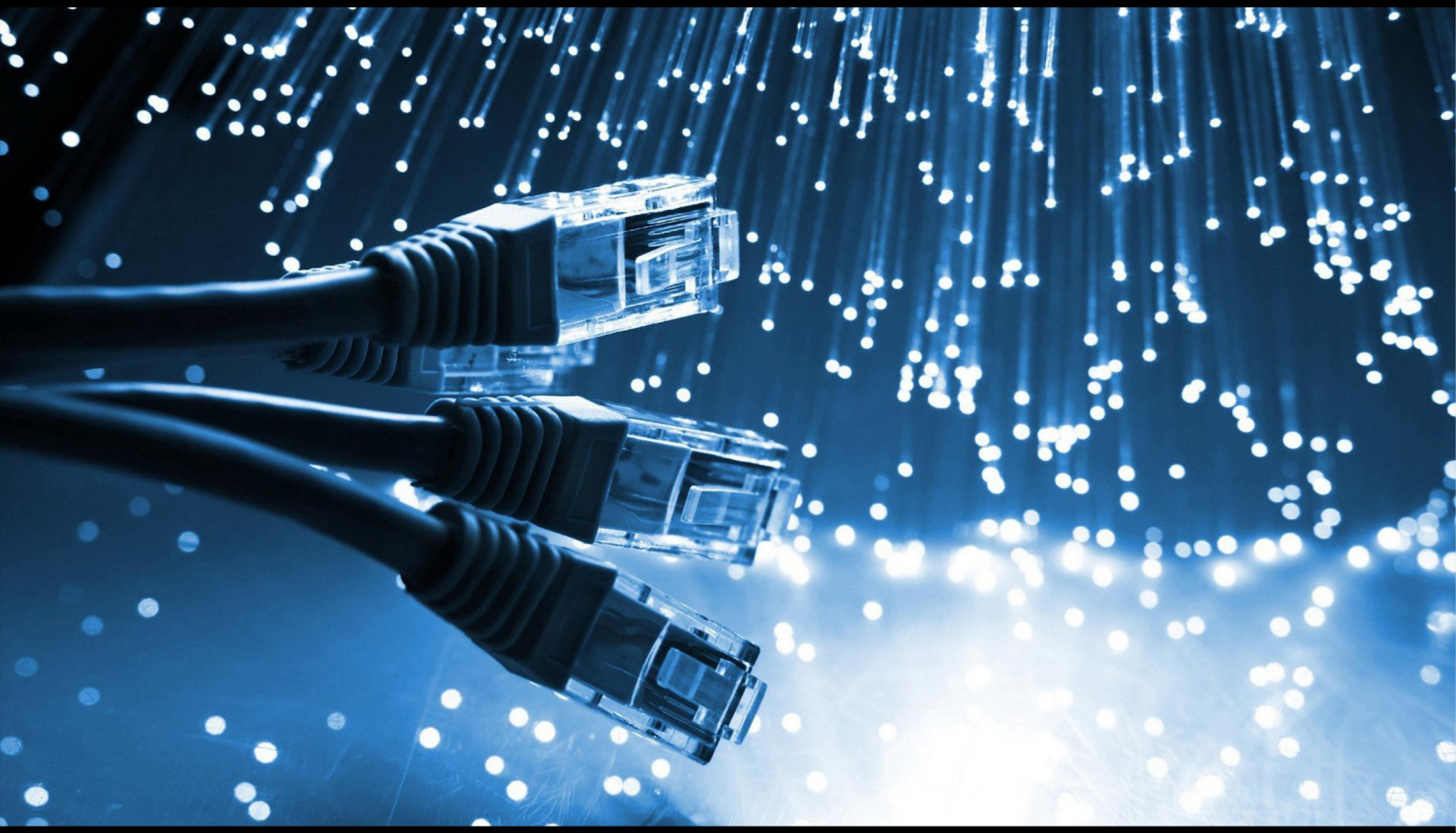
JCSI

Journal of Computer Sciences Institute

Volume 9/2018



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 inż. Dariusz Gutek
dr inż. Grzegorz Kozieł
dr inż. Maciej Pańczyk
dr inż. Jakub Smołka
dr inż. Elżbieta Miłosz
dr inż. Maria Skublewska-Paszkowska
dr Beata Pańczyk
dr inż. Sławomir Przyłucki

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:
Dariusz Gutek
Grzegorz Kozieł
Maciej Pańczyk
Jakub Smołka
Elżbieta Miłosz
Maria Skublewska-Paszkowska
Beata Pańczyk
Sławomir Przyłucki

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 9 (2018) 302-307
Wysłane: 2018-06-22
Przyjęte: 2018-06-27

Ocena zabezpieczeń wybranej platformy mobilnej

Aleksandra Iwaniuk*

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

Streszczenie. Smartfony sukcesywnie zyskiwały na popularności, by w efekcie praktycznie zupełnie wyprzeć tradycyjne telefony. Stwarzają one wiele możliwości, jednak stają się też źródłem zagrożeń. W artykule poddano analizie zabezpieczenia systemu Android. Sprawdzono jakie zagrożenia czyhają na urządzenia mobilne a także jak system Android stara się przed nimi uchronić. Badaniem ankietowym sprawdzono, jak blokowane są ekrany telefonów a także jak często dokonywane są zmiany w blokadzie. Aby sprawdzić, czy blokadę ekranu można zdjąć przeprowadzono testy przy użyciu Google znajdź urządzenie, a także Dr. Fone. Przedstawiono zestawienie wyników a następnie wyciągnięto wnioski.

Słowa kluczowe: Android; bezpieczeństwo; testowanie zabezpieczeń

*Autor do korespondencji.

Adres e-mail: aleksandra.kowaluk3@gmail.com

Security assessment of the selected mobile platform

Aleksandra Iwaniuk*

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

Abstract. Smartphones were gradually gaining popularity, so as to effectively replace traditional telephones. Devices creates many capabilities, but they can be a source of threats. In this article analyzed Android system security. Author checked what threats lurk for mobile devices and how the Android system tries to protect them. The questionnaire survey checked how the phone screens are blocked and how often changes are made to the blockade. To check if the screen lock can be removed, tests were performed using Google to find the device, as well as Fone. A summary of results was presented and conclusions were drawn.

Keywords: Android; security; security testing

*Corresponding author.

E-mail address: aleksandra.kowaluk3@gmail.com

1. Wstęp

Historia Androida rozpoczyna się w momencie założenia firmy Android Inc.,w październiku 2003 r. przez Andyego Rubina, Chrisa Whitea, Nicka Searsa i Richa Micera. W wyniku powstałych trudności w 2005 roku została wykupiona przez Google. W 2007 roku powstało konsorcjum OHA zrzeszające twórców oprogramowania i sprzętu. Dzięki systematycznemu rozwojowi systemów mobilnych Android w 2016 posiadał 64% udziałów w rynku [8].

W związku z dużą ilością użytkowników systemu Android doczekał się wielu wirusów i szkodliwych aplikacji. Zagrożenia te mogą mieć związek z użytkownikiem a także z lukami w zabezpieczeniach [1].

Niniejszy artykuł ma na celu omówienie zabezpieczeń, które chronią system Android, a także zagrożenia czyhające na urządzenia mobilne.

Przyjęto tezę: *Wbrew powszechnie akceptowanemu poglądowi dane użytkownika są bezpieczne.* Poddano również analizie świadomość użytkowników smartfonów a także wyniki odblokowania każdej z metod odblokowania ekranu, przeanalizowano czasy osiągnięte przez każdą z grup. Na tej podstawie wybrano najprostszą metodę. Sprawdzono też, czy jest możliwe zdjęcie blokady z ekranu smartfonu.

2. Rodzaje ataków

Z uwagi na objęcie pozycji lidera w rynku światowym, na urządzenia z systemem Android czyha wiele zagrożeń, działających na różnych założeniach.

2.1. Nieautoryzowany dostęp do danych

Urządzenia mobilne charakteryzuje to, że są praktycznie cały czas przy użytkowniku, stwarza to zagrożenie jego utraty. W momencie utraty urządzenia, jeśli brak skutecznej blokady ekranu może dojść do nieautoryzowanego dostępu do danych [3].

2.2. Ataki na aplikacje

Aplikacje mogą zostać zaatakowane na dwa różne sposoby. Jeden z nich jest statyczny a drugi dynamiczny. Metoda statyczna polega na dokonaniu zmian w aplikacji, konkretnie w plikach dex lub apk. Metoda dynamiczna polega na dokonaniu zmian w trakcie działania aplikacji [4].

2.3. Programy szpiegujące

Programy szpiegujące są to aplikacje zbierające lub kradnące dane użytkownika. Najczęściej źródłem ataku na aplikacje jest adware (nękająca użytkownika reklama), po wejściu w link w tle odbywa się instalacja. Aplikacje te nie tylko naruszają bezpieczeństwo, ale mogą też udzielać dostępu innym aplikacjom a także zmniejszać zasoby urządzenia poprzez wysłanie danych do bazy [3].

2.4. Ataki phishingowe

Ataki phishingowe polegają na podszywaniu się pod zaufane lub nieszkodliwe firmy np. bank. Ataki te mają na celu pozyskanie od użytkownika wrażliwych danych np. hasła i PINu do konta bankowego. Może to osiągnąć na kilka sposobów: MMS z adresem URL do fałszywej strony banku, wysłanie adresu URL poprzez SMS a także atak poprzez połączenia głosowe [5].

2.5. Przelewanie

Przelewanie jest to atak pozwalający na uzyskanie od atakowanych środków pieniężnych a następnie przelanie środków na konto agresora. Może się to odbywać poprzez korzystanie z usług pay-per-use [6].

2.6. Ataki TOCTOU

Luka w zabezpieczeniach TOCTOU (Time of Check to Time of Use) istnieje w systemie Android głównie na zasadzie zmowy nazewnictwa. Uprawnienia są przechowywane w systemie jako ciąg znaków. Dlatego też dwa uprawnienia o takim samym łańcuchu są traktowane jako równoważne, nawet jeśli odnoszą się do niezwiązanych aplikacji. Aplikacja może tym sposobem uzyskać dostęp do zasobów, do których zasoby nie były jej przyznane [10].

2.7. Złośliwe aplikacje

Złośliwe aplikacje można pobrać wraz z aplikacją, którą użytkownik chce pobrać, lub poprzez odwiedzenie podejrzanych stron. Można wyróżnić wiele złośliwych aplikacji takich jak:

• Trojany – są to aplikacje mogące uzyskać kontrolę nad urządzeniem. Często wykazują użyteczne funkcjonalności, przy czym działania szkodliwe wykonywane są w tle. Można z ich pomocą uzyskać dostęp do wrażliwych danych a także zainstalować na urządzeniu inne szkodliwe aplikacje np. botnety.

• Botnety – określa się tym terminem grupę urządzeń zdalnie kontrolowanych i koordynowanych. Wykorzystuje on zaatakowane urządzenia np. do rozsyłania wiadomości SPAM.

• Rootkity – są to aplikacje uprawnione (w nieautoryzowany sposób) do osiągania bogatego zasobu uprawnień. Obecność aplikacji najczęściej jest maskowana przed użytkownikiem.

• Robaki – to samo-replikujące się szkodliwe aplikacje. Mogą one uzyskiwać dostęp do wrażliwych danych użytkownika [2].

3. Zabezpieczenia Androida

Wprowadzono liczne zabezpieczenia mające chronić użytkownika, jego dane i urządzenie.

3.1. Uprawnienia i selekcja uprawnień

Aby aplikacja działała poprawnie muszą jej nadane odpowiednie uprawnienia. Odbywa się to poprzez wpisanie odpowiednich uprawnień do pliku AndroidManifest.xml. Podstawowe aplikacje Android nie posiadają przypisanych domyślnie uprawnień, pozwalających na dostęp do zasobów. Przed instalacją aplikacji wyświetlana jest informacja o wszystkich wymaganych przez aplikację uprawnieniach [6].

Uprawnienia aplikacji podlegają separacji. System Android wymaga, aby każda aplikacja posiadała własny identyfikator użytkownika (UID) i identyfikator grupy (GID). Gwarantuje to, że aplikacje lub procesy nie mają uprawnień dostępu do innych aplikacji lub procesów.

Aby umożliwić separację każda z aplikacji działa w dedykowanym Sandboxie. Sandboxingowi przyświeca zasada, aby żadna aplikacja nie miała możliwości nadpisania i zmian w kodzie innych aplikacji [9].

3.2. Podpis aplikacji

Wszystkie paczki z aplikacjami instalowanymi na urządzeniach muszą posiadać stosowny certyfikat. Dzięki temu użytkownik może stwierdzić autentyczność aplikacji. Pomaga to ustalić prawdziwy związek między użytkownikiem a twórcą. Nie istnieje jednak żaden urząd przyznający certyfikaty. Są więc one generowane przez twórców oprogramowania. Oznacza to, że użytkownik musi się wykazać zaufaniem do twórcy oprogramowania [7].

3.3. Pochodzenie aplikacji

Aby rozpocząć dystrybucję aplikacji za pośrednictwem Google Play Store programista musi założyć konto o odpowiednich uprawnieniach i uiścić opłatę w wysokości 25$. Przed przesłaniem do Google Play aplikacja musi zostać cyfrowo podpisana. Podpisu może oczywiście dokonać twórca oprogramowania [11].

3.4. Ochrona danych

Każda aplikacja działa w odseparowanej części systemu Android. Izolacją systemów od siebie steruje jądro Linux systemu. Dodatkowo mechanizm bezpieczeństwa jest oparty o system uprawnień a także procesów, jakie każda aplikacja może wykonać. Aplikacje są identyfikowane po kluczu aplikacji. Dostęp do danych jest nadawany w czasie rzeczywistym. Bezpieczeństwo danych opiera się głównie na mechanizmie podpisu aplikacji. System, jak i aplikacje potrzebują podpisu do prawidłowego funkcjonowania [8].

3.5. Statyczna i dynamiczna analiza aplikacji

Dwoma podstawowymi praktykami jest statyczna i dynamiczna analiza aplikacji, mająca na celu wykrycie ataku. Analiza statyczna wykorzystuje dekompilację, odszyfrowanie, dopasowywanie wzorców, a także analizę wywołań systemowych. W trakcie analizy program nie jest wykonywany. Podstawową praktyką analizy jest filtrowanie binarne, w celu wykrycia złośliwego kodu, wykorzystując przy tym sygnatury programów.

Inną metodą wykrywania wirusów, jest dynamiczna analiza. Obejmuje ona uruchamianie systemu w kontrolowanym środowisku i obserwację jego zachowania. Zawarta w tym jest obserwacja wymiany plików, połączenia sieciowego, zachodzących procesów itp. Podstawowym sposobem na dynamiczne sprawdzenie oprogramowania jest sandboxing. Sandbox może być definiowany jako środowisko, w którym wykonywany jest proces [10].

4. Metoda badań ankietowych

Badanie ankietowe przeprowadzono w celu zbadania świadomości użytkowników smartfonów o zabezpieczeniach i bezpieczeństwie urządzeń mobilnych. Jednym z założeń, było sprawdzenie, jak użytkownicy blokują swoje urządzenia, czy dokonują zmian w sposobie blokady a także ustalenie najbardziej przyjaznej metody blokowania ekranu.

Zastosowano urządzenia: Xiaomi Redmi 3S Pro, Sony Xperia S a także tablet Sony Xperia Z3. Każde z urządzeń cechuje inna wielkość wyświetlacza, miało to na celu zbadanie, czy wielkość ekranu ma wpływ na wynik odblokowania ekranu.

Na każdym z urządzeń dokonano następujących testów:

- Odblokowanie urządzenia kodem PIN o długości 4 znaków.
- Odblokowanie urządzenia kodem PIN o długości 8 znaków.
- Odblokowanie urządzenia hasłem o długości 4 znaków.
- Odblokowanie urządzenia hasłem o długości 8 znaków.
- Odblokowanie urządzenia wzorem o długości 4 znaków.
- Odblokowanie urządzenia wzorem o długości 8 znaków.

Otrzymane wyniki poddano analizie w oparciu o wyniki uczestników badania. Na tej podstawie przeprowadzono stosowne analizy.

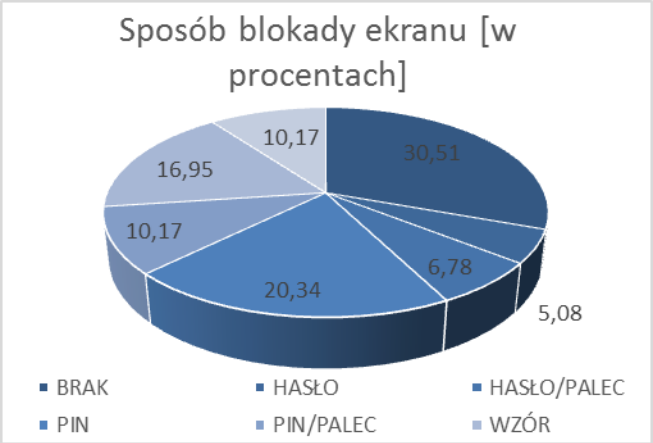
Badanie ankietowe przeprowadzono na grupie 66 osób. Badane osoby były w wieku 18-65 lat. Podzielono je na grupy wiekowe: do 20 lat, 20-30 lat, 30-40 lat, 40-50 lat, a także 50+. Podział miał na celu ułatwienie analizy danych. Badane osoby mieszkają zarówno w mieście jak i na wsi, nie dokonano jednak podziału ze względu na wielkość miasta (np. ze względu na liczbę mieszkańców). Osoby biorące udział w badaniu w większości nie były osobami związanymi z branżą IT.

5. Analiza badań ankietowych

Przeprowadzono analizę danych demograficznych, na ich podstawie zauważono, że większość wśród badanych stanowili mieszkańcy miast, wśród płci dominowali mężczyźni natomiast ze względu na wiek przeważały osoby w wieku 30-40 lat.

Rys. 1. Użycie smartfonu wśród ogółu ankietowanych.

Ankietowanych zapytano o użycie smartfonów (Rys. 1). Można zauważyć, że większość badanych (89,39%) na co dzień używa smartfonów. Nie stosowano podziału na system operacyjny stosowanego urządzenia. Jest to zbieżne z danymi na temat udziału w rynku telefonów [12]. Sprawdzono, jaki sposób blokady jest stosowany przez ankietowanych (Rys.2).



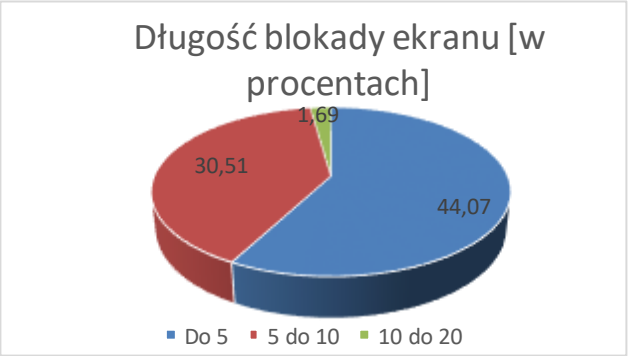
Rys. 2. Sposób blokady ekranu.

Przeanalizowano również deklarowany przez użytkowników sposób blokady ekranu. Niestety większość, bo 30,51% ogółu nie stosuje żadnej blokady ekranu (przeciągnięcie). Wśród powszechnie dostępnych metod blokady ekranu największą popularnością cieszył się kod PIN, na drugim miejscu wzór, najmniejszą zaś hasło. Można również zauważyć sporą popularność danych biometrycznych takich jak odcisk palca. Muszą one jednak być wspierane przez inną blokadę ekranu jak np. kod PIN.

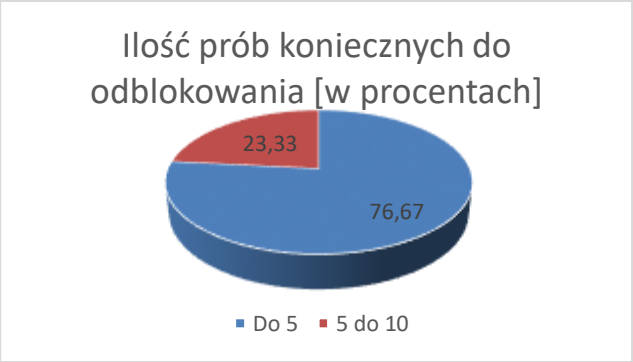
Innym istotnym elementem zabezpieczeń urządzeń mobilnych jest długość blokady ekranu (Rys.3).

Najbardziej popularną długością blokady ekranu jest blokada do 5 znaków. Mniejszą popularnością cieszyły się kody 5-10 znakowe. Najmniejszą grupę stanowiły blokady 10-20 znakowe. Ankietowanych zapytano, czy każda próba odblokowania urządzenia kończy się sukcesem. Ankietowanych którzy przyznali że nie zawsze udaje im się

odblokować urządzenie zapytano ile prób najczęściej potrzebują do odblokowania urządzenia (Rys. 4).

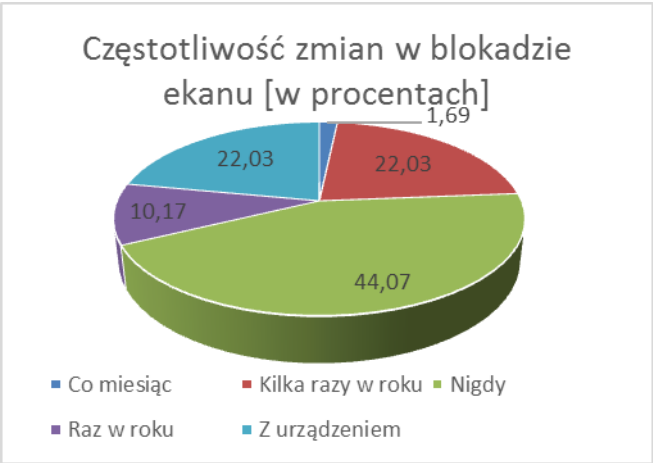


Rys. 3. Długość blokady ekranu.



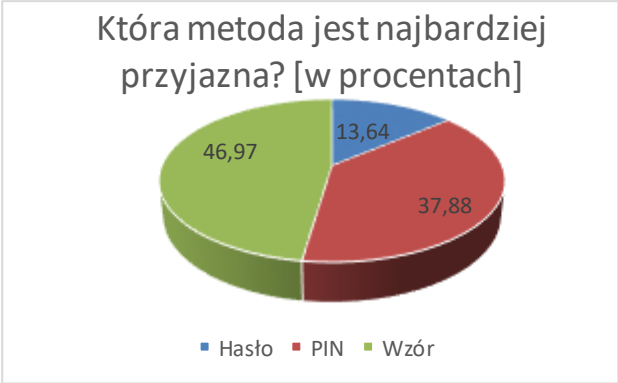
Rys. 4. Liczba prób potrzebnych do skutecznego oblokowania urządzenia.

Najczęściej ankietowani, którzy wymagali wielu prób najczęściej potrzebowali do 5 prób aby odblokować urządzenie. Mniejszą grupę stanowiły osoby potrzebujące 5 do 10 prób odblokowania. Nie wykazano w trakcie badania zapotrzebowania na większą liczę prób. Kolejnym pytaniem zadanym ankietowanym była częstotliwość zmian w blokadzie ekranu. Wliczano w to zmianę w obrębie sposobu np. zmiana z „1234" na „1235", jak również zmiany sposobu blokady np. z kodu PIN na wzór.



Rys. 5. Częstotliwość zmian w blokadzie ekranu.

Zauważyć można, że większość użytkowników nigdy nie dokonuje zmian w blokadzie ekranu. Uwzgledniono w tym wykresie zarówno osoby które stosują jedną z metod blokady ekranu a także osoby nie blokujące telefonu. Najpopularniejszą opcją wśród dokonujących zmian były zmiany wraz ze zmianą urządzenia a także zmiana kilka razy w roku. Zapytano uczestników ankiety o opinię na temat metody najbardziej przyjaznej użytkownikowi (Rys. 6).



Rys. 6. Opinia ankietowanych na temat najwygodniejszej metody blokowania ekranu.

Według ankietowanych metodą najbardziej przystępną użytkownikom jest wzór, najmniej zaś hasło. Opinia ta wyrażona została zarówno przez osoby stosujące na co dzień smartfony jak i użytkowników tradycyjnych urządzeń.

6. Metoda badań nad zdjęciem blokady

Przeprowadzono również badanie fizycznego urządzenia. Poddano analizie tablet Sony Xperia Z3 a także smartfon Xiaomi Redmi 3S Pro. Sukcesem metody będzie zdjęcie blokady bez utraty danych. Sprawdzone zostanie, czy konieczne jest tzw. Rebootowanie systemu, czy uda się to osiągnąć inną metodą. Przeanalizowanych zostanie kilka metod i oceniona zostanie ich skuteczność.

Ocenie zostanie poddana metoda:

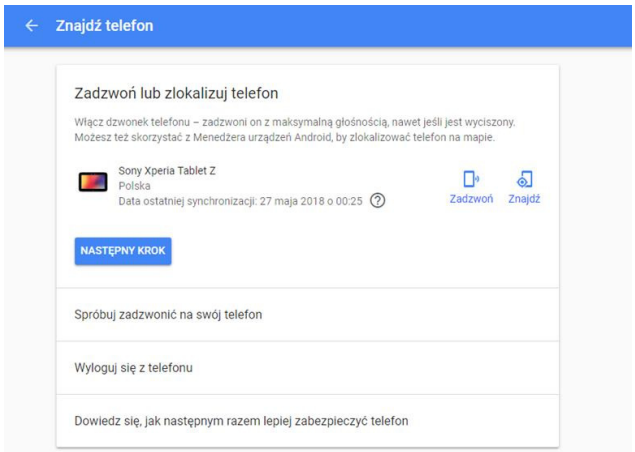
- Google znajdź urządzenie,
- Dr fone.

7. Badania nad zdjęciem blokady ekranu

7.1. Google znajdź urządzenie

Jest to opcja dostarczana przez Google, dzięki której można namierzyć swoje urządzenie, można też je zablokować a także wyczyścić zawartość pamięci. Wystarczy tylko aby urządzenie było włączone, użytkownik musi być zalogowany na koncie Google, musi mieć włączone Wi-fi lub transmisję danych. Można za pomocą tego narzędzia zmienić blokadę ekranu na nową.

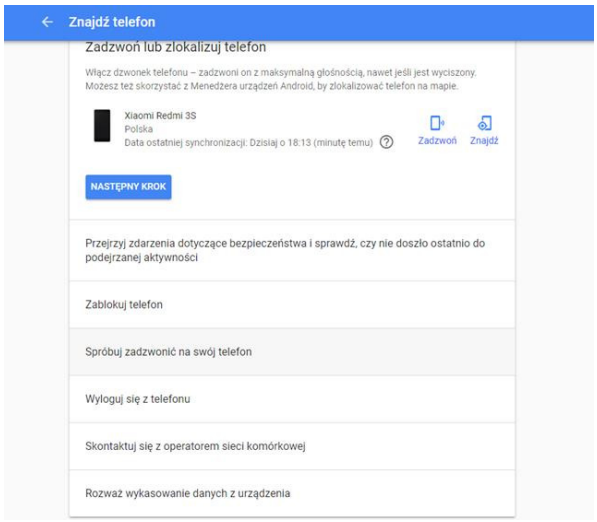
W przypadku tabletu Sony nie było możliwe dokonanie zmiany hasła, z uwagi na brak stosownej opcji w menu narzędzia (Rys. 7).



Rys. 7. Menu główne narzędzia znajdź urządzenie tabletu Sony.

Sprawdzono tą metodę również na urządzeniu Xiaomi (Rys. 8).

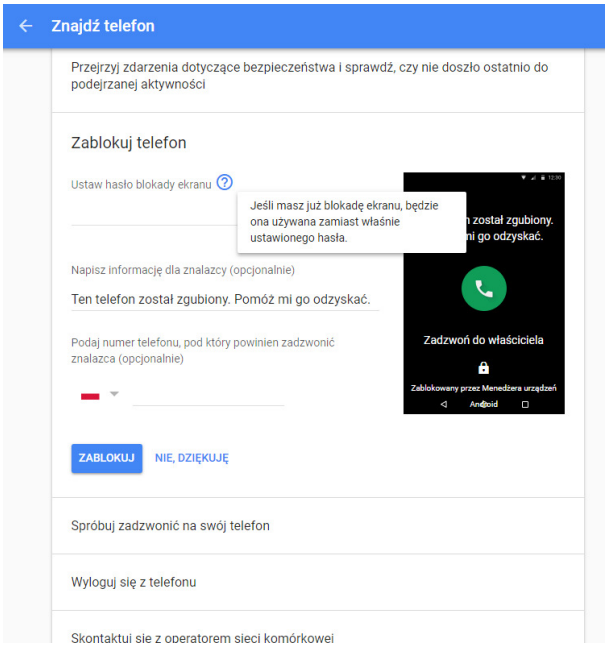
Aby metoda zadziała urządzenie musi być połączone z siecią Wifi lub transferem mobilnym, a także być zalogowanym na koncie Google. W ten sposób po zalogowaniu na Google znajdź urządzenie uzyskuje się dostęp do m.in. lokalizacji urządzenia. Innymi funkcjonalnościami oferowanymi przez Google jest odtworzenie zdalne dzwonka, umieszczenie na ekranie informacji, jak skontaktować się z właścicielem, zablokowanie urządzenia lub jego wyczyszczenie.



Rys. 8. Menu główne narzędzia znajdź urządzenie smartfonu Xiaomi.

W przypadku tego urządzenia opcje wszystkie były dostępne (Rys. 9), wobec czego można było przeprowadzić testy.

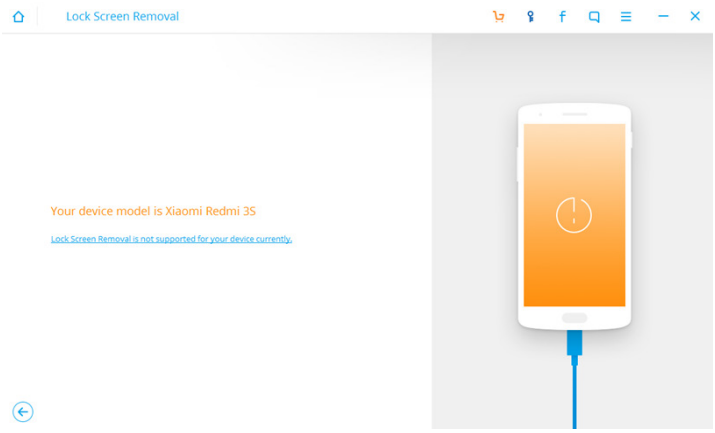
Po wybraniu stosownych opcji „Zablokuj telefon" uzyskuje się kilka opcji, jedną z nich jest interesująca z punktu widzenia testów „Ustaw hasło blokady ekranu". Należy w tym miejscu ustawić nowe hasło odblokowania i odblokować nim urządzenie.



Rys. 9. Opcje zmiany blokady ekranu.

7.2. Dr fone

Dr Fone to program komputerowy oferujący wiele opcji, jedną z nich jest zdjęcie blokady ekranu zdalnie. Do przeprowadzenia próby konieczne jest urządzenie mobilne i komputer z zainstalowanym programem połączonych kablem USB. Aplikacja po wyborze interesującej opcji program łączy się z urządzeniem mobilnym. Niestety w obu przypadkach urządzenie nie jest wspierane przez program (Rys. 10).



Rys. 10. Efekt działania Dr. Fone

Oznaczać to może nieskuteczność metody. Nie można z jej pomocą zdjąć blokady z powodu braku wsparcia przez aplikację.

8. Wnioski

Na podstawie badań przeprowadzonych na 66 osobach zauważono, że większość osób współcześnie używa smartfonów. Niestety większość z nich nie stosuje skutecznej ochrony swojego urządzenia a także danych na nim

zgromadzonych. Wśród osób stosujących skuteczniejszą ochronę urządzenia najpopularniejszą blokadą urządzenia był kod PIN, najmniej popularną było hasło. Można zauważyć spore zainteresowanie danymi biometrycznymi takimi jak odcisk palca. Użytkowników smartfonów zapytano o długość hasła, w większości hasła miały do 5 znaków długości, najczęściej jest to zbyt mało, aby niemożliwe było obejście blokady atakiem brutal force. Duża część użytkowników zadeklarowała konieczność wykonania do 5 prób, aby skutecznie odblokować urządzenie. Najczęściej mimo wiedzy o wpływie zmian w blokadzie ekranu, użytkownicy zmian tych nie dokonują wcale lub dokonują ich wraz ze zmianą urządzenia. Zapytani o najbardziej przyjazną użytkownikowi metodę blokady ekranu wskazano najczęściej wzór, najmniej zwolenników uzyskało hasło.

Na postawie analizy źródeł można wywnioskować, że na urządzenia z systemem Android stworzono wiele rodzajów ataków i zagrożeń, system Android opracował wiele sposób ochrony przez zagrożeniami, zdarza się jednak że zabezpieczenia zawodzą. Często w kontekście bezpieczeństwa najsłabszym z ogniw jest użytkownik nie przykuwający należytej uwagi do bezpieczeństwa urządzenia i danych na nim zgromadzonych.

Przeprowadzone badania z użyciem urządzeń mobilnych można zauważyć, że zdjęcie lub obejście blokady ekranu nie jest łatwym zadaniem i nie zawsze okazuje się skuteczne. Jednym ze sposobów zabezpieczenia przed obejściem blokady jest zachowanie wyłączonego Debugowania przez USB, uchroni to urządzenie przed spora grupą metod korzystających z połączenia USB. Odnosząc się do postawionej tezy można zauważyć, że przy należytej uwadze i prawidłowym nawykom użytkowników dane przechowywane na urządzeniach mobilnych są bezpieczne. W przypadku braku świadomości lub lekceważenia bezpieczeństwa, dane te nie mają zagwarantowanego bezpieczeństwa.

Literatura

[1] Drake J.J. Fora P.O. Lanier Z. Mulliner C. Pidley S. A. Wicherski G., Android Podręcznik Hackera, Helion, 2015.

[2] Bing H., Analysis and Research of System Seciurity Based on Android, Fifth International Conference on Intelligent Computation Technology and Automation, 2012

[3] Mu J. Cui A. Rao J., Android Mobile Security – Threats and Protection, International Conference on Computer, Network and Communication Engineering, 2013.

[4] Wu X. Li X, Hack Android Application and Defence, 3rd International Conference on Computer Science and Network Technology, 2013.

[5] Weidman G., Bezpieczny system w praktyce. Wyższa szkoła hackingu i testy penetracyjne, Helion, 2015.

[6] Rashidi B. Fung C., A Survey of Android Security Threats and Defenses, ResearchGate, 2015.

[7] Fang Z. Han W. Li Y., Permission based Android security: Issues and countermeasures, Research Collection School Of Information Systems, 2014.

[8] Delac G. Silic M. Krolo J., Emerging Security Threats for Mobile Platforms, Faculty of Electrical Engineering and Computing.

[9] Gunasekera S., Android Apps Security, Apress, 2012.

[10] Holla S. Katti M.M., Android based mobile application development and its serciurity, Inernational Journal of Computer Trends and Technology, 2012.

[11] Liebergeld S. Lange M., Android Security, Pitfalls, Lessons Learned and BYOD, 2013.

[12] https://www.spidersweb.pl/2018/02/android-ios-udzialy-rynkowe.html

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 9 (2018) 308-310
Wysłane:2018-06-22
Przyjęte: 2018-06-27

Autorska metoda zabezpieczenia ekranu urządzenia mobilnego

Grzegorz Iwaniuk*

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

Streszczenie. Wraz z rozwojem technologii urządzeń mobilnych wzrasta potrzeba zabezpieczenia tych urządzeń przed nieautoryzowanym dostępem. Aktualnie w powszechnym użyciu są metody odblokowywania urządzeń , których działanie polega na odczytaniu wzorca ruchu lub hasła wprowadzonego przez użytkownika. Artykuł przedstawia stworzenie autorskiej metody zabezpieczenia ekranu urządzenia mobilnego oraz porównania jej z powszechnie używanymi metodami zabezpieczeń.

Słowa kluczowe: Android; blokady ekranu; metody zabezpieczeń

*Autor do korespondencji.

