JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 23 (2022) 165–171
Received: 21 April 2022
Accepted: 27 April 2022

Analysis of the performance of iOS applications developed using native and cross-platform technology

Analiza wydajności aplikacji iOS stworzonych przy użyciu technologii natywnej i crossplatformowej

Marcin Michałowski* , Maria Skublewska-Paszkowska

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

Abstract
Study presented in this paper concerns the comparative analysis of the performance of iOS applications developed using native and cross-platform technologies. For the purpose of the research, two iOS applications were implemented: the first one was created using the Swift programming language, while the second one using the Flutter technology. For both applications, a set of research scenarios was defined, which assumed the examination of the time of execution and CPU consumption during the execution of operations, such as: sorting integers, writing and reading string from a file or writing and reading records from the SQLite database. The conducted analysis showed that it is not possible to clearly state which application is more efficient in terms of execution time and CPU consumption, because they obtained divergent results for different research scenarios. The native application performed better for file and database operations, while the cross-platform one obtained lower time and CPU consumption when sorting numbers.
Keywords: iOS; Flutter; crossplatform applications; native applications

Streszczenie
Badania przedstawione w niniejszym artykule dotyczą analizy porównawczej wydajności aplikacji iOS stworzonych przy użyciu technologii natywnej i crossplatformowej. Na potrzeby badań zostały utworzone dwie aplikacje iOS: pierwsza zaimplementowana przy użyciu języka Swift, natomiast druga przy użyciu technologii Flutter. Dla obu aplikacji określono zestaw scenariuszy badawczych, które zakładały zbadanie czasu wykonania oraz zużycia jednostki obliczeniowej w czasie wykonywania poszczególnych operacji takich jak: sortowanie liczb całkowitych, zapis i odczyt ciągu znaków z pliku oraz zapis i odczyt rekordów z bazy danych SQLite. Przeprowadzona analiza wykazała, że nie da się jednoznacznie stwierdzić, która aplikacja jest bardziej wydajna pod względem czasowym i zużycia procesora, ponieważ uzyskiwały one rozbieżne wyniki dla różnych scenariuszy badawczych. Aplikacja natywna uzyskała lepsze rezultaty w przypadku operacji na plikach i operacji na bazie danych, natomiast aplikacja crossplatformowa wykazała się niższym czasem i zużyciem procesora podczas sortowania liczb.
Słowa kluczowe: iOS; Flutter; aplikacje wieloplatformowe; aplikacje natywne

*Corresponding author
Email address: marcin.michalowski@pollub.edu.pl (M. Michałowski)

1. Wstęp

Na przestrzeni ostatnich kilkunastu lat obserwuje się bardzo dynamiczny rozwój urządzeń i technologii mobilnych [1]. Gdy telefony komórkowe pojawiły się na rynku, na ich zakup mogli sobie pozwolić nieliczni, a ich możliwości były ograniczone jedynie do wykonywania połączeń. Z czasem urządzenia zaczęły oferować możliwość wysyłania wiadomości tekstowych, czy robienia zdjęć. W dzisiejszych czasach telefony przerodziły się w smartfony, które towarzyszą ludziom na co dzień, posiadają dużo większe możliwości i ułatwiają życie ich użytkownikom. Smartfony pełnią funkcję nawigacji, odtwarzaczy audio i wideo, aparatu, umożliwiają dostęp do Internetu oraz wiele innych, co jest możliwe dzięki aplikacjom, które są instalowane na tych urządzeniach.
Rynek urządzeń i aplikacji mobilnych to ogromne korzyści finansowe dla producentów. Ze względu na to, zarówno firmy produkujące urządzenia, jak i programiści kładą duży nacisk na to, aby urządzenia były coraz wydajniejsze, a aplikacje coraz lepiej zoptymalizowane pod względem zużycia zasobów urządzenia.
W rzeczywistości rynek urządzeń mobilnych jest podzielony pomiędzy dwa systemy – Android i iOS, gdzie Android stanowi 69% rynku, iOS 30%, a pozostały 1% dzieli się pomiędzy mniej istotne, najczęściej stare i nierozwijane już systemy [2]. Jeżeli firma tworząca aplikacje chce trafić do możliwie największej liczby użytkowników, musi rozwijać równocześnie dwie aplikacje – jedną dla systemu Android, a drugą na urządzenia z systemem iOS. Takie podejście wiąże się z posiadaniem dwóch zespołów do stworzenia tych aplikacji, a następnie do utrzymania ich i dalszego rozwoju, co oznacza wysokie koszty.
Aktualnie główną technologią do tworzenia aplikacji na platformę iOS jest język programowania Swift, jednak mając na uwadze wysokie koszty produkcji i utrzymania dwóch osobnych aplikacji, pojawił się pomysł programowania crossplatformowego, które umożliwia stworzenie jednej aplikacji, którą będzie można uru-

chomić zarówno na systemie Android, jak i iOS. Na rynku jest kilka technologii, które umożliwiają programowanie międzyplatformowe jak np. React Native, czy Xamarin, jednak w ostatnich latach najbardziej popularnym rozwiązaniem stał się Flutter [3].

