającym im miejsc w szczęce. Precyzję dopasowania określała zarówno lokalizacja i rotacja zęba względem docelowej pozycji, stopień dopasowania weryfikowany był przez aplikację. Na rysunku X przedstawiono zęby w trzech różnych stanach, ten najdalej oddalony od miejsca docelowego ma kolor czerwony, obiekt w stanie pośrednim ma kolor brązowy natomiast ten najlepiej dopasowany zielony. Kolor płynnie zmieniał się z czerwonego na zielony wraz z przybliżaniem do położenia docelowego.



Rys. 7. Poglądowy zrzut ekranu autorskiej aplikacji wykonanej na platformę zSpace



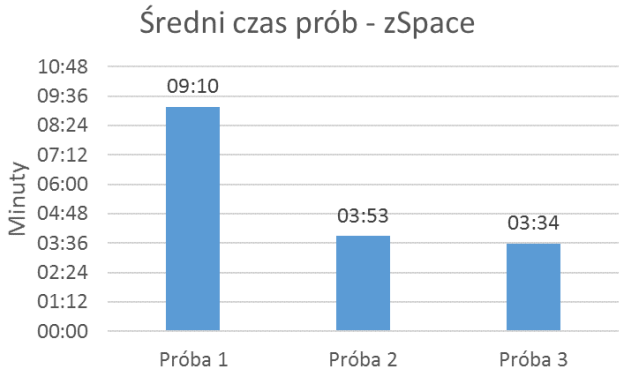
Rys. 8. Poglądowy zrzut ekranu dema wykorzystanego w badaniach

Scenariusz badawczy dotyczący myszy 3D opierał się na trzykrotnym wykonaniu dema zawartego w oprogramowaniu dołączonym do tej myszy. Polegało ono na poruszaniu modelem podwozia samolotu za pomocą myszy 3D i dopasowania elementów używając myszy tradycyjnej poprzez sekwencyjne klikanie na element i jego miejsce docelowe. Dla ułatwienia miejsca te były oznaczone kolorem czerwonym. Nie było możliwe wykonanie zadania bez wykorzystania myszy 3D, a sprawniejsze posługiwanie się nią skutkowało krótszym czasem końcowym.

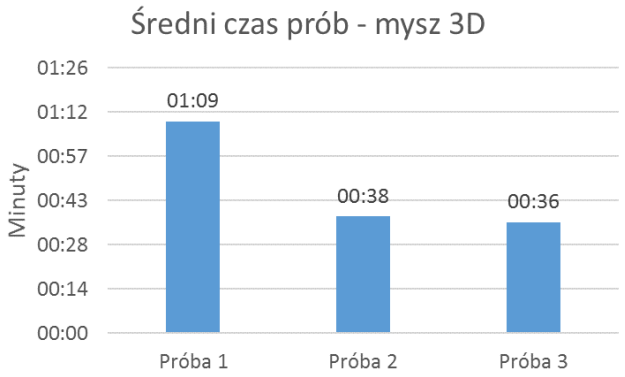
6. Wyniki badań

Podczas przeprowadzania badań wykonano pomiary czasów prób. Otrzymane wyniki zestawiono na poniższych wykresach w formie sumarycznych uśrednionych czasów poszczególnych prób. W celu lepszego zobrazowania uzyskanych wyników usunięto ze średnich wartości odstające (średnia ucinana).

Z otrzymanych wykresów średnich czasów prób można wywnioskować, iż w obu przypadkach kolejne próby były szybsze od poprzedzających. Taką sytuację można tłumaczyć procesem uczenia, który szerzej znany jest pod terminem krzywej uczenia. Oznacza to, iż obiekt badany podejmując kolejne próby skraca czas ich wykonania do momentu osiągnięcia maksimum jego możliwości, czyli de facto asymptoty krzywej uczenia.

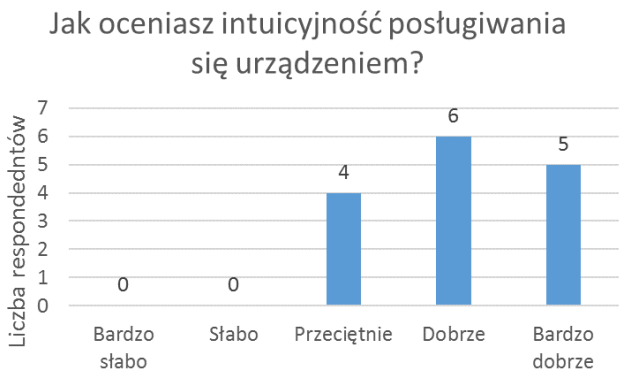


Rys. 9. Uśrednione czasy prób - zSpace



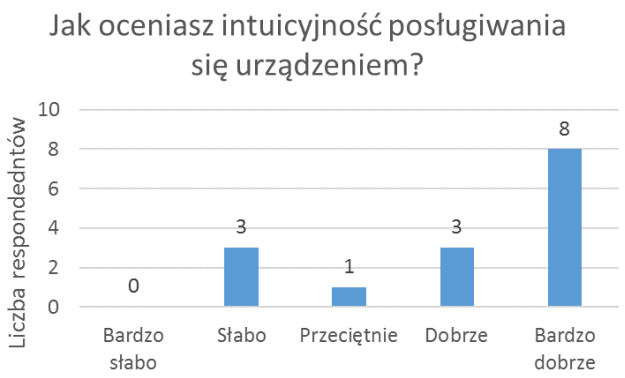
Rys. 10. Uśrednione czasy prób – mysz 3D

Rezultatem przeprowadzonych ankiet są odpowiedzi osób badanych na temat intuicyjności i wygody użytkowania urządzeń. Pięciostopniowa skala (znajdująca się na wykresach na osi X) zastosowana w ankiecie odzwierciedlała następujące wartości: 1 - Bardzo słabo, 2 - Słabo, 3 - Przeciętnie, 4 - Dobrze, 5 - Bardzo dobrze. Natomiast na osi Y znajduje się liczba osób, które zagłosowały na daną opcję.

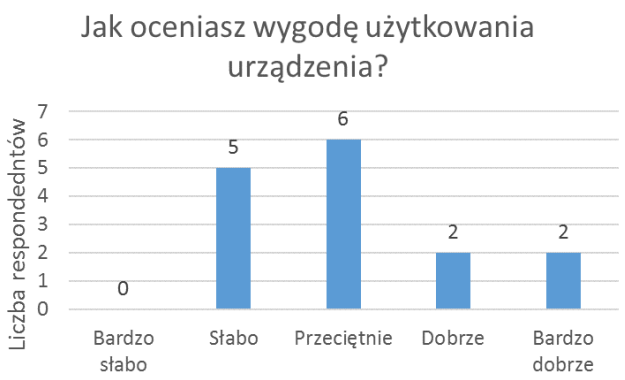


Rys. 11. Wyniki ankiety intuicyjności posługiwania się urządzeniem dla zestawu zSpace

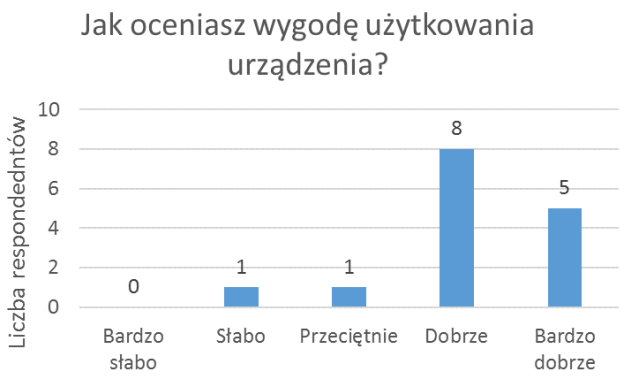
Po podliczeniu średnich wartości uzyskano potwierdzenie wcześniej założonej hipotezy: Średnia ocena intuicyjności zestawu zSpace wyniosła 4,3 natomiast dla myszy 3D 4,2. Następnie przebadano w analogiczny sposób wygodę użytkowania urządzeń.



Rys. 12. Wyniki ankiety intuicyjności posługiwania się urządzeniem dla myszy 3D



Rys. 13. Wyniki ankiety wygody użytkowania dla zestawu zSpace



Rys. 14. Wyniki ankiety wygody użytkowania dla myszy 3D

Ankietowani często narzekali na różne dolegliwości wynikające z długotrwałego używania zSpace, które wynikały z konieczności utrzymywania ręki razem z kontrolerem w powietrzu, co następnie miało odzwierciedlenie w ankiecie. Średnia ocena wygody użytkowania zSpace znacznie odbiega od oceny użytkowania myszy 3D, są to odpowiednio wartości 3,2 dla zSpace i 4,2 dla myszy 3D.

Ankietowani dodatkowo wskazali kilka zastosowań dla badanych technologii. W przypadku zSpace respondenci podali m.in. takie zastosowania jak: modelowanie 3D, motoryzacja, rozrywka, medycyna, projektowanie. Dla myszy 3D wskazano obszary takie jak automatyka, robotyka, modelowanie 3D, motoryzacja, sterowanie dronami, medycyna, przemysł.

7. Wnioski

Na podstawie przeprowadzonych badań można wywnioskować iż rozwiązanie oferowane przez firmę zSpace wykazuje się większą intuicyjnością użytkowania. Otrzymane wyniki argumentować można faktem iż kontroler zestawu zSpace swoją budową przypomina długopis, co implikuje łatwość i naturalność poruszania nim. Kolejnym czynnikiem potwierdzającym zaobserwowany fakt jest precyzja odwzorowania ruchów użytkownika, których translacja następuje w stosunku 1:1. Stylus w połączeniu z monitorem zSpace oferuje lepszą wizualizację i immersyjną interakcję z przestrzeni 3D niż połączenie mysz 3D i monitor.

Według badanych większą wygodą wykazuje się urządzenie dostarczone przez firmę 3Dconnexion. Wynik tego badania skorelowany jest z wypowiedziami respondentów podczas przeprowadzanych prób - około połowa badanych wskazywała na uczucie zmęczenia podczas korzystania ze stylusa. W przeciwieństwie do stylusa, praca z myszą 3D nie wymaga od użytkownika utrzymywania ręki w powietrzu, co miało bezpośredni wpływ na odbiór urządzeń w tym aspekcie. Sprzęt posiada metalowy obciążnik, dzięki któremu nie zmienia swojej pozycji podczas pracy, a ręka może swobodnie leżeć na blacie biurka. Natomiast w przypadku Stylusa ręka przez większość czasu znajduje się w powietrzu co przy dłuższym użytkowaniu obciąża nadgarstek i mięśnie ręki.

Pod uwagę należy wziąć fakt, iż trzech na piętnastu badanych miało wady wzroku co powodowało trudności w odbiorze obrazu 3D. Taki stan rzeczy mógł wpłynąć negatywnie na ogólny odbiór urządzenia co w efekcie mogło zaniżyć wynik technologii zSpace w obu badanych kategoriach.

Zspace jest to sprzęt o wiele droższy, ale też jednocześnie posiadający szerszą gamę zastosowań niż mysz 3D, jest też rozwiązaniem o wiele bardziej kompletnym, gdyż w skład zestawu wchodzi także monitor 3D, dzięki któremu możliwe jest wyświetlanie trójwymiarowego obrazu. Możliwość pisania własnych aplikacji na zestaw zSpace za pomocą oprogramowania Unity3D jest też niewątpliwie dużą zaletą. Mysz 3D to sprzęt wielokrotnie tańszy ale też bardziej ograniczony, ponieważ stanowi jedynie manipulator, dodatkowy kontroler, który wpinamy do portu USB komputera. Można go używać jedynie w aplikacjach, które go wspierają.

Literatura

[1] Szymczyk Tomasz, Skulimowski Stanisław, The use of virtual and augmented reality in the teaching process, INTED 2017 : 11th International Technology, Education and Development

Conference; [Red:] Gomez Chova L., Lopez Martinez A., Candel Torres I. - [B.m.]: IATED Academy, 2017, s. 6570-6577 .- (INTED 2017 Proceedings, ISSN 2340-1079)

[2] T. Szymczyk, MAKE LEARNING MORE INTERESTING BY USING VIRTUAL REALITY, EDULEARN 17 : 9th International Confrence on Education and New Learning Technologies, Barcelona (Spain), 3rd-5th of July, 2017 : conference proceedings ; [Red:] Gomez Chova L., Lopez Martinez A., Candel Torres I. - Barcelona: IATED Academy, 2017, s. 3513-3519 .- (EDULEARN 17 Proceedings, ISSN 2340-1117)

[3] http://www.reeduca.com.mx/img/zSpace2.jpg [12.11.2017]

[4] Getting Started, http://developer.zspace.com/docs/ [12.11.2017]

[5] Style Guide & Best Practices, http://developer.zspace.com/docs/ [09.11.2017]

[6] https://3Dconnexion.com [09.11.2017]

[7] Maki Sugimoto, Augmented Tangibility Surgical Navigation Using Spatial Interactive 3-D Hologram zSpace with OsiriX and Bio-Texture 3-D Organ Modeling, 2015

[8] Patrick Saalfelda, Sylvia Glaßer, Oliver Beuin, Bernhard Preima, The FAUST framework: Free-form annotations on unfolding vascular structures for treatment planning, 2017

[9] Márcio Martins, António Cunha, Irene Oliveira, Leonel Morgado, Usability test of 3Dconnexion 3D mice versus keyboard + mouse in Second Life undertaken by people with motor disabilities due to medullary lesions, 2013

[10] Deyberth Hernando Riaño Nuñez, Jaime Barrios Hernández, Pablo Alejandro Figueroa Forero, Jose Tiberio Herna Peñaloza, Brain Puzzle 3D, 2015

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 126-131
Wysłane: 2017-11-24
Przyjęte: 2017-11-27

Tworzenie modułowych aplikacji JavaScript – porównanie rozwiązania otwartego i komercyjnego

Patrycja Jabłońska*
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Tematem artykułu jest analiza porównawcza dwóch popularnych szkieletów programistycznych do tworzenia modułowych aplikacji w JavaScript: AngularJS (typu open source) i Ext JS (pakiet komercyjny). Do badań wykorzystano dwie autorskie aplikacje o identycznej funkcjonalności, zaimplementowane w obu frameworkach. Porównano strukturę obu aplikacji, łatwość implementacji komponentów GUI, wybrane metryki kodu, dostępność dokumentacji oraz wsparcie społecznościowe. Wyniki zestawiono w postaci tabel.

Słowa kluczowe: AngularJS, ExtJS, JavaScript

*Autor do korespondencji.

Adres e-mail: patrycja.jp@op.pl

Developing application in JavaScript - comparison of commercial and open source solution

Patrycja Jabłońska*
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. Subject of this article is comparative analysis of two popular frameworks in JavaScript: AngularJS (open source) and Ext JS (commercial package). There were two original applications used for this study, each implemented in one of frameworks. Structure of applications, difficulty of implementing GUI components, code metrics, documentation availability and community support were compared. Results were presented in charts.

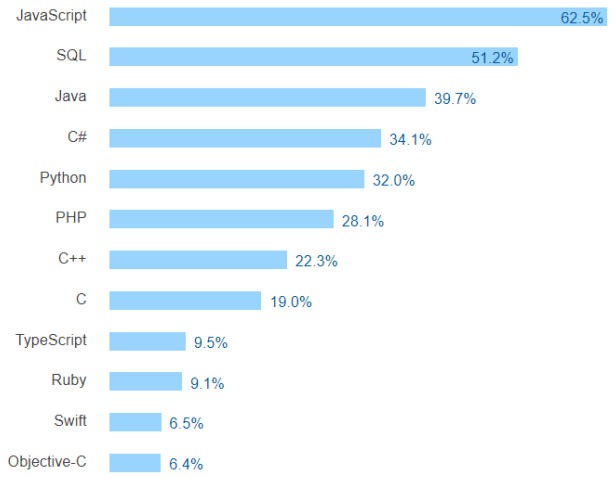
Keywords: AngularJS, ExtJS, frameworks, JavaScript

*Corresponding author.

E-mail address: patrycja.jp@op.pl

1. Wstęp

Według corocznych ankiet prowadzonych przez StackOverflow [1], od czterech lat JavaScript plasuje się na pierwszym miejscu najpopularniejszych technologii programistycznych (Rys.1.)



Rys. 1. Najpopularniejsze technologie według StackOverflow 2017 [1]

Tak duże zainteresowanie powstało dzięki wzrostowi zapotrzebowania na strony internetowe, co z kolei spowodowało szybki rozwój technologii, umożliwiając tym samym poszerzenie zastosowań tych właśnie stron. Do tej pory proste strony zawierające jedynie pewne informacje rozrosły się do serwisów informacyjnych, społecznościowych a nawet pozwalających na zarządzanie różnego typu organizacjami. W związku z wzrostem zastosowań, a także rozmiaru stron, nazywane są one aplikacjami webowymi.

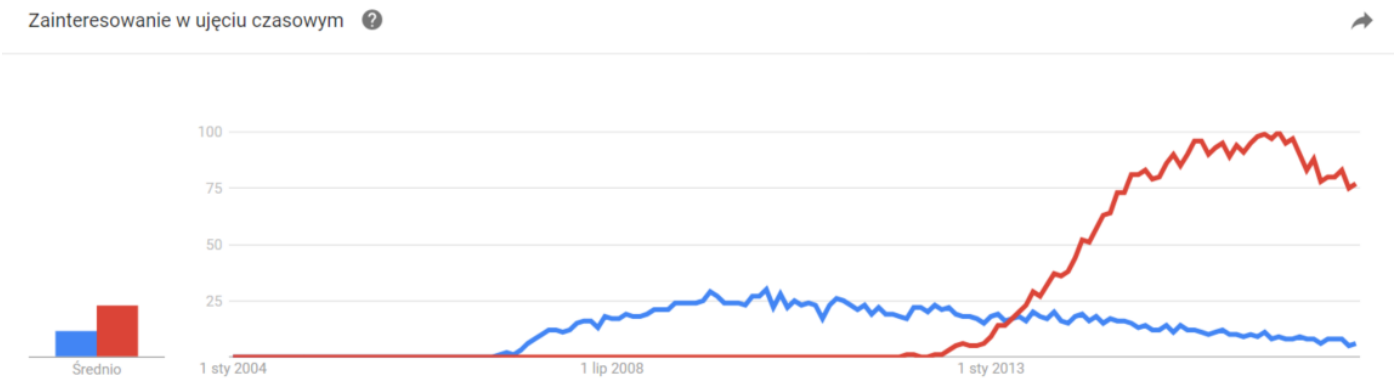
W celu zwiększenia efektywności pracy z serwisami, wprowadzono rozwiązania typu Single Page Application (SPA), co oznacza że po pierwszym załadowaniu strony odświeżane są tylko wybrane jej elementy [2]. Odpowiednio zaprojektowany i zaimplementowany interfejs wczytuje dane partiami, na bieżąco odświeżając treść, dzięki czemu następuje to dynamicznie i niemalże natychmiastowo [3]. Stworzenie tego typu aplikacji wymaga niestety ingerencji JavaScript w strukturę DOM (ang. Document Object Model) [4].

Dużym ułatwieniem przy tworzeniu aplikacji internetowych typu SPA jest zastosowanie różnego typu frameworków. Dodatkowym ich atutem jest usprawnienie implementacji aplikacji. Większość z nich oferuje gotowe

komponenty, dyrektywy, ułatwia zarządzanie stroną, czy przekazywanie danych poprzez modele [5].

Oczywiście jak w przypadku wielu innych technologii i rozwiązań, frameworki JavaScript dzielą się na darmowe oraz komercyjne. W artykule opisany został proces analizy porównawczej dwóch frameworków AngularJS oraz ExtJS. Pierwszy jest przedstawicielem narzędzia typu open-source, może być wykorzystywany komercyjnie bez żadnych opłat, natomiast drugi z nich jest płatną biblioteką.

Jak można zaobserwować na Rys.2, ExtJS (niebieski) jest również dużo starszym frameworkiem. W związku z faktem długiego i dynamicznego rozwoju istnieje kilka wersji biblioteki, w chwili obecnej Sencha [14] zaprezentowało jego szóste wydanie.



Rys. 2. Liczba wyszukań frameworków u od 2004 roku do września 2017 [6] kolor niebiski – Ext JS, czerwony - Angular

AngularJS (czerwony) jest dużo młodszym rozwiązaniem, jednak pierwsza jego wersja mimo dość ograniczonych możliwości bardzo szybko zyskała dużą popularność co skłoniło Google do stworzenia kolejnej, o wiele bardziej rozbudowanej odsłony. W tabeli 1 zestawiono podstawowe właściwości obu rozwiązań.

2. Frameworki

Podstawową różnicą między frameworkami, niezwiązaną bezpośrednio z pisaniem kodu jest licencja na warunkach której jest on dostępny. Udostępnione przez twórców narzędzia wspomagające pracę programistyczne lub możliwość wykorzystania aplikacji dla urządzeń mobilnych również ma duże znaczenie. Informacje te zostały zestawione w Tabeli 1.

Tabela 1. Zestawienie podstawowych informacji o frameworkach

	AngularJS	**ExtJS**
Producent	Google	Sencha
Licencja	Darmowy framework dostępnw licencji MIT	Komercyjny framework, którego cena zaczyna się od 4 475 dolarów (dla zespołu poniżej 5 osób). Dostępny jest również jako 30 dniowy Trial lub za darmo na warunkach licencji GPLv3
Dostępne narzędzia	Angular CLI, Batarang, Grunt, Yeoman, Bower	Sencha cmd, wtyczki dla IDE oraz edytorów kodu, Sencha Architect, Sencha Themer, Sencha Inspector, Sencha Stencils, Sencha Touch
Architektura	MVW(Model-View-Whatever), wspierane są popularne modele: MVC(Model-View-Controller) oraz MVVM(Model-View-ViewModel)	Modele MVC(Model-View-Controller) oraz MVVM(Model-View-ViewModel)
Mobilność	Moduły oraz dyrektywy są responsywne, dodatkowo są dostępne biblioteki angular-gestures oraz ngTouch	Wszystkie komponenty są responsywne, dodatkowo do tworzenia aplikacji na urządzenia mobilne udostępnione jest narzędzie Sencha Touch
Testy	Karma – stworzona specjalnie na użytek testów AngularJS	Sencha Test jest optymalnym narzędziem do testów, ale można je również przeprowadzić za pomocą Jasmine lub Mocha
Przeglądarki	Wspiera przeglądarki: Internet Explorer od wersji 8 do 11 oraz Microsoft Edge 12, Chrome 57-59, Firefox 45-52, Safari 6-11, Opera 43 i wyższe,	Wspiera najnowsze wersje Chrome (59) oraz Firefox (52), a także Microsoft Edge 13-14, Internet Explorer 9-11, Safari 7-10

3. Implementacja aplikacji

Oba projekty zostały zaimplementowane przy użyciu komercyjnego środowiska programistycznego jakim jest PhpStorm (w wersji 2017.1.4), dla systemu operacyjnego Windows 10. Obie aplikacje umożliwiają rejestrację wizyt przez salony usługowe takie jak zakłady fryzjerskie czy kosmetyczne. Jednak po wykonaniu odpowiednich prac programistycznych mogły by służyć do zarządzania dowolnymi wizytami. Każda aplikacja składa się z czterech widoków prezentujących kolejno:

- Wizyty,
- Klientów,
- Pracowników,
- Usługi,

Dla każdego z widoków możliwe są do wykonania akcje dodawania, edycji oraz usuwania elementów listy, przechowywanych w lokalnej pamięci (ang. local storage) aplikacji.

4. Kryteria analizy

4.1. Struktura aplikacji

Pierwszym kryterium jest struktura aplikacji webowej. Składa się ona z wielu różnego rodzaju komponentów, katalogów, plików zasobów statycznych oraz plików konfiguracyjnych, umieszczonych w odpowiedniej strukturze katalogowej.

W tym przypadku oba porównywane frameworki stosują architektoniczny model MVC (ang. Model-View-Controller), który oddziela warstwę logiczną, od kontroli działania strony oraz interfejsu użytkownika [7]. To w pewien analogiczny sposób narzuca obu rozwiązaniom podobną strukturę plików.

4.2. Komponenty GUI

Framework jest swojego rodzaju gotowym szkieletem, na którego bazie buduje się aplikację. Zawiera on gotowe struktury, dlatego też udostępniane komponenty są ważnym elementem każdego szkieletu programistycznego. To one w znacznym stopniu przyczyniają się do oszczędności czasu, a co za tym idzie również środków, przy implementacji aplikacji webowej [8].

Na tym etapie porównane zostały omówione gotowe komponenty udostępniane przez oba rozwiązania, a także trudności ich zaimplementowania.

4.3. Metryki kodu

Badanie tego kryterium oparte jest o rozmiar skryptu, czyli liczbę linii kodu. Parametr ten jest niezwykle istotny z punktu widzenia programisty ponieważ zwykle niesie ze sobą mniejszy jego wkład w implementację.

Porównaniu poddana została liczba linii kodu potrzebna do zaimplementowania wybranego komponentu.

W związku z zależnością liczby linii kodu, a rozmiarem plików, wskazane zostanie również porównanie wielkości samych frameworków oraz gotowych aplikacji.

4.4. Testy wydajnościowe

Aplikacje testowe zaimplementowane w obu analizowanych technologiach zostały przetestowane pod kątem szybkości ładowania stron w przeglądarce oraz zużycia pamięci przy pierwszym ładowaniu aplikacji.

Do testów zostały wykorzystane najpopularniejsze przeglądarki internetowe:

- Google Chrome wersja 62.0.3202.94,
- Mozilla Firefox wersja 48.0,
- Internet Explorer wersja 11.64.16299.0.

4.5. Dostępność dokumentacji

Dostępność dokumentacji jest kolejnym istotnym parametrem analizy. Porównano zarówno oficjalne dokumentacje dostępne na stronach producentów [9, 10], jak i nieoficjalne materiały dostępne w języku angielskim i polskim.

Duża dostępność materiałów oraz łatwość ich przyswajania w znaczącym stopniu przekłada się na naukę danego frameworka oraz możliwość znalezienia odpowiedzi na problem jedynie na podstawie dokumentacji. Istotne jest także jak dużą część materiałów należy przyswoić aby móc zacząć pracę z frameworkiem i stworzyć podstawową działającą aplikację.

4.6. Wsparcie

Kryterium równie istotnym co dokumentacja, jest wsparcie udzielane przez innych użytkowników omawianych rozwiązań. Zarówno na początku nauki jak i przy tworzeniu zaawansowanych aplikacji programiści napotykają różnego typu błędy i trudności, większość z nich nie jest uwzględniona w dokumentacjach. Istotne w takiej sytuacji jest istnienie możliwości uzyskania pomocy od osoby bardziej doświadczonej lub kogoś kto szukał już rozwiązania podobnego problemu.

5. Analiza porównawcza frameworków

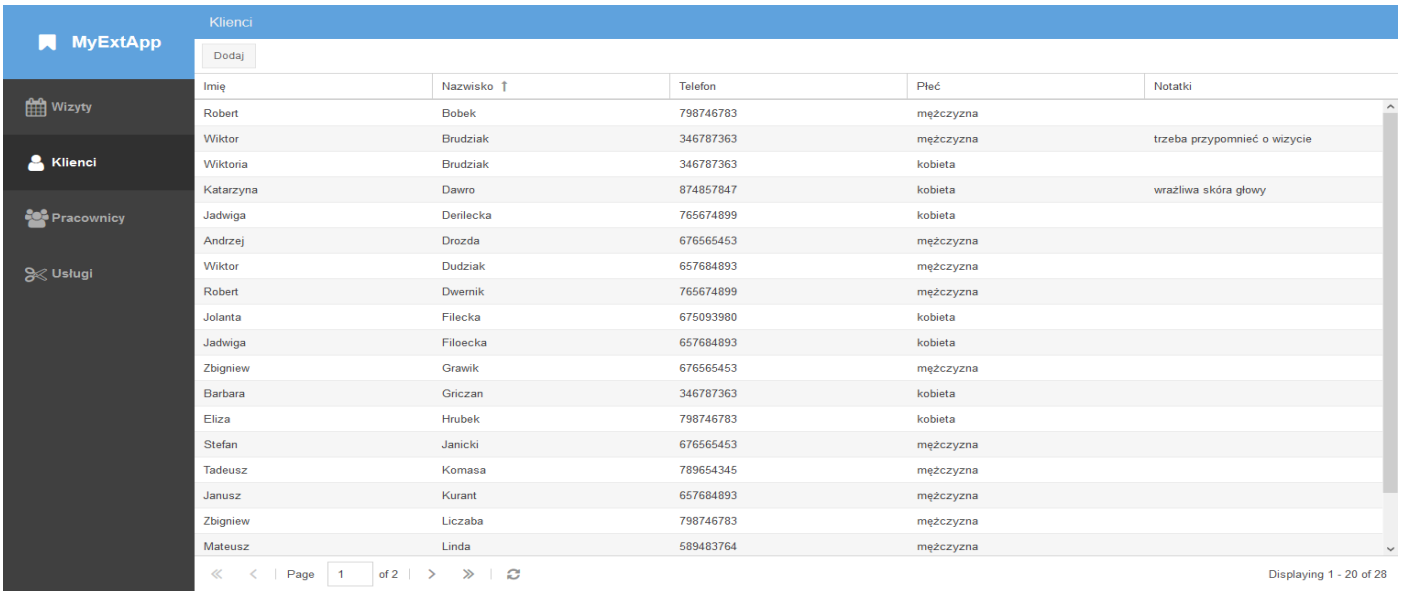
5.1. Struktura aplikacji

Dla aplikacji Ext zalecaną strukturą jest podział na pliki kontrolerów, modeli oraz widoków. Dopiero dla kilku plików wskazanych dla danego modułu w katalogach głównych tworzone są podfoldery. Z kolei w przypadku Angular istotniejsze jest rozdzielenie poszczególnych modułów a wewnątrz nich podział plików w zależności od ich przeznaczenia, warte podkreślenia jest także to, że każdy z komponentów znajduje się w oddzielnym folderze co wynika z konieczności zdefiniowania aż trzech plików dla każdego z nich.

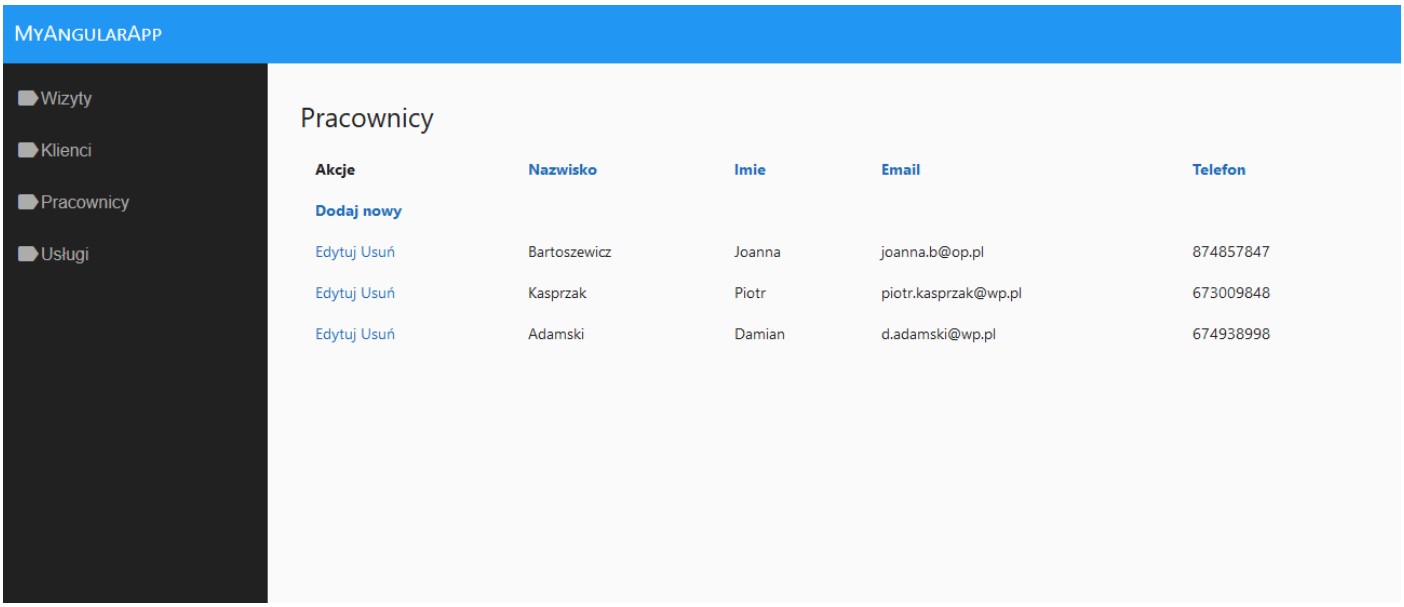
5.2. Komponenty GUI

Przy porównywaniu komponentów konieczne jest zaznaczenie że w aplikacji Angular każdy element należy zdefiniować w dwóch plikach, wskazać wykorzystywany komponent oraz jego strukturę w html, a oddzielnie jego konfigurację.

Najistotniejszymi użytymi w projekcie predefiniowanymi komponentami były te służące do prezentacji danych czyli grid w przypadku Ext JS oraz ng2-smart-table dla Angular. Sama konfiguracja dodanych elementów w przypadku obu frameworków wygląda bardzo podobnie i przypomina definiowanie obiektu. Dużą różnicą jest natomiast możliwość zdefiniowania w Ext JS paska nawigacyjnego i umieszczenia na nim przycisków, których naciśnięcie spowoduje wywołanie akcji, oraz dodanie menu kontekstowego, którego stworzenie jest już bardziej skomplikowane jednak nadal dostępne. W Angular akcje dostępne są pod postacią linków umieszczonych w tabeli prezentującej dane i powielane dla każdego z rekordów, nie przypominają w żaden sposób przycisku i konieczne jest oddzielne zdefiniowanie dla nich styli. Rozwiązanie oferowane przez pierwszy z frameworków jest dużo bardziej estetyczne i intuicyjne. Jeśli chodzi o definiowanie formularzy najbardziej dotkliwą niedogodnością jest konieczność powtórzenia definicji dla dodawania i edycji w aplikacji Angular. Pomijając tą wadę w obu frameworkach formularze otwierane są w oknach dialogowych, dają możliwość wykorzystania tych samych pól i podobnej ich konfiguracji. Z jednym wyjątkiem, ponownie na niekorzyść Angular, nie udostępnia on możliwości wskazania godziny, ani za pomocą odpowiedniej konfiguracji elementu wybierania daty, ani przy użyciu oddzielnego pola.



Rys. 3. Wygląd aplikacji zaimplementowanej przy użyciu Ext JS



Rys. 4. Wygląd aplikacji zaimplementowanej przy użyciu Angular

5.3. Metryki kodu

Jeśli chodzi o liczbę linii kodu potrzebnych dla zaimplementowania konkretnego widoku, analizie poddana została lista personelu. W przypadku Ext JS zdefiniowany został plik zawierający odwołanie i konfigurację predefiniowanego komponentu siatki i składa się on z 52 linii kodu. Z kolei w przypadku Angulara konieczna była definicja dwóch plików, w pierwszym osadzony został komponent i składa się ona jedynie z 6 linii, natomiast plik, w którym został on skonfigurowany, napełniony danymi oraz zostały w nim obsłużone akcje dostępne dla widoku zawiera aż 109 linii kodu. Jeśli chodzi zaś o rozmiarowe porównanie plików, definicja w Ext JS zajmuje 1,21 KB, a przy użyciu Angular 3,08 KB.

Rozmiary samych bibliotek również mają znaczenie dla programisty Angular zajmuje 229 MB, z kolei Ext JS 260 MB. Najistotniejszą jednak miarą jest rozmiar zbudowanej aplikacji. W przypadku Angular zajmuje ona 12 MB, Ext JS -16,9 MB. Aby porównanie było czytelniejsze zostało ono umieszczone w tabeli 2.

Tabela 2. Zestawienie metryk kodu

		Ext JS	Angular
Komponent siatki	linii kodu	53	115
	rozmiar plików	1,21 KB	3,08 KB
Biblioteka		260 MB	229 MB
Projekt		356 KB	230 MB
Zbudowana aplikacja		16,9 MB	12MB

5.4. Testy wydajnościowe

Dla większej czytelności wyników testów zostały one zestawione w postaci tabeli 3.

Tabela 3 Zestawienie różnic w czasie i ilości załadowanych danych dla poszczególnych przeglądarek

	Ext JS		Angular	
	czas transferu (s)	Przetransferowano (MB)	czas transferu (s)	Przetransferowano (MB)
Google Chrome	2,80	11,3	3,29	5,7
Mozilla Firefox	3,08	11,1	5,63	5,7
Internet Explorer	1,27	11,37	2,95	5,65

5.5. Dostępność dokumentacji

W przypadku obu frameworków dostępne są aktualne i wyczerpujące dokumentacje w języku angielskim [9, 10], pozwalające na utworzenie i uruchomienie prostej przykładowej aplikacji. W przypadku ExtJS rozpoczęcie pracy jest dodatkowo ułatwione poprzez możliwość wygenerowania aplikacji startowej.

Zarówno na temat AngularJS, jak i ExtJS powstaje bardzo wiele publikacji, artykułów, wpisów na popularnych stronach o programowaniu [20-25], a w szczególności tych dla web developerów. Oba frameworki opisane są również w wielu książkach oraz szkoleniach, jednak większość z tych dotyczących ExtJS nie posiada tłumaczeń lub odpowiedników w języku polskim. Reguła ta dotyczy wszelkiego rodzaju publikacji, może to sugerować, że w Polsce większą popularność uzyskał AngularJS.

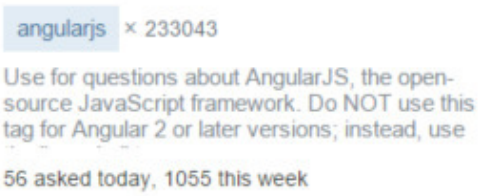
Również na najpopularniejszym zdalnym repozytorium jakim jest GitHub, AngularJS zajmuje wysokie miejsce w zestawieniu najczęściej wykorzystywanych frameworków JavaScript [11]. Łatwiej więc także znaleźć tam gotowe przykłady aplikacji innych użytkowników.

Oczywiście przykłady aplikacji napisanych za pomocą ExtJS również występują w repozytorium jednak jest ich o wiele mniej. Nie utrudnia to jednak zapoznania się z gotowymi aplikacjami, ponieważ w dokumentacji frameworka [10] dostępne są przykłady poszczególnych komponentów a także bardziej rozbudowanych aplikacji z możliwością sprawdzenia ich działania online oraz podglądem kodu.

5.6. Wsparcie

Jeśli chodzi o społeczność internetową na forach poświęconych programowaniu [12, 15 - 19], zdecydowanie więcej znajduje się wątków poświęconych AngularJS. Istnieje także większa szansa, że uda się uzyskać odpowiedź na pytanie zadane w tych tematach.

Różnica ta jest wyraźnie widoczna na wspomnianej już stronie StackOverflow. Możliwe jest na niej wyszukiwanie tematów po słowach kluczowych, a przy okazji prezentowana jest statystyka użycia danego klucza. W przypadku AngularJS liczba wykorzystania hasła jest dziesięciokrotnie większa (Rys. 3- 4.).



Rys. 3. Użycie klucza „angularjs" na stronie StackOverfolw [12]

Mniejsza aktywności użytkowników ExtJS na popularnych forach programistycznych, może być

spowodowana tym, że Sencha prowadzi forum [13], na którym twórcy frameworka odpowiadają na pytania, udzielają porad i pomagają w rozwiązaniu problemów krok po kroku. Programiści więc chętniej korzystają z tego forum niż z mniej specjalistycznych. Trudno jednak stwierdzić czy jest to lepsze rozwiązanie ponieważ konieczne jest dużo dłuższe czekanie na odpowiedź, a nie zawsze rozwiązuje ona problem pytającego.

Rys. 4. Użycie klucza „extjs" na stronie StackOverfolw [12]

6. Wnioski

Biorąc pod uwagę opracowane kryteria powstała tabela 4 podsumowująca framworki w ocenie punktowej. Punkty w skali od 1 do 10 zostały przydzielone subiektywnie na podstawie doświadczeń autorki w trakcie implementacji oraz analizy materiałów.

Tabela 4 Punktowa ocena frameworków

	Ext JS	Angular
Struktura aplikacji	10	6
Komponenty GUI	9	5
Metryki kodu	7	8
Testy wydajnościowe	8	8
Dostępność dokumentacji	6	8
Wsparcie	5	9
Sumaryczna ocena:	**45**	**44**

Na 60 możliwych punktów Ext JS otrzymał 45, a Angular 44, co daje bardzo zbliżony wynik. Jednak każdy z ich jest dobry z innych względów. Ext JS oferuje dużo szerszy zasób komponentów, gotowych motywów stylistycznych, a także jest dużo bardziej dopracowany i wymaga mniejszego nakładu pracy programisty aby aplikacja wyglądała dobrze. Nie jest natomiast najlepszym rozwiązaniem dla początkujących programistów, którzy chcieliby nauczyć się pierwszego frameworka. Dokumentacja jest dość dokładna i możliwe jest znalezienie wielu materiałów i szkoleń dla początkujących, jednak dużo trudniej uzyskać wsparcie w przypadku problemów. Z kolei Angular jest młodszym i dużo mniej rozbudowanym rozwiązaniem, wymaga dużo większego nakładu pracy zwłaszcza w odniesieniu do odpowiedniego wyglądu strony. Jednak jest lepszym rozwiązaniem dla początkujących ponieważ cieszy się aktualnie dużą popularnością, która stale rośnie, co skutkuje ogromną aktywnością społeczności na forach i przyczynia się do łatwiejszego rozwiązywania napotkanych problemów i zdobywania wiedzy.

Literatura

[1] https://insights.stackoverflow.com/survey/2017#overview [06.11.2017]

[2] M. Minović, S. Vesic, Single Page Applications: Trend or Future, Info, 2015

[3] C. Ravi, Staged information flow for JavaScript, Conference on Programming Language Design and Implementation, 2009

[4] M. A. Moyeen, G. G. M. Nawaz Ali i P. H. Joo Chong, An automatic layout faults detection technique in responsive web pages considering JavaScript defined dynamic layouts, Electrical Engineering and Information Communication Technology, 2016

[5] M. C. Enache, Web Application Frameworks, Dunarea de Jos University of Galati, 2015

[6] https://trends.google.com/trends/explore?q=extjs%20javascript, Angularjs%20javascript. [06.11.2017]

[7] L. Zhang, Y.-J. Yang i J.-H. Ni, The research and implement of power communication alarm management system based on MVC and ext JS, w International Conference on Information Management, Innovation Management and Industrial Engineering, 2012.

[8] C. Hiller: Choosing a JavaScript framework. Bleeding Edge Press, 2014,

[9] http://www.angularjs.org. [06.11.2017]

[10] http://docs.sencha.com/extjs/4.2.6/. [06.11.2017]

[11] https://github.com/showcases/front-end-javascript-frameworks?s=stars [06.11.2017]

[12] https://stackoverflow.com/tags [06.11.2017]

[13] https://www.sencha.com/forum/ [06.11.2017]

[14] https://docs.sencha.com/extjs/6.0.2/guides/whats_new/whats_new.html [06.11.2017]

[15] https://www.codeproject.com/ [20.11.2017]

[16] https://forum.pasja-informatyki.pl/ [20.11.2017]

[17] http://forum.codecall.net/ [20.11.2017]

[18] https://bytes.com/ [20.11.2017]

[19] https://4programmers.net/Forum [20.11.2017]

[20] https://toddmotto.com/categories/angular [20.11.2017]

[21] https://medium.mybridge.co/angular-2-0-top-10-articles-for-the-past-month-84aa098c9b10 [20.11.2017]

[22] https://blog.thoughtram.io/categories/angular/ [20.11.2017]

[23] https://hackernoon.com/the-rise-and-fall-of-ext-js-c9d727131991 [20.11.2017]

[24] https://www.markhamstra.com/extjs/2013/extjs-dream-nightmare-jquery/ [20.11.2017]

[25] https://www.sencha.com/blog/category/sencha-ext-js/ [20.11.2017]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 132-137
Wysłane: 2017-11-22
Przyjęte: 2017-11-28

Porównanie platform Wordpress Woocommerce z Magento Community Edition

Cezary Cichocki*
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule porównano działanie dwóch platform „open source" wspomagających e-handel: Wordpressa z najpopularniejszą wtyczką – Woocommerce oraz Magento z serii Community Edition. Przedstawiono wady i zalety obu systemów. Na podstawie przeprowadzonych badań pomiaru szybkości ładowania stron przy użyciu narzędzi online jak np. Google PageSpeed Insights oraz analizie popularności systemów w serwisach społecznościowych i forach internetowych, wyciągnięte zostały wnioski, stwierdzające, która z platform jest lepszym wyborem.

Słowa kluczowe: e-commerce; cms, wordpress woocommerce; magento

*Autor do korespondencji.

Adres e-mail: cez.cichocki@gmail.com

Comparison of Wordpress Woocommerce with Magento Community Edition

Cezary Cichocki*
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. This article compares two open source platforms supports e-commerce, such as: Wordpress with Woocommerce plugin and Magento Community Edition. The paper shows pros and cons of both systems. The main functionalities of them were tested and showed. Based on research - conducted measuring the speed of web page load by using online tools such as Google PageSpeed Insights and system popularity analysis based on social media and Internet forums, conclusion were made.

Keywords: e-commerce; cms; wordpress woocommerce; magento

*Corresponding author.

E-mail address: cez.cichocki@gmail.com

1. Wstęp

Coraz więcej osób ma dostęp do Internetu. Odnosząc się do ogólnodostępnych statystyk, od początku 2017 roku podłączenie do globalnej sieci internetowej ma ponad 50% ludzi na świecie. W Polsce jest to około 70%, co i tak nie jest najlepszym wynikiem w porównaniu do przodujących w statystykach Zjednoczonych Emiratów Arabskich, w których to dostęp do Internetu ma 99% ludzi, czy też Japonii, która znajduje się na drugiej pozycji z wynikiem równym 93% [11]. Coraz powszechniejszy dostęp do sieci internetowej spowodowany jest m.in. popularyzacją urządzeń mobilnych, które powoli wypierają z użytku komputery stacjonarne. To właśnie telefony komórkowe i tablety generują coraz większe natężenie ruchu w Internecie.

Wraz z łatwiejszym, popularniejszym i lepszym połączeniem z siecią www rozwija się handel elektroniczny tzw. e-commerce. Użytkownicy coraz chętniej wybierają tę metodę dokonywania transakcji z prostej przyczyny – ten sposób jest wygodniejszy od tradycyjnych zakupów, a także w większości przypadków tańszy.

W sieci jest mnóstwo sklepów online czy aukcji internetowych. Prym na świecie wiodą takie marki jak amazon.com oraz ebay.com. W Polsce największym portalem aukcyjnym jest Allegro.pl, ale oprócz tego serwisu jest wiele dużych sklepów oraz jeszcze więcej tych mniejszych.

Portal aukcyjny, czy też sklep internetowy łączy jedna rzecz – CMS (ang. Content Management System), czyli system zarządzania treścią. Dostępnych jest wiele platform e-commerce wspierających sprzedaż internetową. Wielkie marki i firmy korzystają z dedykowanych rozwiązań, czyli pisanych systemów specjalnie na zamówienie danej firmy. Reszta korzysta z płatnych platform, ale dostępnych dla każdego lub ogólnodostępnych darmowych systemów tzw. „open source".

Takich skryptów jest wiele, m.in. Shoper, Open Cart, PrestaShop. Autor artykułu skupia się na dwóch najpopularniejszych rozwiązaniach typu „open source" – Wordpress z zainstalowaną wtyczką Woocommerce oraz Magento z serii Community Edition. Przedstawione zostaną główne funkcjonalności opracowanych systemów na potrzeby prac badawczych. Z wyników przeprowadzonych badań wyciągnięte zostaną wnioski. Artykuł postara się odpowiedzieć, który z przedstawionych CMSów jest lepszym rozwiązaniem i dlaczego. Oprócz tego wyjaśnione zostaną pojęcia: E-Commerce i CMS oraz fenomenu sklepów internetowych.

2. E-Commerce i sklepy internetowe

2.1. E-Commerce

E-Commerce, czyli handel elektroniczny, rozumiany jest jako wszystkie czynności związane z zawieraniem transakcji handlowych, które realizowane są przy pomocy komunikacji

i technologii elektronicznej[3]. Najpopularniejszą formą handlu elektronicznego jest handel internetowy. GUS czyli Główny Urząd Statystyczny, określa e-commerce jako elektroniczny handel wykonywany przez Internet, które wykorzystują do tego działania protokół IP[10].

Do zalet e-commerce należy zaliczyć [3]:

- oszczędność czasu i pieniędzy (koszty związane z działalnością internetową są dużo mniejsze niż interes stacjonarny);
- interaktywność, czyli prezentacja produktu;
- możliwość realizacji promocji marketingowych w sieci;
- duży rynek internetowy;
- elastyczność.

Wadami handlu internetowego niewątpliwie są:

- brak bezpośredniego kontaktu ze sprzedającym;
- opóźnienia w dostawie;
- koszty wysyłki;
- ochrona danych osobowych (narażenie na ataki).

2.2. Sklepy internetowe

Najpopularniejszym segmentem e-commerce są sklepy internetowe. Narzędzie to jest najczęściej stosowane przez przedsiębiorców, które otwiera bardzo szeroki dostęp do potencjalnych klientów. Na pewno też jest to prostsze i tańsze rozwiązanie niż chociażby prowadzenie sklepu bądź sieci sklepów stacjonarnych. Oprócz braku barier związanych z np. lokalizacją, koszty związane z prowadzeniem sklepu internetowego są znacznie mniejsze. Prowadząc sklep internetowy nie ma potrzeby wynajmowania lub posiadania specjalnego pomieszczenia oraz liczbę zatrudnionych pracowników można ograniczyć do zera. Wiele mniejszych sklepów online prowadzonych jest z mieszkań właścicieli.

Wydatki związane z prowadzeniem sklepu internetowego zależą w głównej mierze od umiejętności i doświadczenia właściciela oraz od modelu witryny i sposoby zarządzania. Przyszły właściciel sklepu internetowego ma do wyboru kilka opcji – np.:

- wykupując usługi abonamentowe na platformach sklepowych, czyli płacona jest miesięczna kwota i w zamian otrzymywany jest sklep gotowy do prowadzenia;
- zlecenie wykonanie witryny firmie informatycznej;
- przygotowanie sklepu na platformie ‘open source’.

3. Systemy CMS

Prowadzenie i zarządzanie stroną internetową nie jest taką prostą sprawą, a prowadzenie jej bez dobrego systemu zarządzania treścią CMS jest praktycznie niemożliwe dla użytkownika bez wiedzy technicznej.

Rozwinięciu skrótu CMS, oznacza Content Management System, czyli system zarządzania treścią. Jest to oprogramowanie, które umożliwia tworzenie stron WWW bez większej znajomości programowania. Strony oparte o systemy CMS posiadają własny panel administracyjny do edytowania i dodawania treści na witrynie internetowej[1]. Platformy CMS mogą być zarówno płatne (z dodatkowymi funkcjonalnościami) jak i darmowe, typu „open source” – przeważnie prostsze, z mniejszą liczbą funkcji. Nie jest to jednak reguła i są dostępne w sieci rozwiązania darmowe z bardzo rozbudowanymi opcjami i wieloma dodatkowymi rozszerzeniami. Wykorzystanie systemu CMS jest konieczne, jeżeli użytkownik planuje prowadzić sklep internetowy[2].

Jak wspomniano wyżej, są różne typy systemów CMS. Można podzielić je na 3 kategorie[1]:

- CMS „Open Source” – czyli z otwartym kodem źródłowym. Kod takiej platformy jest dostępny publicznie i wszyscy użytkownicy, którzy natrafili na ten system w sieci mogą bezpłatnie z niego korzystać i modyfikować. Na takich platformach łatwiej tworzy się samodzielnie nowe szablony, wtyczki i inne dodatki. Jeżeli w kodzie oprogramowania wystąpią błędy, szybko są one wychwytywane przez internautów korzystających z systemu. Mimo że są to darmowe systemy, nie jest to opcja dla każdego. By wykonać sklep internetowy oparty o CMS, użytkownik musi posiadać chociaż średnią wiedzę informatyczną by z sukcesem i bez problemowo zarządzać witryną. Jeżeli użytkownik nie ma takich zdolności, zawsze można zlecić wykonanie jakiej strony profesjonaliście, co oczywiście wiążę się z kosztami[4].
- CMS płatny, z dodatkowymi funkcjonalnościami. Są to często rozszerzenia darmowych wersji. Wykupienie wersji płatnej sytemu może okazać się dobrym rozwiązaniem dla osób bez wiedzy technicznej, która sama nie wprowadzi zaawansowanych modyfikacji systemowych.
- System CMS hostowany – do kodu źródłowego takiej platformy użytkownik nie ma dostępu. Użytkownik płaci comiesięczny abonament za utrzymanie i konserwacje sklepu internetowego na ich platformie, a on sam nie ma możliwości modyfikowania kodu i dodawania swoich dodatków.

Do ogólnych możliwości sklepów opracowanych na systemach CMS, niezależnie od jego kategorii można zaliczyć:

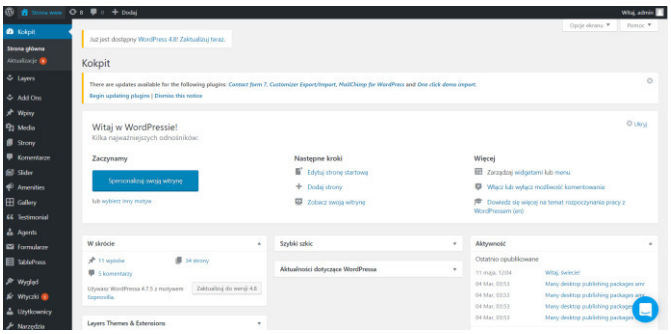
- tworzenie podstron www;
- edycja podstron;
- możliwość rejestracji użytkowników;
- zarządzanie kontami użytkowników;
- dodawanie zdjęć i różnego rodzaju grafiki;
- dodawanie ofert sprzedaży;
- wprowadzanie zmian w ofercie;
- czaty, fora, możliwość komentowania
- rozbudowane formularze kontaktowe czy zamówieniowe;
- możliwość zapisywania maili do newsletterów.

3.1. Wordpress Woocommerce

Wordpress jest obecnie najpopularniejszym systemem zarządzania treścią. Ponad 29% wszystkich dostępnych stron internetowych na świecie zasila właśnie Wordpress i około 10% sklepów internetowych korzysta z rozszerzenia Wordpressa – Woocommerce[16]. Z początku system ten był używany głównie jako narzędzie do tworzenia blogów, jednak z biegiem czasu, gdy powstawały nowe wtyczki i inne rozszerzenia, (m.in. omawiany w tym artykule

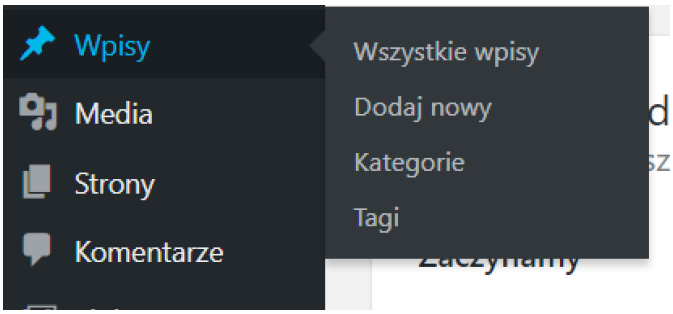
Woocommerce) platforma ta stała się pełnoprawnym narzędziem do tworzenia stron i sklepów internetowych[5].

Wordpress ma prawdopodobnie najbardziej intuicyjny interfejs panelu administratora (główny kokpit przedstawiony na rys. 1), który dodatkowo jest również responsywny, czyli dopasowuje się do rozmiarów ekranu, na którym jest wyświetlany.



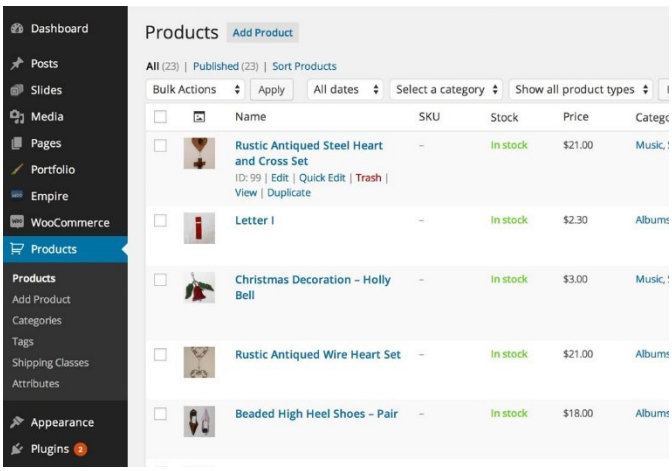
Rys. 1. Panel administracyjny Wordpress

Wszystkie opcje w menu głównym pogrupowane są według ich przeznaczenia. Przykładowo najeżdżając na grupę w menu „Wpisy" – rozwinięte zostaną wszystkie możliwości odnośnie danej grupy.