Adres e-mail: grzegorz.iwaniuk3@gmail.com

Author's method of securing the screen of the mobile device.

Grzegorz Iwaniuk*

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

Abstract. With the development of mobile device technology, the need to protect these devices from unauthorized access is increasing. Currently, there are common methods of unlocking devices whose operation consists in reading the pattern or the password entered by the user. The article presents the creation of an original method of screen protection of a mobile device and its comparison with commonly used screen locking methods.

Keywords: Android; screen lock; security methods

*Corresponding author.

E-mail address: grzegorz.iwaniuk3@gmail.com

1. Wstęp

Wraz z rozwojem technologii urządzeń mobilnych wzrasta potrzeba zabezpieczenia tych urządzeń przed nieautoryzowanym dostępem. Coraz więcej osób, jeżeli nie większość używa nowoczesnych smartfonów nie tylko do dzwonienia i wysyłania SMS ale też do przeglądania Internetu, sprawdzania poczty, sprawdzania portali społecznościowych, robienia operacji bankowych oraz innych czynności, które wymagają dostępu do imiennych kont internetowych.

Wraz ze wzrostem personalizacji urządzeń mobilnych rośnie niebezpieczeństwo nieautoryzowanego dostępu do danych osób nieuprawnionych poprzez przejęcie urządzenia mobilnego. Pierwszą fazą obrony przed takim dostępem są blokady ekranu uniemożliwiające przeglądanie zawartości telefonu bez wcześniejszego wprowadzenia hasła zabezpieczającego. Blokada oparta na haśle dostępu ma uniemożliwić nieuprawniony dostęp ale nie powinna utrudniać uprawnionego dostępu do urządzenia.

Dodatkowo każda z metod zabezpieczenia ma swoje mocne i słabe strony, które wpływają na bezpieczeństwo i wygodę jej użytkowania.

Celem niniejszego artykułu było opisanie autorskiej metody zabezpieczenia ekranu urządzenia mobilnego oraz porównania jej do istniejących metod.

2. Klasyczne metody zabezpieczenia ekranu urządzenia mobilnego

Aktualnie do zabezpieczeń urządzeń mobilnych używa się następujących metod opartych na gestach [1]:

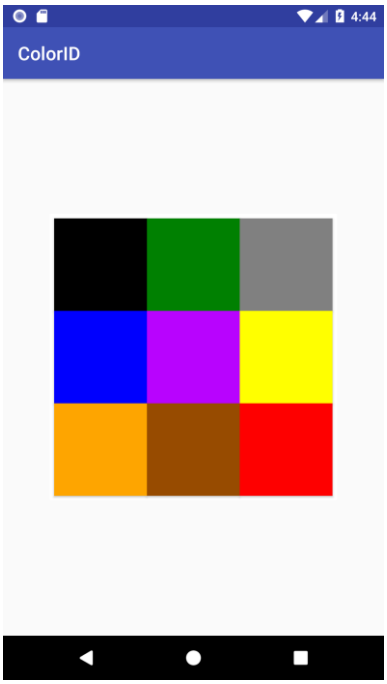
- Odblokowywanie przy użyciu przesunięcia palca - najprostsza metoda na odblokowanie telefonu. Polega ona na przesunięciu palcem po ekranie w określonym kierunku np. w kierunku ikonki otwartej kłódki, lub w którąkolwiek stronę ekranu smartfona. Metoda ta nie zapewnia żadnej ochrony przed nieautoryzowanym dostępem do urządzenia.
- PIN - metoda polegająca na ustawieniu z kombinacji 10 cyfr od 0 do 9 hasła o długości co najmniej 4 znaków. Liczba kombinacji cyfr dla kodu o długości 4 znaków wynosi 10^4 czyli 10000 [2].
- Hasło – metoda podobna do PIN-u, z tą różnicą, że oprócz cyfr używa się w niej małych, wielkich liter oraz znaków specjalnych. Dzięki użyciu liter oraz znaków specjalnych liczba znaków możliwych do użycia wynosi 94. Dzięki możliwości użycia 94 znaków liczba kombinacji przy 4 znakach wynosi 78074896. Hasło o długości 8 znaków ma 6095689385410820 różnych kombinacji [3].
- Wzór (Pattern Lock) - metoda ta polega na narysowaniu wzoru na polu składającym się z 9 punktów. Pole ma wymiar 3 x 3 punkty. Minusem tego rozwiązania jest możliwość podejrzenia przez osoby postronne lub też

może zostać ślad na ekranie, który ułatwi odgadnięcie wzoru [4].

3. Autorska metoda zabezpieczenia ekranu urządzenia mobilnego.

Autorska metoda zabezpieczenie ekranu urządzenia mobilnego została nazwana ColorID. Metoda ColorID w odróżnieniu od wcześniej przedstawionych metod wykorzystuje kombinację kolorów do odblokowania ekranu urządzenia.

Do odblokowania urządzenia należy wybrać kombinację kolorów wprowadzoną przez użytkownika w czasie konfiguracji zabezpieczenia. Na interfejsie metody ColorID zostało zaprogramowanych 9 przycisków ustawionych w 3 rzędy i 3 kolumny. Przy każdym uruchomieniu aplikacji zmieniane są w sposób losowy kolory przycisków. Rysunek 1. przedstawia interfejs metody ColorID.



Rys 1. Interfejs metody ColorID

Dzięki użyciu 9 kolorów liczba możliwych kombinacji dla hasła o długości 4 kolorów wynosi 6561. Wraz ze zmianą długości hasła liczba możliwych kombinacji zmienia się według wzoru:

$$y = 9^x \tag{1}$$

Gdzie: y – liczba kombinacji, x – długość hasła

Liczba możliwych kombinacji w zależności od długości hasła została zaprezentowana w tabeli 1.

Tabela 1. Tabela przestawiająca liczbę możliwych kombinacji w zależności od długości hasła w metodzie ColorID

Długość hasła	1	2	3	4	5	6	7	8
Liczba kombinacji	9	81	729	6561	59049	531441	4782969	43046721

4. Badania nad metodą ColorID

Do zbadania metody zostało wykonanych 400 badań na użytkownikach podzielonych ze względu na:

- Płeć użytkownika
  - Kobieta
  - Mężczyzna
- Wiek użytkownika
  - >20 lat
  - 21-30 lat
  - 31-40 lat
  - 41-50 lat
  - 51+ lat
- Miejsce zamieszkania
  - Wieś
  - Miasto

Po przeprowadzeniu badań zostały wykonane obliczenia średniej liczby prób i średniego czasu potrzebnego do odblokowania urządzenia.

5. Porównanie metody ColorID do tradycyjnych metod odblokowania

5.1. Porównanie metod pod kątem średniej liczby prób potrzebnych do odblokowania urządzenia.

Po przeprowadzeniu badań nad metodą ColorID metoda ta została porównana z wynikami badań nad klasycznymi metodami takimi jak: PIN, hasło oraz wzór [5].

Porównanie średniej liczby prób potrzebnych do odblokowania urządzenia z podziałem na płeć zostało zaprezentowane w tabeli 2.

Tabela 2. Średnia liczba prób potrzebna do odblokowania urządzenia z podziałem na płeć

	PIN	**Hasło**	**Wzór**	**ColorID**
Kobieta	1,04	1,342	1,444	1,275
Mężczyzna	1,042	1,313	1,157	1,295

Porównanie średniej liczby prób potrzebnych do odblokowania urządzenia z podziałem na miejsce zamieszkania zostało zaprezentowane w tabeli 3.

Tabela 3. Średnia liczba prób potrzebna do odblokowania urządzenia z podziałem na miejsce zamieszkania

	PIN	**Hasło**	**Wzór**	**ColorID**
Miasto	1,043	1,296	1,122	1,279
Wieś	1,037	1,372	1,189	1,291

Porównanie średniej liczby prób potrzebnych do odblokowania urządzenia z podziałem na wiek zostało zaprezentowane w tabeli 4.

Tabela 4 Średnia liczba prób potrzebna do odblokowania urządzenia z podziałem na wiek

	PIN	Hasło	Wzór	ColorID
<20	1,091	1,228	1,167	1,148
21-30	1	1,288	1,136	1,253
21-40	1,02	1,334	1,108	1,288
41-50	1	1,348	1,125	1,338
51+	1,091	1,409	1,216	1,4

5.2. Porównanie przeprowadzonych badań pod kątem średniego czasu potrzebnego do odblokowania urządzenia.

Porównanie średniego czasu potrzebnego do odblokowania urządzenia przy pomocy różnych metod z podziałem na płeć zostało zaprezentowane w tabeli 5.

Tabela 5. Średni czas potrzebny do odblokowania urządzenia z podziałem na płeć

	PIN	Hasło	Wzór	ColorID
Kobieta	5,158	8,837	6,282	6,271
Mężczyzna	4,417	7,537	5,516	6,432

Porównanie średniego czasu potrzebnego do odblokowania urządzenia przy pomocy różnych metod z podziałem na miejsce zamieszkania zaprezentowano w tabeli 6.

Tabela 6. Średni czas potrzebny do odblokowania urządzenia z podziałem na miejsce zamieszkania

	PIN	Hasło	Wzór	ColorID
Miasto	4,409	7,435	5,457	6,281
Wieś	5,341	9,281	6,543	6,422

Porównanie średniego czasu potrzebnego do odblokowania urządzenia przy pomocy różnych metod z podziałem na wiek zostało zaprezentowane w tabeli 7.

Tabela 7. Średni czas potrzebny do odblokowania urządzenia z podziałem na wiek

	PIN	Hasło	Wzór	ColorID
<20	3,636	6,121	4,546	4,811
21-30	3,303	6,106	4,894	4,872
31-40	3,922	6,579	5,079	5,778
41-50	3,778	7,486	5,389	7,499
51+	8,636	13,807	9,08	8,798

5. Wnioski

W wyniku badań stwierdzono następujące prawidłowości:

- Kobiety potrzebują średnio mniej prób do odblokowania urządzenia zabezpieczonego metodą ColorID od mężczyzn.
- Mieszkańcy miast potrzebują mniejszej liczby prób do odblokowania urządzenia zabezpieczonego ColorID.
- Wraz ze wzrostem wieku potrzeba większej liczby prób aby odblokować urządzenie chronione metodą ColorID.
- Kobiety potrzebują mniej czasu do odblokowania urządzenia chronionego ColorID.
- Mieszkańcy miast potrzebują mniej czasu do odblokowania urządzenia niż mieszkańcy wsi.
- Wraz ze wzrostem wieku zwiększa się czas potrzebny do odblokowania urządzenia.

We wszystkich powyższych przypadkach metoda ColorID wypadła gorzej od PIN-u i od wzoru, a lepiej od hasła. Dodatkowo można stwierdzić, że hasło jest metodą, która generuje najwięcej pomyłek i potrzeba najwięcej czasu żeby go użyć.

W wyniku przeprowadzonych badań i porównań można stwierdzi, że metoda ColorID może konkurować z powszechnie stosowanymi metodami. Jest to metoda, która zapewnia czas dostępu do urządzenia na poziomie podobnym do klasycznie stosowanych metod oraz podobną liczbę prób potrzebnych do odblokowania urządzenia.

Literatura

[1] https://www.theverge.com/, [18-04-2018[

[2] https://www.droid-life.com, [01-05-2018]

[3] https://www.androidcentral.com, [18-04-2018]

[4] Wang K. Wang Y. Yan J. Zhang Y., Security Analysis and Vulnerability Detection of Gesture-based Lock in Android Applications, IEEE TrustCom-BigDataSet-ISPA, 2016.

[5] Iwaniuk A., Ocena zabezpieczeń wybranej platformy mobilnej, praca magisterska, Politechnika Lubelska, 2018.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 9 (2018) 311-317
Wysłane: 2018-06-25
Przyjęte: 2018-07-01

Metoda zwiększenia pojemności kodów QR – Hexa QR Code

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

Streszczenie. W artykule zaprezentowano autorską metodę zwiększenia pojemności kodów QR poprzez wykorzystanie ich właściwości do odczytu z dowolnej perspektywy. Przedstawione zostały proces kodowania i dekodowania symbolu oraz wyniki testów. W artykule został również zawarty przegląd literatury z dziedziny metod zwiększania pojemności kodów QR.

Słowa kluczowe: kod QR; Hexa QR Code; kody 3D; zwiększenie pojemności

Adres e-mail: daniel.janowski@pollub.edu.pl

Method of increasing the QR code capacity – Hexa QR Code

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

Abstract. The article presents the author's method of increasing the QR code capacity through using their property of being readable from any perspective. The symbol encoding and decoding process and test results are presented. The article also includes a literature review in the field of method of increasing the QR code capacity.

Keywords: QR code; Hexa QR Code; 3D codes; increasing the capacity

E-mail address: daniel.janowski@pollub.edu.pl

1. Wstęp

Kody kreskowe powstały w celu ułatwienia i przyspieszenia identyfikacji towarów i usług. Ilość danych jaka dzięki nim mogła być przechowywana była niewielka, jednak prostota oraz szybkość i łatwość odczytywania, wpłynęły na ich popularność. Niestety, ograniczeniem w ich odczytywaniu, było używanie specjalnych czytników, dane w nich przechowywane mogły być zrozumiałe w ramach danego systemu, a ich niewystarczająca pojemność względem danych, jakie producenci chcieli by w nich zaszywać, spowodowała, że rozpoczęto poszukiwania alternatywnych rozwiązań.

W latach 90-tych zaczęto opracowywać tak zwane kody 2D, czyli takie, w których dane ułożone są nie tylko w poziomie, ale i w pionie. Dzięki temu rozwiązaniu, ilość przechowywanych danych oraz szybkość ich detekcji przez urządzenia odczytujące wielokrotnie wzrosła. Mogą one pomieścić od kilku do kilku tysięcy znaków, co przekłada się na ich uniwersalność.

Do tej pory powstało wiele standardów, jednak ze względu na swoje wady i zalety oraz wymagania jakie są przed nimi stawiane, do najbardziej popularnych należą: Data Matrix, PDF417 oraz kod QR (ang. Quick Response code). Każdy z tych typów ma swoje zastosowanie w konkretnych segmentach przemysłu. Pierwotnie kody 2D wykorzystywane były w logistyce, transporcie i handlu, ale wraz z rozwojem technologii cyfrowych i internetowych, zaczęto wykorzystywać je również w reklamie i celach rozrywkowych, takich jak technologia wirtualnej rzeczywistości Niestety dane jakie próbowano w nich zamieszczać, były zbyt duże jak na ich możliwości, dlatego zaczęto modyfikować je tak, aby mogły one pomieścić coraz więcej informacji. W toku tych działań powstały kody 3D, których trzecim wymiarem był kolor. W niniejszym artykule zostały przedstawione i porównane różne technologie zwiększenia pojemności kodu QR. Została również przedstawiona implementacja nowego, unikalnego algorytmu.

2. Przegląd literatury

2.1. QR code

Rys. 1. Przykład kodu QR w wersji 2

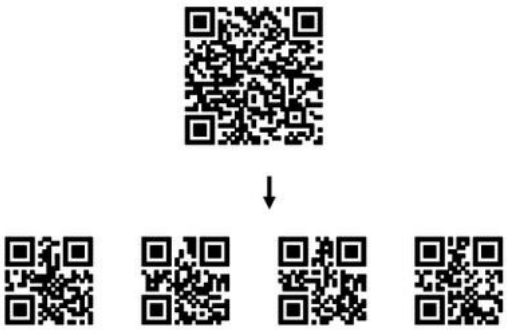
Kod QR (Rys. 1.) został opracowany w 1994 roku przez japońską firmę Denso-Wave, w celu przyspieszenia identyfikacji przez czytniki, przedmiotów szybko poruszających się względem nich. Początkowo używany w przemyśle transportowym i logistyce, szybko został zaadoptowany przez inne gałęzie gospodarki. Stało się to za sprawą upowszechnienia się smartfonów i tabletów, zniesienia licencji na jego stosowanie oraz dużej ilości oprogramowania i bibliotek do jego generowania.

Kod QR został włączony do norm ISO[1] w 2000 roku jako standard ISO/IEC 18004, który definiuje elastyczne rozwiązanie pośród innych kodów 2D. Oferuje on dużą pojemność (do 7 tys. znaków), wsparcie obsługi różnych zestawów znaków (numeryczny (*ang. numeric*), alfanumeryczny (*ang. alphanumeric*), binarny (*ang. byte data*) i Kanji), funkcje zapewniające szybkie i niezawodne wykrycie pod wpływem zmiennego oświetlenia, zmian położenia i orientacji oraz wbudowaną czteropoziomową korekcję błędów algorytmem Reeda-Solomona, którą można przywrócić od ok. 7% do ok. 30% uszkodzonych danych. Charakteryzuje się modularną i stało wymiarową strukturą. Oznacza to, że składa się z modułów będących źródłem niesionej informacji, a liczba modułów w poziomie i w pionie jest stała w obrębie danej wersji. Moduł jest najmniejszą jednostką informacji przechowywanej w kodzie QR, gdzie ciemny moduł oznacza binarną jedynkę, a jasny zero. Kod QR występuje w 40-stu wersjach, które determinują liczbę modułów, co za tym idzie ilość danych możliwych do przechowania [1, 2].

2.2. Tryb „Structured append”

Jedną z funkcjonalności standardowego kodu QR, jest tryb „structured append”, umożliwiający połączenie w sekwencji do 16-stu symboli kodu QR lub podzielenie jednego kodu QR na nawet 16 mniejszych kodów (Rys. 2.).

W tym trybie kody mogą być skanowane w dowolnej kolejności, jednak wymagane jest odczytanie ich wszystkich, żeby poprawnie zrekonstruować zawartą w nich wiadomość. Jest to spowodowane nagłówkami zawartymi w poszczególnych symbolach, zawierającymi informację o długości całej sekwencji i pozycji kodu w sekwencji oraz służą do weryfikacji czy symbole należą do jednej i tej samej sekwencji. [1]



Rys. 2. Przykład użycia trybu "Structured append" do podzielenia kodu QR w wersji 4. na cztery kody w wersji 1. [1]

2.3. Kompresja danych

Maksymalna pojemność kodu QR w wersji 40. wynosi ponad 7 tys. znaków, ale jednocześnie taki kod może być nieczytelny lub za duży do umieszczenia go w pewnych miejscach. Najlepszym w takim przypadku rozwiązaniem jest zmniejszenie ilości danych i użycie niższej wersji kodu, jednak aby to osiągnąć bez ich utraty, trzeba użyć algorytmów kompresujących. Jest to bardzo dobra metoda, aby „zapakować” nawet megabajty danych. Niestety, czytniki kodów QR domyślnie nie wspierają kompresji danych, co ogranicza stosowanie tej techniki tylko dla dedykowanych czytników. Aby przyspieszyć wybór odpowiedniego algorytmu i jego interpretację przez czytnik, można użyć wbudowanej w kod QR funkcjonalności – Extended Channel Interpretation (ECI). Służy ona do informowania systemu odbiorczego, że odczytane dane wymagają pewnego określonego dekodowania, dekompresowania i/lub deszyfrowania, zanim będzie można skorzystać z ich zawartości [3].

2.4. Kody 3D

Nazywa się tak kody, które oprócz dwóch wymiarów odpowiadającym zapisywaniu informacji w poziomie i w pionie, posiadają trzeci wymiar, którym jest kolor. Każdemu kolorowi, mogącemu wystąpić w danym kodzie, przypisana jest inna wartość, odpowiadająca ciągowi kilku bitów, w przeciwieństwie do jedno-bitowego kodu QR.

Używa się w nich zazwyczaj 8-bitowej przestrzeni kolorów RGB, która oferuje 16 777 216 różnych kolorów. Jest to uwarunkowane tym, że aparaty cyfrowe głównie w tej przestrzeni pracują oraz jest to podstawowa przestrzeń dla plików graficznych.

$$16\,777\,216 = 256 \times 256 \times 256 = 2^8 \times 2^8 \times 2^8 = 2^{24} = 2^n \quad (1)$$

$$n = \log_2(\textit{liczba użytych kolorów}) \quad (2)$$

gdzie: n – liczba kodów QR możliwych do połączenia

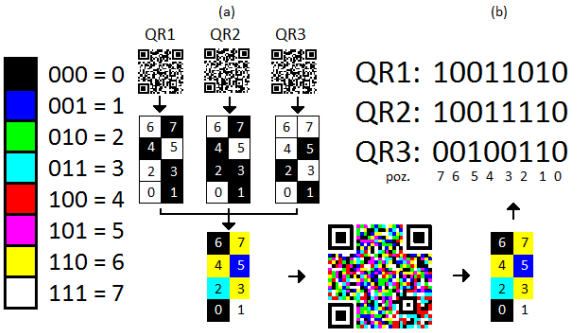
Każdy kanał tej przestrzeni oferuje po 256 różnych odcieni, co równa się możliwości zwiększenia pojemności kodu QR 24-krotnie (1). Na podstawie równania (2), można stwierdzić, że 2^n niezależnych kolorów jest potrzebnych do zakodowania n liczby kodów QR, co za tym idzie dla każdego odrębnego wzoru binarnego jest przypisana odrębna kombinacja wag kanałów R, G, B [4].

Na podstawie technik przedstawionych w dalszej części rozdziału, można wydzielić dwa rodzaje wyznaczania kolorów dla pojedynczych modułów:

- Pozycyjne – polegające na łączeniu kilku kodów QR w jeden kolorowy kod. Liczba łączonych kodów, determinuje liczbę bitów potrzebnych na zakodowanie informacji z każdego kodu. Każdej kombinacji bitów przypisuje się kolor, a pozycja bitu wyznacza kod, z którego pochodzi informacja. Warunkiem jest użycie tej samej wersji kodu we wszystkich wystąpieniach.
  Kodowanie odbywa się poprzez użycie operacji logicznej OR na każdym module z danej pozycji ze wszystkich łączonych kodów, a następnie wybranie koloru odpowiadającemu przeliczonej wartości i wstawienie go na tej samej pozycji w generowanym kodzie. Podczas dekodowania wykrywany jest kolor modułu z danej pozycji, a następnie wybierana jest wartość do niego

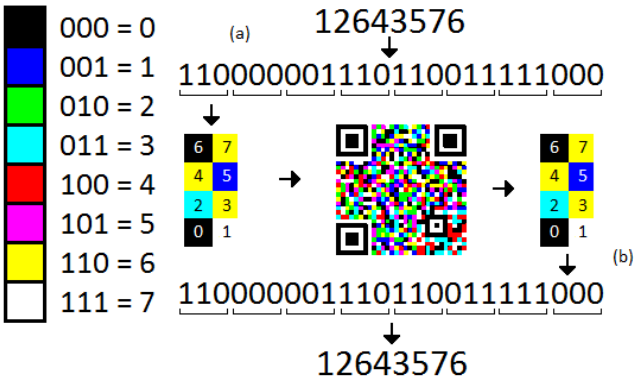
[1] Międzynarodowa Organizacja Normalizacyjna (ang. *International Organization for Standardization*)

przypisana, z której każda kolejna wartość bitu przypisywana jest do kolejnych kodów, które tworzyły kod (Rys. 3.).



Rys. 3. Przykład kodowania trzech czarno-białych słów kodowych do jednego kolorowego słowa kodowego (a) i dekodowania kolorowego słowa kodowego do trzech wartości binarnych (b), w kolorowym kodzie QR

- Wartościowe – polegające na tworzeniu kodu w sposób analogiczny do kodu QR. Liczba kolorów wybranych do zakodowania informacji, determinuje liczbę bitów potrzebnych na zakodowanie jednego modułu. Każdemu kolorowi przypisuje się wartość jaką reprezentuje.
  Kodowanie odbywa się poprzez wyodrębnienie kolejnych ciągów bitów, wybieranie koloru im odpowiadającego i wstawianie go na odpowiedniej pozycji w generowanym kodzie. Podczas dekodowania wykrywany jest kolor modułu z danej pozycji, a następnie wybierana jest wartość do niego przypisana, która jest dołączana do dekodowanej informacji (Rys. 4.).



Rys. 4. Przykład kodowania ciągu znaków do kolorowego słowa kodowego, poprzez wyznaczenie poszczególnych wartości z ciągu bitów (a) i dekodowania kolorowego słowa kodowego do ciągu bitów reprezentujących dane (b)

Poniżej zostały przedstawione po dwie techniki „pozycyjne” i „wartościowe”.

1) Color QR code

Jest to technika „pozycyjna”, polegająca na łączeniu trzech standardowych (czarno-białych) kodów QR w jeden kolorowy, wykorzystując kanały przestrzeni barw CMY do kodowania i RGB do dekodowania. Wzory wyrównania i synchronizacji pozostają czarno-białe, natomiast wzorom detekcji przypisuje się podstawowe barwy przestrzeni barw CMY i RGB (Rys. 5.). Dzięki tej operacji czytnik może synchronizować kolory zeskanowane z oczekiwanymi [5, 6, 7].

Rys. 5. Przykład Color QR code [6]

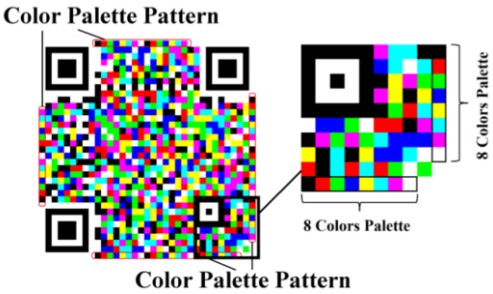
2) Paper Memory Code (PM Code)

Jest to jeden z pierwszych zaprezentowanych kodów 3D bazujących na kodzie QR. Kodowanie danych odbywa się w identyczny sposób jak w poprzedniej metodzie, jednak w tym przypadku pracuje na całej gamie przestrzeni kolorów RGB, dając możliwość połączenia od 3 do 24 czarno-białych kodów QR. Jest to jednak liczba zarezerwowana do użytku lokalnego lub w Internecie. Do zastosowań mobilnych, Wynalazca PM Code sugeruje łączenie od 3 do 8 kodów QR. W PM Code, kolory wzorów detekcji (*ang. finder pattern*), wyrównania (*ang. alignment patterns*) i synchronizacji (*ang. timing patterns*) oraz strefy ciszy (*ang. quiet zone*) są odwrócone w porównaniu do QR code (Rys. 6.) [8, 9, 10].

Rys. 6. Przykład PM Code [10]

3) High Capacity Colored Two Dimensional Code (HCC2D)

Jest to technika „wartościowa”, dzięki temu wymiar kodu HCC2D dostosowywany jest do rzeczywistego rozmiaru danych wejściowych. Wszystkie stałe elementy kodu QR wraz z informacją o wersji i formacie pozostają w tym kodzie niezmienne. Poza tym, udostępnia wszystkie funkcjonalności standardowego kodu QR oraz jest z nim w 100% kompatybilny, czyli czarno-biały kod HCC2D, zostanie bezproblemowo zdekodowany przez czytniki kodów QR. Jeśli kod składa się z co najmniej czterech kolorów. Dochodzi dodatkowy element kodu – paleta użytych barw (*ang. color palette pattern*) – zawierająca posortowane według jasności kolory, które determinują kombinacje bitów do nich przypisane oraz pozwala na lepsze dekodowanie, w przypadku powstania zmiany zabarwienia kodu pod wpływem światła (Rys. 7) [4, 11, 12].



Rys. 7. Przykład HCC2D z uwzględnieniem palety użytych barw [12]

4) Colored Quick-Response Code (CQRcode)

Jest to również technika „wartościowa”, jednak ogranicza się ona do czterech kolorów – trzech podstawowych kolorów przestrzeni RGB i białego. CQR Code ma tylko jeden rozmiar wersji, składający się z 49x49 modułów (wersja 8. kodu QR). W porównaniu do standardowego kodu QR w tej wersji, CQR Code wspiera tylko kodowanie danych binarnych i korekcję błędów opartą o algorytm Reeda-Solomona, natomiast nie występują wzory wyrównania, synchronizacji oraz informacje o wersji (Rys. 8) [13].

Rys. 8. Budowa i przykład CQRcode [13]

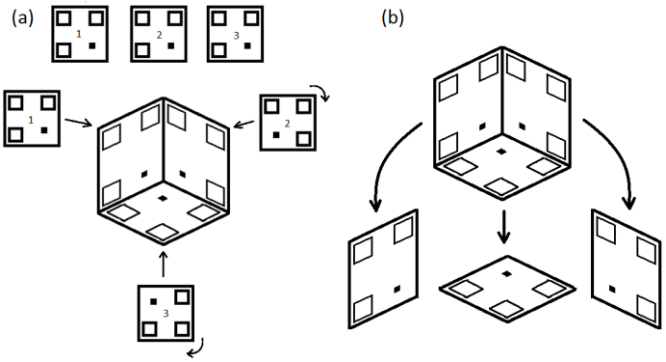
3. Propozycja metody zwiększenia pojemności kodu QR - Hexa QR Code

W przeciwieństwie do zaprezentowanych w poprzednim rozdziale technik, zaproponowana metoda nie generuje ani nie modyfikuje w żaden sposób generowania „ciała” kodu QR. Służy ona do przygotowania miejsca, w którym dane symbole mają być umieszczone. Zostały w niej wykorzystane dwie główne jego zalety – możliwość odczytu z dowolnej perspektywy. Kolejnymi ważnymi cechami były stały wymiar oraz forma kwadratu. Termin kod 3D został potraktowany w tym przypadku bardziej dosłownie, ponieważ do jego generowania wykorzystano geometrię przestrzenną i technologię 3D.

Rys. 9. Przykład Hexa QR Code w wersji 2.

Generowanie kodu odbywa się poprzez wykorzystanie sześcianu, który rzutowany jest izometrycznie na jeden z jego wierzchołków pod kątem 45º. Wierzchołek ten, wyznacza środek tak powstałego na rzutowanej płaszczyźnie sześciokąta foremnego, zaś osie wyznaczają krawędzie oraz ściany sześcianu wykorzystywane w kolejnych krokach. Nic jednak nie stoi na przeszkodzie, żeby użyć przeciwnych krawędzi. Powstanie w ten sposób odwrócona o 180º wersja tej metody.

Do wypełnienia wyznaczonych ścian, mogą zostać użyte trzy niezależne kody QR lub zależne od siebie kody QR utworzone przy użyciu trybu „structured append”. Możliwe jest również użycie kodów 3D, wspierających wyżej wymienione wymagania, aby dodatkowo zwiększyć ich potencjał. Symbol „lewy” wypełnia ścianę bez modyfikacji. Natomiast symbole „prawy” i „dolny”, podczas wypełniania tych ścian, muszą zostać obrócone odpowiednio o 90º i o 180º zgodnie z ruchem wskazówek zegara (Rys. 10.). Dzięki tej operacji każdy z kątów sześciokąta posiada po dwa wzorniki pozycji, a kody zwrócone są ku środkowi figury. Ważnym elementem jest pozostawienie oryginalnej lub nawet nieco większej strefy ciszy każdemu kodowi w trakcie wypełniania, co umożliwi odczytanie ich również zwykłym czytnikiem.



Rys. 10. Schemat kodowania (a) i dekodowania (b) Hexa QR Code

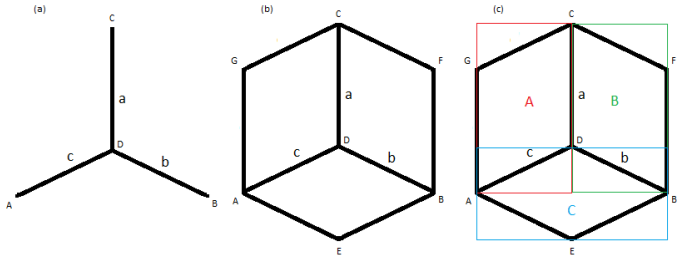
Odczytywanie kodu polega na podzieleniu go na trzy części wzdłuż linii, które wyznaczały poprzednio używane krawędzie. Następnie każdy z tak utworzonych obrazków należy wczytywać do dekodera kodów QR (Rys. 10.).

3.1. Implementacja

Ze względu na wykorzystane w algorytmie kodowania bryły i figury geometryczne, najprostszym sposobem na jego implementację było wykorzystanie grafiki 3D. Oferuje ona wszystkie przedstawione w założeniach operacje i przekształcenia. Dodatkowo algorytm ten może być zaimplementowany nie tylko w językach programowania wysokiego lub niskiego poziomu, ale również w programach do obróbki grafiki trójwymiarowej. Zależnie od wybranej techniki, tworzenie i renderowanie grafiki 3D, odbywa się w inny sposób i przy użyciu innych metod, jednak poniższe instrukcje są uniwersalne.

Podstawowym krokiem jest stworzenie wirtualnej sceny, do której dodawane będą kolejne wymagane obiekty. Pierwszym z nich jest wirtualna kamera, rzutująca izometrycznie na obiekty ustawione w jej polu „widzenia”. To ona wizualizuje i generuje zdjęcie przygotowanej sceny. Wizualizacja bazuje jednak na świetle, dlatego jego utworzenie jest kolejnym krokiem przy tworzeniu sceny. Najlepszym w tym przypadku wyborem, było użycie tak zwanego światła rozproszonego (*ang. ambient light*) oraz przypisanie go do obiektu 3D. Dzięki temu obiekt generuje światło na scenie i zapewnia mu równomierne oświetlenie z każdej strony. Wspomnianym obiektem jest sześcian o wymiarach 1:1:1, ustawiony w środku sceny. Analogicznie do założeń, kamera została ustawiona tak, aby rzutowała pod kątem 45º na jeden z wierzchołków bryły. Ściany sześcianu zostały tak przygotowane, aby materiał, czyli obrazy do nich

przypisywane, nie duplikowały się oraz wypełniały je odpowiednio obrócone.



Rys. 11. Schemat wydzielania obszarów dekodowania w Hexa QR Code

Ze względu na operacje na zdjęciach i obrazach, w algorytmie dekodowania została wykorzystana grafika 2D oraz rysowanie na płaszczyźnie.

Najważniejszym elementem jest wyznaczenie trzech jednakowej długości odcinków, których jeden punkt jest wspólny dla wszystkich, natomiast odcinki były odchylone od siebie pod kątem 120º, tak aby ich punkty tworzyły wierzchołki trójkąta równobocznego. Dzięki temu możliwe jest wykorzystanie ich w kolejnych krokach.

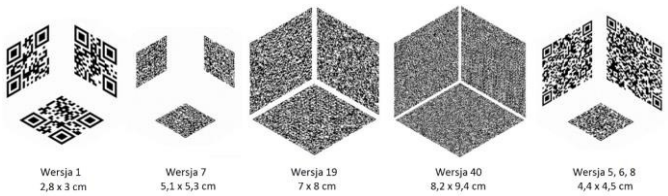
Na rysunku 11. zostały przedstawione trzy części A, B i C, wycinane z wczytanego obrazu, stanowiące obszary, w których dekoder powinien szukać kodów QR. Przed przystąpieniem do wycinania, następuje wyznaczenie trzech dodatkowych punktów, tworzących wraz z innymi punktami, sześciokąt foremny. Jednocześnie tworzone są równoległoboki z odcinkami a, b i c jak przedstawiono na rysunku, wyznaczające miejsca w których powinien znaleźć się kod. Wyznaczenie granic dla każdej części, odbywa się poprzez wyznaczenie najmniejszych i największych wartości x i y, z pośród wierzchołków danych figur. Następnie na ich podstawie wycina się fragmenty z wejściowego obrazu i maskuje, czyli zamalowuje się jednolitym kolorem, miejsca poza granicami poprzednio wyznaczonych równoległoboków. Ma to na celu zapobieganie powstawania przekłamań podczas wykrywania kodu QR, gdyż fragmenty sąsiednich kodów mogły by znaleźć się w wyciętym obrazie. Aby przyspieszyć ten proces, każda część powinna być przetwarzana równolegle.