Flutter jest technologią crossplatformową stworzoną przez Google i przedstawioną podczas Dart Developer Summit w 2015 roku. Aktualnie jest on prężnie rozwijany i daje możliwość stworzenia aplikacji nie tylko na platformy mobilne, ale także na systemy Windows, Linux, Mac, FuchsiaOS oraz jako aplikacje internetowe, które można uruchomić w przeglądarce [1]. Ponadto, w systemie FuchsiaOS, który ma być następcą systemu Android, cały interfejs i aplikacje tworzone są we Flutterze, co pokazuje, że Google będzie rozwijać tę technologię [4].

Pomimo wszystkich wymienionych korzyści płynących ze stosowania Fluttera, zastanawiające jest, czy ta technologia jest warta używania ze względu na wydajność, porównując ją z technologiami natywnymi. Poniższa praca podejmuje problem porównania dwóch identycznych pod względem funkcjonalności aplikacji stworzonych przy użyciu technologii natywnej iOS jaką jest Swift oraz technologii crossplatformowej Flutter pod względem szybkości wykonywania zadań, a także zużycia procesora.

2. Przegląd literatury

Przed przystąpieniem do analizy został wykonany przegląd literatury naukowej dotyczącej aplikacji mobilnych stworzonych przy użyciu narzędzia Flutter i technologii natywnych pod względem ich wydajności. Ze względu na fakt, że technologia Flutter jest dosyć nowa, przegląd wykazał, że jej temat nie występuje dosyć często w publikacjach naukowych, aczkolwiek bazy z literaturą naukową zawierają wiele artykułów porównujących aplikacje stworzone przy użyciu technologii natywnych z aplikacjami stworzonymi przy użyciu technologii crossplatformowych innych niż Flutter, co jest znaczące w kontekście powyższego tematu pracy.

M. Olsson w swojej pracy [5] porównuje aplikacje stworzone przy użyciu technologii natywnych – odpowiednio Kotlin dla Androida i Swift dla iOS, z aplikacją stworzoną przy użyciu narzędzia Flutter. Aplikacje zostały porównane pod względem zużycia procesora, liczby linijek kodu potrzebnych do ich stworzenia oraz wyglądu i ogólnych odczuć z ich używania przez użytkowników. Badania wykazały, że Flutter nie odbiega wydajnością pod względem zużycia procesora od technologii natywnych, a do stworzenia aplikacji w tej technologii potrzeba było prawie 3-krotnie mniej linijek kodu niż do stworzenia aplikacji w języku Swift oraz 2-krotnie mniej niż przy aplikacji napisanej przy użyciu języka Kotlin. Większość użytkowników testujących aplikacje nie zauważyło różnicy pomiędzy nimi, a jedynym mankamentem technologii Flutter okazały się animacje przewijania list, które nie były tak płynne, jak w przypadku natywnych aplikacji.

D. Gałan, K. Fisz i P. Kopniak w swoim artykule [6] porównują dwie aplikacje na platformę Android stworzone przy użyciu technologii natywnej Kotlin oraz crossplatformowej Flutter pod względem szybkości wykonywania konkretnych operacji np. sortowania liczb czy operacji na bazie danych. Autorzy poddali również analizie liczbę linijek kodu źródłowego obu aplikacji, dostępności bibliotek oraz wsparcia społeczności w obu technologiach. Badania wykazały, że aplikacja zaimplementowana za pomocą natywnej technologii jest w większej liczbie przypadków wydajniejsza, a natywna technologia jaką jest Android SDK (Kotlin) ma większą liczbę dostępnych bibliotek i większe wsparcie społeczności niż technologia Flutter.

