JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 25 (2022) 280–287
Received: 1 June 2022
Accepted: 17 October 2022

Comparison of hybrid and native iOS mobile application development technologies

Porównanie technologii hybrydowych i natywnych do wytwarzania aplikacji mobilnych iOS

Michał Kocki, Michał Urban*, Piotr Kopniak

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

Abstract
Mobile applications dedicated to iOS can be developed natively or hybrid. The subject of this paper is to compare these technologies on the example of a created mobile application. The performance of both technologies has been examined on the example of two test devices and smartphone operated by iOS. Considering the performance analysis, the compilation time of the application, reading and writing data from the cloud database, as well as the time of sorting the read data was examined. On the test devices, it was checked how intense is the use of the system. The obtained results confirm that producing a mobile application in native technology is more performance efficient.

Keywords: iOS; native technologies; hybrid technologies; performance

Streszczenie
Aplikacje mobilne dedykowane na system operacyjny iOS mogą być wytwarzane natywnie bądź hybrydowo. Tematyką artykułu jest porównanie tych technologii na przykładzie utworzonej aplikacji mobilnej. Zbadana została wydajność obu technologii na przykładzie dwóch urządzeń testowych i smartfona posiadającego system iOS. Biorąc pod uwagę analizę wydajności, został zbadany czas kompilacji aplikacji, odczyt i zapis danych z bazy danych w chmurze, a także czas sortowania odczytanych danych. Na urządzeniach testowych sprawdzono jak bardzo wytworzone aplikacje obciążają system. Otrzymane wyniki potwierdzają, że wytwarzanie aplikacji mobilnej w technologii natywnej jest wydajniejsze.

Słowa kluczowe: iOS; technologie natywne; technologie hybrydowe; wydajność

*Corresponding author
Email address: michal.urban@pollub.edu.pl (M. Urban)

1. Wstęp

W ostatnich latach można zauważyć wzrost liczby wytwarzanych aplikacji mobilnych. Technologie elektroniki użytkowej, takie jak telefony, tablety, zegarki czy nawet inteligentni asystenci coraz częściej pojawiają się w domach. Aby nadążyć za rozwijającymi się urządzeniami, musiały pojawić się nowe technologie, które pozwoliłyby na umiejętne i szybkie wytwarzanie aplikacji. Dzięki temu postępowi zostały rozwinięte technologie hybrydowe, które w odróżnieniu od natywnych, pozwalają na obsługę wielu systemów operacyjnych jednocześnie.

Jedną z hybrydowych technologii wytwarzania aplikacji mobilnych jest Ionic – wydany w 2013 roku framework daje programistom ogromne możliwości: wytwarzanie aplikacji internetowych i mobilnych na systemy operacyjne iOS i Android pisząc jeden kod [1].

Technologie natywne mają znacznie mniejsze możliwości w kontekście wytwarzania aplikacji na wiele platform, lecz ich popularność jest wciąż porównywalna z technologiami hybrydowymi [2]. Jedną z nich jest język programowania Swift, który został opracowany przez Apple. Prosta struktura języka połączona z intuicyjnością środowiska programistycznego Xcode pozwala na wytworzenie natywnego rozwiązania w szybkim czasie. Ten język programowania jest kompatybilny z Objective-C – językiem programowania, który był rekomendowany przez Apple przed premierą języka Swift, służącym do wytwarzania aplikacji mobilnych [3].

Niniejszy artykuł ma na celu porównanie obu wyżej wymienionych technologii – Ionic i Swift – na podstawie aplikacji mobilnej.

W badaniu porównano czas kompilacji w obu technologiach gotowej aplikacji testowej. Następnie dokonano analizy parametrów zapisu i odczytu danych z bazy danych SQLite.

Znaleziono niewielką liczbę publikacji naukowych na temat porównania technologii hybrydowych i natywnych, o sprzecznych wnioskach. Jedną z nich jest publikacja D. Dobrzańskiego i W. Zabierowskiego, w której zostaje porównana technologia Xamarin i dwie technologie natywne na systemy operacyjne iOS i Android. Wyniki tej pracy pokazują, że wydajność aplikacji pisanych w obu technologiach jest porównywalna [4].