Rys. 2. Menu w panelu administracyjnym Wordpress

Po doinstalowaniu omawianej wtyczki do sprzedaży internetowej – Woocommerce, opcje związane z tą wtyczką znajdą się w menu głównym po lewej stronie (Rys. 3).



Rys. 3. Woocommerce – Panel Admina

Wtyczka Woocommerce doinstalowana do Wordpressa jest bardzo łatwa w konfiguracji. Po instalacji sama utworzy odpowiednie podstrony (np. koszyk, moje konto, moje zakupy itp.). Woocommerce sam w sobie posiada podstawowe funkcjonalności sprzedaży, konfigurację metod płatności, podatków, kosztów i sposobów wysyłki [7]. Dostępnych jest także wiele wtyczek (płatnych i darmowych) rozwijających funkcje platformy.

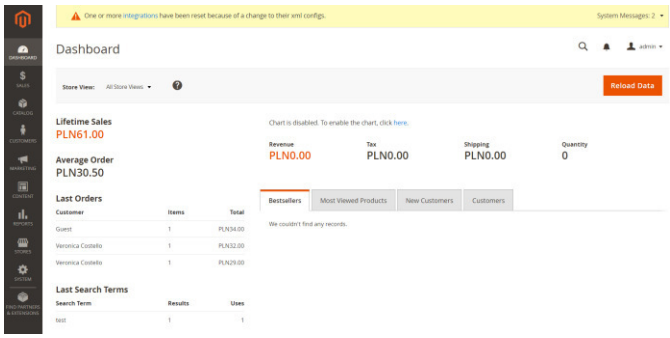
Dodatkowo, cała wielka społeczność Wordpressa wspiera także działania Woocommerce, dzięki temu jeżeli użytkownik ma jakieś pytanie lub problem do rozwiązania to szybko znajdzie odpowiedź. Wprowadzanie większych zmian do systemu wymaga znajomości programistycznych, ale te drobniejsze modyfikacje administrator może wykonywać sam, bez pomocy programisty.

Wykorzystanie Wordpressa z wtyczką Woocommerce polecane jest przy zakładaniu małego i średniego sklepu internetowego. Do utrzymania takiego sklepu ze średnią ilością klientów wystarczy podstawowy serwer udostępniany przez firmy hostingowe.

3.2. Magento Community Editon

Drugim z systemów opisywanych w artykule jest Magento (w wersji Community). Jest to niewątpliwie bardzo popularna i jedna z najbardziej rozbudowanych platform do tworzenia i prowadzenia sklepu internetowego. Platforma ta wykonana została przez firmę Varien (obecnie grupa eBay), która wywodzi się z Kalifornii. System ten oparty jest na języku skryptowym PHP z frameworkiem Zend[6].

Ten system idealnie nadaje się dla osób, które chcą utworzyć duży sklep internetowy z wieloma produktami. Magento posiada mnóstwo funkcjonalności, które ułatwiają zarządzanie zamówieniami, ofertowanym asortymentem, działaniami marketingowymi, bazą klientów itp., m.in. rozbudowany system do analiz, zarządzania produktami, raportowania, wersje mobilne czy też systemy płatności[9]. Jedną z ciekawszych funkcjonalności tego systemu jest możliwość zarządzania wieloma sklepami poprzez jeden panel administracyjny (widok panelu przedstawiono na rysunku 4) .



Rys. 4. Panel administracyjny Magento 2.0.

Tak jak i w Wordpressie Woocommerce, w Magento oprócz podstawowych funkcjonalności (których i tak jest bardzo dużo) można doinstalować dodatkowe, zewnętrzne moduły. Duża część z tych modułów jest pisana przez profesjonalistów, co za tym idzie większa część z nich jest często płatna.

Niestety wielkość tego systemu i możliwości jakie ze sobą niesie w związku z modułowością i rozszerzalnością skryptu niesie za sobą większe wymagania sprzętowe i konieczność obeznania w językach programowania (PHP)

[18]. Do obsługi dużego sklepu internetowego na systemie Magento potrzebna jest naprawdę wydajna maszyna serwerowa, gdyż nie każda firma hostingowa radzi sobie z wymaganiami tej platformy[8].

Na nauczenie się obsługi samego panelu administracyjnego trzeba poświęcić dużo więcej czasu, gdyż nie jest on tak prosty w obsłudze, jak chociażby omawiany wcześniej Woocommerce. Niewątpliwym plusem jest tutaj duża społeczność zgromadzona wokół tej platformy, co na pewno ułatwi naukę obsługi Magento.

Jedną z największych zalet platformy Magento jest jej bezpieczeństwo. Kod został napisany w bardzo przemyślany sposób, nawet sam link do panelu administracyjnego, który jest definiowany podczas instalacji systemu, może utrudnić nieautoryzowany dostęp sklepu od strony administracyjnej.

4. Przeprowadzenie badań

Oba systemy zostały przetestowane pod kątem szybkości ładowania podstron i optymalizacji dzięki dostępnym narzędziom online takim jak np. Google Page Insights[14]. Przy wyborze odpowiedniego systemu CMS popularność jest jednym z ważniejszych czynników. Im platformą zajmuje się więcej ludzi, tym prościej jest znaleźć rozwiązanie błędu, który może wystąpić podczas zarządzania czy konfiguracji sklepu. Dlatego też porównano funkcjonujące grupy i strony użytkowników zgromadzonych na portalach społecznościowych (Facebook i Twitter) oraz forach tematycznych wokół omawianych platform pod kątem wielkości i aktywności w sieci. Bezpośrednie zestawienie rezultatów pozwoli wyłonić zwycięzcę porównania.

4.1. Szybkość ładowania systemów WordPress i Magento

Przy wyborze odpowiedniej platformy e-commerce ważną kwestią jest jej wydajność i szybkość działania. Opóźnienia i wolne ładowanie strony internetowej wpływa negatywnie na optymalizację pod kątem SEO (ang. Search Engine Optimization) – szybkość ładowania strony internetowej wpływa na jej pozycję w rankingu wyszukiwania Google[12]. Według badań przeprowadzonych w 2016 roku przez Google, strony internetowe, które ładują się dłużej niż 5 sekund mogą tracić do 40% konwersji [13].

Na czas ładowania aplikacji internetowych ma wpływ wiele czynników, m.in. serwer hostujący i jego lokalizacja, łącze i przeglądarka internetowa osoby wchodzącej na stronę www, a także optymalizacja wielkości treści i elementów strony i napisanego kodu systemu. W przeprowadzonych testach WordPressa i Magento wykorzystano domyślne szablony graficzne, bez dodatkowych rozszerzeń i wtyczek. W celach testowych użyto serwer hostujący firmy webd.pl, który spełnił wymagania obu systemów:

- System operacyjny Linux;
- 50gb miejsca dyskowego;
- 400 gb miesięcznego transferu danych;
- PHP5/PHP7;
- MySQL w wersji 5;

Testy szybkości ładowania zostały przeprowadzone w oparciu o najpopularniejsze narzędzia online tego typu:

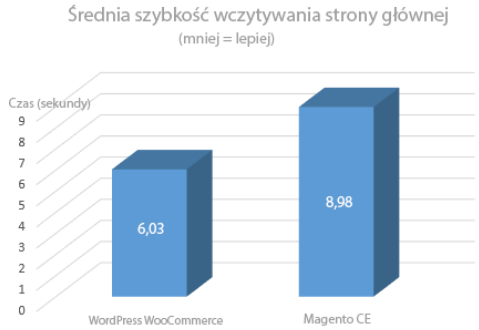
- GTMetrix[21];
- PageScroring[22];
- Tools.PingDom[23];
- WebPageTest[24];
- DotCom Tools[25].

Wszystkie z wymienionych wyżej narzędzi analizują testowaną stronę www przez serwery na całym świecie, a symulacja załadowania strony jest ustawiona na tzw. „pierwsze wejście", czyli w sposób taki gdy użytkownik wchodzi na daną witrynę internetową po raz pierwszy bez zapisanych ciasteczek (ang. Cookies) i pamięci podręcznej w przeglądarce. W ten sposób wykonane testy są bardziej miarodajne.

Testy wykonano na stronach głównych i podstronach produktu obu sklepów postawionych na omawianych platformach. Wyniki cząstkowe przeprowadzonych badań umieszczono w tabelach 1 i 2, a średnią wszystkich rezultatów pokazano na rysunkach 5 i 6.

Tabela 1. Wyniki testów wczytywania strony głównej

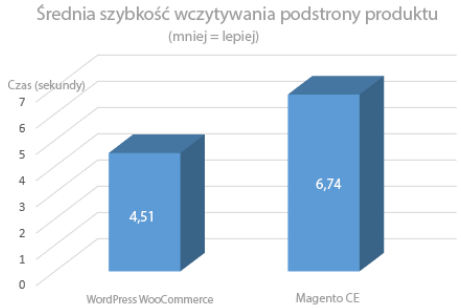
	WordPress WooCommerce [s]	Magento CE [s]
GTMetrix.com	2,90	9,00
PageScoring.com	8,86	4,61
Tools.pingdom.com	2,22	4,92
WebPageTest.org	3,78	7,56
Dotcom-tools.com	4,80	7,60



Rys. 5. Średnia szybkość wczytywania strony głównej

Tabela 2. Wyniki testów wczytywania podstrony produktowej.

	WordPress WooCommerce [s]	Magento CE [s]
GTMetrix.com	3.50	9.50
PageScoring.com	11.66	9.95
Tools.pingdom.com	4.39	7.22
WebPageTest.org	3.79	9.03
Dotcom-tools.com	6.80	9.20



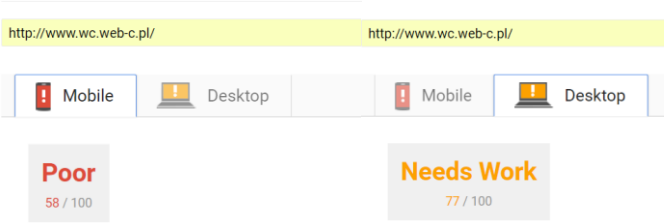
Rys. 6. Średnia szybkość wczytywania podstrony produktu

Korzystając z narzędzia Google Page Insights[14] sprawdzono także wydajność oraz poziom optymalizacji obu aplikacji pod względem objętości plików graficznych i kodu stron. Wyniki PageSpeed Insights mieszczą się w zakresie od 0 do 100 pkt. Wyższy wynik punktowy oznacza lepszą optymalizacje i wydajność aplikacji www. Sklep stworzony na potrzeby artykułu obsługiwany przez WordPressa z wtyczką WooCommerce osiąga następujące wyniki:

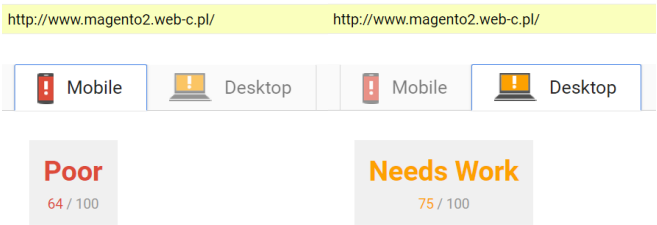
- 58/100 – na urządzeniach mobilnych;
- 77/100 – na urządzeniach desktopowych.

Natomiast strona działająca na Magento 2.0:

- 64/100 – na urządzeniach mobilnych;
- 75/100 – na urządzeniach desktopowych.

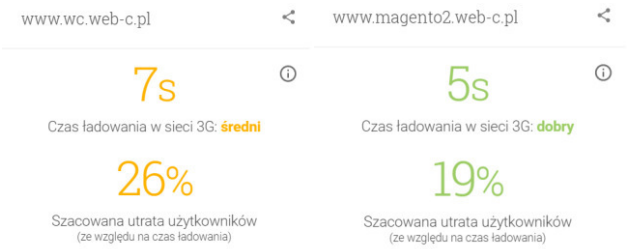


Rys. 7. Wyniki WordPress WooCommerce – PageSpeed Insights



Rys. 8. Wyniki Magento 2.0 – PageSpeed Insights

Aby sprawdzić dokładniej optymalizacje obu sklepów działających na urządzeniach mobilnych przeprowadzony został jeszcze jeden test, przy użyciu kolejnego narzędzia firmy Google – „TestMySite with Google"[15]. Określa on czas potrzebny na pojawienie się zawartości strony w przeglądarce internetowej Google Chrome na urządzeniu mobilnym Moto G4 w sieci 3G. Oprócz tego „TestMySite with Google" określa także szacowaną utratę użytkowników ze względu na czas ładowania strony. Podsumowanie z przeprowadzonych testów przedstawiono na rysunku 9.



Rys. 9. Szybkość ładowania sklepów na urządzeniach mobilnych przy wykorzystaniu sieci 3G

4.2. Społeczności platform WordPress i Magento

Kolejnymi zaletami omawianych systemów są społeczności, które gromadzą się wokół tych platform. Dzięki temu są one dynamicznie rozwijane, powstają nowe funkcjonalności, wtyczki oraz szablony. Im większa społeczność zajmująca się tymi CMSami tym łatwiej zwykłemu użytkownikowi znaleźć odpowiedzi na nurtujące pytania i rozwiązania problemów dotyczących Wordpressa i Magento.

Niniejszy artykuł przedstawia i porównuje aktualne statystyki profili społecznościowych (Twitter, Facebook), grup dyskusyjnych oraz for tematycznych, które zrzeszają sympatyków porównywanych platform webowych.

Profile i grupy na portalach społecznościowych zostały sprawdzone pod kątem polubieni/subskrypcji, a fora tematyczne, na których użytkownicy dzielą się wiedzą techniczną – pod względem ilości zarejestrowanych użytkowników oraz napisanych wiadomości.

Tabela 3 ukazuje przybliżoną ilość polubień oficjalnych profili WordPressa[19][20] i Magento[17][18] (listopad 2017r.).

Tabela 3. Statystyki oficjalnych profili WordPressa i Magento na Facebooku i Twitterze.

	WordPress	Magento
Facebook	1 139 323 polubień	32 006 polubień
Twitter	612 000 obserwujących	74 600 obserwujących

Powyższe wyniki, potwierdzają, że system WordPress jest popularniejszym CMSem. Jego profile społecznościowe są kilkanaście, a nawet kilkadziesiąt razy większe niż profile Magento. Nie oznacza to jednak, że Magento nie ma swoich zwolenników, porównując statystyki grup i for internetowych, wyższość Wordpressa nie jest już tak widoczna.

<u>WordPress / WordPress WooCommerce:</u>

- grupa tematyczna WordPress PL na Facebook.com – 14 546 członków;
- profil WordPress.com na Facebook.com – 349 349 polubień;
- grupa WooCommerce Help & Share na Facebook.com – 18 737 członków;
- forums.wordpress.com – 2 441 583 napisanych postów, brak statystyk dotyczących liczby użytkowników;
- wordpress.org.pl – 3144 użytkowników, 8783 wiadomości;
- zenwordpress.pl – 11 163 użytkowników, 38 144 postów.

<u>Magento:</u>

- Magento Experts na Facebook.com – 6460 członków;
- MeetMagento Polska na Facebook.com – 1505 polubień;
- forum community.magento.com – 276326 użytkowników, 71 363 napisanych postów;
- magentoexpertforum.com – 11 162 użytkowników, 28 213 napisanych postów;
- magentoforum.pl – 28 289 użytkowników, 10 744 napisanych wiadomości;

Ludzie związani z Wordpressem lub Magento, nie gromadzą się już tylko w Internecie. Organizują także różnego rodzaju regionalne spotkania sympatyków danych platform, a także uczestniczą w dużych konferencjach w Polsce oraz na całym świecie. WordPress ma w Polsce

coroczną imprezę pt. „WordCamp”, a developerzy i wielbiciele Magento spotykają się na konferencjach „MeetMagento”. Imprezy te gromadzą nawet po kilkaset osób.

5. Wnioski

Porównane w artykule platformy CMS niewątpliwie różnią się do siebie. Handel internetowy rozwija się z każdym dniem coraz prężniej, a wskazanie jednoznacznego zwycięzcy jest bardzo trudne.

Zapoznając się z rezultatami zebranych danych można odnieść wrażenie, że słusznym i jedynym zwycięzcą jest Wordpress z wtyczką Woocommece., jednak Magento góruje w tak ważnych aspektach jak np. bezpieczeństwo systemu i optymalizacja.

Obie platformy są na licencji „open source” – czyli można korzystać z nich bez opłat i modyfikować ich kod, ale to WordPress WooCommerce z pewnością jest systemem tańszym do wdrożenia, m.in. dlatego, że na Wordpressa istnieje dużo więcej darmowych rozszerzeń i szablonów oraz jest prostszy w konfiguracji, za czym idzie więcej firm czy osób prywatnych może przyjąć takie zlecenie. Do wdrożenia Magento potrzeba niestety dużo więcej wiedzy technicznej i nie każdy może sobie poradzić z instalacją i konfiguracją tego systemu. Jednak niskie koszty początkowe Wordpressa nie oznaczają, że w szerszej perspektywie utrzymanie sklepu na tym systemie będzie tańsze gdyż Magento jest znacznie stabilniejsze – mniej błędów na stronie oznacza mniej pracy dla deweloperów.

Podsumowując, jeśli użytkownik chce utworzyć mały sklep, ze stałą liczbą produktów i niewielką lub średnią ilością klientów, może to spokojnie zrobić wykorzystując Wordpressa z zainstalowaną wtyczką Woocommerce. Jeżeli jednak w planach jest otwarcie sklepu, który będzie musiał obsłużyć w przyszłości dużą liczbę klientów, a w bazie danych będzie znajdowała się pokaźna liczba produktów, warto jest na początku zainwestować i wdrożyć sklep na platformie Magento.

Literatura

[1] Barker D., Web Content Management – Systems, Features, and Best Practices, wydawnictwo O’Reilly 2016.

[2] Barsan G., Oancea R., Considerations on e-commerce platforms, 2014.

[3] Cichoń M., Cisek M. i inni, Biblia e-biznesu, 2013.

[4] Mehta N. Choosing an Open Source CMS: Beginner's Guide, 2009.

[5] Rauland P., Wordpress Cookbook, 2015.

[6] MacGregor A., Magento PHP Developer’s Guide, Packt Publishing, 2013.

[7] Ravensbergen R. , Building E-Commerce Solutions with WooCommerce - Second Edition, Packt Publishing, 2015.

[8] Williams B., Williams C., Learning Magento 2 Administration – Maximize the power of Magento 2 to improve your e-commerce business, Pack Publishing, 2016.

[9] Williams B., Mastering Magento, Packt Publishing, 2012.

[10] http://stat.gov.pl/metainformacje/slownik-pojec/pojecia-stosowane-w-statystyce-publicznej/1778,pojecie.html [20.10.2017]

[11] www.wearesocial.com/special-reports/digital-in-2017-global-overview [20.10.2017]

[12] http://www.adequate.pl/web-analytics/czas-ladowania-strony-wazniejszy-niz-myslisz [20.10.2017]

[13] https://webmasters.googleblog.com/2010/04/using-site-speed-in-web-search-ranking.html [23.10.2017]

[14] https://developers.google.com/speed/pagespeed/insights/ [14.11.2017]

[15] https://testmysite.withgoogle.com/intl/pl-pl [14.11.2017]

[16] https://trends.builtwith.com/shop [14.11.2017]

[17] https://www.facebook.com/magento [14.11.2017]

[18] https://twitter.com/magento [14.11.2017]

[19] https://www.facebook.com/WordPress/ [16.11.2017]

[20] https://twitter.com/wordpress [16.11.2017]

[21] https://gtmetrix.com [18.11.2017]

[22] http://pagescoring.com [18.11.2017]

[23] http://tools.pingdom.com [18.11.2017]

[24] https://webpagetest.org [18.11.2017]

[25] https://dotcom-tools.com/website-speed-test.aspx [18.11.2017]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 138-141
Wysłane: 2017-11-25
Przyjęte: 2017-11-28

Analiza prędkości wykonywania zapytań w wybranych bazach nie SQL-owych

Wojciech Bolesta*

Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł naukowy dotyczy porównania prędkości zapytań dwóch baz danych NoSQL. Opisywanymi bazami danych będą MongoDB oraz CouchDB. W pracy przedstawione zostaną porównania prędkości takich zapytań jak dodawanie danych do bazy, edycja danych bazy danych, usuwania danych z bazy, a także wyszukiwanie danych w bazie danych. Przedstawione zostanie również ogólne porównanie baz, z odpowiedzią na pytanie, która z badanych baz NoSQL jest szybsza.

Słowa kluczowe: NoSQL; MongoDB; CouchDB

*Autor do korespondencji.

Adres e-mail: wojtek.bolesta@wp.pl

Analysis of query execution speed in the selected NoSQL databases

Wojciech Bolesta*

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

Abstract. The scientific article deals with a comparison of the query speed of two NoSQL databases. Described databases will be MongoDB and CouchDB. The work presents speed comparisons of such queries as adding data to the database, editing database data, deleting data from the database, and searching data in the database. Also a general comparison of bases will be presented, with the answer to the question of which of the tested NoSQL databases is faster.

Keywords: NoSQL; MongoDB; CouchDB

*Corresponding author.

E-mail address: wojtek.bolesta@wp.pl

1. Wstęp

W poniższym artykule naukowym przedstawione zostaną wyniki badań dotyczących porównania prędkości wykonywania zapytań uzyskanych z dwóch nierelacyjnych bazach MongoDB oraz CouchDB. Do badania baz danych użyta została stworzona na potrzeby badań baza danych, do której następnie dodane zostały w procesie dodatkowe pozycje w ustalonym schemacie tak w MongoDB jak i CouchDB. Następnie w celu porównania, użyte zostały logi obu programów, dzięki którym uzyskane zostały wyniki prędkości wykonania operacji dodania do baz tych samych informacji za pomocą języka NoSQL[8]. W trakcie wykonywania badań wykonane zostały takie zapytania jak dodawanie do bazy danych, usuwanie danych z bazy, edycja wstawionych wcześniej danych, wyszukanie kilku zestawów danych, a także porównanie prędkości pobierania danych z różnych baz.[10,6,5]

Informacje otrzymane z badań umożliwiają określenie bazy danych, której czas wykonywania zapytań jest krótszy. Różnice w pojedynczych przypadkach, jak można założyć, mogą się okazać nieistotne dla użytków indywidualnych, lecz w przypadku dużych firm, które przeprowadzają operacje na dużej ilości danych, takie różnice mogą jednak okazywać się o wiele większe, przez co wybieranie określonego środowiska jest bardzo istotnym krokiem podczas prowadzenia pracy na danych tekstowych. Bazy NoSQL posiadają o wiele mniejszy możliwy zakres użytkowania, ale w porównaniu do baz SQL można się spodziewać iż w tych wąskich dziedzinach, w których operują, bazy NoSQL okazują niebywałą wydajność[1,10,2]. Wszystkie badania, tak w MongoDB jak i CouchDB przeprowadzone zostały na systemie operacyjnym Windows 7 Home Premium, na tym samym stanowisku testowym.

Cel

Celem poniższego artykułu naukowego jest przedstawienie uzyskanych wyników z analizy prędkości wykonywanych zapytań w wybranych bazach NoSQL, MongoDB oraz CouchDB. W oparciu o posiadane wyniki pomiaru prędkości zapytań o tym samym działaniu w wybranych bazach, dokonana zostanie następnie analiza porównawcza pomiędzy badanymi bazami MongoDB oraz CouchDB.

Teza

Za główne założenie tezy, przygotowanej na potrzeby artykułu naukowego przyjęto fakt iż MongoDB jest bazą wydajniejszą względem czasu od drugiej badanej bazy NoSQL CouchDB.

2. MongoDB

MongoDB jest jednym z najpopularniejszych nierelacyjnych systemów zarządzania bazami danych, napisanym w języku C++. MongoDB cechuje się dużą skalowalnością, wydajnością oraz nie posiada zdefiniowanej struktury obsługi danych. W MongoDB dane przechowywane

są jako dokument JSON, przez co tworzonym aplikacjom umożliwione jest naturalniejsze przetwarzanie danych, przy zachowaniu możliwości tworzenia hierarchii oraz indeksowania.[1,2,3,5]

Aktualnie MongoDB posiada możliwość przeprowadzania operacji na dokumentach, a dzięki korzystaniu z MongoDB Atlas, można również obserwować wydajność bazy danych, a sam program w sposób automatyczny zabezpiecza bazę danych, na której użytkownik przeprowadza prace, a także w porównaniu do desktopowej wersji MongoDB, posiada możliwość odzyskiwania danych dzięki możliwość tworzenia backupów.[8,4]

3. CouchDB

CouchDB jest nowym rodzajem baz danych. Podobnie jak MongoDB nie jest to baza relacyjna, ani obiektowa. Przechowuje ona dokumenty. W CouchDB dokument jest czymś w rodzaju występującej w Java kolekcji map, czyli posiada klucze i wartości[7]. Kluczem w CouchDB zawsze jest String, który jest unikalny w ramach pojedynczego dokumentu. Ciekawą i wartą wspomnienia cechą CouchDB jest fakt, iż dokumenty te są dostępne przez RESTowy interfejs oraz protokół http, zapisane w JSONie, znanym z JavaScripta formacie zapisu złożonych danych, który w przypadku badań przeprowadzanych na potrzeby artykułu naukowego, jest formatem zapisu bazy danych, na której przeprowadzane są badania. Językiem jaki został użyty do napisania CouchDB jest język Erlang. Autor wybrał ten język z powodu jego kompatybilności z pisaniem aplikacji wielowątkowych, co w efekcie sprawiło iż CouchDB jest dobrze skalowalny na wiele procesów/rdzeni.[8,10,9,6]

4. Przykłady

Poniższy Przykład obrazuje schemat jednego przygotowanego zestawu danych z bazy użytej na potrzeby wykonania badań.

Przykład 1. Przykładowy zestaw danych

```
{
   "fields": {
      "edited": "2014-12-20T21:17:56.891Z",
      "name": "Luke Skywalker",
      "created": "2014-12-09T13:50:51.644Z",
      "gender": "male",
      "skin_color": "fair",
      "hair_color": "blond",
      "height": "172",
      "eye_color": "blue",
      "mass": "77",
      "homeworld": 1,
      "birth_year": "19BBY"
   },
   "model": "resources.people",
   "pk": 1
```

Przykład 2. Wprowadzenie danych – MongoDB

```
db.collection.insertOne(
    {
      "fields": {
          "edited": "2014-12-20T21:17:56.891Z",
          "name": "Luke Skywalker",
          "created": "2014-12-09T13:50:51.644Z",
          "gender": "male",
          "skin_color": "fair",
          "hair_color": "blond",
          "height": "172",
          "eye_color": "blue",
          "mass": "77",
          "homeworld": 1,
          "birth_year": "19BBY"
      },
      "model": "resources.people",
      "pk": 1
},
```

Przykład 3. Przykład 3. Wprowadzenie danych – CouchDB

```
{
  "fields": {
        "_id": "2014-12-20T21:17:56.891Z",
        "name": "Luke Skywalker",
        "created": "2014-12-09T13:50:51.644Z",
        "gender": "male",
        "skin_color": "fair",
        "hair_color": "blond",
        "height": "172",
        "eye_color": "blue",
        "mass": "77",
        "homeworld": 1,
        "birth_year": "19BBY"
      },
      "model": "resources.people",
      "pk": 1
},
```

5. Analiza porównawcza

Poniższy rozdział zawiera wyniki badań uzyskanych podczas przeprowadzania badań magisterskich, dla obu badanych baz danych, MongoDB oraz CouchDB. Wszystkie wyniki badań przedstawione w poniższym rozdziale reprezentując wyniki badań dla określonej liczby powtórzeń danej operacji. Poza tabelami 1 oraz 2, wszystkie tabele zawierają operacje dotyczące tysiąca zestawów danych. Tabela 1 przedstawia wynik czasu uzyskanego dla przeprowadzenia operacji dziesięciokrotnie, a tabela 2 reprezentuje wprowadzanie kolejno dla jednego, dziesięciu, stu i tysiąca zestawów danych. Tabela 1 przedstawiona została uwzględniając wymienione ilości wprowadzeń zestawów, gdyż stanowiła przykład największych różnic uzyskanych podczas wprowadzania testowych zestawów danych widocznych w Przykładzie 1 do obu testowanych baz danych. Operacje usuwania, wyszukiwania, oraz aktualizacji danych najlepiej obrazują różnice gdy wykonane zostały na tysiącu testowych zestawów danych.

Tabela 1. Średni czas wykonanie z użyciem klauzuli WHERE

Operacja	CouchDB (ms)	MongoDB (ms)
WHERE field = ?	0,032	0,016
WHERE field zLIKE ‘x%’	0,072	0,040

Wszystkie wyniki widoczne w powyższej tabeli 1 przedstawiają średni wynik wykonania klauzuli dziesięć razy, a wartości przedstawione są w milisekundach. Zauważyć można iż w obu badanych przypadkach czas wykonania MongoDB jest znacznie krótszy.

Tabela 2. Operacja INSERT

Operacja	CouchDB (ms)	MongoDB (ms)
INSERT 1 ROW	1,201	0,187
INSERT 10 ROW	3,057	2,068
INSERT 100 ROW	13,246	8,842
INSERT 1000 ROW	402,352	106,492

Podobnie jak w przypadku klauzuli, podczas operacji wprowadzania danych można zauważyć iż MongoDB jest wydajniejszy, niezależnie od tego, jak wiele zestawów danych zostaje dodanych do bazy danych w pliku JSON. Na podstawie powyższych wyników można zaobserwować iż im więcej informacji wprowadzamy, tym większa jest różnica zauważalna pomiędzy CouchDB, a MongoDB.

Tabela 3. Operacja DELETE

Operacja	CouchDB (ms)	MongoDB (ms)
DELETE	0,822	0,624

Tabela 3 przedstawia czas w jakim badane bazy wykonują operacje usunięcia wprowadzonych zestawów danych. Wynik ten dotyczy ostatniego dodania z tabeli 2 dotyczącego tysiąca zestawów danych. Podobnie jak w powyższych dwóch przypadkach czas wykonywania operacji w MongoDB jest krótszy od uzyskanego w CouchDB.

Tabela 4. Operacja UPDATE

Operacja	CouchDB (ms)	MongoDB (ms)
UPDATE	1,302	1,331

Podobnie jak w tabeli 3 i w tabeli 4 można zauważyć czas wykonania dla tysiąca testowych zestawów danych. W przeciwieństwie do wszystkich poprzednich wyników, tym razem operacja, która zajęła mniej czasu odbyła się w CouchDB. Podczas badania obu baz, jedynie w uaktualnianiu informacji bazy, CouchDB wykazał się lepszym czasem wykonania od MongoDB.

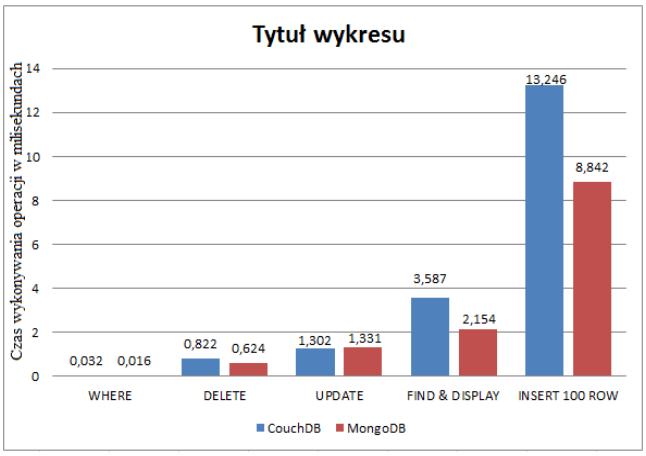
Tabela 5. Operacja FIND & DISPLAY

Operacja	CouchDB (ms)	MongoDB (ms)
FIND & DISPLAY	3,587	2,154

W obu systemach baz danych operacje wyszukania oraz wyświetlenia wykonywane dla tysiąca zestawów danych wprowadzanych do bazy danych, wykonują się w prędkości uznawanej za zadowalającą jak na tak dużą ilość danych, ale także jak w przypadku większości badanych zapytań, tak i to pokazuje iż MongoDB jest wydajniejsze od CouchDB. Rysunek 1 przedstawia obrazowe porównanie umieszczonych w artykule badań. Podczas tego kroku widoczne jest, iż MongoDB poza aktualizacją danych jest wydajniejszy względem czasu, niż druga badana baza NoSQL CouchDB.

6. Wnioski

Podstawowym wnioskiem uzyskanym podczas badań poświęconych porównaniu prędkości zapytań pomiędzy bazami NoSQL CouchDB oraz MongoDB jest potwierdzenie tezy, iż baza MongoDB jest bazą wydajniejszą względem czasu od bazy CouchDB.



Rys.1. Porównanie uzyskanych czasów dla wykonań zapytania.

Poza jednym przypadkiem, aktualizacji danych, baza CouchDB jest bazą zdecydowanie mniej wydajną od bazy MongoDB, która w przypadkach, operacji dodawania, usuwania, wyszukiwania danych oraz prędkości działania pętli jest bazą zdecydowanie wydajniejszą od drugiej badanej bazy.

Największe różnice zaobserwować można w przypadku operacji dodawania tysiąca rekordów do bazy. Wynik uzyskany przez CouchDB jest prawie czterokrotnie wyższy od wyniku uzyskanego przez MongoDB.

Najniższą różnicę natomiast można zaobserwować w przypadku badania operacji aktualizacji danych, w której to MongoDB wypadło nieznacznie gorzej od CouchDB, a różnica pomiędzy badanymi bazami wynosiła zaledwie 0,029 ms

Literatura

[1] R. Henricsson, Document Oriented NoSQL Databases: A comparison of performance in MongoDB and CouchDB using a Python interface, PublicationsElectronic Research Archive Blekinge Tekniska Hogskola; 2011.

[2] P.Noiumkar, T. Chomsiri. A Comparison the Level of Security on Top 5 Open Source NoSQL Databases. Annual International Conference on Information Technology & Applications, 2014.

[3] N. Aloisio Dourado, E. Fernandes,M. Holanda, E. Riberio, RCarvalho MongoDB performance analysis: A comparative study between stand-alone and sharded cluster deployments with open data from Brazilian Bolsa familia program, 12th Iberian Conference on Information Systems and Technologies, 2017.

[4] M. Patil, A. Hanni, C. Tejeshwar, P. Patil, A qualitative analysis of the performance of MongoDB vs MySQL database based on insertion and retriewal operations using a web/android application to explore load balancing — Sharding in MongoDB and its advantages, International Conference on I-SMAC, 2017.

[5] M. Jung, S. Youn, J. Bae, Y. Choi, A Study on Data Input and Output Performance Comparison of MongoDB and PostgreSQL in the Big Data Environment, 8th International Conference on Database Theory and Application, 2015.

[6] M. Brown, Getting Started with CouchDB, Sebastopol, 2012.

[7] S. Gupta, A comparative study of elasticsearch and CouchDB document oriented databases, International Conference on Inventive Computation Technologies, 2016.

[8] K. Kumar, S. Mohanavalli, A performance comparison of document oriented NoSQL databases, International Conference on Computer, Communication and Signal Processing, 2017.

[9] N. Surya, E. Ramez, Quantitative Analysis of Scalable NoSQL Databases, San Francisco. 2016.

[10] P. Pragati, G.Saumya, K. Anil, Analysis of Various NoSql Database, Rajasthan, 2015.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 142-149
Wysłane: 2017-11-28
Przyjęte: 2017-12-02

Zastosowanie .NET Core w budowie aplikacji webowych

Ewelina Piątkowska*, Katarzyna Wąsik*, Małgorzata Plechawska-Wójcik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono zastosowanie .NET Core w budowie aplikacji webowych. Poddano analizie aplikację zbudowaną w oparciu o framework .NET Core. Do analizy frameworka wybrano następujące kryteria porównawcze: wyszukiwanie wzorca w tekście, liczenie czasu parsowania pliku, realizacja operacji na plikach graficznych, dodawanie plików do archiwum, szybkość operacji CRUD. Wymienione kryteria będą porównywane na dwóch różnych systemach operacyjnych – Windows i Linux. Dodatkowo w artykule ukazano wyniki uzyskane po przeprowadzeniu badań oraz ich analizę. Postawiona hipoteza mówiąca, że testy zostaną wykonane szybciej w systemie Windows niż w systemie Linux, została jedynie częściowo potwierdzona.

Słowa kluczowe: aplikacja webowa; .NET Core; Angular 2; Entity Framework

*Autor do korespondencji.

Adresy e-mail: ewelina.piatkowska@pollub.edu.pl, katarzyna.wasik@pollub.edu.pl

The use of .NET Core in web applications development

Ewelina Piątkowska*, Katarzyna Wąsik*, Małgorzata Plechawska-Wójcik
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents the use of .NET Core in web applications development. The analysis covers tests performed on a test application based on the .NET Core framework. The following benchmarking criteria were selected for the framework analysis: pattern search in text, counting parsing time, rendering operations on graphical files, adding files to archives, CRUD operation speed. These criteria were compared on two different operating systems - Windows, Linux. Additionally, the paper presents results obtained after the tests and their analysis. The hypothesis that tests will be executed faster on Windows than on Linux has only been partially confirmed.

Keywords: web application; .NET Core; Angular 2; Entity Framework

*Corresponding author.

E-mail addresses: ewelina.piatkowska@pollub.edu.pl, katarzyna.wasik@pollub.edu.pl

1. Wstęp

W obecnych latach obserwuje się bardzo szybki rozwój Internetu i stron WWW, stąd aplikacje internetowe stały się powszechnie znane i używane na całym świecie. Coraz więcej firm polega na aplikacjach sieciowych, dlatego wartościowość i spójność aplikacji stała się niezbędna [1]. Zaletą aplikacji webowych jest także możliwość korzystania z nich przez dowolną liczbę użytkowników. Poza tym do uruchomienia wystarczy przeglądarka internetowa taka jak np. Mozilla Firefox czy Google Chrome. W tym celu potrzebny jest tylko dostęp do sieci [2].

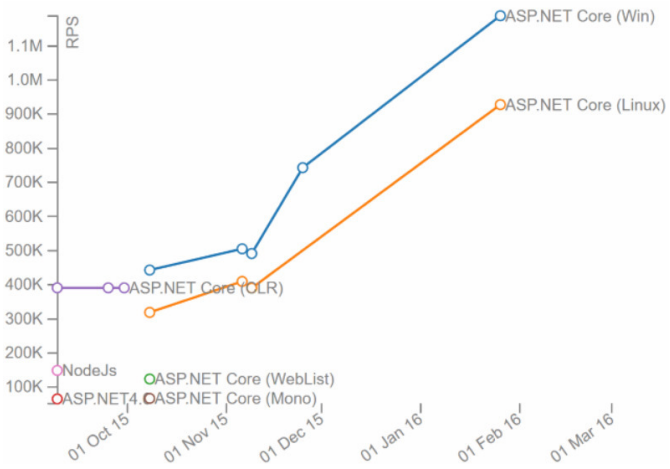
Testowanie jest jednym z pięciu głównych technicznych obszarów działalności inżynierii oprogramowania. Może ono odkryć większość błędów oprogramowania. Tak więc aplikacje internetowe powinny być testowane ostrożnie, aby upewnić się, że wymagane specyfikacje są spełnione przez aplikacje. Testowanie aplikacji webowych jest bardziej złożone niż zwykłych programów, ze względu na charakter dystrybucji, heterogeniczność, niezależność platformy i zgodność [3].

Testowanie aplikacji można realizować za pomocą kodu lub wykorzystując gotowe rozwiązania. Przykładem zewnętrznych oprogramowań są NUnit, XUnit lub Selenium. Niestety nie zawsze jest możliwość wykorzystania tych narzędzi do zbadania wszystkich części aplikacji. Przykładem tego mogą być pomiar wyników czasów realizacji poszczególnych operacji na poziomie serwera [4, 5].

Od 17 maja 2016 roku ASP.NET MVC funkcjonuje jako ASP.NET Core MVC. Jest jedną z najczęściej wybieranych technologii do tworzenia aplikacji internetowych [6]. ASP.NET Core MVC jest to wieloplatformowa, darmowa struktura programistyczna wspomagająca tworzenie aplikacji i usług internetowych z wykorzystaniem wzorca MVC. Dzieli on aplikację na trzy główne składniki: Model, Widok i Kontroler [7]. Łączy zalety starszych frameworków takich jak: ASP.NET MVC, ASP.NET i ASP.NET Web API zawierając je w jednej modułowej strukturze. Poza tym oferuje ona znacznie większą wydajność niż jej poprzednicy i może być wdrożona niemal wszędzie, w tym na Windows Server, Microsoft Azure, Linux czy macOS. Posiada również wbudowane narzędzia ułatwiające współpracę aplikacji ASP.NET Core MVC z kontenerami takimi jak Docker i Pivotal Cloud Foundry [8].

Na poniższym wykresie (Rys. 1), dotyczącym postępu wydajności ASP.NET Core, przedstawiona została zależność liczby żądań na sekundę (RPS) w wyznaczonym czasie [8]. ASP.NET 4.6 oraz Node.Js znajdują się na samym dole i są prawie niewidoczne. Największe postępy w wydajności

można zaobserwować w przypadku bardziej zwinnych systemów, jakimi są Windows i Linux [10]. System Windows został zaznaczony na wykresie niebieską linią, zaś Linux został oznaczony kolorem pomarańczowym [9].



Rys. 1. Wykres postępu wydajności ASP.NET Core [8]

1.1. Cel i obszar badań

Celem badań jest zbadanie wydajności stworzonej aplikacji testowej, bazując na określonych wcześniej kryteriach oraz uruchamiając ją na różnych platformach. W ten sposób będzie możliwe zbadanie jak .NET Core radzi sobie na dwóch różnych systemach: Windows i Linux.

W tym celu pod uwagę wzięto następujące kryteria: wyszukiwanie wzorca w tekście, obliczenie czasu parsowania pliku, realizacja operacji na plikach graficznych (zmiana rozmiaru obrazów), dodawanie plików do archiwum oraz szybkość realizacji operacji CRUD.

1.2. Opis aplikacji testowej

Aplikacja testowa została stworzona w technologii .NET Core w programie Microsoft Visual Studio w języku C#. W celu łatwiejszego tworzenia aplikacji opartej o architekturę MVC skorzystano z frameworka JavaScriptowego - Angular 2. Aplikacja umożliwia rejestrację i logowanie użytkowników w systemie, co daje im możliwość dodawania komentarzy i ich podgląd. Dodawane komentarze pojawiają się na żywo, dzięki wykorzystaniu biblioteki SignalR, bez konieczności odświeżania strony. Są one widoczne dla wszystkich zalogowanych użytkowników i automatycznie zapisywane w stworzonej na Microsoft SQL Server bazie danych. Ułatwieniem logowania jest możliwość zalogowania się za pomocą konta Facebook.

Dodatkowo w celu umożliwienia przetestowania frameworka, każdy zalogowany użytkownik może przetestować aplikację pod kątem dokładniej opisanych w rozdziale 2, niniejszego artykułu, kryteriów. Możliwe jest to po kliknięciu przycisku ”Tests”, znajdującego się w lewym górnym rogu aplikacji.

1.3. Hipotezy badawcze

W każdej pracy badawczej należy określić hipotezę badawczą, która jest jej istotną częścią. W artykule postawiono hipotezę dotyczącą porównania wydajności .NET Core. Założono, że testy zostaną wykonane w systemie Windows w krótszym czasie niż w systemie Linux. Zostanie to sprawdzone za pomocą wyżej wymienionych kryteriów.

2. Kryteria analizy

Poniżej skupiono się na dokładniejszym opisie wszystkich określonych wcześniej kryteriów służących sprawdzeniu wydajności frameworka. Wszystkie testy zostały zrealizowane za pomocą kodu, co stanowi duże ułatwienie w testowaniu i dalszej analizie uzyskanych wyników.

2.1. Wyszukiwanie wzorca w tekście

Dany test polega na wyszukaniu w dużej ilości tekstu konkretnego fragmentu tekstu, czyli wzorca. W jednym z folderów znajduje się plik z tekstem nazwany Text.txt , który zajmuje około 300 megabajtów.

Test wyszukiwania wzorca w tekście znajduje się w metodzie FindStringInText(). Na początku do pamięci jest wczytywany cały tekst zawarty w pliku Text.txt, następnie wyszukiwany jest ostatni indeks, w którym występuje szukane słowo „Lorem”, co ilustruje poniższy przykład 1. Zwracany typ double pokazuje czas działania testu.

Przykład 1. Metoda FindStringInText()

```
public double FindStringInText(string allText, string findingText)
{
    var watch = Stopwatch.StartNew();
    string text = File.ReadAllText(@".\Media\Text.txt");
    text.LastIndexOf("Lorem");
    watch.Stop();
    return TimeSpan.FromMilliseconds(watch.ElapsedMilliseconds).TotalSeconds;
}
```

2.2. Liczenie czasu parsowania pliku

W przypadku tego testu zajęto się obliczeniem czasu przetwarzania obiektu do obiektu typu JSON. Początkowo tworzona jest lista obiektów typu String. W pętli lista jest uzupełniana danymi, a następnie cała lista zostaje parsowana do typu JSON. Wykonywane jest to w linijce JsonConvert.SerializeObject(files), co widoczne jest na przykładzie 2. Do parsowania obiektów jest wykorzystywana biblioteka Newtonsoft.Json.

Przykład 2. Metoda ParseJsonObject()

```
public double ParseJsonObject(object objectForParsing)
{
    List<string> files = new List<string>();
    for (int i = 0; i < 500; i++)
    {
        files.Add(i.ToString());
    }
    var watch = Stopwatch.StartNew();
    JsonConvert.SerializeObject(files);
    watch.Stop();
    return TimeSpan.FromMilliseconds(watch.ElapsedMilliseconds).TotalSeconds;
}
```

2.3. Realizacja operacji na plikach graficznych

Do operacji na obrazkach zostanie wykorzystany resize, czyli zmiana wielkości obrazu. Cały proces odbywa się w metodzie ResizeImage() (Przykład 3).

Przykład 3. Metoda ResizeImage()

```
public double ResizeImage()
{
    var watch = Stopwatch.StartNew();
    using (Image<Rgba32> image = Image.Load(@".\Media\Images\earth.jpg"))
    {
        image.Mutate(x => x
            .Resize(image.Width / 2, image.Height / 2)
            .Grayscale());
        image.Save(@".\Media\Images\resizedImage.jpg");
    }
    watch.Stop();
    return TimeSpan.FromMilliseconds(watch.ElapsedMilliseconds).TotalSeconds;
}
```

W danym teście obliczany jest czas zmiany rozmiaru obrazka. W folderze .../Media/Images jest umieszczony obrazek, który ma rozmiar 15000x15000. W trakcie testu jego rozmiar jest zmniejszany o połowę (do 7500x7500) oraz kolory obrazka przetwarzane są na czarno-białe.

Przetworzony obrazek zostaje umieszczony w folderze .../Media/Images/ResizedImages/resizedImage. Do pracy z obrazkami wykorzystywana jest biblioteka SixLabors.ImageSharp, która jest wieloplatformowa. To samo tyczy się obrazka o wymiarach 4000x4000.

Za operacje na obrazkach odpowiada również filtr Gauss. Fragment tej metody jest widoczny na poniższym przykładzie (Przykład 4).

Przykład 4. Metoda ApplyGausBlur()

```
var bigImageTime = Stopwatch.StartNew();
using (Image<Rgba32> image = Image.Load(pathToBigImage))
{
    image.Mutate(x => x.GaussianBlur(10));
    image.Save(pathToBigSavedImage);
}
bigImageTime.Stop();
```

Na początku w tej metodzie podawana jest ścieżka dostępu do obrazków o dwóch różnych rozmiarach podanych wyżej, następnie tworzony jest obiekt obrazu. Za ich przetworzenie odpowiada biblioteka *ImageSharp*, wykorzystująca metodę *GaussianBlur()*.

2.4. Dodawanie plików do archiwum

W danym teście obliczany jest czas dodawania plików do archiwum. Wszystko odbywa się w metodzie ZipFiles() (Przykład 5).

Przykład 5. Metoda ZipFiles()

```
public double ZipFiles()
{
    FilesHelper.CreateRandomFiles();
    FilesHelper.CleanResultDirectory();

    var watch = Stopwatch.StartNew();
    ZipFile.CreateFromDirectory(@".\Media\FilesForZip", @".\Media\ZipedFile\result.zip");
    watch.Stop();
    return TimeSpan.FromMilliseconds(watch.ElapsedMilliseconds).TotalSeconds;
}
```

W folderze .../Media/FilesForZip umieszczone są pliki do archiwizacji. Zostały one wygenerowane w linijce z metodą FilesHelper.CreateRandomFiles(). Pliki generowane są o różnych rozmiarach, łącznie generowane jest 500 plików.

Stworzony plik .zip zawierający wszystkie utworzone pliki zostanie umieszczony w folderze .../Media/ZippedFile/result.zip. Przed rozpoczęciem testu zawsze sprawdzany jest ten folder. W razie gdyby nie był pusty, wszystkie pliki zostaną usunięte przed wykonaniem testu.

2.5. Szybkość realizacji operacji CRUD

Szybkość realizacji operacji CRUD obliczana jest w metodzie CountSQLQueriesGeneratingTime(). Oblicza ona czas zapytań wykonywanych w bazie danych. Metoda zwraca trzy parametry typu double (czas wyrażony jest w sekundach). Pierwszy parametr jest czasem odpowiedzi wykonania operacji typu INSERT, drugi parametr jest czasem wykonania zapytań typu SELECT, trzeci parametr jest parametrem czasowym dla operacji typu DELETE (Przykład 6).

Przykład 6. Metoda CountSQLQueriesGeneratingTime()

```
public double CountSQLQueriesGeneratingTime()
{
    var watch = Stopwatch.StartNew();

    _commentRepository.GetAll().AsQueryable();

    watch.Stop();
    return TimeSpan.FromMilliseconds(watch.ElapsedMilliseconds).TotalSeconds;
}
```

Komunikacja z bazą danych polega na korzystaniu z frameworka Entity Framework, który jest narzędziem służącym do pracy z bazą danych jak ze zwykłymi obiektami. W sekcji przy obliczeniu operacji INSERT w pętli są tworzone obiekty typu Comment i od razu dodawane są one do kolekcji bazodanowej Comments. Po dodaniu danych do

bazy danych wywoływane jest zapytanie pobierania danych z bazy (SELECT). Po wykonaniu instrukcji SELECT, wszystkie dane będą usunięte - w tym miejscu jest obliczany czas dla operacji typu DELETE.

Testy będą wywoływane wielokrotnie dla każdego typu operacji. Dla każdego rodzaju operacji z osobna zostały wykonane testy z różną ilością powtórzeń wykonania danego zapytania. Na przykład, w przypadku operacji wstawiania wykonywane będzie wstawianie wielu rekordów kolejno. W pierwszym teście operacja ta zostanie powtórzona 1 000 razy, w następnym kroku 5 000 razy, w kolejnym 50 000 razy itd.

3. Metody i przebieg badań

Do badań wybrano dwa systemy operacyjne: Windows i Linux, z założeniem sprawdzenia wydajności platformy .NET Core. Celem optymalnej realizacji badań należało wybrać właściwe metody badawcze. W związku z tym, zdecydowano się na wybór metody porównawczej. Metoda ta została wykorzystana do porównania wyżej wymienionych systemów operacyjnych i wyników przeprowadzonych na nich testów na podstawie stworzonej aplikacji. Uwidoczniła ona różnice w działaniu frameworka na każdym z badanych systemów.

Na początku zainstalowano maszynę wirtualną, a na niej oba systemy operacyjne o takich samych parametrach. Po konfiguracji środowiska i pobraniu niezbędnych narzędzi nastąpiło uruchomienie stworzonej wcześniej aplikacji testowej. Następnie uruchomiono zaprojektowane uprzednio odpowiednie testy.

W celu sprawdzenia jak .NET Core radzi sobie na każdym z badanych systemów należało uruchomić wcześniej stworzoną aplikację zawierającą opracowane testy i przeanalizować uzyskane wyniki, a w końcowym etapie sporządzić ich interpretacje graficzne. Dla uzyskania powyższego rezultatu należało w uruchomionej aplikacji przejść do strony z testami, uruchomić je i poczekać na ich wykonanie.

W przypadku systemu Windows należało, po zainstalowaniu go na maszynie wirtualnej o nazwie VirtualBox i doinstalowaniu potrzebnych narzędzi, uruchomić stworzoną na potrzeby pracy aplikację z testami i przejść do strony, gdzie będą wykonywane testy, następnie uruchomić je i poczekać na wyniki. Jeśli chodzi o system Linux czynności do wykonania pozostają identyczne.

Testy, na każdym z systemów, zostały powtórzone dla operacji CRUD po trzy razy dla 13 wybranych wartości, zaś dla pozostałych kryteriów pięciokrotnie. Z uzyskanych wyników wyliczono wartości średnie i przedstawiono je na wykresach. Całość opisana została w rozdziale 4.

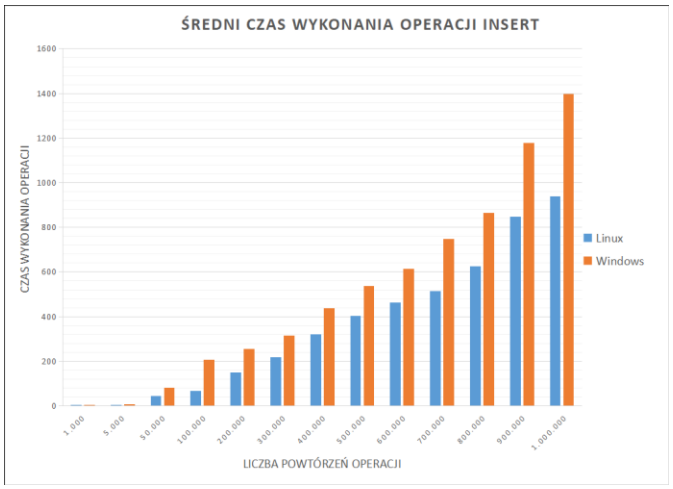
4. Wyniki badań

Po przeprowadzeniu testów poszczególne wyniki dla wybranych uprzednio kryteriów zostały umieszczone w kolejnych tabelkach, zamieszczonych poniżej. W tabeli 1 przedstawiono średnie czasy wykonania powtórzeń instrukcji INSERT dla 13 różnych wartości: 1 000, 5 000, 50 000, 100 000, 200 000, 300 000, 400 000, 500 000, 600 000, 700 000, 800 000, 900 000, 1 000 000 na dwóch badanych systemach operacyjnych: Windows i Linux wraz z odchyleniem standardowym.

Tabela 1. Zestawienie średnich wyników przeprowadzonych testów dla operacji INSERT wraz z odchyleniem standardowym

Liczba powtórzeń/ Badany system	Linux Średni czas [s]	Windows Średni czas [s]	Odchyle-nie std dla wyników Linux	Odchyle-nie std dla wyników Windows
1 000	1,3393	1,8723	0,1386	0,4824
5 000	3,8673	6,6817	0,6474	0,6674
50 000	43,6067	80,682	4,5801	60,6126
100 000	65,6977	206,5337	7,0350	45,5416
200 000	150,1527	254,1607	17,3437	41,6918
300 000	217,637	315,5603	10,628	143,0129
400 000	319,639	438,065	54,3253	248,591
500 000	402,568	538,4593	5,3889	204,3592
600 000	463,533	615,0413	63,4849	215,1631
700 000	513,208	748,264	27,4094	174,56
800 000	625,3034	865,6873	158,9147	118,7052
900 000	846,366	1177,797	66,7951	256,0625
1 000 000	938,0707	1393,6947	123,0148	272,2751

Na rysunku 2 zobrazowano wyniki przedstawione w tabeli 1 dotyczące średniego czasu wykonania operacji INSERT. Na osi X znajduje się liczba powtórzeń wykonania danej operacji, a na osi Y – badana wartość, czyli czas wykonywania operacji. Przykładowo dla 400 000 powtórzeń, dla systemu Linux osiągnięto wynik 319,639s, a dla systemu Windows 438,065s.