Kolejną publikacją podejmującą tematykę porównania aplikacji natywnych i crossplatformowych pod względem ich wydajności jest artykuł P. Kotarskiego, K. Śledzia i J. Smołki [7], w którym porównane zostały aplikacje na platformę Android stworzone przy użyciu narzędzi Android SDK (Java), Android NDK, Apache Cordova i Xamarin. Badania polegały na zmierzeniu czasów sortowania liczb w tablicach, zapisu pliku o rozmiarze 10MB do pamięci urządzenia oraz odczytu pliku tekstowego o rozmiarze 10MB. Przeprowadzone testy wykazały, że Apache Cordova jest wyraźnie gorsza pod względem wydajności niż pozostałe technologie. Ponadto autorzy stwierdzili, że nie można jednoznacznie stwierdzić, która technologia jest najwydajniejsza, ponieważ wszystkie technologie oprócz Apache Cordova uzyskały bardzo zbliżone wyniki.

P. Grzmil, M. Skublewska-Paszkowska, E. Łukasik i J. Smołka w swoim artykule [8] dokonali analizy wydajności mobilnej aplikacji crossplatformowej utworzonej za pomocą technologii Xamarin oraz dwóch natywnych aplikacji dla platformy Android i iOS. Analiza dotyczyła czasu obliczania liczby π z dokładnością do 10000 miejsc po przecinku, zapisu i odczytu z pliku tekstowego obliczonej liczby π, pobierania grafiki o rozmiarze 7MB oraz ustalenia pozycji GPS. Wyniki pokazały, że prawie we wszystkich przypadkach technologie natywne są wydajniejsze niż technologia Xamarin. Autorzy zauważyli ponadto, że Xamarin.Forms pozwala znacznie skrócić czas implementacji aplikacji, co wpływa na zmniejszenie kosztów stworzenia aplikacji mobilnej.

W kolejnej publikacji [9], autorzy D. Dobrzański i W. Zabierowski, porównali ze sobą pod względem wydajności dwie aplikacje stworzone za pomocą technologii natywnych oraz aplikację crossplatformową zaimplementowaną przy użyciu technologii Xamarin. W swoim artykule przeprowadzili badania, dotyczące operacji na bazie danych SQLite, nakładania efektu na grafikę określoną liczbę razy i odświeżania danych w interfejsie użytkownika. Na podstawie wyników stwierdzono, że Xamarin jest wolniejszy niż technologie natywne, jednak nie są to duże różnice czasowe.

O. Axelsson i F. Carlström w swojej pracy [10] porównali aplikacje stworzone przy użyciu technologii natywnych napisanych w języku Swift i Java oraz aplikacji crossplatformowej stworzonej za pomocą technologii React Native. Na podstawie przeprowadzonych badań autorzy stwierdzili, że narzędzie React Native jest mniej

wydajne od technologii natywnych pod względem zużycia procesora i pamięci RAM. Ponadto badania wykazały, że przy użyciu tej technologii da się stworzyć identyczny interfejs użytkownika, jednak nie jest on tak płynny i responsywny jak w przypadku natywnych technologii, co szczególnie widać przy animacjach.
Kolejną publikacją, która porusza podobny problem jest praca M. Rodríguez-Sánchez Guerra [11], w którym porównane zostały ze sobą aplikacje crossplatformowe stworzone przy użyciu następujących technologii: React Native, Ionic i Flutter. Badania polegały na przeprowadzaniu prostych operacji CRUD (ang. Create, Read, Update, Delete) na liście elementów wyświetlanej na ekranie i zmierzenie czasu poszczególnych operacji. Wyniki badań pokazały wyraźną przewagę Fluttera nad pozostałymi technologiami we wszystkich testowanych przypadkach i na obu platformach systemowych.
W ramach pracy postawiono trzy hipotezy badawcze w celu zweryfikowania ich poprawności:

1. Aplikacja stworzona przy użyciu natywnej technologii jest bardziej wydajna pod względem czasowym oraz zużycia procesora w przypadku operacji na plikach.
2. Aplikacja stworzona przy użyciu natywnej technologii jest bardziej wydajna pod względem czasowym oraz zużycia procesora w przypadku operacji na bazie danych.
3. Aplikacja stworzona przy użyciu crossplatformowej technologii jest bardziej wydajna pod względem czasowym oraz zużycia procesora w przypadku operacji sortowania liczb.

3. Metoda badawcza

Celem przeprowadzenia badań zaprojektowano i stworzono dwie aplikacje mobilne przeznaczone dla systemu iOS o identycznych funkcjonalnościach przy użyciu technologii Swift i Flutter. Aby porównać wydajność tych aplikacji zdefiniowano poniższe scenariusze badawcze:

- sortowanie 10000, 100000 i 1000000 losowo wygenerowanych liczb całkowitych z zakresu <0,1000> algorytmem Quicksort,
- zapis 10000, 100000 i 1000000 losowo wygenerowanego łańcucha znaków do pliku tekstowego,
- odczyt łańcucha znaków o długości 10000, 100000 i 1000000 z pliku tekstowego,
- zapis 1000, 10000, 100000 rekordów do bazy danych SQLite.
- odczyt 1000, 10000, 100000 rekordów z bazy danych SQLite.