Kontrastująca do niej jest publikacja autorstwa P. Grzmila, M. Skublewskiej-Paszkowskiej, E. Łukasik i J. Smołki [5], w której autorzy porównują aplikacje napisane natywnie na system Android, z rozwiązaniem w technologii Xamarin. Ten artykuł naukowy wyraźnie

wskazuje, że technologie natywne mają lepszą wydajność względem hybrydowych.

Artykuł naukowy autorstwa T. Dorfer, L. Demetz, S. Huber [6] wskazuje na potencjalne wady technologii hybrydowych w kontekście wykorzystania zasobów urządzeń mobilnych. Wyniki tych badań pokazują, że aplikacje w technologiach natywnych mniej obciążają urządzenie pod kątem baterii, czy też procesora i pamięci RAM.

L. Corral, A. Janes, T. Remencius wydali artykuł naukowy „Potential advantages and disadvantages of multiplatform development frameworks – A vision on mobile environments" w 2012 roku. Praca pokazuje wady i zalety technologii hybrydowych w wielu aspektach – analizowane są potrzeby biznesowe, a także wyzwania i możliwości związane z wytwarzaniem aplikacji w technologiach multiplatformowych. Autorzy prowadzą dyskusję na ten temat, która pozwala wywnioskować, że wady i zalety technologii hybrydowych są definiowane i określane przez potrzeby aplikacji, która ma być wytwarzana [7].

Wśród publikacji dotyczących porównanie technologii wytwarzania aplikacji uniwersalnych i natywnych nie odnaleziono takich, które porównują technologię hybrydową (Ionic) i natywną (Swift) na platformę iOS.

2. Metodyka badań

W celu przeprowadzenia badań, wytworzone zostały dwie bliźniacze aplikacje służące do pomiaru liczby kroków użytkownika wykonanych w ciągu dnia. Pierwszy projekt został utworzony natywnie w języku Swift, drugi w hybrydowej technologii Ionic.

Do utworzenia aplikacji oraz do ich zbadania**,** wykorzystano następujące technologie oraz narzędzia:

- środowisko programistyczne Xcode,
- środowisko programistyczne Visual Studio Code,
- język programowania Swift,
- framework Ionic Capacitor,
- framework Angular,
- język TypeScript.

Pierwsze badanie zostało wykonane na dwóch urządzeniach testowych – laptopach z zainstalowanym systemem operacyjnm macOS Monterey 12.3.1. Pierwsze urządzenie posiada procesor Intel Core i5, 2 GHz; natomiast drugie urządzenie testowe posiada procesor Intel Core i5, 1.4 GHz. Oba urządzenia posiadają 16 GB pamięci RAM.

Badanie dotyczyło czasu kompilacji gotowej aplikacji testowej, która służy do pomiaru liczby kroków wykonanych w ciągu dnia. Celem tego badania było porównanie, która technologia posiada lepszą wydajność w kwestii kompilacji gotowego rozwiązania.

Aby otrzymać odpowiednią próbkę, która posłuży do interpretacji i obiektywnych wyników badania, wykonano kompilację gotowej aplikacji 25 razy.

Badanie zostało wykonane w środowisku Xcode, po uprzednim włączeniu funkcji odczytu szczegółowego czasu kompilacji aplikacji.

Kolejne badania dotyczyły zapisu i odczytu danych z lokalnej bazy danych w aplikacjach wytworzonych w obu technologiach.

W tym przypadku wykorzystana została lokalna baza danych SQLite, która została wybrana ze względu na swoją niezawodność i popularność w środowisku aplikacji mobilnych.

Pierwszy krok badania to uruchomienie aplikacji na urządzeniu testowym i zalogowanie się. Następnie wywołano funkcję zapisu danych z bazy przez odpowiedni przycisk. Każde wywołanie funkcji zapisu powoduje usunięcie wszystkich rekordów z tabeli i zapisanie w niej 100, 10 000 oraz 100 000 nowych rekordów z losowymi danymi. Przed zapisem danych uruchamiany jest z poziomu kodu stoper, który pozwala na zmierzenie czasu wykonywania tej operacji. Czas zapisu wyświetlany jest w konsoli debugującej w Xcode.