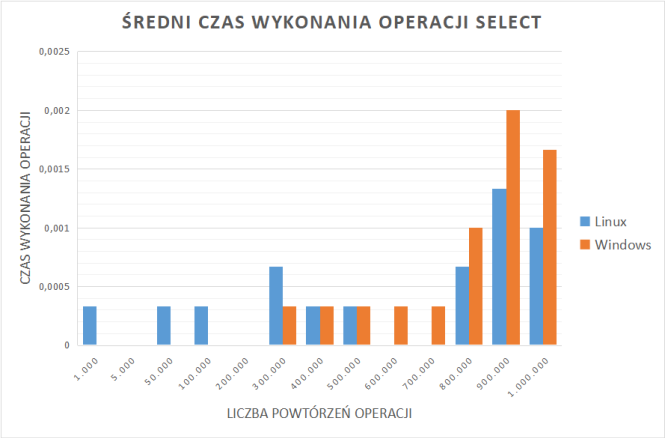
Rys. 2. Średni czas wykonania operacji INSERT

W tabeli 2 przedstawiono średnie czasy wykonania powtórzeń instrukcji SELECT dla 13 różnych wartości na dwóch badanych systemach operacyjnych wraz z odchyleniem standardowym.

Tabela 2. Zestawienie średnich wyników przeprowadzonych testów dla operacji SELECT wraz z odchyleniem standardowym

Liczba powtórzeń / Badany system	Linux Średni czas [s]	Windows Średni czas [s]	Odchyle-nie std dla wyników Linux	Odchyle-nie std dla wyników Windows
1 000	0,00033	0	0,000577	0
5 000	0	0	0	0
50 000	0,0003333	0	0,000577	0
100 000	0,00033	0	0,000577	0
200 000	0	0	0	0
300 000	0,00067	0,000333	0,001154	0,000577
400 000	0,00033	0,000333	0,000577	0,000577
500 000	0,00033	0,000333	0,000577	0,000577
600 000	0	0,000333	0	0,000577
700 000	0	0,000333	0	0,000577
800 000	0,0006667	0,001	0,001154	0,001173
900 000	0,00133	0,002	0,001154	0,001173
1 000 000	0,001	0,001667	0,001732	0,002081

Na rysunku 3 przedstawiono graficzną reprezentację wyników umieszczonych w tabeli 2. Przykładowo dla 1 000 000 powtórzeń w systemie Linux otrzymano wartość średnią równą 0,001s, a dla Windows 0,001667s.



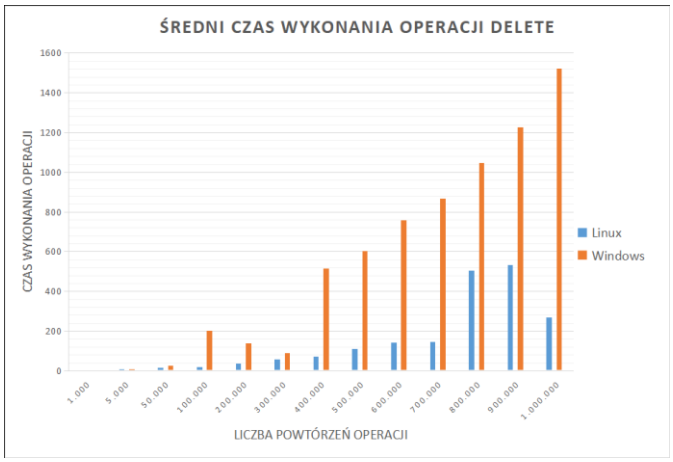
Rys. 3. Średni czas wykonania operacji SELECT

W tabeli 3 przedstawiono średnie czasy wykonania powtórzeń instrukcji DELETE dla 13 różnych wartości na uprzednio wymienionych systemach wraz z odchyleniem standardowym.

Tabela 3. Zestawienie średnich wyników przeprowadzonych testów dla operacji DELETE wraz z odchyleniem standardowym

Liczba powtórzeń / Badany system	Linux Średni czas [s]	Windows Średni czas [s]	Odchyle-nie std dla wyników Linux	Odchylenie std dla wyników Windows
1 000	1,6317	2,2823	0,2194	0,7599
5 000	3,475	8,415	3,6525	1,5167
50 000	13,9693	27,3037	3,9594	22,5443
100 000	19,9017	203,3063	1,8361	54,1722
200 000	36,078	139,4483	4,1181	178,5141
300 000	56,4167	89,1813	2,8853	44,0074
400 000	71,9973	514,9533	7,4012	174,185
500 000	110,907	602,113	13,1123	181,5452
600 000	142,103	756,9773	29,3997	178,5347
700 000	145,295	868,9963	19,112	174,9403
800 000	506,684	1048,5933	528,8153	282,8456
900 000	533,106	1227,003	489,6729	413,0618
1 000 000	268,009	1522,1483	40,5415	383,4439

Rysunek 4 prezentuje średnie wyniki otrzymane po wykonaniu operacji DELETE. Dokładne dane przedstawione są w tabeli 3.



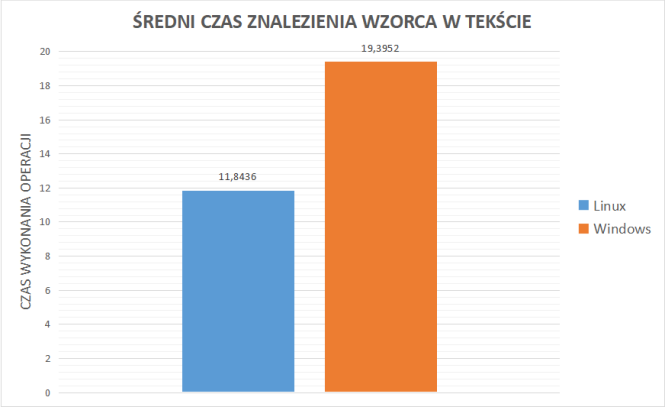
Rys. 4. Średni czas wykonania operacji DELETE

Tabela 4 zawiera czasy wyszukiwania wzorca w tekście, dla pięciu powtórzeń wraz z wartością średnią i odchyleniem standardowym. W systemie Linux średni czas wykonania tej operacji wynosi 11,8436s, zaś w systemie Windows 19,3952s.

Tabela 4. Zestawienie wyników przeprowadzonych testów dla czasu wyszukiwania wzorca w tekście wraz z odchyleniem standardowym

Badany system	Czas [s]
Linux	8,337
	5,604
	5,842
	5,631
	33,804
Średnia dla wyników Linux	**11,8436**
Odchylenie standardowe dla wyników Linux	**12,3299**
Windows	18,558
	18,878
	14,41
	23,999
	21,131
Średnia dla wyników Windows	**19,3952**
Odchylenie standardowe dla wyników Windows	**3,5365**

Na rysunku 5 zestawiono porównanie wartości średnich wyników z tabeli 4.



Rys. 5. Średni czas znalezienia wzorca w tekście

Tabela 5 demonstruje wartości uzyskane podczas testów mierzenia czasu parsowania pliku dla pięciu powtórzeń wraz z wartością średnią i odchyleniem standardowym. W systemie Linux średni czas wykonania tej operacji wynosi 1,689s, zaś w systemie Windows 1,5648s.

Tabela 5. Zestawienie wyników przeprowadzonych testów dla czasu parsowania pliku wraz z odchyleniem standardowym

Badany system	**Czas [s]**
Linux	1,472
	1,607
	1,647
	1,489
	2,23
Średnia wyników dla Linux	**1,689**
Odchylenie standardowe dla wyników Linux	**0,3116**
Windows	1,326
	1,202
	1,512
	1,374
	2,41
Średnia wyników dla Windows	**1,5648**
Odchylenie standardowe dla wyników Windows	**0,4853**

Poniższy rysunek (Rys. 6) pozwala dokładniej zaobserwować niewielką różnicę miedzy otrzymanymi średnimi wynikami.

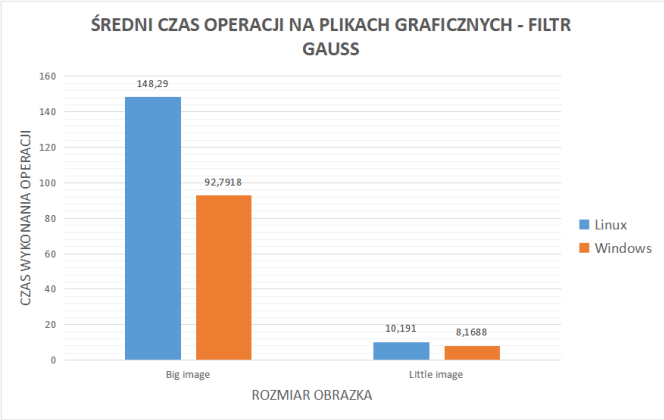
Podczas wykonywania testów na plikach graficznych rozpoczęto od naniesienia filtru Gaussa na obrazy. Uzyskane wyniki zawiera tabela 6 dla pięciu powtórzeń wraz z wartościami średnimi i odchyleniem standardowym. W systemie Linux średni czas wykonania tej operacji dla dużego obrazka wynosi 148,29s, a dla małego 10,191s, zaś w systemie Windows średni czas wykonania operacji na dużym obrazku wynosi 92,7918s, a na małym obrazku 8,1688s.

Rys. 6. Średni czas parsowania pliku

Tabela 6. Zestawienie wyników przeprowadzonych testów dla czasu operacji na plikach graficznych – filtr Gauss wraz z odchyleniem standardowym

Badany system	**Czas [s] – duży obrazek**	**Czas [s] – mały obrazek**
Linux	160,664	10,361
	144,184	10,738
	148.158	9,937
	149,212	9,165
	139,232	10,754
Średnia wyników dla Linux	**148,29**	**10,191**
Odchylenie standardowe dla wyników Linux	**7,9513**	**0,6638**
Windows	93,623	6,763
	90,992	6,831
	91,737	6,924
	90,597	6,67
	97,01	13,656
Średnia wyników dla Windows	**92,7918**	**8,1688**
Odchylenie standardowe dla wyników Windows	**2,6293**	**3,0688**

Średnie wyniki przeprowadzonych testów wyrażono na rysunku 7.



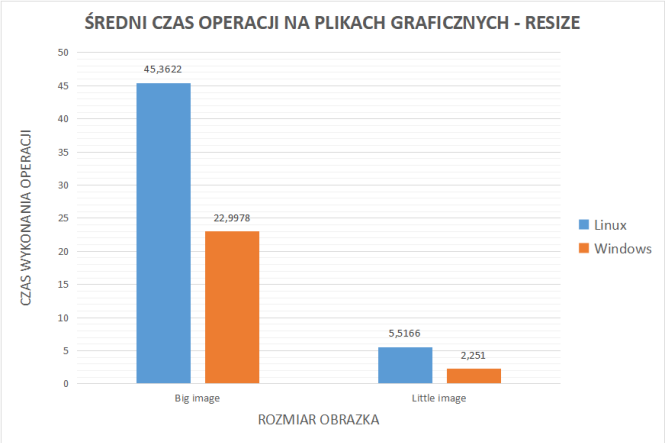
Rys. 7. Średni czas wykonania operacji na plikach graficznych – Filtr Gauss

Wykonując operacje resize otrzymano wyniki, które ukazuje tabela 7 dla pięciu powtórzeń wraz z wartościami średnimi i odchyleniem standardowym. W systemie Linux średni czas wykonania tej operacji dla dużego obrazka wynosi 45,3622s, a dla małego 5,5166s, zaś w systemie Windows średni czas wykonania operacji na dużym obrazku wynosi 22,9978s, a na małym obrazku 2,251s.

Tabela 7. Zestawienie wyników przeprowadzonych testów dla czasu operacji na plikach graficznych – Resize wraz z odchyleniem standardowym

Badany system	Czas [s] – duży obrazek	Czas [s] – mały obrazek
Linux	48,728	15,94
	44,238	2,642
	35,164	2,422
	52,557	2,86
	46,124	3,719
Średnia wyników dla Linux	**45,3622**	**5,5166**
Odchylenie standardowe dla wyników Linux	**6,4989**	**5,8476**
Windows	21,402	1,945
	19,475	1,475
	31,798	3,743
	18,781	1,419
	23,533	2,673
Średnia wyników dla Windows	**22,9978**	**2,251**
Odchylenie standardowe dla wyników Windows	**5,2552**	**0,9737**

W celu dokładniejszego porównania średnich wyników przedstawiono je również na rysunku (Rys. 8).



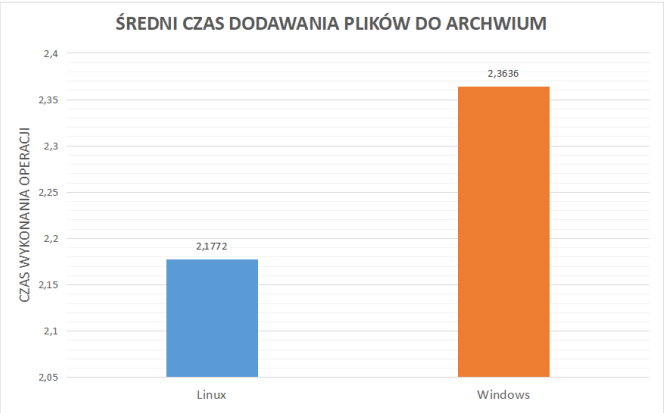
Rys. 8. Średni czas wykonania operacji na plikach graficznych - RESIZE

Tabela 8 pozwala zapoznać się z rezultatem testów mierzenia czasu dodawania plików do archiwum na dwóch systemach operacyjnych dla pięciu powtórzeń wraz z wartością średnią i odchyleniem standardowym. W systemie Linux średni czas wykonania tej operacji wynosi 2,1772s, zaś w systemie Windows 2,3636s.

Tabela 8. Zestawienie wyników przeprowadzonych testów dla czasu dodania plików do archiwum wraz z odchyleniem standardowym

Badany system	Czas [s]
Linux	3,309
	2,937
	3,006
	0,135
	1,499
Średnia wyników dla Linux	**2,1772**
Odchylenie standardowe dla wyników Linux	**1,3394**
Windows	1,523
	1,98
	1,651
	1,337
	5,327
Średnia wyników dla Windows	**2,3636**
Odchylenie standardowe dla wyników Windows	**1,6731**

Średnie wyniki zostały również przeniesione na rysunek (Rys. 9).



Rys. 9. Średni czas dodawania plików do archiwum

5. Wnioski

Celem pracy było sprawdzenie jak .NET Core radzi sobie na dwóch różnych systemach operacyjnych. W tym celu posłużono się metodą porównawczą i założono, że operacje realizowane w systemie Windows będą wykonywane w krótszym czasie. Udało się udowodnić część założeń przyjętych na początku niniejszego artykułu.

Odnośnie wyników pierwszego kryterium jakim jest szybkość wykonywania operacji INSERT, lepszy okazał się system Linux. Udowadniają to wartości zamieszczone w tabeli 1. Z kolei, jeśli chodzi o operacje SELECT, okazało się, że oba systemy mają podobne średnie wyniki. Średni czas wykonania operacji DELETE jednoznacznie wskazuje, że ten test szybciej został wykonany w systemie Linux.

W drugim kryterium uwidacznia się, że Linux jest prawie dwukrotnie szybszy niż Windows. Wyszukiwanie wzorca w tekście odbywa się zdecydowanie sprawniej na tym systemie.

Czas parsowania plików jest porównywalny w obu systemach, choć nieznacznie szybszy jest system Windows, co widoczne jest na rysunku 6.

Przechodząc do omawiania operacji na plikach graficznych należy skupić się w pierwszej kolejności na filtrze Gaussa. Dane przedstawione w tabeli 6 i zilustrowane na rysunku 7 doskonale obrazują szybkość z jaką poradził sobie Windows. Czas wykonywania operacji na plikach graficznych w systemie Windows jest nieporównywalnie krótszy, niż w systemie Linux. Dotyczy to zarówno mniejszych jak i większych obrazów.

Kolejnymi działaniami jakie zostały podjęte na obrazach jest zmiana ich wielkości. Tutaj także widoczne jest, iż Windows uzyskał lepsze rezultaty. Zostało to zilustrowane na rysunku 8.

Średnia dodawania plików do archiwum w obu platformach wynosi nieco powyżej 2 sekund. Jednakże zauważyć można, że Linux poradził sobie z tym nieco lepiej. Windows osiągnął średni wynik 2,3636s, a Linux 2,1772s.

Testy przeprowadzone na podstawie założonych kryteriów pozwalają zauważyć, że postawiona hipoteza została jedynie częściowo potwierdzona. Windows znacznie sprawniej radzi sobie z operacjami na plikach graficznych, takimi jak zmiana rozmiaru obrazu czy nakładanie filtrów. Natomiast jeśli chodzi o operacje związane z plikami tekstowymi to przoduje Linux, na przykład jak ma to miejsce przy wyszukiwaniu wzorca w tekście. Jeżeli chodzi o operacje na bazie danych to testy zostały szybciej wykonane w systemie Linux.

Literatura

[1] P. Nikfard, Functional Testing on Web Applications, University Technology Malaysia, January 2012.

[2] G. A. Di Lucca, A. R. Fasolino, Web Application Testing, Springer, 2006.

[3] P. Nikfard, Testing on Web Applications,Gorgan, Iran, May 2014.

[4] M. Monier, Evaluation of automated web testing tools, International Journal of Computer Applications Technology and Research, Volume 4– Issue 5, 405 - 408, 2015, ISSN:- 2319–8656.

[5] K. S. Dar, S. Tariq, H. J. Akram, U. Ghani, S. Ali, Web Based Programming Languages that Support Selenium Testing, Integrnational Journal of Foresight and Innovation Policy 2015(1), December 2015, s.21-25.

[6] J. Ciliberti, Getting the Most from the New Features in ASP.NET Core MVC, in ASP.NET Core Recipes, 2017, s.139-170.

[7] A. Freeman, Understanding Angular and ASP.NET Core MVC, in Essential Angular for ASP.NET Core MVC, July 2017, s.1-4.

[8] J. Ciliberti, ASP.NET Core Recipes A Problem-Solution Approach, Apress, 2017, s.1-42.

[9] .NET Core, .NET Framework, Xamarin – The "WHAT and WHEN to use it" https://blogs.msdn.microsoft.com/cesardelatorre/2016/06/27/net-core-1-0-net-framework-xamarin-the-whatand-when-to-use-it/ [15.11.2017].

[10] S. Sanderson, What's the Big Idea?, in Pro ASP.NET MVC Framework, January 2009.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 150-157
Wysłane: 2017-11-28
Przyjęte: 2017-12-03

Analiza możliwości obrony przed atakami SQL Injection

Bogdan Krawczyński*, Jarosław Marucha, Grzegorz Kozieł
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie: Publikacja dotyczy ataków SQL Injection, które stanowią jedno z głównych zagrożeń w cyberprzestrzeni. W oparciu o badania literaturowe dokonano klasyfikacji ataków SQL Injection. Celem pracy było przeprowadzenie analizy możliwości obrony przed atakami SQL Injection. Metodykę badawczą oparto na autorskiej aplikacji, zaimplementowanej w technologii JSP (JavaServer Pages) z wykorzystaniem serwera baz danych MySQL.

Słowa kluczowe: Wstrzykiwanie kodu SQL; bezpieczeństwo danych; podatność aplikacji

*Autor do korespondencji.

Adres e-mail: bogdan.krawczynski@pollub.edu.pl

Analysis of protection capabilities against SQL Injection attacks

Bogdan Krawczyński*, Jarosław Marucha, Grzegorz Kozieł
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. Publication refers to SQL Injection attacks whose are one of the most dangerous in a cyberspace. Based on a literature studies, classification of the SQL Injection attacks was prepared. The purpose of the work was to analyse of protections effectiveness against SQL Injection attacks. Research method has been based on author application, which was implemented in JSP (JavaServer Pages) technology using MySQL database server.

Keywords: SQL Injection; data security; application vulnerability

*Corresponding author.

E-mail address: bogdan.krawczynski@pollub.edu.pl

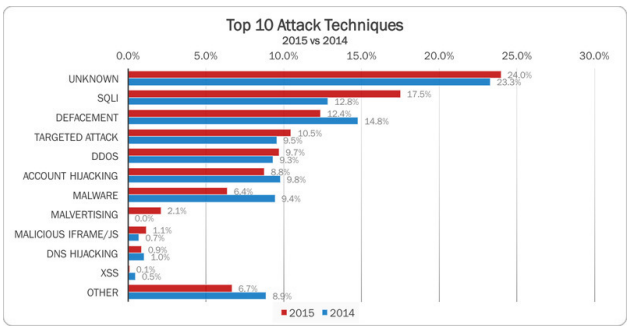
1. Wprowadzenie

Nasilenie globalizacji oraz konsekwentnie postępujący rozwój technologiczny spowodowały, że systemy teleinformatyczne towarzyszą nam niemal w każdej dziedzinie życia. Praktycznie wszystkie z nich, niezależnie od realizowanych funkcjonalności, opierają swoje działanie na gromadzeniu i/lub przetwarzaniu danych. W konsekwencji, doprowadziło to do sytuacji, w której dane stały się kluczowym zasobem.

Równolegle z procesem informatyzacji obserwujemy zjawisko zwane cyberprzestępczością. Pod tym pojęciem kryje się szereg nielegalnych działań dokonywanych przy pomocy Internetu. Łupem hackerów bardzo często stają się dane np. osobowe, przy pomocy których możliwe jest dokonywanie nieautoryzowanych transakcji (np. bankowych). Pomimo istnienia szeregu regulacji administracyjno-prawnych, w postaci dokumentów takich jak ustawa o ochronie danych osobowych, stale dochodzi do eskalacji liczby poszkodowanych oraz strat, powstałych w wyniku ataków hackerskich.

W związku z powyższym odpowiedzialność za bezpieczeństwo danych spoczywa na twórcach oprogramowania. Wynik pracy nie tylko programistów, ale również projektantów i testerów decyduje o końcowej jakości. Produkt dobry i konkurencyjny, to taki który jest użytkowy, ale przede wszystkim bezpieczny. Zapewnienie poufności, dostępności oraz integralności danych jest kluczowe dla każdego klienta docelowego. Niezależnie od tego czy jest nim osoba prywatna, urząd państwowy czy międzynarodowe przedsiębiorstwo.

Pierwsze doniesienia dotyczące ataków SQL Injection datowane są na rok 1998 [1]. Jednak pojęcie to zaczęło funkcjonować dopiero w roku 2000, po wprowadzeniu go przez Chipa Andrewsa po napisaniu i opublikowaniu artykułu „SQL Injection FAQ". Pomimo upływu ponad 15 lat, ataki SQL Injection nie tracą na „popularności" [2]. Wręcz przeciwnie – utrzymują się w czołówce ataków dokonywanych w cyberprzestrzeni, co przedstawiono na rysunku 1.



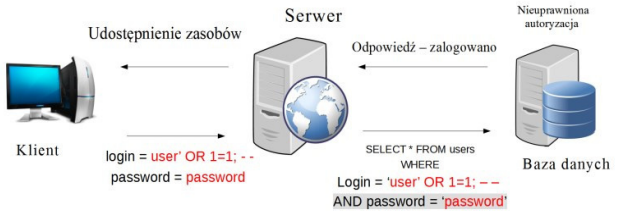
Rys. 1. Ranking 10 najczęstszych ataków w latach 2014-2015 [2]

OWASP (ang. *Open Web Application Security Project*) to światowa, otwarta i profesjonalna organizacja non-profit. Główny celem stowarzyszenia jest „poprawa bezpieczeństwa oprogramowania". Mając na celu obiektywizm w zakresie prowadzonych działań (badań, publikacji), społeczność unika wszelkich powiązań z firmami technologicznymi [3]. Raport organizacji OWASP z bieżącego roku, jako główne zagrożenie dla bezpieczeństwa aplikacji internetowych, wskazuje ich podatności na iniekcję kodu (w tym kodu SQL, tj. SQL Injection) [4].

2. SQL Injection – mechanika ataku

Na rysunku 2. przedstawiono logiczny schemat przeprowadzenia ataku SQL Injection. Aby skutecznie zabezpieczyć się przed atakami SQL Injection należy w pierwszej kolejności zrozumieć ich istotę. SQL Injection w najprostszy sposób można scharakteryzować jako atak na system bazy danych, polegający na wstrzyknięciu (ang. *injection*) niepożądanego kodu SQL za pośrednictwem aplikacji. Zainfekowany kod zostaje przekazany, a następnie wykonany przez serwer bazy danych. W konsekwencji napastnik jest w stanie w sposób nieautoryzowany wstawić, odczytać, zmodyfikować oraz usunąć dane (ang. *CRUD – Create Read Update Delete*). Oznacza to, że ataki SQL Injection naruszają trzy podstawowe własności bezpieczeństwa informacji [5]:

- dostępność – udostępnianie i umożliwienie modyfikacji danych, w odpowiedzi na żądanie uprawnionego użytkownika systemu, realizowane w określonym czasie,
- poufność – gwarancja ochrony danych przed ich nieautoryzowanym dostępem przez osoby, procesy lub inne podmioty,
- integralność – zabezpieczenie danych przed ich nieautoryzowaną modyfikacją lub usunięciem.



Rys. 2. Schemat logiczny ataku SQL Injection

Ponadto, w niektórych przypadkach, ataki SQL Injection mogą posłużyć do przeglądania, zapisu i odczytu plików lokalnych [6].

Aplikacje internetowe, niezależnie od technologii w której powstały, działają w architekturze klient-serwer, którą najlepiej opisuje reguła żądanie-odpowiedź (ang. *request-response*). Klient wysyłając żądanie może przesyłać do serwera wartości (np. formularz logowania), które następnie są wstawiane podczas generowania dynamicznych zapytań do bazy danych. To rodzi zagrożenie stwarzając sytuację, w której możliwe jest przesłanie łańcucha znaków, który zmodyfikuje oryginalną postać zapytania.

3. Klasyfikacja ataków SQL Injection

SQL Injection to obszerny termin, pod którym kryje się szereg różnych technik dokonywania ataku. Umiejętności, specjalistyczna wiedza, ale i kreatywność hackerów przekładają się na liczebność wariantów ataków. W ramach analizy możliwości obrony przed nimi, postanowiono skupić się najpopularniejszych spośród nich. Język SQL w znaczący sposób przekłada się na dostępne techniki ataków. Co więcej, samo wstrzyknięcie kodu SQL – przekazanie parametrów, za pośrednictwem aplikacji internetowej, może odbywać się na kilka sposobów.

Protokół HTTP (ang. *HyperText Transfer Protocol*) umożliwia dwukierunkową i znormalizowaną komunikację pomiędzy klientem a serwerem. Każdy nagłówek (ang. *HTTP header*) żądania klienta oprócz adresu URL żądanego zasobu może zawierać również parametry.

Parametry te mogą być przekazywane przy użyciu [7]:

- metody HTTP GET – za pośrednictwem adresu URL (ang. *Uniform Resource Locator*), czyli ciągu znakowego, w którym oprócz ścieżki żądanego zasobu, przekazywać można wartości parametrów.
- metody HTTP POST – która w porównaniu do poprzedniej, cechuje się większym poziomem bezpieczeństwa, ponieważ przesyła wartości parametrów w sekcji body nagłówka HTTP. Najczęściej są to wartości wprowadzane przez użytkownika w formularzu.
- nagłówka HTTP – w którym, oprócz powyższych metod oraz metadanych, przesyłane są również inne dane na podstawie, których dokonywana jest optymalizacja wyświetlanej strony.
- plików cookie – (ang. *HTTP cookies*), które bardzo często wykorzystywane są np. do utrzymania sesji pomiędzy klientem a serwerem. Bezstanowość protokołu HTTP stanowi pewną przeszkodę dla twórców aplikacji internetowych. Przy pomocy ciasteczek możliwa jest personalizacja serwisu WWW [6].

Strukturalny język zapytań (ang. *Structured Query Language*) umożliwia tworzenie i modyfikowanie baz danych. Ponadto pozwala wstawiać, manipulować oraz usuwać dane. Każda z powyższych operacji odbywa się w analogiczny sposób i jest realizowana przy pomocy zapytania wykonywanego przez serwer bazy danych. Struktura zapytań SQL, podobnie jak sekwencja jego wykonania, jest precyzyjnie określona. Podczas próby wykonania zapytania z błędem składniowym lub niekompatybilnym typem zmiennych, wyświetlony zostaje stosowny błąd.

Ataki SQL Injection można podzielić na trzy główne typy [8]:

1) In-band (wewnątrzpasmowy) – najprostszy i najpopularniejszy typ ataku. Napastnik, przeprowadzając atak, jest w stanie zaobserwować jego rezultat za pośrednictwem tego samego kanału komunikacyjnego. Przykładem takiego ataku może być wyświetlenie dodatkowych informacji przy pomocy operatora UNION, bezpośrednio w aplikacji.
2) Out-of-band (pozapasmowy) – znacznie mniej popularny typ ataku, podczas którego rezultat zwrócony zostaje innym kanałem komunikacyjnym. Alternatywnymi sposobami przekazania wykradzionych informacji może być żądanie HTTP lub DNS, czy nawet wiadomość mail [6].
3) Inference (inferencyjny) – w odróżnieniu od pozostałych, atak nie zwraca danych z bazy danych w sposób bezpośredni. Niemniej jednak, napastnik wstrzykując kod SQL analizuje czas odpowiedzi oraz reakcje aplikacji. W ten sposób jest on w stanie zgrodzmadzić szereg istotnych informacji np. dotyczących struktury bazy danych.

4. Możliwości obrony

Szerokie spektrum różnych technik ataków SQL Injection przekłada się na sytuację, w której nie istnieje jedno, w pełni skuteczne zabezpieczenie przed nimi [9]. Poszczególne sposoby obrony cechuje różny poziom skuteczności. W celu zapewnienia możliwie jak najwyższego poziomu ochrony aplikacji, zalecane jest zastosowanie kompleksowych rozwiązań obronnych. W kontekście bazodanowych aplikacji internetowych jest to jednoznaczne z koniecznością obrony na poziomie: klienta, serwera aplikacji oraz serwera bazy danych.

Zapytania sparametryzowane

Wiele publikacji [6, 8, 9, 10] dotyczących możliwości obrony przed atakami SQL Injection, zaleca stosowanie zapytań sparametryzowanych (ang. *prepared statement*). Predefiniowanie zapytań SQL gwarantuje większe bezpieczeństwo niż dynamicznie generowane zapytania, zawierające dane od użytkownika [6]. Programista, tworząc szkielet zapytania SQL, uniemożliwia napastnikowi modyfikację jego struktury. Znaki zapytania umieszczone wewnątrz tego szkieletu, pozwalają dołączyć wprowadzone przez użytkownika parametry. Zapytania sparametryzowane nie umożliwiają wstawiania nazw tabeli, kolumn czy też wskaźników sortowania (ASC – rosnąco; DESC – malejąco) [9].

Filtrowanie danych

Najbardziej naturalnym i oczywistym sposobem zapobiegania atakom SQL Injection wydaje się być filtrowanie danych wprowadzanych przez użytkownika. Ograniczając się wyłącznie do sprawdzenia poprawności typu wprowadzonych danych, możliwe jest uchronienie się przed wieloma atakami SQL Injection [11]. Jak wspomniano wcześniej – użycie zapytań sparametryzowanych nie zawsze jest możliwe. Wtedy zalecane jest użycie tzw. białej listy (ang. *white list*), która umożliwia użytkownikowi wprowadzanie wyłącznie dozwolonych znaków.

Istnieje również inny wariant obrony w ramach walidacji danych, polegający na wykorzystaniu tzw. czarnej listy. Za jej pośrednictwem możliwe jest przeszukanie wartości parametrów, pod kątem niedozwolonych znaków i wyrażeń. Wśród nich znaleźć się mogą znaki niedrukowalne (ang. *whitespaces*), znaki specjalne (apostrofy, myślniki, średniki itd.) oraz słowa kluczowe języka SQL. Takie rozwiązanie nie jest optymalne i może generować sytuacje problematyczne. Np. podczas próby rejestracji w systemie użytkownika, którego nazwisko zawiera apostrof. Kluczowe, z punktu widzenia bezpieczeństwa, jest jednoczesne stosowanie walidacji danych po stronie klienta oraz serwera. Ograniczanie się do walidacji po stronie klienta jest złą praktyką, która przekłada się na podatność aplikacji na przeprowadzenie ataku SQL [12].

Procedury składowane

Procedury składowane to struktury, które w ramach jednego wywołania realizują określoną sekwencję instrukcji. Można je porównać do funkcji występujących w językach programowania wysokiego poziomu. Struktury te są przechowywane po stronie serwera bazy danych [6]. Dzięki temu gwarantują zwiększoną wydajność oraz większy poziom bezpieczeństwa [9].

Deklarując własną procedurę składowaną, programista może zdefiniować parametry wejściowe, wymuszając tym samym odpowiedni typ zmiennych. Zabezpiecza to warstwę danych przed wstrzyknięciem kodu przy pomocy parametru pobieranego od użytkownika. Co więcej, używając parametrów wyjściowych (ang. *output*) programista jest w stanie precyzyjnie określić formę zwracanego wyniku. Takie podejście uniemożliwia serwerowi bazy danych wyświetlenie dodatkowych danych napastnikowi.

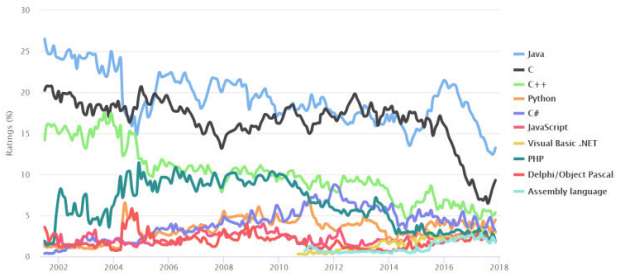
Zasada najmniejszego uprzywilejowania

Wyżej wymienione mechanizmy obronne mają na celu zablokowanie (niedopuszczenie) ataku SQL Injection. W odróżnieniu od nich, zasada najmniejszego uprzywilejowania (ang. *principle of least privilege*) skupia się głównie na minimalizacji strat powstałych w wyniku ataku SQL Injection. Zasada wymusza nadawanie możliwie najmniejszych (ale wystarczających) uprawnień każdemu elementowi systemu realizującego dane zadanie [13]. Jest to dobra praktyka znajdująca zastosowanie nie tylko w systemach teleinformatycznych, ale i w jednostkach wojskowych czy służbach specjalnych.

Stosując zasadę najmniejszego uprzywilejowania w odniesieniu do bazodanowej aplikacji internetowej, nie wystarczy skupić się na odseparowaniu poszczególnych jej modułów. Systemy zarządzania bazami danych umożliwiają tworzenie różnych profili użytkowników, posiadających różne poziomy uprawnień. Administrator systemu, tworząc profil użytkownika, powinien nadać mu możliwie najmniejsze uprawnienia, umożliwiające realizacje funkcjonalności. Klient, który w założeniu, ma mieć dostęp wyłącznie do wyświetlania danych, nie powinien mieć uprawnień do ich modyfikacji, czy usuwania [6].

5. Metodyka badawcza

W celu przeprowadzenia analizy możliwości obrony przed atakami SQL Injection postanowiono stworzyć autorską aplikację internetową. Implementacja przeprowadzona została w taki sposób, aby uzyskać możliwie jak największą podatność aplikacji na ataki typu SQL Injection. W oparciu o przeprowadzone badania literaturowe wyselekcjonowano najpopularniejsze techniki ataków, które szczegółowo przebadano wykorzystując do tego celu niezabezpieczoną wersję aplikacji testowej. W kolejnym kroku, do aplikacji dodano mechanizmy obronne opisane w rozdziale 4. Na koniec ponownie przeprowadzono serię ataków, w obu przypadkach badając „zachowanie" aplikacji oraz operacje przeprowadzone w bazie danych.



Rys. 3. Ranking 10 najpopularniejszych języków programowania na przestrzeni ostatnich 15 lat [14]

Z uwagi na szeroki zakres zagadnienia jakim są ataki SQL Injection, konieczne było zawężenie obszaru badawczego. Głównym kryterium jakim kierowano się podczas wyboru narzędzi była ich popularność. Aplikacja zaimplementowana została w technologii JSP (ang. *JavaServer Pages*) z wykorzystaniem serwera aplikacyjnego Apache Tomcat, oraz serwera bazodanowego MySQL. Java to obiektowy język programowania, utrzymujący się w czołówce rankingów popularności na przestrzeni ostatnich 15 (rysunek 3.) [14].

Przed przystąpieniem do przeprowadzenia eksperymentu, na podstawie przeprowadzonych badań literaturowych, sformułowano następujące hipotezy badawcze:

1. Prewencja umożliwia obronę przed najpopularniejszymi technikami ataków SQL Injection.
2. Stosowanie zasady najmniejszego uprzywilejowania minimalizuje straty.
3. Zapytania parametryzowane, spośród dostępnych rozwiązań, cechują się najwyższym poziomem bezpieczeństwa.

6. Eksperyment

W tym rozdziale przedstawione zostały różne typy ataków SQL Injection. Opisy poszczególnych (wybranych) technik zostały uzupełnione o konkretne warianty poszczególnych ataków. Pierwsza faza polegała na przeprowadzeniu serii ataków na wersję aplikacji całkowicie podatnej na ataki SQL Injection. Co więcej, każdy atak w tej fazie, przeprowadzony został z poziomu użytkownika o nieograniczonych uprawnieniach.

Tautologia

Cel ataku: Wyświetlanie danych, Sprawdzanie podatności parametrów, Nieuprawniona autoryzacja

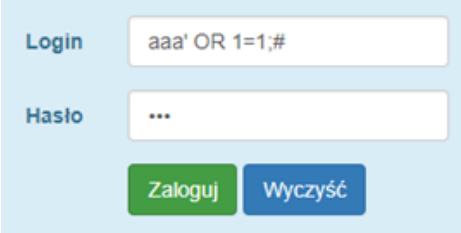
Atak polega na modyfikacji oryginalnej treści zapytania do postaci, w której staje się ono prawdziwie niezależnie od wartości pozostałych parametrów. W logice tautologia, to zdanie prawdziwe na mocy swojej budowy. W kontekście składni języka SQL oznacza to, że niezależnie od warunków zawartych w klauzuli WHERE ich wypadkowa wartość zawsze będzie równa prawdzie. Zdecydowana większość aplikacji internetowych funkcjonujących w sieci WWW, w procesie autoryzacji wymaga od użytkownika podania loginu i hasła. Następnie na podstawie wprowadzonych przez użytkownika danych odbywa się jego identyfikacja.

Przykład 1. Zapytanie SQL – autoryzacja

```
SELECT * FROM uzytkownicy WHERE
 login = 'podanyLogin' AND
 haslo = 'podaneHaslo';
```

Zapytanie realizujące pobranie rekordu z tabeli *uzytkownicy* o podanym przez użytkownika loginie i haśle, przedstawiono na przykładzie 1. W przypadku podania poprawnego loginu oraz hasła – czyli wartości dokładnie takich, które znajdują się w odpowiednich kolumnach tabeli *uzytkownicy*, identyfikacja zostanie zakończona pomyślnie. W przypadku podania niepoprawnych danych w formularzu logowania – zapytanie nie zwróci żadnego rekordu – brak użytkownika o podanym loginie i haśle w tabeli *uzytkownicy*.

Analizując strukturę powyższego zapytania należy zauważyć, że fragment „*SELECT * FROM uzytkownicy*" zostanie wykonany tylko i wyłącznie w momencie, gdy prawdziwy będzie warunek występujący po klauzuli WHERE. Umiejętna manipulacja oryginalnego kodu SQL, z wykorzystaniem operatora logicznego OR oraz dopisaniu do niego zawsze prawdziwego warunku, pozwala obejść proces autoryzacji.



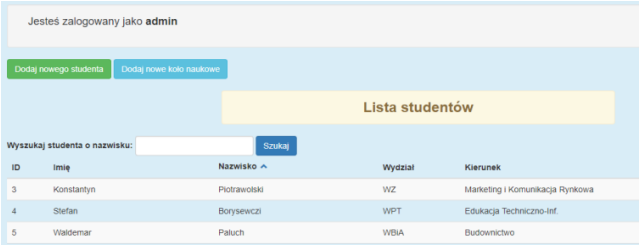
Rys. 4. Formularz logowania – tautologia

Możliwy wariant takiego ataku przedstawiono na rysunku 4., natomiast przykład 2. zawiera treść zapytania generowanego dynamicznie w oparciu o wprowadzone parametry.

Przykład. 2. Zapytanie SQL – tautologia

```
SELECT * FROM uzytkownicy
WHERE login='aaa' OR 1=1;
  #' AND haslo='xxx';
```

W rezultacie pobrane z bazy danych zostaną wszystkie rekordy z tabeli *uzytkownicy*, a aplikacja wyświetli stronę glowna.jsp, co przedstawiono na rysunku 5. Jak widać, napastnik dostał nieuprawniony dostęp do listy wszystkich studentów, ponieważ został zalogowany na konto administratora.



Rys. 5. Tautologia – rezultat ataku na formularz logowania

UNION SELECT

Cel ataku: Nieuprawniona autoryzacja, Wyświetlenie danych, Odtwarzanie struktury bazy danych

Język SQL umożliwia łączenie wyników dwóch lub więcej zapytań, do jednej postaci wynikowej. Struktura takiego zapytania została przedstawiona na przykładzie 3.

Przykład 3. Zapytanie SQL – struktura UNION SELECT

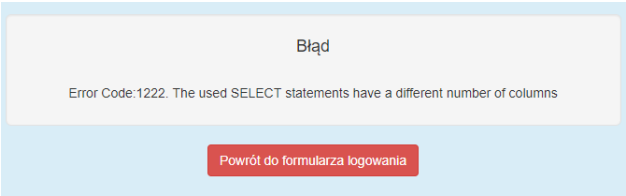
```
SELECT kol_1, kol_2,(...),kol_X FROM tab_1
UNION
SELECT kol_1, kol_2,(...),kol_X FROM tab_2;
```

Aby zapytanie z operatorem UNION było poprawnie wykonane, konieczne jest spełnienie dwóch warunków:

1. Wszystkie zapytania połączone operatorem UNION muszą pobierać dane z dokładnie tej samej liczby kolumn,

2. typy danych odpowiadających sobie kolumn muszą być takie same lub kompatybilne.

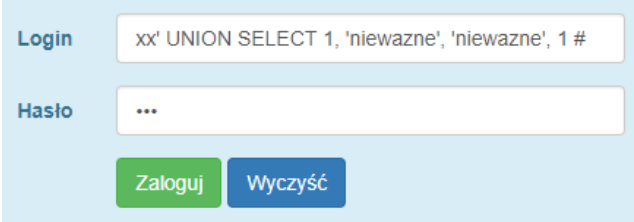
Jeśli warunki nie zostaną spełnione, zwrócony zostanie wyjątek, a zapytanie nie wykona się. Treść wiadomości wyświetlanej w aplikacji po podaniu niewłaściwej liczby kolumn przedstawiono na rysunku 6.



Rys. 6. Błąd – UNION SELECT – niepoprawna liczba kolumn

Z treści wiadomości nie wynika wprost informacja o liczbie kolumn z oryginalnego zapytania. Oznacza to, że w celu ustalenia tej liczby, napastnik musi zastosować metodę prób i błędów. To, w większości przypadków, oznacza konieczność generowania dużej liczby zapytań. Hackerzy bardzo często automatyzują ten proces pisząc własne skrypty lub korzystając z istniejących narzędzi [6].

Innym zastosowaniem techniki UNION SELECT jest obejście procesu autoryzacji. Omawiając atak z użyciem tautologii przybliżony został przebieg procesu autoryzacji użytkownika. Odbywa się ona na podstawie wprowadzonych przez użytkownika parametrów: login oraz hasło.



Rys. 7. Formularz logowania – UNION SELECT – obejście autoryzacji

Na rysunku 7. przedstawiono możliwy wariant ataku UNION SELECT. Wprowadzając frazę „*xx' UNION SELECT 1,'niewazne','niewazne', 1#*" w polu *login*, dokonana zostanie modyfikacja oryginalnego zapytania SQL, do postaci przedstawionej na przykładzie 4.

Przykład 4. UNION SELECT – obejście autoryzacji – zapytanie SQL

```
select * from uzytkownicy where login='xx'
UNION SELECT 1, 'niewazne', 'niewazne', 1 #' and haslo='';
```

W wyniku tego, zamiast pustej listy zwrócony zostanie rekord o wartościach: 1, niewazne, niewazne, 1, co przedstawiono na rysunku 8. W konsekwencji napastnik zostanie zalogowany do systemu na konto z poziomem uprawnień administratora systemu.

	idUzytkownik	login	haslo	rodzajUprawnien
	1	niewazne	niewazne	1

Rys. 8. UNION SELECT – autoryzacja, wynik zapytania

Piggy-backed

Cel ataku: Wstawianie i modyfikowanie danych, Wyświetlanie danych, Wykonanie poleceń zdalnych

Opisane wyżej formy ataków skupiały się na modyfikacji kodu oryginalnego zapytania. Technika *piggy-backed* umożliwia wstrzyknięcie dodatkowego zapytania SQL, które zostaje wykonane bezpośrednio po oryginalnym zapytaniu. Szansa powodzenia tego ataku, jest uwarunkowana przede wszystkim konfiguracją serwera bazy danych [11]. Domyślnie MySQL nie umożliwia wykonywania serii zapytań [15]. Niemniej jednak, podczas nawiązywania połączenia z bazą danych, możliwe jest odblokowanie tej opcji. Przykład 5. przedstawia nawiązanie połączenia wraz z odblokowaniem opcji wykonywania wielu zapytań (ang. *multiple queries*).

Przykład 5. Odblokowanie opcji wykonywania wielu zapytań

```
("jdbc:mysql://localhost:3306/mojabaza?
allowMultiQueries=true", "root", "root");
```

Technika piggy-backed daje napastnikowi dużą elastyczność. Zakres możliwych działań jest uwarunkowany rodzajem uprawnień użytkownika, których dokonuje się konfigurując serwer bazy danych. Jeżeli użytkownik posiada uprawnienia do modyfikowania czy usuwania danych, może się to wiązać z nieodwracalną utratą danych. Poniżej przedstawiono możliwy wariant ataku piggy-backed.

Przykład 6. Piggy-backed – usunięcie tabeli *studenci*

```
SQLProjection/edytujstudenta.jsp?id=7'; drop table studenci; #
```

Przykład 6. przedstawia fragment kodu, który wstrzyknięto za pośrednictwem adresu URL. W rezultacie, zapytanie SQL przekazane do bazy danych wyglądać będzie tak, jak zaprezentowano to na przykładzie 7.

Przykład 7. Piggy-backed – usunięcie tabeli *studenci* - zapytanie SQL

```
SELECT * FROM studenci
WHERE idStudent ='7';
drop table studenci; #';
```

Powyższy przykład bardzo dobrze obrazuje zagrożenie, jakie niesie za sobą umożliwienie realizacji więcej niż jednego zapytania. Po wykonaniu oryginalnie zaprojektowanego zapytania wyświetlającego studenta o podanym identyfikatorze, cała tabela *studenci* zostanie usunięta. Warto zaznaczyć, że napastnik po znaku średnika, może dopisywać kolejne zapytania.

Blind SQL Injection

Cel ataku: Sprawdzanie podatności parametrów, Odtwarzanie struktury bazy danych, Wyświetlanie danych

Tzw. „ślepy atak przeprowadzany jest najczęściej, gdy napastnik nie dysponuje żadnymi informacjami dotyczącymi bazy danych (system zarządzania bazą danych, wersja, struktura itd.) [6]. Z reguły jest to mozolny i długotrwały proces, polegający na wstrzykiwaniu kolejnych zapytań zbudowanych przy pomocy operatorów logicznych (AND/OR). Na podstawie zwracanych rezultatów (prawda lub fałsz) napastnik sukcesywnie gromadzi kolejne informacje, począwszy od sprawdzenia czy wstrzykiwanie kodu SQL w danym systemie jest w ogóle możliwe. W przypadku standardowego ataku SQL Injection, wynik zapytania jest wyświetlany przez przeglądarkę. Natomiast stosując „ślepy" napastnik nie dostaje wyników w sposób jawny. Poniżej przedstawiono możliwe warianty tzw. ślepego ataku.

Error based

Korzystając z konstrukcji przedstawionej na przykładzie 8. możliwe jest ustalenie numeru wersji serwera bazy danych, z wykorzystaniem tzw. ślepego ataku bazującego na błędach.

Przykład 8. Konstrukcja do sprawdzenia numeru wersji serwera

```
3' AND substring(version(),1,1)=4;#
```

W rezultacie, wykonany zostanie następujący kod SQL – przykład 9.

Przykład 9. Blind SQL – zapytanie SQL sprawdzające numer wersji serwera

```
SELECT * FROM studenci
WHERE idStudent ='3'
AND substring(version(),1,1)=5;# '";
```

Jeśli nastąpi przekierowanie do widoku studenta o identyfikatorze 3, to numer wersji serwera bazy danych zaczyna się od cyfry '5'. Jeśli nie, napastnik może manipulować cyfrą wersji, do momentu ustalenia właściwej.

Time based

Podobny mechanizm występuje w przypadku tzw. ślepych ataków bazujących na czasie odpowiedzi. Napastnik w analogiczny sposób ustala istotne informacje, tym razem analizując czas odpowiedzi. Funkcja *BENCHMARK* umożliwia realizację danej instrukcji określona liczbę razy. Stosując technikę tzw. ślepego ataku bazującego na czasie odpowiedzi, hackerzy bardzo często wykorzystują funkcję SLEEP, który „usypia" serwer na określony czas (podany w sekundach). Możliwy wariant takiego ataku przedstawiono na przykładzie 10.

Przykład 10. Blind SQL – zapytanie SQL sprawdzające numer wersji serwera

```
1 UNION SELECT IF
(SUBSTRING(version(),1,1)=5,
BENCHMARK(5000000,
MD 5(CHAR(1))),null),null
```

Modyfikacja URL

Bardzo często parametry przesyłane przez adres URL są kodowane przy użyciu kodowania procentowego (ang. *percent encoding*). Każdy bajt zamieniony zostaje na jego dwucyfrowy odpowiednik, poprzedzony znakiem procenta (%). Korzystając z dostępnych w Internecie źródeł, możliwe jest zakodowanie fragmentu kodu SQL np. do postaci przedstawionej na rysunku 10.

http://localhost:8084/SQLProjection/delete.jsp?
id=25%35%27%20%4F%52%20%27%35%27%3D%27%35
||
25' OR '5' = '5

Rys. 10. Wstrzyknięcie kodu – URL, kodowanie procentowe

W rezultacie do zapytania realizującego (w założeniu) usunięcie studenta o podanym identyfikatorze (25), dodana zostanie fraza „*OR '5' = '5* ". Przedstawia to przykład 11.

Przykład 11. Wstrzyknięcie kodu – URL, zapytanie SQL

```
DELETE FROM studenci WHERE idStudent ='25' OR '5' = '5';
```

Druga faza eksperymentu przeprowadzona została w sposób analogiczny do fazy pierwszej. Poniżej uwzględniono poszczególne mechanizmy obronne.

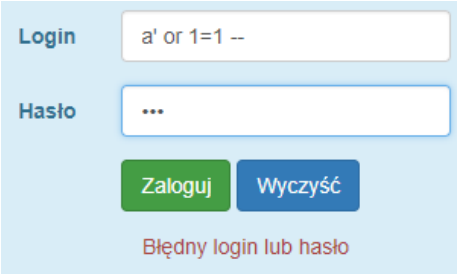
Zapytania parametryzowane

Na przykładzie 12. zaprezentowano fragment kodu źródłowego aplikacji, realizującego autoryzacje użytkownika (logowanie). W odróżnieniu od wcześniejszego rozwiązania, do wykonania zapytania wykorzystano zapytanie sparametryzowane. Dzięki niemu zapytanie nie jest tworzone na zasadzie konkatenacji ciągu znakowego. Szkielet zapytania wyposażony w znaki zapytania (symbole zastępcze), umożliwia wstawienie wartości pobranych od użytkownika.

Przykład 12. Zapytanie sparametryzowane – autoryzacja użytkownik

```
String sqlQuery = "SELECT * FROM uzytkownicy
WHERE login = ? and haslo = ? ";
PreparedStatement pstmt = conn.prepareStatement(sqlQuery);
pstmt.setString(1, login);
pstmt.setString(2, haslo);
ResultSet rs = pstmt.executeQuery();
```

Na załączonym przykładzie zaobserwować można również dwuetapowy mechanizm wykonania zapytania SQL. Funkcja *prepareStatement* realizuje wstępne wykonanie zapytania, do którego następnie dołączone są brakujące wartości parametrów. W ten sposób uniemożliwiona zostaje potencjalna ingerencja użytkownika w strukturę zapytania SQL.



Rys. 11. Zapytania parametryzowane – nieudana próba ataku przy użyciu tautologii

Procedury składowane

Stosowanie procedur składowanych to dobra praktyka, która pozwala poprawić nie tylko bezpieczeństwo, ale i wydajność aplikacji. Stosowanie parametrów wejściowych pozwala zadbać o kompatybilność typów danych. Dodatkowo pozwala zablokować próbę wykonanie zapytania w przypadku wprowadzenia np. zbyt długiego ciągu znaków dla parametru typu *varchar*. Jest to bardzo istotne, ponieważ wstrzyknięcia kodu SQL z reguły są dosyć rozbudowane

Przykład 13. Procedura wyszukująca studenta o podanym nazwisku

```
CREATE DEFINER=`root`@`localhost`
PROCEDURE `wyszukiwarkaAdmin`(IN vnazwisko varchar(20))
BEGIN
    SELECT idStudent, imie, nazwisko, wydzial, kierunek
    FROM studenci WHERE nazwisko LIKE CONCAT('%',vnazwisko,'%');
END
```

Przykład 13. przedstawia procedurę realizującą wyszukanie studenta o podanym nazwisku. Zapis „(IN vnazwisko varchar(20))" oznacza, że procedura przyjmuje jeden parametr wejściowy (pobierany od użytkownika), typu varchar, nie dłuższy niż 20 znaków. Podczas próby podania

dłuższego ciągu znaków, próba wykonania procedury zakończy się niepowodzeniem.

Docelowo, warto zadbać o zdefiniowanie parametrów wyjściowych, ich typu i zakresu. Jest to dodatkowe zabezpieczenie, które udaremni ewentualną próbę wyprowadzenia dodatkowych danych.

Filtrowanie danych

Według przeglądu literatury z zakresu ataków SQL Injection, wynika, że filtrowanie danych to mechanizm obronny niegwarantujący maksymalnego poziomu zabezpieczeń. Niemniej jednak zaleca się stosowanie filtrowania. Należy pamiętać o tym, że ograniczenie się do walidacji danych po stronie klienta, nie jest wystarczające. Znacznie bardziej znacząca jest walidacja po stronie serwera. Taki stan rzeczy wynika z tego, że istnieją narzędzia, które umożliwiają spreparowanie żądania HTTP.

Przykład 14. Funkcja walidująca długość parametru

```
private boolean czyZwalidowany1(String parametr) {
        return parametr.length() > 50 ? false : true;
    }
```

Przykład 14. przedstawia kod funkcji odpowiedzialnej za sprawdzenie, czy wartość podanego parametru jest typu string (tekstowego) i jest nie dłuższa niż 50 znaków. Oprócz tego, konieczne jest „przeszukanie" parametru pod kątem słów kluczowych języka SQL, czy znaków specjalnych (np. #, -, --, spacja, ‘,”). Funkcja zaprezentowana na przykładzie 15. jest bardziej rozbudowana. Dzięki niej walidowana jest nie tylko długość, ale i zawartość przekazywanego parametru. Dopuszcza ona wprowadzenie wyłącznie (małych i dużych) liter. Jest to przykład zastosowania tzw. białej listy.

Przykład 15. Funkcja walidująca długość i zawartość parametru

```
private boolean czyZwalidowany2(String parametr) {
Pattern p = Pattern.compile("^[a-zA-Z]{0,20}$");
Matcher m = p.matcher("parametr");
return m.find() ? true : false;
}
```

Maskowanie błędów

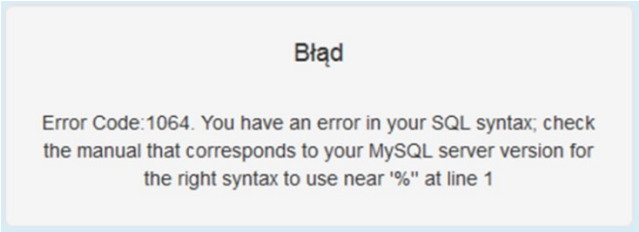
Implementując logikę aplikacji należy mieć świadomość, że dla potencjalnego napastnika treść zwracanych błędów posiada znaczący potencjał informacyjny. Aplikacja w wersji produkcyjnej powinna zwracać kompletne treści komunikatów (wraz z kodem błędu) wyświetlanych w sytuacjach wyjątkowych. Takie podejście pomaga programiście szybciej rozwiązywać problemy występujące podczas tworzenia oprogramowania.