4. Testy zaimplementowanego algorytmu

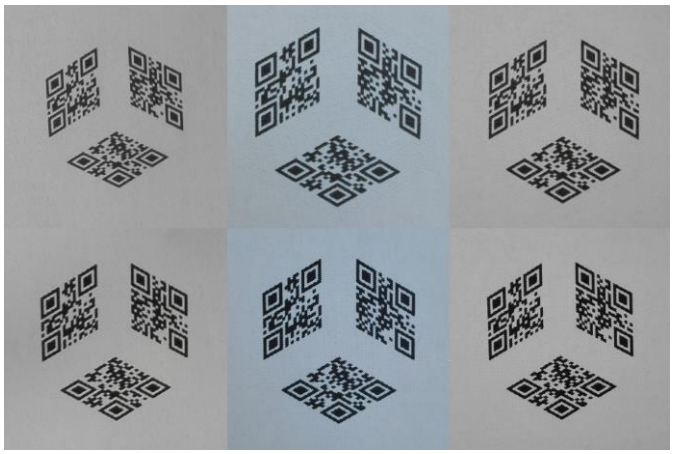
W celu implementacji i przeprowadzenia testów powyższych algorytmów, została przygotowana aplikacja na komputer stacjonarny w technologii WPF, generująca i dekodująca Hexa QR Code, przy użyciu biblioteki Zxing.NET [14]. Testy zostały podzielone na dwa etapy, oba przeprowadzone na jednym komputerze stacjonarnym wyposażonym w procesor Intel Core i5 1,6 GHz, 8 GB pamięci RAM i kartę graficzną Nvidia GeForce GT 2 GB.

W pierwszym etapie, wygenerowane symbole były od razu wczytywane do dekodera, aby sprawdzić jego możliwości, oraz to jaka jakość jest potrzebna, aby poprawnie odczytać nawet najwyższe wersje kodu QR. Drugi etap miał na celu przetestowanie Hexa QR Code'u w bardziej „naturalnych" warunkach.

Do testów zostało wygenerowanych pięć Hexa QR code'ów w rozdzielczości 1000x1000 px (Rys. 12.). Cztery w danym zestawie składały się z kodów QR o takiej samej wersji (1., 7., 20. i 40.), natomiast jeden składał się z kombinacji trzech różnych wersji (5., 6. i 8.). Zostały one wydrukowane drukarką atramentową Canon MG3053 i laserową Brother HL-2250DN, a jako próbki posłużyły ich zdjęcia, zrobione w świetle lampki biurowej oraz w świetle dziennym w nasłonecznionym i zacienionym miejscu, wbudowanym aparatem cyfrowym 8 Mpx w Huawei P8 Lite (Rys. 13). Symbole zostały wydrukowane w różnych wielkościach, ponieważ w przeciwnym wypadku były by całkowicie nieczytelne. Aby zostać poprawnie odczytane przez dekoder, musiały być uprzednio przygotowane. Należało z każdego zdjęcia wyciąć fragment w kształcie kwadratu tak, aby jego środek pokrywał się orientacyjnie ze środkiem symbolu.



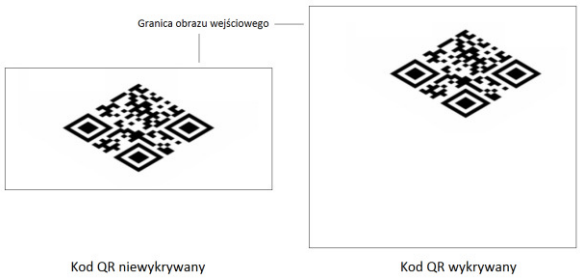
Rys. 12. Wygenerowane symbole użyte do testów



Rys. 13. Zdjęcia symboli w wersji 1. wydrukowane przy użyciu drukarki atramentowej (wiersz 1.) i drukarki laserowej (wiersz 2.), wykonane w świetle dziennym w nasłonecznionym (kolumna 1.) i zacienionym (kolumna 2.) miejscu oraz w świetle lampki biurowej (kolumna 3.)

Niestety możliwości odczytu Hexa QR Code, są uzależnione od możliwości odczytu biblioteki dekodującej. W niniejszej pracy została wykorzystana biblioteka ZXing.NET, która jest bardzo dobrym wyborem w przypadku odczytywania kodów QR, ustawionych na wprost z niewielkim marginesem błędu. Zniekształcenia wynikające z odmiennego sposobu odczytywania symbolu, powodują w tym przypadku niestabilne zachowanie się dekodera, a wykrycie kodu QR zaszytego w Hexa QR Code, jest często losowe. Trudności z dekodowaniem kodu QR objawiały się, gdy pochylony symbol był obrócony o kąt 150º, 240º lub 330º

(±10º – 15º) względem środka, a po przeprowadzeniu kilku dodatkowych testów przy użyciu tej biblioteki oraz biblioteki ZXing [15], okazało się, że wpływ na wykrycie takiego kodu ma również jego położenie i rozmiar na obrazie wejściowym (Rys. 14).

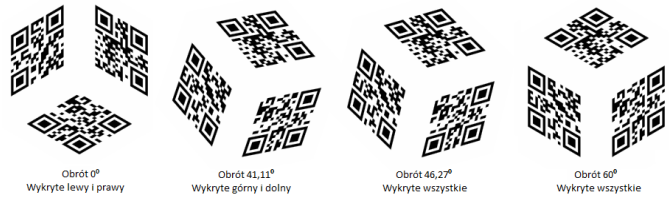


Rys. 14. Wpływ położenia pochylonego symbolu i rozmiaru obszaru roboczego, na możliwość jego wykrycia przez bibliotekę ZXing.Net i ZXing

Tabela 1. Porównanie wyników dekodowania Hexa QR code, otrzymanych podczas testów

Typ zdjęć	Warunki	Obrót	Wersja 1	Wersja 7	Wersja 19	Wersja 40	Wersja 5,6,8
Wygenerowane	-	0º	2/3	3/3	X	X	2/3
		13º	3/3	2/3	2/3	X	2/3
		46º	3/3	3/3	1/3	X	3/3
Wydruk atramentowy	Dzienne - nasłonecznione	0º	1/3	3/3	X	X	2/3
		13º	2/3	1/3	X	X	2/3
		46º	3/3	2/3	X	X	2/3
	Dzienne - zacienione	0º	1/3	3/3	1/3	X	1/3
		13º	1/3	1/3	X	X	1/3
		46º	1/3	2/3	X	X	1/3
	Sztuczne - lampka biurowa	0º	3/3	2/3	1/3	X	2/3
		13º	2/3	2/3	1/3	X	2/3
		46º	3/3	2/3	X	X	2/3
Wydruk laserowy	Dzienne - nasłonecznione	0º	3/3	X	X	X	2/3
		13º	2/3	X	X	X	2/3
		46º	3/3	X	X	X	2/3
	Dzienne - zacienione	0º	2/3	1/3	X	X	1/3
		13º	3/3	X	X	X	1/3
		46º	3/3	X	X	X	X
	Sztuczne - lampka biurowa	0º	3/3	X	X	X	1/3
		13º	2/3	X	X	X	2/3
		46º	3/3	X	X	X	1/3

Takie ograniczenie, miało szczególny wpływ na dolny obszar wykrywania, ponieważ kod QR w nim zawarty nie zawsze był wykrywany przez dekoder. Wczytanie obrazu z Hexa QR Code obróconym pod odpowiednim kątem na szczęście pomagało (Rys. 15.), dlatego w drugim etapie, wszystkie przygotowane zdjęcia zostały dodatkowo obrócone o kąt 13º oraz 46º. W tabeli 1. zostały zestawione wyniki testów wykrycia symboli dla poszczególnych konfiguracji.



Rys. 15. Wpływ obrotu Hexa QR Code, na możliwość wykrycia poszczególnych kodów QR przez bibliotekę ZXing.Net i ZXing

5. Wnioski

Przedstawiona w niniejszej pracy metoda zwiększenia pojemności kodu QR jest innowacyjna, jednak testy wykazały, że nie jest pozbawiona wad.

Zaproponowany algorytm kodowania jest prosty i w dużej mierze korzysta z możliwości grafiki 3D, co wspomaga jego przenośność pomiędzy technologiami. Dlatego może być zaimplementowany nie tylko w językach programowania wysokiego lub niskiego poziomu, ale jest to również możliwe w programach do obróbki grafiki trójwymiarowej. Takie rozwiązanie może pomóc w ograniczeniu kosztów jego implementacji, na przykład poprzez używanie bezpłatnego oprogramowania.

Fundamentalną rolę odgrywa tu dekoder symboli, który również musi być przystosowany do pracy w warunkach, jakie przewiduje specyfikacja kodów QR – możliwości odczytywania symbolu z różnej perspektywy. Technika ta nie ogranicza się tylko do kodów QR. Służy ona do przygotowania miejsca, w którym dane symbole mają być umieszczone, zatem użycie kodu, który wspiera możliwość odczytania pochylonego lub obróconego symbolu, jest możliwe. Zalecane jest jednak, aby zarówno symbol jak i wydruk były jak najlepszej jakości, a wybrany rozmiar był adekwatny do wybranej wersji symbolu, tak aby zapewnić jego poprawne odczytanie w różnych warunkach.

W porównaniu do przedstawionych w rozdziale 2.4 metod korzystających z koloru do zwiększenia pojemności symbolu, Hexa QR Code zajmuje 2,6 razy więcej powierzchni drukowanej. Porównując go jednak z trzema standardowymi kodami QR, ułożonymi obok siebie w podobnym stylu (dwa na górze, jeden na dole), zajmuje 0,86 razy mniej miejsca. Atutem też jest wsparcie odczytania każdego z zawartych symboli przez standardowe czytniki kodów QR.

Ze względu na użycie nieprofesjonalnych drukarek oraz ograniczenie rozdzielczości generowanego symbolu do 1000x1000 px, jakość wydruków testowych była niska, dlatego najlepiej dekodowane były kody w wersjach 1. – 8.

Literatura

[1] ISO/IEC, 18004:2006 Information technology — Automatic identification and data capture techniques — QR Code 2005 bar code symbology specification, Geneva: ISO/IEC, 2006.

[2] Denso-Wave Inc., „QRcode.com", http://www.qrcode.com/en/ [01.06.2018]

[3] AIM, Inc., „AIM", https://www.aimglobal.org/ [01.06.2018]

[4] A. Grillo, A. Lentini, M. Querini i G. F. Italiano, „High Capacity Colored *Two* Dimensional Codes," *Proceedings of the IMCSIT,* tom 5, pp. 709-716, 2010.

[5] O. Bulan, G. Sharma i V. Monga, „High Capacity Color Barcodes Using Dot Orientation and Color Separability," *SPIE-IS&T Electronic Imaging,* tom 7254, nr 725417, pp. 1-7, 2009.

[6] O. Bulan, H. Blasinski i G. Sharma, „Color QR Codes: Increased Capacity Via Per-Channel Data Encoding and Interference Cancellation," *Color and Imaging Conference Final Program and Proceedings,* nr 14627-0126, pp. 156-159, 2011.

[7] H. Blasinski i S. Fok, Mobile Color Barcode Reader, Stanford: Stanford University, 2011-2012.

[8] E. Jenks , „QR Based PM Code", 11.01.2007, http://symblogogy.blogspot.com/2007/01/qr-based-pm-code-best-3d-symbology-ever.html [01.06.2018]

[9] H. Kato, K. T. Tan i D. Chai, Barcodes for Mobile Devices, 1 red., Nowy Jork: Cambridge University Press, 2010.

[10] Inc., PMCode, „PMCode", http://pm-code.com/eng/ [01.06.2018]

[11] M. Querini, A. Grillo, A. Lentini i G. F. Italiano, „2D Color barcodes for mobile phones," *International Journal of Computer Science and Applications,* tom 8, nr 1, pp. 136-155, 2011.

[12] M. Querini i G. F. Italiano, „Reliability and Data Density in High Capacity Color Barcodes," *Special Issue on Advances in Systems, Modeling, Languages and Agents,* tom 11, nr 4, p. 1595–1615, 2014.

[13] M. E. V. Melgar, A. Zaghetto, B. Macchiavello i A. . C. A. Nascimento1, „CQR codes: Colored quick-response codes," w *IEEE Second International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, Berlin, 2012.

[14] M. Jahn, „ZXing.Net", https://github.com/micjahn/ZXing.Net [01.06.2018]

[15] S. Owen, „ZXing", https://github.com/zxing/zxing [01.06.2018]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 9 (2018) 318-323
Wysłane: 2018-06-30
Przyjęte: 2018-07-03

Analiza porównawcza sposobów tworzeniu aplikacji dla systemu Android z wykorzystaniem technologii Xamarin

Michał Bartkiewicz*, Adrian Dziedzic

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

Streszczenie. Praca przedstawia analizę technologii Xamarin w dwóch trybach pracy: Xamarin.Forms oraz Xamarin.Native, które pozawalają na stworzenie aplikacji na urządzenia mobilne z systemem Android. Porównanie dotyczy liczby linii wygenerowanego kodu, wydajności poszczególnych elementów oraz rozmiaru zainstalowanej aplikacji i pliku instalacyjnego apk. Analiza została dokonana na podstawie dwóch takich samych aplikacji stworzonych oddzielnie w obu podejściach. w wyniku analizy wskazano bardziej efektywne rozwiązanie do wybranych zastosowań.

Słowa kluczowe: Xamarin; Android; C#; .NET

*Autor do korespondencji.

Adres e-mail: michal.bartkiewicz@pollub.edu.pl

Comparative analysis of approches in developing Android applications using Xamarin technology

Michał Bartkiewicz*, Adrian Dziedzic

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

Abstract. Article shows analysis of Xamarin technology in two modes: Xamarin Forms and Xamarin Native, used for developing applications for mobile devices with Android system. Comparison concerns the number of generated lines of code, performance of each part and size of installed application and size of apk installation file. Analysis was based on two identical applications created using both approaches. As a result of the analysis the more efficient approach for given purpose has been indicated.

Keywords: Xamarin; Android; C#; .NET

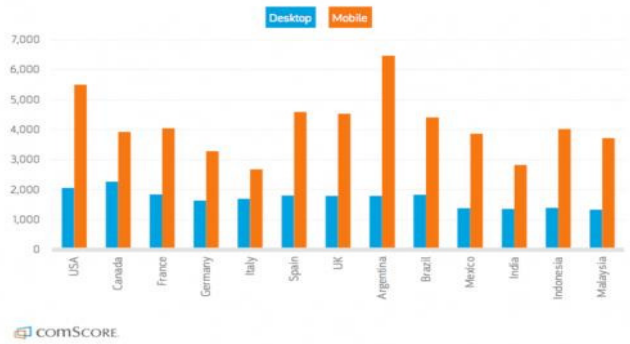
*Corresponding author.

E-mail address: michal.bartkiewicz@pollub.edu.pl

1. Wstęp

Dominującą częścią rynku z perspektywy wykorzystywanego oprogramowania przez zwykłych użytkowników stały się niewątpliwie smartfony. Urządzenia te nie służą już tylko do wysyłania wiadomości sms, czy dokonywania połączeń głosowych z innym odbiorcą, jak to miało miejsce w przypadku pierwszych telefonów komórkowych, ale umożliwiają wykonywanie wielu różnych czynności. Czynności te mogą być związane z pozyskiwaniem informacji, wykorzystaniem wbudowanych czujników i synchronizacją z innym sprzętem, czy też oprogramowaniem. Telefony mogą być również wykorzystywane do wspierania operacji biznesowych, takich jak dokonywanie płatności, a także służą celom czysto rozrywkowym. Niewątpliwą zaletą tych urządzeń, przyczyniającą się do popularności jest ich kompaktowy rozmiar, dzięki czemu każdy może mieć je zawsze pod ręką.

Rynek urządzeń mobilnych swoją popularnością przewyższył rynek urządzeń desktopowych. Użytkownicy więcej czasu spędzają z urządzeniami mobilnymi, do których zaliczają się również tablety, niż z komputerami osobistymi klasy PC [1]. Taka sytuacja spowodowała nagły wzrost zapotrzebowania na wytwarzanie oprogramowania, które jest dedykowane na urządzenia mobilne.



Rys. 1. Porównanie użycia urządzeń mobilnych oraz desktopowych przez użytkowników przy wykorzystaniu dłuższym niż 2 minuty [1]

Od niedawna rynek technologiczny dotyczący rozwiązań mobilnych się zmienił. Zaczęły pojawiać się nowe technologie, które często pozwalają na budowanie aplikacji na wiele różnych platform sprzętowych. Pozwala to na zaoszczędzenie czasu i pieniędzy, ponieważ stworzenie natywnych aplikacji na każdą platformę wymaga innego

podejścia i znajomości wielu technologii. Do rozwiązań cross-platformowych zalicza się również technologia Xamarin. Technologia Xamarin powstała w 2011 roku, a w 2016 została przejęta przez firmę Microsoft.

2. Xamarin

Technologia ta pozwala na tworzenie oprogramowania na wielu urządzeniach z różnymi systemami operacyjnymi. Wykorzystywana jest głównie do tworzenia aplikacji na telefony z systemem Android, Windows Mobile oraz IOS, ale jej możliwości są znacznie większe. Pozwala również na tworzenie aplikacji na telewizory, tablety oraz wszystkie urządzenia wspierane przez technologię UWP [2].

Technologia UWP (ang. Universal Windows Platform) umożliwia tworzenie aplikacji uniwersalnych na platformy o bardzo różnej wydajności sprzętowej oraz rozmiarze ekranu. Możliwe jest tworzenie aplikacji zaczynając od urządzeń wbudowanych aż do wielkich tablic interaktywnych Surface Hub oraz budowanie aplikacji na komputery z systemem Windows [3].

Główną zaletą technologii Xamarin jest możliwość pisania aplikacji z wykorzystaniem języka C# wykorzystując platformę programistyczną .NET. w obu podejściach jakie stosuje technologia Xamarin współdzieli się logikę biznesową, która wykorzystywana jest w każdym urządzeniu, na które buduje się program. Metody biznesowe tworzy się raz i są one dostępne dla każdej platformy [4].

W niniejszej pracy badania skupią się jedynie na urządzeniach mobilnych, wykorzystujących system Android, dlatego też dalsze opisy będą dotyczyły jedynie tego systemu. Są to Xamarin Forms i Xamarin Native.

2.1. Xamarin Forms

Xamarin Forms pozwala nie tylko współdzielić logikę biznesową, ale również graficzny interfejs użytkownika. Wiele platform składa się z bardzo podobnych komponentów, takich jak etykiety, pola tekstowe, przyciski, czy możliwość wyświetlania listy elementów. Xamarin Forms dostarcza wspólnego interfejsu, który zostaje odpowiednio tłumaczony na natywne rozwiązania [5].

Xamarin Forms wykorzystuje bardzo podobny styl tworzenia aplikacji, jak wcześniejsze technologie firmy Microsoft, czyli technologię Universal Windows Platform, czy starszą technologię Windows Presentation Foundation. W podanych rozwiązaniach do tworzenia interfejsu użytkownika wykorzystuje się pliki XAML. Są to pliki XML ze zdefiniowanymi atrybutami służącymi do tworzenia graficznych komponentów. Tak jak w plikach XML, tak pliki XAML mają strukturę drzewiastą, ponieważ dany element może się zawierać w innym elemencie [6].

Oprócz plików XAML, występują tutaj pliki z kodem w języku C#, w którym można tworzyć różne funkcje do programu. Każdy plik XAML jest powiązany z klasą C# odpowiadającą stronie graficznej, w której można w sposób programistyczny zarządzać elementami GUI. w aplikacji tworzonej w podejściu Forms został przyjęty właśnie taki styl pisania aplikacji, który wykorzystuje pliki XAML z powiązanymi plikami C#. Taki sposób pisania programów nosi nazwę wzorca widoku autonomicznego [7].

2.2. Xamarin Native

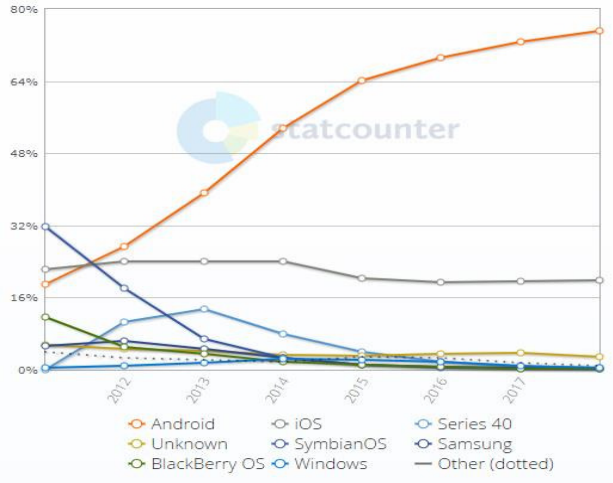
Drugie podejście pozwala na tworzenie natywnych aplikacji, dostosowanych pod każdą platformę z osobna. Nie ma tutaj, jak w przypadku Xamarin Forms, możliwości współdzielenia interfejsu użytkownika, ale cała logika biznesowa może być wykorzystywana na każdej platformie sprzętowej.

Architektura oraz sposób pisania aplikacji jest w zasadzie identyczny, jak podczas tworzenia tej aplikacji w ich macierzystych technologiach. Na systemie Android występują klasy aktywności wraz z ich cyklem życia, pliki XAML z graficznym interfejsem użytkownika, czy też usługi. Różnicą jest jedynie inne środowisko programistyczne oraz wykorzystywany język, wraz z wprowadzonymi konwencjami (np. nazwa metody rozpoczynająca się wielką literą) [8].

Podejście to pozwala pisać aplikacje bardziej złożone niż oferuje Xamarin Forms ze względu na wykorzystywanie API danej platformy, bez konieczności współdzielenia jej z innymi rozwiązaniami. Duża liczba rozwiązań dostępnych na system Android jest możliwa do implementacji podczas korzystania z Xamarin Native.

2.3. Android

Android w ostatnich latach sukcesywnie zdobywał popularność i obecnie jest to najbardziej popularny system operacyjny, wykorzystywany na urządzeniach typu smartfon. Jego udział w rynku to około 75 %, podczas gdy urządzenia z systemem iOS zajmujące drugie miejsce mają udział około 20%. Pozostałe systemy operacyjne straciły już na znaczeniu i stanowią niewielki procent rynku [9].



Rys. 2. Światowy udział systemów operacyjnych na urządzeniach typu smartphone [9]

Aplikacje w tym systemie od zawsze można tworzyć w języku Java, oraz od niedawna w języku Kotlin, stworzonym przez firmę JetBrains.

Android jest systemem dostarczanym na wielu różnych urządzeniach. W maju 2017 r. na konferencji Google I/O ogłoszono, że są ponad 2 miliardy aktywnych urządzeń z systemem Android.

3. Opis badania

Przy porównaniu technologii Xamarin Forms i Xamarin Native zostaną wzięte pod uwagę następujące czynniki:

- Wydajność aplikacji w przypadkach opisanych w rozdziale 3;
- Rozmiar zainstalowanej aplikacji;
- Rozmiar wygenerowanego pliku instalacyjnego apk;
- Liczba linii kodu w projekcie.

W artykule postawiono następujące hipotezy:

- H1: Aplikacje napisane w technologii Xamarin Native są wydajniejsze od tych napisanych w Xamarin Forms pod względem renderowania elementów widoku;
- H2: Aplikacje napisane w technologii Xamarin Forms wymagają porównywalnej liczby linii napisanego kodu, jeżeli rozważana jest tylko jedna platforma.

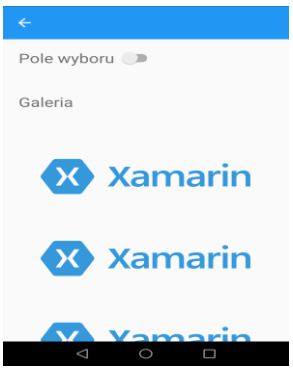
4. Opis aplikacji testowej

W celu przeprowadzenia badań zostały stworzone dwa projekty. Pierwszy z nich został przygotowany w rozwiązaniu Xamarin Forms, a drugi w Xamarin Native. Oba projekty zostały dostosowane jedynie dla systemu Android oraz zostały zbudowane na podstawie pustego szablonu podczas tworzenia projektu, dzięki czemu nie został wygenerowany nadmiarowy kod. Oba projekty posiadają taki sam wygląd i funkcjonalności, dzięki czemu można porównać parametry wydajnościowe oraz zobaczyć różnicę pomiędzy technologiami.

Aplikacja składa się z czterech stron graficznych. Pierwsza to strona powitalna służąca do nawigacji do odpowiednich testów dostępnych na innych ekranach (Rys. 3).

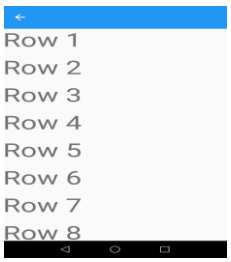
Rys. 3. Widok startowego ekranu testowanej aplikacji

Druga strona (wizytówka) składa się z prostych komponentów graficznych, wykorzystywanych w nowoczesnych aplikacjach mobilnych. Strona ta wyświetla takie elementy jak etykiety tekstów, pola tekstowe, przełączniki, przyciski oraz galerię. Cały ekran został dodatkowo ujęty w widoku ScrollView (element pozwalający na przesuwanie ekranu i podgląd obiektów, które się na nim nie mieszczą) [10]. Zostały tutaj wybrane jedynie komponenty dostępne na wszystkie platformy, żeby w prosty sposób dało się je zaimplementować w Xamarin Forms (Rys. 4).



Rys. 4. Widok ekranu wizytówki testowanej aplikacji, źródło wykorzystanego obrazka [11]

Trzecia strona to widok listy. Zastosowano tutaj kontrolki, które pomagają w pracy z tego typu danymi. w obu podejściach jest to kontrolka ListView (Rys. 5). Czwarta strona jest podobna do strony trzeciej z jedną różnicą - tutaj elementy listy zostały dodawane ręcznie, bez korzystania z wbudowanych komponentów kolekcji. Każdy wiersz listy został utworzony w sposób programistyczny, a następnie przypisany do widoku, tak aby wszystkie układały się wertykalnie (tak samo jak w stronie trzeciej). Dodatkowo całość strony jest skrolowana przez użycie wcześniej wspomnianego elementu ScrollView. Dla stron z listą danych test został przeprowadzony dla 1 000, 10 000 i 100 000 elementów.



Rys. 5. Widok ekranu listy rozwijanej testowanej aplikacji

5. Platforma testowa

Aplikacja została uruchomiona na urządzeniu o parametrach pokazanych w tabeli 1.

Tabela 1. Parametry urzadzenia testowego

Nazwa parametru	Wartość parametru
System operacyjny	Android
Wersja systemu operacyjnego	7.0
Nazwa urządzenia	Honor 8
Procesor CPU	Hisilicon Kirin 950
Pamięć RAM	4,0 GB
Pamięć urządzenia	32 GB
Rozdzielczość	1080 x 1920

6. Wyniki badań

Pierwszym etapem analizy było sprawdzenie wydajności po utworzeniu testowej aplikacji poprzez sprawdzenie czasu potrzebnego na załadowanie strony (Tabela 2). Początkowo testowana była strona wizytówka, a każdy test różnił się od siebie liczbą obrazów umieszczonych wewnątrz widoku.

Tabela 2. Czas ładowania strony wizytówki

Nazwa testu	Czas [s] w Xamarin Native	Czas [s] w Xamrin Forms
Strona wizytówka z 5 obrazkami	0,164	0,405
Strona wizytówka z 50 obrazkami	0,166	0,430
Strona wizytówka z 500 obrazkami	0,204	1,399
Strona wizytówka z 2000 obrazkami	0,317	10,048

Kolejne badanie przeprowadzone było na liście generowanej automatycznie. Lista ta zawierała różną liczbę wierszy w każdym teście. w obu przypadkach wykorzystywany był komponent ListView (Tabela 3).

Tabela 3. Czas ładowania strony z kontrolką ListView

Liczba wierszy	Czas [s] w Xamarin Native	Czas [s] w Xamrin Forms
1 000 wierszy	0,144	0,221
10 000 wierszy	0,400	0,252
100 000 wierszy	2,843	0,464
1 000 000 wierszy	27,832	2,317
10 000 000 wierszy	284,793	21,629

Ostatnie badanie przeprowadzane było również na generowaniu listy elementów, jednak teraz były one dodawane ręczenie (programowo) do layoutu, układającego elementy jeden pod drugim. Wyniki prezentuje tabela 4.

Tabela 4. Czas ładowania strony z generowaną ręcznie listą

Liczba wierszy	Czas [s] w Xamarin Native	Czas [s] w Xamrin Forms
1 000 wierszy	0,389	1,9
10 000 wierszy	2,912	112
50 000 wierszy	-	-
100 000 wierszy	-	-

Powyższa analiza przeprowadzona była również dla 50 000 oraz 100 000 wierszy, jednak czas oczekiwania na listę w podejściu Forms był większy niż pół godziny i badania zostały przerwane.

Analiza rozmiaru aplikacji zarówno po zainstalowaniu, jak przygotowaniu do procesu instalacji w postaci pliku apk.

Ostatni test związany był z liczbą linii kodu potrzebnego do napisania testowych aplikacji. Platforma GitHub, na której przechowywane były projekty w postaci dwóch oddzielnych prywatnych repozytoriów pozwala na odczytanie tych informacji, wliczając w to różne wygenerowane dodatkowo elementy (tabela 6).

Tabela 5. Rozmiar aplikacji

Nazwa testu	Rozmiar [MB] w Xamarin Native	Rozmiar [MB] w Xamrin Forms
Rozmiar aplikacji po utworzeniu projektu w trybie debbuger	19,50	23,42
Rozmiar aplikacji po ukończeniu w trybie debbuger	19,43	23,51
Rozmiar aplikacji po utworzeniu projektu w trybie release	15,30	18,01
Rozmiar aplikacji po ukończeniu w trybie release	15,32	18,11
Rozmiar apk po ukonczeniu aplikacji	14,2	17,5
Rozmiar apk po utworzeniu projektu	14,2	17,5

Tabela 6. Liczba linii kodu potrzebnej do utworzenia aplikacji

Nazwa testu	Liczba linii w Xamarin Native	Liczba linii w Xamrin Forms
Liczba linii kodu po wygenerowaniu projektu	731	7727
Liczba linii kodu po ukończeniu projektu	7247	10029
Liczba linii kodu potrzebna na implementacje	301	198

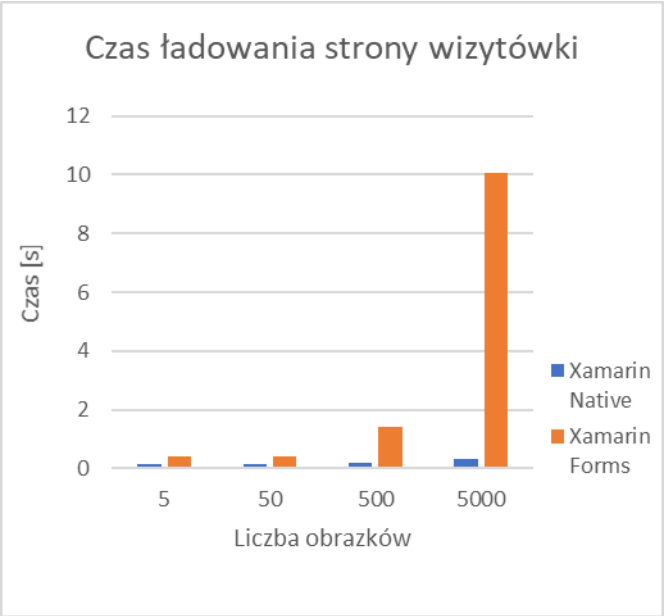
7. Analiza wyników

Pierwszy test polegał na zmierzeniu czasu wczytywania strony zawierającej różną liczbę obrazków. Wyniki jednoznacznie wskazują, że Xamarin Native jest szybszy – zarówno przy małej i dużej liczbie obrazków. Im więcej obrazów, tym większa przewaga. Xamarin Native poradził sobie bez problemu nawet z 2000 obrazków w akceptowalnym czasie – 317 milisekund podczas gdy Xamarin Forms potrzebował ponad 10 sekund (Rys. 6).

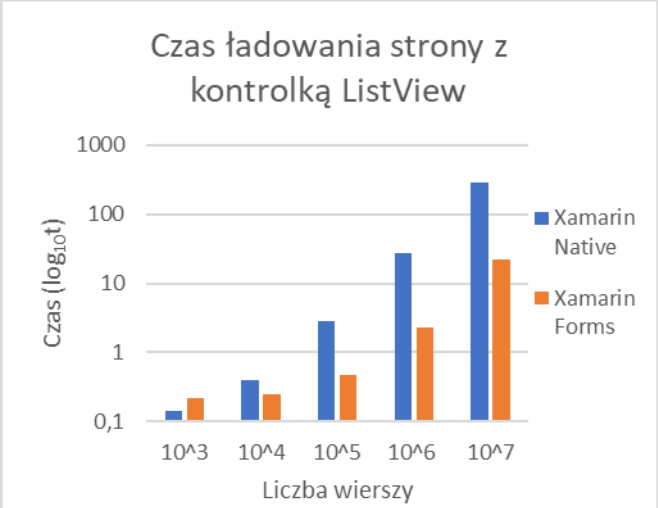
Drugi test polegał na porównaniu szybkości wczytywania listy z danymi. Przy 1000 elementów Xamarin Native okazał się szybszy, natomiast przy większej liczbie elementów znacznie lepiej wypada Xamarin Forms. Należy jednak zaznaczyć, że testowana była kontrolka ListView, która dla natywnej aplikacji androida nie jest już zalecana na rzecz wykorzystywania wydajniejszego komponentu RecyclerView [12] (Rys. 7).

Badania przeprowadzone na potrzeby artykułu „Performance analysis of native and cross-platform mobile applications" [4] również dotyczyły wydajności Xamarin Native i Xamarin Forms, ale skupiono się na innych aspektach aplikacji. Porównano wydajność obliczeniową, czas odczytu plików, szybkość pobierania obrazków oraz czas wyznaczania położenia. Eksperyment nie wskazał jednoznacznie wydajniejszego rozwiązania - Xamarin Native okazało się szybsze w przypadku pobierania obrazków i odczycie oraz zapisie plików, natomiast Xamarin Forms było

szybsze w wyznaczaniu położenia i obliczeniach. Różnice pomiędzy podejściami okazały się mniejsze niż w tym eksperymencie, gdzie wykorzystanie Xamarin Native okazało się znacznie wydajniejsze w większości przeprowadzonych testów [4].

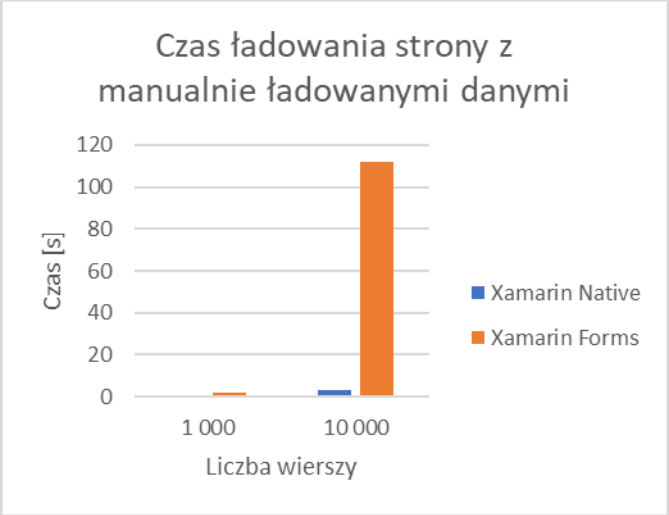


Rys. 6. Wykres pomiaru czasu załadowania strony z wizytówką



Rys. 7. Wykres w skali logarytmicznej pomiaru czasu załadowania strony z kontrolką ListView

Kolejny test polegał na zmierzeniu czasu wczytywania listy przy ręcznym dodawaniu jej elementów. Sposób ten okazał się dużo wolniejszy przy obu podejściach, natomiast znacznie lepiej wypadł Xamarin Native. Przy 1000 wierszy na wczytanie wystarczyło niecałe 400 milisekund, co jest czasem akceptowalnym, natomiast Xamarin Form potrzebował prawie 2 sekundy (Rys. 8). Ze względu na szybki wzrost czasu ładowania wraz ze wzrostem liczby elementów testy dla 50 000 i 100 000 wierszy nie zostały przeprowadzone. Już przy 10 000 wierszy czasy przestały być akceptowalne - w przypadku Xamarin Native były to prawie 3 sekundy, a w przypadku Xamarin Forms prawie 2 minuty. Dziesięciokrotnie większej liczby wierszy nie udało się załadować w obu podejściach, dla Xamarin Native system Android wyłączył aplikację po paru minutach od próby wczytania. Dla Xamarin Forms aplikacja próbowała wczytać dane przez ponad godzinę, jednak bez żadnego skutku - widok nie załadował się. Należy również wspomnieć, że w podejściu Xamarin Forms, dla 100 000 wierszy, podczas przewijania występowały odczuwalne przycięcia, w drugim podejściu nie było takiego problemu.