Kolejny krok to wywołanie funkcji odczytującej dane z bazy danych, która powoduje wysłanie zapytania w języku SQL i interpretację wyniku w odpowiedniej formie. Każde uruchomienie funkcjonalności pobiera i interpretuje w zależności od badania 100, 10 000 i 100 000 rekordów.

Dodatkowo po kliknięciu przycisku w aplikacji uruchamiany jest stoper wywoływany bezpośrednio z kodu aplikacji, który pozwala zmierzyć szczegółowy czas pobierania danych.

Dla uzyskania obiektywnych i wiarygodnych wyników, zapis, a także odczyt przykładowych danych powtórzono 25 razy.

3. Badania

Eksperymenty polegały na porównaniu wydajności aplikacji napisanych w technologiach Ionic Capacitor oraz iOS Swift. Porównywany był czas kompilacji ukończonej aplikacji testowej, a także wydajność w połączeniu z bazą danych SQLite. Każde badanie zostało wykonane w 25 próbach, aby zapewnić wiarygodność wyników.

3.1. Badanie czasu kompilacji

Tabela 1 przedstawia porównanie czasu kompilacji aplikacji pomiędzy dwiema technologiami – iOS i Ionic. Można zauważyć, że kompilacja w środowisku hybrydowym jest szybsza, niż natywnie.

Technologia natywna kompiluje gotową aplikację w czasie pomiędzy 18,099 s, a 21,967 s, co daje amplitudę różnicy czasu na poziomie 3,868 s. Technologia hybrydowa Ionic ma znacznie niższy wynik w tym przypadku – różnica najwyższego i najniższego czasu wynosi 1,331 s.

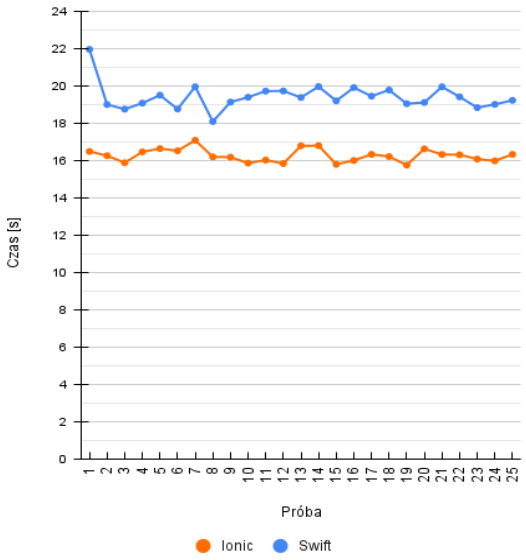
Na uwagę zasługuje również fakt, że technologia Ionic osadza swoją aplikację na widoku łączącym (Bridge Controller), który wyświetla natywnie aplikację w technologii webowej na urządzeniu mobilnym. Skutkiem jest to, że niezależnie od tego czy aplikacja jest pusta, czy ma w sobie funkcjonalności, to jej czas kompilacji jest podobny.

Technologia natywna kompiluje każdy element osobno i z każdym dodaniem nowego pakietu lub nowej funkcjonalności, czas kompilacji będzie coraz większy.

Na podstawie tych wyników możemy stwierdzić, że technologia Ionic szybciej kompiluje gotową aplikację niż Swift.

Tabela 1: Czas kompilacji w technologii Ionic i Swift

Lp.	Swift – czas [s]	Ionic – czas [s]
1	21,967	16,498
2	19,011	16,268
3	18,764	15,892
4	19,086	16,474
5	19,512	16,649
6	18,770	16,533
7	19,964	17,095
8	18,099	16,207
9	19,143	16,190
10	19,402	15,872
11	19,732	16,035
12	19,737	15,848
13	19,389	16,800
14	19,973	16,807
15	19,209	15,807
16	19,920	16,013
17	19,458	16,341
18	19,789	16,224
19	19,054	15,764
20	19,121	16,636
21	19,963	16,335
22	19,421	16,318
23	18,845	16,087
24	19,021	15,995
25	19,235	16,340



Rysunek 1: Porównanie czasu kompilacji gotowej aplikacji w obu technologiach.