Ważne, aby przed wdrożeniem aplikacji zadbać o umiejętne maskowanie błędów. Należy kierować się tym, aby użytkownik uzyskiwał taką porcję informacji, jaka jest mu potrzebna do realizacji danego zadania. Przykładowo, w sytuacji niepowodzenia autoryzacji (logowania) nie należy precyzować, który parametr jest nieprawidłowo (login, hasło, czy oba).

Poniżej zaprezentowano odpowiedź aplikacji podczas próby zalogowania z użyciem parametrów:

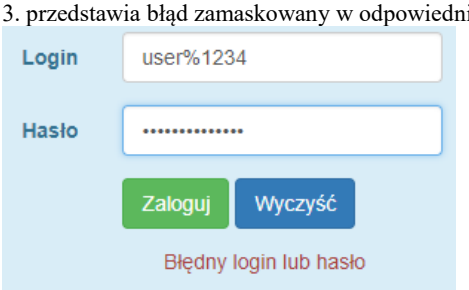
- Login: user%1234
- Hasło: xxxxxxxx

Na rysunku 12. zaprezentowano błąd wyświetlany w aplikacji podatnej na ataki SQL Injection.



Rys. 12. Błąd zwracany przez aplikację (niezamaskowany)

Rysunek 13. przedstawia błąd zamaskowany w odpowiedni sposób.



Rys. 13. Błąd zwracany przez aplikację (zamaskowany)

7. Wnioski

Przeprowadzone badania wykazały prawdziwość wszystkich hipotez badawczych sformułowanych w rozdziale 5. Spośród wszystkich wykorzystanych mechanizmów obronnych, najwyższą skutecznością wykazały się zapytania sparametryzowane oraz procedury składowane.

Istotnym wnioskiem uzyskanym w wyniku analizy jest fakt, że nieprawidłowa obsługa błędów stanowi podatność aplikacji na ataki SQL Injection. Implementując aplikację należy pamiętać o odpowiednim maskowaniu zwracanych błędów. Takie działanie w znaczący sposób utrudni działanie napastnikowi.

Na podstawie przeprowadzonych badań potwierdzono, że zasada najmniejszego uprzywilejowania pozwala dokonać minimalizacji strat. Co więcej, zgodnie z oczekiwaniami, pozwoliła ona zablokować jedynie niektóre ataki. uniemożliwiając przeprowadzenie modyfikacji danych, bez wymaganych uprawnień. W wyniku dyskusji, opracowano przykładowy model uprawnień użytkowników w systemie. Model przedstawiono w tabeli 1.

Tabela 1. Model uprawnień użytkowników systemu

Typ konta	Uprawnienia
użytkownik	SELECT
edytor treści	SELECT, INSERT, UPDATE
moderator serwisu	CREATE, DROP, ALTER, EXECUTE, DELETE
administrator	GRANT, BACKUP, CREATE USER, SHUTDOWN

Niewątpliwie przeprowadzone badania nie wyczerpują zagadnienia. W niniejszym opracowaniu skupiono się na analizie statycznych metod obrony przed atakami SQL Injection. Należy pamiętać o dynamicznych metodach ochrony aplikacji. Monitorowanie pracy serwera bazy danych z całą pewnością gwarantuje zwiększenie poziomu bezpieczeństwa. Prowadzona w czasie rzeczywistym detekcja i obsługa incydentów bezpieczeństwa w wielu

przypadkach pozwala zapobiec naruszeniu dostępności, poufności i integralności danych.

Literatura

[1] How Was SQL Injection Discovered? https://www.esecurityplanet.com/network-security/how-was-sql-injection-discovered.html [20.11.2017]

[2] Top 10 Attack Techniques – 2015 vs. 2014 http://www.hackmageddon.com/2016/01/11/2015-cyber-attacks-statistics [12.11.2017]

[3] Co oferuje nam OWASP? http://websecurity.pl/co-oferuje-nam-owasp [15.11.2017]

[4] OWASP: The 10 Most Critical Web Application Security Risks, https://www.owasp.org/images/b/b0/OWASP_Top_10_2017_RC2_Final.pdf

[5] Norma PN-SIO/IEC-17799:2005 Technika informatyczna. Praktyczne zasady zarządzania bezpieczeństwem informacji, PKN, 2007.

[6] J. Clarke, SQL Injection Attacks and Defense, Syngress Publishing, Inc., 2012.

[7] SQL Injection through HTTP Headers, http://resources.infosecinstitute.com/sql-injection-http-headers [14.11.2017]

[8] Amirmohammad Sadeghian, Mazdak Zamani, Suhaimi Ibrahim, SQL Injection is Still Alive: A Study on SQL Injection Signature EvasionTechniques, 2013 International Conference on Informatics and Creative Multimedia, 2013.

[9] OWASP: SQL Injection Prevention Cheat Sheet, https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet [17.11.2017]

[10] Chandershekhar Sharma, S.C. Jain, Analysis and Classification of SQL Injection Vulnerabilities and Attacks on Web Applications, Konferencja: International Conference on Advances in Engineering & Technology Research, ICAETR – 2014.

[11] William G.J. Halfond, Jeremy Viegas, Alessandro Orso, A Classification of SQL Injection Attacks and Countermeasures, Proceedings of the International Symposium on Secure Software Engineering, 2006.

[12] How to: Protect From SQL Injection in ASP.Net, https://msdn.microsoft.com/en-us/library/ff648339.aspx [16.11.2017]

[13] Microsoft Sercurity Overview, https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/security-overview [22.11.2017]

[14] Tiobe Index for November 2017, https://www.tiobe.com/tiobe-index/ [23.11.2017]

[15] M. Dymek, M. Nycz, A. Gerka, Analiza statycznych metod obrony przed atakami SQL, ZESZYTY NAUKOWE POLITECHNIKI RZESZOWSKIEJ 294, Elektrotechnika 35 RUTJEE, z. 35 (2/2016), kwiecień-czerwiec 2016, s. 47-56.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 158-161
Wysłane: 2017-11-29
Przyjęte: 2017-12-04

Porównanie efektywności szkieletów AngularJS i Meteor

Oleksandr Chornyi*, Marek Miłosz

Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono rezultaty porównania szybkości tworzenia kodu i ładowania projektów w różnych frameworkach w celu zbadania, który z nich wydaje się być najlepszym wyborem w dłuższej perspektywie czasu. Analizę przeprowadzono pod kątem badania eksperymentalnego, wykonania oprogramowania przykładowego projektu oraz badania literaturowego. Przeprowadzone porównanie pozwoli wskazać framework o lepszej wydajności i prostszy w wykorzystaniu.

Słowa kluczowe: AngularJS; Meteor; framework; porównanie

*Autor do korespondencji.

Adres e-mail: oleksandr.chornyi@pollub.edu.pl

Effectiveness Comparison of the AngularJS and Meteor frameworks

Oleksandr Chornyi*, Marek Miłosz

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

Abstract. This article presents the results of comparing code developing speeds and project loads across different frameworks to explore which of them seems to be the best choice in the long time. The analysis was carried out in terms of the exploratory study, the design of the sample project and the literature review. Comparison will make it possible to point to a better performance framework and prolong its use.

Keywords: AngularJS; Meteor; framework; comparison;

*Corresponding author.

E-mail address: oleksandr.chornyi@pollub.edu.pl

1. Wstęp

Szkielety JavaScript nabrały ogromnej popularności wśród web-developerów, ponieważ pomagają one efektywniej i szybciej tworzyć strony internetowe i pisać ładny, czysty kod nie wykorzystując metod które zajmują dużo czasu. Struktura Javascript frameworków pozwala zmniejszyć czas opracowania aplikacji dzięki automatyzacji wielu modeli dla konkretnie wykorzystywanych celi. Pomimo tego frameworki pozwalają projektantowi pisać bezbłędny i czytelny kod. Framework implikuje operacje i dzięki temu framework pozwala programistom pracować mniej, a czynić więcej.

Najczęściej frameworki pracują szybko dzięki temu, że wykonują tylko to, co wymaga od ich użytkownik, podczas gdy CMS wykonuje o wiele więcej obliczeń i zapytań takich jak "on/off moduł", "if on" i inne podobne do tego funkcje.

Frameworki JavaScript dają programiście wiele plusów i przewag odpowiadających postawionemu zadaniu, lecz jednocześnie sprawiają niedogodność swoim mnogim wyborem, ponieważ już w chwili obecnej istnieje duża liczba różnych frameworków i każdy z nich ma swoje plusy jak i minusy. Co można zrobić w jedym frameworku, tego już się nie da w drugim i na odwrót. A więc należy dobrze pomyśleć i wybrać ten który odpowiada nam najlepiej.

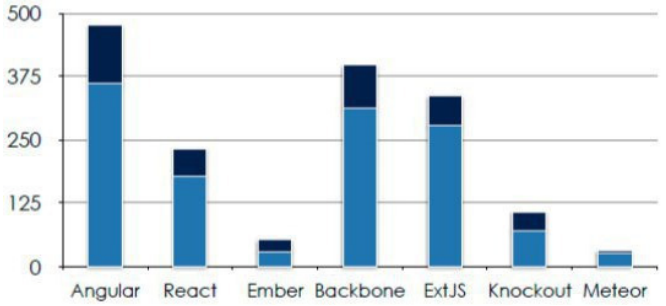
2. Framework AngularJS i framework MeteorJS

AngularJS został stworzony i opracowany przez dwóch programistów – Miszko Heweri i Adam Abronson w 2009 roku w celu zapewnienia usługi danych JSON i ułatwienia tworzenia aplikacji [1, 2]. Przed wydaniem AngularJS jako biblioteki open source dla użytkowników, był to pomysł na biznes umieszczony w domenie "GetAngular.com" [3]. Kolejna wersja Angular2 została wydana siedem lat później w 2016 r. I dopiero rok później, w kwietniu 2017 r. ukazał się nowy Angular4 [3].

Z kolei MeteorJS został stworzony 2 lata później, czyli w 2011 roku pod nazwą Skybreak, lecz dopiero na początku stycznia 2012 roku został użyty przez wszystkich programistów w zwykłym wyglądzie [4-6]. Oprócz tego MeteorJS można nazwać full - stuck frameworkim - to oznacza, że może on pracować tak samo po stronie klienta, jak i po stronie serwera.

Różnice pomiędzy frameworkami będą przedstawione w tablicy 1, ale nie warto ich dosłownie porównywać jako w całości jednakowe frameworki, ponieważ MeteorJS - to full - stack framework, a AngularJS - to mvc po stronie klienta. Jednak główne aspekty ich pracy można porównać i spróbować wyznaczyć lidera [7, 8-10].

Na rynku pracy AngularJS jest bardzo poszukiwanym językiem programowania. Skalę ofert pracy ukazuje poniższy rysunek, który został sporządzony i zaprezentowany na stronie internetowej freelancerów HeadHunter w 2016 r. [11].



Rys. 1. Wykres popytu i podaży na języki programowania w firmach zajmujących się wytwarzeniem oprogramowania. [11]

Jasnoniebieski kolor przyporządkowuje ilość osób poszukujących pracy, a granatowy kolor oznacza ilość zaproponowanych stanowisk pracy w poszczególnych językach programowania. Jak widać Meteor nie cieszy się dużym zainteresowaniem pracodawcór jak również przyszłych pracowników.

W tabeli 1 widać, że w niektórych momentach AngularJS przegrywa z MeteorJS, lecz nie można zapominać o tym, że dane Framework niezupełnie są równorzędne.

Tabela 1. Przewagi AngularJS i Meteor [3]

	angular js	**meteor js**
Full-stack framework	Nie, wyłącznie MVC po stronie klienta	Tak
Backend	Obojętnie	Node.js
Console Utility	Niema	Jest
Dynamiczny związek html z danymi po stronie klienta	Tak	Tak
Render html po stronie serwera	Nie	Tak, nie natywny
npm pakiety	Jest możliwość połączyć w browserify	Przez pakiety własnego menegera pakietów
Wielokrotne użycie między stroną klienta i serwerem	Niskie	Wysokie
REST API	Nie, ale jest możliwość dodania	Jest
DB	Jakakolwiek	Jakakolwiek
Odnowa dodatku bez uruchamiania ponownie	Nie	Tak html, css, js
Strona internetowa	Angular.io	Meteor.com

3. Metodyka badawcza

Podstawą metodologii badań tej pracy dyplomowej będzie opracowania naukowe na temat opracowania i napisania web-aplikacji. Zostaną rozpatrzone plusy i minusy każdego z frameworków względem siebie.

Niemałe znaczenie w praktycznej części tego badania ma uogólnianie zasad napisania kodu i szybkości pracy każdego z frameworków w równorzędnych warunkach.

Metodą porównania danych frameworków AngularJS i MeteorJS będą następujące punkty:

- Podłączanie frameworków do projektu
- Struktura napisania kodu(jednakowej funkcji)
- Szybkość ładowania strony
- Szybkość wykonania tej funkcji.

Porównanie szybkości ładowania strony będzie wykonane przy pomocy wzoru:

$$S = \frac{Z1 + Z2 + Z3 + \cdots + Z4}{K} \tag{1}$$

gdzie: Z – czas ładowania strony, K – liczba eksperymentów.

Średnia liczba napisanych linii kodu przedstawia się następująco:

$$E = \frac{\frac{S_{js}}{F_{js}} + \frac{S_{ts}}{F_{ts}} + \frac{S_{html}}{F_{html}}}{X} \tag{2}$$

gdzie: S_{js}- liczba napisanych linii, *js* - rozszerzenie pliku, F_{ts} - liczba plików z rozszerzeniem .ts, *X* – liczba plików.

Badania wykonane zostały na komputerze o następujących parametrach technicznych:

- Processor Intel Core i5-650 @ 3.20 GHz (4 Cores);
- RAM: 6 Gb.
- Windows 10

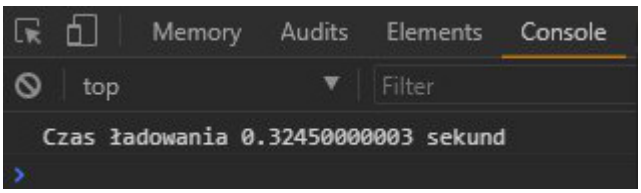
W celu wykonania eksperymentu będzie wykonane ładowanie projektów już ze statyczne dodanymi zadaniami w liczbie 5000 i 15000. Rezultaty będą zapisane do tablic i po wszystkich testach będą obliczone średnie szybkości ładowania stron.

Szybkość ładowania stron będzie realizowana za pomocą funkcji JavaScript:

Przykład 1. Funkcja licząca szybkość ładowania stron

```
var startTime = new Date();
function showElapsedTime() {
var testSiteUrl = location.href;
var testSiteString = String(testSiteUrl).slice
(testSiteUrl.indexOf("www"));
var endTime = new Date();
var platform = navigator.platform;
var msgString = "Время загрузки „ +
Number(elapsedTime/1000) + „секунд ";
document.getElementById(„vremia").innerHTML =
msgString;}onload = function() {showElapsedTime();}
```

Rezultaty powyższego kodu bedą wyświetlane w konsoli, którą włączamy w przeglądarce. Powyższy kod wykona się po załadowaniu strony.



Rys. 2. Wynik wykonanego skryptu w konsoli Google Chrome.

4. Realizacja badań i rezultaty

W tym rozdziale będzie opisane stworzenie projektu na AngularJS i Meteor. Celem projektu było stworzenie listy spraw do załatwienia - "To do list". Do stworzenia aplikacji zostanie wykorzystany AngularJS i dodatkowe utility-y takie jak stworzenie samego DOM poprzez zaprogramowanie HTML CSS (Cascading Style Sheets) oraz MeteorJS.

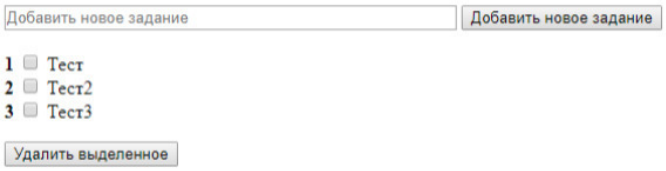
Przykład 2. Funkcje dodawania, usunięcia, tasku AngularJS

```
var todo_list = angular.module('TodoListApp'), [])
.controller('ListController', function($scope){
$scope.addItem = function (){
$scope.item.push({detauk: $scope.newItem.detail,
stat: 'Active'});
}
$scope.deleteItem = function(item){
var index = $scope.item.indexOf(item);
$scope.item.splice(index, 1);
}
$scope.completeItem = function (item){
item.stat = "Complete";
}
```

Przykład 3. Funkcje dodawania, usunięcia taska Meteor

```
'submit from': function(event){
event.preventDefault();
var todoName = $('[name="todoName"').val();
Todos.insert({
name: todoName,
completed: false,
createdAt: new Date()
});
}
'click .delete-todo': function(event){
event.preventDefault();
var documentId = this._id;
}
```

Powyższe listingi, pomimo róznic w składni wykonują się tak samo i mają identyczną funkcjonalność.



Rys. 2. Zrzut ekranowy prezentuje wykonanie obydwu funkcij.

Budowanie aplikacji w Meteorze wymaga krótszych linii kodu, ale działąca aplikacja wykorzystuje więcej tworzących ją plików czego nie można powiedzieć o Angularze.

Za pomocą frameworków AngularJS i MeteorJS istnieje możliwość stworzenia praktycznie identycznych web - aplikacji, typu landing page. Frameworki te są potężnymi narzędziami do tworzenia web - aplikacji, które ułatwiają programowanie jeśli porównujemy je z innymi tradycyjnymi frameworkami. Wybór frameworków najczęściej zależy od postawionych zadań i wymogów zleceniodawcy projektu. Należy patrzeć na oddzielne, konkretne przypadki i wybierać spośród nich najlepszy. Kiedy musimy szybko wykonać web-opracowanie gdzie trzeba użyć frameworków JavaScript i nie ma potrzeby wykorzystywania funkcji back-end-owych oraz pokazujących front-end. W takim wypadku oczywisty wybór pada na AngularJS.

AngularJS dość często nazywają MVW (Model - View - Whatever) frameworkiem i jego zaletą jest szybkie tworzenia kodu i natychmiastowe odzwierciedlenie przemian na stronie back - end po stronie klienta. Dzięki jego zasłużeniu w technice nazywają go najbardziej popularnym frameworkiem JavaScript dla opracowania jednostronicowych stosowań (SPA Single - Page - Applications) i może on pochlubić się największym wsparciem społeczności.

Tabela 2. Ilość plików i linii kodu wykorzystanych przez AngularJS

Rozszerzenie plika	Liczba plików	Liczba linii
.js	4	576
.ts	3	201
HTML	1	179
Suma	8	956

Tabela 3. Ilość plików i linii kodu wykorzystanych przez Meteor

Rozszerzenie plika	Liczba plików	Liczba linii
.js	3	423
.ts	2	152
HTML	1	115
Suma	6	670

Tabela 3. Porównanie liczby linii kodu AngularJS i MeteorJS

Rozszerzenie plika	Angular	Meteor
.js	144	141
.ts	67	76
HTML	179	115
Średnia	390	332

W tabeli 3 są zawarte wyniki liczby napisanych linii kodu dla każdego rozszerzenia plików i za pomocą formuły opisanej wyżej uzyskujemy ich średnią. Możemy zauważyć, że w Angularze wykorzystano więcej linii kodu i w perspektywie zajmowało to więcej czasu przy tworzeniu aplikacji.

W eksperymencie z wymiarami szybkości ładowania stron zrobionych zostało kilka pomiarów. Zadanie wykonane było na różnych przeglądarkach, ale na tym samym komputerze.

Tabela 4. Szybkość ładowania projektu bez statycznych tasków

Framework/Przeglądarka	Chrome	FireFox	Safari
AngularJS	0.323s	0.340s	0.331s
Meteor	0.396s	0.401s	0.339s

Ponieważ aplikacje nie mają określonego obciążenia, różnica czasu ładowania jest nieco inna. Jeśli spojrzymy na wyniki w kategoriach perspektywy możemy dostrzec, że struktura AngularJS jest ładowana szybciej niż framework Meteor. Wniosek jest taki, że przy dużych aplikacjach zaoszczędzimy czas ładowania stron korzystając z Angulara, a to wpłynie na ogólną wydajność.

W tabeli 5 pokazane są wyniki z dodaniem 5000 statycznych zadań a w tabeli 5 pokazane są wyniki z dodaniem 15000 statycznych zadań.

Tabela 5 Szybkość ładowania projektu z 5000 statycznych zadań

Framework/Przeglądarka	Chrome	FireFox	Safari
AngularJS	1.6s	1.89s	1.55s
Meteor	1.9s	2.03s	1.82

Tabela 6 Szybkość ładowania projektu z 15000 statycznych zadań

Framework/Przeglądarka	Chrome	FireFox	Safari
AngularJS	3.11s	3.30s	3.15s
Meteor	4.02s	4.10s	3.97s

Tabela 7 Szybkość ładowania projektu - średnia

Framework/Przeglądarka	Chrome	FireFox	Safari
AngularJS	3.11s	3.30s	3.15s
Meteor	4.02s	4.10s	3.97s

W tabeli 7 pokazane są średnie czasy ładowania stron biorąc pod uwagę wszystkie eksperymenty.

Dzięki temu eksperymentowi widać, że liczba linii napisanych w aplikacjach nie różni się w ogólnym zakresie, ale w przyszłości różnica ta będzie rosnąć i stawać się znacząca. Angular wymaga więcej linii kodu, aby stworzyć aplikację, ale wygrywa prędkością ładowania stron. Ponieważ głównym kryterium była szybkość pobierania aplikacji, Angular jest wyraźnym liderem, aczkolwiek wymaga więcej pracy w postaci pisania większej ilości kodu. AngularJS ładuje strony szybciej i w perspektywie większych projektów będzie pracował efektywniej.

5. Porównanie efektywności AngularJS i MeteorJS

Porównanie Meteor i Angular jest trochę skomplikowane, ponieważ w rzeczywistości są to bardzo różne frameworki. Cechą wspólną jest to, że są napisane w JavaScript. Meteor jest pełną strukturą, która pracuje na serwerze Node.js. Angular działa tylko po stronie klienta w przeglądarce. Są one odrębnymi szkieletami, dlatego opisano osobisty pogląd na każdy z ich i to, co u każdego z nich jest "rodzynkiem".

AngularJS i Meteor nie należą do tej samej kategorii. Dlatego też nie można uczyć się jednego, polegając na podobieństwie drugiego.

Obie struktury mają koncepcję tego, co Meteor nazywa "reaktywnymi źródłami danych". W Angularze mamy obiekt obwodu. Stwarza on hierarchię, w której możemy przypisać wartości i funkcje. Później mamy możliwość otrzymania dostępu do tych wartości ze swoich szablonów. Zmiany zostaną automatycznie zaktualizowane po zmianie wartości w obszarze, dając aplikacji pojedynczy punkt w swoim modelu. Angular umożliwia także monitorowanie tych źródeł danych i wykonywanie niestandardowych działań podczas zmiany wartości. Meteor zawiera kilka źródeł reaktywnych danych. Najprostszym przykładem jest sesja. Obiekt sesji umożliwia nam przechowywanie dowolnych par klucz-wartość w czasie rzeczywistym w pamięci przeglądarki. Kursory są kolejnym ważnym źródłem danych reaktywnych w Meteor.

Kursor jest zwracany przy zapytaniu bazy danych. Połączenie reaktywnych źródeł danych i podkategorii - to właśnie sprawia, że Meteor jest tak niezwykły. Gdy zmiany dotyczą danych na jednym kliencie, zmiany te są natychmiast odzwierciedlane w widokach i wysyłane do serwera, a następnie strumienie zmian są zwracane do wszystkich innych połączonych klientów. Kursory umożliwiają również śledzenie i obserwowanie zmian w danych.

6. Dyskusja

Angular2 dostarczany jest z ogromnym spisem funkcji, które pozwolą opracować wszystko, zaczynając od web-dodatków do desktop-owych i mobilnych zastosowań. Framework jest zbudowany na TypeScript od Microsoft którego celem jest stworzenie JavaScript bardziej wyrafinowanym i atrakcyjnym dla dużych projektów. Funkcje Angular2 posiadają architekturę na podstawie komponentów, udoskonalony DI (dependency injection - wprowadzenie zależności), związek między komponentami i tym podobne.

Porównanie AngularJS z Meteorem nie jest, być może, wyczerpujące z tej racji, że Meteor jest full-stuck frameworkiem, a AngularJS jest frameworkiem MVC. Badania nad frameworkami pokazują, że ze względu na prędkość wykonania kodu wygrywa AngularJS, a biorąc pod uwagę konieczną ilość tworzonych linii kodu zwycięzcą jest Meteor.

Na dziszejszy dzień główne kryterium oceny jest czas wykonania aplikacji i z tego wynika że AngularJS jest bardzej przyszłościowy.

Literatura

[1] AngularJS Web Application Development Blueprints. Vinci Rufus, 2014.

[2] Mastering MeteorJS Application Development. Jebin B V, 2015

[3] Mastering Web Application Development with AngularJS. Pawel Kozlowki, 2013.

[4] Node.js, MongoDB, and AngularJS Web Development. Brad Dayley, 2014.

[5] Meteor Design Patterns. Marcelo Reyna,.

[6] Learning AngularJS for .NET Developers. Alex Pop, 2015.

[7] Instant Meteor JavaScript Framework Starter. Gabriel Manricks, 2013.

[8] 2015Pro Angular – Edition 2. Adam Freeman, 2017.

[9] Getting Started with Gulp. Travis Maynard, 2016.

[10] Meteor: Full-Stack Web Application Development. Fabian Vogelsteller Isaac Strack Marcelo Reyna, 2016.

[11] https://hh.ru/ [18.03.2017]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 162-167
Wysłane: 2017-12-02
Przyjęte: 2017-12-09

Analiza wydajnościowa platformy Ionic 2

Robert Pyć*, Małgorzata Plechawska-Wójcik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono analizę wydajnościową platformy Ionic 2. Poddano analizie dwie aplikacje jedna napisaną w języku natywnym Android, czyli w Javie, druga napisano z pomocą frameworka Ionic 2. Do analizy frameworka wybrano następujące kryteria: czas renderowania różnych rodzajów multimediów. płynność działania, zużycie zasobów oraz działanie aplikacji przy obciążonym systemie. Przeprowadzone badania dowiodły iż framework działa znacznie wolniej od aplikacji napisanej w natywnym framework'u Androida.

Słowa kluczowe: aplikacja hybrydowa, Ionic, Ionic 2, Android, Wydajność frameworka

*Autor do korespondencji.

Adres e-mail: robert.pyc@pollub.edu.pl

Efficiency analysis of the Ionic 2 platform

Robert Pyć*, Małgorzata Plechawska-Wójcik
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The paper presents efficiency analysis of the Ionic 2 platform. Two test applications were analysed, one written in the native Android language, Java, and the second written using the Ionic 2 framework. The following criteria were selected for the framework analysis: rendering time of various media types, fluidity, resource consumption and application working while system load. The study has shown that the framework works much slower than an application written in native Android language.

Keywords: hybrid application; Ionic; Ionic 2 Android; Efficiency of framework

*Corresponding author.

E-mail address: robert.pyc@pollub.edu.pl

1. Wstęp

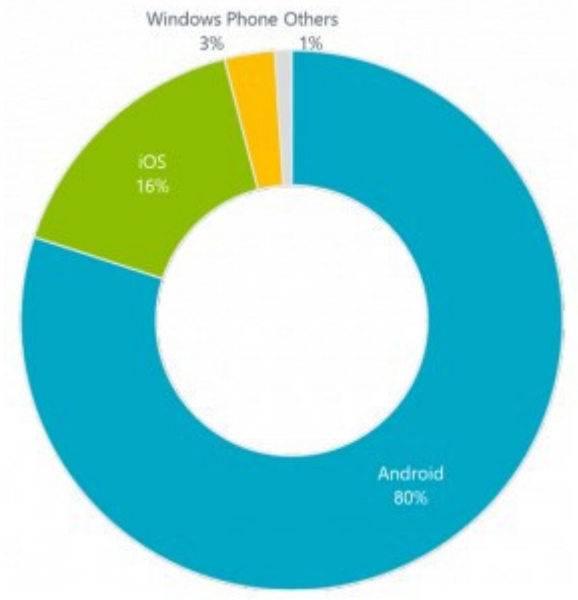
W ostatnich latach liczba użytkowników urządzeń mobilnych znacząco wzrosła w porównaniu do wcześniejszych. lat. Według raportu THE RADICATI GROUP, INC. w 2018 roku liczba użytkowników urządzeń mobilnych sięgnie 6.2 mld, co stanowi około 84% ludzkości. Przewiduje się także, że liczba urządzeń mobilnych przekroczy 12 mld[2][3].

	2014	2015	2016	2017	2018
Worldwide Mobile Users (M)	5,674	5,808	5,945	6,085	6,228
Total Mobile Devices* (M)	7,733	8,627	9,628	10,825	12,165
Mobile Devices Per Business User	1.36	1.49	1.62	1.78	1.95

Rys. 1. Liczba urządzeń mobilnych oraz użytkowników w latach 2014-2018 [2]

Tak dynamicznie wzrastająca liczba użytkowników otworzyła nowe możliwości dla programistów. W 2016 roku liczba aplikacji w Google Play sięgnęła 1,6 mln[4]. Obecnie na rynku urządzeń mobilnych dominują trzy systemy: iOS, Android oraz Windows Phone. Udział w rynku urządzeń mobilnych w roku 2016 prezentuje rysunek 2 [5].

Każdy system cechuje się innym językiem programowania, dla przykładu dla systemu Android jest to język Java, dla Windows Phone C# a dla systemu iOS Objective C/Swift. Istnieją także aplikacje hybrydowe, które dzięki zastosowaniu języka HTML, JavaScript oraz CSS, umożliwiają uruchamianie na dowolnej platformie mobilnej[7].



Rys. 2. Udział w rynku urządzeń mobilnych w roku 2016[6].

Aplikacje hybrydowe stają się obiecującym rozwiązaniem do obsługi wielu platform mobilnych. Dostarczają zarówno specyficzne narzędzia dla platformy oraz interakcje użytkownika za pomocą kodu JavaScript [8]. Aplikacje hybrydowe pomagają programistom tworzyć wiele

programów dla różnych platform bez większego nakładu pracy. Jednym z frameworków, które umożliwiają takie programowanie jest Ionic 2. Zapewnia on użytkownikom wszystkie komponenty, narzędzia i funkcjonalności używane w rodzimym rozwoju aplikacji mobilnych[9][10].

1.1. Cel i obszar badań

Celem badań jest sprawdzenie wydajności framework'a Ionic 2, poprzez stworzenie dwóch aplikacji o tych samych funkcjonalnościach. Pierwsza z nich została napisana w języku natywnym Androida - Java, przy użyciu środowiska Android Studio. Druga napisana została przy użyciu Ionic 2. W ten sposób będzie możliwe zbadanie jak framework radzi sobie z renderowaniem różnych rodzajów multimediów.

W celu dokładnego przeprowadzenia analizy określono następujące kryteria badawcze: zużycie zasobów badanego urządzenia, czas obsługi zlecenia, szybkość działania aplikacji, działanie aplikacji podczas obciążenia systemu przez inne oprogramowania pracujące w tym samym momencie na testowanym urządzeniu.

1.2. Hipotezy badawcze

W niniejszym artykule pod tytułem „Analiza wydajności framework'a Ionic 2" postawiono hipotezę: "Aplikacje stworzone za pomocą framework'a Ionic 2 są wydajniejsze niż aplikacje napisane przy użyciu natywnego framework'a." Zostanie to sprawdzone za pomocą wyżej wymienionych kryteriów.

2. Materiały i metody

Przyjęto, iż do przetestowania framework'a Ionic 2 najlepsza będzie aplikacja menadżera plików multimedialnych. Przyjęto iż będzie on odtwarzał następujące typy plików: obrazy o rozszerzeniach jpg oraz jpeg, animacje typu gif, krótkie filmy o rozszerzeniu mp4 i 3gp, jak również pliki tekstowe txt.

W odtworzenia obrazów dodano dodatkową funkcjonalność polegającą na zmianie jasności oraz kontrastu wybranego przez użytkownika obrazu.

Podobnie jak w przypadku obrazów do odtworzenia plików tekstowych dodano dodatkową funkcjonalność, wyszukiwanie wzorca w tekście.

3. Kryteria analizy

Poniżej skupiono się na dokładniejszym opisie wszystkich określonych wcześniej kryteriów służących sprawdzeniu wydajności framework'a Ionic 2.

Pierwszym badanym kryterium było zużycie zasobów. Zbadane zostało przy użyciu zewnętrznego oprogramowania. Pomogło ono zdobyć informacje o zużyciu procesora oraz pamięci RAM podczas działania aplikacji natywnego framework'a, jak i tej napisanej przy pomocy Ionic 2. Zużycie sprawdzano na czterech rodzajach multimediów: krótkich filmach, animacjach, dokumentach tekstowych w formacie txt, obrazach o różnych rozdzielczościach

Jako drugie kryterium przyjęto czas obsługi zlecenia. Kryterium to zostanie zbadane dzięki zaprogramowaniu w aplikacji funkcjonalności, która po wykonaniu zleconego zadania pokaże informacje z czasem realizacji w milisekundach. Zostanie to zbadane zgodnie z wcześniej przedstawionymi kryteriami, badając różne pliki o rożnych wielkościach oraz rozdzielczościach.

Kolejnym kryterium poddanym analizie została szybkość działania. W tym kryterium brano pod uwagę czas jaki jest potrzebny do uruchomienia aplikacji oraz czy poszczególne ekrany aplikacji wczytują się płynnie.

Jako ostatnie postanowiono zbadać działanie aplikacji przy równoczesnym obciążeniu systemu innymi procesami, oraz późniejszym ponownym przetestowaniu aplikacji pod kątem wszystkich powyższych kryteriów.

3.1. Wyszukiwanie wzorca w tekście

Dany test polega na wyszukaniu w dużej ilości tekstu konkretnego fragmentu tekstu, czyli wzorca. W folderze znajdują się pliki z tekstem o rozszerzeniu txt. Są one różnego rozmiaru od 200kB do 2 MB.

Fragment kodu przedstawiony poniżej (Rys. 3) przedstawia szukanie w tekście wzorca. Jeżeli zostanie on znaleziony, szukany fragment podświetla się na czerwono.

Przykład. 1. Fragment aktywności TextActivity odpowiedzialny za wyszukanie wzorca w aplikacji Androida

```
 if (fullText.contains(criteria)) {
                int indexOfCriteria = fullText.indexOf(criteria);
                int lineNumber =
tv.getLayout().getLineForOffset(indexOfCriteria);
                String highlighted = "<font
color='red'>"+criteria+"</font>";
                fullText = fullText.replace(criteria, highlighted);
                tv.setText(Html.fromHtml(fullText));
                textWrapper.scrollTo(0,
tv.getLayout().getLineTop(lineNumber));
            }else {
                fullText = fullText.replace("<font
color='red'>", "");
                fullText = fullText.replace("</font>", "");
               tv.setText(fullText);
            }
```

3.2. Liczenie czasu wykonywania zadania

Została zaprogramowana funkcjonalność, dzięki której po zakończeniu wykonywania każdej czynności aplikacja wyświetla czas jaki został poświęcony na realizację. Poniższy fragment kodu (Rys. 4) przedstawia przykładową metodę odpowiedzialną za wyświetlenie czasu.

Przykład. 2: Przykładowy kod pokazujący czas realizacji zadania w aplikacji Androida

```
protected void onPostExecute(Bitmap bitmap) {
            super.onPostExecute(bitmap);
            if (bitmap != null) {
               ImageView myImage = (ImageView)
findViewById(R.id.imageView);
               myImage.setImageBitmap(bitmap);
            }
            long time = System.currentTimeMillis() -
startTime;
            Toast.makeText(PhotoActivity.this,"Czas ładowania
zdjęcia: "+time + "ms",Toast.LENGTH_LONG).show();
            progressbar.dismiss();
         }
```

3.3. Realizacja operacji na plikach graficznych

Analizę operacji na plikach graficznych postanowiono przeprowadzić poprzez zastosowanie obrazów o różnej rozdzielczości oraz użycie filtrów w postaci zmiany kontrastu oraz jasności wyświetlonego obrazu. Po każdej zmianie filtrów wyświetlany jest czas realizacji zdania.

W przypadku aplikacji androida funkcja ta została zrealizowana poprze metodę którą reprezentuje rysunek 5. Przyjmuje ona wartości jasności oraz kontrastu jako parametry a następnie poprzez wykorzystanie metody colormatrix w obiekcie Paint rysuje nowa bitmapę.

Przykład 3. Metoda zmiany jasności oraz kontrastu w aplikacji androida

```
public static Bitmap changeBitmapContrastBrightness(Bitmap
bmp, float contrast, float brightness,float saturation)
  {
     ColorMatrix cm = new ColorMatrix(new float[]
          {
               contrast, 0, 0, 0, brightness,
               0, contrast, 0, 0, brightness,
               0, 0, contrast, 0, brightness,
               0, 0, 0, 1, 0
          });
     Bitmap ret = Bitmap.createBitmap(bmp.getWidth(),
bmp.getHeight(), bmp.getConfig());

     Canvas canvas = new Canvas(ret);
     Paint paint = new Paint();
     paint.setColorFilter(new ColorMatrixColorFilter(cm));
     canvas.drawBitmap(bmp, 0, 0, paint);
     return ret;
  }
}
```

W przypadku aplikacji Ionic 2 zastosowano style CSS. Poprzez fragment kodu przedstawiony na rysunku 6 można w czasie rzeczywistym zmieniać jasność oraz kontrast wybranego obrazu.

Przykład 4. Funkcja w aplikacji Ionic 2 do zmiany jasności oraz kontrastu

```
<img id="myImage" src="{{ data }}" [ngStyle]="{'filter':
'brightness(' + brightness/100 + ') '+'contrast(' +
contrast/100 + ')'}"/>
```

3.4. Odtwarzanie animacji gif

Do prawidłowego odtworzenia animacji gif zainstalowano dodatek Glide, który wspomaga ich wyświetlanie oraz umożliwia ich poruszanie.

4. Badania

Badania zostały przeprowadzone na trzech urządzeniach mobilnych o różnej charakterystyce sprzętowej. Były to:

1) Sony Xperia M2
2) Xiaomi Redmi 4x
3) Xiaomi Mi 6

Na samym początku zbadano czas renderowania poszczególnych elementów multimedialnych, zwracając uwagę na zużycie procesora oraz pamięci RAM. Użycie zasobów urządzenia mobilnego zbadano z pomocą narzędzia "Tinycore". Oprogramowanie to działając w tle zbiera informacje o aktualnym stanie użytkowania procesora oraz RAM-u a następnie wyświetla je w postaci paska

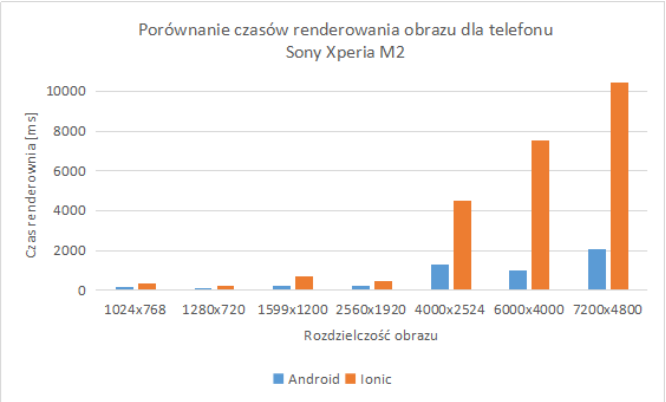
4.1. Galeria

Jako pierwszą postanowiono podda badaniu galeria obrazów. Poniższa tabela (Tabela 1) reprezentuje czas renderowania obrazów z galerii podany w milisekundach.

Tabela 1. Czas renderowania obrazów z galerii. Dane w ms

Rozdzielczość grafiki	Sony Xperia M2		Xiaomi Mi 6		Xiaomi Redmi 4x	
	Android	Ionic	Android	Ionic	Android	Ionic
1024x768	150	367	47	121	76	266
1280x720	114	193	45	69	84	154
1599x1200	235	672	68	141	131	389
2560x1920	210	485	87	147	137	427
4000x2524	1289	4517	250	1246	707	3683
6000x4000	1014	7541	220	1357	560	4861
7200x4800	2081	10456	455	1548	1125	5587

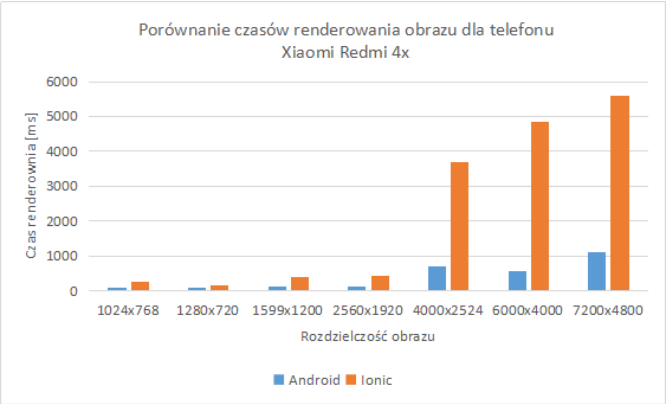
Poniższe wykresy prezentują wykresy słupkowe czasu renderowania obrazów z galerii dla telefonów Sony Xperia M2 (rysunek 3), Xiaomi Redmi 4x (rysunek 4), Xiaomi Mi 6 (rysunek 5). Oś X oznacza rozdzielczość prezentowaną w pikselach natomiast oś X czas renderowania grafiki wyrażony w milisekundach.



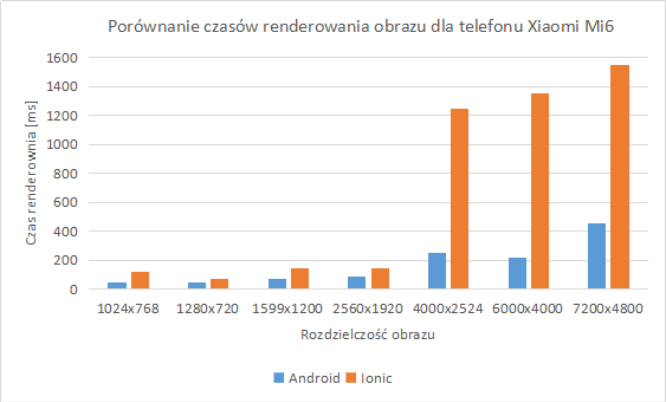
Rys 3. Porównanie czasów renderowania obrazu dla telefonu Sony Xpieria M2.

W przypadku telefonu marki Sony można było zauważyć znaczne zużycie procesora, zwiększało się o ok 30% przy uruchamianiu obrazów szczególnie tych o większej rozdzielczości. Przy badaniu zużycia pamięci Ram nie pojawiły się znaczące zmiany.

Przy badaniu wydajności aplikacji na telefonach marki Xiaomi nie zauważano wzrost zużycia procesora w przypadku otwierania obrazów o ok 10-15%, natomiast nie odnotowano zmiany w użyciu pamięci RAM.



Rys 4. Porównanie czasów renderowania obrazu dla telefonu Xiaomi Redmi 4x



Rys 5. Porównanie czasów renderowania obrazu dla telefonu Xiaomi Mi6

Następnie skupiono się na wykorzystaniu funkcjonalności zmiany jasności oraz kontrastu. W przypadku aplikacji napisanej za pomocą frameworka Ionic 2, można zaobserwować zmiany wprowadzone przez filtry w czasie rzeczywistym. Zauważono także, iż użycie procesora w przypadku dynamicznej zmiany filtrów rośnie na wszystkich badanych urządzeniach, Xperia około 20%, Redmi 4x w przybliżeniu 50%, natomiast na Mi6 około 10%.

W oprogramowaniu napisanym w natywnym framework'u Androida, czas potrzebny na wczytanie pliku graficznego z nałożonym filtrem nie różni się znacznie od czasu potrzebnego do wczytania oryginalnego obrazu. Zostało to potwierdzone na wszystkich urządzeniach mobilnych, co przedstawia tabela 2.

Tabela 2. Czas renderowania obrazów z galerii w ms wraz z czasem renderowania filtrów

Rozdzielczość obrazu	**Sony Xperia M2**		**Xiaomi Mi 6**		**Xiaomi Redmi 4x**	
	Android	**Filtr**	**Android**	**Filtr**	**Android**	**Filtr**
1024x768	150	153	47	45	76	75
1280x720	114	107	45	50	84	90
1599x1200	235	267	68	74	131	145
2560x1920	210	245	87	91	137	140
4000x2524	1289	1157	250	264	707	763
6000x4000	1014	1057	220	206	560	497
7200x4800	2081	2154	455	502	1125	1245

Następnie powtórzono powyższe badania dla systemu obciążonego innymi aplikacjami działającymi w tle. W przypadku urządzenia Xiaomi Mi 6, nie udało się obciążyć pamięci RAM oraz procesora w stopniu który powodowałby zmiany w uzyskanych czasach renderowania plików graficznych. Przy smartfonie Xiaomi Redmi 4x zwiększono zużycie procesora o 50% oraz zużycie RAM-u do około 80%. Jednak, co zaskakujące, nie udało się zaobserwować wydłużenia czasu renderowania obrazów w żadnym z przypadków.

4.2. Animacje gif

Następnie skupiono się na przeanalizowaniu animacji gif. Testy pomogły uzyskać informacja na temat czasu jaki aplikacja potrzebowała na uruchomienie danego pliku.

Jako pierwszą sprawdzono aplikacje Ionic 2. Stwierdzono iż w przypadku urządzenia Sony Xperia M2 oraz animacji o wymiarach 240x320 framework potrzebuje średnio 189 ms natomiast animacje o mniejszych wymiarach około 97 ms. Następnie proces powtórzono na urządzeniu Xiaomi Mi6,w tym przypadku większe animacje wczytywać się ok 98 ms natomiast mniejsze potrzebowały 53 ms. Jako ostatnie urządzenie sprawdzono Xiaomi Redmi 4x. W tym przypadku czasy ładowania wynosiły niemalże tyle samo co w przypadku urządzenia marki Sony.

Aplikacja napisana w natywnym framework'u Androida potrzebowała jedynie około 2 ms na otworzenie dowolnej animacji w przypadku wszystkich urządzeń

Nie zauważono stałych wzrostów zużycia procesora oraz pamięci RAM. Wystąpiło jedynie tymczasowe podwyższenie zużycia procesora przy otwieraniu pliku.

Następnie powtórzono powyższe badania, ja w przypadku galerii, na obciążonych systemach. Jednakże nie wpłynęło to na wydłużenie czasu ładowania plików, czy też zwiększenie zużycia zasobów.

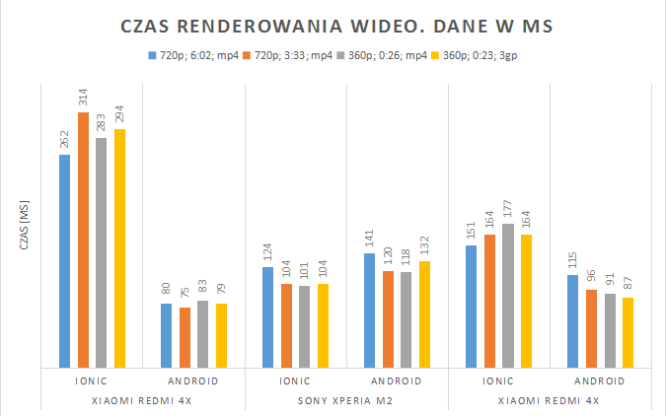
4.3. Wideo

Podobnie jak w przypadku galerii oraz animacji badanie plików wideo polegało na zmierzeniu czasu wczytywania poszczególnych filmów. Do badanie wybrano pliki z rozszerzeniem .mp4 oraz .3gp o jakościach 720p oraz 360p. Wyniki badania prezentuje poniższa tabela 3.

Tabela 3. Czas wczytywania wideo. Dane w ms oraz w minutach i sekundach

Jakość filmu	**Długość wideo [mm:ss]**	**Xiaomi Mi6**		**Sony Xperia M2**		**Xiaomi Redmi 4x**	
		Ionic	**Android**	**Ionic**	**Android**	**Ionic**	**Android**
720 p	6:02	262	80	124	141	151	115
720 p	3:33	314	75	104	120	164	96
360 p	0:26	283	83	101	118	177	91
360 p	0:23	294	79	104	132	164	87

Poniższy wykres (rysunek 6) prezentuje czas jaki jest potrzebny do wczytania filmu uwzględniając długość filmu jego rozszerzenie jak i język aplikacji na której jest on uruchamiany.



Rys 6. Czas renderowania wideo. Dane w ms, długość filmu podana w formacie minuty:sekundy.

Jak w przypadku poprzednich kryteriów tak i w tym teście badano wpływ obciążenia systemu na płynność wczytywania wideo. Różnice były minimalne, mieściły się w granicach 20 ms.

Analiza zużycia zasobów w trakcie uruchamiania oraz trwania wideo wykazała nieznaczny stały wzrost zużycia procesora, wynosił on mniej niż 5% w przypadku badanych urządzeń. Nie nie zaobserwowano zmian w zużyciu pamięci RAM.

4.4. Tekst

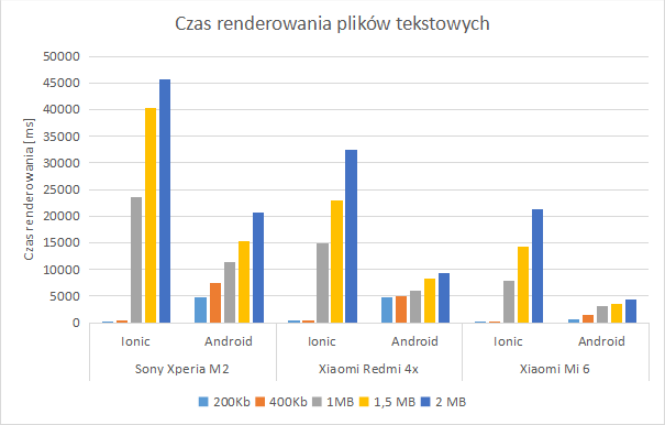
W badaniach na plikach tekstowych brano pod uwagę czas wczytania poszczególnych plików oraz czas potrzebny do wyszukania podanego wzorca w wybranym tekście Poniższa tabela 4 przedstawia długość oczekiwania na wykonanie zadania.

Tabela 4. Czasy wczytywania plików tekstowych. Dane w ms

Rozmiar pliku	Sony Xperia M2		Xiaomi Redmi 4x		Xiaomi Mi 6	
	Ionic	Android	Ionic	Android	Ionic	Android
200Kb	284	4862	192	4812	74	732
400Kb	390	7462	362	4951	100	1495
1MB	23591	11468	14897	6045	7941	3154
1,5 MB	40245	15286	23007	8253	14286	3549
2 MB	45721	20648	32468	9365	21279	4281

Rysunek 7 ilustruje porównanie czasów wczytywania plików tekstowych, uwzględniając rozmiar, dla poszczególnych urządzeń mobilnych.

Następnie poddano badaniu opcje wyszukiwania wzorca w tekście. W przypadku aplikacji Ionic 2 jedynie na urządzeniu Xiaomi Mi6 było możliwe płynne wyszukanie. Na pozostałych urządzeniach, przy próbie wpisania szukanego wyrazu, oprogramowanie nie reagowało. Czasy realizacji zadania przedstawia tabela 5



Rys 7. Porównanie czasów renderowania plików tekstowych

Tabela 5. Czas wyszukania wzorca w tekście. Dane w ms

Rozmiar pliku	Xiaomi Mi 6		Sony Xperia M2		Xiaomi Redmi 4x	
	Ionic	Android	Ionic	Android	Ionic	Android
200Kb	5412	425		3052		1608
400Kb	5971	970		6815		4703
1MB	6874	1246		8561		7539
1,5 MB	7932	1701		10241		8824
2 MB	10548	2111		15262		11093

Przy odczycie plików tekstowych zauważono znaczne zwiększenie zużycia procesora, trwające ok 5-10 sekund po zakończeniu realizacji zadania. W przypadku urządzenia Sony zaobserwowano zużycie procesora wynoszące 100%. Jeśli chodzi o pozostałe telefony wzrost był zauważalny, lecz znacznie mniejszy niż przy smartfonie marki Sony, gdyż wynosił on 25% dla urządzenia Xiaomi Redmi 4x oraz około 10% dla urządzenia Xiaomi Mi6.

4.5. Płynność działania

Jako ostatnie zbadano płynność działania. Już przy uruchamianiu obu aplikacji zauważono iż napisana w języku Androida uruchamia się znacznie szybciej niż Ionic 2.

W przypadku galerii, wczytywanie obrazów o mniejszej rozdzielczości odbywa się płynnie natomiast przy większych rozdzielczościach trzeba poczekać na wczytanie.

Pliki wideo oraz animacje gif w obu aplikacjach działają płynnie oraz nie można dostrzec żadnych opóźnień.

Zarówno aplikacja Androida jak i Ionic'a miała pewnie problemy z plikami tekstowymi. Czas oczekiwania na wczytanie całego pliku jest dłuższy przy większych plikach sięga nawet do 45 sekund w przypadku urządzenia marki Sony.

Wyszukiwanie w aplikacji Ionic 2, jak już było wspomniane wcześniej, udało się zrealizować jednie na urządzeniu o najlepszych parametrach sprzętowych, czyli Xiaomi Mi6. W przypadku reszty urządzeń czynność ta była niemożliwa ze względu na zawieszanie się aplikacji. W przypadku aplikacji napisanej w języku Androida, wyszukanie było możliwe, jednak wymagało ono chwilowego oczekiwania.

5. Wnioski

W niniejszym artykule przyjęto hipotezę iż aplikacje napisane z użyciem framework'a Ionic 2 są wydajniejsze od aplikacji napisanych w natywnym framework'u Androida. Dużym zaskoczeniem były wyniki badań aplikacji napisanej za pomocą framework'a Ionic 2. Spodziewano się iż okaże się ona lepszym rozwiązaniem pod względem wydajnościowym. Jednak testy jednoznacznie obalają postawioną hipotezę.

Ionic 2 może okazać się dobrym rozwiązaniem do szybkiego tworzenia aplikacji typowo biznesowych dzięki temu, że korzysta z narzędzia Angular 2. Największą, a zarazem decydującą dla badań, zaletą pisania programu przy użyciu języków natywnych platform mobilnych, jest homogeniczność rozwiązań. Tworzony projekt jest przeznaczony tylko dla jednej platformy.

W przypadku obciążenia systemu, przez inne aplikacje, spodziewano się odnotować znaczny wzrost czasu otwierania poszczególnych multimediów, jednak wyniki badań wybranych kryteriów nie potwierdziły tego.

Literatura

[1] A. Tonini, L. Fischer, J Carlos Balzano de Mattos, L Brisolara Analysis and Evaluation of the Android Best Practices Impact on the Efficiency of Mobile Applications,

[2] https://www.radicati.com/wp/wpcontent/uploads/2014/01/Mobile-Statistics-Report-2014-2018-Executive-Summary.pdf [22.11.2017]

[3] G. Shrivastava, P. Kumar, Privacy Analysis of Android Applications: State-of-art and Literary Assessment.

[4] https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/[21.11.2017]

[5] https://www.computerworld.pl/news/Najpopularniejsze-platformy-mobilne-wykorzystywane-w-biznesie,393299.html [17.11.2017]

[6] X. Chen; Z. Zong, Android App Energy Efficiency: The Impact of Language, Runtime, Compiler, and Implementation.

[7] S. Helal,R. Bose W. Li, Mobile Platforms and Development Environments Synthesis Lectures on Mobile and Pervasive Computing.

[8] A. Gupta, A. Gaffar H, Hybrid Application Development using Ionic Framework & AngularJS.

[9] S. Lee; J. Dolby; S. Ryu, HybriDroid: Static analysis framework for Android hybrid applications.

[10] C. Griffith, Mobile App Development with Ionic 2: Cross-Platform Apps with Ionic, Angular .& Cordova , O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, 2017-04-07.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 168-171
Wysłane: 2017-12-04
Przyjęte: 2017-12-09

Porównanie wydajności technologii Xamarin i Java przy pracy z bazą danych

Oleh Datsko*, Elżbieta Miłosz
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Praca z bazą danych jest jedną z podstawowych rzeczy przy tworzeniu aplikacji. Każda technologia/język oprogramowania używa własnych sposobów do pracy z bazą. W przypadku Android jest używany system zarządzania bazą danych SQLite. Przedstawiona analiza dotyczy zapisu, odczytu i usuwania danych z tabeli. Ze względu na to, że system został ograniczony w porównaniu do wersji desktopowej, dla każdego eksperymentu jest używana ograniczona liczba danych.

Słowa kluczowe: analiza; Xamarin; Java; baza danych

*Autor do korespondencji.