Rys. 8. Wykres pomiaru czasu załadowania strony z manualnie dodawanymi wierszami

Ostatnie porównanie dotyczyło wykorzystania miejsca w pamięci przez aplikacje. w każdym przypadku rozmiar aplikacji w Xamarin Forms był większy o około 3 MB, co stanowi około 20% więcej zajętej pamięci niż w przypadku aplikacji Xamarin Native.

Mniejsza liczba linii kodu przemawia za podejściem Xamarin Forms, jednak należy zaznaczyć, że tworzenie widoków w XAML jest dużo bardziej przejrzyste i wymaga mniejszej liczby linii kodu. Kontrolki zdefiniowane w podejściu Xamarin Native często wymagają więcej zajętego miejsca, ponieważ atrybuty są dłuższe i należy pisać je jeden pod drugim, aby kod był czytelny. w XAML atrybuty kontrolki nie mają takich długich nazw i można je wypisać po kilka w jednej linijce. Jednak aplikacja ta jest stosunkowo mała, a Xamarin Forms zajął około 100 linijek kodu mniej, co oznacza prawię 1/3 całości.

8. Wnioski

Większość wyników zdecydowania przemawia na korzyść Xamarin Native, zarówno pod względem wydajności, jak i zajmowanej pamięci. Pozwala on na szybsze wyrenderowanie widoków oraz płynniejszą pracę z dużą liczbą komponentów. w Xamarin Forms czas potrzebny na załadowanie danych czasami jest odczuwalny przez użytkownika systemu, a niektóre elementy się przycinają.

Z kolei Xamarin Forms radzi sobie lepiej, jeżeli mowa o szybszym tworzeniu aplikacji. Pozwala on pisać aplikacja z mniejszą ilością kodu, a dodatkowo uruchamiać aplikację na wielu platformach sprzętowych. w tworzeniu aplikacji tylko na jeden dedykowany system nie ma dużej różnicy

pomiędzy wykorzystywanym kodem, ponieważ oba rozwiązania korzystają z tych samych modułów przy wykonywaniu operacji na modelu danych. Różnicą jest natomiast sposób definiowania widoków, jednak i to zależy od sposobu implementacji przez programistę. Natomiast dla wielu platform sprzętowych, różniących się od siebie, Xamarin Forms pozwolił by zredukować znacznie ilość kodu, co zapewniło by szybszą implementację aplikacji.

Literatura

[1] https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/ [22.06.18]

[2] C. Petzold, Creating Mobile Apps with Xamarin.Forms, Microsoft Press, 2016

[3] https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide [23.06.18]

[4] P. Grzmil, M. Skublewska-Paszkowska, E. Łukasik, J. Smołka, Performance analysis of native and cross-platform mobile applications, Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska 2017 T. 7, nr 2, str. 50--53

[5] G. Taskos, Xamarin Cross-Platform Development Cookbook, Packt Publishing, 2016

[6] J. Peppers, G. Taskos, C. Bilgin, Xamarin: Cross-Platform Mobile Application Development, 2016

[7] J. Matulewski, MVVM i XAML w Visual Studio 2015, helion 2016

[8] Mark Reynolds, Xamarin Mobile Application Development for Android, Packt Publishing, 2014

[9] http://gs.statcounter.com/os-market-share/mobile/worldwide/#yearly-2011-2018 [22.06.18]

[10] J. Morris, Android User Interface Development Beginner's Guide, Packt Publishing, 2011

[11] https://ermlab.com/2015/10/22/xamarin-forms-przykladowa-aplikacja/ [22.06.18]

[12] https://docs.microsoft.com/en-us/xamarin/android/user-interface/layouts/list-view/ [22.06.18]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 9 (2018) 324-327
Wysłane: 2018-06-29
Przyjęte: 2018-07-04

Ocena jakości wybranych narzędzi do automatyzacji testów aplikacji

Łukasz Szczepkowicz*, Beata Pańczyk
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Celem niniejszego artykułu jest porównanie wybranych narzędzi do realizacji testów oprogramowania. Do analizy wybrano 7 narzędzi typu open source: Selenium IDE, Selenium WebDriver, JMeter, SoapUI, SikuliX, AutoIT, Katalon Studio. Badania przeprowadzono za pomocą dedykowanych aplikacji testowych (desktopowej, internetowej oraz usługi sieciowej).

Słowa kluczowe: testowanie automatyczne; narzędzia automatyzacji testów

*Autor do korespondencji.

Adres e-mail: lukasz.szczepkowicz@gmail.com

Quality evaluation of selected tools to automate application testing

Łukasz Szczepkowicz*, Beata Pańczyk
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The subject of this article is a comparison of selected tools for the implementation of software tests. Seven open source tools were selected for the analysis: Selenium IDE, Selenium WebDriver, JMeter, SoapUI, SikuliX, AutoIT, Katalon Studio. The research was carried out using dedicated test applications (desktop, web and web service) with the same functionality.

Keywords: automatic testing; test automation tools

*Corresponding author.

E-mail address: lukasz.szczepkowicz@gmail.com

1. Wstęp

Aplikacje internetowe, webowe czy mobilne są integralną częścią życia człowieka ale większość użytkowników spotkało się przynajmniej raz z niepoprawnym ich działaniem. Aby zapewnić wysoką jakość oprogramowania jego producenci przed wydaniem produktu przeprowadzają testy aplikacji, dzięki czemu minimalizują ryzyko awarii systemu. Coraz większe znaczenie ma automatyzacja testów, która staje się niezastąpionym elementem testowania oprogramowania, minimalizując jednocześnie czas i koszty wytworzenia gotowego produktu.

Niniejszy artykuł został poświęcony porównaniu programów do automatyzacji testów różnych aplikacji, dostępnych w postaci rozwiązań typu open source. Analizy i oceny dokonano dla 7 najpopularniejszych:

- Selenium IDE,
- Selenium WebDriver,
- JMeter,
- SoapUI,
- SikuliX,
- AutoIT,
- Katalon Studio.

2. Kryteria porównania

W celu porównania narzędzi zostały zaimplementowane dedykowane aplikacje testowe w postaci aplikacji desktopowej, internetowej oraz usługi sieciowej (Web service). Każda aplikacja posiada taką samą złożoność (liczbę ekranów, liczbę pól na ekranie itp.). Ocena narzędzi została wystawiona na podstawie następujących kryteriów:

- łatwość instalacji,
- łatwość obsługi,
- wsparcie techniczne,
- szybkość działania,
- wady oraz zalety,
- umiejętności potrzebne do pracy z danym narzędziem,
- dodatkowe czynniki wpływające na ocenę narzędzia.

3. Ocena narzędzi

3.1. Selenium IDE

Narzędzie Selenium IDE to zintegrowane oprogramowanie będące dedykowaną wtyczką dla przeglądarki Mozilla Firefox, służące do przeprowadzania testów funkcjonalnych aplikacji webowych. Narzędzie to jest łatwe w użyciu z uwagi na dołączoną funkcjonalność nagrywania testu, dlatego też może być odpowiednie dla początkujących testerów [1].

Selenium IDE jest wtyczką do przeglądarki dlatego jego instalacja nie jest problematyczna. Zintegrowane narzędzie do nagrywania testu znacznie upraszcza jego obsługę.

Narzędzie nie posiada wsparcia technicznego producenta z uwagi na to, że jest to rozwiązanie typu open source.

Jednakże jest to popularne narzędzie, w Internecie nie jest trudno znaleźć informacje na jego temat bądź uzyskać wsparcie innych użytkowników.

Jest to bardzo proste narzędzie i celowo szybkość jego działania została obniżona. Jednakże przeglądarka (na której uruchamia się test), jest włączona przed jego uruchomieniem, dlatego nie traci się czasu uruchamianie samej przeglądarki.

Narzędzie nie wymaga umiejętności programowania lub specjalnej wiedzy informatycznej. Obsługuje tylko jedną przeglądarkę, na której zostało zainstalowane, dlatego za jego pomocą nie jest możliwe zrealizowanie profesjonalnych testów automatycznych. Wady oraz zalety narzędzia przedstawia tabela 1.

Tabela 1. Wady i zalety Selenium IDE

Wady	Zalety
Brak możliwości wyboru daty z rozwijalnego kalendarza, Brak możliwości dodania pliku, Skrypt powstały poprzez narzędzie do rejestracji akcji jest często konwertowany niewłaściwie.	Łatwe w obsłudze, Wbudowane narzędzie do rejestrowania akcji, Możliwość eksportu skryptu na różne języki programowania (java, php).

3.2. Selenium Web Driver

Selenium WebDriver to powszechnie stosowany framework do testów automatycznych na przeglądarkach internetowych. Skrypty testowe pisane są za pomocą środowiska programistycznego, dlatego też możliwe jest połączenie bibliotek Selenium z wieloma dodatkowymi narzędziami, przez co jego możliwości są ogromne. Selenium dostępne jest dla kilku popularnych języków programowania takich jak: Java, C#, Ruby, Python [1, 2].

Selenium WebDriver nie jest łatwy w instalacji. Wymagana jest znajomość narzędzia maven lub obsługa środowiska programistycznego. Nie jest też łatwe w obsłudze - niezbędne jest posiadanie umiejętności programowania na poziomie minimum podstawowym.

Nie ma wsparcia technicznego producenta z uwagi na to, że jest to rozwiązanie open source. Jednakże jest to popularne narzędzie, dlatego też w Internecie nie jest trudno znaleźć informacje na jego temat bądź uzyskać wsparcie innych użytkowników.

Selenium WebDriver to biblioteki dedykowane dla wybranego języka programowania. Najdłużej podczas testu trwa uruchamianie przeglądarki. Sam test jest szybki i często wymaga dodania czasów oczekiwania, aby strona internetowa zdołała się prawidłowo wczytać.

Narzędzie wymaga umiejętności programowania. Przy bardziej złożonych testach wymagane jest również znajomość obsługi baz danych. Narzędzie to nie jest łatwe w obsłudze ale z uwagi na swoją specyfikę możliwe jest jego integracja z takimi narzędziami jak maven, git, Jenkins, TestNG + TestNG Reporting, AutoIT czy Apache POI. Dzięki temu możliwe jest stworzenie profesjonalnych testów automatycznych. Wady oraz zalety narzędzia przedstawia tabela 2.

Tabela 2. Wady i zalety Selenium WebDriver

Wady	Zalety
Trudne w utrzymaniu testy z dużą liczbą kroków, Wymagana umiejętność programowania i obsługi środowiska programistycznego, Przymusowe dodawanie czasu oczekiwania z powodu zbyt szybkiego działania narzędzia.	Testy dla wielu przeglądarek, Możliwość integracji z innymi narzędziami testowymi, Niezależny od języka programowania.

3.3. Apache JMeter

Jest to program w języku java, dedykowany do tworzenia, wykonywania oraz monitorowania przebiegu testów obciążeniowych, przeciążeniowych oraz wydajnościowych (częściowo również testów funkcjonalnych). Z powodzeniem wykorzystywany jest do symulowania i badania wzmożonego ruchu na serwerach, grupie serwerów oraz w sieci [3, 4].

JMeter jest łatwy w instalacji. Wystarczy rozpakować paczkę zip i uruchomić plik z programem. Niestety nie jest łatwy w obsłudze - niezbędna jest znajomość protokołów sieciowych.

JMeter nie posiada wsparcia technicznego producenta (jest darmowe). W Internecie natomiast można znaleźć informacje oraz wsparcie innych użytkowników.

JMeter przeznaczony jest do badania wydajności, obciążenia oraz przeciążenia. Nie jest możliwe porównanie jego szybkości działania z innymi testowanymi aplikacjami z uwagi na charakter narzędzia, dlatego też szybkość działania nie była oceniana. Wady oraz zalety narzędzia przedstawia tabela 3.

Tabela 3. Wady i zalety JMeter

Wady narzędzia	Zalety narzędzia
Wymagane realistyczne odwzorowanie ustawień klient/server, Ograniczenia maszyny testowej mogą pojawić się w opóźnieniach w czasach odpowiedzi, Narzędzie zużywa wiele zasobów.	Możliwość testowania wielu typów systemów, Duża skalowalność dla zwiększenia zakresu obciążenia.

3.4. Soap UI

SoapUI jest idealnym narzędziem wspomagającym testy funkcjonalne usług sieciowych. Możliwości tego narzędzia wykraczają poza sam protokół SOAP, ponieważ pozwalają na testowanie aplikacji wykorzystujących takie technologie jak REST, JMS, AMF oraz JDBC. Dodatkowo SoapUI korzysta z języka Groovy przez co możliwa jest jego rozbudowa [5, 6].

SoapUI jest łatwy w instalacji i w obsłudze. Realizacja prostego testu jak w przypadku testowanej aplikacji Web-serwice nie jest skomplikowana. Bardziej złożone testy

wymagają jednak większej znajomości narzędzia oraz sposobów komunikacji REST oraz SOAP.

SoapUI nie posiada wsparcia technicznego. W Internecie natomiast można znaleźć informacje na jego temat oraz wsparcie innych użytkowników.

Narzędzie przeznaczone jest do testów Web-service. Z uwagi na charakter testowanych systemów, działanie narzędzia jest błyskawiczne. Wymaga znajomości sposobów komunikacji REST oraz SOAP. W wersji PRO (wersja płatna) posiada dodatkowe funkcjonalności umożliwiające testowanie wydajności. Niestety w wersji darmowej opcja ta nie jest dostępna. Wady i zalety narzędzia przedstawia tabela 4.

Tabela 4. Wady i zalety SoapUI

Wady narzędzia	Zalety narzędzia
Własny wbudowany klient więc testy nie odzwierciedlają standardowego użycia Web-service, Testy SoapUI nie zastąpią testów funkcjonalnych aplikacji.	Możliwość testowania aplikacji typu Web-service, Używanie narzędzia nie wymaga znajomości programowania, Automatyczne generowanie zapytań w oparciu o plik WSDL, Wbudowane asercje.

3.5. SikuliX

Działanie SikuliX polega na sprawdzeniu czy zdefiniowany wycinek ekranu jest dostępny na ekranie komputera. W przypadku jego znalezienia możliwe jest podjęcie na nim akcji takiej jak kliknięcie elementu czy przekazanie tekstu. SikuliX jest dostępny w postaci programu desktopowego, ale również jako biblioteki, dzięki czemu możliwe jest programowanie testów za jego pomocą [7].

Instalując narzędzie SikuliX należy wiedzieć jaki rodzaj instalacji jest potrzebny (dostępne są różne instalacje produktu). Jest łatwy w obsłudze jeśli wybrano aplikację typu desktop. Wersja bibliotek wykonywalnych wymaga jednak umiejętności programowania, tak jak w przypadku Selenium WebDriver. SikuliX nie posiada wsparcia technicznego producenta. W Internecie także mało jest informacji na jego temat. SikuliX nie jest szybkim narzędziem, ale samo działanie testów jest efektywne.

Narzędzie wymaga umiejętności programowania tylko jeśli użyto bibliotek wykonywalnych SikuliX. Aplikacja typu desktop nie wymaga umiejętności programowania lub specjalnej wiedzy informatycznej aby go używać. Z uwagi na GUI - narzędzie sprawdza się dla aplikacji desktopowych i internetowych. Wady oraz zalety narzędzia przedstawia tabela 5.

Tabela 5. Wady i zalety SikuliX.

Wady narzędzia	Zalety narzędzia
Testy wykonywane są powoli, Podobne wycinki ekranu mogą być błędnie interpretowane, Brak wsparcia technicznego producenta oraz informacji w Internecie.	Możliwość testowania aplikacji z GUI, Bardzo łatwa obsługa narzędzia typu desktop.

3.6. AutoIT

AutoIT to język zaprojektowany do automatyzowania aplikacji na systemach Microsoft Windows już w 1999 roku. Jest rozwijany do dziś przez AutoIt Team. Aktualna wersja języka (v3.3.14.5) ma składnię podobną np. do Visual Basic czy JavaScript [8, 9].

AutoIT nie jest trudne w instalacji. Sskłada się z kilku komponentów wchodzących w skład jednego narzędzia. Aby prawidłowo ich używać należy zapoznać się z wszystkimi oferowanymi komponentami. Dodatkowo skrypty testowe są pisane w specjalnym języku podobnym do Basic co jest niewielkim utrudnieniem. Nie posiada wsparcia technicznego producenta. W Internecie natomiast mało jest informacji związanych z tym narzędziem.

AutoIT działa szybko. Skrypt może działać bez potrzeby uruchamiania któregoś z komponentów AutoIT. Wymaga znajomości języka skryptowego w przypadku realizacji bardziej skomplikowanych testów. Jest dostępne jako zestaw aplikacji typu desktop oraz bibliotek wykonywalnych. Wady oraz zalety narzędzia przedstawia tabela 6.

Tabela 6. Wady i zalety AutoIT

Wady narzędzia	Zalety narzędzia
Narzędzie korzysta z własnego języka skryptowego, Dostępny tylko dla systemu Windows, Brak wsparcia technicznego producenta oraz informacji w Internecie.	Możliwość testowania aplikacji desktopowych, Dodatkowe narzędzie do rejestrowania testu z którego następnie budowany jest skrypt testowy,

3.7. Katalon Studio

Katalon Studio to darmowe rozwiązanie do testów automatycznych. Jest oparte na frameworkach Selenium oraz Appium. Pierwsze publiczne wydanie narzędzia odbyło się we wrześniu 2016 roku, więc jest to stosunkowo nowe rozwiązanie [10].

Jest łatwe w instalacji. Do jego używania potrzebne jest jednak założenie konta na stronie producenta https://www.katalon.com/. Katalon Studio nie jest trudny w obsłudze. Posiada intuicyjny interfejs a same składowe testów są podzielone w osobnych modułach. Narzędzie mimo to jest bardzo rozbudowane.

Katalon Studio został utworzony przez firmę Katalon LLC we wrześniu 2016 roku. Na oficjalnej stronie programu znajdują się poradniki i filmy jak korzystać z narzędzia. Dodatkowe, bardziej zaawansowane wsparcie techniczne może być płatne. Poza oficjalną stroną w Internecie nie ma o nim zbyt wielu informacji. Katalon Studio działa ze średnią prędkością w odniesieniu do innych weryfikowanych narzędzi. Narzędzie nie wymaga umiejętności programowania. W przypadku jednak testów aplikacji mobilnych dobrze jest znać technologie mobilne jak np. Android natomiast w przypadku usług sieciowych - sposobów komunikacji z serwerem. Narzędzie pozwala grupować przypadki testowe w zestawy oraz przechowuje oddzielnie obiekty interpretujące

elementy strony, dlatego też może służyć do realizacji profesjonalnych testów automatycznych. Wady oraz zalety narzędzia przedstawia tabela 7.

Tabela 7. Wady i zalety Katalon studio.

Wady narzędzia	Zalety narzędzia
Mało popularne narzędzie, Płatne dodatkowe wsparcie techniczne,	Możliwość testowania na wielu przeglądarkach, Możliwość testowania aplikacji mobilnych, webowych oraz Web-service.

3.8. Zestawienie wyników

W tabeli 8 przedstawiono wyniki przeprowadzonej analizy. Poszczególne kryteria uzyskały oceny w skali 1-5 (gdzie 5 jest oceną najwyższą). W przypadku konieczności posiadania dodatkowych umiejętności – odejmowany jest 1 lub 2 punkty. Dodatkowe czynniki natomiast oceniano uwzględniając ich pozytywny bądź negatywny wpływ na oceniane narzędzie. Ilość punktów ujemnych lub dodatnich ustalono na podstawie istotności badanego czynnika.

Tabela 8. Ocena wybranych narzędzi

Ocena	Selenium IDE	Selenium WebDriver	JMeter	SoapUI	SikuliX	AutoIT	Katalon Studio
Łatwość instalacji	5	3	5	5	4	4	4
Łatwość obsługi	5	2	3	4	4	3	4
Wsparcie techniczne	4	4	3	3	2	2	4
Szybkość działania	4	5	brak	5	3	5	4
Dodatkowe umiejętności	0	-2	-1	-1	0	-1	-1
Dodatkowe czynniki	-2	5	3	0	2	2	3
Ocena końcowa	**16**	**17**	**14**	**16**	**15**	**15**	**18**

4. Wnioski

Na podstawie przeprowadzonych badań na bazie trzech różnych aplikacji testowych stwierdzono, że nie ma uniwersalnego narzędzia testowego, umożliwiającego kompleksowe testy automatyczne. Wybór narzędzi musi być przemyślany i dostosowany do konkretnego rodzaju aplikacji.

Automatyzacja testów jest niezbędna, szczególnie w przypadku złożonych systemów. Istnieje wiele narzędzi do wyboru, również tych darmowych, analizowanych w niniejszym artykule. Niestety nie jest możliwe przetestowanie, za pomocą jednego narzędzia, różnych rodzajów systemów lub wykonanie różnych rodzajów testów.

Te bardziej zaawansowane narzędzia automatyzacji testów wymagają od użytkownika dodatkowych umiejętności np. programowania, obsługi komunikacji z serwerem, znajomości algorytmów itp. Są również narzędzia, z których można korzystać bez specjalistycznej wiedzy np. SikuliX czy Selenium IDE. Takie narzędzia są najbardziej odpowiednie dla osób, które dopiero zaczynają przygodę z automatyzacją testów. Stosowanie bardziej złożonych narzędzi takich jak Katalon Studio czy Selenium WebDriver wymaga większego zaangażowania, ale testy napisane za ich pomocą są lepsze, bardziej profesjonalne i efektywne.

Każde z zaprezentowanych w artykule narzędzi, posiada opisane wcześniej wady i zalety. Ocena końcowa została wystawiona na podstawie indywidualnych przemyśleń i doświadczeń, więc każdy użytkownik może wykorzystać ten materiał jako wskazówkę, które z nich będzie najlepszym wyborem do realizacji testów różnego rodzaju aplikacji.

Literatura

[1] Sams P.: Selenium. Automatyczne testowanie aplikacji. Wydawnictwo Helion, 2017.

[2] Gundecha U.: Selenium i testowanie aplikacji. Receptury. Wydawnictwo Helion, 2017.

[3] Halili E. H.: Apache JMeter. Wydawnictwo Packt Publishing, 2008.

[4] https://www.youtube.com/watch?v=mgTE8it2KvA [15.06.2018]

[5] https://www.youtube.com/watch?v=izA_Xgbj7II [15.06.2018]

[6] http://cezarywalenciuk.pl/blog/programing/post/soapui--program-do-testowania-uslug-soap-i-rest [06.2018]

[7] https://www.youtube.com/watch?annotation_id=annotation_1950573397&feature=iv&index=2&list=PL1A2CSdiySGJJNe3WzCezcI7by5SPfJTS&src_vid=VdCOg1bCmGo&v=I-UYoezac4Q [06.2018]

[8] https://www.youtube.com/watch?v=3RmxYAxhzl0 [06.2018]

[9] https://pl.wikipedia.org/wiki/AutoIt [06.2018]

[10] https://en.wikipedia.org/wiki/Katalon_Studio [06.2018]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 9 (2018) 328-333
Wysłane: 2018-09-10
Przyjęte: 2018-09-14

Analiza porównawcza narzędzi typu Front-End Code Playground

Mateusz Magier*, Beata Pańczyk
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Niniejszy artykuł zawiera analizę porównawczą wybranych narzędzi typu Front-End Code Playground. Zawarte tabele opracowane zostały na podstawie rezultatów testów rozważanych aplikacji i analizy ich dokumentacji. Porównanie zrealizowane zostało pod kątem kilku kryteriów (m.in.: cech edytorów, niestandardowych trybów tworzenia projektów, wsparcia dla preprocesorów i metod eksportu projektu). Wnioski wyciągnięte na podstawie porównania pozwoliły na podzielenie rozpatrywanych narzędzi na trzy grupy (aplikacje o: wysokim, umiarkowanym i niskim poziomie zaawansowania).

Słowa kluczowe: frontend; aplikacje internetowe; CSS; JavaScript

*Autor do korespondencji.

Adres e-mail: mateusz.magier@pollub.edu.pl

Comparative analysis of Front-End Code Playground tools

Mateusz Magier*, Beata Pańczyk
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. This article contains comparative analysis of selected Front-End Code Playground tools. The included tables were performed on the results of tests of the considered applications and the analysis of their documentation. The comparison was made for several criteria (including: features of code editors, custom modes of creating projects, support for preprocessors and project export methods). The conclusions drawn from the comparison allowed dividing the considered tools into three groups (applications with: high, moderate and low level of advancement).

Keywords: frontend; web applications; CSS; JavaScript

*Corresponding author.

E-mail address: mateusz.magier@pollub.edu.pl

1. Wstęp

Według rezultatów corocznej ankiety – przeprowadzanej wśród członków społeczności StackOverflow - najpopularniejszymi technologiami programistycznymi w 2018 roku są odpowiednio: JavaScript (przez sześć lat z rzędu na pierwszym miejscu w zestawieniu), HTML oraz CSS [1]. Programiści korzystający z wyżej wymienionych języków nierzadko stoją przed dylematem związanym w wyborem narzędzia programistycznego. Rozwój aplikacji internetowych na przestrzeni lat przyczynił się do tego, że oprogramowanie desktopowe nie jest już jedyną opcją w kwestii środowisk służących do tworzenia kodu w HTML, CSS i JavaScript. Istnieją aplikacje spełniające to zadanie, które działają za pośrednictwem przeglądarki internetowej. Aplikacje te nazywane są „narzędziami typu Front-End Code Playground" (FECP).

Celem niniejszego artykułu jest przeprowadzenie analizy porównawczej wybranych aplikacji typu FECP. Opisane zostaną kryteria, względem których przeprowadzone zostanie porównanie. Rezultaty analizy zostaną zawarte w tabelach, które mogą okazać się pomocne dla osób stojących przed wyborem oprogramowania typu FECP spełniającego ich wymagania.

Ponieważ narzędzia typu Front-End Code Playground są właściwie nieobecne w literaturze fachowej, niniejszy artykuł stanowi również próbę uporządkowania informacji na ten temat. Przeprowadzone badania literaturowe wykazały, że tematyka ta jest poruszana jedynie w źródłach internetowych (na podstawie tych źródeł dokonano wyboru aplikacji będących przedmiotem porównania) [2-4]. Oprócz tego natrafiono na jedną próbę przeprowadzenia porównania narzędzi typu FECP [5].

Do przeprowadzanych testów wybrano dwanaście aplikacji typu FECP. Należą do nich: CodePen [6], CSSDeck [7], CSSDesk [8], Dabblet [9], JS Bin [10], JSFiddle [11], Kodnest [12], Liveweave [13], Pinocode [14], Plunker [15], SoloLearn [16] oraz Web Maker [17]. Analiza porównawcza została przeprowadzona za pomocą gruntownych testów wyżej wymienionych aplikacji (które zweryfikowały poszczególne kryteria porównania), a także przestudiowania ich dokumentacji.

2. Narzędzia typu Front-End Code Playground

Główna różnica między aplikacjami typu FECP a oprogramowaniem desktopowym polega na tym, że ten pierwszy rodzaj narzędzi działa za pośrednictwem przeglądarki internetowej. Aplikacja taka może działać w trybie online lub offline.

Większość tego typu narzędzi udostępnia użytkownikowi edytory HTML, CSS i JavaScript, dodatkowym elementem bywa również konsola JavaScript. Ważną cechą aplikacji typu FECP jest automatyczne odświeżanie widoku (tzw. *Live Preview*) [18]. Dostępne edytory często wyposażone są w szereg cech usprawniających proces tworzenia kodu (np. wsparcie dla wtyczki Emmet oraz sprawdzanie poprawności kodu). Niektóre narzędzia pozwalają na wykorzystanie preprocesorów, a także na dodawanie do projektu zasobów CSS i JavaScript. Niewątpliwie ciekawą funkcjonalnością jest możliwość rozpoczęcia tworzenia projektu od gotowego szablonu, bazującego na wybranej bibliotece CSS/JavaScript (tzw. *boilerplate*) [19].

Niektóre narzędzia udostępniają użytkownikom niestandardowe tryby tworzenia projektów, które pozwalają np. na równoczesną pracę wielu osób nad tym samym kodem [20]. Wiele aplikacji posiada opcję udostępniania i osadzania tworzonego kodu. Kolejną przydatną funkcjonalnością jest eksport projektu, dzięki czemu użytkownik może zapisać rezultaty swojej pracy np. w postaci pliku JSON lub ZIP.

Niektóre aplikacje dysponują bazą publicznych projektów użytkowników (członkowie społeczności mogą uzyskać dostęp do rezultatów pracy innych osób). Dość powszechną opcją jest również możliwość sklonowania kodu innego użytkownika i rozwijania go na swój sposób (tzw. *fork*) [21].

3. Analiza porównawcza narzędzi

3.1. Tryb działania

Pierwszą cechą stanowiącą kryterium porównania aplikacji jest tryb działania – narzędzie może działać w trybach: online i/lub offline. W trybie online użytkownik musi posiadać dostęp do Internetu, aby korzystać z oprogramowania. Natomiast w trybie offline nie jest wymagane połączenie z Internetem – narzędzie funkcjonuje jako rozszerzenie przeglądarki internetowej. Dostępność trybów dla poszczególnych narzędzi przedstawiono w tabeli 1.

Tabela 1. Zestawienie dostępnych trybów działania

Narzędzie	Tryb działania	
	online	offline
CodePen	tak	nie
CSSDeck	tak	nie
CSSDesk	tak	nie
Dabblet	tak	nie
JS Bin	tak	nie
JSFiddle	tak	nie
Kodnest	tak	nie
Liveweave	tak	nie
Pinocode	nie	tak
Plunker	tak	nie
SoloLearn	tak	nie
Web Maker	tak	tak

3.2. Cechy edytorów kodu

Kolejne kryterium dotyczy cech edytorów kodu w rozważanych aplikacjach typu FECP. Rozpatrywane cechy dotyczą: wsparcia dla Emmet (będącego rozszerzeniem edytorów HTML i CSS, umożliwiającego stosowanie skróconej notacji), kolorowania składni (wyróżniania elementów składowych kodu za pomocą np. koloru), wyświetlania podpowiedzi (autouzupełniania słów kluczowych), podświetlania dopasowanych nawiasów, możliwości zawijania kodu (pozwalającego na uzyskanie lepszej czytelności zagnieżdżonego kodu), skrótów klawiaturowych, numerowania linii w edytorach, sprawdzania poprawności składni oraz wyświetlania komunikatów o błędach. Wsparcie dla wybranych funkcjonalności edytorów w poszczególnych narzędziach przedstawiono w tabeli 2.

Tabela 2. Wsparcie dla wybranych funkcjonalności edytorów w narzędziach typu FECP

Narzędzie	Funkcjonalność edytora								
	Emmet	Kolorowanie składni	Podpowiedzi	Brace matching	Zawijanie kodu	Skróty klawiaturowe	Numerowanie linii	Sprawdzanie składni	Komunikaty o błędach
CodePen	tak	tak	tak	tak	tak	tak	tak	tak	tak
CSSDeck	tak	tak	nie	nie	nie	tak	tak	HTML, CSS	nie
CSSDesk	nie	tak	nie	CSS	nie	nie	tak	nie	nie
Dabblet	nie	tak	nie	nie	nie	tak	poz. kursora	nie	nie
JS Bin	tak	tak	JS	tak	tak	tak	tak	CSS, JS	JS
JSFiddle	tak	tak	tak	tak	tak	tak	tak	tak	JS
Kodnest	tak	tak	CSS	CSS, JS	tak	tak	tak	nie	nie
Liveweave	tak	tak	tak	JS	nie	tak	tak	tak	CSS, JS
Pinocode	tak	tak	tak	tak	tak	tak	tak	CSS, JS	JS
Plunker	nie	tak	tak	tak	tak	tak	tak	tak	tak
SoloLearn	nie	tak	tak	tak	tak	tak	tak	tak	tak
Web Maker	tak	tak	tak	tak	tak	tak	tak	CSS, JS	JS

3.3. Tryby tworzenia i prezentowania projektów

Trzecim kryterium porównawczym jest dostępność niestandardowych trybów tworzenia i prezentowania projektów. Przeprowadzone testy pozwoliły na zidentyfikowanie trzech niestandardowych trybów. Pierwszym z nich jest tryb współpracy (tzw. *Collab Mode*), udostępniający możliwość równoczesnej pracy wielu osób nad tym samym kodem (aktualizacja zawartości edytorów dokonuje się automatycznie u wszystkich użytkowników w grupie, gdy stan projektu ulegnie zmianie).

Kolejny tryb – nauczania (w zależności od narzędzia określany jako *Professor Mode* [22] lub *Codecasting* [23]) – jest modyfikacją trybu współpracy. W grupie korzystającej wspólnie z narzędzia dokładnie jedna osoba może edytować kod – pozostali członkowie otrzymują projekt tylko do

odczytu. Ostatnim trybem jest tryb prezentacji, który dostosowuje wygląd i funkcjonalność narzędzia do sytuacji, w której kod jest np. wyświetlany za pomocą projektora dużej grupie odbiorców (jedną z jego cech jest ograniczenie widocznych komponentów interfejsu narzędzia do niezbędnego minimum). Wsparcie dla trybów niestandardowych w poszczególnych narzędziach przedstawiono w tabeli 3.

Tabela 3. Dostępność trybów niestandardowych w narzędziach typu FECP

Narzędzie	Tryb niestandardowy		
	współpracy	nauczania	prezentacji
CodePen	tak (premium)	tak (premium)	tak (premium)
CSSDeck	nie	nie	nie
CSSDesk	nie	nie	nie
Dabblet	nie	nie	nie
JS Bin	nie	tak	nie
JSFiddle	tak	nie	nie
Kodnest	nie	nie	nie
Liveweave	tak	nie	nie
Pinocode	nie	nie	nie
Plunker	nie	nie	nie
SoloLearn	nie	nie	nie
Web Maker	nie	nie	nie

3.4. Przeglądanie projektów innych użytkowników

Narzędzia typu FECP można również porównać pod względem możliwości przeglądania kodu utworzonego przez innych użytkowników aplikacji. Przeprowadzone testy pozwoliły wyróżnić dwa sposoby dostępu do publicznych projektów. Pierwszym z nich jest globalne przeglądanie projektów, realizowane zazwyczaj za pomocą podstrony istniejącej w obrębie serwisu, która zawiera bazę projektów innych użytkowników. Podstrona ta często zawiera interfejs pozwalający na filtrowanie projektów (najpopularniejsze, najwyżej oceniane, najnowsze). Drugim sposobem jest przeglądanie projektów z poziomu konta użytkownika (zawierającej kod utworzony przez danego użytkownika). Listy publicznych projektów często wyposażone są w informacje o opinii innych członków społeczności (wyrażanych za pomocą metryk takich jak: liczba wyświetleń, liczba polubień, oraz liczba komentarzy). Dostępność tej funkcjonalności dla poszczególnych narzędzi przedstawiono w tabeli 4.

Tabela 4. Dostęp do projektów innych użytkowników

Narzędzie	Przeglądanie projektów	
	globalnie	z konta użytkownika
CodePen	tak	tak
CSSDeck	tak	tak
CSSDesk	nie	nie
Dabblet	nie	nie
JS Bin	nie	nie
JSFiddle	nie	nie
Kodnest	tak	tak
Liveweave	nie	nie
Pinocode	nie	nie
Plunker	tak	tak
SoloLearn	tak	tak
Web Maker	nie	nie

3.5. Dostępność narzędzi rozszerzających HTML, CSS i JavaScript

Niektóre narzędzia typu FECP udostępniają użytkownikom możliwość tworzenia kodu z wykorzystaniem preprocesorów i innych narzędzi (wielu programistów do tworzenia kodu frontend wykorzystuje właśnie preprocesory) [24]. Wsparcie dla preprocesorów i narzędzi HTML, CSS i JavaScript w poszczególnych narzędziach przedstawiono w tabelach 5-7.