3.2. Badanie czasu zapisu do bazy danych

Kolejnym etapem badań było porównanie czasów zapisu danych, który wyrażono w sekundach, które zostało wykonane na urządzeniu testowym – smartfonie iPhone 12 mini, posiadającym system iOS. Aplikacje podłączały się do lokalnej bazy danych SQL obsługiwanej przez system SQLite. Aplikacje wysyłały dane przy pomocy zapytań SQL.

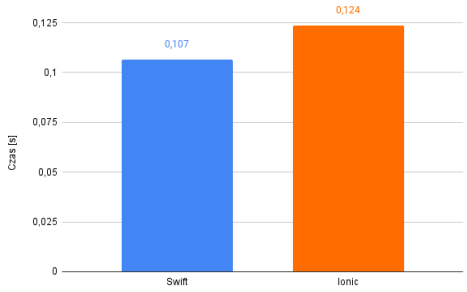
W tym przypadku udostępniane dane to liczba kroków użytkownika i identyfikator czasowy.

3.2.1. Zapis 100 rekordów do bazy

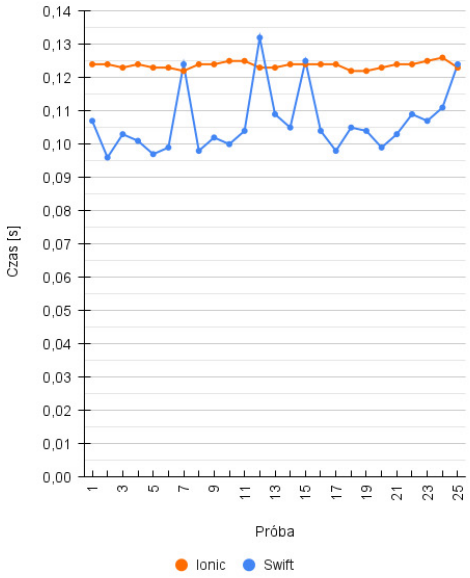
Na podstawie tabeli 2 można wywnioskować, że, czas zapisu 100 rekordów do bazy danych jest zauważalnie krótszy w przypadku języka programowania Swift. Potwierdza to rysunek 2, na którym porównano uśrednione czasy zapisu. Różnica pomiędzy badanymi technologiami jest na poziomie 14%.

Tabela 2: Czas zapisu 100 rekordów do bazy danych w technologii Ionic i Swift

Lp.	Swift – czas [s]	Ionic – czas [s]
1	0,107	0,124
2	0,096	0,124
3	0,103	0,123
4	0,101	0,124
5	0,097	0,123
6	0,099	0,123
7	0,124	0,122
8	0,098	0,124
9	0,102	0,124
10	0,100	0,125
11	0,104	0,125
12	0,132	0,123
13	0,109	0,123
14	0,105	0,124
15	0,125	0,124
16	0,104	0,124
17	0,098	0,124
18	0,105	0,122
19	0,104	0,122
20	0,099	0,123
21	0,103	0,124
22	0,109	0,124
23	0,107	0,125
24	0,111	0,126
25	0,124	0,123



Rysunek 2: Porównanie uśrednionego czasu zapisu 100 rekordów do bazy danych.

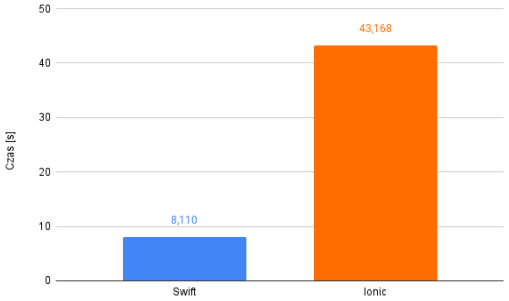


Rysunek 3: Porównanie czasów zapisu 100 rekordów do bazy danych.