Adres e-mail: oleh.datsko@pollub.edu.pl

Performance comparison between Xamarin and Java database operations

Oleh Datsko*, Elżbieta Miłosz
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. Work with database is one of base things in developing an application. Every technology/language uses own ways to work with database. In the case of Android the SQLite relational database management system is used. Analysis applies the read, write and delete of items from the table. SQLite is like a minimized version of desktop database management system so for every experiment very limited elements count is used.

Keywords: analysis; Xamarin; Java; database

*Corresponding author.

E-mail address: oleh.datsko@pollub.edu.pl

1. Wstęp

Głównym problemem współczesnych urządzeń mobilnych jest czas ich pracy na jednym naładowaniu baterii przy dużych obciążeniach często spotykanych w aplikacjach mobilnych wykonujących różne operacje, na przykład praca z bazą danych.

Bazą danych SQLite jest plik o formacie dowolnym wybranym przez użytkownika, który zapisuje się jako zwykły plik w systemie z tym tylko, że jest w formacie binarnym, więc jest dostępny w wielu systemach operacyjnych bez dodatkowych przetwarzań [1]. Użycie SQLite w Xamarin oraz Java prawie się nie różni, dlatego implementacja jest prawie taką samą, a składnia SQLite jest niezmienną na wszystkich platformach bądź to Android czy Windows). Obie technologie są bardzo podobne nawet w tym, że umieszczają klasy SQLite w tym samym miejscu – w pakiecie android.dabatase.sqlite w Javie [2] oraz w przestrzeni nazw Android.Database.Sqlite w Xamarin [3].

Wszystkie dane można przechowywać na dysku w zwykłym pliku, odczytując i zapisując go za każdym razem przy zmianie danych, ale są przypadki, kiedy system zarządzania bazą danych ma większą kontrolę nad danymi i traci mniej czasu na akcje zapisu/odczytu/usunięcia. Zgodnie z oficjalnym testowaniem, odczytywanie w SQLite jest w 2 razy szybsze, a zapisywanie jest wolniejsze niż gdyby użyto zwykłego pliku [4].

Celem artykułu jest porównanie wydajności przy pracy z systemem zarządzania bazą danych SQLite przez technologię Xamarin i Java. W zakres badania wchodzi porównanie szybkości odczytywania, zapisywania oraz usunięcia danych. Przy tym, usunięcie i odczytywanie sprawdzane jest na 2 sposoby – po kolei, oraz wszystkie za jednem razem. Różnicą tych dwóch metod jest to, że przy pierwszym nie jest używany SQLite na wszystkie 100% jak w drugim przypadku, zarządzanie wykonywaniem wszystkich operacji poszukiwania i usunięcia elementu leży całkowicie na odpowiedzialności samego systemu zarządzania bazą.

2. Plan badań

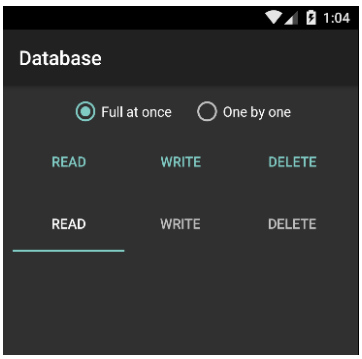
Celem badań jest sprawdzenie czasu pracy SQLite przy użyciu dwóch technologii: Xamarin.Android i Java. Dane technologie są zrealizowane w bardzo podobny sposób z tą różnicą, że są napisane używając, gdzie była taka potrzeba, wszystkich dostępnych podejść z języków programowania (C# i Java).

Hipotezy badawcze zdefiniowane do weryfikacji są następujące:

- Odczytywanie z bazy odbywa się szybciej w Java.
- Zapisywanie do bazy jest szybsze w Java.
- Usunięcie z bazy jest szybsze w Java.

Podstawą do przypuszczenia, że wyniki Javy we wszystkich trzech przypadkach będą lepsze jest to, że Android został napisany z myślą o użyciu Java (jej składni) jako technologii do tworzenia aplikacji mobilnych, a Xamarin powstał znacznie później i używa JNI (ang. Java Native Interface) i wiązanie z Java z obiektami .NET, jako pośrednik między Androidem a .NET (bezpośredni dostęp do API Androida jest niemożliwy, bo system na to nie pozwala) [5].

Dla przeprowadzenia badania zostały napisane identyczne aplikacje (Rys.1) oddzielnie w Xamarin.Android i Java, w których interfejs jest taki sam (cecha Xamarin.Android, że interfejs nie różni się od interfejsu napisanego w Java).



Rys. 1. Widok interfejsu do testowania bazy danych.

Dla odczytywania i usunięcia danych zostały wykorzystano dwie metody: pojedynczo czy wszystko naraz. Działa to w taki sposób, że wybierając wszystko naraz, baza danych zwróci lub usunie wszystkie elementy wykorzystując tylko jedną iterację w metodzie głównej, która wywołuje metodę z dostępem do bazy, a w tej już zwraca wszystkie elementy z bazy iterując używając kursora. Drugi sposób działa w taki sposób, że iteracja odbywa się w metodzie głównej, gdzie z metody dostępu do bazy zwraca się obiekt zawierający id. Krok iteracji kończy się i zaczyna się kolejny, w którym są wyszukiwane elementy z identyfikatorem id, który jest większy od przekazanego w parametrze.

Inną jest metoda zapisywania danych, ponieważ dostęp do bazy możliwy jest tylko w sposób pojedynczy, ponieważ jest konieczne przetwarzanie obiektu w inny typ, a w danym przypadku to jest ContentValues, który później przekazuje się do metody wstawienia wpisu do bazy.

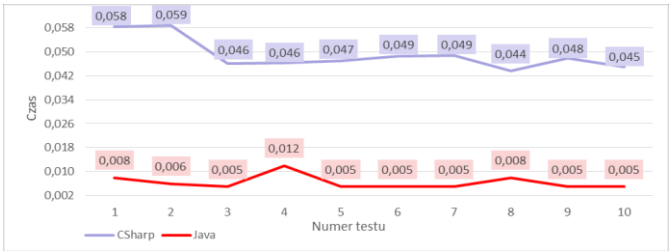
Opisane powyżej metody badania są aktualne dla Xamarin.Android i Java z tym, że w .NET używa się klasy stopwotcher dla uzyskania czasu pracy, który jest polecany zamiast DateTime, ponieważ ma większą dokładność czasu. W aplikacji Java czas pracy jest pobierany przez pakiet System, który zawiera metodę nanoTime. Nie patrząc na to, że dokładność jest bardzo duża, ale jednostką miary czasu jest milisekunda (dokładność jest do jednej milisekundy), przez co została napisana metoda przetwarzająca (metoda dodana do klasy bazowej, która dziedziczy po klasie Activity, od której dziedziczy aktywność testu).

Oprócz zapisywania czasu w samej aplikacji, czas wykonywania metody tak samo jest pobierany z Android Monitor – potężnego narzędzia dla debugowania Android aplikacji, które zawiera wykresy używania CPU, pamięci operacyjnej, sieci oraz GPU aplikacji [6]. Także narzędzie pozwala na logowanie akcji z kodu za pośrednictwem narzędzia z nazwą logcat.

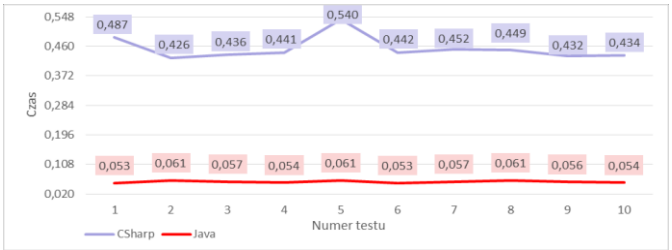
Przeprowadzone badanie zawiera wyniki zapisywania 10 testów (na każdą z dwóch metod). Przed każdym eksperymentem aplikacja uruchamia się ponownie, żeby oczyścić śmieci, które pozostały od poprzedniego eksperymentu (teoretycznie nie jest to potrzebne, ale dane mogą mieć niegatywny wpływ na kolejny test). Eksperyment z odczytywaniem bazy danych (za jednym razem) przeprowadzono dla 100, 1000, 10000 i 100000 elementów, dla zapisywania i usunięcia za jednym razem eksperyment przeprowadzono dla 100, 1000 oraz 10000 rekordów. Dla usunięcia przeprowadzono jeszcze kolejny test z usunięciem pojedynczym, który zawiera 100 elementów.

3. Wyniki badań

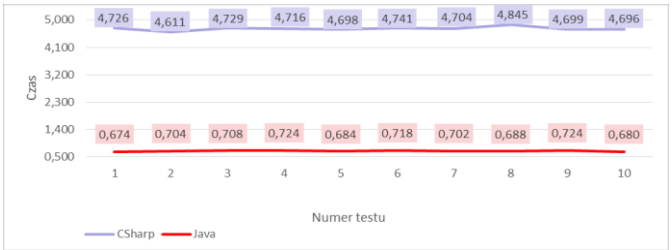
Wyniki badań są przedstawione w postaci wykresów (Rys.2 - Rys.12) oraz w tabeli porównawczej (Tabela 1), gdzie znajdują się procentowe wartości czasu wykonywania i obciążenia CPU. Czas podany na wykresach jest w sekundach.



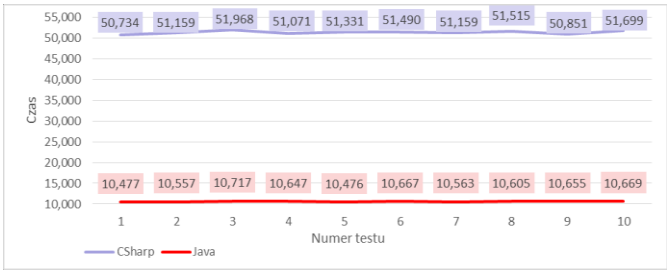
Rys. 2. Test 100 elementów przy odczytywaniu bazy danych.



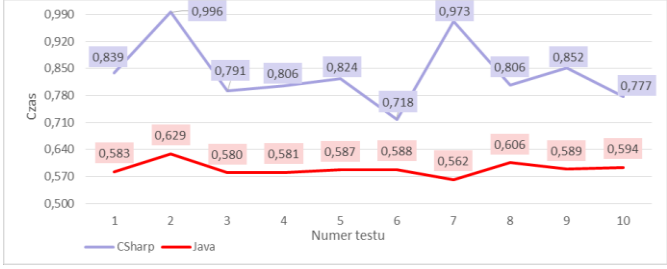
Rys. 3. Test 1000 elementów przy odczytywaniu bazy danych.



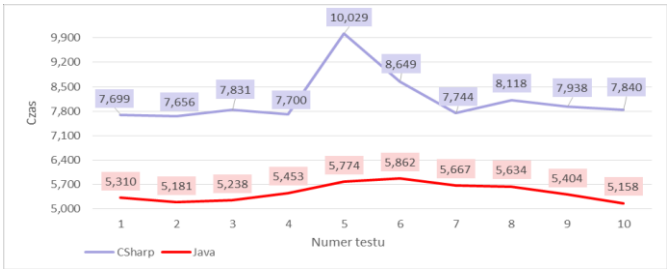
Rys. 4. Test 10000 elementów przy odczytywaniu bazy danych.



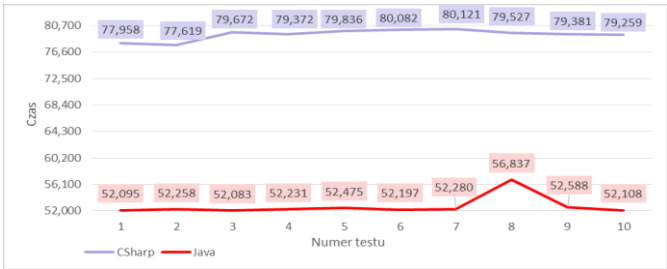
Rys. 5. Test 100000 elementów przy odczytywaniu bazy danych.



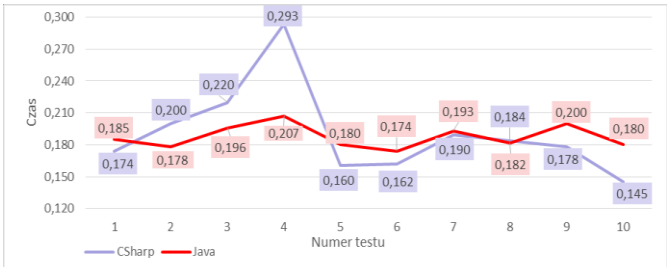
Rys. 6. Test 100 elementów przy zapisie do bazy danych.



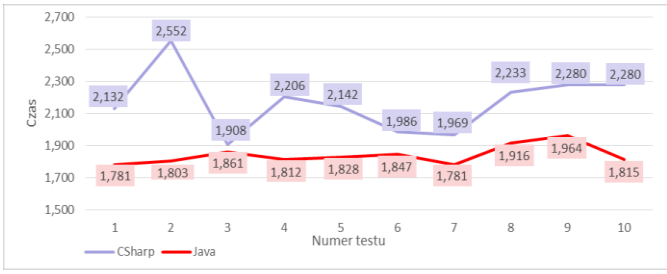
Rys. 7. Test 1000 elementów przy zapisie do bazy danych.



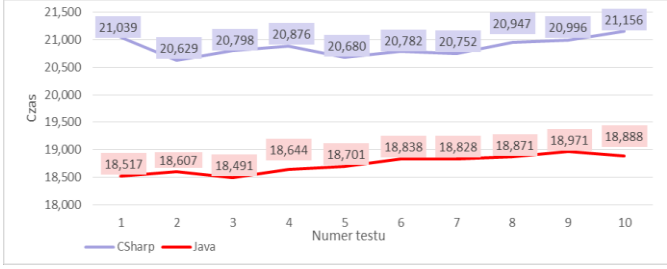
Rys. 8. Test 10000 elementów przy zapisie do bazy danych.



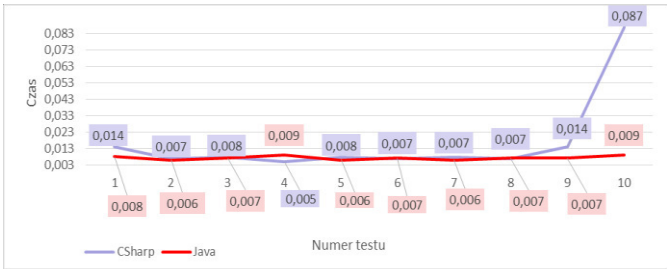
Rys. 9. Test 100 elementów przy usunięciu z bazy danych.



Rys. 10. Test 1000 elementów przy usunięciu z bazy danych.



Rys. 11. Test 10000 elementów przy usunięciu z bazy danych.



Rys. 12. Test 100 elementów przy pojedynczym usunięciu z bazy danych.

4. Wnioski

Na podstawie przeprowadzonych badań otrzymano wyniki szybkości wykonywania operacji z bazą danych używając technologii Xamarin.Android i Java (Tabele 1-3).

Dodatkowy test został przeprowadzony dla pojedynczego usunięcia z tabeli 100 elementów. Wyniki testu są następujące: czas usunięcia wykorzystując Java jest na 125,6% mniejszy niż w Xamarin.Android. Przy tym, obciążenie CPU jest na 37,3% mniejsze na korzyść Java.

Tabela. 1. Dane (w procentach) Java w porównaniu do Xamarin.Android przy odczytaniu danych (minus oznacza przewagę Xamarin.Android)

Liczba elementów	Czas	CPU (aplikacja)	CPU (jądro)
100	665,3	2,3	13,0
1000	700,3	45,9	10,3
10000	573,2	-3,9	-71,4
100000	383,8	-0,9	-28,6

Z Tabeli 1 wynika, że odczyt danych przy użyciu Java jest od 3 do 7 razy szybszy niż analogiczny, obciążenie CPU aplikacją jest mniejsze przy mniejszej ilości danych (od 2,3% do 45,9%), ale przy dużej ilości danych zwycięża Xamarin.Android.

Tabela. 2. Dane (w procentach) Java w porównaniu do Xamarin.Android przy zapisywaniu danych (minus oznacza przewagę Xamarin.Android)

Liczba elementów	Czas	CPU (aplikacja)	CPU (jądro)
100	42,1	32,2	8,7
1000	48,5	7,8	-39,6
10000	50,4	37,0	-27,8

Z Tabeli 2 wynika, że szybkość zapisu do bazy nie jest na takim samym poziomie zwycięstwa jak to było przy odczycie, ale czas wykonywania tej samej operacji jest na 40%-50% mniejszy od analogicznego w Xamarin.Android. Jeżeli chodzi o obciążenie CPU to Java tu tak samo pokazuje absolutne zwycięstwo.

Zgodnie z Tabelą 3, czas wykonywania w Java jest mniejszy na 1%-18% niż analogiczny w Xamarin.Android. Z tym, że w teście z 1000 elementów, obciążenie CPU jest na 23,5% mniejsze w Xamarin.Android.

Tabela. 3. Dane (w procentach) Java w porównaniu do Xamarin.Android przy pojedynczym usunięciu danych (minus oznacza przewagę Xamarin.Android)

Liczba elementów	Czas	CPU (aplikacja)	CPU (jądro)
100	1,6	39,6	31,3
1000	17,8	-23,5	-39,6
10000	11,4	16,7	-46,2

Wyniki przedstawione w tabelach 1-3 potwierdzają założone hipotezy:

1. Odczytywanie z bazy odbywa się szybciej w Java.
2. Zapisywanie do bazy jest szybsze w Java.
3. Usunięcie z bazy jest szybsze w Java.

Literatura

[1] G. Allen, M. Owens, The Definitive Guide to SQLite, Apress, 2010.

[2] android.database.sqlite, https://developer.android.com/reference/android/database/sqlite/package-summary.html [01.11.2017].

[3] Android.Database.Sqlite Namespace, https://developer.xamarin.com/api/namespace/Android.Database.Sqlite/ [01.11.2017]

[4] 35% Faster Than The Filesystem, https://www.sqlite.org/fasterthanfs.html [04.11.2017]

[5] Working With JNI, https://developer.xamarin.com/guides/android/advanced_topics/java_integration_overview/working_with_jni/ [05.11.2017]

[6] Android Monitor Basis https://developer.android.com/studio/profile/am-basics.html#byb [05.11.2017]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 172-177
Wysłane: 2017-12-10
Przyjęte: 2017-12-12

Analiza porównawcza reakcji na bodźce wzrokowe i słuchowe w badaniach potencjałów wywołanych EEG

Łukasz Tyburcy*, Małgorzata Plechawska-Wójcik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł opisuje badania polegające na porównaniu czasów reakcji na bodźce wzrokowe i słuchowe przy pomocy potencjałów wywołanych EEG. Do realizacji badań wykorzystano dwa eksperymenty. Pierwszy badał czasy reakcji na bodźce wzrokowe, drugi badał czasy reakcji na bodźce słuchowe. Po przeprowadzeniu analizy danych uzyskane rezultaty pozwoliły określić, że bodźce wzrokowe wywołują szybszą reakcję niż bodźce słuchowe.

Słowa kluczowe: potencjały wywołane; bodźce; elektroencefalografia

*Autor do korespondencji.

Adres e-mail: lukasz13052@gmail.com

Comparative analysis of reactions to visual and auditory stimuli in research on EEG evoked potentials

Łukasz Tyburcy*, Małgorzata Plechawska-Wójcik
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The paper describes results of comparison of reactions times to visual and auditory stimuli using EEG evoked potentials. Two experiments were used to applied. The first one explored reaction times to visual stimulus and the second one to auditory stimulus. After conducting an analysis of data, received results enable determining that visual stimuli evoke faster reactions than auditory stimuli.

Keywords: evoked potentials; incentives; electroencephalography

*Corresponding author.

E-mail address: lukasz13052@gmail.com

1. Wstęp

Ostatnie lata przyniosły dynamiczny rozwój informatyki. Miało to duży wpływ na rozwój dziedzin nauki, w których wykorzystywane są komputery. Jednym z takich przykładów jest medycyna. Obecnie większość badań przeprowadza się przy pomocy zaawansowanych urządzeń, które swoje obliczenia opierają na rozwiązaniach informatycznych. Poszerza to w znaczący sposób możliwości badań ludzkiego organizmu. Najważniejszym organem ludzkim jest mózg. Steruje wszelkimi procesami zachodzącymi w ciele człowieka oraz pozwala odbierać i przetwarzać bodźce. Dzięki rozwojowi technologii możliwe jest bezinwazyjne badanie tego organu. W artykule przedstawiono badanie mające na celu stwierdzić, które bodźce wywołują szybsze reakcje: wzrokowe czy słuchowe. Do analizy zastosowano metodę elektroencefalograficznych potencjałów wywołanych.

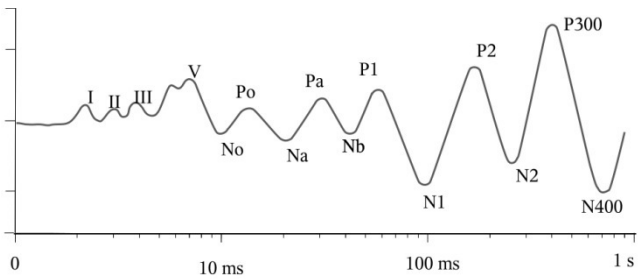
2. Elektroencefalografia

Elektroencefalografia (EEG) jest nieinwazyjną metodą pozwalającą rejestrować czynności bioelektryczne mózgu. W dzisiejszych czasach do tego typu badań wykorzystuje się zaawansowane elektroencefalografy. Aktywność mózgu rejestrowana jest przy pomocy elektrod umieszczonych na głowie pacjenta. Cały proces polega na określeniu zmian potencjałów elektrycznych. Pochodzą one z neuronów, które znajdują się w mózgu[1]. Obecnie elektroencefalografię wykorzystuje się do rozpoznania wielu schorzeń, takich jak padaczka, zaburzenia senne, choroba Parkinsona, choroba Alzheimera oraz wiele innych. Wykorzystywana jest również w innych dziedzinach nauki takich jak: interfejsy mózg-komputer oraz biofeedback. Obecnie wiele artykułów poświęconych jest tematyce badań ludzkiego mózgu oraz reakcji na bodźce. Leen Van Beek wraz z zespołem badaczy w swojej publikacji [2] przedstawia badania dotyczące czasu reakcji oraz liczby błędów popełnionych w działaniach arytmetycznych u dzieci w zależności od wielkości sumy działania. W swojej publikacji dowodzą, że działania, których wynikiem była duża liczba powodują dłuższy czas reakcji (średnio 1129ms do 1707ms) oraz to, że tego typu operacje powodują więcej błędów. Guangming Ran wraz z Xu Chen w swojej pracy badawczej[3] zajmują się rozpoznawaniem obrazów w zależności od poziomu lęku społecznego. W swojej publikacji stwierdzili, że uczestnicy z wysokim poziomem lęku społecznego, rozpoznawali gniewne twarze dokładniej niż szczęśliwe twarze. W przypadku komponentów P100 i P200 uczestnicy wykazywali zwiększoną aktywność mózgu w odniesieniu do niezadowolonych twarzy w porównaniu ze szczęśliwymi twarzami, co sugeruje nadmierną czujność wobec tych pierwszych. Wiele przeprowadzanych dotychczas badań wykorzystujących elektroencefalografię dotyczy reakcji na bodźce dźwiękowe. K. N. Spreckelmeyer w swojej pracy [4] bada, czy mózg jest zdolny do rozróżniania tonów o zupełnie odmiennej ekspresji emocjonalnej poprzez rejestrowanie potencjałów wywołanych. Kolejny artykuł dotyczący dźwięków został zredagowany pod nadzorem Huisheng L. W swojej pracy [5] badacze dowodzą, że rytm muzyki ma

wpływ na zmiany nastroju człowieka, zaczynając od szczęścia, a kończąc na poddenerwowaniu i dyskomforcie. Z kolei Yun - Hsuan eChang opisuje badanie, które sprawdza, czy stan oczu (zamknięte/otwarte) ma wpływ na odbiór bodźców dźwiękowych oraz aktywność mózgu. W publikacji [6] stwierdzono, że uczestnicy ocenili dźwięki jako bardziej przyjazne w sytuacji gdy oczy były otwarte. Mith Smike w swoich badaniach [7] powiązał aktywność fizyczną z aktywnością mózgu. Zbadał jaki wpływ na zadanie Go/NoGo (reaguj/nie reaguj) mają ćwiczenia krótkotrwałe, umiarkowane oraz o wysokiej intensywności. Badanie polega na reagowaniu na konkretny, ustalony wcześniej bodziec oraz powstrzymaniu się od reakcji w momencie wystąpienia innego rodzaju bodźca. Wyniki poznawcze mierzono w trzech sesjach trwających po 10 minut. Po przeprowadzeniu badań stwierdzono, że trening o wysokiej intensywności pogarsza wyniki RT u stale aktywnych dorosłych w porównaniu z ćwiczeniami relaksacyjnymi lub z ćwiczeniami o umiarkowanej intensywności. Tradycyjnie techniki EEG wykorzystuje się przede wszystkim w badaniach medycznych. Andreas Mueler wraz z zespołem badaczy w swoim artykule [8] skupił się na badaniu ADHD przy pomocy potencjałów wywołanych. Na grupie badanych w których znajdowało się 148 osób (74 zdrowe i 74 chore) przeprowadzono badanie Go/NoGo. Badania wykazały, że osoby cierpiące na ADHD popełniają większą liczbę błędów w stosunku do osób zdrowych.

2.1. Potencjały wywołane

Pojęciem potencjałów wywołanych określa się aktywność elektryczną mózgu spowodowaną bodźcem bądź zdarzeniem. Wyróżniamy potencjały słuchowe, wzrokowe oraz somatosensoryczne. Obserwując sygnał EEG można zauważyć charakterystyczne przebiegi. Są one wywołane aktywnością mózgu spowodowaną procesami myślowymi bądź koncentracją uwagi na obiekcie lub wykonywanej czynności. Aby wykorzystać potencjały do badań mózgu stosuje się metodę uśredniania. Zakłada ona, że na sumaryczną aktywność ludzkiego mózgu (sumaryczny sygnał EEG przesyłany przez elektrody) składają się specyficzne wzorce aktywności związane z przetwarzaniem konkretnych bodźców. Uśrednianie sygnału dla wielu prób spowoduje zatarcie się zsumowanego, niespecyficznego sygnału i wyodrębnienie na pierwszy plan sygnału specyficznego, związanego z przetwarzaniem określonych bodźców [9]. Mówiąc o potencjałach wywołanych należy wspomnieć o załamkach. Jest to pojęcie ściśle związane z badaniem aktywności mózgu. Pojęciem załamka określa się przejściowy wzrost bądź spadek potencjału. Załamki określane są konkretnymi literami. Literą P w przypadku, gdy następuje wzrost potencjału, literą N w przypadku spadku potencjału. Pojęciem latencji określa się przybliżoną liczbę milisekund, liczoną od momentu wystąpienia danego bodźca do chwili pojawienia się potencjału. Najbardziej charakterystycznym oraz najprostszym do zaobserwowania jest potencjał P300. Pojawia się on najczęściej około 300ms od chwili wystąpienia bodźca. Jest to rodzaj potencjału endogennego, czyli takiego, który wykazuje bardziej złożone reakcje na bodźce. Przeciwieństwem są potencjały egzogenne, które są niezależne od uwagi badanego. P300 można zaobserwować głównie w sygnale pochodzącym z obszarów mózgowych odpowiedzialnych za pamięć, podejmowanie decyzji oraz przetwarzanie zadań. Kolejny charakterystyczny załamek to N400. Ma związek z procesami językowymi i charakteryzuje się specyficznymi warunkami w jakich można zarejestrować odpowiedź. Załamek można wywołać w paradygmacie określanym jako semantyczny. Polega na tym, że osobie badanej w losowej kolejności prezentuje się dwa typy zdań: pierwsze poprawne semantycznie ("Posmarowałem bułkę masłem"), oraz drugie niepoprawne semantycznie ("Posmarowałem bułkę biurkiem"). Rejestrując odpowiedź mózgu na każdy rodzaj zdań, można zauważyć, że zdania poprawne nie wywołują odpowiedzi N400, a duże prawdopodobieństwo wystąpienia powstaje w sytuacji wypowiedzenia ostatniego słowa ze zdania niepoprawnego semantycznie. Rys. 1 przedstawia najbardziej popularne rodzaje załamków.



Rys. 1. Rodzaje załamków [10]

2.2. Artefakty

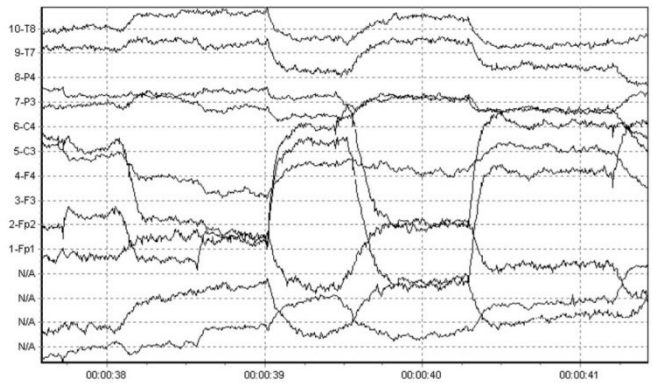
Nieodłącznym elementem każdego sygnału EEG są artefakty. Pojęciem tym określa się zniekształcenia rzeczywistego przebiegu fal mózgowych. Ze względu na źródło pochodzenia wyróżnia się artefakty techniczne oraz biologiczne. Źródłem powstawania artefaktów technicznych może być sieć energetyczna bądź aparatura medyczna. Artefakty biologiczne są spowodowane zachowaniem organizmu badanego pacjenta podczas przeprowadzania eksperymentu. Źródłem tego typu artefaktów jest ruch gałki ocznej (Rys. 2), ruch mięśni szkieletowych, drżenie głowy, pulsacja tętnicza, wydzielanie się potu na skórze lub zaciskanie zębów. Ogólnie źródłem tego typu artefaktów są wszystkie organy oprócz mózgu. Największe zakłócenia powoduje ruch gałką oczną. Z tego względu, aby uzyskać prawidłowy pomiar ważny jest ogólny stan psychiczny i fizyczny badanego. Istnieje wiele metod umożliwiających korekcję artefaktów. Najpopularniejsze metody to PCA (Principal Component Analysis) oraz ICA (Independent Component Analysis). Ogólnodostępne oprogramowanie (WinEEG) przeznaczone do analizy udostępnia wiele opcji pozwalających korygować powstałe artefakty.

3. Metoda badań

Badanie polegało na zarejestrowaniu reakcji na bodźce wzrokowe i słuchowe. Późniejsza analiza uzyskanych czasów pozwala stwierdzić, które reakcje są szybsze: wzrokowe czy słuchowe. Badaniu poddanych zostało 6 osób różnej płci w przedziale wiekowym od 18 do 25 lat.

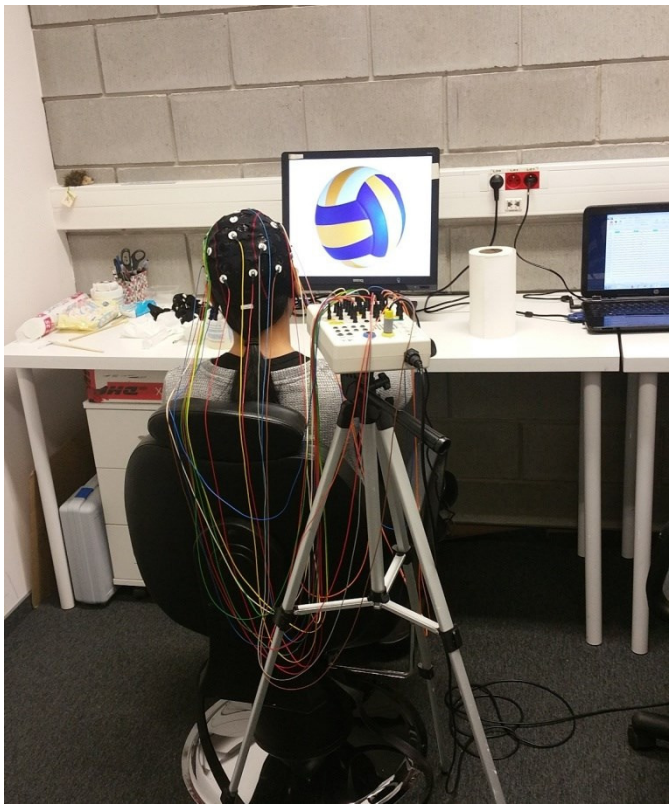
Do przeprowadzenia prawidłowego badania niezbędne było skonstruowanie eksperymentów, które pozwoliły uzyskać dane konieczne do analizy. Badanie składało się

z dwóch etapów. W pierwszym etapie pozyskano czasy reakcji na bodźce wzrokowe. Drugi etap miał na celu uzyskanie czasów reakcji na bodźce słuchowe.



Rys. 2. Artefakty powstałe w wyniku ruchu gałką oczną [11]

Przed rozpoczęciem eksperymentu każdy badany został zapoznany z instrukcją opisującą w jaki sposób przebiegać będzie dany eksperyment. W badaniu dotyczącym reakcji wzrokowych (Rys. 3) na ekranie monitora wyświetlane były rysunki.



Rys. 3. Badana osoba podczas eksperymentu wzrokowego

Dla ustalonego rodzaju obrazka użytkownik miał za zadanie wcisnąć lewy przycisk myszy. Eksperyment słuchowy przebiegał w analogiczny sposób. W momencie usłyszenia konkretnego, wcześniej ustalonego dźwięku badany musiał wcisnąć lewy przycisk myszy. Każdy z eksperymentów składał się z objaśnienia badania oraz serii nieuporządkowanych, następujących po sobie prób. Dla eksperymentu badającego reakcje wzrokowe wykorzystano piłki występujące w popularnych grach drużynowych, takich jak: piłka nożna, siatkówka, koszykówka oraz baseball. Badany miał za zadanie wcisnąć lewy przycisk myszy w momencie pojawienia się piłki do gry w piłkę nożną. Przyjęto, że jest to reakcja Go. Za próbę Nogo uważa się sytuację kiedy badany nie wcisnął przycisku w momencie pojawienia się piłki do gry w piłkę nożną oraz gdy przycisk wciśnięto dla innego obrazka. W analogiczny sposób skonstruowano eksperyment słuchowy. Jedyną różnicą jest to, że do badania wykorzystano inny rodzaj bodźców. Są to dźwięki popularnych instrumentów takich jak: pianino, gitara basowa, saksofon oraz syntezator. Zadaniem badanego była reakcja na niski dźwięk gitary basowej. Eksperyment zarówno dla części wzrokowej jak i słuchowej składał się ze 150 prób w których 135 było oznaczone jako reakcja NoGo, a 15 jako Go. Czas ekspozycji bodźca wynosił 1500 ms. Opóźnienia między próbami wynosiły 500 ms. Tabela 1 przedstawia fragment wyników eksperymentu wzrokowego. Zawiera ona numery kolejnych bodźców, rodzaj reakcji (Go-1, NoGo-2) oraz czasy reakcji uzyskane przez badanego dla poszczególnych eksperymentów.

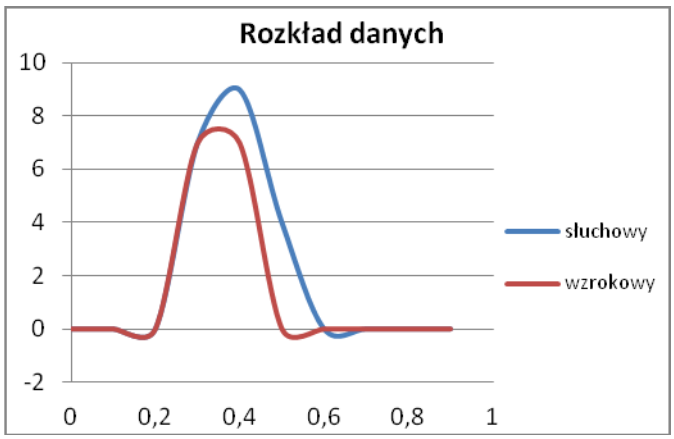
4. Uzyskane wyniki

Przeprowadzone eksperymenty pozwoliły uzyskać dwa zestawy danych dla każdego badanego. Dzięki wykorzystaniu oprogramowania WinEEG możliwe było wyeksportowanie uzyskanych wyników do pliku tekstowego, a następnie do pliku xlsx.

Tabela 1. Fragment tabeli przedstawiającej czasy reakcji dla poszczególnych eksperymentów

Eksperyment wzrokowy			Eksperyment słuchowy		
Nr. bodźca	Go/NoGo	Czas (ms)	Nr. bodźca	Go/NoGo	Czas (ms)
1	2	0	1	2	0
2	2	0	2	2	0
3	2	0	3	2	0
4	2	0	4	2	0
5	2	0	5	2	0
6	1	324	6	2	0
7	2	0	7	2	0
8	2	0	8	1	332
9	2	0	9	2	0

Dalsza analiza uzyskanych wyników pozwoliła stwierdzić, że dane mają rozkład normalny, zarówno w przypadku eksperymentu wzrokowego jak i słuchowego. Ilustruje to wykres znajdujący się na Rys. 5.

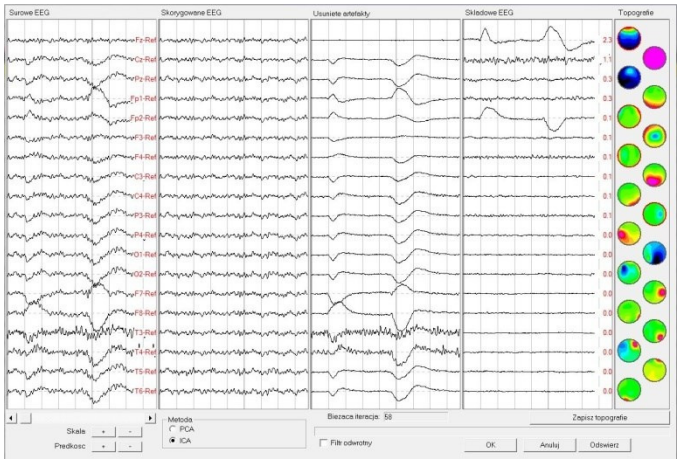


Rys. 5. Rozkład uzyskanych danych

Dzięki korekcji artefaktów oraz przeprowadzonej analizie uzyskano wykresy ERP oraz widma mocy dla poszczególnych części mózgu.

4.1. Korekcja artefaktów

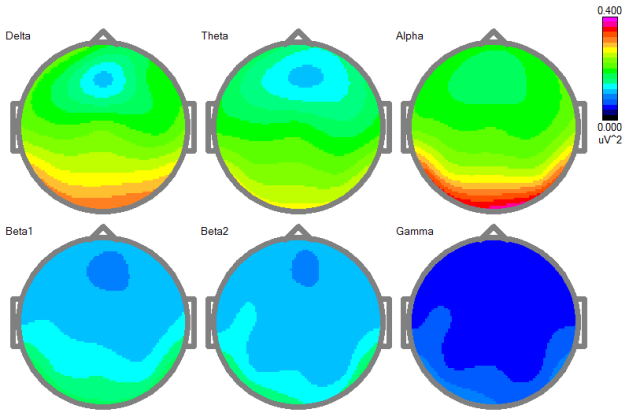
Aby przejść do dalszego etapu analizy wymagane jest usunięcie powstałych artefaktów. Spośród dostępnych metod korekcji najlepiej sprawdziła się metoda ICA. Dzięki wykorzystaniu oprogramowania WinEEG możliwe było uzyskanie przebiegów pozbawionych zakłóceń. Rys. 6 przedstawia surowy przebieg EEG oraz przebieg pozbawiony artefaktów. Już na pierwszy rzut oka widać, że wykorzystana metoda w znacznym stopniu poprawia jakość sygnału.



Rys. 6. Korekcja artefaktów metodą ICA

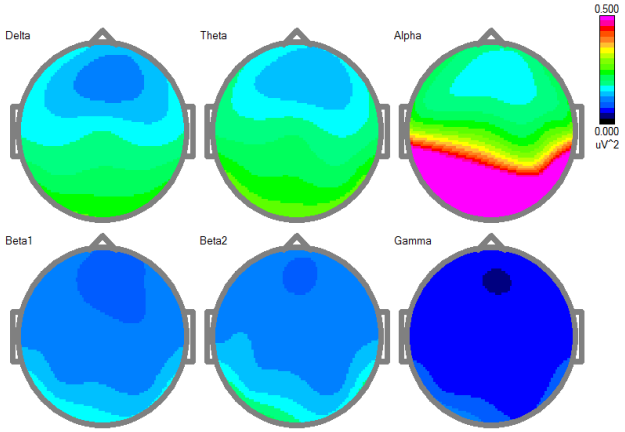
4.2. Mapy mocy

Mapy mocy widma EEG pokazują moc sygnału w poszczególnych fragmentach mózgu. Obserwując poniższe ilustracje można zauważyć, że moc sygnału zmienia się w zależności od wykonywanych czynności oraz rodzaju eksperymentu.



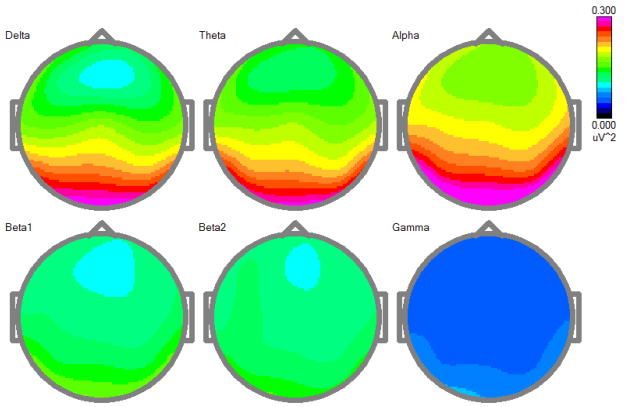
Rys. 7. Mapy mocy oczy otwarte

Mapy umieszczone na Rys. 7 przedstawiają zwiększoną aktywność płata potylicznego. Badany poproszony był o skupienie się na ekranie monitora oraz zrelaksowanie. Zwiększona aktywność płata potylicznego związana jest z wykorzystaniem zmysłu wzroku do obserwowania monitora. Dominują fale alfa. Oznacza to, że pacjent był zrelaksowany.



Rys. 8. Mapy mocy oczy zamknięte

Rys. 8 przedstawia aktywność mózgu badanego gdy ten miał zamknięte oczy, Można zauważyć spadek mocy w płacie czołowym. Aktywność wykazują głównie fale alfa. Są to fale spoczynkowe, charakterystyczne dla stanu odprężenia, gdy oczy badanego są zamknięte.



Rys. 9. Mapy mocy eksperyment wzrokowy

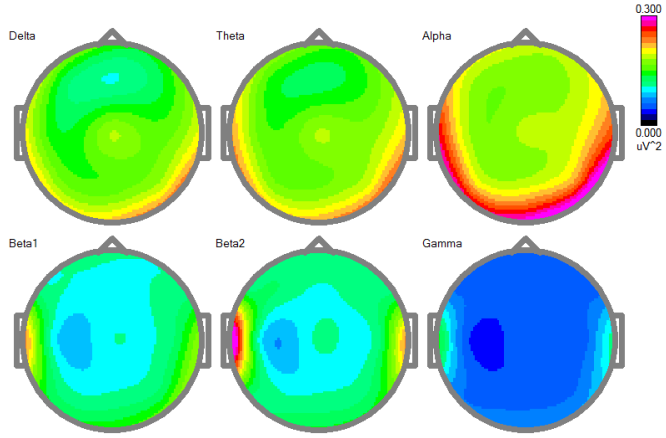
Dla eksperymentu wzrokowego przedstawionego na Rys. 9 mapa widma mocy ukazuje zdecydowanie zwiększoną aktywność w płacie potylicznym. Oznacza to, że badany w szczególny sposób wykorzystuje zmysł wzroku. W pewnym stopniu zwiększyła się aktywność fal beta co oznacza, że mózg badanego pracuje intensywniej podczas eksperymentu. W badaniu dotyczącym reakcji na bodźce słuchowe pominięto pierwsze dwa etapy. Mapa przedstawiona na Rys. 10 obrazuje aktywność mózgu podczas badania reakcji na bodźce dźwiękowe.

Mapa widma mocy przedstawia największą aktywność w płatach skroniowych. Są one odpowiedzialne za zmysł słuchu. Badanie dotyczyło reakcji na dźwięki, stąd zwiększona aktywność w tych rejonach mózgu.

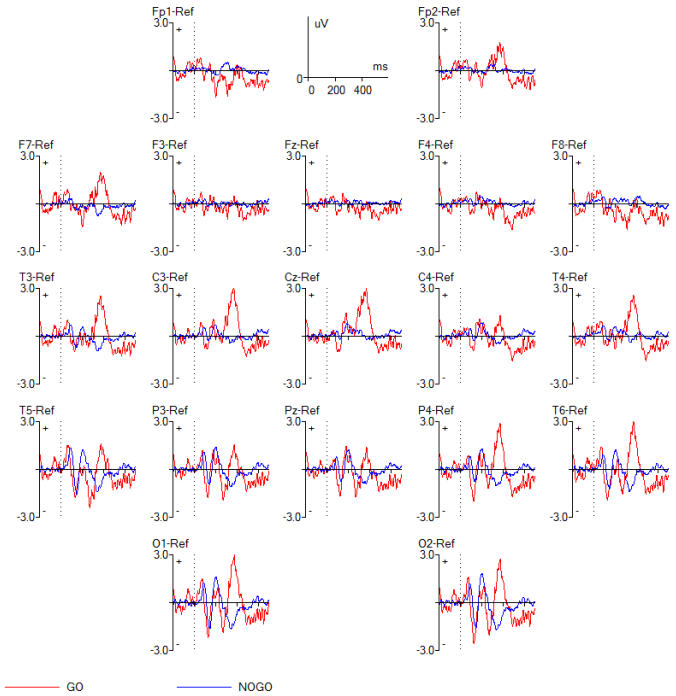
4.3. Wykresy ERP

Wykresy przedstawiają uśrednioną wartość potencjałów wywołanych dla wszystkich prób. Przerywana linia oznacza moment pojawienia się bodźca. Na wykresach oznaczono czas przed pojawieniem się bodźca (200 ms) oraz czas po pojawieniu się bodźca (800 ms). Na wykresach nie umieszczono linii wskazującej moment końca bodźca. Tego typu zapis pozwala dokładniej określić moment wystąpienia

potencjału wywołanego. Obserwując wykresy można zauważyć, że następuje to około 300 ms od momentu pojawienia się bodźca. Spowodowane jest to opóźnieniem, które powstaje podczas docierania bodźca do mózgu oraz kliknięcia w przycisk myszy. Wykresy przedstawiono na Rys. 11 oraz Rys. 12.



Rys. 10. Mapy mocy eksperyment słuchowy

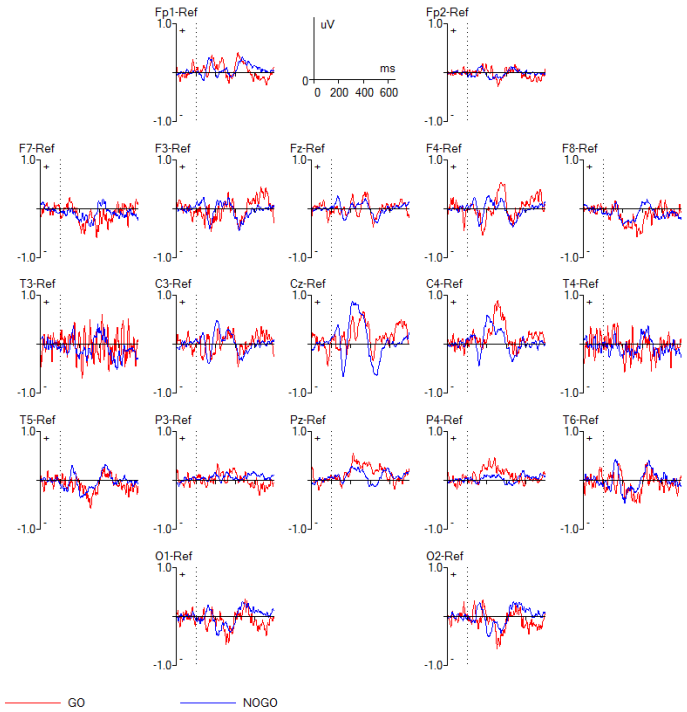


Rys. 11. Wykres ERP eksperyment wzrokowy

4.4. Wykresy pudełkowe

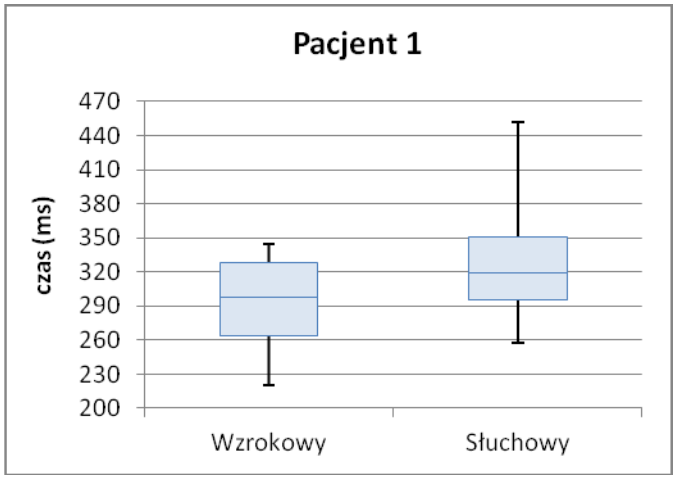
Wykres pudełkowy (wykres skrzynkowy) służy do prezentacji wyników lub porównywania danych. Zawiera informacje odnośnie położenia, rozproszenia i rozkładu uzyskanych danych. Do wykonania wykresu potrzebne są wartości: maksymalna, minimalna, pierwszy kwantyl, trzeci kwantyl oraz mediana.

Na Rys. 13 przedstawiono dane uzysane dla losowego pacjenta. W przypadku eksperymentu wzrokowego mediana wynosi 298ms, wartość minimalna 220ms, a maksymalna 344ms. Wyniki skupiają się w przedziale od 260 do 310 ms. Średni czas reakcji na bodziec wzrokowy wyniósł 298ms.



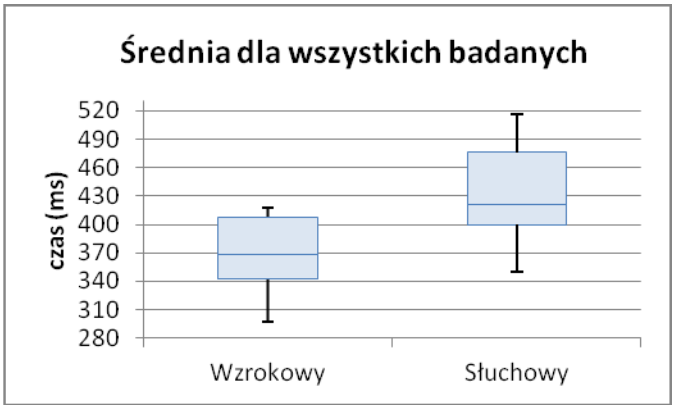
Rys. 12. Wykres ERP eksperyment słuchowy

Z kolei dla eksperymentu słuchowego mediana wynosi 328ms, wartość minimalna 260ms, a maksymalna 452ms. Wyniki skupiają się w przedziale od 300ms do 350ms. Średni czas reakcji na bodziec słuchowy wyniósł 351ms. Kolejny wykres przedstawiony na Rys. 14 obrazuje dane uzyskane dla wszystkich badanych. Wykresy zbudowano na podstawie średnich wartości eksperymentu wzrokowego i słuchowego dla każdego badanego.



Rys. 13. Wykres pudełkowy dla losowego badanego

W przypadku uśrednionych danych dla eksperymentu wzrokowego mediana wynosi 368ms, wartość minimalna 297ms, a maksymalna 417ms. Wyniki skupiają się w przedziale od 340 do 410ms. Średniczas reakcji na bodziec wzrokowy wyniósł 367ms. Z kolei dla uśrednionyh danych dla eksperymentu słuchowego mediana wynosi 421ms, wartość minimalna 351ms, a maksymalna 516ms. Wyniki skupiają się w przedziale od 400 do 480ms. Średni czas reakcji na bodziec słuchowy wyniósł 433ms.



Rys. 14. Wykres pudełkowy dla wszystkich badanych

Po stwierdzeniu, że uzyskane dane mają rozkład normalny, możliwe było przeprowadzenie testu T. Pozwala to określić czy dane są od siebie różne w sensie statystycznym. Z przeprowadzonych obliczeń w programie excel wynika, że dla danych losowego pacjenta poziom prawdopodobieństwa p wyniósł 0,03874. Podobna sytuacji wystąpiła dla średnich czasów wszystkich badanych. W tej sytuacji poziom prawdopodobieństwa p wyniósł 0,0021. Są to wartości mniejsze od klasycznego poziomu istotności (0,05). Uzyskane wyniki pozwalają odrzucić hipotezę o braku różnic między grupami

5. Wnioski

Analizując uzyskane wyniki można odpowiedzieć na pytanie, które reakcje są szybsze. Biorąc pod uwagę dane dotyczące pojedynczego badanego, reakcje na bodźce wzrokowe były szybsze o około 67ms. Sytuacja była podobna dla reszty badanych. Różnica wynosiła od 60 do 80ms. Analizując średnie wyniki dla wszystkich badanych sytuacja wygląda podobnie. Reakcja na bodziec wzrokowy jest szybsza o 66ms.

Przeglądając wykresy ERP można zauważyć, że lepszym rozwiązaniem byłoby skrócenie czasu ekspozycji bodźca. W takiej sytuacji na wykresie znalazłby się moment początku oraz końca ekspozycji bodźca. Przedstawione w artykule badanie ogranicza się do oznaczenia początku ekspozycji. Wpływa to na poprawienie czytelności wykresu oraz pozwala ukazać moment pojawienia się potencjału wywołanego. Obserwując wykresy ERP można zauważyć, że największy wzrost następuje około 300 ms od momentu pojawienia się bodźca. Jest to związane z pojawieniem się potencjału P300, który jest charakterystyczny dla sytuacji podejmowania decyzji oraz przetwarzania zadań, co ma miejsce podczas przeprowadzania eksperymentu.

Literatura

[1] J. Rowan, E.Tolunsky, A. Sobieszek, Podstawy EEG z miniatlasem, wyd. 1, Elsevier Urban and partner, Wrocław 2004, 4-5.

[2] L. eVan Beek, P. eGhesquiere, B. eDe Smedt, L. eLagae, The arithmetic problem size effect in children: an event-related potential study Frontiers in Human Neuroscience, 2014.

[3] G. Ran, X. Chen, The Impact of Top-Down Prediction on Emotional Face Processing in Social Anxiety, Frontiers in Psychology, 2017.

[4] T. Münte, K. eSpreckelmeyer, E. eAltenmüller, H. eColonius, Preattentive processing of emotional musical tones: a multidimensional scaling and ERP study, Frontiers in Psychology, 2013.

[5] L. Huisheng, W. Mingshi, Y. Hongqiang, EEG Model and Location in Brain when Enjoying Music, Proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference Shanghai, China, September 1-4, 2005.

[6] Y. eLee, Y. eChang, S. Hsieh , Experiencing affective music in eyes-closed and eyes-open states: an Electroencephalography study, Frontiers in Psychology, 2015.

[7] M. Smith, J. Tallis, A. Miller, D. Clarke, L. Guimarães-Ferreira, M. Duncan, The effect of exercise intensity on cognitive performance during short duration treadmill running, Journal of Human Kinetics, 2016.

[8] A. Mueller, G. Candrian, J. Kropotov, V. Ponomarev, G. Baschera, Classification of ADHD patients on the basis of independent ERP components using a machine learning system, Mueller et al. Nonlinear Biomedical Physics, 2010.

[9] W. Kułak, Kierownik: prof. dr hab. n. med. W. Sobaniec Neurologia dziecięca, 15/2006 Nr 29.

[10] https://brain.fuw.edu.pl/edu/index.php/Plik:Zalamki.png [04.12.2017]

[11] User Tutorial:EEG Measurement Setup: http://www.bci2000.org/wiki/index.php/User_Tutorial:EEG_Measurement_Setup [04.12.2017]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 178-182
Wysłane: 2017-12-07
Przyjęte: 2017-12-11

Analiza użytkowa frameworka AngularJS w kontekście prostej aplikacji internetowej

Krzysztof Pawelec*

Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Celem artykułu było przeprowadzenie analizy możliwości użytkowych frameworka frontendowego AngularJS w porównaniu do języka programowania JavaScript, na którym bazuje. Wybrano kilka funkcjonalności frameworka i zestawiono je z samodzielnie zaimplementowanymi rozwiązaniami w JavaScript. Porównania dokonano według określonych kryteriów: prostoty użycia, możliwości wykorzystania i zdatności do wielokrotnego zastosowania. Utworzone skrypty potwierdzają przyjęte założenia, że jest możliwe napisanie w JavaScript uproszczonych i zdolnych do ponownego użytku implementacji przydatnych mechanizmów znajdujących się w AngularJS.