Każdy scenariusz został wykonany 20 razy.

3.1. Środowisko badawcze

Do zbadania czasu wykonania scenariuszy badawczych wykorzystane zostały dwie aplikacje na platformę iOS o tych samych funkcjonalnościach. Aplikacje rejestrowały i wyświetlały czasy wykonania poszczególnych operacji. Ponadto do zbadania zużycia procesora podczas wykonywania operacji użyto narzędzia Activity Monitor. Urządzeniem testowym, na którym zostały przeprowadzone badania był smartfon Apple iPhone 7. Szczegółowa specyfikacja urządzenia została przedstawiona w Tabeli 1.

Tabela 1: Parametry urządzenia mobilnego, na którym przeprowadzono badania

Model	Apple iPhone 7
System operacyjny	iOS 12
Pamięć RAM	2GB
Procesor	Apple A10 Fusion
Liczba rdzeni	4
Taktowanie procesora	2,34GHz
Pamięć wbudowana	32GB

Wyżej opisane urządzenie zostało podłączone kablem typu Lightning do komputera Apple MacBook Air M1 i za jego pomocą zostały zainstalowane poszczególne aplikacje. Połączenie z komputerem zostało również wykorzystane do użycia narzędzia Activity Monitor w celu zmierzenia zużycia procesora podczas wykonywania poszczególnych operacji. W Tabeli 2 zostały przedstawione szczegółowe parametry komputera.

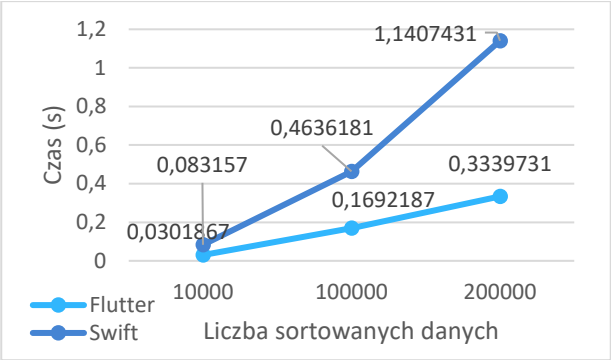
Tabela 2: Parametry komputera wykorzystanego do badań

Model	Apple MacBook Air M1
System operacyjny	macOS Monterey 12.1
Pamięć RAM	16GB
Procesor	Apple M1
Liczba rdzeni	8
Taktowanie procesora	3,2GHz
Pamięć wbudowana	256GB

4. Analiza wyników

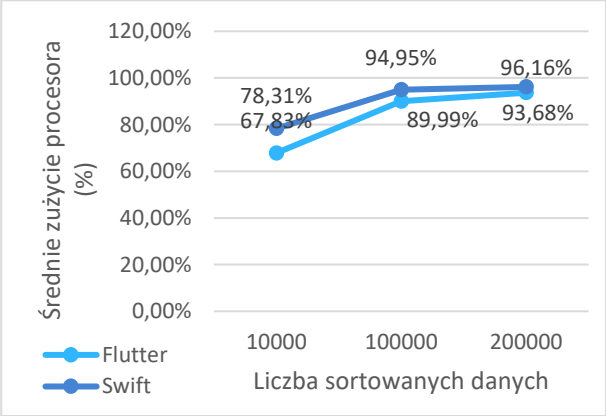
4.1. Sortowanie liczb całkowitych

Na Rysunku 1 przedstawiono średni czas wykonania sortowania w zależności od liczby danych. Można zauważyć, że aplikacja stworzona przy użyciu technologii Flutter wykonała scenariusz sortowania szybciej niż aplikacja stworzona przy użyciu języka Swift. Najmniejszą różnicę czasową uzyskano przy sortowaniu 10000 liczb i wyniosła ona ok. 0,05s, co daje ok. 275% na korzyść aplikacji crossplatformowej, natomiast największa różnica została uzyskana podczas sortowania 200000 liczb i wyniosła 0,8s, co oznacza, że aplikacja natywna była wolniejsza o ok. 341% w porównaniu do aplikacji stworzonej przy użyciu technologii Flutter. Można zauważyć, że różnica czasowa rośnie wraz z liczbą sortowanych liczb.



Rysunek 1: Średni czas sortowania liczb.