3.2.2.Zapis 10 000 rekordów do bazy

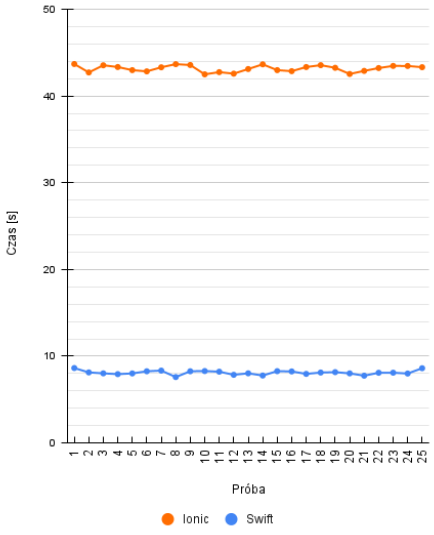
Tabela 3: Czas zapisu 10 000 rekordów do bazy danych w technologii Ionic i Swift

Lp.	Swift – czas [s]	Ionic – czas [s]
1	8,637	43,691
2	8,133	42,719
3	8,025	43,544
4	7,935	43,344
5	8,016	42,984
6	8,265	42,842
7	8,342	43,312
8	7,599	43,674
9	8,266	43,576
10	8,291	42,500
11	8,217	42,754
12	7,857	42,578
13	8,029	43,112
14	7,774	43,654
15	8,278	42,984
16	8,246	42,866
17	7,963	43,341
18	8,118	43,557
19	8,164	43,254
20	8,022	42,541
21	7,756	42,897
22	8,093	43,225
23	8,100	43,472
24	8,005	43,465
25	8,616	43,324



Rysunek 4: Porównanie uśrednionego czasu zapisu 10 000 rekordów do bazy danych w obu technologiach.

Tabela 3 prezentuje wyniki badania zapisu 10 000 rekordów do bazy danych. Można wywnioskować, że znaczną przewagę ma w tym przypadku język programowania Swift. Framework Ionic zapisuje taką liczbę rekordów średnio o 85% wolniej (rysunek 4).

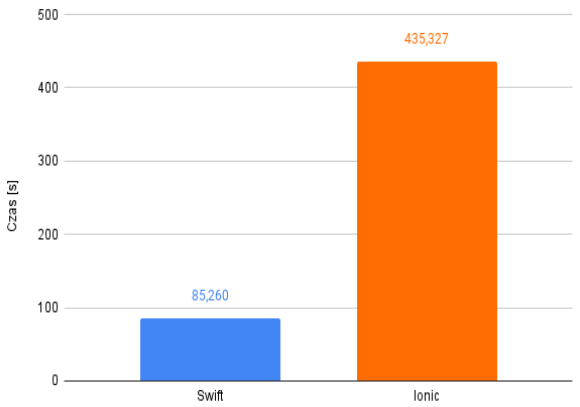


Rysunek 5: Porównanie czasów zapisu 10 000 rekordów do bazy danych.

3.2.3. Zapis 100 000 rekordów do bazy

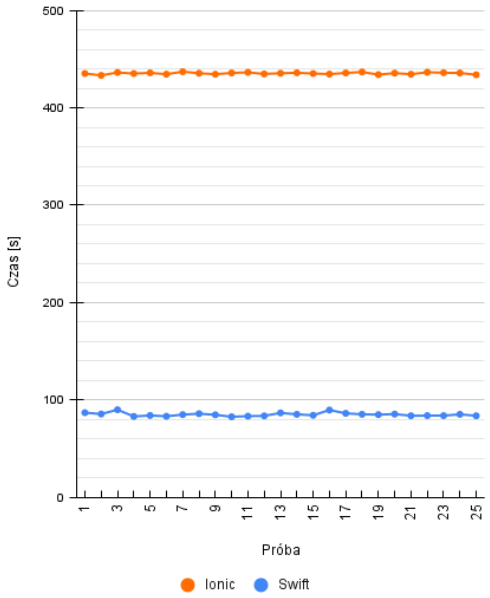
Tabela 4: Czas zapisu 100 000 rekordów do bazy danych w technologii Ionic i Swift

Lp.	Swift – czas [s]	Ionic – czas [s]
1	87,127	435,182
2	85,748	433,232
3	90,161	436,214
4	83,275	435,120
5	84,265	435,841
6	83,446	434,458
7	85,124	437,032
8	86,098	435,463
9	84,944	434,412
10	82,952	435,741
11	83,542	436,341
12	83,864	434,698
13	86,905	435,442
14	85,432	435,911
15	84,451	435,132
16	89,843	434,546
17	86,465	435,741
18	85,451	436,741
19	85,131	433,986
20	85,669	435,541
21	83,985	434,421
22	84,128	436,471
23	84,090	435,943
24	85,441	435,713
25	83,956	433,841



Rysunek 6: Porównanie uśrednionego czasu zapisu 100 000 rekordów do bazy danych w obu technologiach.