Słowa kluczowe: framework; użyteczność; AngularJS; JavaScript

*Autor do korespondencji.

Adres e-mail: krzysztof.pawelec1@pollub.edu.pl

Usability analysis of AngularJS framework in the context of simple internet application

Krzysztof Pawelec*

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

Abstract. The goal of this article was to perform analysis of usability possibilities of frontend framework AngularJS compared to native programming language JavaScript, on which it is based on. Several framework funcionalities were chosen and set together with self implemented solutions in JavaScript. Comparison was made according to specified criteria: usage simplicity, possibilities of utilization and reusability. Created scripts confirm accepted assumptions, that in JavaScript it is possible to write simplified and reusable implementations of useful mechanisms, which are present in AngularJS.

Keywords: framework; usability; AngularJS; JavaScript

*Corresponding author.

E-mail address: krzysztof.pawelec1@pollub.edu.pl

1. Wstęp

W ostatnich latach istnieje coraz większe zapotrzebowanie na aplikacje internetowe, gdzie duża część logiki umiejscowiona jest po stronie frontendu [1]. Aby przyspieszyć proces ich tworzenia powstały frameworki, które potrafią wykonać coraz więcej często realizowanych przez programistę czynności związanych głównie z budową struktury aplikacji i zarządzaniu danymi pomiędzy jej warstwami [2]. Warto zastanowić się nad tym, czy do każdej aplikacji – bez względu na skomplikowanie i wielkość - potrzeba dołączać framework. Z uwagi na udostępnianie wielu funkcjonalności, nierzadko wymuszenie zastosowania określonej struktury aplikacji i posiadania zdolności do kontrolowania zmiany danych w jej obrębie, pliki źródłowe frameworków zawierają bardzo dużą ilość kodu, a zatem zajmują więcej miejsca na dysku.

Z racji zwiększonego nacisku na wydajność, skalowalność i responsywność aplikacji na popularności zyskują metody dynamicznego tworzenia treści i doładowywania do niej danych z serwera w razie potrzeby, co znacząco zmniejsza konieczność odświeżania strony [3]. Przykładem jest model *Single Page Application*. Stosowanie go sprawia, że przy przełączaniu podstron ich zawartość ładuje się szybciej, przez co wydawać by się mogło jakby aplikacja była zbudowana z pojedynczej strony. Jednym z powodów jest przesyłanie z hosta mniejszej ilości odpowiednio opisanych danych, które są przygotowane do wyświetlenia i umieszczane w widoku w warstwie logiki aplikacji klienckiej [4]. Proces ten, w zależności od potrzeb, wymaga implementacji mniej lub bardziej złożonych funkcji.

Celem artykułu było porównanie wybranych funkcjonalności AngularJS w odniesieniu do własnych implementacji tych mechanizmów utworzonych w języku JavaScript.

Wykonane skrypty pomogły zweryfikować hipotezy postawione w artykule:

- własne implementacje mechanizmów dostępnych we frameworku mogą być stosowane w prostych przypadkach podczas budowy aplikacji,
- własne implementacje porównywanych funkcjonalności mogą być reużywalne – czyli zdatne do wielokrotnego użycia dla danych wejściowych dostarczonych w akceptowanej przez mechanizm formie.

1.2. Obszary badawcze

AngularJS oferuje wiele przydatnych narzędzi do tworzenia aplikacji typu *Single Page*, m.in.: router, serwis *$http* odpowiedzialny za łączność z zewnętrznymi hostami, zautomatyzowany *cache* przechowujący raz utworzone dane

w celu szybszego dostępu do nich, walidatory formularzy, serwis *$sce* zapewniający podstawową ochronę przed atakami *XSS* [6].

Z racji braku dostatecznego miejsca w niniejszym artykule, do porównania wybrano następujące funkcjonalności służące do dynamicznego budowania warstwy treści w aplikacji:

1) interpolacja wyrażeń,
2) powielanie elementów HTML.
3) monitorowanie zmian danych w warstwach treści i logiki.

Poszczególne funkcjonalności aplikacji będą najpierw zademonstrowane z użyciem frameworka AngularJS, a potem przedstawiona zostanie implementacja w języku JavaScript. Na koniec każdego badania zestawione będą ze sobą, w formie tabeli, subiektywne oceny sprawdzonych funkcjonalności na podstawie kryteriów:

- prostota użycia,
- możliwości zastosowania,
- reużywalność.

Punkty przyznawane będą w skali od 1 – 5, gdzie 1 oznacza najgorszą a 5 najlepszą ocenę.

Warto wspomnieć, że AngularJS posiada wbudowany mechanizm kompilacji kodu HTML [5]. Jego rolą jest rekursywne iterowanie po wskazanych węzłach DOM i sprawdzanie, czy dodane na nim atrybuty mają nazwę zapisaną w odpowiedniej konwencji. W pozytywnym przypadku wartość atrybutu zostaje połączona z odpowiadającym jej polem lub metodą w obiekcie *$scope*. Ten z kolei stanowi łącznik pomiędzy modelem – w nim zlokalizowane są skrypty z logiką aplikacji – a widokiem – prezentuje on przetworzone przez framework wartości do wyjściowej formy, którą widzi użytkownik. Atrybuty HTML zawierające w nazwie przedrostek *ng-*, po kompilacji, stanowią w AngularJS dyrektywy. Ich rolą jest informowanie frameworka o sposobie i ewentualnym przebiegu procesu dynamicznej modyfikacji określonego elementu HTML [6].

Na potrzeby analizy zaimplementowano w JavaScript uproszczony proces przeszukiwania drzewa DOM i odnajdywania elementów z atrybutami o specyficznej nazwie.

Interpolacja jest przydatna podczas umieszczania dopasowanych do konkretnego fragmentu tekstu wartości z modelu danych.

Składają się na nią z reguły dwa znaczniki – otwierający i zamykający. Obejmują one wyrażenia języka JavaScript oraz ewentualne fragmenty tekstu. Dzięki temu parser HTML potrafi rozpoznać miejsce, w którym powinien ewaluować występujące tam dane w określonym kontekście, którym w jest obiekt *$scope* [6].

Przykład 1. Kod HTML dla interpolacji wyrażeń w AngularJS.

```
<ul><li>[{{technology}}] Jest to {{text + " " +
getIncrementedNumber()}} przykładowego tekstu.</li>
<li>[{{technology}}] Jest to {{text + " " +
getIncrementedNumber()}} przykładowego tekstu.</li>
<li>[{{technology}}] Jest to {{text + " " +
getIncrementedNumber()}} przykładowego tekstu.</li>
</ul>
```

W elementach <li>, z przykładu nr 1, umieszczono tekst ze znacznikami interpolacji:„{{” oraz „}}”. Pomiędzy nimi znajdują się nazwy pól, które w modelu mają przypisane odpowiednie wartości.

Skrypt w modelu zawierał surowe użycie $interpolate. Podczas zwyczajnego korzystania z frameworka nie ma potrzeby odwoływania się do tego serwisu, ponieważ jest on wywoływany automatycznie.

Przykład 2. Kod skryptu dla interpolacji wyrażeń w AngularJS.

```
var context, interpolate;
var elementWithInterpolation = document.querySelector(‘ul');
angular.injector(['ng']).invoke(function($interpolate,
$rootScope) {
interpolate = $interpolate;
context = $rootScope;
context.technology = 'AngularJS';
context.text = 'linijka numer';
context.iterator = 0;
context.getIncrementedNumber = function() {
return ++context.iterator;
}; });
for (var i = 1; i <= 3; i++) {
angular.element(elementWithInterpolation).html(interpolate(e
lementWithInterpolation.innerHTML)(context));
}
```

W przykładzie 2 zadeklarowano zmienne pomocnicze **context** oraz **interpolate**. Służą one za aliasy, do których przypisywane są odpowiednio serwisy: *$interpolate* i *$rootScope*. Użycie interpolacji sprowadziło się do wywołania serwisu *$interpolate*, gdzie przekazano kod HTML z listy *<ul>*. Przetworzoną zawartość umieszczono z powrotem w liście.

Do implementacji algorytmu interpolacji w JavaScript (przykład nr 3) zastosowano wyrażenia regularne, dzięki którym można wyszukać i, w razie potrzeby, zamienić wystąpienia określonych znaków w tekście [7]. Kod HTML nie uległ zmianom.

Przykład 3. Kod skryptu dla interpolacji w JavaScript.

```
var context = {
text: 'linijka numer',
technology: 'JavaScript',
iterator: 0, getIncrementedNumber: function() {
return ++this.iterator;
} };
var elementWithInterpolation = document.querySelector('ul');
function customInterpolate(text, expression) {
var matchValueBetweenBrackets = /\{{(.*?)\}}/g;
var matchSeparatorsNotSurroundedByQuotes =
/\s*\+\s*"([^"]*)"\s*\+\s*/;
var matchValueAndFunctionInvocation = /\w+(\(\))?/g;
return text.replace(matchValueBetweenBrackets,
function(match, foundSubstring) {
var expressionsWithoutSeparators = foundSubstring.
replace(matchSeparatorsNotSurroundedByQuotes, '$1');
return evaluatedValues =
expressionsWithoutSeparators.replace(matchValueAndFunctio
nInvocation, function(match) {
return match.indexOf('(') !== -1 ? expression[match.slice(0, -
2)]() : expression[match];
}); } }
function appendToDOM(content, parent) {
parent.innerHTML = content;
}
for (var i = 1; i <= 3; i++) {
```

```
appendToDOM(customInterpolate(elementWithInterpolation.innerHTML, context), elementWithInterpolation);
}
```

Własna implementacja interpolacji nie jest łatwa. Trudność polega na odpowiednim dobraniu wyrażeń regularnych, a od ich liczby i skomplikowania zapisu zależy wydajność działania mechanizmu. Mimo, że rozróżnia on pola kontekstu od metod, to obsługuje tylko separatory w postaci spacji.

Natywna implementacja nie jest też w stanie obsłużyć sytuacji, gdy w wyrażeniu będzie ulokowane działanie matematyczne. Serwis $interpolate waliduje typy danych wejściowych - wartości innego typu niż string są na niego rzutowane [8].

Interpolacja w AngularJS jest znacznie bardziej rozbudowana, gdyż wyrażenia w niej umieszczane są obserwowane przez pętlę $digest. Akceptowane są ścieżki do wyświetlania pól i wywoływania metod z zagnieżdżonych struktur danych. Framework posiada wbudowane zabezpieczenia przed atakami XSS w formie serwisów $sceProvider i $SanitizeProvider.

Tabela 1. Oceny kryteriów dla badania interpolacji.

Kryterium:	AngularJS:	JavaScript:
Prostota:	5	2.5
Możliwości:	4	2
Reużywalność:	4.5	3
Średnia:	**4.5**	**2.5**

2.2. Powielanie elementów HTML

Na stronach internetowych często spotkać można listy. Przeznaczone są do grupowania powiązanych ze sobą informacji, aby powiązać i przedstawić je w czytelny sposób. Język HTML oferuje specjalne znaczniki, które wyróżniają ten rodzaj treści. Są nimi: *<ul>*, *<ol>* i *<dl>*. Stosuje się je w zależności od formy jaką ma pełnić lista: nieuporządkowaną, uporządkowaną lub opisową [9].

Listy są dobrym przykładem elementów HTML, na których można pokazać proces powielania. AngularJS posiada przeznaczoną do tego dyrektywę *ng-repeat*. Framework udostępnia za jej pośrednictwem pomocne zmienne, m.in.: *$index*, *$first*, *$last*, które informują o aktualnym indeksie elementu oraz czy zajmuje on pierwsze, czy ostatnie miejsce w kolekcji źródłowej [10].

Przykład 4. Kod HTML dla powielania elementów w AngularJS.

```
<div ng-app="test" ng-controller="ctrl">
<ul>
<li ng-repeat="line in twoLines">[{{$index + 1}}]Pierwsza lista - na górze.</li> </ul>
<ul>
<li ng-repeat="line in fourLines">[{{$index + 1}}]Druga lista - na dole.</li>
</ul> </div>
```

Klonowanie elementów HTML w AngularJS realizowane jest za pośrednictwem dyrektywy *ng-repeat*. W kodzie z przykładu 4 zamieszczono w nich referencje do tablic **twoLines** i **fourLines** (utworzonych w przykładzie nr 5), po których następowała iteracja. Udostępniona z *ng-repeat* zmienna *$index* oznacza indeks aktualnie iterowanego elementu w tablicy.

Przykład 5. Kod skryptu dla powielania elementów w AngularJS.

```
angular.module('test', []).controller('ctrl', function($scope) {
$scope.twoLines = [1, 2];
$scope.fourLines = [1, 2, 3, 4];
});
```

W przypadku natywnej implementacji, kod HTML jest mniejszy, gdyż nie zawiera znacznika * * z dodatkowymi dyrektywami. W elementach *<ul>* znajdują się pojedyncze znaczniki *<li>* z atrybutami **data-lines**, w których wpisano liczbę linijek do wyświetlenia. Wewnątrz elementów wstawione zostały wyrażenia interpolacyjne oraz zwykły tekst.

W skrypcie z przykładu 6 posłużono się funkcjami **appendToDom** i **customInterpolate** użytymi w poprzednim przykładzie. Pierwszą z nich przebudowano, aby usuwała pierwszy, nieprzetworzony element listy, gdy całość będzie już przeformowana.

Przykład 6. Kod skryptu dla powielania elementów w JavaScript.

```
function appendToDOM(content, parent, clearParentContent)
{
if (clearParentContent)
parent.removeChild(parent.children[0]);
parent.appendChild(content);
}
var listRefs = document.querySelectorAll('li');
var context = {
lineNumber: 0,
getLineNumber: function() {
return ++this.lineNumber;
} };
for (var i = 0; i < listRefs.length; i++) {
var listRefsItem = listRefs[i];
for (var j = 0, len = listRefsItem.getAttribute('data-lines'); j < len; j++) {
var clonedListRefsItem = listRefsItem.cloneNode(true);
clonedListRefsItem.innerHTML = customInterpolate(listRefsItem.innerHTML, context)
appendToDOM(clonedListRefsItem, listRefsItem.parentNode, j === len - 1);
} }
```

Liczba docelowych kopii odczytywana jest z atrybutu **data-lines**. Zawartość elementu przekazywana jest do interpolacji. Po przetworzeniu tekst wstawiany jest do sklonowanego elementu *<li>*, po czym dodawany do rodzica, czyli listy *<ul>*.

Różnica względem AngularJS polega na innej numeracji w drugiej liście. Dyrektywa *ng-repeat* tworzy nowy *$scope*, więc każda z list ma liczby skupione w zasięgu lokalnym [10]. Natywna implementacja funkcjonalności w JavaScript nadaje liczby inkrementując je za każdym razem dla następnej listy. Aby temu zaradzić należało by np. rozszerzyć funkcję **customInterpolate** o resetowanie licznika dla każdej listy podczas ewaluacji interpolowanych wyrażeń.

Tabela 2. Oceny kryteriów dla badania powielania elementów HTML.

Kryterium:	AngularJS:	JavaScript:
Prostota:	4	3
Możliwości:	4.5	2.5
Reużywalność:	4.5	2
Średnia:	**4.33**	**2.5**

2.3. Monitorowanie zmiany danych w warstwach treści i logiki

Nieodłączną częścią aplikacji, w której użytkownik może wprowadzać dane i wchodzić w interakcje z jej widokiem, jest mechanizm monitorujący zmiany w obrębie danych. Framework posiada mechanizm, który umożliwia sprawdzanie, a w razie konieczności aktualizację, danych pod kątem spójności pomiędzy modelem a widokiem. Obserwacji poddać można wartości typu prymitywnego i obiekty, zarówno o płaskiej strukturze, jak i z licznymi zagnieżdżeniami. Oczywiście, wraz ze wzrostem skomplikowania rozkładu danych i ich liczby, czas tego procesu wydłuża się. Pętla *Digest Loop* jest najbardziej istotnym mechanizmem w AngularJS. Odpowiada za synchronizację danych w modelu i widoku, zarówno automatycznie gdy wartości podłączone są do obiektu *$scope*, jak i ręcznie przez dostępne metody: *$watch*, *$watchCollection* i *$watchGroup* [11][12].

Przykład 7. Kod HTML dla monitorowania zmian w danych w AngularJS.

```
<div ng-app="test" ng-controller="ctrl">
<label for="field">Określ długość listy:
<input id="field" ng-model="amount" type="number">
<ul>
<li ng-repeat="item in list; track by $index">{{$index + 1}}
element listy.
<button type="button" ng-
click="removeItem($index)">X</button>
</li>
</ul>
</div>
```

W przykładzie 7 użyto dyrektywy *ng-model*, która ustawia wartość z pola *$scope.amount* na taką, jaka znajduje się w atrybucie **value** elementu *<input>* [13]. Listę *$scope.list* można dynamicznie wydłużać i skracać. Jej długość kontrolowana jest przez liczbę wpisaną do wspomnianego pola. Dodatkowo kliknięcie w przycisk *<button>* usuwa jeden element z listy.

Przykład 8. Kod skryptu dla monitorowania zmian w danych w AngularJS.

```
angular.module('test', []).controller('ctrl', function($scope) {
$scope.amount = '';
$scope.list = [];
$scope.removeItem = function( index ) {
$scope.list.splice(index, 1);
$scope.amount--;
}
$scope.$watch('amount', function() {
$scope.list = new
Array(+$scope.amount).fill($scope.amount);
});
});
```

W modelu (przykład 8) zastosowano metodę *$scope.$watch*, gdzie wykryta zmiana w obserwowanym polu **amount** umożliwia zmianę aktualnie przypisanej tablicy do *$scope.list* na nową, o długości wpisanej przez użytkownika w znaczniku *<input>*. Metoda *$scope.removeItem* usuwa wybrany za pośrednictwem indeksu element z tablicy i dekrementuje wartość pola *$scope.amount*, które połączone jest z *<input>*..

HTML dla natywnej implementacji (przykład 9) ma znacznie mniej kodu niż w AngularJS. Element *<ul>* jest początkowo pusty, ponieważ później jest on uzupełniany dynamicznie.

Przykład 9. Kod HTML dla monitorowania zmian w JavaScript.

```
<label for="field">Określ długość listy:
<input id="field" type="number">
<ul></ul>
```

Skrypt z przykładu nr 10 korzysta z wcześniej przygotowanej funkcji **customInterpolate**.

Przykład 10. Kod skryptu dla monitorowania zmian w JavaScript

```
function removeFromDom() {
listRef.removeChild(listRef.children[listRef.children.length -
1]);
context.index--;
}
var context = {
index: 0,
getIndex: function() {
return ++this.index;
}
};
var list = [];
var element = document.createElement('li');
element.innerHTML = '{{getIndex()}} element listy.<button
type="button">X</button>';
var listRef = document.querySelector('ul');
var input = document.querySelector('input');
input.addEventListener('input', function(evt) {
var val = evt.target.value;
if (val < context.index) {
removeFromDom();
} else {
context.index = 0;
while(listRef.children.length) {
listRef.removeChild(listRef.children[listRef.children.length - 1]
);
}
for (var i = 0; i < val; i++) {
var clonedElement = element.cloneNode(true);
clonedElement.innerHTML =
customInterpolate(element.innerHTML, context);
listRef.appendChild(clonedElement);
}
}
});
document.querySelector('ul').addEventListener('click',
function(evt) {
if (evt.target.tagName === 'BUTTON') {
removeFromDom();
input.value = context.index;
}
})
```

Użyte zostały dwie metody *addEventListener* na elementach DOM. Służą one do reagowania na określone w parametrze zdarzenia, gdzie przy każdym wystąpieniu wywołują podaną funkcję zwrotną – tam przekazywany jest obiekt zdarzenia [14]. W aplikacji, metody nasłuchujące reagują na zdarzenia **input** oraz **click**. Ich obsługa w funkcjach zwrotnych dzieli się na trzy części. Podczas zmiany liczby w polu *<input>*, albo usuwany jest ostatni *<li>*, albo jeden jest dodawany na koniec listy. Kliknięcie w przycisk usuwa konkretny element. Wszystkie przypadki aktualizują licznik długości listy **context.index**.

Problemem podczas nasłuchiwania zdarzeń może być np. sytuacja, gdy obiekt DOM, na którym są podłączone zostanie usunięty – ponowne jego utworzenie nie spowoduje

wznowienia nasłuchiwania jeśli nowy element nie będzie tym samym co poprzedni. Można temu zaradzić przechowując usunięty element w cache. Dzięki zjawisku propagacji zdarzeń możliwe jest natomiast ograniczenie liczby metod nasłuchujących daną grupę elementów, gdy podłączy się jedną z nich do wspólnego kontenera [14].

Metody obecne w AngularJS obserwują wartości obiektu *$scope*. nie tylko w płaskiej strukturze danych, lecz także na głębokim poziomie zagnieżdżenia [12].

Tabela 3. Oceny kryteriów dla badania monitorowania zmiany danych.

Kryterium:	**AngularJS:**	**JavaScript:**
Prostota:	4.5	4
Możliwości:	4	3
Reużywalność:	4	3.5
Średnia:	**4.16**	**3.5**

Przedstawione w analizie skrypty udowadniają, że przydatne funkcjonalności, oferowane przez framework AngularJS, można, w uproszczonych wersjach, wprowadzić do aplikacji samemu. Obie hipotezy zostały potwierdzone, z zaznaczeniem, że aspekt reużywalności natywnie zaimplementowanych mechanizmów może być ograniczony.

Średnia punktów z trzech testów (tabele nr 1, 2 i 3) dla frameworka wynosi: 4.33pkt., zaś dla języka JavaScript: 2.83pkt. Nie ulega wątpliwości, że praca z AngularJS jest łatwiejsza, ponieważ istotne procedury wspomagające budowę aplikacji są przez framework wykonywane bez ingerencji programisty. Wyróżnić tu można: skomplikowany proces kompilacji drzewa DOM, synchronizacja modelu z widokiem oraz obsługa DOM – czyli restrukturyzacja i modyfikacja elementów widoku. Natomiast w przypadkach, gdy zachodzi konieczność realizacji niestandardowej funkcjonalności, framework zazwyczaj udostępnia narzędzia przydatne do tego celu.

Implementacja w czystym JavaScript własnych mechanizmów do obsługi tego, co AngularJS automatyzuje jest dosyć pracochłonna. W istocie wykonanie uproszczonych funkcji bazując na natywnym języku ogranicza możliwości przez nie oferowane, jak choćby walidacja wartości wejściowych lub niezależny zakres dostępności dla danych. Nie wykluczone, że może pojawić się potrzeba rozszerzenia konkretnego mechanizmu. Nie mniej, dodanie takiego rodzaju API jest możliwe i warto być tego świadomym.

Rezygnacja lub ograniczenia używania frameworka w dobrze dobranych przypadkach zmniejszy ilość danych, które przeglądarka musi pobrać oraz przyspieszy działanie aplikacji. Bowiem, mimo że metody z API frameworka są lepiej dopracowane pod względem obsługi różnych przypadków użycia i oddają do dyspozycji programisty dużo bardziej rozbudowane mechanizmy niż ma to miejsce przy posługiwaniu się natywnym JavaScript, to są one w wydaniu AngularJS mniej wydajne [15]. Przykładowo, jeśli w modelu danych zostanie ustawionych przesadnie dużo wartości podłączonych do obiektu *$scope*, a także gdy znacząca liczba parametrów do dyrektyw i komponentów będzie przekazywana przez wiązania dwukierunkowe, spowoduje to obciążenie pętli *$digest*, która będzie sprawdzać dużą liczbę danych za każdym razem – nawet gdy zmiana nie będzie się bezpośrednio do nich odnosić [16]. Wpłynie to na opóźnienia w synchronizacji modelu z widokiem i nadmierne obciążenie podzespołów sprzętowych.

Nie należy zatem traktować funkcjonalności frameworka jako przedmiotu do nadużywania z powodu wygody, czy automatyzacji wszystkiego, co jest potencjalnie możliwe. W każdym przypadku – czy to skromnej aplikacji, czy rozbudowanej – warto jest przemyśleć co konkretnie będzie przydatne podczas procesu programowania i w miarę możliwości starać się ograniczać korzystanie z tych funkcjonalności, które pogarszają efektywność działania, albo zastąpić gotowe narzędzia mniejszymi odpowiednikami lub napisać własne skrypty pomocnicze [17]. Albowiem, można zaimplementować mechanizmy w podstawowym stopniu odpowiadające tym dostępnym we frameworku i czerpać z nich korzyści przy zmniejszonym wpływie na szybkość działania aplikacji.

Literatura

[1] K. Nyg˚ard, Single page architecture as basis for web applications, Aalto University: School of Science, Espoo, 2015

[2] https://www.thebalance.com/what-is-a-front-end-framework-and-why-use-one-2071948 [27.11.2017]

[3] W. Chansuwath, T. Senivongse, A Model-Driven Development of Web Applications Using AngularJS Framework, Department of Computer Engineering at Chulalongkorn University, Bangkok, 2016.

[4] M. S. Mikowski, J. C. Powell, Single Page Web Application, Manning Publications Co., Shelter Island in New York, 2014

[5] https://docs.angularjs.org/guide/compiler [27.11.2017]

[6] A. Lerner, ng-book – The complete Book on AngularJS, Fullstack.io, 2013.

[7] J. Friedl, Mastering Regular Expressions, 3rd Edition, O'Reilly Media, 2006.

[8] https://docs.angularjs.org/guide/interpolation#how-the-string-representation-is-computed [27.11.2017]

[9] https://www.w3.org/wiki/HTML_lists [27.11.2017]

[10] https://docs.angularjs.org/api/ng/directive/ngRepeat [27.11.2017]

[11] https://docs.angularjs.org/guide/scope#scope-life-cycle [27.11.2017]

[12] https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$rootScope.Scope-methods [27.11.2017]

[13] https://docs.angularjs.org/api/ng/directive/ngModel [27.11.2017]

[14] S. Alimadadi, S. Sequeira, A. Mesbah, K. Pattabiraman, Understanding JavaScript Event-Based Interactions, Electrical and Computer Engineering at University of British Columbia, Vancouver, 2014.

[15] M. Ramos, M. Tulio Valente, R. Terra, AngularJS Performance: A Survey Study, Computing Research Repository - ArXiv, 2017

[16] M. Ramos, M. Tulio Valente, R. Terra, G. Santos, AngularJS in the Wild: A Survey with 460 Developers, Association for Computing Machinery, 2016

[17] http://blog.scalyr.com/2013/10/angularjs-1200ms-to-35ms/ [27.11.2017]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 183-190
Wysłane: 2017-12-14
Przyjęte: 2017-12-19

Analiza możliwości zastosowania platformy Xamarin do budowy aplikacji wieloplatformowych mobilnych

Michał Dras*, Grzegorz Fila, Małgorzata Plechawska-Wójcik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono platformę Xamarin wykorzystywaną do tworzenia aplikacji wieloplatformowych na systemy: Android, iOS, MacOS oraz platformę Universal Windows Platform. Omówiona została sama platforma, a także aplikacja testowa oraz badania przeprowadzone w celu przeanalizowania możliwości platformy i jej efektywności w budowaniu aplikacji. Badania dotyczyły części wspólnej kodu, rozmiaru aplikacji po zainstalowaniu oraz jej wydajności. Pokazały one, że z pomocą Xamarina możliwe jest tworzenie aplikacji o części wspólnej kodu powyżej 80%, przy zachowaniu niewielkiego rozmiaru i zadowalającej wydajności.

Słowa kluczowe: xamarin; aplikacje wieloplatformowe; systemy mobilne; c#

*Autor do korespondencji.

Adres e-mail: dras.michael@gmail.com

Analysis of Xamarin capabilities for building mobile multi-platform applications

Michał Dras*, Grzegorz Fila, Małgorzata Plechawska-Wójcik
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents Xamarin platform which is used to create cross-platform application for Android, iOS, MacOS and Universal Windows Platform. This article shows Xamarin platform and a test application that has been used to investigate platform's capabilities and effectiveness in creating multi-platform applications. Inspections prove that Xamarin allows to create multi-platform applications in a more effective way without losing too much on performance of these applications on individual platforms and systems.

Keywords: xamarin; multi-platform applications; mobile systems; c#

*Corresponding author.

E-mail address: dras.michael@gmail.com

1. Wstęp

Obecnie aplikacje mobilne są coraz popularniejsze. Jest to nieodzownie związane z ciągle zwiększającym się udziałem smartfonów w rynku urządzeń mobilnych. Wśród smartfonów największy udział mają urządzenia z systemem Android. Drugie miejsce zajmuje system iOS, natomiast trzecie Windows 10 Mobile. Każdy z wyżej wymienionych systemów wymaga oddzielnie pisanej aplikacji: Android wymaga aplikacji napisanej w Javie, iOS w Objective-C, a Windows 10 Mobile w C#. Tworzenie takiej samej aplikacji trzy razy jest mało efektywne. W celu rozwiązania tego problemu powstały aplikacje wieloplatformowe, które są pisane raz, ale działają na wielu platformach. Przykładem platformy wykorzystywanej do budowania aplikacji wieloplatformowych jest Xamarin [1].

Zdaniem autorów [2] wydajność aplikacji stworzonych z pomocą Xamarina jest porównywalna do wydajności aplikacji natywnych. Posiada on także dużą liczbę bibliotek, skracających czas pisania aplikacji. Autor [3] twierdzi z kolei, że Xamarin pozwala wykorzystać w pełni możliwości API platform docelowych oraz wszystkie zalety języka C#.

W celu przeanalizowania możliwości platformy Xamarin przy tworzeniu wieloplatformowych aplikacji mobilnych powstała aplikacja testowa i przeprowadzona została seria badań. Mają one wykazać prawdziwość hipotezy, która mówi, że dzięki zastosowaniu Xamarina, możliwe jest uzyskanie dużej części kodu wspólnego dla wszystkich platform docelowych i zachowanie przy tym zadowalającej wydajności aplikacji.

2. Xamarin

Xamarin jest platformą wykorzystywaną do tworzenia aplikacji wieloplatformowych przy pomocy języka C#. Platforma ta została stworzona przez firmę o tej samej nazwie, która została założona przez Miguela de Icaza 16 maja 2011 roku. Nazwa Xamarin wywodzi się od nazwy gatunku małpki - tamaryny, natomiast pierwsza litera jest nawiązaniem do poprzedniej platformy stworzonej przez firmę tego samego założyciela - Ximian. Xamarin został przejęty przez Microsoft 24 lutego 2016 roku. Od tego czasu Xamarin jest dalej rozwijany, a Xamarin SDK został udostępniony na zasadzie open-source.

Xamarin powstał w oparciu o architekturę Windows Runtime, co pozwala na budowanie aplikacji Universal Windows Platform bez użycia dodatkowych bibliotek natywnych [4]. Aplikacje pisane na pozostałe dwa systemy, czyli Android i iOS, wymagają bibliotek natywnych, które są

dostępne z poziomu Xamarina. Są to Xamarin.Android oraz Xamarin.iOS.

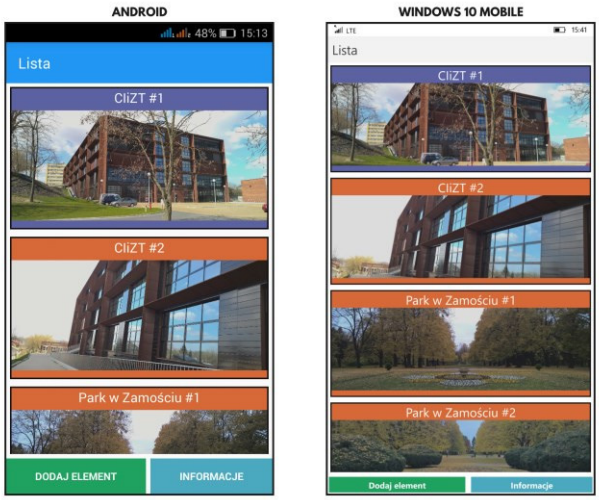
Jak już wcześniej wspomniano, aplikacje muszą być pisane przy pomocy jednego języka, którym jest C#. Aplikacje takie nie będą działały na Androidzie i iOS ponieważ platformy te wymagają innego języka. W celu zapewnienia działania aplikacji napisanych przy pomocy Xamarina, stworzone zostały dwa sposoby ich przystosowania do konkretnej platformy [5].

Pierwszym ze sposobów jest mechanizm zaprojektowany dla potrzeb Androida. Aplikacja w trakcie działania wprowadza komunikację między Javą a platformą .Net. Komunikacja ta zapewnia działanie aplikacji dla platformy .Net na platformie Java.

Drugim sposobem, przeznaczonym dla potrzeb iOS-a, jest prekompilowanie kodu do kodu wykorzystywanego przez system docelowy – Objecive-C. W przypadku tego systemu, aplikacja nie może nawiązywać komunikacji z systemem, jeżeli nie jest ona stworzona w natywnym języku.

3. Aplikacja testowa

W celu przeprowadzenia badań, w środowisku Xamarin stworzona została aplikacja na systemy Android i Windows 10 Mobile (UWP). Jej główną funkcją jest budowanie listy zdjęć (galerii) o możliwościach CRUD (Create, Read, Update, Delete) (Rys. 1). Możliwe jest zatem dodawanie nowych elementów, a także przeglądanie, edytowanie i usuwanie już istniejących.



Rys. 1. Strona główna aplikacji testowej

3.1. Interfejs

W tworzeniu aplikacji wykorzystano API Xamarin.Forms. Umożliwia on wykonanie jednego wspólnego interfejsu dla wszystkich platform docelowych. Stosuje się do tego języki XAML lub C# [6].

Xamarin.Forms posiada wbudowane kilka typów stron i układów, a także ponad 20 rodzajów kontrolek, które można wykorzystać do budowy interfejsu. Każdy z tych elementów musi mieć swojego odpowiednika na każdej z platform docelowych. W czasie działania aplikacji elementy pochodzące z Xamarin.Forms są mapowane właśnie na elementy natywne, odpowiednie dla danego systemu [7].

Zdarza się, że kontrolki natywne posiadają parametry, których nie da się zmodyfikować we wspólnej części kodu. Rozwiązaniem może być wówczas stworzenie niestandardowej kontrolki (Przykład 1), dziedziczącej po jednej z kontrolek Xamarin.Forms oraz rendererów dla każdej z platform docelowych (Przykład 2). To właśnie renderer określa sposób wyświetlania elementu na danej platformie [8].

Przykład 1. Przykładowy kod kontrolki niestandardowej

```
using Xamarin.Forms;
namespace Apka
{
public class CustomButton : Button
{
}
}
```

Przykład 2. Przykładowy kod renderera

```
[assembly: ExportRenderer(typeof(CustomButton),
typeof(CustomButtonRenderer))]
namespace Apka.Droid
{
public class CustomButtonRenderer : ButtonRenderer
{
protected override void
OnElementChanged(ElementChangedEventArgs<Button> e)
{
base.OnElementChanged(e);
if(Control != null)
{
Control.SetPadding(5, 2, 5, 2);
}
}
}
}
```

Xamarin.Forms pozwolił uczynić wspólną większość kodu odpowiedzialnego za interfejs aplikacji. Każda z platform wymaga jedynie wywołania metody inicjalizującej API Xamarin.Forms oraz utworzenia klas odpowiednich rendererów, w przypadku stosowania niestandardowych kontrolek.

3.2. Przechowywanie danych

W celu zachowania danych po zamknięciu aplikacji, zaimplementowano w niej funkcję zapisu listy elementów do pliku (Przykład 3). Najpierw dane są serializowane do formatu JSON, następnie wybierany jest folder i tworzony jest plik, w którym informacje te zostaną umieszczone [9].

Przykład 3. Metoda zapisująca listę elementów do pliku

```
public static async void saveJSON()
{
var d = JsonConvert.SerializeObject(Elements.AllElements);
IFolder rootFolder = FileSystem.Current.LocalStorage;
IFolder folder = await
rootFolder.CreateFolderAsync("ApkaData",
CreationCollisionOption.OpenIfExists);
IFile file = await folder.CreateFileAsync("elements.json",
CreationCollisionOption.ReplaceExisting);
await file.WriteAllTextAsync(d);
}
```

Skorzystano tu z frameworka Newtonsoft.Json oraz API PCLStorage [10]. Umożliwiły one zrealizowanie operacji zapisu i odczytu danych z całkowicie wspólnym kodem.

3.3. Przechwytywanie obrazów

Podczas dodawania elementu, zdjęcia przechwytywane są z aparatu urządzenia lub pobierane z pamięci wewnętrznej. Można do tego wykorzystać dowolną aplikację do obsługi aparatu lub przeglądania plików [11].

Do zrealizowania tej funkcji posłużyła wtyczka MediaPlugin. Kod jest w pełni współdzielony. Wymagane jest jedynie nadanie odpowiednich uprawnień.

3.4. Lokalizacja

Podczas dodawania nowego elementu do listy można także pobrać aktualną lokalizację z wykorzystaniem GPS-u lub połączenia internetowego i automatycznie uzupełnić jedno z pól formularza.

Najpierw, przy pomocy wtyczki GeolocatorPlugin, pobierane są współrzędnie geograficzne, a następnie na ich podstawie określany jest prawdopodobny adres miejsca, w którym znajduje się użytkownik (tzw. reverse geocoding) [12].

Wykorzystywane jest do tego API Xamarin.Forms.Maps, które w celu ustalenia adresu korzysta z usług właściwych platformie docelowej (Android – Google Maps, Windows 10 Mobile – Bing Maps) [13].

Na wypadek gdyby Xamarin.Forms.Maps nie udało się określić adresu, stworzono alternatywne rozwiązanie. Jest nim klasa CustomGeocoder.cs (Przykład 4), która reverse geocoding wykonuje z pomocą API Google Maps i zapytań HTTP [14].

Przykład 4. Klasa wykonująca reverse geocoding z wykorzystaniem API Google Maps i zapytań HTTP

```
public class CustomGeocoder
{
private string key = "<klucz_do_API_Google_Maps>";
public async Task<IEnumerable<string>>
ReverseGeocoding(double latitude, double longitude)
{
HttpClient client = new HttpClient();
var request =
string.Format(@"https://maps.googleapis.com/maps/api/geo
code/json?latlng={0},{1}&key={2}&language=pl", latitude,
longitude, key);
var response = await client.GetStringAsync(request);
var results = JObject.Parse(response)["results"];
List<string> list = new List<string>();
foreach (JToken i in results)
{
list.Add((string)i["formatted_address"]);
}
return list.AsEnumerable();
}
}
```

Poza wywołaniem w kodzie każdej z platform metody inicjalizującej Xamarin.Forms.Maps oraz, w przypadku Androida, dodaniem w manifeście informacji o kluczu do API Google Maps, reszta kodu jest wspólna. Niezbędne jest jednak przypisanie odpowiednich uprawnień, by aplikacja mogła korzystać z usług lokalizacyjnych.

3.5. Dane o urządzeniu

Jedną z pobocznych funkcji aplikacji jest wyświetlanie informacji o urządzeniu oraz odczytów z sensorów. Podane są tu: typ urządzenia, model, system operacyjny oraz jego wersja, stan baterii, jej status, a także dostępność połączenia internetowego. Sensory, z których można uzyskać odczyty, to z kolei: akcelerometr, żyroskop, magnetometr, kompas, czujnik orientacji, czujnik oświetlenia.

W realizacji tej funkcji pomogły następujące wtyczki: Plugin.DeviceInfo, Plugin.Battery, Plugin.Connectivity oraz Plugin.Sensors. Całość zrealizowana została w pełni wieloplatformowo.

3.6. Filtry graficzne

Do jednej ze stron aplikacji, wyświetlającej szczegółowe informacje o danym elemencie, dodano możliwość wyświetlania zdjęcia z zastosowaniem jednego z dostępnych filtrów. Są to Grayscale, Sepia, Invert Color oraz Black & White (Rys. 2). Nakładane są one na obrazy załadowane do pamięci podręcznej.

Rys. 2. Tryby wyświetlania obrazu

Do tego celu wykorzystano bibliotekę FFImageLoading, która służy do wczytywania i wyświetlania obrazów z różnych źródeł i ma do tego przeznaczone własne kontrolki, mogące zastąpić te, które pochodzą z Xamarin.Forms [15]. Skorzystanie z niej wymaga dodania wywołania metody inicjalizującej do części kodu każdej z platform docelowych. Pozostała część kodu jest wspólna dla każdego systemu.

4. Metody i przebieg badań

Zbadane zostały: część wspólna kodu źródłowego, rozmiar aplikacji po zainstalowaniu oraz jej wydajność.

W celu zbadania części wspólnej kodu aplikacji wykorzystano metrykę linii kodu źródłowego, będącą najprostszą metryką rozmiaru oprogramowania [16]. Pomogła w tym aplikacja LocMetrics, zliczająca liczbę linii kodu w wybranych katalogach. Wśród informacji, które zwraca, są: liczba linii fizycznych kodu źródłowego oraz liczba linii logicznych kodu źródłowego. Ta pierwsza oznacza całkowitą liczbę linii kodu z wyłączeniem linii pustych oraz linii składających się jedynie z komentarza. Druga natomiast dotyczy wyłącznie linii wpływających na logikę aplikacji, zawierających instrukcje. Pomijane są w niej zatem linie fizyczne kodu źródłowego, które zawierają jedynie nawias klamrowy a instrukcje zawarte w kilku liniach są liczone jako jedna. Dla języków, których aplikacja nie wspiera, takich jak XAML, linie logiczne kodu nie są zliczane. W związku z tym, że definiujące interfejs graficzny, pliki XAML stanowią znaczną część kodu współdzielonego, w tym konkretnym przypadku należy mieć ograniczone zaufanie do tej wartości.

Badanie rozmiaru aplikacji polegało na jej zainstalowaniu i odczytaniu rozmiaru z jednej ze stron w ustawieniach urządzenia.

Dla sprawdzenia wydajności przeprowadzone zostały pomiary czasu trwania najważniejszych funkcji aplikacji:

1) zapisu – składającego się z serializacji listy obiektów do formatu JSON i zapisania tych danych w pliku tekstowym,
2) odczytu – czyli wczytania danych z pliku tekstowego, deserializacji ich oraz dodania uzyskanych elementów do listy,
3) wczytywania i wyświetlania obrazu,
4) dodawania, modyfikowania i usuwania elementów listy,
5) pobierania współrzędnych geograficznych za pomocą GPS-u.

Czas trwania zapisu i odczytu został rozpatrzony dla listy składającej się z 1, 10, 100 oraz 1000 elementów. Wczytywanie i wyświetlanie obrazu zostało przetestowane dla plików o rozmiarach 0,5 MB, 1 MB, 2 MB, 5 MB z wyróżnieniem dwóch przypadków – bez użycia filtru oraz z zastosowaniem filtru Invert Color. Dla każdego przypadku przeprowadzono 10 pomiarów. Na ich podstawie wyznaczona została średnia wartość oraz odchylenie standardowe.

Fragment kodu wykorzystany do ustalenia czasu trwania poszczególnych operacji przedstawia przykład 5.

Przykład 5. Fragment kodu służący do pomiaru czasu wykonania badanych funkcji

```
DateTime startTime = DateTime.Now;
<fragment_kodu_którego_czas_wykonania_badamy>
DateTime stopTime = DateTime.Now;
TimeSpan roznica = stopTime - startTime;
await DisplayAlert("Wykonano", "Operacja trwała " +
roznica.TotalMilliseconds + " milisekund.", "OK");
```

Urządzenia, na których zostały przeprowadzone pomiary to:

- myPhone Luna (Android),
- Nokia Lumia 930 (Windows 10 Mobile).

5. Rezultaty badania

5.1. Część wspólna kodu

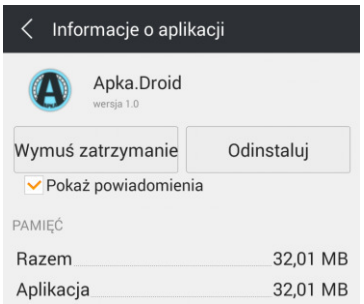
Pierwsze badanie dotyczyło części wspólnej kodu źródłowego aplikacji. Wyniki zostały zebrane w tabeli 1.

Tabela 1. Wyniki pomiarów liczby linii kodu

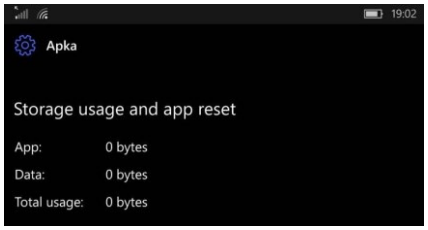
		Wspólne	Android	UWP	Suma
Pliki źródłowe		24	4	6	34
Linie kodu		1365	143	156	1664
Linie fizyczne kodu	liczba	1248	118	101	1467
	udział	85%	8%	7%	100%
Linie logiczne kodu	liczba	546	67	53	666
	udział	82%	10%	8%	100%

5.2. Rozmiar aplikacji

Zbadano również rozmiary aplikacji. Po zainstalowaniu na urządzeniu z systemem Android, aplikacja zajmuje 32 MB (Rys. 3). Niestety dla systemu Windows 10 Mobile nie udało się uzyskać takich informacji (Rys. 4).



Rys. 3. Informacje o aplikacji zainstalowanej na urządzeniu z systemem Android



Rys. 4. Informacje o aplikacji zainstalowanej na urządzeniu z systemem Windows 10 Mobile

5.3. Wydajność aplikacji

Badanie wydajności aplikacji składa się z pięciu części. Pierwsza z nich dotyczy czasu trwania operacji zapisu. Jej wyniki znajdują się w tabelach 2 i 3.

W drugiej mierzony jest czas trwania operacji zapisu. Wyniki zestawione są w tabelach 4 i 5.

Trzecia część, której wyniki zawarto w tabelach od 6 do 9, bada czas wczytywania obrazu, z wyróżnieniem dwóch przypadków – z zastosowaniem filtru oraz bez.

Czwarta bada czas wykonywania podstawowych operacji na elementach listy, czyli dodawania, modyfikowania oraz usuwania. Wyniki tych pomiarów przedstawiono w tabelach 10 i 11.

Piąta i ostatnia część badania wydajności mierzy czas potrzebny do uzyskania współrzędnych geograficznych z wykorzystaniem GPS-u. Jej rezultaty przedstawia tabela 12.

Tabela 2. Wyniki pomiarów czasu trwania operacji zapisu na platformie Android

		1 el. [ms]	**10 el. [ms]**	**100 el. [ms]**	**1000 el. [ms]**
Pomiar	1.	0,432	0, 723	2,974	21,851
	2.	0,465	0,666	3,167	22,420
	3.	0,425	1,583	2,877	22,531
	4.	0,412	0,651	2,858	22,378
	5.	0,428	0,661	2,916	23,146
	6.	0,433	0,872	2,825	22,265
	7.	0,426	2,628	2,704	22,424
	8.	0,415	0,649	5,501	26,303
	9.	0,431	0,658	2,732	23,047
	10.	0,440	0,661	3,373	22,868
Średnia		0,4307	1,003222222	3,1927	22,9233
Odchylenie standardowe		0,0146367	0,680985275	0,835582	1,248438

Tabela 3. Wyniki pomiarów czasu trwania operacji zapisu na platformie Windows 10 Mobile

		1 el. [ms]	**10 el. [ms]**	**100 el. [ms]**	**1000 el. [ms]**
Pomiar	1.	3,9965	15,6255	35,4962	312,5101
	2.	2,9987	15,6334	34,9970	315,4994
	3.	3,4945	15,6218	37,4948	316,5297
	4.	2,9961	15,6227	36,0075	312,5032
	5.	3,9967	15,6240	36,9962	311,4941
	6.	3,0008	15,6264	36,0194	321,0005
	7.	2,9962	9,9963	35,9946	312,0176
	8.	3,0019	15,4961	35,9967	312,0266
	9.	2,9967	15,6328	31,9961	311,0093
	10.	3,0044	5,4990	35,4960	312,5258
Średnia		3,24825	14,0378	35,64945	313,7116
Odchylenie standardowe		0,423575	3,4807864	1,473602	3,10026

Tabela 4. Wyniki pomiarów czasu trwania operacji odczytu na platformie Android

		1 el. [ms]	**10 el. [ms]**	**100 el. [ms]**	**1000 el. [ms]**
Pomiar	1.	1654,906	1688,741	1669,456	1751,026
	2.	1647,045	1632,476	1654,561	1764,562
	3.	1668,891	1659,274	1672,452	1773,378
	4.	1654,799	1659,415	1699,332	1726,952
	5.	1673,361	1648,650	1626,493	1725,662
	6.	1645,767	1665,450	1657,700	1747,956
	7.	1661,355	1639,579	1650,282	1759,079
	8.	1645,569	1637,717	1649,915	1744,079
	9.	1649,144	1681,694	1655,540	1759,636
	10.	1669,651	1660,462	1695,999	1743,724
Średnia		1657,0488	1657,346	1663,173	1749,605
Odchylenie standardowe		10,6195012	18,42991	22,01099	15,40394

Tabela 5. Wyniki pomiarów czasu trwania operacji odczytu na platformie Windows 10 Mobile

		1 el. [ms]	10 el. [ms]	100 el. [ms]	1000 el. [ms]
Pomiar	1.	68,9995	77,8771	105,5278	492,1302
	2.	65,0024	62,7341	107,7692	517,0285
	3.	64,5440	62,7782	107,5160	490,4102
	4.	64,0228	78,1275	110,0255	499,5718
	5.	64,0035	62,1838	106,5323	498,7774
	6.	66,5266	62,4927	107,7483	499,3605
	7.	64,5081	78,1222	107,0353	500,3359
	8.	69,5097	78,1291	107,5269	499,7116
	9.	65,5249	62,2166	108,0259	495,0052
	10.	65,5310	62,5118	116,4986	501,0458
Średnia		65,81725	68,71731	108,4206	499,3377
Odchylenie standardowe		1,970343	8,046801	3,06108	7,20961

Tabela 6. Wyniki pomiarów czasu trwania wczytywania obrazu bez zastosowania filtru na platformie Android

		0,5 MB [ms]	1 MB [ms]	2 MB [ms]	5 MB [ms]
Pomiar	1.	311,595	595,477	1077,081	1963,799
	2.	347,015	582,990	1047,758	1952,180
	3.	329,200	565,402	1071,214	1931,721
	4.	331,272	550,589	1004,119	1994,243
	5.	324,357	581,034	1031,175	1872,798
	6.	302,834	600,055	1062,640	1917,439
	7.	351,377	553,275	1016,760	1954,589
	8.	352,768	637,666	1013,993	1900,748
	9.	313,108	595,436	1056,865	1980,196
	10.	302,274	611,104	1016,176	1969,109
Średnia		326,580	587,3028	1039,778	1943,6822
Odchylenie standardowe		19,185677	26,7537568	26,57354	37,759064

Tabela 7. Wyniki pomiarów czasu trwania wczytywania obrazu bez zastosowania filtru na platformie Windows 10 Mobile

		0,5 MB [ms]	1 MB [ms]	2 MB [ms]	5 MB [ms]
Pomiar	1.	286,7113	382,8265	455,2751	721,0278
	2.	282,2281	356,5726	433,3450	714,3192
	3.	272,1987	316,7453	498,5671	697,8735
	4.	275,5580	321,0543	514,9757	734,9586
	5.	285,0366	358,8566	484,0400	666,5381
	6.	288,0256	334,5981	501,2007	655,8544
	7.	260,8198	338,8908	468,3557	673,7759
	8.	277,4814	354,3209	454,1504	690,5607
	9.	249,8218	349,9877	486,7857	675,2426
	10.	301,6631	305,8886	458,1208	642,8750
Średnia		277,9544	341,97414	475,48162	687,30258
Odchylenie standardowe		14,65442	23,151483	25,7039056	29,782769

Tabela 8. Wyniki pomiarów czasu trwania wczytywania obrazu z zastosowaniem filtru Invert Color na platformie Android

		0,5 MB [ms]	1 MB [ms]	2 MB [ms]	5 MB [ms]
Pomiar	1.	450,955	742,429	1311,640	2939,336
	2.	463,117	725,451	1391,905	2980,945
	3.	472,645	772,903	1361,115	2860,552
	4.	439,756	788,207	1308,342	2916,138
	5.	399,070	718,667	1364,266	2840,092
	6.	446,607	769,982	1324,268	2828,857
	7.	461,681	730,500	1330,405	2994,383
	8.	435,950	793,799	1323,011	2918,065
	9.	474,052	707,653	1396,121	2864,818
	10.	442,487	787,835	1387,202	2961,229
Średnia		448,632	753,7426	1349,828	2910,4415
Odchylenie standardowe		21,975134	32,332681	34,27878	59,345783

Tabela 9. Wyniki pomiarów czasu trwania wczytywania obrazu z zastosowaniem filtru Invert Color na platformie Windows 10 Mobile

		0,5 MB [ms]	**1 MB [ms]**	**2 MB [ms]**	**5 MB [ms]**
Pomiar	1.	875,9948	985,2309	1282,3005	2145,4621
	2.	939,9395	1029,6358	1201,2254	2028,7621
	3.	872,5265	1060,9050	1276,1367	1977,8877
	4.	924,6700	1031,2712	1216,4592	2039,8187
	5.	915,2091	981,3049	1241,5329	2134,4103
	6.	851,2652	972,7751	1208,9054	2076,4461
	7.	907,1745	959,2436	1265,7399	2068,1005
	8.	911,0692	1002,1670	1203,6973	2096,6950
	9.	859,9536	1038,2657	1196,6281	2125,6613
	10.	884,5239	988,7826	1199,5672	2079,6493
Średnia		894,2326	1004,9582	1229,21926	2077,2893
Odchylenie standardowe		29,48787	33,140198	34,085145	52,022536

Tabela 10. Wyniki pomiarów czasu trwania operacji dodawania, modyfikowania oraz usuwania elementu listy na platformie Android

		Dodawanie [ms]	**Modyfikowanie [ms]**	**Usuwanie [ms]**
Pomiar	1.	115,291	91,112	2,992
	2.	98,114	100,127	3,224
	3.	97,016	89,197	3,206
	4.	103,978	85,640	3,192
	5.	98,504	92,716	3,119
	6.	87,124	118,434	3,137
	7.	81,205	96,558	3,112
	8.	102,348	107,599	3,247
	9.	79,643	87,968	3,347
	10.	96,291	111,527	3,164
Średnia		95,9514	98,0878	3,174
Odchylenie standardowe		10,817411	11,08283921	0,094615

Tabela 11. Wyniki pomiarów czasu trwania operacji dodawania, modyfikowania oraz usuwania elementu listy na platformie Windows 10 Mobile

		Dodawanie [ms]	**Modyfikowanie [ms]**	**Usuwanie [ms]**
Pomiar	1.	213,2765	200,8310	15,6274
	2.	222,6895	203,1583	20,9389
	3.	216,1872	267,1688	1,0011
	4.	238,1435	224,6395	4,0031
	5.	200,4493	215,9724	6,0340
	6.	248,7288	249,7972	4,0022
	7.	247,1553	231,7920	1,0039
	8.	247,4104	233,3013	4,0022
	9.	259,4967	235,9488	15,6515
	10.	232,0573	232,0764	4,0027
Średnia		232,55945	229,4686	7,6267
Odchylenie standardowe		18,916328	20,05894	7,0589414

Tabela 12. Wyniki pomiarów czasu potrzebnego do uzyskania, za pomocą GPS-u, współrzędnych geograficznych na platformach Android oraz Windows 10 Mobile

		Android [ms]	**Windows 10 Mobile [ms]**
Pomiar	1.	24763,093	26981,7056
	2.	4996,948	20502,6369
	3.	14102,487	8094,5966
	4.	8092,174	41586,0219
	5.	10246,351	16941,7255
	6.	8127,505	10565,7142
	7.	7136,173	7482,8751
	8.	10184,288	42251,9974
	9.	9116,895	12898,4021
	10.	16089,681	4605,6059
Średnia		11285,56	19191,12812
Odchylenie standardowe		5740,5913	13694,10964

6. Wnioski

Tworzenie jednej aplikacji na wiele platform z wykorzystaniem rozwiązań wieloplatformowych jest efektywne pod wieloma względami.

Przede wszystkim nie jest wymagana znajomość różnych języków programowania. Wymagana jest jedynie znajomość języka C#, ponieważ Xamarin jest platformą opartą o platformę .Net oraz architekturę Windows Runtime.

Tworząc aplikację udało się uzyskać część wspólną kodu na poziomie 85% linii fizycznych i 82% linii logicznych. Pozwoliło to znacząco skrócić czas programowania w porównaniu do pisania oddzielnych aplikacji na każdą z platform.

Aplikacja ma zaimplementowane wiele funkcji i korzysta z dużej liczby bibliotek, więc jej rozmiar wynoszący 32 MB na platformie Android nie jest zbytnio wygórowany.