Tabela 5. Dostępność preprocesorów HTML w poszczególnych aplikacjach

Narzędzie	Preprocesor HTML			
	Haml	Markdown	Slim	Pug
CodePen	tak	tak	tak	tak
CSSDeck	nie	nie	nie	nie
CSSDesk	nie	nie	nie	nie
Dabblet	nie	nie	nie	nie
JS Bin	nie	tak	nie	tak
JSFiddle	nie	nie	nie	nie
Kodnest	tak	nie	nie	tak
Liveweave	nie	nie	nie	nie
Pinocode	nie	tak	nie	tak
Plunker	nie	nie	nie	nie
SoloLearn	nie	nie	nie	nie
Web Maker	nie	tak	nie	tak

Tabela 6. Dostępność narzędzi rozszerzających CSS w poszczególnych aplikacjach typu FECP

Narzędzie	Preprocesor/narzędzie CSS						
	Less	Sass		Stylus	Myth	PostCSS	Atomizer
		SCSS	SASS				
CodePen	tak	tak	tak	tak	nie	tak	nie
CSSDeck	nie	nie	nie	nie	nie	nie	nie
CSSDesk	nie	nie	nie	nie	nie	nie	nie
Dabblet	nie	nie	nie	nie	nie	nie	nie
JS Bin	tak	tak	tak	tak	tak	nie	nie
JSFiddle	nie	tak	nie	nie	nie	nie	nie
Kodnest	tak	tak	tak	tak	nie	nie	nie
Liveweave	nie	nie	nie	nie	nie	nie	nie
Pinocode	tak	tak	tak	tak	nie	nie	tak
Plunker	nie	nie	nie	nie	nie	nie	nie
SoloLearn	nie	nie	nie	nie	nie	nie	nie
Web Maker	tak	tak	tak	tak	nie	nie	tak

Tabela 7. Dostępność narzędzi rozszerzających JavaScript w poszczególnych aplikacjach typu FECP

Narzędzie	Preprocesor/narzędzie JavaScript					
	CoffeeScript	LiveScript	TypeScript	Babel	Traceur	ClojureScript
CodePen	tak	tak	tak	tak	nie	nie
CSSDeck	nie	nie	nie	nie	nie	nie
CSSDesk	nie	nie	nie	nie	nie	nie
Dabblet	nie	nie	nie	nie	nie	nie
JS Bin	tak	tak	tak	tak	tak	tak
JSFiddle	tak	nie	tak	tak	nie	nie
Kodnest	nie	nie	tak	tak	nie	nie
Liveweave	nie	nie	nie	nie	nie	nie
Pinocode	tak	nie	tak	tak	nie	nie
Plunker	nie	nie	nie	nie	nie	nie
SoloLearn	nie	nie	nie	nie	nie	nie
Web Maker	tak	nie	tak	tak	nie	nie

3.6. Metody eksportowania projektów

Kolejnym kryterium porównawczym jest sposób, w jaki użytkownik może wyeksportować utworzony kod. Do zidentyfikowanych metod należą: plik ZIP, plik HTML, plik JSON (format pozwalający na zapisanie nie tylko zawartości edytorów, ale także ustawień projektu) oraz sklonowanie projektu do narzędzia CodePen. W tabeli 8. zaprezentowano dostępność rozpatrywanych metod eksportu w poszczególnych narzędziach typu FECP.

Tabela 8. Dostępność metod eksportu projektu w poszczególnych aplikacjach

Narzędzie	Metoda eksportu projektu			
	Plik ZIP	Plik HTML	Plik JSON	CodePen
CodePen	tak	nie	nie	nie
CSSDeck	nie	nie	nie	nie
CSSDesk	nie	tak	nie	nie
Dabblet	nie	nie	nie	nie
JS Bin	nie	tak	nie	nie
JSFiddle	nie	nie	nie	nie
Kodnest	tak	tak	nie	nie
Liveweave	tak	tak	nie	nie
Pinocode	nie	tak	tak	tak
Plunker	tak	nie	nie	nie
SoloLearn	nie	nie	nie	nie
Web Maker	nie	tak	tak	tak

3.7. Sposoby zapisywania projektów

Następną rozpatrywaną cechą jest metoda zapisywania kodu stworzonego przez użytkownika. Niektóre narzędzia wykorzystują mechanizm localStorage do tymczasowego zapisywania bieżącego stanu projektu. Powszechną metodą jest wykorzystanie adresu URL. W celu zapisywania kodu tworzonego przez użytkowników wykorzystywana jest także usługa Gist (służąca do szybkiego tworzenia małych repozytoriów). Kolejną zidentyfikowaną metodą jest zapisanie projektu po stronie serwera bez bezpośredniego odnośnika URL (ten sposób w zestawieniu reprezentuje kolumna „serwer"). Ostatnią metodą jest wykorzystanie systemu plików komputera użytkownika (sposób występujący narzędziach działających w trybie offline). W tabeli 9. zaprezentowano wsparcie dla różnych mechanizmów zapisywania projektów w poszczególnych narzędziach typu FECP.

Tabela 9. Wsparcie dla metod zapisywania projektu w aplikacjach typu FECP

Narzędzie	Metoda zapisywania projektu				
	localStorage	URL	Gist	serwer	lokalnie
CodePen	nie	tak	tak	nie	nie
CSSDeck	tak	tak	nie	nie	nie
CSSDesk	tak	tak	nie	nie	nie
Dabblet	tak	tak	tak	nie	nie
JS Bin	tak	tak	tak	nie	nie
JSFiddle	nie	tak	nie	nie	nie
Kodnest	nie	tak	nie	nie	nie
Liveweave	tak	tak	nie	nie	nie
Pinocode	nie	nie	nie	nie	tak
Plunker	tak	tak	nie	tak	nie
SoloLearn	nie	tak	nie	nie	nie
Web Maker	tak	nie	nie	tak	tak

3.8. Dodawanie zasobów CSS i JavaScript

Wiele aplikacji typu FECP pozwala na dodawanie do projektu bibliotek oraz szkieletów programowych CSS i JavaScript. Wyróżniono trzy sposoby realizowania tej funkcjonalności. Pierwszą z nich jest tzw. metoda „quick-add" (użytkownik wybiera zasób z predefiniowanej listy). Drugi sposób polega na wykorzystaniu systemu dostarczania treści (np. CDNJS). Ostatnim sposobem jest podanie przez użytkownika bezpośredniego linku do zasobu. W tabeli 10. przedstawiono wsparcie dla różnych metod dodawania zasobów w rozważanych narzędziach typu FECP.

Tabela 10. Wsparcie dla różnych metod dodawania zasobów do projektu

Narzędzie	Metoda dodawania zasobów		
	lista	wyszukiwarka	link
CodePen	tak	tak	tak
CSSDeck	tak	nie	nie
CSSDesk	nie	nie	nie
Dabblet	nie	nie	nie
JS Bin	tak	nie	nie
JSFiddle	tylko JavaScript	tak	tak
Kodnest	tak	nie	tak
Liveweave	tak	nie	nie
Pinocode	tak	tak	tak
Plunker	tak	tak	tak
SoloLearn	nie	nie	nie
Web Maker	tak	tak	tak

4. Wnioski

Powyższe zestawienia pozwoliły na wyciągnięcie kilku wniosków. Zdecydowana większość aplikacji typu FECP funkcjonuje w trybie online. Dokładnie jedno narzędzie – Web Maker – dostępne jest w dwóch trybach. Narzędzia typu FECP są zróżnicowane pod względem oferowanych funkcjonalności edytorów kodu. Do aplikacji dysponujących najlepiej wyposażonymi edytorami należą: CodePen, JS Bin, JSFiddle, Pinocode oraz Web Maker.

Stosunkowo niewielka liczba narzędzi wspiera niestandardowe tryby tworzenia i prezentowania projektów (wszystkie tryby wspierane są w aplikacji CodePen, jednak są one niedostępne dla użytkowników dysponujących bezpłatną wersją konta; oprócz tego trzy narzędzia wspierają tryb współpracy lub nauczania – JS Bin, JSFiddle i Liveweave).

Pięć z dwunastu rozważanych narzędzi udostępnia bazę publicznych projektów innych użytkowników (CodePen, CSSDeck, Kodnest, Plunker i SoloLearn). Pod względem dostępności preprocesorów i narzędzi HTML/CSS/JavaScript, najlepszymi aplikacjami okazały się: CodePen, JS Bin, Kodnest, Pinocode oraz Web Maker. Najczęściej wspieranymi metodami eksportu są: plik ZIP oraz plik HTML (cztery narzędzia nie udostępniają możliwości wyeksportowania utworzonego kodu). Najczęściej stosowanymi sposobami zapisywania projektów okazały się: magazyn localStorage oraz adres URL. Zdecydowana większość narzędzi typu FECP wspiera co najmniej jedną metodę dodawania zasobów CSS/JavaScript (najczęściej wspierają jest „quick-add”).

Analiza zestawień zawartych w niniejszym artykule pozwoliła na podzielenie rozpatrywanych narzędzi typu FECP na trzy grupy (według poziomu zaawansowania). Pierwszą z nich jest grupa o **wysokim poziomie zaawansowania**. Do jej cech charakterystycznych należą:

- zaawansowane, liczne funkcjonalności,
- wsparcie dla preprocesorów i bibliotek,
- dostępność co najmniej jednego niestandardowego trybu tworzenia projektów,
- mało znaczące wady lub ich brak,
- bardzo dobre wyposażenie edytorów.

Kolejną wyróżnioną grupą są narzędzia o **niskim poziomie zaawansowania**, które charakteryzują się:

- brakiem zaawansowanych funkcjonalności,
- brakiem wsparcia dla preprocesorów,
- niedziałającymi funkcjonalnościami dostępnymi w interfejsie aplikacji,
- słabym wyposażeniem edytorów,
- brakiem niestandardowych trybów tworzenia projektów.

Do ostatniej grupy – narzędzi o **umiarkowanym poziomie zaawansowania** – zaliczono aplikacje charakteryzujące się cechami wymienionymi zarówno przy opisie pierwszej, jak i drugiej grupy. Taki podział pozwolił na przyporządkowanie rozważanych aplikacji typu FECP do poszczególnych grup w następujący sposób:

- narzędzia o wysokim poziomie zaawansowania: CodePen, JS Bin, JSFiddle,
- narzędzia o umiarkowanym poziomie zaawansowania: Kodnest, Liveweave, Pinocode, Plunker, Web Maker,
- narzędzia o niskim poziomie zaawansowania: CSSDeck, CSSDesk, Dabblet, SoloLearn.

Powyższy podział – podobnie jak zestawienia porównawcze – może okazać się dobrym źródłem dla osób podejmujących decyzję dotyczącą wyboru narzędzia typu Front-End Code Playground. Narzędzia o wysokim poziomie zaawansowania wydają się być najlepszą opcją dla doświadczonych programistów frontend (np. korzystających z preprocesorów, lub tworzących kod w zespole). Aplikacje o niskim poziomie zaawansowania mogą z kolei stanowić wybór początkujących programistów o niewielkich wymaganiach względem narzędzi.

Literatura

[1] Stack Overflow Developer Survey 2018, https://insights.stackoverflow.com/survey/2018/#technology [09.09.2018]

[2] 7 of the Best Code Playgrounds — SitePoint, https://www.sitepoint.com/7-code-playgrounds/ [09.09.2018]

[3] 10 Best Code Playgrounds for Developers | Code Geekz, https://codegeekz.com/best-code-playgrounds-for-developers/ [09.09.2018]

[4] 8 Code Playgrounds to Test your JavaScript Applications and Skills | Jscramber Blog, https://blog.jscrambler.com/8-code-playgrounds-to-test-your-javascript-applications-and-skills/ [09.09.2018]

[5] Comparison of online source code playgrounds - Wikipedia, https://en.wikipedia.org/wiki/Comparison_of_online_source_code_playgrounds#Online_web_client-side_source_code_playgrounds [09.09.2018]

[6] CodePen, https://codepen.io/ [09.09.2018]

[7] HTML5, CSS3, JS Demos, Creations and Experiments | CSSDeck, http://cssdeck.com/ [09.09.2018]

[8] CSSDesk - Online CSS Sandbox, http://www.cssdesk.com/ [09.09.2018]

[9] dabblet.com, http://dabblet.com/ [09.09.2018]

[10] JS Bin - Collaborative JavaScript Debugging, http://jsbin.com/ [09.09.2018]

[11] Create a new fiddle – JSFiddle, https://jsfiddle.net/ [09.09.2018]

[12] Kodnest - Realtime online code editor & playground for Frontend developers, http://kodhus.com/kodnest/ [09.09.2018]

[13] HTML, CSS and JavaScript demo – Liveweave, https://liveweave.com/ [09.09.2018]

[14] Pinocode - Chrome Web Store, https://chrome.google.com/webstore/detail/pinocode/kijgcmknihlnlfollpgdmchlidcplcmp [09.09.2018]

[15] Plunker, http://plnkr.co/ [09.09.2018]

[16] SoloLearn: Learn to Code for Free!, https://www.sololearn.com/ [09.09.2018]

[17] Web Maker - A blazing fast & offline web playground, https://webmakerapp.com/ [09.09.2018]

[18] Auto Updating Previews - CodePen Blog, https://blog.codepen.io/documentation/editor/auto-updating-previews/ [09.09.2018]

[19] What is boilerplate and why do we use it? Necessity of coding style guide, https://medium.freecodecamp.org/whats-boilerplate-and-why-do-we-use-it-let-s-check-out-the-coding-style-guide-ac2b6c814ee7 [09.09.2018]

[20] Collab Mode - CodePen Blog, https://blog.codepen.io/documentation/pro-features/collab-mode/ [09.09.2018]

[21] Forks - CodePen Blog, https://blog.codepen.io/documentation/features/forks/ [09.09.2018]

[22] Professor Mode - CodePen Blog, https://blog.codepen.io/documentation/pro-features/professor-mode/ [09.09.2018]

[23] What is codecasting?, https://remysharp.com/2013/11/14/what-is-codecasting/ [09.09.2018]

[24] R. Queirós, A Survey on CSS Preprocessors, 6th Symposium on Languages, Applications and Technologies (SLATE 2017), http://drops.dagstuhl.de/opus/volltexte/2017/7943/pdf/OASIcs-SLATE-2017-8.pdf.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 9 (2018) 334-339
Wysłane: 2018-09-14
Przyjęte: 2018-09-19

Porównanie wybranych narzędzi do przeprowadzania testów jednostkowych

Piotr Strzelecki*, Maria Skublewska - Paszkowska
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł przedstawia analizę porównawczą wybranych narzędzi służących do przeprowadzania testów jednostkowych. Analizie zostały poddane trzy najpopularniejsze frameworki: MSTest, NUnit oraz xUnit.net. Analiza polega na porównaniu szybkości wykonywania testów przez narzędzia w sposób szeregowy oraz równoległy. Badania przeprowadzono z wykorzystaniem autorskiej aplikacji na platformę .NET.

Słowa kluczowe: testy jednostkowe; testowanie; .NET

*Autor do korespondencji.

Adres e-mail: piotr.strzelecki93@gmail.com

Comparison of selected tools to perform unit tests

Piotr Strzelecki*, Maria Skublewska - Paszkowska
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The paper presents a comparative analysis of selected tools to perform unit tests. The analysis covers three most popular frameworks: MSTest, NUnit and xUnit.net. The analysis concerns the comparison of the speed of tests performing by the tools in serial and parallel manner. The tests were carried out by the author's application dedicated to .NET platform.

Keywords: unit tests; testing; .NET

*Corresponding author.

E-mail address: piotr.strzelecki93@gmail.com

1. Wstęp

W dzisiejszych czasach powstające projekty są bardzo złożone, wielkie i podatne na błędy. Z tego powodu testowanie wytwarzanego oprogramowania ma kluczowe znaczenie. Testy powinny być pisane profesjonalnie i skutecznie, czyli tak, aby wynik końcowy był sprawdzony pod każdym możliwym kątem, dzięki czemu użytkownicy z niego korzystający byliby zadowoleni z wydanej aplikacji. Korzystanie z narzędzi przeznaczonych do tworzenia testów jednostkowych znaczne upraszcza pisanie i zarządzanie zestawami takich testów, które sprawdzają poprawność działania wytwarzanej aplikacji.

Artykuł podejmuje temat porównania wybranych narzędzi do przeprowadzania testów jednostkowych. Analizy dokonano na bazie wykonanej aplikacji internetowej opartej o platformę .NET. Celem badania było porównanie narzędzi MSTest, NUnit oraz xUnit.net pod kątem szybkości czasów wykonywania tworzonych przez nich testów budowanych w sposób szeregowy oraz równoległy. Przeprowadzone badanie pozwoli programiście na wybranie odpowiedniego narzędzia, służącego do tworzenia testów jednostkowych.

2. Przegląd literatury

Zanim zostały przeprowadzone badania, które są tematem tego artykułu, dokonano analizy dostępnych materiałów badawczych. Nie było dotychczas analizy porównawczej narzędzi, które tutaj są porównywane. W literaturze natomiast, można znaleźć kilka przykładów dotyczących przeprowadzania testów jednostkowych. Według pracy Augustyna D. R. [1] framework NUnit stał się standardem dla rozwiązań w zakresie testów jednostkowych w technologii .NET. Badania [2, 3] pokazują, że stosowanie testów jednostkowych jest jednym z podstawowych technik pozwalających zapewnić niezawodność wytwarzanego oprogramowania, a wraz z jego rozwojem powstały inne metodyki, ułatwiające tworzenie takich testów, m.in. TDD (ang. Test-Driven Development).

3. Testy jednostkowe

Testy jednostkowe (ang. unit test), nazywane również modułowymi, to w programowaniu sposób testowania wytwarzanego oprogramowania poprzez wykonywanie testów, które weryfikują poprawność działania pojedynczych elementów programu. Najczęściej tymi elementami są klasy, funkcje, interfejsy oraz procedury [4]. Badane części kodu testowane są oddzielnie od innych komponentów, przez co są izolowane od innych zależności, które mogłyby wpłynąć na wynik testu. Izolację taką otrzymuje się przy pomocy stosowanych zaślepek, obiektów imitacji (ang. mock) [5].

Sam test polega na tym, że jego autor dostarcza dane wejściowe, test wykonuje pewne instrukcje i sprawdza, czy rezultat działań zgodny jest z oczekiwaniami. Sprawdzenie poprawności jego działania wykonuje się najczęściej przy pomocy metod pomocniczych tzn. asercji.

Testy jednostkowe przeprowadzane są najczęściej dla przygotowanych uprzednio danych testowych i mają na celu ujawnienie jak największej liczby błędów. Pomagają one wykryć nieprawidłowości już na poziomie implementacji danej aplikacji, dzięki czemu mogą być one naprawione zaraz po ich odnalezieniu. Testy modułowe powinny być z założenia szybkie w uruchomieniu oraz proste w implementacji, przez co stają się one świetnym elementem do automatyzacji, co z kolei ułatwia natychmiastowe niemal sprawdzenie, czy zmiany w kodzie nie naruszyły funkcjonalności, która działała poprawnie przed wprowadzeniem zmian. Testy jednostkowe zyskały na popularności w szczególnej technice wytwarzania oprogramowania tj. TDD (ang. Test-Driven Development), czyli wytwarzaniem sterowanym testowaniem, gdzie pisanie oprogramowania zaczyna się od tworzenia testów [6].

4. Badane technologie

Wszystkie porównywane narzędzia były typu open source. W przeprowadzonej analizie brały udział:

- MSTest – (Microsoft's Test Framework) narzędzie wbudowane do IDE Visual Studio. Stworzone za jego pomocą testy można uruchomić w środowisku Visual Studio lub używając poleceń w konsoli. Aktualna wersja stabilna to 1.3.2, która jest rozwinięciem produktu Microsoft Test Framework pod nazwą MSTest V2 [7].
- NUnit – framework, stworzony dla platformy .NET i wywodzący się z rodziny programów xUnit. Testy stworzone przy pomocy tego narzędzia można uruchomić w konsoli, w Visual Studio po uprzednim pobraniu Test Adaptera lub za pomocą programu NUnit Gui – dedykowanego, niezależnego programu z interfejsem graficznym do uruchamiania i zarządzania testami napisanymi w tym frameworku. Aktualna wersja frameworka to 3.10.1 [8].
- xUnit.net – najnowsze (powstałe w 23.04.2008 r. [9]) narzędzie do tworzenia testów jednostkowych na platformie .NET, stworzone przez autora frameworka NUnit. Testy można uruchamiać z poziomu: wiersza poleceń oraz w Visual Studio, po uprzednim pobraniu pakietu xunit.runner.visualstudio. Aktualną wersją stabilną jest wersja 2.4.0 [10].

4.1. Asercje

Podczas pisania testów jednostkowych przy pomocy narzędzi do tego przeznaczonych głównymi elementami, z których korzysta programista są asercje. Dzięki nim właśnie możliwe jest ustalenie, czy dany przypadek testowy uznany jest za pomyślny lub nieudany.

Asercje są to statyczne metody znajdujące się w klasie Assert. Klasa ta jest mostem między testowanym kodem, a frameworkami. W przypadku przekazania do klasy Assert argumentów niezgodnych z założeniem, framework rozpoznaje, że dany test się nie powiódł i informuje o tym odpowiednim komunikatem [11].

NUnit używa dwóch modeli asercji: pierwszy to model klasyczny, a drugi to model ograniczenia (ang. constraint model), w którym wykorzystywana jest składnia płynna (ang. fluent) [12]. Składnia ta zawsze zaczyna się od Assert.That, po którym następuje ograniczenie np.: `Assert.That(myString, Is.EqualTo("Hello"));`

W tabeli 1 znajduje się porównanie asercji w poszczególnych frameworkach. W przypadku NUnit podane są dwa modele. Znaczenie podanych asercji można określić poprzez ich nazwę, dlatego umożliwia to korzystanie z nich bez wcześniejszej wiedzy na temat pracy z tymi narzędziami.

Tabela 1. Asercje dostępne we frameworkach [13]

NUnit	MSTest	xUnit.net
AreEqual, Is.EqualTo	AreEqual	Equal
AreNotEqual, Is.Not.EqualTo	AreNotEqual	NotEqual
AreNotSame, Is.Not.SameAs	AreNotSame	NotSame
AreSame, Is.SameAs	AreSame	Same
Contains, Does.Contain	brak	Contains
Does.Not.Contain	brak	DoesNotContain
Throws.Nothing	brak	brak
Fail	Fail	brak
Greater, Is.GreaterThan	brak	brak
Is.InRange	brak	InRange
IsAssignableFrom, Is.AssignableFrom	brak	IsAssignableFrom
IsEmpty, Is.Empty	brak	Empty
False, Is.False	IsFalse	False
IsInstanceOf, Is.InstanceOf<T>	IsInstanceOf Type	IsType<T>
IsNaN, Is.NaN	brak	brak
IsNotAssignableFrom, Is.Not.AssignableFrom<T>	brak	brak
IsNotEmpty, Is.Not.Empty	brak	NotEmpty
IsNotInstanceOf, Is.Not.InstanceOf<T>	IsNotInstanc eOfType	IsNotType<T>
IsNotNull, Is.Not.Null	IsNotNull	NotNull
IsNull, Is.Null	IsNull	Null
IsTrue, Is.True	IsTrue	True
Less, Is.LessThan	brak	brak
Is.Not.InRange	brak	NotInRange
Throws<T>, Throws.TypeOf<T>	ThrowsExce ption<T>	Throws<T>

5. Przebieg badań

5.1. Środowisko testowe

Przeprowadzone badania zostały wykonywane na maszynie, której środowisko zostało przedstawione w tabeli 2.

Tabela 2. Parametry sprzetu komputerowego

Element	Stan
System operacyjny	Windows 10, 64 bit
Procesor	Intel Core i5-3210M, 2.50 GHz x2
Pamięć RAM	8 GB
Dysk HDD	SATA III, 500 GB

5.2. Aplikacja testowa

W celu przeprowadzenia badań testowych została utworzona aplikacja internetowa, na podstawie której zostały przeprowadzane testy jednostkowe. Aplikacja ta imituje

działanie internetowego konta bankowego. System ten został napisany w języku C# z użyciem frameworka ASP.NET MVC (ang. model-view-controller) 5.2.6, w wersji frameworka .NET 4.6.1 oraz bazy danych Microsoft SQL Server 2016 LocalDB.

Aplikację można podzielić na dwa moduły: część przeznaczoną dla użytkownika oraz część zastrzeżoną tylko dla administratora systemu. W części użytkownika znajdują się funkcyjności takie jak: rejestracja, logowanie, edytowanie danych konta, przeglądanie wykonanych płatności na swoim koncie, wykonanie nowej płatności. Administrator systemu dodatkowo ma możliwość zarządzania wszystkimi danymi znajdującymi się w bazie danych.

Lista najważniejszego oprogramowania użytego w projekcie została przedstawiona w tabeli 3.

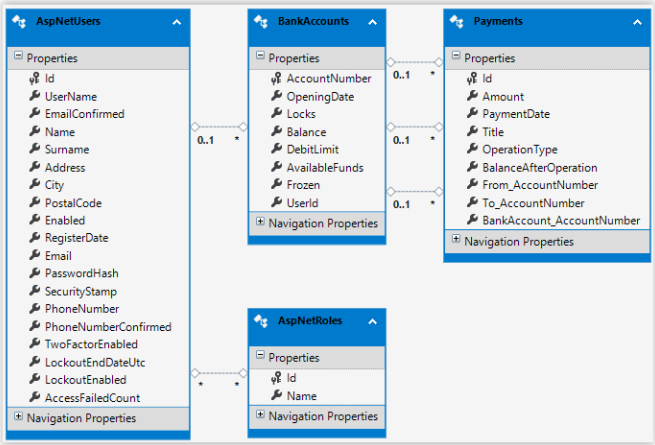
Tabela 3. Narzędzia zastosowane do budowy aplikacji

Narzędzie	Wersja
Język C#	6.0
.NET Framework	4.6.1
Serwer IIS	IIS 10.0. Express
Baza danych	MS SQL Server 2016 LocalDB
Visual Studio	2017 Community
Entity Framework	6.2.0
ASP.NET MVC	5.2.6
Ninject	3.3.4
Castle Core	4.2.1
MSTest	1.3.1
MSTest Adapter	1.3.1
NSubstitute	3.1.0
NUnit	3.10.1
NUnit3TestAdapter	3.10.0
xUnit	2.3.1
xUnit Runner Visual Studio	2.3.1

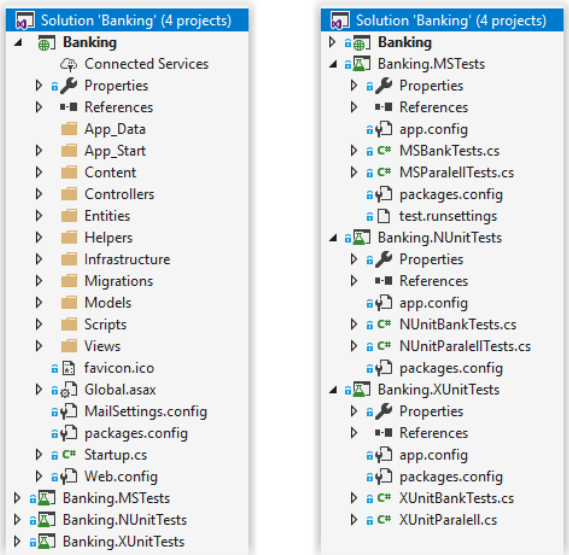
Na rysunku 1 przedstawiony został częściowy schemat bazy danych użytej w aplikacji. Zawiera on najważniejsze tabele, wykorzystane przy pracy z systemem. Główną tabelą jest tabela BankAccounts, która jest odzwierciedleniem konta bankowego. Każdy użytkownik może mieć przypisane wiele ról zdefiniowanych w systemie, które są przechowywane w tabeli AspNetRoles. Użytkownik ma możliwość posiadania wielu kont bankowych, co zostało przedstawione na relacji pomiędzy tabelą użytkowników oraz tabelą kont bankowych. Kolejną encją są płatności, które są bezpośrednio związane z kontem bankowym posiadacza. W tabeli tej znajdują się wszystkie informacje niezbędne do wykonania przelewu na konto innego użytkownika istniejącego w systemie.

5.3. Struktura aplikacji

Na rysunku 2 przedstawiona została struktura badanej aplikacji. Na całe rozwiązanie składają się 4 projekty. Pierwszym jest aplikacja MVC, której struktura widnieje po lewej stronie rysunku. Pozostałymi projektami są projekty testów jednostkowych, w których testowana jest aplikacja główna. Każdy z tych projektów dotyczy innego wybranego narzędzia do przeprowadzania testów jednostkowych, które były badane. Przykładowy projekt testowy został podzielony na część, gdzie wykonywane są testy szeregowe oraz część z testami uruchamianymi równolegle.



Rys. 1. Fragment schematu bazy danch aplikacji



Rys. 2. Struktura rozwiązania w Visual Studio

5.4. Porównanie wydajności testów uruchamianych szeregowo

Na potrzebę porównania szybkości działania frameworków został utworzony test, w którym ma miejsce czasochłonna i obliczeniowo wymagająca operacja. W metodzie tej symulowane jest utworzenie stu tysięcy kont bankowych oraz dodanie ich do kolekcji, jaką jest lista. Zadaniem frameworka jest sprawdzenie, czy którykolwiek z elementów tej listy nie jest równy null. Przykład 1 przedstawia tę metodę napisaną przy pomocy narzędzia MSTest. W pozostałych dwóch frameworkach kod tego testu wygląda identycznie.

W tabeli 4 zostały przedstawione wyniki trzykrotnego uruchomienia testu wydajnościowego. Różnica czasów wykonania pomiędzy tymi narzędziami jest bardzo widoczna. Najlepszy średni czas otrzymał MSTest, którego wynik to 243 ms na wykonanie testu. Na drugim miejscu znajduje się NUnit

z prawie dwukrotnie większym czasem. Natomiast najsłabsze wyniki osiągnęło narzędzie xUnit.net, które miało średni czas prawie 3,5 razy dłuższy od frameworka MSTest z miejsca pierwszego, tj. oraz dwa razy gorszy od frameworka na NUnit z drugiego miejsca.

Przykład 1. Kod testu wydajnościowego

```
[TestMethod]
public void All_Bank_Account_In_List_Are_Not_Null()
{
    List<BankAccount> bankList = new List<BankAccount>();

    for (int i = 0; i < 100000; i++)
    {
        BankAccount ba = new BankAccount(Guid.NewGuid(), 1000);
        bankList.Add(ba);
    }

    CollectionAssert.AllItemsAreNotNull(bankList);
}
```

Odchylenie standardowe obrazuje jak bardzo wartości jakieś zmiennej są rozrzucone wokół średniej. Obliczona wartość odchylenia standardowego w każdej z prób w danym narzędziu dała podobny wynik do różnic czasów wykonania pomiędzy narzędziami.

Tabela 4. Czasy wykonania testu wydajnościowego w poszczególnych narzędziach

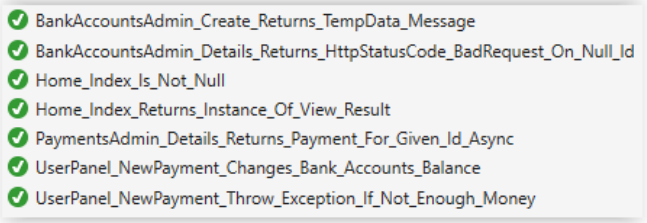
Narzędzie	MSTest			NUnit			xUnit.net		
Próba	1	2	3	1	2	3	1	2	3
Czas wykonania [ms]	242	239	248	425	411	430	814	840	832
Średni czas [ms]	243			422			829		
Odchylenie standardowe [ms]	4,58			9,85			13,32		

Na rysunku 3 przedstawiona jest lista testów jakie były uruchamiane w następnym badaniu. Nazwa każdego z testu jest równocześnie opisem mówiącym, co w danym teście się wykonuje. Nazwa testu ma schemat:

NazwaKontrolera_NazwaAkcji_OczekiwanyWynik.

Testy te kolejno mają za zadanie:

- Utworzenie nowego kontra bankowego z poziomu administratora na podstawie przekazanego modelu tworzy oczekiwany widok, do którego zostaje przekazana tymczasowa informacja pod postacią wiadomości.
- Wywołanie akcji Details konta bankowego po przekazaniu nieprawidłowego argumentu zwraca kod http 400.
- Akcja strony głównej z kontrolera Home zwraca odpowiedni widok, który nie jest wartością null.
- Kontroler Home po wywołaniu akcji Index zwraca obiekt typu ViewResult.
- Wywołanie metody Details z kontrolera PaymentsAdmin zwraca do widoku model, który jest odpowiednim obiektem płatności.
- Wywołanie przez użytkownika płatności powoduje odpowiednią zmianę stanu jego konta bankowego oraz stanu konta bankowego odbiorcy.
- Próba wykonania płatności z wartością większą niż aktualne saldo użytkownika wykonującego przelew powoduje wyrzucenie odpowiedniego wyjątku.



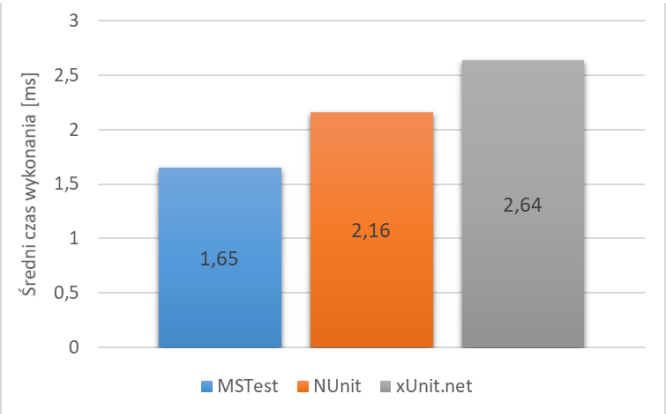
Rys. 3. Lista uruchamianych testów

Tabela 5 zawiera porównanie czasów uruchomienia szeregowo wszystkich testów , jakie znajdują się w projekcie (oprócz wcześniej opisanego testu wydajnościowego). Każdy z narzędzi uruchamiał w sumie 7 testów, których lista została przedstawiona na rysunku 3, a czas trwania takiego uruchomienia był zapisywany w tabeli.

Tabela 5. Czasy uruchomienia wszystkich testów z projektu w poszczególnych narzędziach

Narzędzie	MSTest			NUnit			xUnit.net		
Próba	1	2	3	1	2	3	1	2	3
Czas wykonania [s]	1,57	1,89	1,50	2,17	2,19	2,11	2,62	2,71	2,59
Odchylenie standardowe [s]	0,21			0,04			0,06		

W celu porównania wyników czynność ta była powtarzana trzykrotnie. Wyniki badania, przedstawione w postaci wykresu na rysunku 5, są porównywalne do poprzednio wykonanego testu wydajnościowego. Analizowane narzędzia plasują się na tych samych miejscach tj. najlepiej wypadł MSTest, a najgorzej xUnit.net. Porównanie wyników przeprowadzonych badań pokazało, że tak samo w przypadku uruchamiania jednego testu i w przypadku uruchamiania kilka testów, średnie różnice czasów między narzędziami są porównywalne.



Rys. 4. Porównanie średnich czasów uruchomienia wszystkich testów z projektu

5.5. Porównanie wydajności testów uruchamianych równolegle

Nie jest niczym niezwykłym, że projekt ma tysiące lub dziesiątki tysięcy testów jednostkowych. Deweloperzy chcą mieć pewność, że będą w stanie szybko przeprowadzić wszystkie te testy przed zatwierdzeniem ich kodu. Współczesna maszyna deweloperska może mieć procesor z 8

wirtualnymi rdzeniami oraz od 8 do 16 GB pamięci RAM (lub więcej). Te procesory są marnowane, gdy tylko jeden z nich może zostać przypisany do danego zadania. Dlatego najlepszym sposobem zapewnienia, że testy jednostkowe mogą działać z pełną prędkością komputera hosta, jest uruchomienie wielu z nich w tym samym czasie. Narzędzia porównywane w tej pracy posiadają sposób wykorzystania całej mocy obliczeniowej procesora, jakim jest równoległe wykonywanie testów jednostkowych. Każde z badanych w tej pracy narzędzi posiada możliwość zrównoleglania testów jednostkowych.