Tabela 4 prezentuje wyniki badania zapisu do lokalnej bazy danych w wymiarze 100 000 rekordów. Średnia dla języka programowania Swift to w tym przypadku 85,260 s, a dla frameworka Ionic – 435,327 s (rysunek 6). Wyniki różnią się średnio o 350 s na korzyść języka programowania Swift.



Rysunek 7: Porównanie czasów zapisu 100 000 rekordów do bazy danych.

Wyniki badania czasu zapisu w bazie danych prezentują znaczną przewagę technologii natywnej nad technologią hybrydową. Dla każdej liczby rekordów średni czas zapisu do bazy danych jest o dużo niższy w technologii Swift.

Dodatkowo, rysunek 5 pokazuje następujące amplitudy wahania czasu zapisu: 1,038 s dla aplikacji napisanej w technologii Swift; i 1,191 s dla aplikacji hybrydowej wykonanej w technologii Ionic. Wahania w obu przypadkach nie są znaczące (ok. 1 s), więc można stwierdzić, że oba rozwiązania mają podobną niezawodność oraz stabilność wykonywanych operacji.

Na podstawie powyższego badania można wywnioskować, że technologia natywna znacznie szybciej obsługuje lokalną bazę danych.

3.3. Badanie czasu odczytu danych z bazy danych

Następnie zbadano szybkość odczytu danych, która została wyrażona w sekundach. Podobnie jak dla zapisu danych, wykorzystana została lokalna baza danych oparta o SQLite. Odczytane dane zostały wypisane w konsoli debugującej aplikację. Tabele 5 oraz 6 przedstawią wyniki czasowe odczytu danych z bazy kolejno dla 10 000 i 100 000 rekordów w obu technologiach.

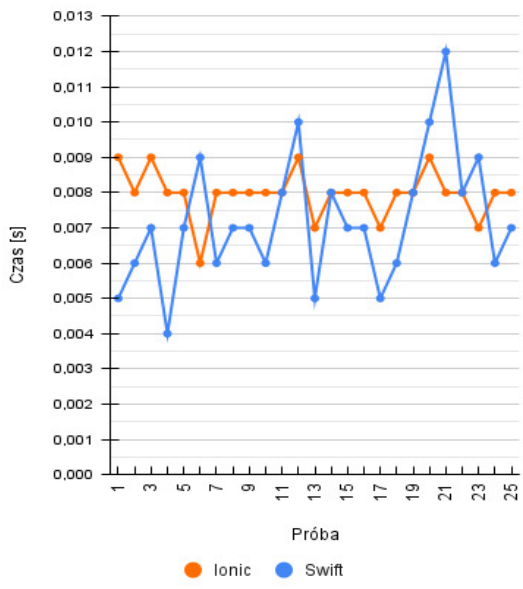
Odczyt 100 rekordów z bazy

Tabela 5: Czas odczytu 100 rekordów z bazy danych w technologii Ionic i Swift

Lp.	Swift – czas [s]	Ionic – czas [s]
1	0,005	0,009
2	0,006	0,008
3	0,007	0,009

4	0,004	0,008
5	0,007	0,008
6	0,009	0,006
7	0,006	0,008
8	0,007	0,008
9	0,007	0,008
10	0,006	0,008
11	0,008	0,008
12	0,010	0,009
13	0,005	0,007
14	0,008	0,008
15	0,007	0,008
16	0,007	0,008
17	0,005	0,007
18	0,006	0,008
19	0,008	0,008
20	0,010	0,009
21	0,012	0,008
22	0,008	0,008
23	0,009	0,007
24	0,006	0,008
25	0,007	0,008

Badanie odczytu 100 rekordów z bazy danych wskazuje nam, że obie porównywane technologie radzą sobie z tym procesem podobnie. Na podstawie rysunku 8 można wywnioskować, że czasy odczytu między technologiami mają marginalną różnicę.