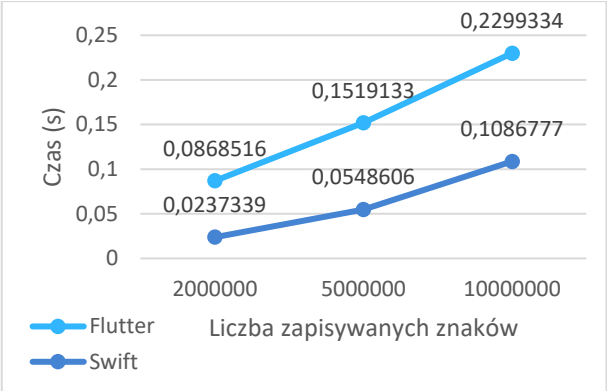
Na Rysunku 2 przedstawiono średnie zużycie procesora podczas sortowania w zależności od ilości liczb. Można zauważyć, że aplikacja crossplatformowa uzyskała niższe procentowe obciążenie jednostki obliczeniowej niż aplikacja natywna. Różnice wyniosły odpowiednio ok. 10,5% w przypadku 10000 liczb, ok. 5% w przypadku 100000 liczb i ok. 2,5% w przypadku 200000 liczb. Można zatem zauważyć, że różnice w zużyciu między aplikacjami maleją wraz ze wzrostem ilości sortowanych liczb.



Rysunek 2: Średnie zużycie procesora podczas sortowania liczb.

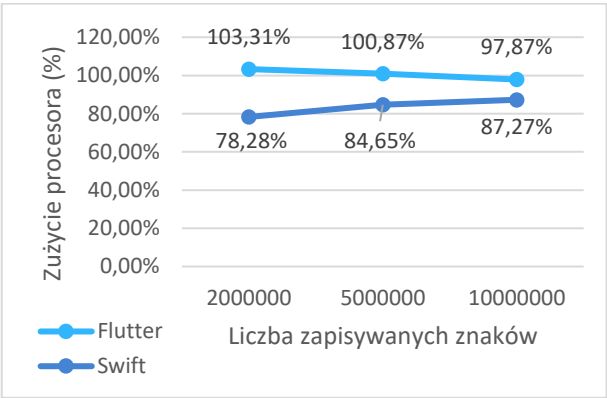
4.2. Zapis znaków do pliku

Na Rysunku 3 przedstawiono średni czas zapisu ciągu znaków do pliku z rozszerzeniem „.txt”. Można zauważyć, że aplikacja stworzona przy użyciu technologii natywnej szybciej wykonała ten scenariusz badawczy. Najmniejszą różnicę czasową uzyskano podczas zapisu 2000000 znaków i wyniosła ona ok. 0,043s, co daje ok. 365% na korzyść aplikacji natywnej, natomiast największa różnica czasowa została uzyskana w przypadku zapisu 10000000 znaków i wyniosła ona 0,12s, co oznacza, że aplikacja crossplatformowa była wolniejsza o ok. 211% w porównaniu do aplikacji stworzonej przy użyciu języka programowania Swift. Można zauważyć, że różnice w czasie wykonania pomiędzy aplikacjami rosną wraz ilością zapisywanych znaków.



Rysunek 3: Średni czas zapisu znaków do pliku.

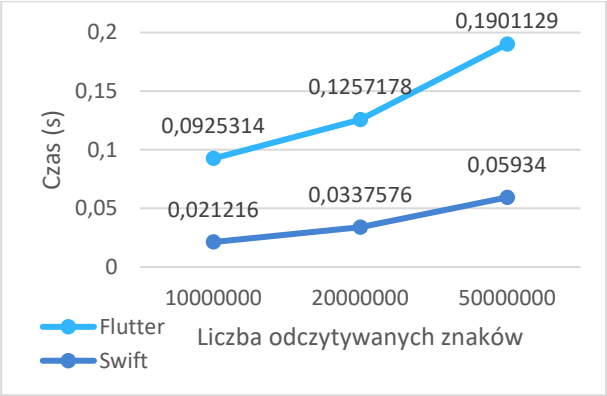
Na Rysunku 4 przedstawiono średnie zużycie procesora podczas zapisu ciągu znaków do pliku. Można zaobserwować, że aplikacja stworzona przy użyciu języka programowania Swift potrzebowała mniej czasu procesora i uzyskała niższe wyniki procentowe obciążenia jednostki obliczeniowej. Różnice w wynikach pomiędzy aplikacjami zmniejszają się wraz ze wzrostem liczby zapisywanych znaków, a największa różnica była przy 2000000 znaków i wyniosła 25%. Warto również podkreślić, że aplikacja crossplatformowa do wykonania zapisu wykorzystała drugi rdzeń procesora o czym świadczy średnie zużycie przekraczające 100%.