Operacja zapisu wykonywana jest szybciej na urządzeniu z systemem Android, natomiast w przypadku odczytu to Windows 10 Mobile jest szybszy.

Na czas wczytywania obrazu wpływ ma waga pliku oraz zastosowanie filtru. Zwiększenie rozmiaru pliku powoduje wydłużenie czasu wczytywania. Podobny efekt daje użycie filtru. W większości przypadków lepsze wyniki osiągnął Windows 10 Mobile. Może to świadczyć o tym, że platforma UWP, która powstała w oparciu o architekturę Windows Runtime, radzi sobie lepiej z aplikacją stworzoną w oparciu o tę samą architekturę, niż platforma, na której aplikacja uruchamiana jest w połączeniu z Javą, tak jak w przypadku Androida.

Średnie czasy dodawania i modyfikowania elementów listy są do siebie podobne i ok. 30 razy wyższe od średniego czasu usuwania elementu. Sytuacja taka ma miejsce w przypadku obu testowanych platform. Dodatkowo, średnie wartości dla systemu Android są ok. 2,3-2,4 razy niższe niż w przypadku Windows 10 Mobile.

W badaniu czasu uzyskiwania współrzędnych z pomocą GPS-u otrzymano zróżnicowane wyniki. Wystąpiły w nim wartości w zakresie od kilku do kilkudziesięciu sekund. Choć średnia wskazuje na korzyść Androida, należy pamiętać, że na pracę systemu nawigacji wpływać może wiele czynników, wewnętrznych oraz zewnętrznych.

Pomimo dłuższych czasów wykonywania niektórych operacji, można uznać, że poziom wydajności aplikacji stoi na zadowalającym poziomie. Ponadto duża część kodu jest wspólna dla wszystkich platform docelowych. W związku z tym, należy stwierdzić, że postawiona na początku hipoteza badawcza została potwierdzona.

Literatura

[1] Martinez M., Lecomte S., Towards the quality improvement of cross-platform mobile applications, 2017.

[2] Gerasimov V., Bilovol S., Ivanova K., Comparative Analysis Between Xamarin and Phonegap for .Net, System technologies, 2015, vol. 96.

[3] Radi A., Evaluation of Xamarin Forms for Multi-Platform Mobile Application Development, Technical Library, 2016, paper 249.

[4] Bilgin C., Mastering Cross-Platform Development with Xamarin, Packt Publishing, 2016.

[5] Peppers J., Xamarin 4.x Cross-Platform Application Development. Third Edition, Packt Publishing, 2016.

[6] Build a Native Android UI & iOS UI with Xamarin.Forms, https://www.xamarin.com/forms [22.11.2017].

[7] Tunalı V., Erdogan S., Comparison of Popular Cross-Platform Mobile Application Development Tools, 2. Ulusal Yönetim Bilişim Sistemleri Kongresi (YBS2015), Erzurum, 2015.

[8] Johnson P., Cross-platform UI Development with Xamarin.Forms, Packt Publishing, 2015.

[9] Serializing and Deserializing JSON, https://www.newtonsoft.com/json/help/html/SerializingJSON.htm [22.11.2017].

[10] Daniel Plaisted, PCL Storage, https://github.com/dsplaisted/PCLStorage [dostęp 22.11.2017].

[11] James Montemagno, Take & Pick Photos and Video Plugin for Xamarin and Windows, https://github.com/jamesmontemagno/MediaPlugin [22.11.2017].

[12] James Montemagno, Checking Current Location, https://jamesmontemagno.github.io/GeolocatorPlugin/CurrentLocation.html [dostęp 22.11.2017].

[13] Map – Xamarin, https://developer.xamarin.com/guides/xamarin-forms/user-interface/map/ [22.11.2017].

[14] Google Maps Geocoding API, https://developers.google.com/maps/documentation/geocoding/intro [22.11.2017].

[15] Daniel Luberda, FFImageLoading - Xamarin.Forms API, https://github.com/luberda-molinet/FFImageLoading/wiki/Xamarin.Forms-API [22.11.2017].

[16] Sencer Sultanoğlu, Software Size Estimating, http://yunus.hun.edu.tr/~sencer/size.html [22.11.2017].

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 191-194
Wysłane: 2017-12-10
Przyjęte: 2017-12-18

Analiza porównawcza wybranych programów do optycznego rozpoznawania tekstu

Edyta Łukasik[a,*], Tomasz Zientarski[a]

[a] Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Celem artykułu jest porównanie trzech programów do optycznego rozpoznawania teksu. Zdefiniowany został problem optycznego rozpoznawania tekstu i przedstawione główne jego zastosowania. Opisano działanie tej technologii i krótko scharakteryzowano najważniejsze dostępne na rynku programy realizujące omawiane zagadnienie. Następnie poddano testom wybrane programy wykorzystując dwie próbki pisma maszynowego w języku polskim. Określono szybkość procesu rozpoznawania tekstu. Poprawność rozpoznania znaków i wyrazów w analizowanym tekście została także określona.

Słowa kluczowe: rozpoznawanie tekstu, OCR; Tesseract; Ocrad; GOCR

*Autor do korespondencji.

Adres e-mail: e.lukasik@pollub.pl

Comparative analysis of selected programs for optical text recognition

Edyta Łukasik[a,*], Tomasz Zientarski[a]

[a] Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The aim of the article is to compare three programs for the optical text recognition. The problem of the optical text recognition has been defined. Next, briefly the functionality of this technology was described. The most important programs realizing the discussed problem were also characterized. The selected programs were tested using two samples of machine writing in Polish. The speed of the text recognition process was determined. The correctness of characters and words recognition in the analyzed text was also specified.

Keywords: Optical Character Recognition; OCR; Tesseract; Ocrad; GOCR

*Corresponding author.

E-mail address: e.lukasik@pollub.pl

1. Wstęp

Optyczne rozpoznawanie znaków (ang. Optical Character Recognition – OCR) jest technologią, która umożliwia konwertowanie różnego typu dokumentów do zbiorów edytowalnych i umożliwiających ich przeszukiwanie. Obejmuje ona przetwarzanie papierowych dokumentów, które są zeskanowane, plików w formacie pdf, zdjęć a także zapis komputerowy pisma odręcznego. Oprogramowanie OCR potrafi rozpoznać litery na obrazie, zbudować z nich słowa, a także zdania. Zastosowane w tej technologii algorytmy powinny gwarantować inteligencję programu, która to możliwie jak najbardziej będzie zbliżona do ludzkiego zmysłu wzroku.

Zastosowanie tej technologii pozwala na zaoszczędzenie czasu, który trzeba byłoby poświęcić na przepisanie analizowanego tekstu. Przydaje się ona w zastąpieniu pracy człowieka przy przepisywaniu dużej ilości tekstu utrwalonego na papierze, przy przenoszeniu zbiorów bibliotek do zasobów cyfrowych, a także do pomocy niepełnosprawnym. Kolejnym zastosowaniem tej technologii jest wdrożenie systemu obiegu dokumentów ze zintegrowanym modułem OCR w procesach biznesowych. Pozwala to znacznie przyspieszyć i zoptymalizować działanie firmy. Przykładem może być tutaj aplikacja służąca do przenoszenia zawartości faktur formularza elektronicznego [1].

2. Proces OCR

Klasyczny algorytm służący do optycznego rozpoznawania znaków składa się z czterech kroków [2]:

- przetwarzanie wstępne;
- analiza rozkładu;
- rozpoznawanie znaków;
- wyjście.

Pierwszy z nich obejmuje przetwarzanie wstępne (ang. preprocessing) czyli jest to odpowiednie przygotowanie obrazka do efektywniejszego działania algorytmu. Obejmuje ono binaryzację obrazka, usunięcie szumu i jego obrócenie w taki sposób, aby wyrównać w poziomie tekst. Czynności te mogą zostać zaimplementowane w programie lub wykonane przed skanowaniem. Drugim krokiem jest analiza rozkładu

tekstu (ang. layout analysis) umożliwiająca identyfikacje kolumn, tabel czy akapitów w tekście. Kolejnym krokiem procesu jest rozpoznawanie znaków (ang. character recognition) w każdej linii tekstu. Najpierw dzielona jest ona na wyrazy, a następnie w każdym słowie rozpoznawane są pojedyncze znaki. W tym miejscu algorytm korzysta z bazy danej zawierającej znaki i wybiera najlepszą wartość. Dla podniesienia skuteczności stosuje się także słowniki słów i funkcje kary dla znaków, które powodują, iż słowo nie znajduje się w słowniku [3].

Ważną sprawą jest czas trwania takiego procesu OCR i poziom błędów otrzymany na wyjściu. Jeżeli obraz jest dokładny, nie posiada szumu, a znaki są oddzielone od siebie, wtedy skuteczność algorytmu jest wysoka. Dokładność metody zależy również od rozdzielczości obrazu.

Oprogramowanie OCR z mniejszym błędem rozpoznaje tekst pisany na maszynie niż ręcznie. Powodem jest fakt, iż w pisaniu maszynowym ten sam znak zawsze wygląda tak samo, przy użyciu tej samej czcionki. Dodatkowo pismo ręczne jest o wiele bardziej zróżnicowane niż maszynowe, a skutkuje to tym, iż o wiele trudniej jest dopasować pojedyncze znaki do tych zgromadzonych w bazie danych.

Dokładność otrzymanych wyników może być mierzona na dwa sposoby:

1. na poziomie znaków (ang. character level occuracy) według wzoru

$$A_C = \frac{100 \cdot Ec}{C},$$

gdzie: *Ec* jest to liczbą błędnie rozpoznanych znaków, natomiast *C* liczba wszystkich znaków;

2. na poziomie słów (ang. word level occuracy) według wzoru

$$A_W = \frac{100 \cdot Ew}{W},$$

gdzie *Ew* jest to liczbą błędnie rozpoznanych słów, natomiast *W* liczbą wszystkich słów.

3. Biblioteki realizujące OCR – przegląd

3.1. Oprogramowanie ABBYY

Program ABBYY FineReader jest jednym z programów realizujących technologię OCR. Jego działanie opiera się o trzy podstawowe zasady: integralność, celowość i przystosowalność (ang. integrity, purposefulness, adaptability-IPA). Według zasady integralności obserwowany obiekt musi być zawsze traktowany jako „całość" składająca się z wielu powiązanych części. Zasada celowości mówi o tym, że każda interpretacja danych musi służyć jakiemuś celowi. Zasada przystosowalności oznacza, że program musi być zdolny do samodzielnej nauki. Działanie programu polega na podziale strony dokumentu na bloki tekstu, tabele, obrazy itp. Następnie wyodrębnione linie są dzielone na pojedyncze znaki, które są porównywane ze zbiorem wzorcowym. W trakcie działania algorytmu stawiane są różne hipotezy i analizowane różne warianty. Po przetworzeniu odpowiedniej liczby hipotez podejmowana jest decyzja o kształcie tekstu [4].

Program ten zawiera słowniki w 48 językach. Wynikiem działania omawianego programu jest dokument, który może być zapisany w jednym z formatów: DOC, RTF, XLS, PDF, HTML, TXT itd. lub wyeksportowany bezpośrednio do aplikacji biurowej np. Microsoft Word. Jego interfejs jest bardzo intuicyjny w obsłudze.

3.2. Silnik Tesseract

Tesseract to silnik OCR z obsługą Unicode i możliwością rozpoznawania ponad 100 języków. Moduł Tesseract-polish służy opracowaniu metody do rozpoznawania tekstów języku polskim dla programu Tesseract OCR [5]. Jest on zaopatrzony w silnik stworzony przez Google. Nie posiada on GUI.

3.3. Komponent Aspose.OCR

Aspose.OCR jest komponentem optycznego rozpoznawania tekstu, który umożliwia programistom .NET dodanie funkcjonalności OCR i OMR do tworzonych przez nich aplikacji w technologii .NET i Java. Narzędzie to dostarcza klas pozwalających programistom na optyczne rozpoznawanie tekstu i optycznych znaków z obrazów. Komponent ten jest zaimplementowany przy użyciu Managed C# i może być używany z dowolnym językiem platformy .NET [6].

3.4. Program GOCR

GOCR to program OCR opracowany na licencji publicznej GNU. Umożliwia on konwertowanie zeskanowanych obrazów tekstu na pliki tekstowe. Obecnie jest to jOCR umieszczony w sourceforge. GOCR może być używany z różnymi front-endami, co sprawia, że bardzo łatwo można go podłączyć do różnych systemów operacyjnych i architektur. Może otwierać wiele różnych formatów obrazu, a jego jakość poprawia się z dnia na dzień [7].

3.5. Aplikacja Ocrad

Ocrad jest to program OCR dostępny na licencji GNU oparty na metodzie ekstrakcji cech. Przetwarza on obrazy w formatach pbm (bitmapa), pgm (skala szarości) lub ppm (kolor) i tworzy tekst, który zapisywany jest w formatach bajtowych (8-bitowych) lub UTF-8. Zawiera on także również analizator układu tekstu, który może oddzielać kolumny lub bloki tekstu zwykle znajdowane na drukowanych stronach. Może on być używany jako samodzielna aplikacja konsolowa lub jako backend do innych programów [8].

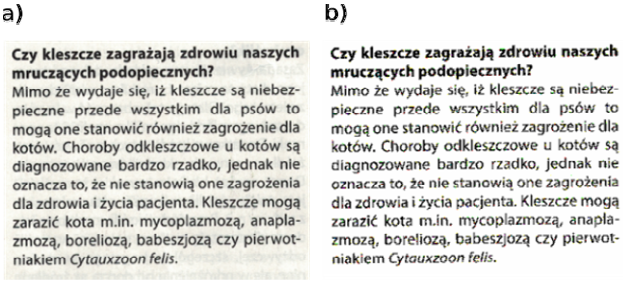
4. Cel pracy

Celem niniejszej pracy jest wskazanie aplikacji, która w zeskanowanych danych tekstowych rozpozna jak najwięcej poprawnego tekstu. Określony zostanie także wpływ palety kolorów w zeskanowanych obrazach wejściowych na poprawność i szybkość procesu rozpoznawania tekstu.

5. Analiza efektywności rozpoznawania tekstu przez wybrane programy OCR

W niniejszej pracy porównane zostały trzy aplikacje służące do optycznego rozpoznawania tekstu: Tesseract, GOCR, Ocrad.

Jako dane testowe użyto zeskanowanego z rozdzielczością 300dpi fragmentu tekstu. Wykonano dwie próbki, których fragmenty pokazano na Rysunku 1. Pierwsza próbka (T1) zawierała zeskanowany w pełnej palecie kolorów tekst, druga (T2) powstała z pierwszej przez redukcję palety kolorów do odcieni szarości. Tak otrzymane próbki zostały poddane procesowi OCR. W trakcie testów mierzono czas trwania procesu OCR. Ponadto obliczano dwie wielkości charakteryzujące jakość otrzymanego tekstu. Procentowy błąd poprawności rozpoznania wyrazów (A_W) oraz znaków (A_C) w stosunku do oryginału.



Rys. 1. Zeskanowany tekst do pliku w przestrzeni kolorów RGB (a) oraz w odcieniach szarości (b)

Wymienione we wstępie programy zostały zainstalowane ze standardowymi parametrami na komputerze klasy PC (i5M560, 8MB RAM, linux 64bit).

W pierwszej kolejności mierzono czas potrzebny do rozpoznania tekstu w analizowanym obrazie, przy czym czas mierzono od momentu uruchomienia skryptu do jego zakończenia. Wyniki przedstawiono w Tabeli 1.

Tabela 1. Czas trwania procesu rozpoznawania tekstu [s]

Próbka	Tesseract ver. 3.03	Ocrad ver. 0.22	GOCR ver 0.49
T1	9,11	0,137	3,11
T2	8,02	0,10	2,15

Czas potrzebny do rozpoznania tekstu w zeskanowanym obrazie nie był zbyt długi. Widać jednak wyraźnie dwie zależności. W każdym przypadku czas potrzebny do analizy pliku ze zubożoną paletą kolorów był zawsze nieznacznie krótszy od tego z pełną paletą. Ponadto, najkrótszy czas osiągnął program o nazwie Ocrad, ponad 60 razy szybciej od najwolniejszego programu.

Po testach szybkości przyszedł czas na testy jakościowe. Najprostszą i zarazem najczytelniejsza metodą oceny jest policzenie liczby błędnych wyrazów i znaków w stosunku do wzorcowego tekstu. Przeprowadzono obliczenia błędu poprawnego rozpoznania wyrazów i znaków w wynikowym tekście. Otrzymane wyniki przedstawiono w Tabeli 2.

Tabela 2. Błąd poprawnego rozpoznania wyrazów/znaków w analizowanym tekście

Nazwa aplikacji	Próbka T1		Próbka T2	
	Aw	Ac	Aw	Ac
Tesseract	6,78%	1,32%	6,76%	1,38%
Ocrad	46,43%	7,38%	48,23%	7,67%
GOCR	13,11%	2,61%	12,56%	2,11%

Testowe próbki zawierały 244 wyrazy składające się z 1896 znaków. Najprecyzyjniejszym programem okazał się produkt o nazwie Tesseract dający plik wyjściowy z najmniejszą liczbą niepoprawnych rozpoznań, zarówno na poziomie wyrazów jak i znaków. Najgorszy wynik osiągnął Ocrad.

6. Wnioski

Przeprowadzone badania pokazały, że niewątpliwym liderem wśród programów do OCR jest Tesseract. Stworzony w latach 90 dwudziestego wieku przez firmę HP, a obecnie udostępniany przez Google [5]. Nie jest on zbyt szybki, ale daje bardzo dobre jakościowo pliki wynikowe.

Ponadto, testowane programy w obecnych wersjach nie są wrażliwe na użytą paletę kolorów pliku graficznego zawierającego analizowany obraz. Dotyczy to szczególnie programu Tesseract, który w wersji 2.0 dawał procent niepoprawnych rozpoznań na poziomie 70-90% gdy obraz był zeskanowany z pełna paletą barw [9]. W obecnej wersji programu zostało to poprawione, jak pokazują otrzymane wyniki.

Zastanawiające rezultaty otrzymano dla programu Ocrad. Rozpoznawanie tekstu przebiegało błyskawicznie, ale otrzymane wyniki nie nadawały się do dalszej obróbki. W pracy [9] pokazano, że daje on wyniki lepsze od programu GOCR. Z kolei autor w tekście [10] stwierdza, że Ocrad daje dobrą rozpoznawalność tekstu ale tylko, gdy w oryginalnym skanowanym dokumencie była użyta standardowa czcionka, czyli Times New Roman. Okazuje się, że program ten stosuje zupełnie inne algorytmy rozpoznawania liter w porównaniu do pozostałych omawianych. Oparte są one na binarnym porównywaniu kształtu znaków [8]. Daje mu to bardzo dużą szybkość działania i bardzo dużą wrażliwość na kształt użytej czcionki. W rozważanym przypadku na niską dokładność wyniku wpłynął fakt analizy tekstu z polskimi znakami diakrytycznymi.

Literatura

[1] Bieniecki, Analiza wymagań dla metod przetwarzania wstępnego obrazów w automatycznym rozpoznawaniu tekstu,

http://wbieniec.kis.p.lodz.pl/research/files/05_Bronislawow_OCR.pdf [12.11.2017].

[2] Tobias Blanke, Michael Bryant, Mark Hedges, Open source optical character recognition for historical research, Journal of Documentation 68 (2012), 659-683.

[3] Inad Aljarrah, Osama Al-Khaleel, Khaldoon Mhaidat, Mu'ath Alrefai, Abdullah Alzu'bi, Mohammad Rabab'ah, Automated System for Arabic Optical Character Recognition with Lookup Dictionary, Journal of Emerging Technologies in Web Intelligence 4 (2012), 362-370.

[4] Abbyy Technology Portal, https://abbyy.technology/en:start, [22.11.2017].

[5] The Tesseract open source OCR engine, http://code.google.com/p/tesseract-ocr [20.11.2017].

[6] https://products.aspose.com/ocr, [01.11.2017].

[7] GOCR open-source character recognition, http://jocr.sourceforge.net, [25.11.2017].

[8] www.gnu.org/software/ocrad/manual/ocrad_manual.html, [10.12.2017].

[9] Review of Linux OCR software, https://www.mathstat.dal.ca/~selinger/ocr-test [01.12.2017].

[10] Linux OCR Software Comparison, httpswww.splitbrain.org/blog/2010-06/15-linux_ocr_software_comparison, [02.12.2017].

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 195-199
Wysłane: 2017-12-09
Przyjęte: 2017-12-19

Porównanie możliwości tworzenia aplikacji internetowych w środowisku JEE na przykładzie Spring Boot i Vaadin

Beniamin Abramowicz*, Beata Pańczyk

Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przeanalizowano możliwości tworzenia aplikacji internetowych na platformie Java Enterprise Edition przy zastosowaniu Vaadin i Spring Boot. Do przeprowadzenia analizy opracowano dwie aplikacje testowe typu CRUD, zaimplementowane z wykorzystaniem obu technologii. Przetestowano elementy implementacji, wydajność pracy z bazą danych oraz efektywność ładowania stron w przeglądarce.

Słowa kluczowe: Vaadin; Spring Boot; JEE; aplikacje internetowe

*Autor do korespondencji.

Adres e-mail: beniamin.abramowicz@pollub.edu.pl

Comparison of web applications development possibilities in JEE environment by the example of Spring Boot and Vaadin

Beniamin Abramowicz*, Beata Pańczyk

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

Abstract. The paper presents the analysis of capabilities of creating web applications on the Java Enterprise Edition platform using Spring Boot and Vaadin. To perform the analysis, there were used test applications, implemented in both technologies. Elements of implementation, work efficiency with the database and the efficiency of page loading in the browser were tested.

Keywords: Vaadin; Spring Boot; JEE; web applications

*Corresponding author.

E-mail address: beniamin.abramowicz@pollub.edu.pl

1. Wstęp

Do tworzenia aplikacji internetowych w języku Java wykorzystywana jest platforma Java Enterprise Edition (JEE) oraz najbardziej popularny szkieletem programistyczny Spring MVC [1]. W ostatnich latach pojawiły się nowe rozwiązania mające na celu usprawnienie procesu tworzenia aplikacji na tej platformie. Przykładem takich technologii jest Spring Boot oraz Vaadin.

Vaadin [2] upraszcza tworzenie interfejsu graficznego użytkownika, Spring Boot [3] natomiast automatyzuje proces konfiguracji aplikacji. Obie technologie znajdują się w pierwszej dziesiątce pod względem popularności (Rys. 1) [4].

Vaadin jest frameworkiem zaprojektowanym tak, aby tworzenie i utrzymywanie wysokiej jakości webowych interfejsów użytkownika było łatwe. Obsługuje dwa różne modele programowania: stronę serwera (ang. back end) i stronę klienta (ang. front end). Model serwera pozwala zapomnieć o stronie webowej i programowaniu interfejsu użytkownika [2, 5]. Strona klienta jest wykonywana jako JavaScript w przeglądarce. Nie potrzeba do tego żadnych dodatkowych wtyczek. Vaadin opiera się na wsparciu Google Web Toolkit dla szerokiej rangi przeglądarek, więc programista nie musi się martwić o wsparcie z ich strony.

Spring Boot jest platformą programistyczną stworzoną w celu uproszczenia pisania aplikacji internetowych w Spring MVC. Dzięki niemu konfiguracja aplikacji w celu jej poprawnego uruchomienia i działania zajmuje dużo mniej czasu. Wiele niezbędnych elementów konicznych do prawidłowej konfiguracji jest uproszczonych do zaledwie kilku linijek kodu.

Java Web Frameworks Index — REBELLABS

Rank	Framework	Popularity
1	Spring MVC	32.00
2	JSF	22.30
3	GWT	9.84
4	Spring Boot	9.45
5	Grails	8.50
6	Struts	7.21
7	Play framework	6.00
8	Vaadin	2.75
9	Dropwizard	1.21
10	JHipster	0.74

Rys 1. Popularność frameworków JEE [4]

2. Cel badań

Do badań wykorzystano dwie aplikacje testowe o tych samych funkcjonalnościach typowej aplikacji CRUD. Aplikacja Vaadin i aplikacja Spring Boot korzystają ze wspólnej bazy danych MySQL. *Back end* obu aplikacji napisany jest w Spring Boot. Vaadin zapewnia *front end*

pierwszej aplikacji testowej. Druga aplikacja wykorzystuje widoki html generowane za pomocą silnika Thymeleaf [6]. Dodatkowo stosuje bibliotekę jQuery oraz Ajax. Do pracy z danymi obie aplikacje korzystają z Hibernate [7]. Pomimo pewnych elementów wspólnych obu aplikacji testowych – implementacja strony serwerowej jest różna.

Celem badań było przeanalizowanie możliwości nowych rozwiązań oraz zbadanie ich wydajności w odniesieniu do aplikacji webowych w środowisku JEE.

3. Aplikacje testowe

Do przygotowania aplikacji testowych wykorzystano:

- Eclipse Oxygen w wersji 4.7.0 – środowisko programistyczne;
- serwer bazy danych MySQLw wersji 5.1.44;
- serwer Apache Tomcat w wersji 8.0;
- Java Enterprise Edition w wersji 8;
- frameworki Vaadin i Spring Boot;
- Hibernate – ORM do pracy z danymi.

Utworzone aplikacje testowe obsługują magazyn hurtowni komputerowej. Umożliwiają:

- wyświetlanie wszystkich produktów z hurtowni;
- dodawanie produktów;
- edycję istniejących produktów;
- usuwanie produktów;
- wyszukiwanie produktów korzystając z różnych kryteriów wyszukiwania.

Oprogramowanie i parametry urządzenia testowego przedstawiono w tabeli 1.

Tabela 1. Parametry i oprogramowanie urządzenia testowego

Parametry/Oprogramowanie	Wersja
System operacyjny	Windows 10 64-bit
Procesor	Intel core i3-2370M 2 rdzenie 2,4 GHz
Pamięć RAM	4 GB DDR3 SDRAM 665MHz
MySQL	5.1.44
Apache Tomcat	8.0
Java	JEE 8

4. Analiza możliwości Vaadin i Spring Boot

W ramach badań:

- porównano struktury obu aplikacji;
- przeprowadzono testy wydajności pracy na bazie danych (pobierania, wyświetlania, zapisywania i wyszukiwania danych w bazie);
- przeprowadzono testy wydajnościowe interfejsu w przeglądarce internetowej;
- porównano podstawowe metryki kodu.

4.1. Struktura aplikacji

Każda z aplikacji posiada charakterystyczną dla siebie strukturę i hierarchię plików. W przypadku technologii Vaadin i Spring Boot występuje dość dużo podobieństw wynikających z modelu aplikacji (Rys. 2):

- **Model aplikacji** – pliki oznaczone kolorem żółtym;
- **Kontroler aplikacji** – pliki oznaczone kolorem jasno zielonym. W przypadku Vaadin kontroler jest zastąpiony prezenterem, którego elementy są zintegrowane z widokami;
- **Widoki aplikacji** – pliki oznaczone kolorem jasno niebieskim;
- **Repozytorium** – pliki oznaczone kolorem czerwonym;
- **Klasa główna uruchamiająca aplikację** – plik oznaczony kolorem jasno różowym;
- **Nawiązanie sesji przy pomocy Hibernate** – pliki oznaczone kolorem ciemno szarym;
- **Style aplikacji** – folder oznaczony kolorem fioletowym;
- **Folder plików wykonywalnych** – folder oznaczony kolorem jasno szarym;
- **Konfiguracja bazy danych i Hibernate** – pliki oznaczone kolorem pomarańczowym;
- **Konfiguracja projektu aplikacji** – plik oznaczony kolorem błękitno-zielonym.

4.2. Wydajność pracy z danymi

Testy wydajności obu aplikacji polegały na pomiarach czasu wykonywania operacji na danych w bazie zgodnie z 5 scenariuszami:

S1 - wyświetlenie wszystkich produktów (100 pomiarów).
S2 - dodanie 100 produktów (100 pomiarów).
S3 - wyszukiwanie po nazwie produktu (10 pomiarów).
S4 - wyszukiwanie po nazwie i typie produktu (10 pomiarów).
S5 - wyszukiwanie po nazwie, typie i zakresie ceny produktu (10 pomiarów).

Przykład 1 pokazuje kod dodający 100 produktów i pomiar czasu dla Vaadin i Spring Boot. Przykład 2 przedstawia kod z pomiarem czasu wyszukania produktu dla Vaadin a przykład 3 – dla Spring Boot.

Przykład 1. Dodanie produktów i pomiar czasu – Vaadin i Spring Boot

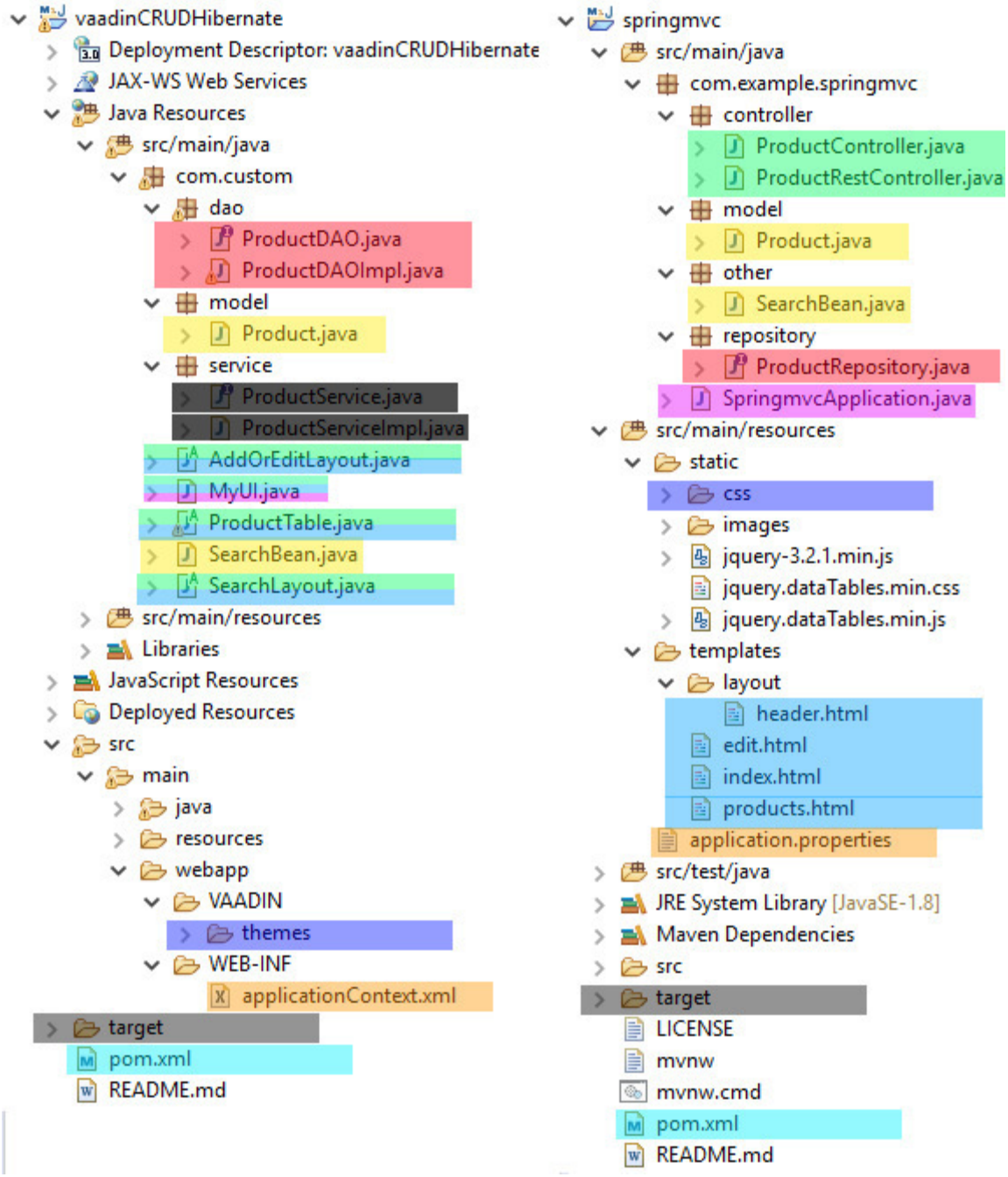
```
long t = System.currentTimeMillis();
for(int i = 0; i<100; i++) {
    Product p = new Product();
    p.setCategory("category");
    p.setDescription("description");
    p.setName("name"+ i);
    p.setPrice(123.0 + i);p.setType("type");
    productService.addProduct(p);
}
long t1 = System.currentTimeMillis();
System.out.println("/product/add100 ->" + (t1-t));
```

Przykład 2. Pomiar czasu wyszukiwania produktu – Vaadin

```
long t = System.currentTimeMillis();
List<Product> a = this.productDao.search(searchBean);
long t2 = System.currentTimeMillis();
System.out.println("/product/search1 ->" + (t2-t));
```

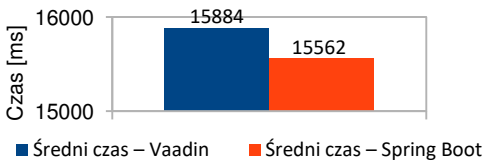
Przykład 3. Pomiar czasu wyszukiwania produktu – Spring Boot

```
Specification<Product>productSpecification=ProductSpecificat
ions.search(searchBean);
List<Product> list =
productRepository.findAll(productSpecification);
long t1 = System.currentTimeMillis();
System.out.println("/product/search1 ->" + (t1-t));
```

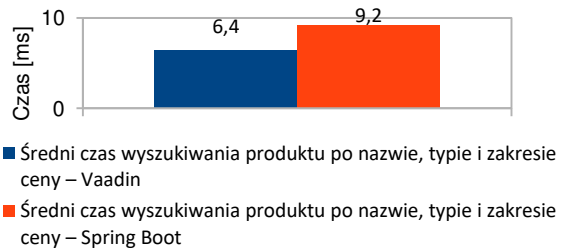


Rys 2. Struktura i hierarchia plików aplikacji Vaadin i Spring Boot

Rysunek 3 przedstawia wykres średniego czasu dodania 100 produktów dla obu aplikacji, natomiast rysunek 4 wykres średniego czasu wyszukiwania po kryteriach: nazwa, typ, zakres ceny.



Rys 3. Średni czas dodania 100 produktów



Rys 4. Średni czas wyszukiwania produktów

W tabeli 2 przedstawiono średni czas dla poszczególnych scenariuszy.

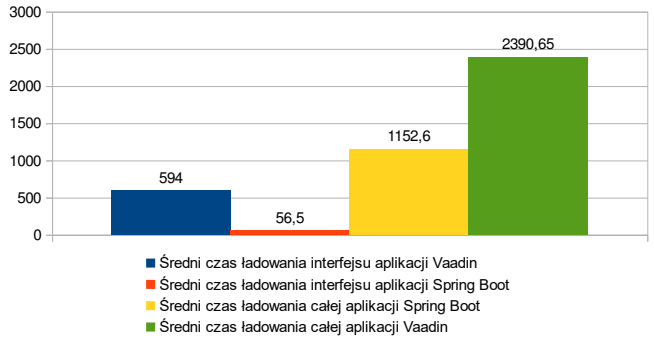
Tabela 2. Średnie czasy operacji na danych dla poszczególnych scrnariuszy

Scenariusz nr	Średnia (Vaadin) [ms]	Średnia (Spring Boot) [ms]
1	13	10
2	15884	15562
3	3,8	6
4	3,5	6
5	6,4	9,2

Analizując wyniki otrzymane z testów można stwierdzić, że wyświetlanie i dodawanie rekordów z bazy jest nieznacznie szybsze w Spring Boot, jednak Vaadin radzi sobie lepiej w wyszukiwaniu produktów nawet dwa razy szybciej.

4.3. Wydajność interfejsu graficznego w przeglądarce

Do wykonania pomiarów wykorzystano ***Performance Panel***, narzędzie deweloperskie dostępne w Google Chrome, które: zbiera dane o poszczególnych modułach ładowanych podczas wejścia na stronę. Na rysunku 5 przedstawiono wykres średnich czasów renderowania interfejsu oraz pobrania wszystkich modułów aplikacji. Wyniki jednoznacznie pokazują, że ładowanie interfejsu graficznego dla Spring Boot jest nawet kilkunastokrotnie szybsze niż dla Vaadin. Wynika to przede wszystkim z tego, że widoki Vaadin są implementowane w języku Java i dopiero podczas uruchamiania aplikacji zostają przetłumaczone na język html. Natomiast widoki Spring Boot, pisane w ThymeLeaf, nie wymagają takiej konwersji.



Rys 5. Średnie czasy renderowania interfejsu i ładowania całej strony dla obu aplikacji

4.4. Metryki kodu

Do zliczenia linii kodu obu aplikacji wykorzystano program LocMetrics [8]. Tworzy on logi, w których liczba linii kodu jest wyświetlona z podziałem na poszczególne pliki, pomijając puste linie i komentarze. W tabeli 3 przedstawiono liczbę linii kodu dla każdej z aplikacji.

Tabela 3. Metryki kodu z podziałem na pliki dla Vaadin i Spring Boot

Nazwa pliku	Rola pliku	Liczba linii kodu Vaadin	Liczba linii koduSpring Boot
Product.Java	Model	71	74
SearchBean.java	Model	86	98
ProductController.java	Kontroler	-	59
ProductRestController.java	Kontroler	-	188
SpringmvcApplication.java	Klasa główna uruchamiająca aplikację	-	13
MyUI.java	Klasa główna uruchamiająca aplikację/widok	175	-
ProductTable.java	Widok	108	-
AddOrEditLayout.java	Widok	71	-
SearchLayout.java	Widok	112	-
Header.html	Widok	-	39
Edit.html	Widok	-	56
Index.html	Widok	-	10
Products.html	Widok	-	211
ProductDAO.java	Interfejs	20	-
ProductDAOImpl.java	Repozytorium/implementacja interfejsu	136	-
ProductRepository.java	Repozytorium	-	14
ProductService.java	Interfejs	24	-
ProductServiceImpl.java	Operacje na bazie/implementacja interfejsu	88	-
ApplicationContext.xml	Konfiguracja bazy danych i Hibernate	39	-
Applications.properties	Konfiguracja bazy danych i Hibernate	-	7
Pom.xml	Konfiguracja projektu	308	74
SpringmvcApplicationTest.java	Klasa testowa sprawdzająca Spring Boot	-	18
Razem linii kodu	-	**1238**	**861**
Waga pliku wykonywalnego [MB]	-	**43.97**	**35.63**

Aplikacja w Spring Boot zajmuje o około 1/3 mniej kodu niż aplikacja w Vaadin. Jest to spowodowane przede wszystkim uproszczeniem konfiguracji aplikacji do minimum: konfiguracja projektu, połączenia z bazą danych, Hibernate, repozytoriów. W konsekwencji również plik wykonywalny aplikacji Spring Boot ma mniejszy rozmiar o prawie 10 MB.

5. Ocena Spring Boot i Vaadin

Subiektywna ocena autorów obu frameworków przedstawiona jest w tabeli 4. Zastosowano skalę ocen 1-5, gdzie 5 ocena najwyższa..

Tabela 4. Ocena obu technologii

Kryterium	Vaadin	Spring Boot
Dostępność do materiałów/poradników	3	4
Dostęp do narzędzi programistycznych	5	5
Przejrzystość struktury plików aplikacji	4	5
Przejrzystość modelu tworzenia aplikacji	3	4
Łatwość implementacji interfejsu graficznego	5	4
Łatwość implementacji zaplecza aplikacji	2	4
Łatwość konfiguracji	3	5
Wygoda tworzenia stylów aplikacji	5	3
Wydajność wyświetlania rekordów z bazy	3	5
Wydajność dodawania rekordów do bazy	4	4
Wydajność wyszukiwania rekordów używając różnych kryteriów wyszukiwania	5	4
Szybkość renderowania interfejsu	2	5
Szybkość uruchamiania aplikacji	2	5
Liczba linii kodu potrzebna do stworzenia w pełni funkcjonalnej i działającej aplikacji (5 - bardzo mało, 1 - bardzo dużo)	3	4
Ocena końcowa	**49**	**61**

6. Wnioski

Technologie Spring Boot i Vaadin pozwalają na szybkie tworzenie aplikacji internetowych. Spring Boot ułatwia konfigurację aplikacji, połączenie z bazą danych oraz konfigurację repozytorium, podczas gdy Vaadin przyspiesza tworzenie widoków dzięki zastosowaniu języka Java, a także zawarciu prezentera, czyli pośrednika pomiędzy modelem a widokami. Dzięki temu wyszukiwania elementów po określonych kryteriach i wyświetlenie wyników odbywa się szybciej niż w Spring Boot. Jednocześnie tworzenie widoków bezpośrednio w języku Java ma też wady, a mianowicie widoki muszą być konwertowane z języka Java do HTML co wydłuża renderowanie interfejsu i ładowanie całej strony. Spring Boot jest szybszy między innymi dlatego, iż widoki mogą być pisane w HTML za pomocą ThymeLeaf. Każda z technologii ułatwia programowanie aplikacji internetowych w JEE. Wybór zależy w dużej mierze od tego czy potrzebna jest szybka konfiguracja bazy i szybkie wczytywanie stron, czy prosty do implementacji interfejs użytkownika.

Literatura

[1] http://www.oracle.com/technetwork/java/javaee/tech/index.html [11.10.2017]

[2] https://vaadin.com/docs [1.12.2017]

[3] https://spring.io/guides/gs/spring-boot/, [1.12.2017]

[4] https://zeroturnaround.com/webframeworksindex/ [14.12.2017]

[5] https://demo.vaadin.com/book-examples/book, 12.03.2017

[6] http://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html, [16.12.2017]

[7] http://hibernate.org/, [16.12.2017]

[8] http://www.locmetrics.com [14.12.2017]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 200-204
Wysłane: 2017-12-09
Przyjęte: 2017-12-12

Porównanie szkieletów aplikacji Angular i BackboneJS na przykładzie aplikacji internetowej

Mateusz Moczulski*, Małgorzata Plechawska-Wójcik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł zawiera porównanie szkieletów aplikacji Angular i BackboneJS. Dla potrzeb badawczych zostały utworzone dwie aplikacje jednostronicowe, oferujące identyczne funkcjonalności. Zmierzony i porównany został czas ładowania elementów aplikacji, liczba linii napisanego kodu, rozmiar skompilowanych aplikacji, społeczność zgromadzona wokół szkieletu oraz czas potrzebny na wykonanie aplikacji. Otrzymane wyniki nie wykazały rozwiązania wyraźnie lepszego do zastosowania w aplikacjach jednostronicowych.

Słowa kluczowe: JavaScript; Angular; BackboneJS; aplikacja jednostronicowa

*Autor do korespondencji.

Adres/adresy e-mail: mateusz.moczulski@dzielo.pl, m.plechawska@pollub.pl

A comparative analysis of selected Java Script frameworks in the context of web applications on the example of Angular and BackboneJS

Mateusz Moczulski*, Małgorzata Plechawska-Wójcik
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The paper contains comparison between Angular and BackboneJS frameworks. For research purposes, two single page applications were created. Both providing the same functionalities. In this article such elements were examined and compared: time of loading elements, number lines of code, size of compiled application. community gathered around frameworks and time needed to create the application using frameworks. The obtained results did not indicate a solution that was clearly better to be used in single page applications.

Keywords: JavaScript; Angular; BackboneJS; Single Page Application

*Corresponding author.

E-mail address: mateusz.moczulski@dzielo.pl, m.plechawska@pollub.pl

1. Wstęp

JavaScript jest jednym z najpopularniejszych języków programowania, a wraz z rozwojem Node.js zyskał na popularności zarówno w zakresie front-end, jak i back-end [1]. Framework JavaScript różni się od biblioteki JavaScript w swoim strumieniu kontrolnym. Biblioteka oferuje funkcje, które mają być wywoływane przez swój kod nadrzędny, podczas gdy framework definiuje cały projekt aplikacji. Framework wywołuje oraz używa kodu napisanego przez developera w pewien szczególny sposób. Większość szkieletów JavaScript jest zgodnych z paradygmatem model-widok-kontroler, zaprojektowanym w celu segregowania aplikacji WWW na ortogonalne jednostki w celu poprawy jakości kodu i łatwości konserwacji [2]. Takimi szkieletami są Angular oraz Backbone.

Angular oraz Backbone wykorzystywane są zazwyczaj do wytwarzania aplikacji jednostronicowych (ang. Single Page Application). Aplikacja jednostronicowa to aplikacja internetowa lub witryna internetowa, która współdziała z użytkownikiem poprzez dynamiczne przepisywanie bieżącej strony, zamiast ładowania całych nowych stron z serwera. Takie podejście pozwala uniknąć przerw w korzystaniu użytkownika z kolejnych stron, dzięki czemu aplikacja zachowuje się bardziej jak aplikacja desktopowa. W SPA wszystkie niezbędne kody - HTML, JavaScript i CSS - są pobierane przy pojedynczym załadowaniu strony lub odpowiednie zasoby są dynamicznie ładowane i dodawane do strony w razie potrzeby, zwykle w odpowiedzi na działania użytkownika [3]. Strona nie wczytuje się ponownie w żadnym momencie procesu ani nie kontroluje transferu na inną stronę, chociaż za pomocą skrótu lokalizacji lub interfejsu API historii HTML5 można zapewnić percepcję i możliwość nawigacji osobnych stron logicznych w aplikacji. Aplikacja taka jest znacznie szybsza od aplikacji, w której każda strona jest ładowana oddzielnie [4].

Frameworkami dostępnymi obecnie na rynku są także Backbone oraz Angular. Artykuł podejmuje temat porównania powyższych szkieletów. W każdym z nich została utworzona identyczna aplikacja, która oferowała te same funkcjonalności.

2. Angular

Angular 4 jest jednym z frameworków dostępnych na rynku wspomagających wytwarzanie SPA. Jest wspierany i rozwijany przez developerów firmy Google. Pierwsze wydanie Angular miało miejsce w 2009r. Cały czas produkt był rozwijany i w 2017 roku weszła wersja 4.0. Od wersji 2.0 Angular korzysta z TypeScript zamiast JavaScript [5].

Angular to zestaw narzędzi do tworzenia aplikacji internetowych. Jest w pełni rozszerzalny i działa dobrze z innymi bibliotekami. Każda cecha może zostać zmodyfikowana lub wymieniona, aby pasowała do potrzeb. HTML jest bardzo dobry do deklarowania statycznych dokumentów, jednak nie jest skuteczny, podczas próby użycia go do deklarowania dynamicznych widoków w aplikacjach internetowych [6]. Angular umożliwia rozszerzenie słownictwa HTML w aplikacji. Powstałe środowisko jest niezwykle wyraziste, czytelne oraz szybkie. Niewątpliwą zaletą Angular jest powiązanie danych. Jest to automatyczny sposób aktualizowania widoku, gdy model się zmienia, a także aktualizuje model, gdy zmienia się widok. Jest to bardzo użyteczne, ponieważ eliminuje manipulacje DOM (Obiektowy Model Dokumentu, *ang. Document Object Model*); z listy rzeczy, na które trzeba zwracać uwagę [7][6].

3. BackboneJS

Backbone.js to lekka biblioteka JavaScript, która dodaje strukturę do kodu klienckiego. Ułatwia to zarządzanie aplikacją, pozostawiając kod, który jest dłużej możliwy do utrzymania.

Backbone jest dojrzały, popularny i ma zarówno tętniącą życiem społeczność deweloperską, jak i mnóstwo wtyczek i rozszerzeń [8]. Korzysta on z wzorca MVC. Wykorzystywany jest do tworzenia nietypowych aplikacji przez firmy takie jak Disqus, Walmart, SoundCloud i LinkedIn.

Backbone dostarcza minimalny zestaw do strukturyzacji danych (modele, kolekcje) i interfejsów użytkownika (widoki, adresy URL), które są przydatne podczas tworzenia dynamicznych aplikacji przy użyciu kodu JavaScript. Można użyć zalecanej architektury oferowanej przez twórców lub rozszerzyć ją, aby spełnić wymagania developera. Biblioteka nie skupia się na widżetach ani nie zastępuje sposobu, w jaki są tworzone obiekty. Dostarcza ona narzędzia do manipulacji i kwerendy danych w aplikacji [9].

4. Wybór odpowiedniego szkieletu

Frameworki JavaScript stają się bardzo pomocne przy szybkim tworzeniu aplikacji internetowych. Dlatego ważne jest, aby wybrać najlepszy szkielet dostosowany do stawianych wymagań. Istotnym czynnikiem, który decyduje o wyborze szkieletu są jego koszty, developerzy wolą korzystać z rozwiązań darmowych. Ważnym czynnikiem wyboru szkieletu jest także przestrzeń dyskowa jaka jest zajmowana przez gotową aplikację. Istotne znaczenie podczas wyboru szkieletu odgrywają również:

1) modułowość;
2) przenośność;
3) możliwość manipulacji DOM,
4) częste aktualizacje [10].

5. Procedura badawcza

Przed przystąpieniem do realizacji tematu należało określić kryteria analizy, jaka będzie przeprowadzona podczas porównywania frameworka Angular oraz Backbone. W artykule szkielety będą badane pod względem praktycznym. Analiza praktyczna ma na celu analizę działającego produktu. Pozwala ona określić czy napisana aplikacja spełnia wymagania wydajnościowe oraz oczekiwania klienta. Do zastosowanych kryteriów porównawczych należą: analiza czasu ładowania elementów, analiza liczby linii kodu oraz analiza rozmiaru skompilowanego kodu.

Na potrzeby badań założono również testowe Rest API, z którego aplikacje po załadowaniu pobierają potrzebne dane. W przypadku obu aplikacji pobieranie danych odbywa się od razu po włączeniu.

Do wyświetlenia badanej tabeli w przypadku Angular posłużyła metoda *ngFor* (Przykład 1). Jest to metoda umożliwiająca wyświetlenie elementów kolekcji w postaci HTML. W przypadku Backbon posłużyła do tego metoda *mapObject* (Przykład 2). Działa ona analogicznie jak opisana wcześniej metoda *ngFor*.

Przykład 1. Wyświetlanie tabeli rekordów – Angular

```
<tr *ngFor="let carss of cars; let i = index">
<td>
{{carss.id}}
</td>
<td>
{{carss.mark}}
</td>
<td>
{{carss.model}}
</td>
<td>
{{carss.price}}
</td>
<td>
{{carss.yearBook}}
</td>
<td>
{{carss.carMileage}}
</td>
<td width = 150 px>
<button id={{i+1}} type="button" class="btn btn-success"
(click)="onClick($event)">Pokaż szczegóły</button>
</td>
</tr>
```

Przykład 2. Wyświetlanie tabeli rekordów - Backbone

```
<%
_.mapObject(cars, function(cars) { %>
<tr>
<td>
<%- cars.get('id') %>
</td>
```

```
<td>
<%- cars.get('mark') %>
</td>
<td>
<%- cars.get('model') %>
</td>
<td>
<%- cars.get('price') %>
</td>
<td>
<%- cars.get('yearBook') %>
</td>
<td>
<%- cars.get('carMileage') %>
</td>
<td width = 150 px>
<button id="button" class="btn btn-success" data-
carsId=<%- cars.get('id') %>>Pokaż szczegóły</button>
</td>
</tr>
```

6. Wyniki badań

Pierwszym badanym elementem będzie czas ładowania elementów aplikacji (Tabela 1). Jest to ważne kryterium badawcze szkieletów. Przed wyborem frameworka warto się zastanowić jakie elementy będą szybciej działały w jakiej technologii. Jest to szczególnie istotne w przypadku pisania rozbudowanej aplikacji, z której będzie korzystało wielu użytkowników.

Tabela 1. Czasy ładowania elementów aplikacji

	Angular [ms]	Backbone [ms]
Ładowanie całego projektu	1867	615
Widok - kontakt	1.24	0.71
Widok - rejestracja	0.95	0.98
Widok - logowanie	1.37	0.84
Widok - wszystkie samochody	3.27	7.89
Widok - strona główna	1.63	1.93
Widok – szczegóły samochodu	2.32	3.42

Badania zostały przeprowadzone także pod kątem szybkości ładowania elementów do tabeli (Tabela 2). Przeprowadzono badania na różnej liczbie rekordów, aby osiągnąć bardziej widoczne różnice.

Tabela 2. Czasy ładowania tabeli z rekordami

	Angular [ms]	Backbone [ms]
15 rekordów	2.98	7.89
100 rekordów	3.16	10,78
5000 rekordów	3381.70	3209.50
20000 rekordów	8928.80	9872.90
40000 rekordów	14829.58	12022.91
70000 rekordów	29742.29	26927.98
120000 rekordów	54687.29	49782.98

Zbadano również liczbę linii kodu wykorzystanej do wykonania aplikacji (Tabela 3). Wykonanie niektórych elementów aplikacji w różnych frameworkach wymaga napisania różnej liczby kodu. Mniejsza liczba linii kodu często przekłada się na mniejszy czas wymagany na napisanie danego elementu. W badaniu liczona jest liczba linii kodu oraz liczba znaków wykorzystana do wykonania danego elementu.

Tabela 3. Liczba linii kodu wykorzystana do napisania elementu

	Angular (liczba linii kodu/liczba znaków)	Backbone (liczba linii kodu/liczba znaków)
Slider na stronie głównej	20/491	25/859
Przełączanie pomiędzy widokami	14/437	26/835
Pobranie danych z samochodami	4/131	15/253
Wyświetlenie tabeli z samochodami	66/1154	66/1891
Wyświetlanie zdjęć samochodu	9/249	14/257
Wyświetlanie danych samochodu	31/719	34/828
Wysłanie danych wprowadzonych podczas rejestracji do serwera	8/301	6/161
Wyświetlenie komunikatu o zalogowaniu	11/218	19/377
Cała aplikacja	232/4315	311/6198

Zbadano również rozmiar aplikacji (Tabela 4). Im większy projekt tym dłużej będzie się ładował na urządzeniach użytkowników. Dlatego warto zabiegać, aby cały projekt miał jak najmniejszą wagę.

Tabela 4. Rozmiar skompilowanych projektów

	Angular	Backbone
Rozmiar całego projektu	520 kb	32 kb

Następnym badanym elementem będzie badanie społeczności korzystającej z danego szkieletu (Tabela 5). Większa liczba użytkowników świadczy o większej możliwości pomocy na forach internetowych. Często też doświadczeni użytkownicy nagrywają swoje materiały i udostępniają w sieci.

Tabela 5. Społeczność frameworków w wybranych serwisach

	Angular	Backbone
stackoverflow.com [11]	66119 zapytań	20413 zapytań
gitter.im [12]	800tys. użytkowników	-
github.com [13]	27 tys. gwiazd	26 tys. gwiazd
youtube.com [14]	700 tys. materiałów	30 tys. materiałów

Następnym elementem jaki będzie analizowany jest czas jaki został wykorzystany na wykonanie danego elementu (Tabela 6). Czas był mierzony za pomocą aplikacji Licznik

czasu zadań dostępnej w przeglądarce Chrome. Czasy zostały zaokrąglone do dziesiętnych części godziny.

Twórcy frameworków starają się jak najbardziej skrócić czas jaki jest niezbędny do tworzenia elementów aplikacji przez deweloperów. Czas zaoszczędzony na aplikacji często przekłada się na niższe koszty wytworzenia aplikacji, Czas potrzebny na wykonanie aplikacji w danym frameworku jest podstawowym kryterium branym pod uwagę do wyceny przez firmy zajmujące się wytwarzaniem oprogramowania. Zaoszczędzony czas można przeznaczyć na większą liczbę testów co przekłada się nieraz na jakość wytworzonego oprogramowania.

Tabela 6. Czas pracy wykorzystany na wykonanie elementów aplikacji

	Angular [h]	Backbone [h]
Utworzenie strony głównej	1	1,2
Utworzenie slidera	2.5	2.2
Utworzenie przejścia pomiędzy widokami	0.5	1.8
Utworzenie pozostałych widoków	2.5	4.2
Wyświetlenie tabeli z wszystkimi samochodami	0.8	3
Wyświetlenie zdjęć w szczegółach samochodu	1.5	1.5
Wyświetlenie widoku z podglądem zdjęcia	2.6	2
Pobranie danych do modelu	1.6	2.2

W tabeli 7 przedstawiono zbiorczą ocenę od 1 do 5 (5 – najlepsza ocena, 1 – najgorsza) szkieletów jakie uzyskały w badanych kryteriach. Jest to subiektywna ocena autorów artykułu.