W celu zademonstrowania jak wielką rolę odgrywa równoległe uruchamianie testów został utworzony zestaw identycznych metod, które zostaną włączone sekwencyjnie oraz równolegle, po czym porównane zostaną czasy ich wykonania.

Przykład 2. Metoda do zrównoleglenia

```
[Test]
public void Test1()
{
    Thread.Sleep(1000);
    Assert.IsTrue(true);
}
```

Kod z przykładu 2 przedstawia metodę, która ma za zadanie zasymulowanie pewnego odstępu czasu, w tym przypadku jest to jedna sekunda oraz wykonanie asercji, która jest zawsze udana, tj. sprawdzenie, czy wartość true jest prawdą. Kod testu w każdym z narzędzi wygląda tak samo. Porównywane są czasy kolejno pięciu, dziesięciu, dwudziestu oraz pięćdziesięciu takich uruchomionych metod.

Tabela 6. Porównanie czasów uruchomionych testów w sposób szeregowy oraz równoległy

Narzędzie	Uruchomienie	Czas wykonania [s]		
		Szeregowe	Równoległe	Szeregowe/ równoległe
MSTest	5 testów	6,15	3,24	1,90
	10 testów	11,13	4,41	2,52
	20 testów	21,25	6,28	3,38
	50 testów	51,32	14,33	3,58
	Średnia	**22,46**	**7,07**	**2,85**
	SD	**20,24**	**5,00**	**0,78**
NUnit	5 testów	6,32	3,34	1,89
	10 testów	11,36	4,3	2,64
	20 testów	21,48	6,28	3,42
	50 testów	51,63	14,51	3,56
	Średnia	**22,70**	**7,11**	**2,88**
	SD	**20,29**	**5,08**	**0,77**
xUnit.net	5 testów	6,47	4,45	1,45
	10 testów	11,6	4,6	2,52
	20 testów	21,63	6,55	3,30
	50 testów	54,36	14,87	3,66
	Średnia	**23,52**	**7,2**	**2,73**
	SD	**21,51**	**4,93**	**0,98**

W tabeli 6 zostały przedstawione czasy uruchamiania różnej liczby testów w dwóch trybach: szeregowym oraz równoległym. Otrzymane wyniki dla każdego z narzędzi są do siebie bardzo podobne. Wraz ze wzrostem liczby jednocześnie uruchomionych metod testowych wzrasta stosunek czasu uruchomienia szeregowego do równoległego. Iloraz ten dąży do wartości 4, ponieważ testy były uruchamiane na maszynie posiadającej dwa rdzenie procesora, z czego każdy jest dwuwątkowy. Wszystkie narzędzia zostały skonfigurowane tak, aby używać wszystkich możliwych rdzeni komputera, na którym pracuje. Najszybszym z badanych w tym teście narzędzi okazał się MSTest, któremu średnia czasu uruchomienia testów wyniosła 22,46s w trybie szeregowym oraz 7,07s w trybie równoległym. Jednak różnica czasów między badanych frameworkami jest niewielka.

6. Wnioski

Testy jednostkowe powinny uruchamiać się bardzo szybko. Programista powinien być w stanie uruchomić cały zestaw testów jednostkowych w ciągu kilku sekund, a zdecydowanie nie w minutach i godzinach. Powinien mieć możliwość szybkiego uruchomienia ich po zmianie kodu. Stosowanie narzędzi do wspomagających tworzenie takich testów znacznie to upraszcza. Przedstawione w artykule trzy najpopularniejsze do tego narzędzia różnią się miedzy sobą składnią, jednak ich funkcjonalność pozostaje podobna. Jak wynika z przeprowadzonego badania wszystkie z frameworków równie dobrze radzą sobie z równoległym uruchamianiem testów, co jest ogromnym atutem dla każdego z nich. Natomiast w szeregowym, czyli klasycznym, uruchamianiu testów, wyniki nie są już takie jednoznaczne. Pod względem szybkości uruchamiania najlepszym narzędziem okazał się produkt firmy Microsoft, czyli MSTest. Drugie miejsce przypadło dla narzędzia NUnit, natomiast najsłabszy wynik otrzymało narzędzie xUnit.net. Choć szybkość wykonywania testów jednostkowych przez narzędzia do tego przeznaczone, nie jest jedynym kryterium, jakie należałoby brać pod uwagę przy ich porównywaniu, to jednak jest to jeden z ważniejszych mierników, na które zwraca uwagę programista przy wyborze narzędzia, z którym ma zamierzać pracować.

Literatura

[1] Augustyn D. R.: Rozwój narzędzi programowych wspierających automatyzację testów jednostkowych dla technologii .NET. Bazy danych. Rozwój metod i technologii. Bezpieczeństwo, wybrane technologie i zastosowania. WKŁ, Warszawa 2008

[2] Sochacki, G. , Pańczyk, B.: Test-Driven Development jako narzędzie optymalizacji procesu wytwarzania oprogramowania na platformie JEE. Journal of Computer Sciences Institute, 2017

[3] Jabłoński M., Karbowańczyk M.: Automatyczna ocena kompletności projektu programistycznego. Zeszyty Naukowe Wyższej Szkoły Informatyki, 2017

[4] Jureczko M.: Testowanie oprogramowania, Politechnika Wrocławska, 2011

[5] Bukowski M., Paterek P.: Obiekt pozorny. A może coś innego? Kierunek rozwoju testów jednostkowych, TESTER.pl, 2007, nr 10

[6] Certyfikowany tester. Plan poziomu podstawowego Wer. 1.0, przeł. Bereza-Jarociński B., Jaszcz W., Klitenik H.,

Nowakowska J., Sabak J., Seredyn A., Stapp L., Ślęzak P., Żebrowski Ł., SJSI, 2006

[7] Dokumentacja narzędzia MSTest https://msdn.microsoft.com/en-us/library/hh694602.aspx [09.03.2018]

[8] Dokumentacja narzędzia NUnit https://github.com/nunit/docs/wiki [09.03.2018]

[9] Pierwsze wydanie frameworka xUnit.net http://jamesnewkirk.typepad.com/posts/2008/04/xunitnet-10-rel.html [09.03.2018]

[10] Dokumentacja narzędzia xUnit.net https://xunit.github.io/#documentation [09.03.2018]

[11] Osherove R.: Testy jednostkowe. Świat niezawodnych aplikacji. Wydanie II, Helion, 2014

[12] NUnit - asercje https://github.com/nunit/docs/wiki/Assertions [09.03.2018]

[13] Porównanie xUnit.net z innymi frameworkami – asercje http://xunit.github.io/docs/comparisons.html#assertions [31.03.2018]

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 9 (2018) 340-344
Wysłane: 2018-09-20
Przyjęte: 2018-10-09

Określenie skuteczności zabezpieczeń aplikacji internetowych przed różnymi metodami ataków sieciowych

Mateusz Erbel*, Piotr Kopniak
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule poruszono kwestię bezpieczeństwa aplikacji internetowych. Omówiono najpopularniejsze rodzaje ataków oraz metody zabezpieczania przed nimi aplikacji internetowych. W pracy przeprowadzono badania skuteczności zabezpieczeń aplikacji internetowych. Metodykę badawczą oparto na autorskiej aplikacji, zaimplementowanej w technologii PHP. Wynikiem badań jest propozycja rozwiązań mających na celu poprawę bezpieczeństwa aplikacji.

Słowa kluczowe: Ataki sieciowe; Aplikacje internetowe; XSS; SQL Injection

*Autor do korespondencji.

Adres e-mail: mateuszerbel@gmail.com

Assessment of the web application security effectiveness against various methods of network attacks

Mateusz Erbel*, Piotr Kopniak
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article discusses the issue of the security of Internet applications. The most popular types of attacks and methods of securing web applications against them are discussed. The study conducted the effectiveness of security of web applications. The research methodology was based on the proprietary application implemented in PHP technology. The result of the research is a proposal of solutions aimed at improving application security.

Keywords: Network attacks; Internet applications; XSS; SQL Injection

*Corresponding author.

E-mail address: mateuszerbel@gmail.com

1. Wstęp

W obecnych czasach bezpieczeństwo niewątpliwie stało się jednym z najważniejszych aspektów aplikacji internetowych. Nie stworzono jeszcze aplikacji która byłaby całkowicie bezpieczna i niepodatna na żaden atak. Podczas tworzenia aplikacji programista powinien zaplanować je w taki sposób aby poziom ich ochrony był jak najwyższy, niestety często zdarza się, że aplikacje internetowe atakowane są kilka lat po ich wydaniu, przez co zamysł programistyczny musi tworzyć barierę, która pozwoli chronić dane i aplikację przez wiele lat.

Niniejszy artykuł pokazuje jak ważnym problemem jest bezpieczeństwo aplikacji internetowych, a także jak istotne jest to aby poziom zabezpieczeń był najwyższy. Temat bezpieczeństwa jest bardzo istotny dla aplikacji internetowych dlatego też został wybrany do analizy. Analiza polegała na przetestowaniu zabezpieczeń wprowadzonych do stworzonego w tym celu serwisu internetowego.

Istnieje wiele metod ataków na aplikacje internetowe. Przykładami takich ataków mogą być [1]:

- ataki odmowy dostępu DoS/DDoS,
- podsłuchanie nieszyfrowanego ruchu,
- ataki związane z wykonaniem wrogiego kodu,
- zagrożenia związanie z logowaniem i sesją użytkownika.

Aby uchronić się przed takimi zagrożeniami niezwykle ważne jest odpowiednie skonfigurowanie serwera, stworzenie funkcji i modułów, które zabezpieczą przed wykonaniem wrogiego kodu zawartego w danych wejściowych. Odpowiednie przechowywanie haseł w bazie danych oraz przeprowadzenie testów bezpieczeństwa w celu sprawdzenia czy zabezpieczenia zostały odpowiednio zaimplementowane.

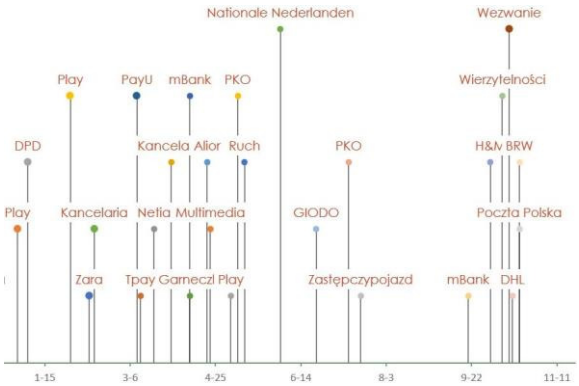
2. Wprowadzenie do bezpieczeństwa

Aplikacje internetowe mogą być atakowane z wielu powodów. Jedne z przyczyn mogą mieć podłoże finansowe, inne psychologiczne, polityczne, militarne a jeszcze inne mogą odbywać się automatycznie. Możemy wyróżnić kilka podstawowych czynników, które nakłaniają hakerów do ataków [2]:

- Korzyści materialne – często jest to główny powód ataków, pozyskanie dóbr np. takich jak środki finansowe mogą posłużyć do przeprowadzenia kolejnych ataków;
- Wandalizm – umyślnie niszczenie czyjejś pracy;
- Szpiegostwo – pozyskanie danych w celach politycznych, przemysłowych lub na własny użytek;

- Sprawdzenie swoich możliwości – sprawdzanie, testowanie własnych umiejętności;
- Przypadek – ataki wykonywane przez boty zazwyczaj wybierające losowe zasoby Internetu w celach znalezienia i wykorzystania luk.

Konsekwencji ataków tak samo jak czynników do ich przeprowadzenia jest wiele. Od pewnego czasu dość popularne są ataki phishingowe. Rysunek 1. przedstawia większość znanych kampanii e-mailowych wymierzonych w klientów firmy, których one dotknęły [3].



Rys.1. Kampanie phishingowe w ostatnich latach

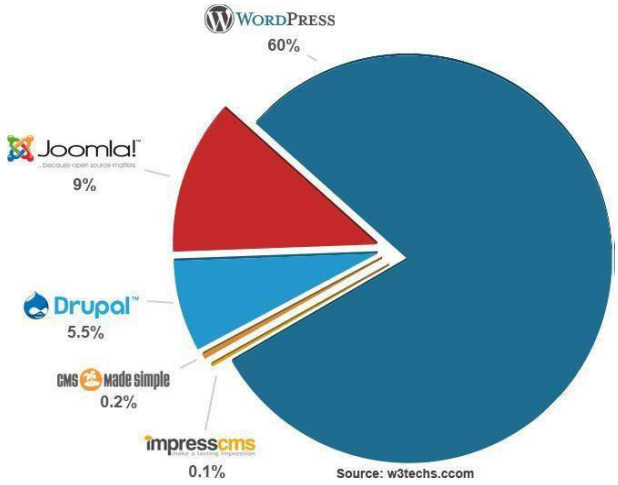
Za bezpieczeństwo aplikacji w głównej mierze odpowiada jej twórca. Niestety nie zawsze osoby tworzące oprogramowanie mają odpowiednie kwalifikacje. Częstą praktyką jest powierzanie ważnych mechanizmów programistom z małym doświadczeniem w tworzeniu aplikacji internetowych. Nie znają oni jeszcze znacznej liczby podatności przez co nie przykładają uwagi da zabezpieczenia aplikacji przed nimi.

Kolejny ważny problem wynika z typów tworzonych aplikacji internetowych. Aplikacje mogą być tworzone pod konkretnego klienta, często wynika to z potrzeby integracji z istniejącą już aplikacją. Dzięki indywidualnym rozwiązaniom dostęp do kodu źródłowego jest ograniczony przez co ograniczona jest też liczba osób znających mechanizmy wewnętrze i punkty krytyczne aplikacji. Niestety takie aplikacje przechodzą zazwyczaj podstawowe testy bezpieczeństwa przez co możliwe podatności mogą pojawić się dopiero po wdrożeniu aplikacji.

Drugim typem a zarazem najczęściej występującym są aplikacje powielane. Bazują one już na istniejących rozwiązaniach. Przykłady takich systemów przedstawia rys. 2 [9].

Inaczej niż w aplikacjach budowanych pod konkretnych klientów, ich kod jest jawny a ilość wdrożonych serwisów znaczna. Niestety w przypadku aplikacji powielanych czas od wykrycia podatności do jej załatania jest uzależniony znacząco od producenta oprogramowania. Zanim to nastąpi atakujący może dokonywać ataków na dane aplikacje powielane.

Powyższe powody są główną przyczyna podatności aplikacji na ataki. Trzeba jednak pamiętać, że wszystkie techniki ataków na aplikacje webowe posiadają indywidualne warunki, w których mogą zajść.



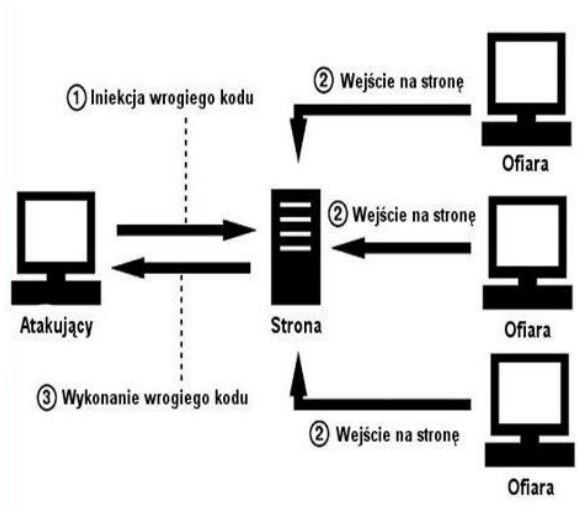
Rys.2. Udział systemów CMS na rynku w 2017 roku

3. Rodzaje ataków

W aplikacjach możemy zauważyć punkty wejścia czyli dane pobierane oraz punkty wyjścia np. takie jak dane i czynności generowane. Na pewną część danych wejściowych użytkownik ma pełny wpływ inne są dostępne tylko w konkretnych sytuacjach. Do danych wejściowych, na które użytkownik ma pełen wpływ zaliczają się wszelkie parametry przekazywane przez klienta w żądaniu HTTP. Natomiast do takich danych jak pliki konfiguracyjne, usługi sieciowe, dane innych źródeł klient ma dostęp w określonych przypadkach [4].

3.1. Ataki XSS

Jednym ze sposobów ataku na serwis WWW, są ataki typu XSS (Cross-site scripting) polegają na osadzeniu w treści atakowanej strony złośliwego kodu zazwyczaj jest to JavaScript , który wyświetlony innym użytkownikom może dążyć do wykonania przez nich niepożądanych akcji [5]. Przykładowy schemat ataku przedstawia rysunek 3.



Rys.3. Przykładowy schemat ataku XSS

Ataki XSS dzielą się na różne kategorie, główne z nich to Reflected i Stored. W pierwszym kod JavaScript zaszywany jest w linku, który następnie atakujący przesyła do ofiary. Po wejściu w link łączy się on z aplikacją w celu przekazania fragmentu HTML zawierającego kod JavaScript. Aplikacja zwraca ofierze wynik, w postaci wykonania wrogiego kodu w przeglądarce [6].

Drugim rodzajem ataku jest Stored inna spotykana nazwa to Presistent, najbardziej złośliwa odmiana, w odróżnieniu od Reflected polega na umieszczeniu kodu JavaScript po stronie serwerowej. Atakujący może podczas dodawania komentarza na stronie zamieścić w nim złośliwy kod. Dzięki odpowiedniemu filtrowaniu danych można uniknąć takich sytuacji.

3.2. Atak CSRF

Inną metodą ataków na serwisy internetowe jest atak CSRF, zmusza on zalogowaną przeglądarkę ofiary do wysłania sfałszowanego żądania http, między innymi pliku cookie sesji ofiary jak i wszelkich innych automatycznie włączonych informacji uwierzytelniających, do podatnej aplikacji internetowej. Atakujący zmusza przeglądarkę ofiary do generowania żądań, które aplikacja uważa za normalne żądania ofiary.

Istnieje cały szereg metod, które utrudniają przeprowadzenie skutecznego ataku CSRF [7]:

- Dodawanie do każdego formularz ukrytych pól, zawierających liczbę pseudolosową, która przekazywana jest wraz z żądaniem wykonania akcji. Następnie wszystkie akcje bez ukrytej wartości lub z niepokrywającą się z liczbą zachowaną po stronie serwera są ignorowane.
- Im ważniejsze informacje przechowuje strona, tym krótszy powinien być dopuszczalny czas bezczynności i okres ważności zalogowania.
- Wprowadzenie haseł jednorazowych, logowania dwuskładnikowego w znacznym stopniu uniemożliwia to osobie niepowołanej spreparowanie poprawnego żądania do serwera aplikacji.
- Wykonywanie ważnych żądań powinno wymagać ponownej autoryzacji.
- Zamiast dodawania do formularza ukrytych pól z liczbą pseudolosową, można porównać zawartość cookie służącego do uwierzytelnienie z wartością przesyłaną w żądaniu HTTP oraz z wartością zapisaną po stronie serwera.

3.3. Atak SQL Injection

Ostatnim atakiem związanym z wykonaniem wrogiego kodu, przedstawionym w pracy będzie SQL Injection. Jest to dość częsta i jednocześnie jedna z najniebezpieczniejszych podatności aplikacji webowych. Polega na wstrzyknięciu do aplikacji fragmentu zapytania SQL. Często przez brak odpowiedniej walidacji parametrów przekazywanych przez użytkownika możliwe jest wstrzyknięcie dodatkowego kodu SQL.

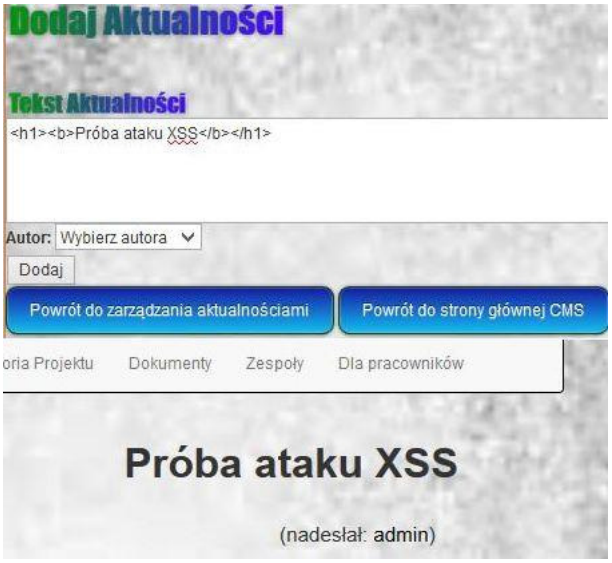
Wykorzystując podatność SQL Injection mamy bardzo wiele możliwości, co jakiś czas w Internecie pojawiają się bazy danych z loginami, hasłami do różnych stron czy portali. W większości przypadków są to dane pozyskane właśnie za pomocą tego ataku. Należy pamiętać, że ataki SQL Injection są drugimi najczęściej występującymi atakami na aplikacje webowe. W praktyce atakujący korzysta ze spreparowanego zapytania UNION SELECT aby pobrać interesujące go informację, które zawiera atakowana baza danych. Nieodłączną częścią udanego ataku na aplikację internetową jest kompromitacja jego twórcy. Niestety zazwyczaj narażona zostaje reputacja instytucji lub całego serwisu dlatego niezwykle ważnym jest odpowiednie zabezpieczenie się przed atakami SQL Injection. W celu zminimalizowania powodzenia ataku SQL Injection na aplikację należy zastosować się do kilku zasad [8]:

- Odpowiednia walidacja danych - jeżeli w formularzu oczekujemy nazwiska, sprawdzamy czy dany ciąg znaków zawiera wyłącznie litery (preg_match()), jeżeli oczekujemy liczby sprawdźmy czy na pewno tylko liczby zostały przekazane (is_numeric()).
- Używanie parametryzowanych zapytań do bazy danych.
- Odcinanie danych, które z punktu widzenia danej akcji są nieistotne .
- Ustawianie mniejszych uprawnień dla użytkowników bazy danych.
- Uniemożliwienie wyświetlanie błędów w aplikacji.

4. Badania

W trakcie przeprowadzanych badań strona jest atakowana a następnie zabezpieczana przed wykonanymi wcześniej atakami. Stworzona aplikacja umożliwia dodawanie aktualności, nowych użytkowników, dostęp do ukrytych podstron. Wymienione moduły będą główną częścią wykonywanych badań.

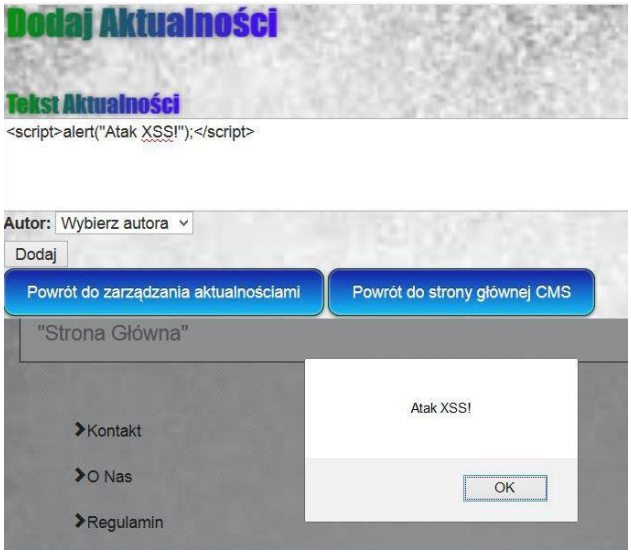
Podczas dodawania aktualności sprawdzono czy możliwa jest edycja wyświetlanych aktualności za pomocą znaczników co przedstawia rysunek 4.



Rys.4. Dodanie nowej aktualności wraz ze znacznikami HTML

W aktualnościach został opublikowany wpis „Próba ataku XSS” a dzięki znacznikom <h1> i <b> został pogrubiony i przedstawiony jako nagłówek. Jak widać

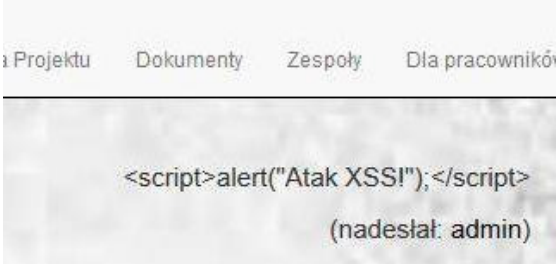
aplikacja jest podatna na ataki XSS, kolejnym krokiem będzie próba osadzenia w aplikacji kodu JavaScript co przedstawia rysunek 5.



Rys.5. Osadzenie kodu JavaScript w aplikacji

Poprzez osadzenie kodu JavaScript w aplikacji, alert będzie pokazywał się każdemu użytkownikowi który wejdzie w zakładkę aktualności. Możemy używać też znaczników <img> po to aby na atakowanej stronie osadzić dowolny obrazek. W celu zabezpieczenia aplikacji została włączona funkcję magic_quotes_sybase, która powoduje dodawanie znaków „\" w tablicach Get, Post oraz Cookie przed znacznikami specjalnymi.

Kolejną włączoną funkcją jest htmlspecialchars($string), przeszukuje ona tekst w poszukiwaniu znaczników PHP i HTML następnie zamienia znaki specjalne (<,>,',",&), na ich bezpieczne odpowiedniki. Dzięki zastosowaniu tej funkcji wyświetla się nieaktywny kod HTML, co przedstawia rysunek 6.



Rys.6. Dodanie kodu JavaScript po uruchomieniu funkcji

Podczas zmiany hasła użytkownika w systemie jest ono przesyłane w adresie URL, za pomocą ataku CSRF można próbować zmienić hasło dla zalogowanego użytkownika modyfikując adres.

Przykład 1. Zmodyfikowany adres URL

```
http://localhost/PD/admin/users/?password_new=123456&p
assword_conf=12456Change=Change#
```

Dla zwiększenia powodzenia ataku można skorzystać ze stron zmieniających linki wtedy ofiara nie będzie widzieć co jest zapisane w prawdziwym linku. Następnie użytkownik z zalogowaną sesją musi kliknąć w przesłany link. Pomóc w tym mogą różne rodzaje phishingu. Po wejściu ofiary w podany link automatycznie hasło zostanie zmienione na to wpisane w adresie URL.

Istnieje szereg metod, które w znacznym stopniu mogą utrudnić atak typu CSRF, w aplikacji wprowadzono sesje użytkownika. Ich głównym zadaniem jest sprawdzanie czy użytkownik jest zalogowany lub wylogowanie go po ustalonym czasie bezczynności co przedstawia listing 1.

Przykład 2. Wylogowanie po czasie bezczynności

```
<?php
$intTimeoutSeconds = 600;
session_start();
if(isset($_SESSION['intLastRefreshTime']))
{
if(($_SESSION['intLastRefreshTime']+$intTimeoutSeconds)<
time())
{
session_destroy();
session_start();
} }
$_SESSION['intLastRefreshTime'] = time();
?>
```

Zmienna $intTimeoutSeconds została ustawiona na 600 dzięki temu użytkownik po dziesięciu minutach nieaktywności zostanie automatycznie wylogowany.

Kolejnym zabezpieczeniem jest wymuszenie na użytkowniku ponownego wpisania hasła dla krytycznych operacji

W ostatniej części badań sprawdzono podatność aplikacji na ataki typu SQL Injection. Wykorzystując zakładkę aktualności i modyfikując jej adres URL wydostano istotne informacje z bazy danych. Podczas wyświetlania aktualności z bazy aplikacja wykonuje zapytanie.

Przykład 3. Wyświetlanie aktualności

```
$sql = 'SELECT id, text, authorid FROM aktualnosci
WHERE id = :id';
```

Jeżeli aplikacja jest podatna na atak, można modyfikując adres URL wykonywać polecenia w bazie danych.

Przykład 4. Zmodyfikowany adres URL

```
http://localhost/pd/aktualnosci.php?id=-1' union select
(dalsza część zapytania); --
```

Znakiem cudzysłów zamknięto wykonywane polecenie aby następnie można dopisać własne, na końcu użyto średnika i dwóch myślników, które w dialekcie języka SQL stosowanego w systemie zarządzania bazami danych MySQL służą do komentowania, dzięki temu dalsza część zapytania aplikacji jest traktowana jako komentarz i się nie wykona. W celu lepszego poznania struktury bazy danych wykonano atak z poleceniem:

Przykład 5. Atak w celu poznania struktury bazy

```
union select tabelname, 2 information_schema.tabels;--
```

tabela ta zawiera nazwy wszystkich tabel występujących w bazie danych. Wynik takiego zapytania przedstawia rysunek 7.

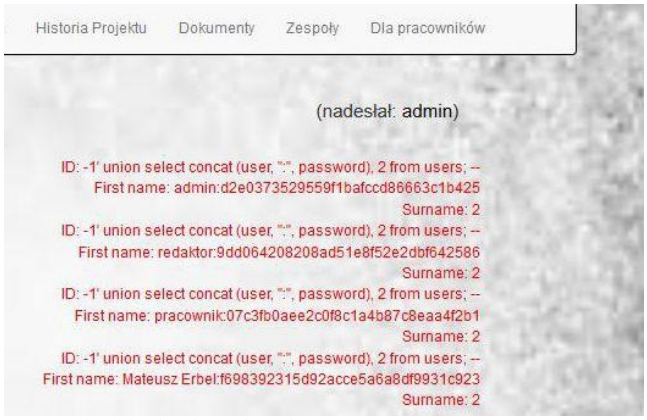
Rys.7. Wynik zapytania wyświetlającego tabele w bazie danych.

Następnie można zaatakować dowolnie wybraną tabelę, w tym przypadku wybrano tabelę users, metodą prób i błędów zmodyfikowano adres URL do następującej postaci:

Przykład 6. Zmodyfikowany adres URL

```
http://localhost/pd/aktualnosci.php?id=-1′ union select concat (user, ":", password), 2 from users; --
```

Dzięki funkcji concat() loginy i hasła będą obok siebie rozdzielone znakiem dwukropka, wynik zapytania przedstawia rysunek 8.



Rys.8. Loginy i hasła z bazy danych

Głównym zabezpieczeniem przed SQL Injection jest odpowiednie filtrowanie danych wprowadzanych przez użytkownika.

5. Wnioski

Dzięki przeprowadzonym badaniom, określono zagrożenia, którym podlegają aplikacje internetowe oraz zaproponowano możliwości zabezpieczeń przed najpopularniejszymi metodami ataków. Zanim zaimplementowano wszystkie zabezpieczenia z powodzeniem udało się wykonać szereg ataków. Dzięki wykorzystaniu luk XSS agresor mógł osadzać na stronie dowolne znaczniki HTML, zdjęcia czy nawet kod JavaScript. Najbardziej niebezpieczny okazał się atak SQL Injection, który skutkował wyciekiem loginów i zahashowanych haseł użytkowników. Aplikacja została zabezpieczona przed wcześniej wymienionymi atakami, niestety przez ich złożoność do tej pory może pozostać niezabezpieczona. Kolejne badania powinny skupić się na atakach SQL Injection, które mogą przynieść największe szkody dla twórcy.

Literatura

[1] Zagrożenia aplikacji internetowych http://tadek.pietraszek.org/publications/kasprowski03_zagrozenia.pdf, luty 2018.

[2] Ziaja A.: Practical break-in analysis, PWN, 2017.

[3] Thomas, najbardziej uciążliwy cyberprzestępca https://zaufanatrzeciastrona.pl/post/thomas-najbardziej-uciazliwy-polski-cyberprzestepca-zatrzymany-przez-policje/, marzec 2018.

[4] Mueller J.: Security for Web Developers. O'Reilly Media, 2015.

[5] Hope P, Walther B.: Web Security Testing Cookbook, O'Reilly Media, 2012.

[6] Agarwal M, Singh A.: Metasploit. Receptury pentestera. Helion 2014.

[7] Ataak CSRF, https://haker.edu.pl/2016/04/23/atak-csrf-xsrf-i hasla-wep-9/, czerwiec 2018.

[8] Prasad P.: Testy penetracyjne nowoczesnych serwisów. Helion 2017.

[9] W3Techs – extensive and reliable web technology surveys, https://w3techs.com/, marzec 2018.

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 9 (2018) 345-349
Wysłane: 2018-09-24
Przyjęte: 2018-10-04

Metoda synchronizacji i obróbki danych pozyskanych z różnych zestawów czujników inercyjnych na potrzeby analizy chodu człowieka

Aleksandra Góźdź, Maciej Kalinowski*, Piotr Kopniak

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

Streszczenie. W artykule przedstawione zostały wyniki badań dotyczących synchronizacji danych pozyskanych z dwóch systemów akwizycji ruchu na potrzeby analizy chodu człowieka. Oba systemy to zestawy czujników inercyjnych firmy Xsens: MT Awinda i Xbus Kit. W artykule omówiony zostanie format pliku do zapisu danych uzyskanych z pomiarów, oraz opracowana metoda synchronizacji pomiarów pochodzących z obydwu zestawów. W wyniku przeprowadzonych badań udało się także ujednolicić dane generowane przez czujniki umiejscowione w różnych miejscach na ciele człowieka do jednego układu odniesienia. Opracowana metoda zapewnia także progresywne przetwarzanie wartości kątów z zakresu od -180° do 180° na wartości bezwzględne w stosunku do położenia początkowego czujników.

Słowa kluczowe: akwizycja ruchu; czujniki inercyjne; synchronizacja pomiarów; przetwarzanie danych; Xsens; Java.

*Autor do korespondencji.

Adres e-mail: koliczyna@gmail.com

Method of synchronization and data processing from differents inertial sensors kits sources for the human gait analysis

Aleksandra Góźdź, Maciej Kalinowski*, Piotr Kopniak

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

Abstract. The article talks about results of data synchronization measurements sourced from two recording gait systems for human gait analyses. Two systems are Xsens sensor kits: MT Awinda, Xbus Kit. The article cover file format used to save data and synchronization method for sensor measurement from above mentioned kits. On the basis of the studies carried out, sensor measurement from different places on human body are unify to a common frame of reference. The discussed method provides also progressive data processing for angles range from -180° to 180° conversion to the absolute angle value from initial sensor settings.

Keywords: motion capture; inertial sensors; measurement synchronization; data processing; Xsens; Java.

*Corresponding author.

E-mail address: koliczyna@gmail.com

1. Wstęp

Postęp technologiczny sprawił, że dzięki coraz bardziej precyzyjnym urządzeniom możemy zbadać ruch człowieka. Czujniki inercyjne pomimo niewielkich rozmiarów są w stanie odczytać wiele istotnych informacji. Takie dane mogą być wykorzystywane w określeniu patologii chodu, które wynikają z chorób kręgosłupa [1], rehabilitacji osób chorych na Parkinsona [2, 3], bądź leczeniu osób niepełnosprawnych, które odzyskują możliwość poruszania się lub zakładają protezy [4]. Dzięki takim danym przebieg leczenia i powrotu do zdrowia może być nie tylko szybszy, ale także skuteczniejszy. Dodatkowo dzięki czujnikom ruchu możliwe jest stworzenie naturalnego interfejsu sterowania, np. robotami [5, 6]. Dokładność analizy ruchu zależy między innymi od liczby czujników oraz ich umiejscowienia na badanym obiekcie, dlatego tak ważna może okazać się synchronizacja i obróbka danych inercyjnych z wielu źródeł. Istotnym problemem badawczym jest opracowanie skutecznych metod synchronizacji i obróbki danych uzyskiwanych z różnych urządzeń pomiarowych [7].

Przeprowadzone badania opisane w tym artykule miały na celu opracowanie sposobu zestrojenia próbek danych rejestrowanych przez dwa zestawy czujników inercyjnych firmy Xsens: MT Awinda (3 czujniki bezprzewodowe) oraz Xbus Kit (2 czujniki przewodowe). Do pomiarów wykorzystane zostało specjalnie napisane oprogramowanie. Część wykonanych operacji można byłoby dodatkowo zautomatyzować już przy pobieraniu danych z Xsensa, ponieważ firma udostępnia SDK (zestaw narzędzi dla programistów) do swoich czujników. Przykładem użycia tego SDK może być praca [8], w której przedstawione zostało stworzenie wrappera w języku Java na SDK Xsensa. Taka optymalizacja miałabyn uzasadnienie przy użyciu masowym, do celów badawczych łatwiejsze jest przetwarzanie danych bezpośrednio z plików pomiarowych.