Rysunek 8: Porównanie czasów odczytu 100 rekordów z bazy danych.

Na podstawie powyższych danych można obliczyć prędkość odczytu rekordów. Swift (prędkość odczytu na poziomie 14285 rekordów na sekundę) w tym przypadku również wygrywa z frameworkiem Ionic (prędkość odczytu na poziomie 12500 rekordów na sekundę), osiągając wynik o 25% wyższy.

3.3.1. Odczyt 10 000 rekordów z bazy

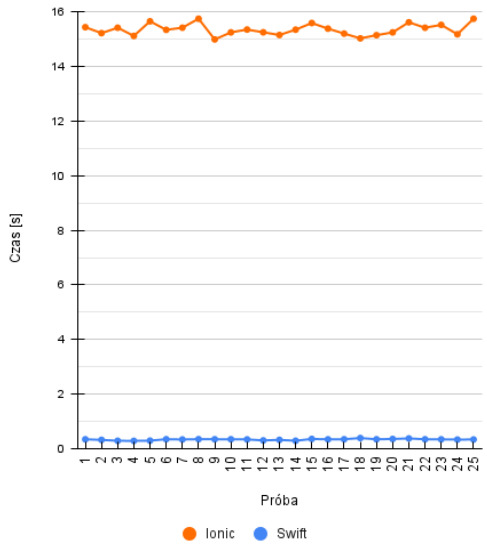
Tabela 6: Czas odczytu 10 000 rekordów z bazy danych w technologii Ionic i Swift

Lp.	Swift – czas [s]	Ionic – czas [s]
1	4,493	15,432
2	4,814	15,214
3	4,041	15,412
4	4,555	15,112
5	3,917	15,647
6	4,655	15,335
7	2,624	15,413
8	5,218	15,741
9	5,273	14,984
10	4,720	15,242
11	5,315	15,345
12	5,289	15,245
13	4,728	15,146
14	4,625	15,341
15	2,677	15,584
16	3,847	15,379
17	4,046	15,197
18	5,006	15,023
19	3,546	15,139
20	3,415	15,242
21	3,117	15,611
22	4,284	15,412
23	3,812	15,517
24	3,783	15,173
25	4,281	15,741

Czas odczytu danych różni się między sobą w poszczególnych próbach. Na podstawie rysunku 9 można wywnioskować, że aplikacja napisana w języku programowania Swift jest stabilniejsza w kwestii odczytu danych z bazy danych.

Na podstawie wyników (tabela 6) możemy również wyliczyć liczbę odczytanych rekordów na sekundę. Wynik dla języka programowania to dużo większa liczba odczytanych rekordów na poziomie 29 tysięcy rekordów na sekundę. Framework Ionic odczytuje w tym

samym czasie tylko 652 rekordy, co potwierdza przewagę natywnego rozwiązania.

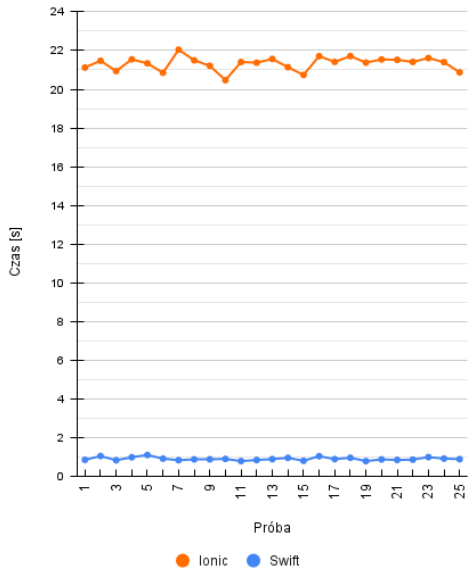


Rysunek 9: Porównanie czasów odczytu 10 000 rekordów z bazy danych.