Tabela 7. Zbiorcza ocena szkieletów w poszczególnych kryteriach

	Angular	Backbone
Czas ładowania elementów aplikacji	4	5
Czas ładowania tabeli z rekordami	4	5
Liczba linii kodu	5	3
Rozmiar skompilowanego kodu	2	5
Społeczność frameworków	5	2
Czas wykorzystany na wykonanie aplikacji	5	3
Suma	25	23

7. Analiza wyników

Angular oraz Backbone osiągają podobne wyniki czasu ładowania elementów (Tabela 1 i 2). Różnice między nimi mogą pojawiać się w przypadku ładowania bardzo dużych ilości danych. Czasy ładowania widoków w aplikacji są zbliżone. Lepszy czas ładowania całej aplikacji osiąga Backbne. W przypadku ładowania rekordów do tabeli lepsze wyniki osiąga Backbone, jednak aby zauważyć różnicę w czasie ładowania elementów muszą być to wielkości rzędu co najmniej kilkudziesięciu tysięcy rekordów. W przypadku ładowania zdjęć do aplikacji nieco lepiej radzi sobie Angular, jednak również aby różnica była zauważalna musi być to bardzo duża liczba zdjęć.

Backbone jest frameworkiem, który wymaga napisania większej ilości kodu (Tabela 3) w porównaniu do Angular. Wykorzystanie narzędzia dla programistów Angular CLI służącego do budowania elementów aplikacji za pomocą komend ułatwia oraz przyśpiesza pracę nad aplikacją. Napisanie aplikacji w Backbone wymagało ok 43% większej liczby znaków niż w Angular

Rozmiar skompilowanego kodu (Tabela 4) w Angular jest znacznie większy niż w przypadku Backbone. Ma na to wpływ to co jest dodawane przez sam szkielet do projektu oraz sama waga frameworka. Rozmiar kodu może mieć wpływ na czas wczytywania aplikacji co potwierdza tabela 1. Różnica może być jeszcze bardziej odczuwalna przez użytkowników ze słabym połączeniem internetowym.

Angular posiada większą społeczność (Tabela 5), co powoduje, że łatwiej będzie znaleźć użytkownikowi rozwiązanie problemów, oraz uzyskać pomoc od innych użytkowników. Istnieje więcej materiałów do nauki Angular dostępnych bezpłatnie w sieci. Producenci Angular organizują również specjalne spotkania dla społeczności zrzeszonej wokół swojego produktu.

Czas potrzebny na wykonanie aplikacji (Tabela 6) za pomocą Backbone był o ok 26% większy niż w przypadku Angular Jest to dość znaczna różnica, która w przypadku bardziej skomplikowanej aplikacji mogła by mieć decydujący wpływ na wybór szkieletu. Różnica ta może znacznie przekładać się na koszty potrzebne na wykonanie aplikacji.

8. Wnioski

W dzisiejszych czasach technologie, które są stosowane do wytwarzania oprogramowania cały czas ulegają zmianie. Często zastępują je rozwiązania nowsze i bardziej rozbudowane, dające developerom więcej możliwości. Obecnie na rynku dostępnych jest wiele frameworków typu open-source, które umożliwiają tworzenie aplikacji prostych oraz bardziej zaawansowanych. Wraz z zastosowaniem nowych szkieletów praca developerów jest szybsza oraz wymaga napisania mniejszej liczby kodu. Niniejsza praca porównuje wybrane rozwiązania, pozwala programiście na wybór odpowiedniego szkieletu do zastosowania we własnej aplikacji.

Otrzymane wyniki pozwoliły stwierdzić, że oba frameworki nadają się do zastosowania w aplikacji jednostronicowej co pokazuje tabela 7. Angular posiada większą społeczność, powoduje to że programiście łatwiej będzie znaleźć rozwiązanie problemów, uzyskać pomoc od innych użytkowników, oraz znaleźć materiały do nauki. Analiza czasu ładowania elementów pozwoliła stwierdzić, że nie ma dużej różnicy w wynikach obu szkieletów. Backbone osiągnął lepszy wynik w przypadku ładowania tabeli z różną liczby rekordów. Różnice te były by jednak

zauważalne dopiero w przypadku dziesiątek tysięcy elementów. Analiza złożoności kodu pokazała, że wykonanie aplikacji z wykorzystaniem szkieletu Angular wymaga mniejszej ilości kodu do napisania. Czas potrzebny na wykonanie aplikacji z wykorzystaniem Angular był znacznie mniejszy.

Wybór konkretnego szkieletu zależy więc od potrzeb i preferencji. Oba badane frameworki dostarczają kompletny zestaw funkcji do wykonania aplikacji internetowej typu single page aplication. Oba szkielety nie są pozbawione wad oraz zalet. Ułatwiają one oraz skracają czas jaki jest potrzebny na wykonanie rozbudowanej aplikacji. Główną zaletą Angular jest duża społeczność zgromadzona wokół szkieletu oraz niewielka ilość czasu i liczba linii kodu potrzebna na wykonanie aplikacji. Największą wadą szkieletu jest duży rozmiar skompilowanego projektu. Największą zaletą Backbone jest mały rozmiar skompilowanego projektu oraz niewielka ilość czasu potrzebne na załadowanie elementów aplikacji. Największą wadą tego szkieletu jest mała społeczność korzystająca z frameworka, wysoka liczba linii kodu potrzebnego na wykonanie aplikacji oraz dłuższy czas potrzebny na wykonanie aplikacji niż w przypadku Angular.

Literatura

[1] Larry Ullman,: *Nowoczesny język JavaScript*, Helion, Gliwice 2013.

[2] Tim Amber, Nicholas Cloud,: *JavaScript Frameworks for Modern Web Dev,* Apress*,* Nowy Jork 2015.

[3] Michael S. Mikowski, Josh C. Powell*,: Single Page Web Applications*, Manning, Nowy Jork 2014.

[4] Mikito Takada,: *Single Page Application in Depth*, ebook, 2012.

[5] Ari Lerner, Felipe Coury, Nate Murray, Carlos Taborda,: *ng-book 2 The Complete Book on Angular 4*, Nowy Jok 2017.

[6] Jesse Palmer, Corinna Cohn, Michael Giambalvo, Craig Nishina ,; *Testing Angular Applications*, Manning, Nowy Jork 2017.

[7] Angular 4 documentation - https://angular.io/docs/ts/latest/ [01.12.2017]

[8] Addy Osmani ,: *Developing Backbone.js Applications*, O'Reilly, 2013.

[9] BackboneJS documentation - http://backbonejs.org/documentation/ [01.12.2017]

[10] Pano, A.; Graziothin, D.; Abrahamsson, P.; *What leads developers towards the choice of a JavaScript* framework?, 2016.

[11] stackoverflow.com [01.12.2017]

[12] https://gitter.im [01.12.2017]

[13] https://github.com/ [01.12.2017]

[14] https://www.youtube.com/ [01.12.2017]

JOURNAL COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 205-209
Wysłane: 2017-12-03
Przyjęte: 2017-12-12

Analiza porównawcza zastosowania szkieletów Angular2 i Ember.js

Jan Palak*, Małgorzata Plechawska-Wójcik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Celem artykułu jest analiza porównawcza bibliotek Angular2 z Ember.js. Na potrzeby testów została opracowana specjalna aplikacja internetowa. W świetle kryteriów porównawczych analizie poddano: szybkość generowania strony, wyświetlanie dużej liczby danych, obsługa multimediów, mechanizmy zabezpieczeń, dostępność dokumentacji, obsługę różnych formatów danych, integrację z portalami społecznościowymi. Ponadto celem jest również prezentacja cech wspólnych oraz różnic obu bibliotek.

Słowa kluczowe: Angular2; Ember.js; porównanie bibliotek Javascript

*Autor do korespondencji.

Adres e-mail: jpalak@o2.pl

Comparative analysis of the usage of Angular2 and Ember.js frameworks

Jan Palak*, Małgorzata Plechawska-Wójcik
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The focus of this article is the comparative analysis of the usage of the Angular2 and Ember.js frameworks. The requirement for tests of these frameworks has been developed though special web applications. The analysis was performed with the following comparative criteria: the speed of the website generation, presentation of large amount of data, media services, security mechanism, documentation availability, different data format handling, and integration with social media. Also, this paper aims to present common features and differences between the two frameworks regarding their characteristic features.

Keywords: Angular2; Ember.js; comparative analysis; Javascript frameworks

*Corresponding author.

E-mail address: jpalak@o2.pl

1. Wstęp

Możliwości JavaScript w porównaniu do zastosowania kiedyś a dziś różnią się diametralnie. Jeszcze nie dawno, bo 10 lat temu wykorzystywano go głównie do prostych operacji na poziomie formularza. Dzisiaj już jest o wiele bardziej zaawansowany i rozbudowany, nie tylko w przeglądarkach internetowych ale także w telefonach [1]. Model podwójnego wiązania elementów używanych w Javascript jest teraz powszechnie używany w dużych bibliotekach [2]. Ponadto służy do sterowania modułami jak i całymi stronami. sterowane jest za pomocą niego moduły a także całe strony [3].

W niniejszym artykule porównano ze sobą dwie popularne biblioteki Angular 2 oraz Ember.js. W obu technologiach została opracowana specjalna aplikacja testowa, która pobiera dane z serwera. Zbadano wydajność pracy bibliotek, ich możliwości oraz popularność wśród programistów.

2. Angular 2

Angular 2 to następca równie popularnej pierwszej wersji oznaczonej pod nazwą AngularJS [4]. Największą nowością w drugiej wersji Angular jest pełne wsparcie biblioteki ECMAScript 6 oraz języka TypeScript [5]. Sam język TypeScript cechami przypomina język Java: posiada klasy, interfejsy, adnotacje. Głównym zadaniem TypeScript jest kompilacja kodu do zwykłego JavaScriptu [6]. Proces pozwala szybciej pisać kod przez programistę, który jest odpowiednio zamieniany na postać najbardziej zrozumiałą dla przeglądarki [7].

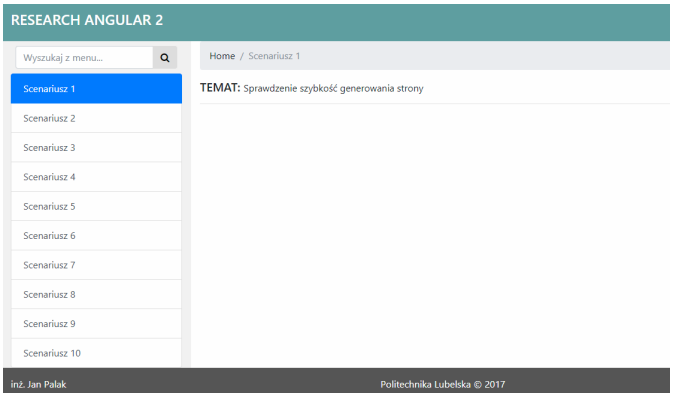
3. Ember.js

Otwarta biblioteka Javascript do tworzenia dużych aplikacji internetowych wykorzystująca wzorzec MVC (Model-Widok-Kontroler) [8]. Jest jednym z najpopularniejszych bibliotek Javascript do kontrolowania widoku na stronie. Wiele popularnych firm tj. Yahoo, Groupon, Discourse wykorzystuje obecnie tą bibliotekę w swoich aplikacjach. Biblioteka umożliwia tworzenie strony single-page-application opartej o szablony HTMLBars, które służą do automatycznego generowania widoku [9]. Są one kontrolowane przez warstwę kontrolera oraz routing aplikacji [10].

4. Opis aplikacji testowych

Do przeprowadzenia eksperymentu został przygotowany szablon (rys. 1), na podstawie którego opracowano logikę w Angular 2, Ember.js. Obydwie aplikacje mają charakter strony single page application, która polega na dynamicznym poruszaniu się stronie internetowej bez przeładowywania całej

witryny. Szablon zawiera w menu odniesienia do 10 scenariuszy, których opis znajduje się w rozdziale „Opis procedury badawczej”. Nad menu znajduje się wyszukiwarka do odpowiedniego scenariusza po nazwie, a po prawej stronie panel do wyświetlenia głównej części strony.



Rys. 1. Szablon aplikacji internetowej

Szablon został wykonany w najnowszej wersji biblioteki CSS: Bootstrap, która z dniem dzisiejszym została oznaczona jako v4.0.0-alpha.6.

Wspólna część back-end obydwu aplikacji została opracowana w technologii REST API w implementacji Java. Za pomocą odpowiednich URI z aplikacji internetowej można wywoływać, pobierać czy też autoryzować się po stronie serwera aplikacyjnego Tomcat. Odpowiedzi z serwera za pomocą badanych bibliotek można przetwarzać i sterować na stronie za pomocą kontrolera.

5. Opis procedury badawczej

Wszystkie testy zostały przeprowadzone według 11 scenariuszy badawczych.

Scenariusz 1: Szybkość generowania strony

W tym przypadku sprawdzona została szybkość ładowania strony. Wielkość jest wyrażona w liczbie wiadomości na stronie (od 10 do 100) Na jedną wiadomość składa się 100 wyrazów, obrazek i dwa filmy zewnętrzne z dwóch różnych źródeł.

Scenariusz 2: Obsługa popularnych zabezpieczeń

W tym scenariuszu sprawdzono czy dane biblioteki obsługują zabezpieczenia takie jak: Ciasteczka, LocalStorage (pamięć wewnętrzna przeglądarki), reCAPTCHA, Szyfrowanie, JWT (Javascript WebToken). Są to najpopularniejsze technologie, w których można przechowywać ukryte dane, niedostępne dla innych użytkowników.

Scenariusz 3-6: Obsługa różnych formatów danych

Te scenariusze miały za zadanie sprawdzenie czy badane technologie obsłużą formaty danych takie jak: JSON, XML, YAML.

Scenariusz 7: Integracje z portalami społecznościowymi

Scenariusz ten miał za zadanie sprawdzenie czy możliwa jest autoryzacja z portalami społecznościowymi tj. Facebook.

Scenariusz 6: Sprawdzenie obsługi cache

Ten eksperyment miał na celu sprawdzenie czy w danej bibliotece został zaimplementowany mechanizm do przechowywania danych w celu eliminacji pobrania ich ponownie bądź wprowadzenie przez użytkownika.

Scenariusz 7: Dostęp do bazy Firebase

To zadanie miało na celu sprawdzenie czasów operacji typu odczyt, zapis i usunięcie danych z bazy Firebase. Są to proste dane generowane przez określoną liczbę razy. Kolejno 10, 100, 1 tys., 2 tys. i 3 tysiące.

Scenariusz 8: Dostęp do bazy typu SQLite

W tym przypadku podobnie jak w scenariuszu 7 zbadane zostały czasy operacji na bazie przeglądarkowej typu SQLite.

Scenariusz 9: Szybkość współpracy z innymi frameworkami – Google Charts

Ten scenariusz pokazuje, która z analizowanych technologii jest najszybsza pod względem komunikacji z innymi bibliotekami. W tym przypadku jest to popularna biblioteka do generowania wykresów. Sprawdzono czas generowania liczby punktów do czasu jego wyświetlenia, a następnie sprawdzono w której technologii najszybciej się one wygenerują.

Scenariusz 10: Szybkość ładowania obrazków

Rozpatrzono jak technologie porównywane poradzą sobie z przetwarzaniem i wyświetlaniem na stronie dużej liczby obrazków.

Scenariusz 11: Popularność użycia

W tym zadaniu pokazana została liczba projektów wykonanych w danej technologii w serwisie repozytorium GitHub.

Tezę badawczą sformułowano następująco: *Ember.js jest szybszą biblioteką w porównaniu do Angular 2 pod względem renderującym stronę, najlepiej współpracującą z innymi bibliotekami i najbardziej bezpieczną*

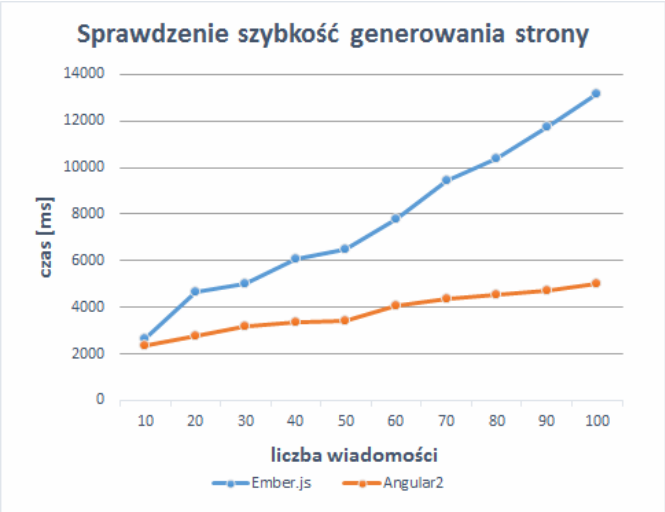
6. Wyniki badań

Do badań został wykorzystany laptop ASUS N551JM (Intel i7-4710HQ/16GB/250GB SSD/Win8 GTX860) oraz przeglądarka internetowa Chrome w wersji 62.0.3202.94 (64-bitowa).

W zależności od scenariusza została powtórzona określona liczba prób. Wyniki zostały uśrednione. W artykule przedstawiono 3 najistotniejsze scenariusze a wyniki wszystkich podano w podsumowaniu. W opisywanym przypadku rozpatrzono szybkość generowania strony, dostęp do bazy Firebase oraz szybkość współpracy z Google Charts.

6.1. Testy szybkości generowania strony

Pierwszym testem była szybkość wyświetlenia wiadomości, na którą składa się 100 wyrazów różnej długości, obrazek, film zewnętrzny i film wewnętrzny. Te informacje zwraca serwer REST, który działa lokalnie na komputerze. Czas ładowania strony jest zdefiniowany od momentu przeładowania do zakończenia wykonania wszelkich skryptów na stronie (Rys. 2).



Rys. 2. Wyniki testów szybkości aplikacji wyświetlania wiadomości strony.

Na podstawie rys. 2 można wyraźnie odczytać, że aplikacja napisana w Angular 2 o wiele szybciej wczytuje wiadomości ze zdjęciami i filmami niż Ember.js. Przyczynia się do tego sumaryczny czas, w którym czynniki składają się z różnych czasów: ładowania, wykonujących skrypt, renderowania, wyświetlania i bezczynności. W scenariuszu pełne ładowanie strony zajmuje 70% czasu wykonania skryptów. Jak dowodzi powyższe badanie, sumaryczne algorytmy wykorzystane w Angular 2 są szybsze.

Kolejnym testem jest szybkość wyświetlania dużej liczby obrazków (Rys. 3).

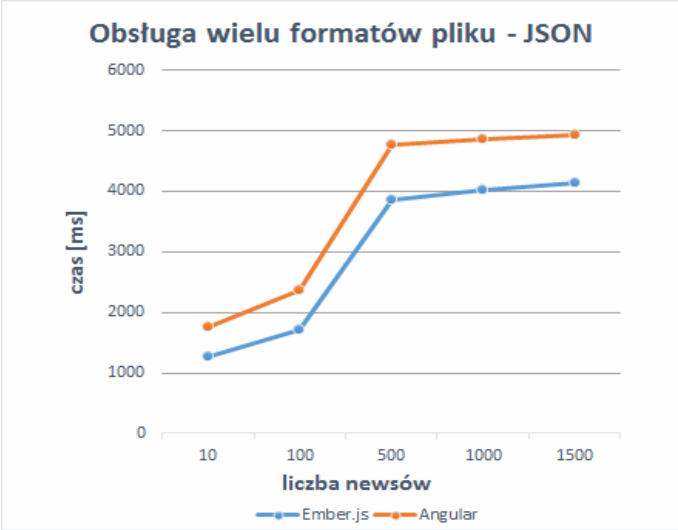
Z przeprowadzonych badań wynika, że w przypadku Ember.js informacje bajtowe są szybciej przetwarzane i wyświetlane. Przewaga ta zaczyna się w momencie, gdy wyświetlanych jest więcej jak 100 obrazków średniej wielkości tzn. 500px na szerokość i wysokość.

Rys. 3. Wyniki testów szybkości ładowania obrazków.

6.2. Bezpieczeństwo aplikacji internetowych

Pod względem zabezpieczeń obie technologie reprezentują prawie wszystkie formy popularnych zabezpieczeń . Wyjątkiem jest szyfrowanie w Ember.js. Nie ma żadnych modułów, które mogłyby zabezpieczyć w prosty sposób dane. Angular 2 za to udostępnia wszystkie rodzaje szyfrowań jak również resztę zabezpieczeń tj. reCAPTCHa, JWT (JSON Web Token) LocalStorage (wewnętrzna pamięć przeglądarki), Ciasteczka w przeglądarce internetowej.

W następnej kolejności zbadano przetwarzanie tekstu, na ten czynnik składa się parsowanie tekstu z formatu JSON do postaci modelu DOM (Document Object Model) wyświetlanych na stronie. Wyniki przedstawiono na rys. 4.



Rys. 4. Wyniki testów przetwarzania tekstu z formatu typu JSON.

Z rys. 4 wynika, że Ember.js przetwarza tekst szybciej niż Angular 2.

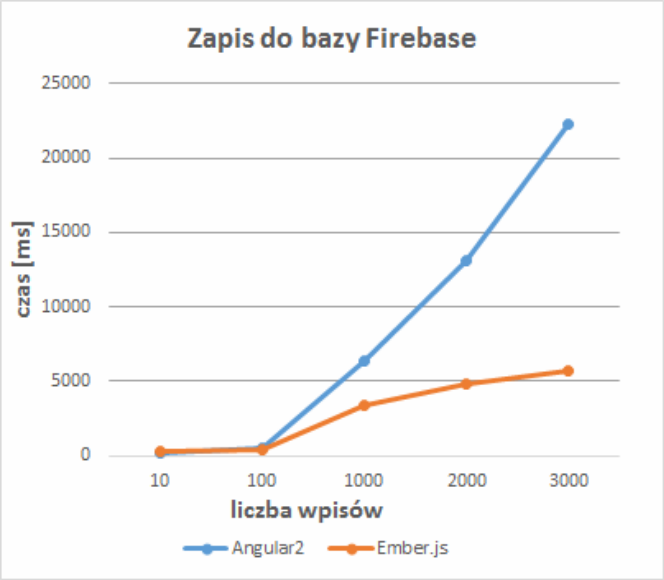
6.3. Wsparcie różnych formatów danych oraz jej szybkość parsowania.

Inne formaty danych tj. XML czy YAML to Ember.js nie umożliwiają ich wsparcia w swojej architekturze przetwarzania informacji. Cała architektura tej technologii została stworzona w taki sposób, że komunikacja odbywa się tylko i wyłącznie za pomocą notacji JSON. Angular 2 jest tutaj wyjątkiem, gdyż jego architektura pozwala tworzyć obejścia i parser, który może różną notację sprowadzać do postaci postawowego typu JSON.

Badając współpracę bibliotek z integracją z portalami społecznościowym sprawdzono najpopularniejszy na świecie Facebook. Obie porównywane technologie wspierają wewnętrzną aplikacje Facebook API. Wsparcie to można wykorzystać do wielu czynnosci: od pobierania podstawowych danych do zdjęć z galerii użytkownika. Oczywisty jest fakt, że użytkownik musi zaakceptować zgodę na wykorzystanie tych danych w aplikacji.

6.4. Dostęp do bazy danych

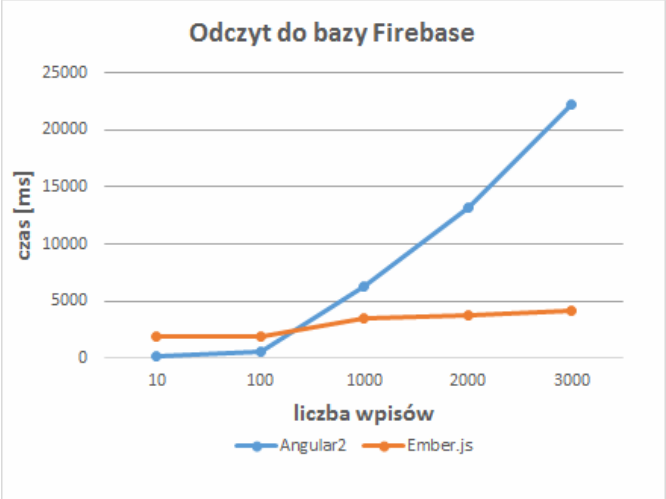
W kolejnym teście zbadano szybkość podstawowych operacji na zenętrznej bazie danych typu Firebase (Rys. 5, 6, 7). Do tej bazy ładuje się określoną ilość danych oraz mierzy czas jaki potrzebny jest do jej wykonania.



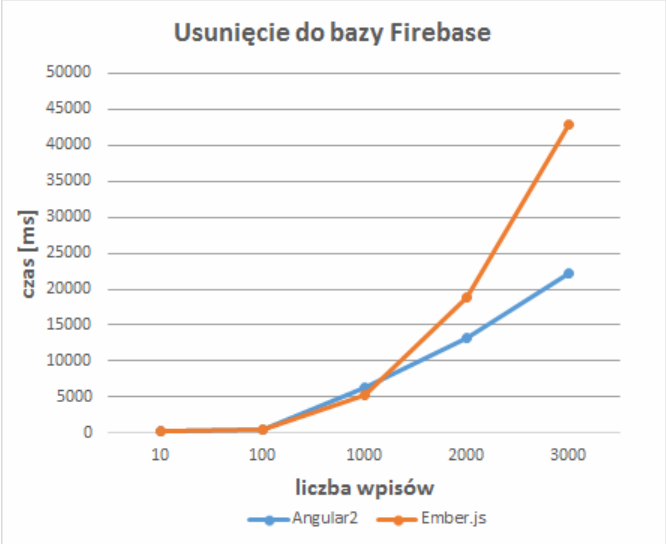
Rys. 5. Wyniki testów zapisu danych do bazy danych Firebase.

Z rysunków 5, 6 oraz 7 wynika, że zapis jak i odczyt w Ember.js jest znacznie szybszy niż w Angular 2. Z kolei, jeżeli chodzi o usunięcie danych, w Angular jest to znacznie szybsze. Wynika to z tego, że w Ember.js każdy rekord jest identyfikowany po konkretnym Id. Każdy z nich należy iterowac i czyścić. Natomiast w przypadku Angular 2 sprawa sprowadza się do przypisania pustej tablicy do bazy danych co powoduje jej wyczyszczenie.

Badając inny typ dostępu do bazy danych typu SQLite można dojść do wniosku, że Angular 2 nie wspiera bazy danych w przeglądarce. Można tylko symulować taki zbiór danych na emulatorach telefonów komórkowych typu Android [11]. W przypadku Ember.js otrzymać można wszelkie podstawowe wsparcie [12].



Rys. 6. Wyniki testów odczytu danych z bazy danych Firebase.



Rys. 7. Wyniki testów usunięcia danych z bazy danych Firebase.

6.5. Szybkość współpracy z innymi bibliotekami

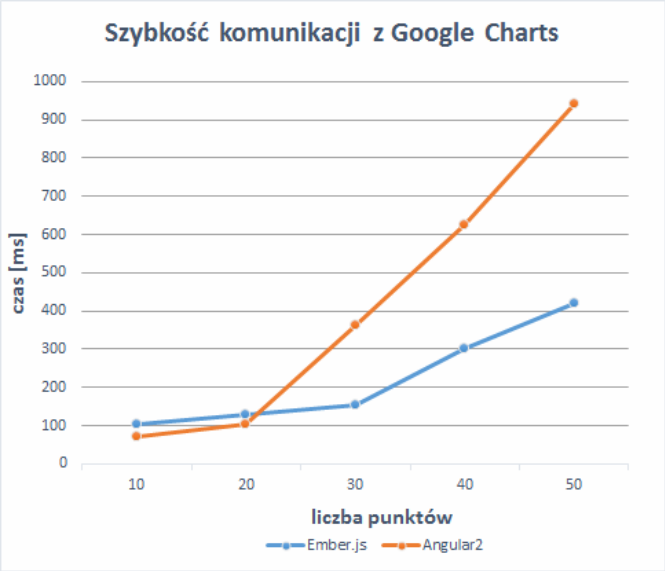
Kolejne badanie przedstawia szybkość komunikacji z inną biblioteką do generowania wykresów typu Google Charts. Wyniki przedstawia rysunek 8. Wyniki te pokazują sporą przewagę Ember.js nad Angular.

6.6. Popularność

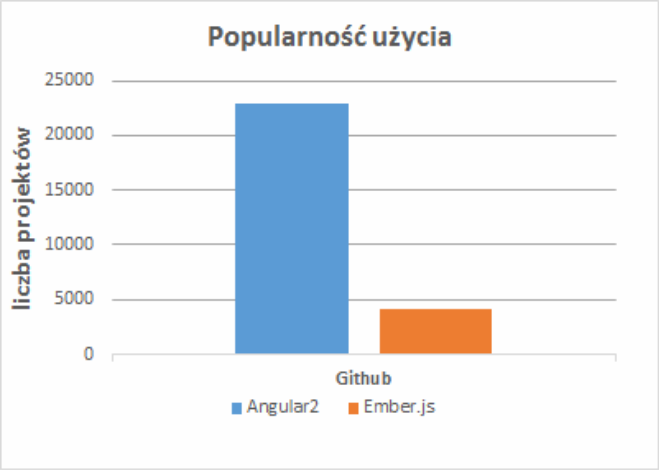
Ostatnim testem, któremu poddane zostały technologie jest ich popularność wśród programistów oraz popularnść w sieci.

Rys. 9 przedstwia aż 5-krotną przewagę Angular 2 nad Ember.js. Dotyczy to również stron typu stackoverflow, gdzie programiści proszą o pomoc i ją otrzymują, co nadaje takim

serwisom funkcję źródła wiedzy dla programistów. Ponadto z przerowadzonych badań wynika, że więcej tematów jak i rozwiązań można otrzymać z Angulara 2 niż z Ember.js.



Rys. 8. Wyniki szybkości komunikacji z Google Charts.



Rys. 9. Diagram kolumnowy ilości projektów w danych technologiach.

7. Wnioski

Zaimplementowano aplikację webową typu SPA (Single-Page-Application) Angular 2 i Ember.js. Funkcjonalność tych aplikacji służyła do porównania określonych funkcji w obu technologiach z wykorzystaniem specjalnie przygotowanych scenariuszy. Za pomocą deweloperskich ustawień w przeglądarce Chrome można z dużą dokładnością zmierzyć czas wykonania określonych operacji.

Podsumowując wszystkie scenariusze i zestawiając je według wag o różnym stopniu przydatności: 3 – istotne, 2 – średnio istotne, 1 – mało istotne.

Można za pomocą funkcji sum wagowych wyznaczyć lepszą technologię.

Tabela 1. Zestawienie metodą funkcji sum wagowych obu technologii.

Scenariusz	Ember.js	Angular 2	Waga	Ember.js z wagą	Angular2 z wagą
1	0	1	3	0	3
2	4	5	3	12	15
3	1	0	3	3	0
4	0	1	1	0	1
5	0	1	1	0	1
6	1	1	3	3	3
7	2	1	3	6	3
8	1	0	2	2	0
9	1	0	3	3	0
10	1	0	3	3	0
11	0	1	3	0	3
Suma				**32**	**29**

Przeprowadzone badania dowodzą, że Ember.js jest szybszą biblioteką od Angular 2, widać to szczególnie w komunikacji z Google Charts. Angular 2 jest bardziej bezpieczny pod względem szyfrowania wiadomości. Podsumowując w niektórych scenariuszach Angular 2 jest lepszą technologią od Ember.js. Różnica ta jest jednak niewielka. Pod względem innych czynników np. popularności czy wsparcia technicznego przez innych programistów większą przewagę posiada Angular 2. Natomiast pod względem funkcjonalnym i szybkości znacznie lepiej sprawdza się Ember.js. Wobec czego jest potwierdzeniem tezy badawczej, że Ember.js jest szybszym rozwiązaniem od konkurenta.

Literatura

[1] D. Magnusson, E. Grenmyr, An Investigation of Data Flow Patterns Impact on Maintainability When Implementing Additional Functionality, Linnaeus University, Faculty of Technology, Department of Computer Science, 2016.

[2] U. Shakya, Using a Framework to develop Client-Side App A Javascript Framework for cross-platform application, Helsinki Metropolia University of Applied Sciences, 6 Nov 2014.

[3] J. Koetsier, Evaluation of JavaScript frameworks for the development of a web-based user interface for Vampires, Informatica — Universiteit van Amsterdam, June 8, 2016.

[4] G. Kunz, Angular 2. Tworzenie interaktywnych aplikacji internetowych, Helion 2017.

[5] P. Deeleman, Learning Angular 2, Packt-Publishing 2016.

[6] M. Nayrolles, Angular 2 Design Patterns and Best Practices, Packt-Publishing 2017.

[7] M. Gechev, Switching to Angular 2, Packt-Publishing 2016.

[8] S. Shrestha, Ember.js front-end framework – SEO challenges and frameworks comparison, Haaga-Helia University of Applied Science 2015.

[9] M. Bodmer, Instant Ember.js Application Development How-to, Packt-Publishing 2013.

[10] J. Cravens, T. Q. Brady, Ember.js dla webdeveloperów, Helion 2015.

[11] www.github.com/litehelpers/Cordova-sqlite-storage [08.12.2017].

[12] www.pmjs.com/package/ember-sqlite-adapter [08.12.2017].

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 7 (2018) 210-214
Wysłane: 2018-01-11
Przyjęte: 2018-01-15

Efektywność sztucznych sieci neuronowych w rozpoznawaniu znaków pisma odręcznego

Marek Miłosz*, Janusz Gazda

Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Sztuczne sieci neuronowe są jednym z narzędzi współczesnych systemów odtwarzania z obrazów tekstów, w tym pisanych odręcznie. W artykule przedstawiono rezultaty eksperymentu obliczeniowego mającego na celu analizę jakości rozpoznawania cyfr pisanych odręcznie przez dwie sztuczne sieci neuronowe (SSN) o różnej architekturze i parametrach. Jako podstawowe kryterium jakości rozpoznawania znaków użyto wskaźnika poprawności. Poza tym analizie poddano liczbę neuronów i ich warstw oraz czas uczenia SSN. Do stworzenia SSN, oprogramowania algorytmów ich uczenia i testowania wykorzystano język Python i bibliotekę TensorFlow. Obydwie SSN uczono i testowano przy pomocy tych samych dużych zbiorów obrazów znaków pisanych odręcznie.

Słowa kluczowe: rozpoznawanie znaków; pismo odręczne; sztuczne sieci neuronowe

*Autor do korespondencji.

Adres e-mail: m.milosz@pollub.pl

Effectiveness of artificial neural networks in recognising handwriting characters

Marek Miłosz*, Janusz Gazda

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

Abstract. Artificial neural networks are one of the tools of modern text recognising systems from images, including handwritten ones. The article presents the results of a computational experiment aimed at analyzing the quality of recognition of handwritten digits by two artificial neural networks (ANNs) with different architecture and parameters. The correctness indicator was used as the basic criterion for the quality of character recognition. In addition, the number of neurons and their layers and the ANNs learning time were analyzed. The Python language and the TensorFlow library were used to create the ANNs, and software for their learning and testing. Both ANNs were learned and tested using the same big sets of images of handwritten characters.

Keywords: character recognition; handwriting; artificial neural networks

*Corresponding author.

E-mail address: m.milosz@pollub.pl

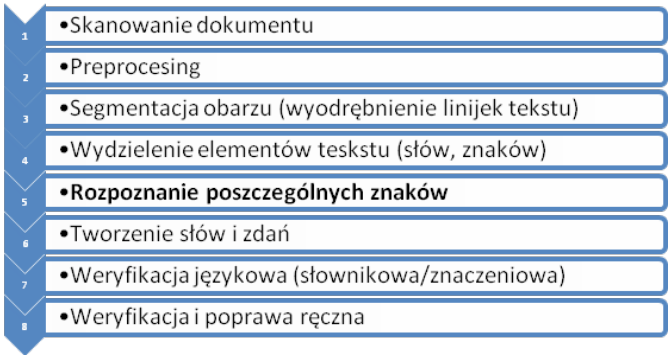
1. Wstęp

Optyczne rozpoznawanie znaków (ang. *Optical Character Recognition, OCR*) polega na przetworzeniu obrazu rastrowego tekstu w zakodowany cyfrowo tekst. Proces ten jest jednym z zadań cyfryzacji zbiorów dokumentów, w tym pisanych odręcznie – wówczas jest to inteligentne rozpoznawania tekstów (ang. *Intelligent Character Recognition, ICR*). Potrzeba takiego przekształcenia zeskanowanego automatycznie obrazu dokumentu w tekst występuje przede wszystkim w bibliotekach, w których zgromadzono olbrzymie zasoby zbiorów drukowanych i tworzonych odręcznie, oraz w muzeach. Oprócz tego potrzeby takie występują w obszarze biznesu i e-państwa. Systemy OCR są nieodzownymi elementami aplikacji biznesowych obiegu dokumentów, w których jeszcze przez jakiś czas będą pojawiać się dokumenty elektronicznie lub ręcznie tworzone/wypełniane, wydrukowane, podpisane i skanowane. W miarę cyfryzacji gospodarki wolumen dokumentów skanowanych, a szczególnie tych wypełnianych odręcznie, będzie się zmniejszać [1]. Tym niemniej w chwili obecnej w niektórych obszarach e-gospodarki i e-państwa pewna część dokumentów jest wypełniana odręcznie. W związku z czym ICR stanowią się nieodzownymi i alternatywnymi do ręcznego wprowadzania treści elementami systemów klasy ERP (ang. *Enterprise Resource Planning*), G2C (ang. *Government-to-Citizen*) czy też RPA (ang. *Robotic Process Automation*) [2].

2. Typowy schemat rozpoznawania pisma odręcznego

Systemy OCR (a także ICR) realizują cały szereg algorytmów i programów informatycznych w tym (rys. 1):

- skanowanie dokumentów,
- obróbka plików graficznych rastrowych pozyskanych w procesie skanowania i modyfikacji,
- analiza struktury zawartości plików rastrowych,
- rozpoznawanie pojedynczych znaków,
- tworzenie słów (jednostek leksykalnych) i zdań,
- weryfikacja leksykalna,
- weryfikacja ręczna poprawności tekstu i jego poprawa.



Rys. 1. Schemat rozpoznawania pisma odręcznego – stos metod i algorytmów (opracowanie własne na podstawie [3])

Część z tych algorytmów jest dość znana. Przykładowo są to algorytmy segmentacji obrazów [4] lub weryfikacji słownikowej rozpoznanego słowa.

Jednym z istotnym stosie metod, stosowanych w procesie OCR/ICR, jest algorytm rozpoznawania poszczególnych znaków (punkt 5 na rys. 1).

3. Metody rozpoznawania pojedynczych znaków pisma

Algorytmy rozpoznawania (klasyfikacji) poszczególnych znaków przekształcają dane rastrowe na tekstowe. Bazują one na następujących klasyfikatorach [5]:

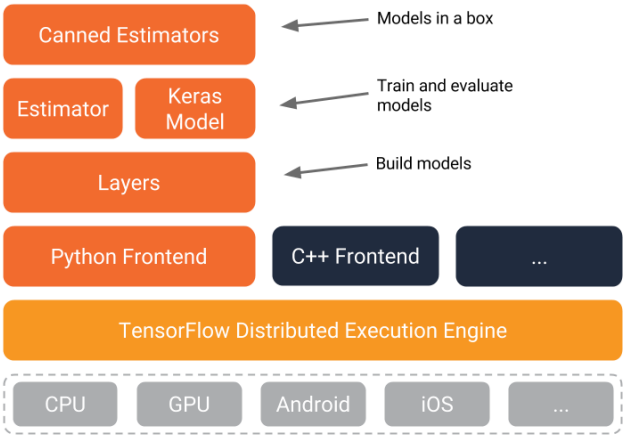
- k-NN – k-najbliższych sasiadów – algorytm centroidów,
- klasyfikator bayesowki,
- NN (ang. *Neural Network*) – sieci neuronowe,
- HMM (ang. *Hidden Markov Model*) – ukryte modele Markowa,
- SVM (ang. *Suport Vector Machine*) – maszyna wektorów wspierających (nośnych).

Wszystkie te metody bazują na podawaniu na wejściu wektora pozyskanego jako rezultat analizy obrazu (etap 4. z rys. 1), np. wartości pixeli obrazu zawierającego znak, i określeniu do której grupy klasyfikacyjnej (tj. do jakiego znaku) „pasuje" dany wektor. Podane powyżej klasyfikatory określają miarę „dopasowania" znaku do grupy, tj. klasyfikację znaku. Proces ten, wykorzystujący nauczoną SSN, przedstawiony został na rys. 2.

Rys. 2. Schemat klasyfikacji znaku przez SSN

4. Architektura platformy TensorFlow

Istnieje cały szereg bibliotek wykorzystujących uczenie maszynowe, w tym SSN. Do najbardziej znanych należą: TensorFlow, Theano, Keras i Scikit-learn [6]. Biblioteka TensorFlow została opracowana w laboratoriach firmy Google, jako biblioteka typu open source, i jest przeznaczona do tworzenia programów wykorzystujących uczenie maszynowe oraz do badania SSN [7]. TensorFlow obecnie jest całą platformą o wielopoziomowej strukturze (rys. 3).



Rys. 3. Wielopoziomowa struktura TensorFlow [7]

TensorFlow wykorzystuje w zasadzie dowolne, dostępne zasoby obliczeniowe, od pojedynczego mikroprocesora (CPU) czy też karty graficznej (GPU), poprzez ich klaster, aż po chmurę obliczeniową, a nawet platformy urządzeń mobilnych [7]. TensorFlow posiada API do typowych języków programowania, od C++, poprzez Python do Javy i Java Script.

TensorFlow jest od paru lat wykorzystywany do rozwiązywania wielu zadań sztucznej inteligencji, takich jak analiza wyrazu twarzy [8] i zachowań ludzkich [9], klasyfikacja danych [10] i obiektów [11] oraz prognozowanie przyszłości [12].

5. Analiza efektywności rozpoznawania przez sieci o różnej architekturze i parametrach

5.1. Opis problemu

Celem badania jest analiza wpływu architektury SNN, nauczonej rozpoznawania znaków (cyfr) pisanych odręcznie, na proces (czas) uczenia sieci oraz na jakość rozpoznawania przy różnych liczbach neuronów w warstwach.

Można sformułować następujące pytanie badawcze:

Czy dodanie drugiej warstwy ukrytej w SSN zwiększy jakość rozpoznawania znaków pisma odręcznego?

By dać odpowiedź na pytanie badawcze należy rozpatrzyć dwie SSN bez sprzężenia zwrotnego (ang. *Feedforward*) różniące się liczbą warstw ukrytych: jedna i dwie. Sieci, proces uczenia oraz testowania implementuje się w języku Python z wykorzystaniem biblioteki TensorFlow. Wszystkie dwa typy sieci należy uczyć i testować przy pomocy tych samych zbiorów znaków. Implementacja eksperymentu obliczeniowego powinna być zrealizowana w identycznym środowisku sprzętowo-programistycznym. Należy także

modyfikować liczbę neuronów w poszczególnych warstwach ukrytych kolejnych eksperymentach obliczeniowych.

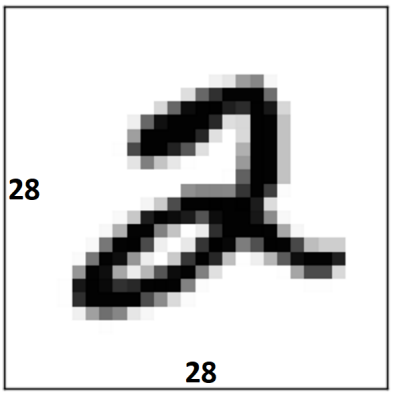
W rezultacie eksperymentu określony będzie wskaźnik poprawności nauczonej sieci (mierzony w % poprawnych rozpoznań znaków w zbiorze testowym) i czas uczenia sieci.

W związku z różną architekturą SSN do porównania obu wariantów wykorzystane zostanie porównanie łącznej (sumarycznej) liczby neuronów we wszystkich warstwach SSN i poprawności rozpoznawania znaków. Porównanie to zostanie dokonane przy wartościach wskaźnika poprawności 90% i 94%.

5.2. Zbiór danych uczących i testujących

W eksperymencie wykorzystano ogólnie dostępny zbiór obrazów cyfr pisanych odręcznie [13]. Zbiór ten składa się z dwóch podzbiorów: uczącego (60 tys. znaków) i testowego (10 tys. znaków). Cyfry pisane odręcznie są umieszczone w postaci obrazów o stałej wielkości i wyśrodkowane w prostokącie obrazu. Oprócz tego na zbiór składają się dwa pliki zawierające etykiety (tekstowe) każdego obrazu.

Obrazy cyfr były poddane operacji czyszczenia z szumów oraz poddane procesowi normalizacji, poprzez zmianę wielkości w taki sposób by obraz cyfry zmieścił się z kwadracie 20x20 punktów z zachowaniem proporcji [13]. Następnie kwadraty te umieszczono centralnie w obrazy o rozdzielczości 28x28 punktów. Cyfry odwzorowane są w szarej jedno-bajtowej skali kolorów (w zakresie od 0 do 255) – rys. 4.



Rys. 4. Przykład obrazu cyfry z bazy MNIST [13]

5.3. Plan eksperymentu

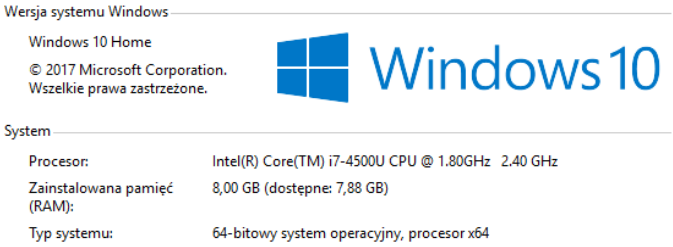
Obie sieci neuronowe trenowane będą przy pomocy tego samego zbioru uczącego, by następnie sprawdzić jakość rozpoznawania cyfr pisanych odręcznie przy pomocy zbioru testującego [13].

Przy pomocy biblioteki TensorFlow będą implementowane SSN określonej architektury i parametrach (liczbie neuronów w warstwach). Obie sieci będą miały 784 wejścia (28x28) i 10 wyjść, odpowiadających każdej z 10 cyfr. Warstwa wejściowa i wyjściowa będą miały dodatkowe wejście zwane bias, czyli zerowe. Wszystkie neurony będą miały funkcję aktywacji: max(0,z). Początkowe wartości wag wszystkich warstw będą generowane losowo i modyfikowane w trakcie uczenia sieci. Ze względu na ograniczenie pamięci RAM proces uczenia będzie podzielony na 10 kroków po 6000 znaków każdy.

Oprócz implementacji SSN należy zaimplementować proces ich uczenia i testowania. Wykorzystuje się do tego odpowiednie procedury biblioteki TensorFlow.

5.4. Realizacja eksperymentu

Eksperyment przeprowadzony został na komputerze o parametrach jak na rys. 5. W trakcie badania nie były uruchamiane żadne dodatkowe procesy, a procesor był wykorzystywany w 100% przez program uczenia sieci.



Rys. 5. Parametry systemu użytego do realizacji eksperymentu [6]

Na rys. 6 i 7 przedstawiono fragmenty programów realizujących sieci poszczególne. Przykładowy program uczenia sieci jednowarstwowej przedstawiony został na rys. 8. Trenowanie obydwu sieci odbyło się przy pomocy analogicznego algorytmu.

```
15  def neural_network_model(data):
16
17      hidden_layer1 = {'weights': tf.Variable(tf.random_normal([784, n_nodes_hl1])),
18                       'biases': tf.Variable(tf.random_normal([n_nodes_hl1]))}
19
20      output_layer = {'weights': tf.Variable(tf.random_normal([n_nodes_hl1, n_classes])),
21                      'biases': tf.Variable(tf.random_normal([n_classes]))}
22
23      l1 = tf.add(tf.matmul(data, hidden_layer1['weights']), hidden_layer1['biases'])
24      l1 = tf.nn.relu(l1)
25
26      output = tf.add(tf.matmul(l1, output_layer['weights']), output_layer['biases'])
27      return output
```

Rys. 6. Definicja sieci z jedną warstwą ukrytą [6]

```
19  def neural_network_model(data):
20      # input_data * weights + biases
21      hidden_l1 = {'weights': tf.Variable(tf.random_normal([784, n_nodes_hl1])),
22                   'biases': tf.Variable(tf.random_normal([n_nodes_hl1]))}
23
24      hidden_l2 = {'weights': tf.Variable(tf.random_normal([n_nodes_hl1, n_nodes_hl2])),
25                   'biases': tf.Variable(tf.random_normal([n_nodes_hl2]))}
26
27      output_l = {'weights': tf.Variable(tf.random_normal([n_nodes_hl2, n_classes])),
28                  'biases': tf.Variable(tf.random_normal([n_classes]))}
29
30      l1 = tf.add(tf.matmul(data, hidden_l1['weights']), hidden_l1['biases'])
31      l1 = tf.nn.relu(l1)
32
33      l2 = tf.add(tf.matmul(l1, hidden_l2['weights']), hidden_l2['biases'])
34      l2 = tf.nn.relu(l2)
35
36      output = tf.add(tf.matmul(l2, output_l['weights']), output_l['biases'])
37      return output
```

Rys. 7. Definicja sieci z dwoma warstwami ukrytymi [6]

```python
def train_neural_network(x):
    start_time = time.time()
    prediction = neural_network_model(x)
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=prediction, labels=y))
    # optimizer value = 0.001, Adam similar to SGD
    optimizer = tf.train.AdamOptimizer().minimize(cost)
    epochs_no = 10

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        # training
        for epoch in range(epochs_no):
            epoch_loss = 0
            for _ in range(int(mnist.train.num_examples / batch_size)):
                epoch_x, epoch_y = mnist.train.next_batch(batch_size)
                _, c = sess.run([optimizer, cost], feed_dict={x: epoch_x, y: epoch_y})
                # code that optimizes the weights & biases
                epoch_loss += c
            print('Epoch', epoch, 'completed out of', epochs_no, 'loss:', epoch_loss)

        # testing
        correct = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))
        accuracy = tf.reduce_mean(tf.cast(correct, 'float'))
        print('Accuracy:', accuracy.eval({x: mnist.test.images, y: mnist.test.labels}))
    print("--- %s seconds ---" % (time.time() - start_time))
```

Rys. 8. Program uczenia sieci z jedną warstwą ukrytą [6]

6. Rezultaty eksperymentu

W tab. 1 i 2 przedstawiono rezultaty eksperymentu dla SSN z jedną i dwoma warstwami ukrytymi.

Tabela 1. Analiza poprawności rozpoznawania znaków i czasu uczenia sieci z jedną warstwą ukrytą [6]

Liczba neuronów w warstwie ukrytej	Poprawność (%)	Czas uczenia (s)
10	66,98	6,92
50	89,06	10,93
100	92,37	16,32
200	93,63	25,45
300	94,08	34,61
400	94,36	45,85
500	94,15	56,38
600	94,64	66,62
700	94,89	75,03
800	94,84	84,82

By odpowiedzieć na pytanie badawcze dokonano przekształcenia rezultatów badań poprzez zsumowanie liczby neuronów w sieci o dwóch warstwach ukrytych i uporządkowanie tych rezultatów wzrastająco według współczynnika poprawności rozpoznawania znaków pisma odręcznego.

Tab. 3 zawiera rezultaty porównania efektywności obydwu SSN (tj z jedną i dwoma warstwami ukrytymi) w rozpoznawaniu znaków pisma odręcznego.

Tabela 2. Analiza poprawności rozpoznawania znaków i czasu uczenia sieci z dwoma warstwami ukrytymi [6]

Liczba neuronów w pierwszej warstwie ukrytej	Liczba neuronów w drugiej warstwie ukrytej	Poprawność (%)	Czas uczenia (s)
10	10	47,16	7,47
10	50	43,84	8,15
10	100	45,49	8,51
10	200	32,16	9,83
10	300	43,3	11,01
10	400	36,89	11,77
10	500	33,36	12,64
50	10	40,29	11,34
50	50	85,9	12,13
50	100	89,6	12,97
50	200	88,18	14,94
50	300	89,18	15,86
100	10	28,46	15,63
100	50	78,8	17,3
100	100	91,82	18,08
100	200	92,14	21,1
100	300	92,2	23,71
100	400	93,12	23,46
100	500	92,33	26,23
200	10	33,84	26,01
200	50	79,16	28,09
200	100	92,35	29,5
200	200	93,38	32,15
200	300	93,53	36,37
200	400	93,58	37,71
200	500	93,86	42,08
300	10	14,21	36,2
300	50	74,12	37,32
300	100	93,54	39,5
300	200	94,21	45,11
300	300	94,06	48,09
400	10	35,46	46,93
400	50	92,33	48,73
400	100	93,66	50,7
400	200	94,21	57,33
400	300	94,6	60,18
400	400	94,7	66,54
400	500	94,96	71,23
500	10	17,8	57,51
500	50	68,69	59,22
500	100	93,21	63,37
500	200	94,82	68,17
500	300	94,86	73,57
500	400	94,81	80,93
500	500	94,89	87,45

Tabela 3. Porównanie efektywności rozpoznawania znaków przez SSN

Liczba warstw ukrytych	Poprawność, %	Łączna liczba neuronów	Czas uczenia, [s]
1	>90%	100	16
	>94%	300	34
2	>90%	200	18
	>94%	600	48

7. Podsumowanie

Rezultaty eksperymentu wskazują, że w warunkach normalizacji obrazów znaków pisanych do niezbyt dużego obszaru graficznego (20x20 punktów w 256 poziomach szarości), najlepszą SSN jest sieć o jednej warstwie ukrytej (identycznie nauczona jak sieć z dwoma warstwami). Liczba neuronów w takiej sieci jest dwa razy mniejsza w porównaniu z siecią z dwoma warstwami (tab. 3). Podobnie czas uczenia sieci z jedną warstwą ukrytą o analogicznej jakości w porównaniu z siecią o dwóch warstwach jest krótszy (tab. 3).

Odpowiedź na pytanie badawcze jest zatem **negatywna** – dodatkowa warstwa dla odpowiednio dobranej liczby neuronów SSN jednowarstwowej nie zwiększa efektywności rozpoznawania znaków pisma odręcznego.

Już przy niewielkiej liczbie neuronów SSN o jednej warstwie ukrytej ma bardzo dobrą jakość rozpoznawania cyfr pisanych odręcznie. Poprawność powyżej 90% osiągana jest już przy niecałych 100 neuronach. Jest to niewątpliwie rezultat normalizacji obrazów graficznych cyfr i dużego zbioru uczącego, a także małej liczbie wariantów rezultatu (10 cyfr).

Rezultat przedstawionego eksperymentu i wnioski z niego płynące może być wykorzystany w procesie doboru algorytmów (w tym przypadku: normalizacji znaku i struktury SSN) w procesie tworzenia systemu rozpoznawania pisma odręcznego.

Literatura

[1] ICR – czy warto skanować pismo odręczne?, http://ocrwdokumentach.pl/icr-rozpoznawanie-pisma-odrecznego/ [11.01.2018]

[2] S. Anagnoste, Robotic Automation Process - The next major revolution in terms of back office operations improvement, Proceedings of The International Conference on Business Excellence, vol 11(1) (2017), 676-686.

[3] W. Kacalak, M. Majewski, A New Method for Handwriting Recognition Using Artificial Neural Networks. Intelligent Engineering Systems Through Artificial Neural Networks, 16 (2006), 459-465.

[4] J. Smołka, M. Skublewska-Paszkowska, E. Łukasik, Algorithm for selecting optimal clustering parameters used for over-segmentation reduction. PRZEGLAD ELEKTROTECHNICZNY, 9, (2016), 250-256.

[5] Z. Gomółka, B. Twaróg, E. Żesławska, Rozpoznawanie pisma odręcznego za pomocą sztucznych sieci neuronowych, Technical News, (2013), 98-102.

[6] J. Gazda, Zastosowanie sztucznych sieci neuronowych do rozpoznawania tekstu. Praca dyplomowa pod kierunkiem M.Miłosza, Lublin, (2018), 42.

[7] What is the TensorFlow machine intelligence platform? https://opensource.com/article/17/11/intro-tensorflow [11.01.2018]

[8] HaoBiao, Dae-Seong Kang, The Research of Face Expression Recognition based on CNN using Tensorflow. Journal of Advanced Information Technology and Convergence, 7, (2017), 55-63.

[9] A. Ignatov, Real-time human activity recognition from accelerometer data using Convolutional Neural Networks. Applied Soft Computing, 62, (2018), 915-922.

[10] F. Ertam, G. Aydin, Data classification with deep learning using Tensorflow. 2017 International Conference on Computer Science and Engineering (UBMK), (2017), 755-758.

[11] N. Gavai, Y. Jakhade, S. Tribhuvan, R. Bhattad, MobileNets for flower classification using TensorFlow. 2017 International Conference on Big Data, IoT and Data Science (BID) Big Data, IoT and Data Science, (2017), 154-158.

[12] J. Evermann, J. R. Rehse, P. Fettke, XES tensorflow - Process prediction using the tensorflow deep-learning framework. Proceedings of the 29th International Conference on Advanced Information Systems Engineering, 1848, (2017), 41-48.

[13] The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/ [11.01.2018]