Rysunek 4: Średnie zużycie procesora podczas zapisywania znaków do pliku.

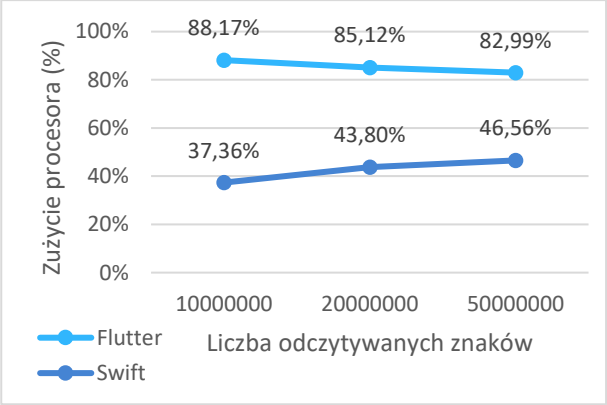
4.3. Odczyt znaków z pliku

Na Rysunku 5 przedstawiono średni czasu odczytu ciągu znaków z pliku o rozszerzeniu „.txt”. Można zauważyć, że aplikacja wytworzona przy użyciu technologii natywnej szybciej wykonała odczyt z pliku. Najmniejszą różnicę czasową uzyskano podczas odczytu 10000000 znaków i wyniosła ona ok. 0,07s, co daje ok. 436% na korzyść aplikacji natywnej, natomiast największa różnica czasowa została uzyskana w przypadku 50000000 znaków i wyniosła ok. 0,13s, co oznacza, że aplikacja crossplatformowa była wolniejsza o ok. 320% w porównaniu do aplikacji natywnej. Można zauważyć, że różnice między aplikacjami procentowo maleją wraz ze wzrostem liczby odczytywanych znaków.



Rysunek 5: Średni czasu wykonania odczytu znaków z pliku.

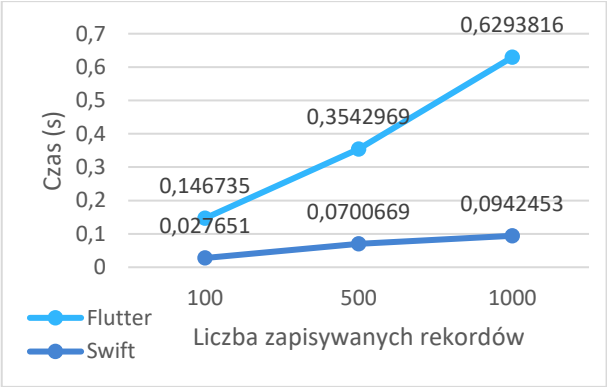
Na Rysunku 6 przedstawiono wykres średniego zużycia procesora podczas odczytu znaków z pliku w zależności od liczby odczytywanych znaków. Można zaobserwować, że aplikacja natywna uzyskała znacznie niższe wyniki zużycia jednostki obliczeniowej. Największą różnicę widać w przypadku odczytu 10000000 znaków i wyniosła ona aż 50%. Dysproporcje w zużyciu procesora między aplikacjami zmniejszają się wraz ze wzrostem liczby odczytywanych znaków.



Rysunek 6: Średnie zużycie procesora podczas odczytu znaków z pliku.

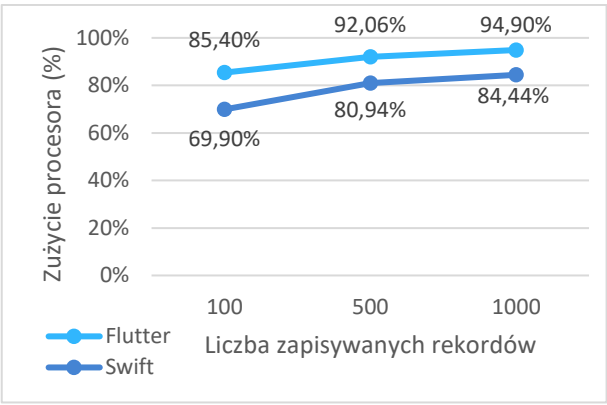
4.4. Zapis rekordów do bazy danych

Na Rysunku 7 przedstawiono średni czas zapisu rekordów do bazy danych SQLite w zależności od ich liczby. Można zaobserwować, że aplikacja zaimplementowana przy użyciu języka programowania Swift szybciej wykonała zapis rekordów do bazy danych. Najmniejszą różnicę czasową uzyskano w przypadku zapisu 100 rekordów i wyniosła ona ok. 0,12s, co daje ok 530% na korzyść aplikacji natywnej, natomiast największa różnica została uzyskana w przypadku 1000 rekordów i wyniosła ok. 0,53s, co oznacza, że aplikacja crossplatformowa była wolniejsza o ok. 667%. Można zauważyć, że różnice w wynikach między aplikacjami rosną wraz ze wzrostem liczby zapisywanych rekordów.