W czasie pomiarów czujniki były zamontowane odpowiednio na: biodrach oraz symetrycznie na udach, goleniach oraz stopach. Dlatego każdy plik z pomiarami ma inny lokalny układ odniesienia.

W badaniach wzięło udział 10 osób, każdy badany przeszedł 8 razy dystans 10 metrów. Pierwsze 4 przejścia badanego odbyły się z czujnikami na: biodro, lewe udo, prawe udo, lewą goleń, prawą goleń. Ostatnie 4 przejścia

z nałożonymi czujnikami na: biodro, lewe udo, prawe udo, lewą stopę, prawą stopę.

Niepoprawnie przeprowadzona synchronizacja prowadzi do błędnego zestawu danych przy analizie ruchu, co z kolei może skutkować złymi wnioskami z przeprowadzonego badania bądź eksperymentu

2. Przegląd literatury

Opracowana metoda wynika z przeglądu prac dotyczących ogólnej synchronizacji czasów zestawów pomiarowych tylko na podstawie danych otrzymywanych z poszczególnych czujników.

W opracowaniu [9] autorzy przedstawiają metodę na synchronizację pomiarów z czujników rejestrujących fale sejsmiczne za pomocą wyszukania tej samej fali zarejestrowanej przez każdy czujnik, a następnie wyrównanie zarejestrowanego czasu w każdym pomiarze poprzez zastosowanie przesunięcia.

Badania [10] sugerują aby dodatkowo wywołać ręcznie zdarzenie, które zostanie zarejestrowane przez wszystkie czujniki np. wstrząśnięcie wszystkimi czujnikami w jednej chwili, wydanie dźwięku, błysku, w zależności od danych jakie czujniki rejestrują. Następnie można w łatwy sposób odnaleźć takie wydarzenie w pomiarach z oddzielnych czujników i zsynchronizować czasy danych pomiarowych.

3. Czujniki inercyjne firmy Xsens

Czujniki inercyjne dzięki jednoczesnemu zastosowaniu żyroskopu, magnetometru i akcelerometru umożliwiają pomiar położenia przestrzennego obiektu oraz jego ruchu względem osi trójwymiarowego układu współrzędnych. Pomiary są wykonywane z częstotliwością 100Hz oraz eliminację błędów pomiarowych (Xsens) [11].

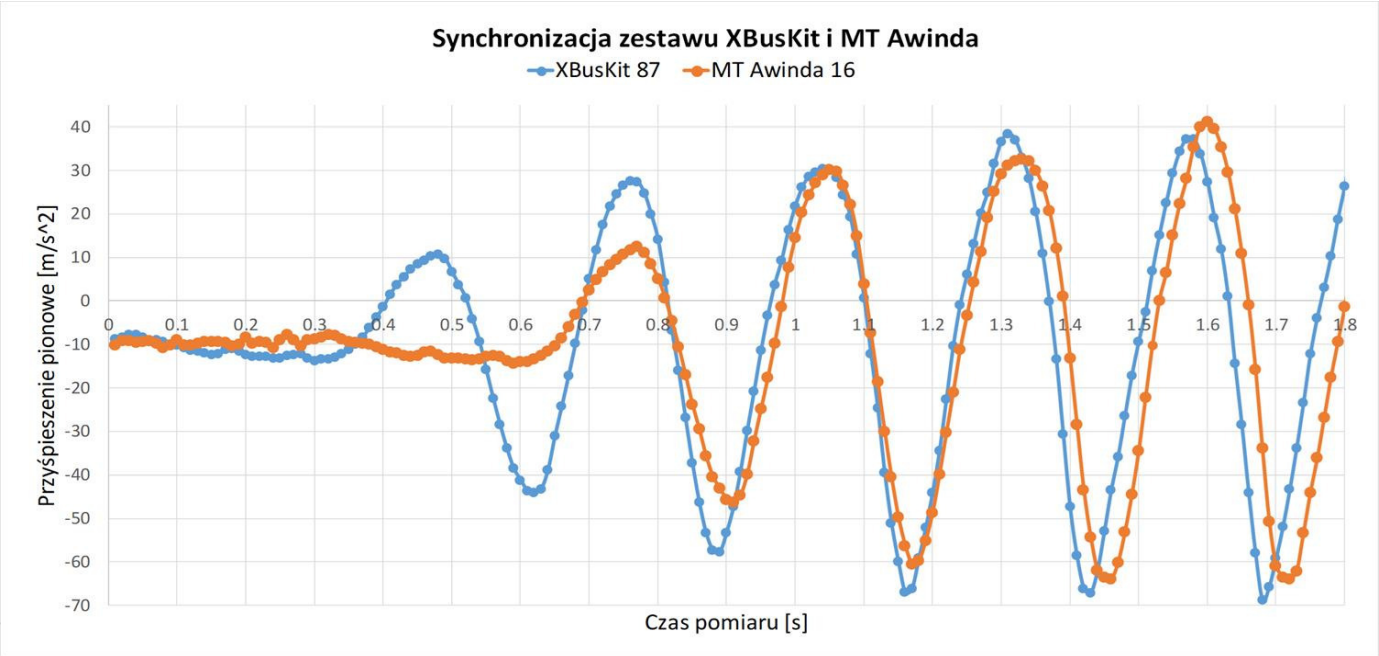
Firma Xsens dostarcza czujniki oparte o technologię MEMS (mikroukład elektromechaniczny) [12]. Firma ta opracowała jednostki inercyjne, systemy pomiaru ruchu człowieka oraz technologię Sensor Fusion, która pozwala na płynne pomiary rzeczywistości, dzięki zgrupowaniu kilku czujników w jeden zestaw i jednoczesne zapisywane na komputerze danych pomiarowych z całego zestawu.

W badaniach zostały wykorzystane czujniki z zestawu Awinda (3 czujniki MTw) oraz z XBus Kit (2 czujniki MTx). Łączą się one za pomocą protokołu radiowego umożliwiającego odbieranie i rejestrację danych, czyli śledzenie ruchu osoby badanej w czasie rzeczywistym z minimalnym okresem 10 µs między kolejnymi próbkami. Zasięg miedzy czujnikami a stacją wynosi od 20 metrów w pomieszczeniu do 50 metrów na zewnątrz. Natomiast czas pracy czujnika trwa do 6 godzin. Dzięki zastosowaniu specjalnej stacji można jednocześnie pracować z dużą liczbą czujników jednocześnie. Umożliwia ona na komunikację nawet 20 sensorom. Do kolejnych zalet czujników można zaliczyć wagę, która wynosi zaledwie 16 g, oraz ich niewielkie rozmiary.

Każdy czujnik potrafi odczytać następujące parametry w czasie rzeczywistym w trzech osiach przestrzeni trójwymiarowej: prędkość kątową (od -2000 stopni/s do 2000 stopni/s), przyspieszenie (-160 m/s2 do 160 m/s2), pole magnetyczne (-1.9 B do 1.9 B (Gaussa)). Dodatkowo zapewnia informację o rotacji czujnika, kąty przechylenia, odchylenia i pochylenia.

4. Metoda synchronizacji czujników inercyjnych

Synchronizacja czujników wykonana na potrzeby opisywanych badań opiera się na wytworzeniu charakterystycznego ruchu dla wszystkich czujników jednocześnie. Charakterystyczny znacznik tworzony jest poprzez jednoczesne wstrząśnięcie czujnikami (po jednym z każdego zestawu) zaraz po włączeniu pomiarów na każdym z zestawów. Taki sygnał jest łatwy do znalezienia w danych inercyjnych dostarczonych przez konkretne czujniki.



Rys. 1. Synchronizacja zestawu XBusKit i MT Awinda

Po zestawieniu ze sobą danych z pierwszych sekund pomiaru, otrzymano następujący wykres przedstawiony na rysunku (Rys. 1).

Przesunięcie pomiarów z zestawu XBusKit dzięki przebiegowi pomiarów o charakterze sinusoidalnym i wielu ekstremom lokalnym jest dobrze widoczne, a po dokładnym sprawdzeniu różnicy czasów próbki z wystąpienia pierwszego znaczącego ekstremum lokalnego można wyrównać czasy startów wszystkich pomiarów jednego zestawu względem drugiego zestawu.

Pozwoliło to na synchronizację na poziomie okresu czasu pomiędzy poszczególnymi próbkami. Przy wybranej częstotliwości pomiarowej urządzeń wynoszącej 100 Hz, niedokładność pomiarowa wynosi zaledwie 0.01s co zapewnia dane wystarczająco zsynchronizowane do przeprowadzenia analizy chodu człowieka.

Jednak, aby można było zaprezentować dane w postaci wykresu należało je wcześniej odpowiednio przetworzyć tzn., przede wszystkim przypisać kolejnym numerom pakietów odpowiadający im czas wystąpienia.

5. Opis danych rejestrowanych przez czujniki

Każdy czujnik zapisywał dane w pliku binarnym o formacie "MTB" rozpoznawanym przez oprogramowanie firmy Xsens, które umożliwia wyeksportowanie danych pomiarowych do pliku płaskiego "CSV". Dane w pliku zapisane były w taki sposób, że w każdym wierszy znajdowała się jedna próbka pomiaru. W kolumnach znalazły się:

1) numer pakietu - liczba całkowita od 0 do 65535,
2) przyspieszenia w kierunkach osi X/Y/Z, m/s2 (trzy kolumny danych),
3) obrót wokół osi X/Y/Z, liczby zmiennoprzecinkowe, kąty od -180° do 180° (w trzech kolejnych kolumnach).

6. Ujednolicenie numeracji i dodanie czasu pakietów

Numer pakietu jest kolejną liczbą całkowitą, której wartość wynosi 1 przy każdym starcie pomiarów i jest cyklicznie inkrementowana dla kolejnych próbek, oraz przeładowuje się po przekroczeniu maksymalnej wartości. W zależności od częstotliwości urządzenia można obliczyć czas między kolejnymi pakietami, np. przy ustawionej częstotliwości urządzenia na 100Hz czas między pomiarami wynosił 0.01s. Znaczącym utrudnieniem w procesie synchronizacji danych pomiarowych okazały się zaginione pakiety, które nie pojawiały się ostatecznie w danych pomiarowych.

Zatem, aby przygotować dane do późniejszej analizy należało je dodatkowo przetworzyć. Przetworzenie polegało na uzupełnieniu pliku o wiersze z indeksami zaginionych próbek, a następnie wyzerowaniu numeru pierwszej danej oraz przenumerowaniu ich względem numeru pierwszej próbki. Na końcu tej operacji każda kolejna próbka miała numer o jeden większy od poprzedniej.

Z tak obrobionych danych można odczytać czas zapisu każdej próbki jest to:

$$czas\_zapisu = nr\_próbki * \\ * 1/częstotliwość\_urządzenia \quad (1)$$

Aby łatwiej było przetwarzać dane do pliku płaskiego została dodana kolumna z czasem zapisu próbki. Tak przetworzone dane pomiarowe można porównać m. in. w celu synchronizacji czasów pomiarów między dwoma różnymi zestawami. Dokonanie takiej synchronizacji umożliwia poprawną analizę pomiarów i jest ostatnim krokiem obróbki czasów pakietów. Zestawienie zsynchronizowanych pomiarów przedstawia rysunek (Rys. 2). Na legendzie pomiar z każdego czujnika jest podany poprzez słowo oznaczające względem której osi był wykonany pomiar (Pochylenie/Odchylenie) oraz liczba oznaczająca numer identyfikacyjny czujnika.

Rys.2. Zsynchronizowane pomiary kąta obrotu czujników podczas pierwszych kroków człowieka

7. Obróbka kątów pochylenia w 3 osiach

Kąt pochylenia przyjmuje wartości od -180 do 180 stopni. Przekroczenie granicy powoduje natychmiastową zmianę znaku czyli przeskok z wartości dodatniej na ujemną i odwrotnie. Zakres kątów zawierający przekroczenia granicy znaku utrudniał analizę obrotów urządzenia, jeśli te były często w okolicach minimalnej i maksymalnej wartości. Widoczne na poniższym rysunku (Rys. 3.) przeskoki wartości sygnału oznaczonego szarą linią były właśnie spowodowane uzyskaniem wartości powyżej 180°. Aby zamienić te kąty na bardziej czytelne z punktu widzenia analizy chodu człowieka należy zamienić je na absolutne wartości obrotów w stosunku do kąta obrotu przy starcie pomiaru. Zrealizowano to za pomocą wykrywania przeskoku większego o 90°, gdzie w zależności od kąta α przed przeskokiem i kąta β po przeskoku, absolutny kąt γ wynosił:

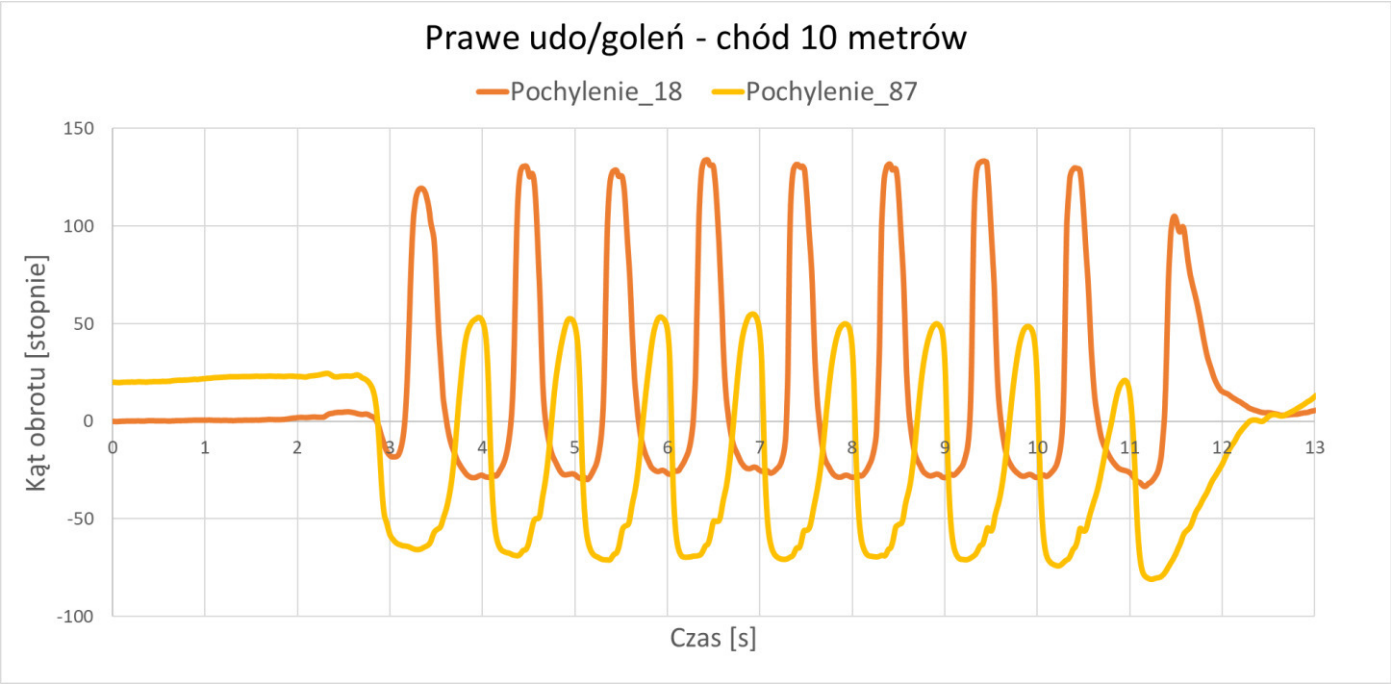
$$180^o > \alpha > 0^o, -180^o < \beta < 0^o \quad (2)$$

$$\gamma = 360^o + \beta \quad (3)$$

$$-180^o < \alpha < 0^o, 180^o > \beta > 0^o \quad (4)$$

$$\gamma = -360^o + \beta \quad (5)$$

Taka zmiana zapobiega występowaniu charakterystycznych przeskoków na wykresie. Następnie w zależności od ułożenia czujników, które było najczęściej lustrzane np. lewe udo, prawe udo, należało ujednolicić położenie układów współrzędnych poprzez modyfikację wartości kątów obrotów odbitych symetrycznie względem osi, czyli zmienić na wartości o przeciwnych znakach. Ostatnim krokiem było zrównanie wartości początkowych tak aby śledzenie zmian w cyklu chodu człowieka było łatwiejsze. Przykładem po obróbce danych kątowych może być porównanie przebiegu czasowego zmian kątów prawego uda z prawą golenią (Rys. 3).



Rys.3. Prawe udo/goleń - chód na dystansie 10m

8. Wnioski

Pomimo, że nie istnieje dużo źródeł w tematyce synchronizacji danych pochodzących z różnych czujników inercyjnych jest to temat ciekawy badawczo. Pozwala ciągle na opracowywanie nowych rozwiązań począwszy od tych technicznych dotyczących formatu danych zapisanego pomiaru, po zagadnienia związane ze spójnością danych oraz obróbką danych do usprawnienia analizy konkretnego zjawiska.

Opisana wyżej synchronizacja została użyta przy badaniach mających na celu rejestrowanie artefaktów chodu człowieka. Z uwagi na posiadanie dwóch zestawów czujników trzeba było znaleźć sposób na połączenie danych pozyskanych z tych sensorów w całość, aby wyniki były spójne. Opisana metoda sprawdziła się bardzo dobrze, ponieważ dane pochodzące z dwóch niezależnych zestawów opisują zachodzące zjawiska podczas chodu w spójny sposób z dużą dokładnością pomiarową. Różnica między rzeczywistymi czasami próbek może wynosić najwyżej okres jednej próbki, co daje maksymalny błąd synchronizacji czasu wynoszący 10 ms przy taktowaniach czujników z częstotliwością 100 Hz.

Metoda zapewnia taką samą precyzję jak w metodzie zastosowanej przez zespół Lukaca [9]. Dzięki wykorzystaniu wywołania ręcznego zdarzenia, jak zasugerowano w publikacji zespołu Bannacha [10] cechuje ją niezależność od środowiska w jakim znajdują się czujniki oraz ruchów badanych.

Literatura

[1] Haddas R., Ju K. L., Belanger T., Lieberman I. H., The use of gait analysis in the assessment of patients afflicted with spinal disorders, European Spine Journal, 2018

[2] Schlachetzki J. C. M., Barth J., Marxreiter F., Gossler J., Kohl Z., Reinfelder S. i inni, Wearable sensors objectively measure gait parameters in Parkinson's disease, Plos One, 2017

[3] Margiotta N., Avitabile G., Coviello G., A Wearable Wireless System for Gait Analysis for Early Diagnosis of Alzheimer and Parkinson Disease, IEEE, 2016

[4] Chen W., Xu Y., Wang J., Zhang J., Kinematic Analysis of Human Gait Based on Wearable Sensor System for Gait Rehabilitation, Journal of Medical and Biological Engineering, 2016

[5] Kopniak P., Kamiński M., Natural interface for robotic arm controlling based on inertial motion capture, IEEE, 2016

[6] Kopniak P., Kamiński M., Żyła K., Zdalne sterowanie ramieniem robota z wykorzystaniem inercyjnych czujników rejestracji ruchu, Logistyka, 2014

[7] Sacha E., Metody trójwymiarowej analizy ruchu człowieka, Aktualne Problemy Biomechaniki, 2008

[8] Kopniak P., Java wrapper for Xsens motion capture system, IEEE, 2014

[9] Bannach D., Amft O., Lukowicz P., Automatic Event-Based Synchronization of Multimodal Data Streams from Wearable and Ambient Sensors, Springer, 2009

[10] Lukac M., Davis P., Clayton R., Estrin D., Recovering temporal integrity with data driven time synchronization, IEEE, 2009

[11] Xsens 3D motion tracking, https://www.xsens.com, 18.06.2018

[12] Unnamed Systems Technology, https://www.unmannedsystemstechnology.com/2018/07/whitepaper-understanding-xsens-mems-based-motion-trackers/, 18.06.2018

JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 9 (2018) 350-358
Wysłane: 2018-09-25
Przyjęte: 2018-10-09

Badanie funkcjonalności aplikacji wykonanej w technologii .NET Core na platformie Raspberry Pi II

Tomasz Piotr Sajnóg*, Dariusz Czerwiński
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule przedstawiono wyniki analizy funkcjonalności aplikacji wykonanej w technologii .NET Core firmy Microsoft na platformie Raspberry Pi 2. Badania zostały zrealizowane poprzez implementację testowych aplikacji wykorzystujących biblioteki firmy Microsoft oraz opracowane przez społeczność zgromadzoną wokół technologii .NET Core. Skupiono się na zastosowaniu niniejszej technologii w zastosowaniach Internetu Rzeczy. Przy porównaniu brano pod uwagę możliwość wykorzystania interfejsów Raspberry Pi w technologii .NET Core oraz oficjalnie wspieranego języka Python i popularnej technologii Node.js. Porównano również wydajność tych technologii w badanym środowisku sprzętowym.

Słowa kluczowe: .NET Core; Raspberry Pi; Internet Rzeczy

*Autor do korespondencji.

Adres e-mail: tomeksajnog@wp.pl

Testing the functionality of the application made in .NET Core technology on the Raspberry Pi II platform

Tomasz Piotr Sajnóg*, Dariusz Czerwiński
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The paper presents the results of functionality analysis of application made in Microsoft .NET Core technology on Raspberry Pi 2 platform. The research was done by implementing applications using libraries provided by Microsoft and developed by .NET Core community. The focus was placed on usage of this technology in the Internet of Things applications. The comparison of the possibility of using Raspberry Pi interfaces in .NET Core, officially supported Python language and popular Node.js technology was made. The performance of these technologies was also compared.

Keywords: .NET Core; Raspberry Pi; Internet of Things

Corresponding author.

E-mail address: tomeksajnog@wp.pl

1. Wstęp

W dzisiejszych czasach coraz częściej jest mowa o tzw. Internecie Rzeczy. Jest to koncepcja gromadzenia, przesyłania oraz przetwarzania informacji przez różnego typu urządzenia. Najczęściej są to rozmaite czujniki zbierające informacje o środowisku, w którym się znajdują. Obecnie najczęściej używa się technologii Internetu Rzeczy do monitoringu środowiska, zwłaszcza jakości powietrza, a także w automatyce domowej [1].

Jedną z najpopularniejszych platform IoT jest komputer jednopłytkowy Raspberry Pi. Początkowo Raspberry Pi opracowano do nauki informatyki w szkołach, lecz znalazł wiele innych zastosowań. Najczęściej stosowany jest jako prosty serwer, który zużywa niewiele energii. Platforma ta bazuje na układach SoC firmy Broadcom i używa architektury ARM [2].

Jako systemy operacyjne instalowane na platformach typu Raspberry wykorzystuje się systemy oparte na jądrze Linux, m.in. Raspbian, ale istnieje specjalna edycja Windows 10 IoT Core, która jest darmowa [2]. Niestety różni się znacząco od systemu Windows znanego z komputerów i telefonów i jest jedynie platformą do uruchamiania aplikacji. Posiada wiele ograniczeń związanych z dostępnością sterowników do urządzeń peryferyjnych podłączanych do Raspberry Pi [3].

Popularność Raspberry Pi została osiągnięta dzięki obecności złącza GPIO, które udostępnia wiele interfejsów komunikacyjnych. Pozwala to na bezpośrednie podłączenie do układu SoC różnych czujników i elementów wykonawczych bez potrzeby stosowania konwerterów. Zwykły komputer zazwyczaj nie pozwala na to bez zastosowania konwerterów podłączanych najczęściej przez złącze USB. Złącze zapewnia także zasilanie peryferiów, choć nie powinny być to urządzenia energochłonne.

Oficjalnym językiem programowania na platformę Raspberry Pi jest Python i najwięcej przykładowych programów jest właśnie w tym języku. Również człon Pi w nazwie pochodzi od zmodyfikowanej nazwy tego języka i początkowo to Python miał być jedynym wspieranym językiem [4]. Jest on uznawany za prosty do nauki i polecany początkującym programistom. Jednak jego składnia znacząco

różni się od składni języka C i pochodnych, tj. C++, C#, Java i innych popularnych języków [5].

Kolejną popularną technologią wykorzystywaną do tworzenia aplikacji na platformie Raspberry Pi jest Node.js. Jest to środowisko do uruchamiania aplikacji napisanych w języku JavaScript. Najpopularniejszym zastosowaniem są serwery WWW i aplikacje internetowe. Podstawą jego działania jest silnik V8 opracowany przez firmę Google dla przeglądarki Chrome oraz biblioteka libUV. Technologia została stworzona w celu ułatwienia wprowadzania powiadomień push na stronach internetowych [6].

Od niedawna dla Raspberry Pi jest dostępna technologia .NET Core, która jest oficjalną implementacją .NET na otwartej licencji. Technologia .NET jest wieloplatformowa, więc może działać na innych systemach niż Windows, z którym od zawsze kojarzono .NET. Inną implementacją jest znacznie starsze Mono, lecz firma Xamarin została przejęta przez Microsoft i rozwój Mono nie jest tak dynamiczny jak .NET Core. Opracowanie Mono było możliwe dzięki standaryzacji specyfikacji .NET oraz obietnicy braku roszczeń patentowych ze strony firmy Microsoft. Jest to odmienna sytuacja niż w przypadku Javy, co pokazuje przykład procesów sądowych wytaczanych firmie Google przez Oracle za użycie patentów związanych z Javą w systemie Android [7,8].

Dla celów badawczych opracowano aplikacje w następujących językach: Python, JavaScript i C#. Ostatni z języków jest w pełni obiektowym językiem stworzonym na potrzeby .NET przez zespół Andersa Hejlsberga w latach 1998-2001 [9,10]. Jako platformę do analizy wybrano Raspberry Pi 2, ponieważ technologia .NET Core wymaga nowszego procesora, który jest obecny w Raspberry Pi 2 oraz 3. W drugim przypadku można wykorzystać 64-bitowe środowisko uruchomieniowe pod warunkiem uruchomienia 64-bitowego systemu operacyjnego, ale standardowy system Raspbian jest jedynie 32-bitowy.

Celem badawczym, który został w artykule przedstawiony jest analiza funkcjonalności oraz wydajności aplikacji .NET Core na platformie Raspberry Pi 2 w porównaniu do aplikacji powstałych w środowiskach Python i Node.js.

Na początku badań założono, że technologia .NET Core jest w stanie obsługiwać interfejsy oraz komponenty podłączone do platformy Raspberry Pi, a wykonanie tych procedur jest wydajniejsze niż w konkurencyjnych technologiach.

2. Dostępne biblioteki .NET dla Raspberry Pi

W trakcie badań literaturowych znaleziono jedynie dwie biblioteki, które były dedykowane Raspberry Pi: Unosquare.RaspberryIO oraz Raspberry# IO.

Biblioteka Unosquare.RaspberryIO jest oparta na natywnej bibliotece WiringPi i może być uruchomiona w środowiskach Mono oraz .NET Core. W aplikacjach zainstalowano ją przy pomocy narzędzia NuGet [11].

Drugą biblioteką jest Raspberry# IO, która wymagała pobrania kodu źródłowego z repozytorium i niezbędnych modyfikacji w postaci wygenerowania nowych plików .csproj, ponieważ biblioteka nie była przystosowana do .NET Core. Zmodyfikowano również kod samej biblioteki, ponieważ z powodu błędu w systemie Raspbian, nie można było uruchomić aplikacji wymagających rozpoznania typu procesora, który był identyfikowany jako BCM2835. Jest to układ z pierwszych edycji Raspberry Pi, które nie są wspierane przez .NET Core [12].

3. Analiza funkcjonalności i wydajności

W celu przeprowadzenia badań porównawczych opracowano aplikacje dedykowane dla platformy Raspberry Pi 2. Aplikacje wykonano w językach Python, JavaScript i C#.

3.1. Metoda badawcza

Wykonane aplikacje sprawdzały działanie podstawowych operacji, czyli wyświetlania tekstu i operacji matematycznych, a także działania komponentów sprzętowych, które zostały podłączone do Raspberry Pi przez interfejsy: SPI, I2C, GPIO i UART.

Za kryterium oceny funkcjonalności przyjęto poprawne działanie aplikacji w technologii .NET Core w stosunku do innych technologii. Wyniki, które uzyskiwane były przez aplikacje .NET powinny być zbliżone do tych, które były uzyskiwane przez aplikacje napisane w językach Python i Node.js.

Za kryterium oceny wydajności przyjęto czas wykonania danej operacji, który zmierzono funkcjami wbudowanymi w daną technologię. Każda operacja została wykonana 10 razy, ale z wyjątkiem inicjalizacji, która została zmierzona tylko raz w niektórych przypadkach. Wyznaczono czas średni, maksymalny oraz minimalny. Wartości czasu podano w milisekundach i zaokrąglono do 5 miejsc po przecinku. W przypadku analizy wydajności aplikacji internetowej wykorzystano program ApacheBench, który wchodzi w skład serwera Apache.

3.2. Stanowisko badawcze

Jako sprzęt badawczy wykorzystano Raspberry Pi 2 Model B V1.1. Nośnikiem dla systemu operacyjnego i aplikacji była karta microSDHC marki SanDisk Ultra o pojemności 16 GB i klasie szybkości 10.

Zainstalowano system operacyjny Raspbian Stretch Lite wydany 27 czerwca 2018 r. Wybrano wersję Lite, ponieważ nie posiada ona środowiska graficznego, które nie było użyte, a mogłoby mieć wpływ na wyniki. Po instalacji systemu uruchomiono usługę SSH i wszystkie potrzebne interfejsy komunikacyjne, a także zaktualizowano pakiety. Systemem zarządzano przez połączenie SSH przy pomocy programu PuTTY. W systemie znajdował się domyślnie zainstalowany interpreter języka Python 3.5.3. Zainstalowano z pakietów binarnych dostarczonych przez NodeSource najnowszą wersję Node.js w wersji 8.11.4, ponieważ wersja 10 nie chciała

działać z jedną z bibliotek dla Node.js, a w repozytorium systemu nie była dostępna żadna nowsza wersja. Według instrukcji ze strony biblioteki Unosquare.RaspberryIO [11] zainstalowano środowisko .NET Core. Zainstalowane SDK było w wersji 2.1.401, a środowisko uruchomieniowe w wersji 2.1.3.

Komponenty sprzętowe podłączano przez taśmę 40-pin znaną z napędów standardu IDE, do której podłączano krótkie przewody ze złączami męskimi, a ich końce do płytki stykowej, na której umieszczano elementy. Podłączono zasilacz z wyjściem MicroUSB o napięciu 5V i wydajności 2A. Podłączono również przewód sieci Ethernet, a komputer sterujący był podłączony przez sieć bezprzewodową Wi-Fi 802.11g.

Aplikacje dla .NET Core skompilowano jeszcze na komputerze przy pomocy polecenia *dotnet publish -c Release*, ponieważ kompilacja ich na Raspberry Pi mogłaby zająć więcej czasu oraz znacznie zużyć kartę pamięci. Z tego samego powodu nie zdecydowano się na kompilację aplikacji do aplikacji typu Self Contained App, czyli posiadających wbudowane środowisko uruchomieniowe, ponieważ ich rozmiar wynosi ok. 50 MB i znajduje się w nich bardzo dużo plików. Zdecydowano się uruchamiać aplikacje w sposób zbliżony do konkurencyjnych technologii, czyli podając polecenie *dotnet* i po nim nazwę pliku DLL z kodem pośrednim danej aplikacji.

3.3. Opracowane aplikacje do testowania

Opracowano następujące aplikacje w trzech technologiach:

1) wyświetlanie napisu *Hello World!* na ekranie konsoli
2) wykonanie prostych operacji matematycznych
3) prosta aplikacja internetowa wyświetlająca napis *Hello World!* po otwarciu w przeglądarce
4) wysyłanie napisu *TEST* za pomocą portu szeregowego
5) włączanie i wyłączanie diody elektroluminescencyjnej podłączonej do GPIO 17
6) odczyt temperatury z czujnika Dallas DS18B20
7) odczyt temperatury i wilgotności z czujnika DHT22
8) odczyt natężenia światła z czujnika BH1750
9) odczyt wartości z przetwornika analogowo-cyfrowego MCP3008 z podłączonym czujnikiem temperatury LM35 i konwersja na napięcie i temperaturę
10) wyświetlanie napisu *TEST* na wyświetlaczu OLED ze sterownikiem SSD1306
11) odczyt identyfikatora tagu Mifare przy pomocy czytnika MFRC522.

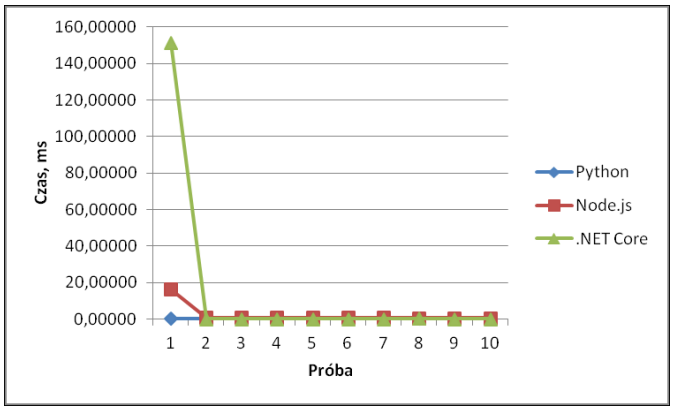
Aplikacja migająca diodą podłączoną do GPIO została opracowana w dwóch wersjach w celu porównania obu bibliotek dla .NET, ponieważ jedna z nich jest jedynie nakładką na WiringPi, a druga bezpośrednio operuje na rejestrach procesora z poziomu języka C#.

4. Wyniki badań

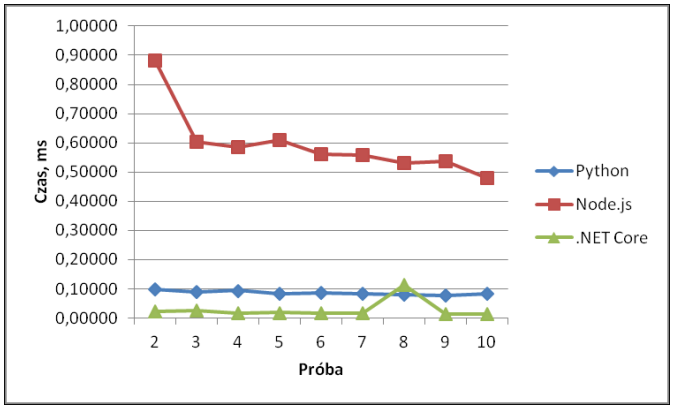
Wszystkie aplikacje udało się uruchomić bez problemów, choć w jednym przypadku aplikacja .NET nie była w stanie odczytać wskazań czujnika DHT22 z powodu implementacji biblioteki i konieczności zachowania sztywnych reguł czasowych przy obsłudze czujnika.

4.1. Wyświetlanie napisu na ekranie

Opracowane aplikacje zadziałały prawidłowo i wyświetliły napis *Hello World!* na konsoli. Na rysunkach 1 i 2 przedstawiono wykresy czasu wykonania. Zdecydowano się na dwa wykresy z powodu dłuższego czasu dla pierwszej próby, który powoduje nieczytelność wykresu dla pozostałych prób.