3.3.2.Odczyt 100 000 rekordów z bazy

Tabela 7: Czas odczytu 100 000 rekordów z bazy danych w technologii Ionic i Swift

Lp.	Swift – czas [s]	Ionic – czas [s]
1	0,861	21,123
2	1,058	21,475
3	0,840	20,941
4	0,995	21,546
5	1,112	21,341
6	0,925	20,854
7	0,845	22,046
8	0,886	21,497
9	0,890	21,212
10	0,909	20,472
11	0,798	21,411
12	0,852	21,374
13	0,899	21,567
14	0,965	21,142
15	0,812	20,741
16	1,051	21,712
17	0,899	21,416
18	0,969	21,713
19	0,796	21,379
20	0,884	21,541
21	0,856	21,522
22	0,873	21,413
23	1,003	21,614
24	0,925	21,402
25	0,896	20,875



Rysunek 10: Porównanie czasów odczytu 100 000 rekordów z bazy danych.

Badanie odczytu z bazy danych wykonane na 100 000 rekordach wykazało, że język Swift podczas odczytu danych z bazy danych jest wydajniejszy od technologii hybrydowej. Framework Ionic zapisuje taką liczbę rekordów średnio o około 20 sekund wolniej od języka programowania Swift.

Na podstawie wykresu (rys. 10) możemy wywnioskować, że Swift stabilniej odczytuje dane z bazy danych.

W przypadku 100 000 rekordów, prędkości zapisu dla obu technologii mają duże różnice. Framework Ionic w przypadku odczytu takiej ilości danych jest wolniejszy i osiąga prędkość 4687 rekordów na sekundę, natomiast język programowania Swift odczytuje dane z prędkością ponad 111 tysięcy rekordów na sekundę.

Odczyt 100, 10 000, a także 100 000 rekordów z bazy danych, zbadany zarówno dla języka programowania Swift, jak i frameworku Ionic potwierdza, że język programowania Swift posiada szybszy odczyt z bazy danych niż framework Ionic.

4. Wnioski

W artykule została przedstawiona analiza porównawcza technologii natywnej z wykorzystaniem języka Swift oraz hybrydowej Ionic na przykładzie autorskiej aplikacji mobilnej z identycznymi funkcjonalnościami. Aplikacja została przetestowana pod względem czasu

kompilacji, wydajności w połączeniu z lokalną bazą danych SQLite, mechanizmów sortowania oraz zużycia zasobów urządzenia.

Na podstawie wyników, które udało się otrzymać podczas badania, można stwierdzić, że aplikacja wytworzona natywnie kompiluje się wolniej, niż aplikacja napisana w technologii hybrydowej. Natomiast biorąc pod uwagę połączenie z lokalną bazą danych, można zdecydowanie stwierdzić, że technologia natywna Swift jest szybsza zarówno podczas zapisu, jak i odczytu danych z SQLite.

Jeżeli chcemy zdecydować się na tworzenie aplikacji tylko na system iOS z pominięciem systemu Android, szybszym rozwiązaniem jest wybranie technologii natywnej. W przypadku, gdy nasza aplikacja nie polega na funkcjach obciążających zasoby jak np. streaming to lepszym wyborem jest aplikacja hybrydowa. Jeden deweloper jest w stanie stworzyć jeden kod na system iOS, Android, aplikacje internetową oraz desktopową. Wystarczy specjalista od technologii webowych co jest dużą oszczędnością budżetu porównując do zatrudnienia czterech deweloperów specjalizujących się w każdej z wymienionych technologii.

Literatura

[1] Introduction to Ionic, https://ionicframework.com/docs#license, [18.01.2022]

[2] T. Vilček, T. Jakopec, Comparative analysis of tools for development of native and hybrid mobile applications, 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (2017) 1516-1521.

[3] The Swift Programming Language Documentation, https://docs.swift.org/swift-book/index.html, [18.01.2022]

[4] D. Dobrzański, W. Zabierowski, The Comparison of Native Apps Performance on iOS (Swift) and Android with Cross-platform Application – Xamarin. IJMCS Journal 8(3) (2017) 122-126.

[5] 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 (2017) 50-53.

[6] T. Dorfer, L. Demetz, S. Huber, Impact of mobile cross-platform development on CPU, memory and battery of mobile devices when using common mobile app features, Procedia Computer Science 175 (2020) 189-196.

[7] L. Corral, A. Janes, T. Remencius: Potential advantages and disadvantages of multiplatform development frameworks – A vision on mobile environments, Procedia Computer Science 10 (2012) 1202–1207.