Rysunek 7: Średni czas wykonania zapisu rekordów do bazy danych.

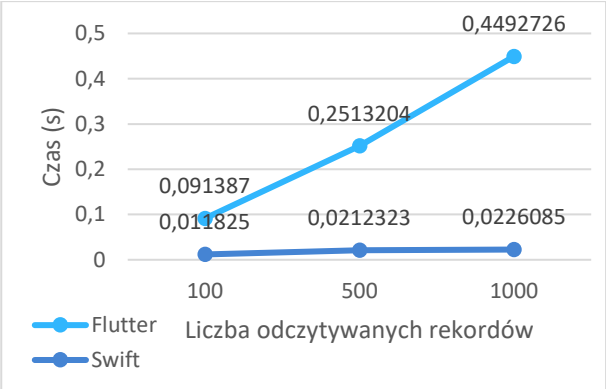
Na Rysunku 8 przedstawiono średnie zużycie procesora podczas zapisu rekordów do bazy danych w zależności od ich liczby. Można zaobserwować, że aplikacja stworzona przy użyciu technologii natywnej uzyskała niższe zużycie procesora dla wszystkich zestawów danych. Największą różnicę widać przy zapisie 100 rekordów i wynosi ona ok. 15%, natomiast najmniejszą różnicę uzyskano w przypadku 1000 rekordów i wyniosła ona ok. 10,5%. Można zauważyć, że dysproporcje wyników pomiędzy aplikacjami zmniejszają się wraz ze wzrostem liczby zapisywanych rekordów.



Rysunek 8: Średnie zużycie procesora podczas zapisu rekordów do bazy danych.

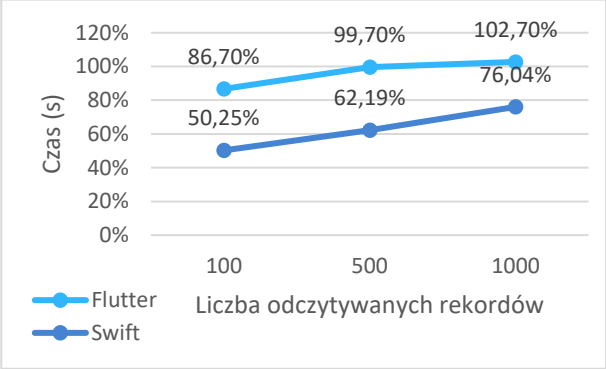
4.5. Odczyt rekordów z bazy danych

Na Rysunku 9 przedstawiono średni czas wykonania odczytu rekordów z bazy danych SQLite w zależności od ich liczby. Można na nim zaobserwować, że aplikacja stworzona przy użyciu technologii natywnej szybciej wykonała ten scenariusz badawczy dla wszystkich zestawów danych. Największą dysproporcję widać podczas odczytu 1000 rekordów i wynosi ona ok. 0,42s, czyli prawie 20 razy tyle ile wyniósł czas odczytu w przypadku aplikacji natywnej. Najmniejszą różnicę czasową uzyskano w przypadku odczytu 100 rekordów i wyniosła ona ok. 0,08s, co oznacza, że aplikacja crossplatformowa była o ok. 772% wolniejsza w porównaniu do aplikacji stworzonej przy użyciu języka programowania Swift. Można zauważyć, że różnice w czasie wykonania rosną wraz ze wzrostem liczby odczytywanych rekordów.



Rysunek 9: Średni czas wykonania odczytu rekordów z bazy danych.

Na Rysunku 10 przedstawiono wykres średniego obciążenia procesora podczas odczytu rekordów z bazy danych SQLite w zależności od ich liczby. Można na nim zauważyć, że aplikacja zaimplementowana przy użyciu języka Swift uzyskała niższe zużycie procesora dla wszystkich zestawów danych. Najmniejszą różnicę uzyskano w przypadku 1000 rekordów i wyniosła ona ok. 27%, natomiast największa różnica została uzyskana w przypadku 500 rekordów i wyniosła ona ok. 37,5%. Można zauważyć, że dysproporcje w wynikach między aplikacjami zmniejszają się wraz z liczbą odczytywanych rekordów.



Rysunek 10: Średnie zużycie procesora podczas odczytu rekordów z bazy danych.