Rys. 1. Wykres czasu wykonania wyświetlenia napisu dla wszystkich prób



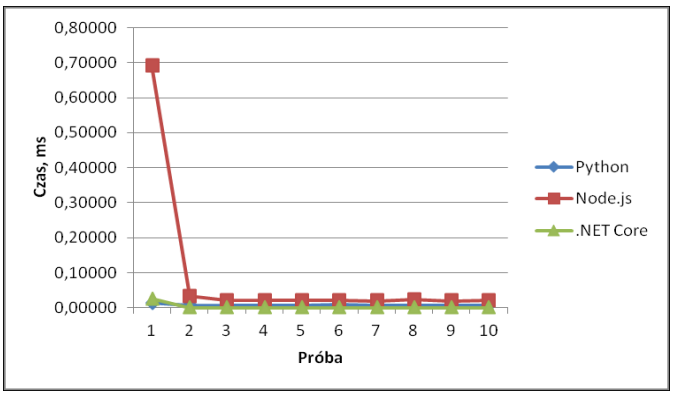
Rys. 2. Wykres czasu wykonania wyświetlenia napisu bez pierwszej próby

Pierwsze wykonanie jest znacznie dłuższe dla .NET Core, ale przy kolejnych próbach .NET Core jest najwydajniejszą technologią. W tym przypadku Node.js znacznie dłużej wykonywał operację wyświetlania napisu niż pozostałe technologie.

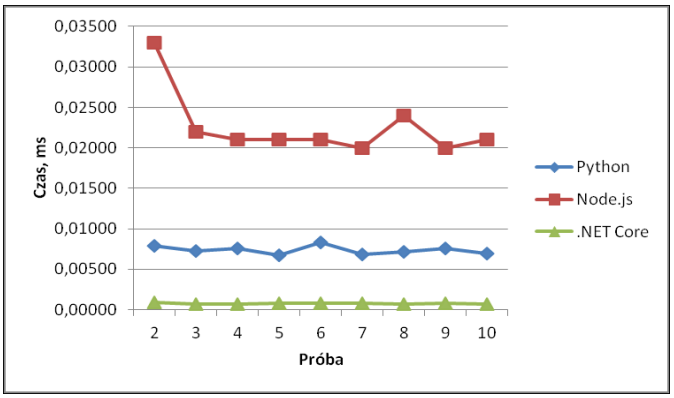
4.2. Podstawowe operacje matematyczne

Opracowane aplikacje wyświetlały jedynie czas wykonania operacji bez prezentowania wyników. Sprawdzono operacje dodawania, odejmowania, mnożenia i dzielenia dwóch liczb: x=0,5 i y=0,25. Na rysunkach 3-7 przedstawiono wykresy dla każdej z operacji.

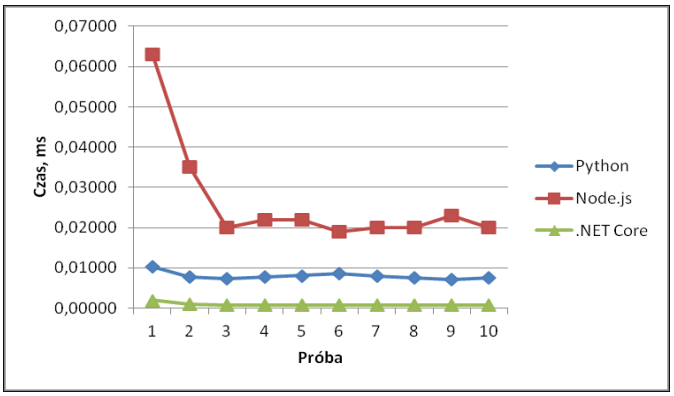
W tym przypadku pierwsze wykonanie jest znacznie dłuższe dla Node.js. Natomiast w każdej sytuacji .NET Core był najwydajniejszą technologią.



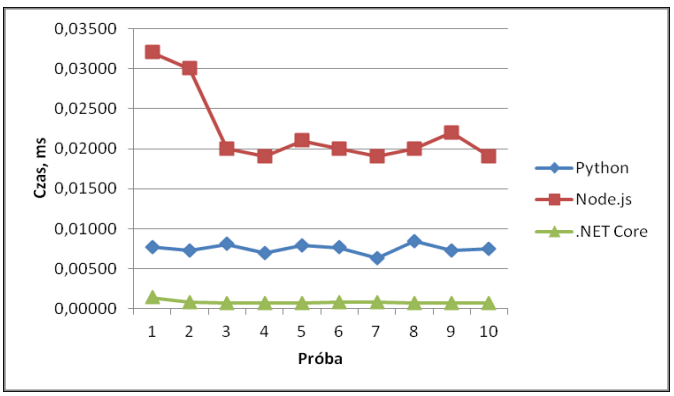
Rys. 3. Wykres czasu wykonania dla wszystkich prób operacji dodawania



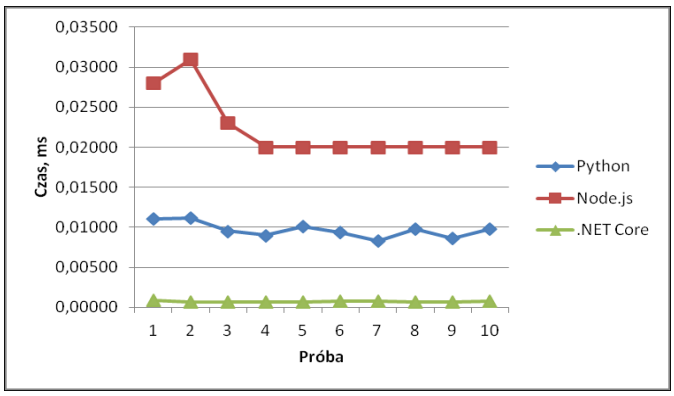
Rys. 4. Wykres czasu wykonania dla operacji dodawania bez pierwszej próby



Rys. 5. Wykres czasu wykonania dla operacji odejmowania



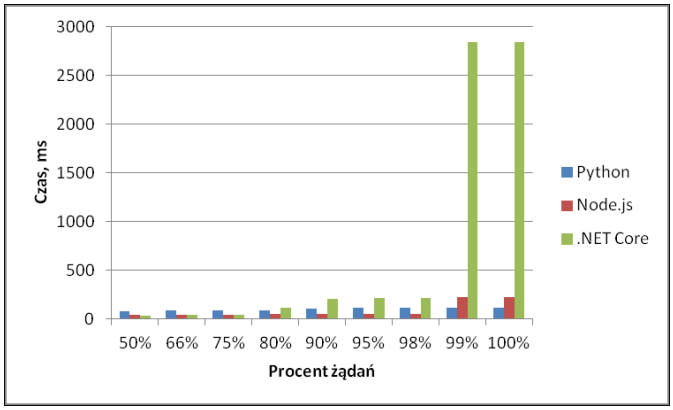
Rys. 6. Wykres czasu wykonania dla operacji mnożenia



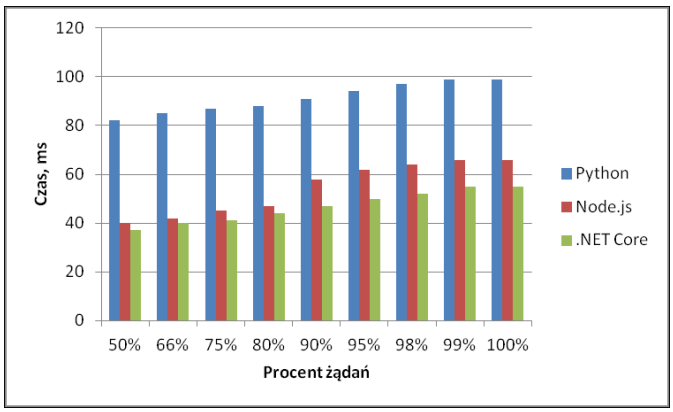
Rys. 7. Wykres czasu wykonania dla operacji dzielenia

4.3. Aplikacja internetowa

Opracowano aplikacje w ten sposób, aby każda nasłuchiwała na porcie TCP 5000. W języku Python napisano aplikację z wykorzystaniem szkieletu programistycznego Flask, w Node.js użyto Express.js, a w .NET Core wykorzystano ASP.NET Core. Ostatnia z technologii powodowała wyświetlanie się tekstu w przeglądarce inaczej niż w pozostałych, ponieważ nie był wysyłany nagłówek *Content-Type* z typem MIME danych, który w pozostałych technologiach był ustawiony na *text/html*.



Rys. 8. Czasy realizacji żądań dla pierwszej próby



Rys. 9. Czasy realizacji żądań dla drugiej próby

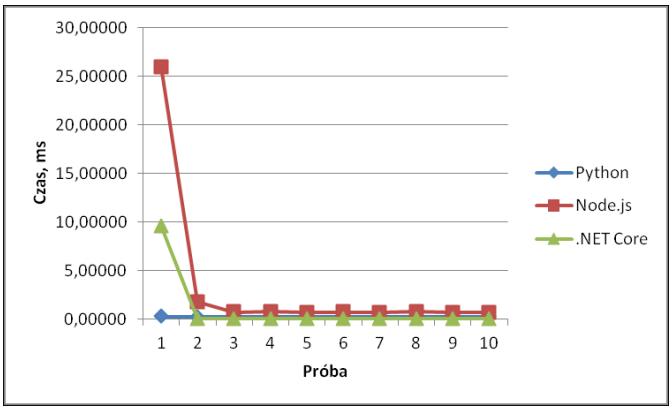
Aplikacje przetestowano programem ApacheBench, który został ustawiony do wysłania 100 żądań w tym 10 powinno

odbyć się jednocześnie. Wykonano dwie próby, aby zaobserwować zachowanie się technologii przy drugiej próbie. Na rysunkach 8 i 9 przedstawiono wykresy z danych generowanych przez program, które pokazują procentowy udział konkretnych czasów obsługi żądań.

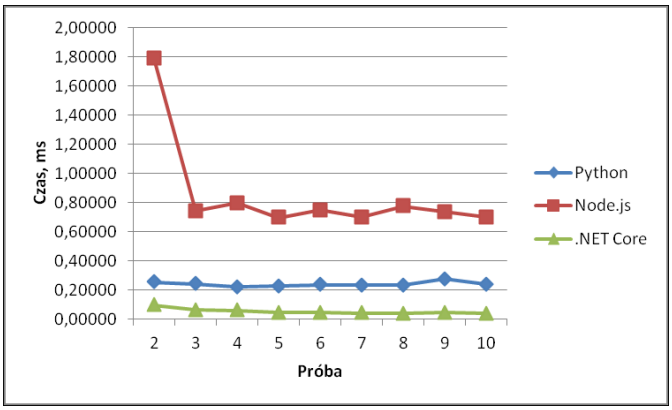
Pierwsza próba dla .NET Core była znacząco mniej wydajna, ale przy drugiej próbie okazała się to najwydajniejsza technologia, a język Python z kolei najmniej wydajny. Jednocześnie z raportu generowanego przez program wynikło, że przy drugiej próbie .NET Core obsłużył ponad 250 żądań/s.

4.4. Wysyłanie napisu przez port szeregowy

Opracowane aplikacje wysyłały napis TEST przez port szeregowy Raspberry Pi, do którego podłączono konwerter USB-UART oparty na układzie PL2303, a wyniki działania aplikacji obserwowano w drugim oknie PuTTY. W tym przypadku użyto biblioteki *NetCoreSerial*, która została pobrana z NuGet, ponieważ oficjalna biblioteka Microsoftu *System.IO.Ports* wspiera obecnie tylko system Windows. Wraz z napisem wysyłano sekwencję CRLF, aby napis wyświetlał się w nowej linii na systemie Windows. Aplikacje działały poprawnie i na ekranie ukazywał się zdefiniowany napis. Na rysunkach 10 i 11 przedstawiono wykresy dla wszystkich prób oraz wariant bez pierwszej próby.



Rys. 10. Wykres czasu wykonania wysłania napisu dla wszystkich prób

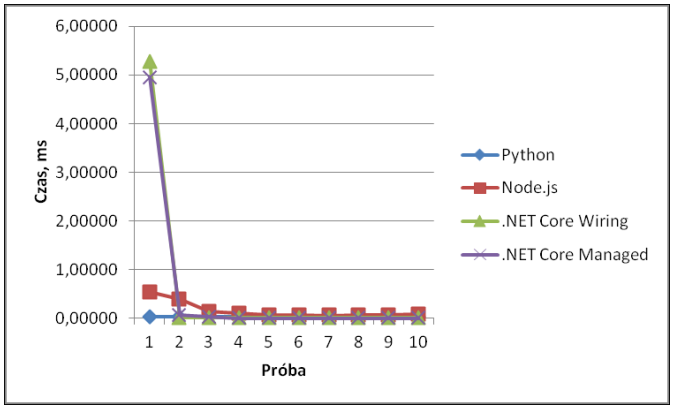


Rys. 11. Wykres czasu wykonania wysłania napisu bez pierwszej próby

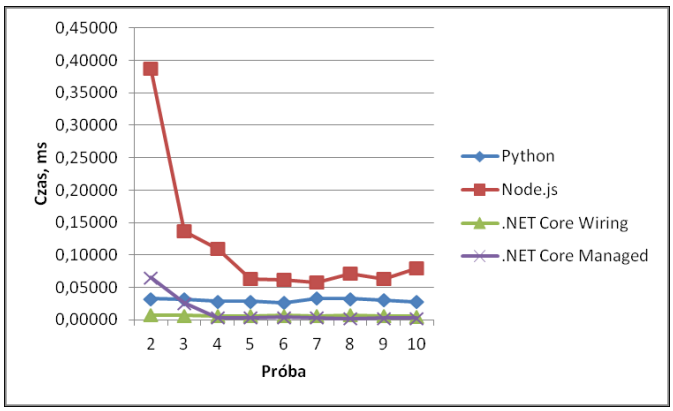
Technologia .NET Core w tym przypadku była najwydajniejsza z wyjątkiem pierwszego wykonania, które i tak było znacznie szybsze niż w przypadku Node.js, a ta technologia była najmniej wydajna.

4.5. Włączanie i wyłączanie diody

Aplikacje inicjalizowały pin GPIO 17 jako wyjście, a potem cyklicznie włączały i wyłączały diodę czerwoną podłączoną do GPIO 17 przez rezystor 200 Ω. Zmierzono również czas inicjalizacji dla każdej z technologii. Czas ten był najkrótszy dla języka Python i wynosił 0,14958 ms, dla Node.js 9,83300 ms, a dla .NET Core 693,73150 ms dla aplikacji korzystającej z biblioteki Unosquare.RaspberryIO oraz 260,63710 ms dla aplikacji używającej biblioteki Raspberry# IO. Pierwsza z bibliotek wykorzystuje WiringPi, a druga bezpośrednio operuje na rejestrach układu Broadcom BCM2836. Na rysunkach 12 i 13 przedstawiono wykresy czasu wykonania. Pierwszą aplikację oznaczono jako .NET Core Wiring, a drugą jako .NET Core Managed, ponieważ używa kodu zarządzanego.



Rys. 12. Wykres czasu wykonania przełączania diody dla wszystkich prób



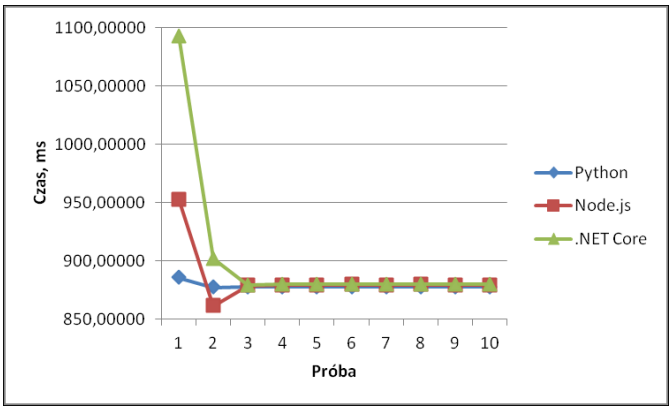
Rys. 13. Wykres czasu wykonania przełączania diody bez pierwszej próby

W przypadku aplikacji .NET Core pierwsze wykonanie było najdłuższe i to w przypadku obu metod, choć pierwsza aplikacja była nieznacznie wolniejsza. Dla kolejnych prób .NET był najwydajniejszy przy obu metodach, ale nieznaczną przewagę miała aplikacja wykorzystująca operacje na rejestrach procesora. Node.js z kolei był najmniej wydajny. Wykazuje to, że .NET Core jest bardzo wydajny przy

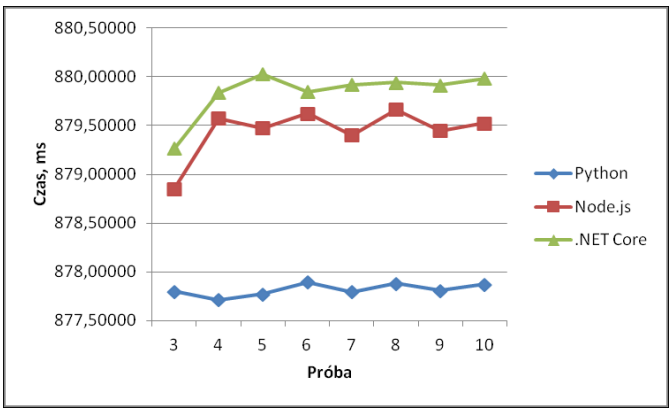
operacjach realizowanych w jak największym stopniu w języku C#, a mniej przy operacjach związanych z wykonaniem kodu z bibliotek natywnych.

4.6. Odczyt temperatury z czujnika Dallas DS18B20

Odczyt temperatury realizowany jest bez jakichkolwiek bibliotek poza standardową biblioteką dla każdej technologii, ponieważ jest to prosty odczyt i przetworzenie zawartości pliku udostępnionego przez moduł jądra systemu operacyjnego. Unikalny identyfikator czujnika został wpisany do każdej aplikacji w kodzie. Zmierzone wartości temperatury były zbliżone do siebie. Na rysunkach 14 i 15 przedstawiono wykresy czasu wykonania.



Rys. 14. Wykres czasu wykonania odczytu temperatury dla wszystkich prób

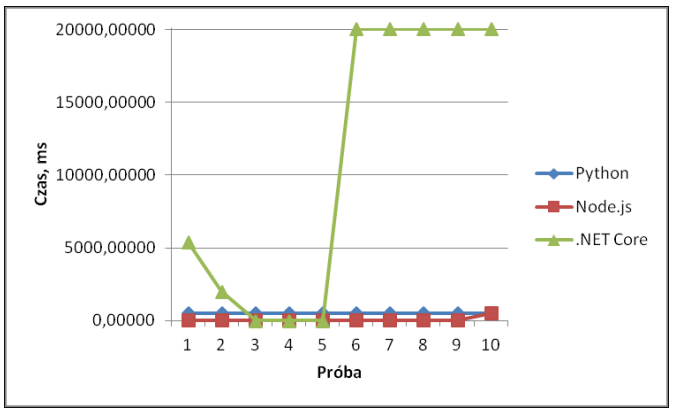


Rys. 15. Wykres czasu wykonania odczytu temperatury bez pierwszej próby

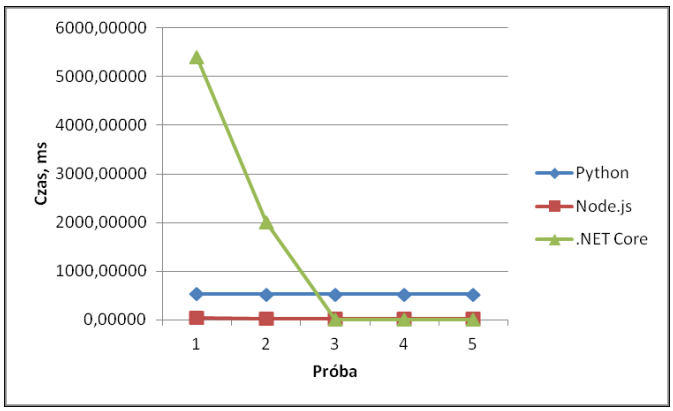
W tym przypadku .NET Core był najwolniejszy. Nieznacznie szybszy był Node.js, a najwydajniejszy okazał się Python. Czasy te jednak mogą być zaburzone przez konieczność komunikacji z czujnikiem przez jądro systemowe, a sam czujnik nie jest zbyt szybki.

4.7. Odczyt temperatury i wilgotności z czujnika DHT22

Jest to jedyny test, który się nie powiódł dla technologii .NET Core. Jedynie pierwsze wykonanie dało pozytywny rezultat i zbliżone wyniki. Kolejne cztery próby dawały nieprawdziwe wartości temperatury i wilgotności, a ostatnie 5 prób w ogóle nie powiodło się, ponieważ wystąpił błąd odczytu. Czujnik ten wymaga bardzo sztywnych zależności czasowych na linii GPIO. Wykorzystano do tego bibliotekę Raspberry# IO. Na rysunkach 16 i 17 przedstawiono wykresy dla wszystkich oraz pierwszych pięciu prób.



Rys. 16. Wykres czasu wykonania odczytu temperatury i wilgotności dla wszystkich prób



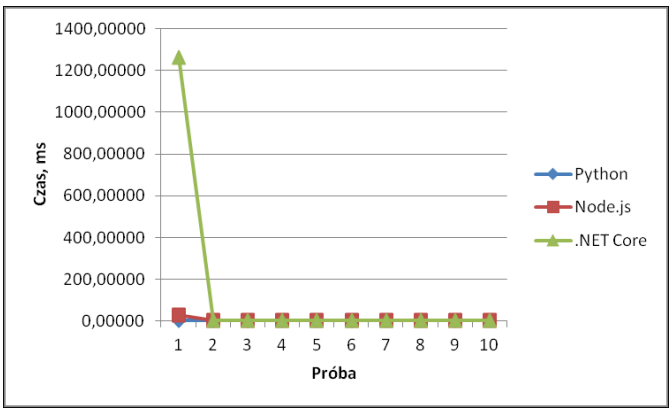
Rys. 17. Wykres czasu wykonania odczytu temperatury i wilgotności dla pięciu pierwszych prób

W tym przypadku technologia .NET była najmniej użyteczna pomimo szybkich odczytów wartości nieprawidłowych. Odczyt wartości prawidłowej był najdłuższy i trwał ponad 5 sekund. Konkurencyjne technologie posiadały biblioteki korzystające z natywnych bibliotek, a .NET Core korzystał jedynie z techniki opisanej wcześniej przy obsłudze GPIO.

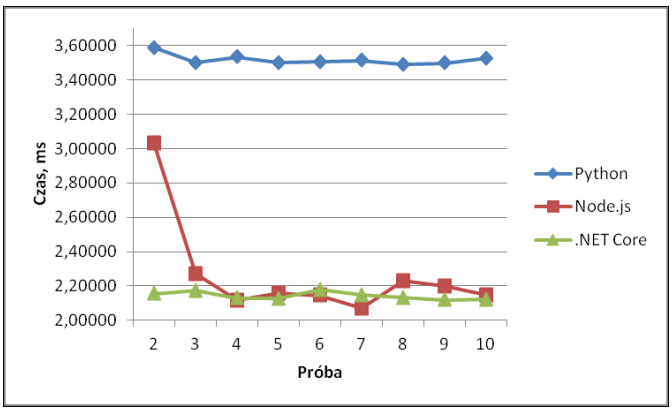
4.8. Odczyt natężenia światła z czujnika BH1750

Ten czujnik podłączono przez sprzętowy interfejs I2C. Na rysunkach 18 i 19 przedstawiono wykresy czasu wykonania. Uzyskiwane wartości pomiaru były zbliżone do siebie.

Pierwsze wykonanie jest najdłuższe dla .NET Core, ale już kolejne wykonania odczytu są zbliżone do Node.js i znacznie wydajniejsze niż w przypadku języka Python. Aplikacja .NET Core wykorzystywała bibliotekę Raspberry# IO, która I2C obsługuje bezpośrednio przez rejestry procesora, a pozostałe technologie korzystają z funkcji systemowych.



Rys. 18. Wykres czasu wykonania odczytu natężenia światła dla wszystkich prób



Rys. 19. Wykres czasu wykonania odczytu natężenia światła bez pierwszej próby

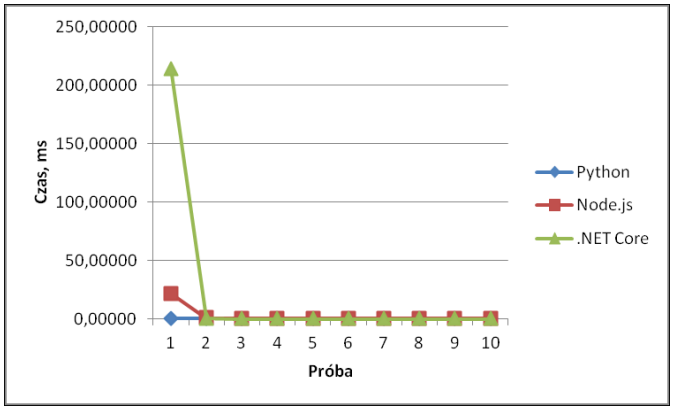
4.9. Odczyt wartości z przetwornika A/C MCP3008

Wymagane było stworzenie własnej klasy obsługującej przetwornik w bibliotece Raspberry# IO, aby obsługiwała sprzętowe SPI, ponieważ domyślna implementacja zakładała tylko SPI programowe. Biblioteka wspiera oba rodzaje implementacji SPI, więc to wykorzystano. Własną klasę opracowano na podstawie oryginalnej klasy, ale zmieniono procedury komunikacji przez interfejs SPI. Do kanału 0 przetwornika podłączono czujnik analogowy LM35, który wytwarza na wyjściu napięcie zmieniające się o 10 mV na każdy stopień Celsjusza. Uzyskiwano wartości zbliżone do siebie, choć mogły one się różnić w zależności od zakłóceń, a także dokładności samego czujnika i przetwornika. Wpływ mogła mieć również dokładność napięcia 3,3V. Na rysunkach 20 i 21 przedstawiono wykresy czasu wykonania.

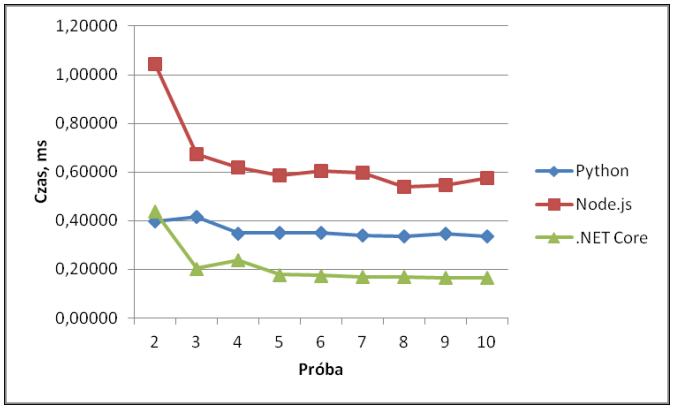
Również i tym razem pierwsze wykonanie dla .NET Core było najdłuższe, ale już kolejne były wydajniejsze, a z kolei Node.js był najmniej wydajny.

4.10. Wyświetlanie napisu na wyświetlaczu OLED

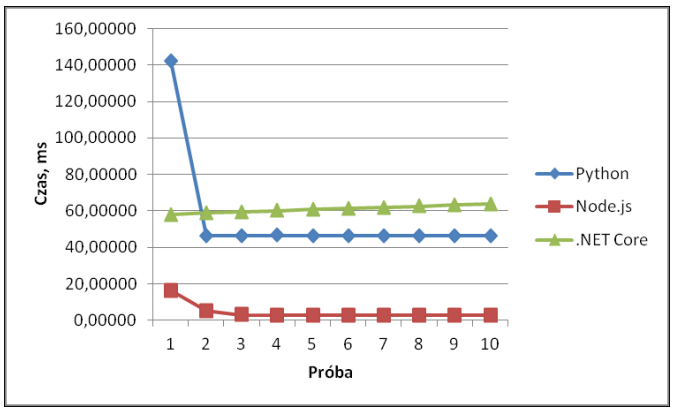
Aplikacje wyświetlały napis TEST na wyświetlaczu OLED 128x64 opartym na sterowniku SSD1306. Znów zaszła konieczność modyfikacji biblioteki Raspberry# IO, ponieważ domyślna implementacja wykorzystywała interfejs I2C. Wyświetlacz natomiast posiadał jedynie SPI, choć istnieje możliwość zmiany wykorzystywanego interfejsu przez wlutowanie rezystorów stanowiących zworki w odpowiednim miejscu. Wykorzystano SPI sprzętowe jak w poprzednim przypadku, ponieważ pozostałe aplikacje również wykorzystywały sprzętowy interfejs SPI. Na rysunkach 22 i 23 przedstawiono wykresy czasu wykonania.



Rys. 20. Wykres czasu wykonania odczytu wartości dla wszystkich prób

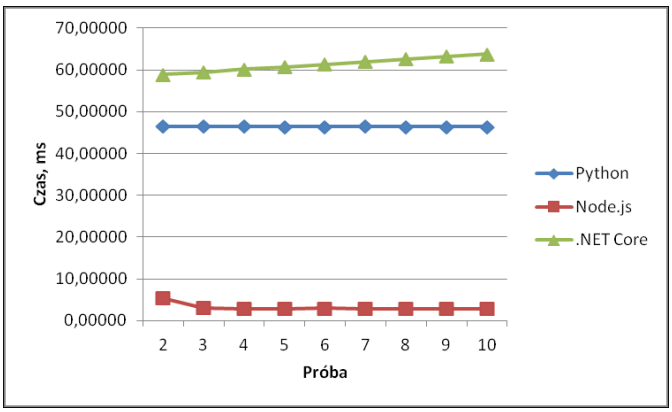


Rys. 21. Wykres czasu wykonania odczytu wartości bez pierwszej próby



Rys. 22. Wykres czasu wykonania wyświetlenia napisu dla wszystkich prób

W tym przypadku najwydajniejsza okazała się technologia Node.js, a najmniej wydajny był .NET Core, choć pierwsze wykonanie było znacznie dłuższe w przypadku języka Python. Również czas inicjalizacji dla Pythona wyniósł 13,97596 ms, dla Node.js 51,00600 ms, a dla .NET Core 473,25510 ms. Zaobserwowano również niewielkie wydłużanie się czasu w przypadku .NET Core w kolejnych próbach.

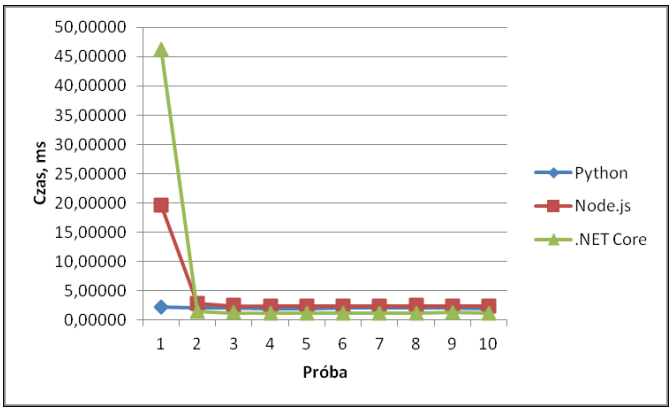


Rys. 23. Wykres czasu wykonania wyświetlenia napisu bez pierwszej próby

Biblioteka w przypadku języka Python w ogóle nie przewiduje wyświetlania tekstu, ale jest to realizowane przez inne biblioteki, które wytwarzają obraz gotowy do wysłania przez bibliotekę obsługującą wyświetlacz. Pozostałe technologie posiadają wbudowaną w biblioteki tablicę zawierającą wartości do wysłania dla każdego znaku ASCII.

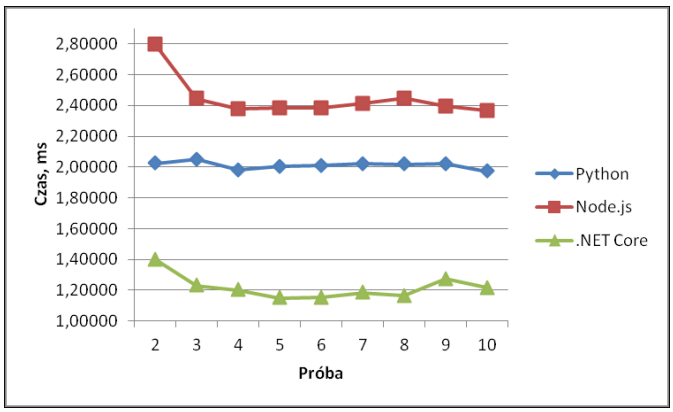
4.11. Odczyt identyfikatora tagu Mifare przy pomocy czytnika MFRC522

Po przyłożeniu breloczka zbliżeniowego załączonego wraz z modułem czytnika odczytywany był 10 razy unikalny identyfikator tagu. Aplikacje oczekiwały na zbliżenie znacznika i dopiero wtedy odczytywany był identyfikator. Wykorzystano tym razem bibliotekę Unosquare.RaspberryIO. Na rysunkach 24 i 25 przedstawiono wykresy czasu wykonania.



Rys. 24. Wykres czasu wykonania odczytu identyfikatora dla wszystkich prób

Z wyjątkiem pierwszego wykonania .NET Core był najwydajniejszy i wyraźnie szybszy od języka Python i Node.js. Najwolniejszy okazał się Node.js, a nieznacznie od niego był wolniejszy Python. Nie sprawdzono sytuacji w przypadku zbliżenia w tym samym czasie wielu znaczników Mifare.



Rys. 25. Wykres czasu wykonania odczytu identyfikatora bez pierwszej próby

5. Wnioski

Uzyskane wyniki świadczą o tym, że technologia .NET Core w obecnym stadium rozwoju jest obiecującą technologią do zastosowania w urządzeniach o niewielkiej mocy obliczeniowej, a do takich należy Raspberry Pi 2. Z wyjątkiem dość długich pierwszych wywołań była to w większości technologia najwydajniejsza. Świadczy to o dobrej optymalizacji środowiska uruchomieniowego. Optymalizacji powinien jednak zostać poddany kompilator JIT, aby szybciej kompilował kod pośredni do kodu maszynowego, choć na wydajność ma wpływ modułowa budowa .NET Core, ponieważ załadowanie jednego pliku wykonywalnego pociąga za sobą załadowanie oraz kompilację w locie plików będących zależnościami. Można jednak skompilować samodzielnie pliki do kodu maszynowego lub wykorzystać CoreRT, którego wynikiem jest niewielki plik niezależny od .NET i zawierający kod maszynowy. Druga technika jednak może prowadzić do problemów w przypadku wykorzystywania mechanizmów refleksji. Technologia ASP.NET Core posiada bardzo wydajny serwer WWW napisany w języku C#.

Dostępne biblioteki bardzo dobrze obsługują interfejsy oraz komponenty. Wyjątek stanowi jedynie czujnik DHT22, choć prawdopodobnie w przypadku próby napisania aplikacji w pozostałych technologiach bez użycia natywnego modułu efekt mógłby być podobny, co w przypadku .NET Core. Niektóre aplikacje wymagały modyfikacji kodu jednej z bibliotek, aby w ogóle mogły działać lub było możliwe porównanie wydajności obsługi interfejsu sprzętowego.

Technologia .NET Core rozwija się bardzo szybko. Zawdzięcza to udostępnieniu kodu źródłowego na licencji open source, co umożliwia jej rozwój przez firmy i osoby prywatne, a nie tylko przez Microsoft. Do niedawna .NET kojarzono jedynie z zamkniętą technologią, która mogła być uruchomiona jedynie pod systemem Windows.

Literatura

[1] https://pl.wikipedia.org/wiki/Internet_rzeczy [12.09.2018]

[2] https://pl.wikipedia.org/wiki/Raspberry_Pi [12.09.2018]

[3] Singh K.J., Kapoor D.S. Create Your Own Internet of Things: A survey of IoT platforms. IEEE Consumer Electronics Magazine, 6(2), (2017), 57-68

[4] Upton E., Halfacree G., Raspberry Pi. Przewodnik Użytkownika, Helion, 2013.

[5] https://pl.wikipedia.org/wiki/Python [12.09.2018]

[6] https://pl.wikipedia.org/wiki/Node.js [12.09.2018]

[7] https://en.wikipedia.org/wiki/.NET_Core [12.09.2018]

[8] https://pl.wikipedia.org/wiki/Mono_(projekt) [12.09.2018]

[9] Pańczyk B., Badurowicz M., Programowanie obiektowe - Język C#, Politechnika Lubelska, 2013.

[10] https://pl.wikipedia.org/wiki/C_Sharp [12.09.2018]

[11] https://github.com/unosquare/raspberryio [12.09.2018]

[12] https://github.com/raspberry-sharp/raspberry-sharp-io [12.09.2018]