5. Wnioski

Celem niniejszej pracy była analiza porównawcza aplikacji iOS stworzonych przy użyciu technologii natywnej, jaką jest język programowani Swift oraz technologii crossplatformowej, jaką jest Flutter. Aby zrealizować założony cel przeprowadzono badania w ramach przygotowanych wcześniej scenariuszy badawczych, dla których zmierzono czas realizacji oraz zużycie procesora podczas ich wykonywania.

W pierwszym scenariuszu zbadano operację sortowania tablicy liczb całkowitych z zakresu <1, 1000> o określonym rozmiarze. Badanie wykazało, że aplikacja crossplatformowa znacznie szybciej wykonuje operacje sortowania niezależnie od rozmiaru tablicy sortowanych liczb, a przy tym mniej obciąża jednostkę obliczeniową.

Kolejnym badanym scenariuszem był zapis ciągu znaków alfanumerycznych do pliku z rozszerzeniem „txt”. Z przeprowadzonych badań wynikło, że aplikacja natywna dużo szybciej wykonuje zapis do pliku na dysku, a przy tym znacznie mniej obciąża procesor. W przypadku aplikacji stworzonej przy użyciu technologii Flutter został zaangażowany drugi rdzeń procesora, a pomimo tego czas wykonania operacji był większy niż w aplikacji natywnej.

Aplikacja stworzona przy użyciu języka Swift okazała się szybsza i bardziej wydajna również przy kolejnym scenariuszu, jakim był odczyt ciągu znaków alfanumerycznych z pliku o rozszerzeniu „.txt”. Po przeanalizowaniu wyników stwierdzono, że technologia natywna radzi sobie lepiej z operacjami na plikach zarówno pod względem czasu wykonania i obciążenia procesora.

Ostatnie dwa scenariusze dotyczyły zapisu i odczytu rekordów z bazy danych SQLite. Z wykonanych badań wynikło, że aplikacja natywna szybciej wykonuje operacje na bazie danych, a przy tym mniej obciąża jednostkę obliczeniową.

Na podstawie przeprowadzonych badań udało się potwierdzić wszystkie postawione hipotezy. Aplikacja natywna zaimplementowana przy użyciu języka programowania Swift okazała się bardziej wydajna pod względem szybkości wykonania oraz zużycia procesora zarówno w przypadku operacji na plikach, jak i operacji na bazie danych, natomiast aplikacja crossplatformowa stworzona przy użyciu technologii Flutter uzyskała lepsze wyniki czasowe oraz mniej obciążała procesor w przypadku operacji sortowania liczb całkowitych.

Z przeprowadzonych badań wynika, że zastosowanie odpowiedniej technologii powinno być dostosowane do funkcjonalności implementowanej aplikacji.

Literatura

[1] M. Napoli, Beginning Flutter: A Hands On Guide to App Development, John Wiley & Sons, 2019

[2] Mobile & Tablet Operating System Market Share Worldwide https://gs.statcounter.com/os-market-share/mobile-tablet/worldwide/#daily-20220107-20220107-bar [dostęp: 08.01.2022]

[3] Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021 "https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/" [dostęp: 10.01.2022]

[4] Fuchsia OS Official Site https://fuchsia.dev/ [dostęp: 08.01.2022]

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

[6] D. Gałan, K. Fisz, P. Kopniak, A multi-criteria comparison of mobile applications built with the use of Android and Flutter Software Development Kits, Journal of Computer Sciences Institute 19 (2021) 107-113. https://doi.org/10.35784/jcsi.2614

[7] P. Kotarski, K. Śledź, J. Smołka, Analysis of the impact of development tools used on the performance of the mobile application, Journal of Computer Sciences Institute 6 (2018) 68-72. https://doi.org/10.35784/jcsi.642

[8] 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 7(2) (2017) 50-53. https://doi.org/10.5604/01.3001.0010.4838

[9] D. Dobrzański, W. Zabierowski, The comparison of native apps performance on iOS (Swift) and Android with cross-platform application – Xamarin: student project, International Journal of Microelectronics and Computer Science 8 (2018) 112-116

[10] O. Axelsson, F. Carlström, Evaluation Targeting React Native in Comparison to Native Mobile Development. Ergonomics and Aerosol Technology, LUP Student Papers (2016) 105 http://lup.lub.lu.se/student-papers/search/publication/8886469

[11] M. Rodríguez-Sánchez Guerra, Cross-platform development frameworks for the development of hybrid mobile applications: Implementations and comparative analysis. Escuela superior de ingenieria grado en ingenieria informatika (2018) 86. https://rodin.uca.es/handle/10498/20951