JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 28 (2023) 217–222
Received: 25 May 2023
Accepted: 20 June 2023

Performance comparison of Flutter platform GUI in web and native environments

Porównanie wydajności interfejsu graficznego platformy Flutter w środowisku webowym i natywnym

Juliusz Piskor*, Marcin Badurowicz

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

Abstract
This article compares the performance of the Flutter framework's graphical user interface (GUI) in a Windows platform native environment and in a web environment on the same device. The purpose of the article is to make a comparative analysis of screen building and rebuilding times in both environments. For the purpose of the paper, a test application has been created in which screens of different types of complexity are included. The stated theses: "The Flutter platform is less efficient in the web environment compared to the native environment in terms of view loading times." and "The Flutter platform is less efficient in a web environment compared to a native environment in terms of view rebuilding time" have been proven.

Keywords: performance comparison; multi-platform; GUI performance

Streszczenie
Artykuł dotyczy porównania wydajności interfejsu graficznego frameworka Flutter w środowisku natywnym na platformie Windows oraz na środowisku webowym, na tym samym urządzeniu. Celem artykułu jest dokonanie analizy porównawczej czasów budowy i przebudowy ekranów na obydwu środowiskach. Na potrzeby pracy została wykonana aplikacja testowa w której skład wchodzą ekrany o różnym typie złożoności. Postawione tezy: “Platforma Flutter jest mniej wydajna w środowisku webowym w porównaniu do środowiska natywnego pod względem czasu ładowania widoków.” oraz “Platforma Flutter jest mniej wydajna w środowisku webowym w porównaniu do środowiska natywnego pod względem czasu przebudowywania widoków” zostały udowodnione.

Słowa kluczowe: porównanie wydajności; wieloplatformowość; wydajność interfejsu graficznego

*Corresponding author
Email address: juliusz.piskor@pollub.edu.pl (J. Piskor)

1. Wstęp

Planowanie projektu informatycznego jest niezwykle ważnym etapem w procesie tworzenia oprogramowania [1]. W dzisiejszych czasach, kiedy rynek technologiczny jest bardzo konkurencyjny, kluczowe jest, aby projekt był nie tylko innowacyjny i funkcjonalny, ale także dostosowany do potrzeb użytkowników.

Właściwie zaplanowany projekt IT może zminimalizować koszty, przyspieszyć czas realizacji oraz przyczynić się do powodzenia biznesowego projektu. Dlatego też, uwzględnienie etapu planowania jest niezbędne dla zespołów programistycznych, aby projekt IT był zrealizowany w sposób efektywny i zgodny z oczekiwaniami klienta. W tym kontekście, wybór odpowiednich narzędzi, technologii oraz zasobów, w tym wybór platformy docelowej, stanowi kluczowe zagadnienie, które musi być uwzględnione już na etapie planowania projektu IT.

Kilkanaście lat temu głównymi pytaniami jakie zadawały sobie zespoły programistów podczas procesu planowania danego projektu były te, związane z wykorzystaniem technologii oraz, co za tym idzie, platform docelowych dla tworzonych projektów [2]. Jednym z problemów, który często pojawiał się w kontekście takiego wyboru, gdy okazał się on błędny w trakcie trwania projektu, było trzymanie się go przez twórców do samego końca, ze względu na nieopłacalność zmiany wyboru w trakcie pracy nad nim. Wynikało to z faktu, że zmiana narzędzi i technologii wiązała się z koniecznością ponoszenia dodatkowych kosztów związanych m.in. z przeprowadzeniem szkoleń dla zespołu programistycznego lub zakupem licencji na nowe oprogramowanie. Prowadzić mogło to do sytuacji w której menedżerowie projektów byli świadomi, że wcześniejszy wybór platformy docelowej był błędny, a mimo tego zmuszeni byli dokończyć dany projekt.

W odpowiedzi na przedstawione problemy coraz większą popularnością cieszą się technologie wieloplatformowe. Dzięki nim możliwe jest zmniejszenie wielkości zespołów programistycznych, redukcja czasu i zasobów potrzebnych do tworzenia projektu [3], a co kluczowe w kontekście poniższej pracy, możliwe jest jak największe przesunięcie w czasie decyzji o wyborze platformy dla oprogramowania, ponieważ decyzja ta może zapaść już po rozpoczęciu prac nad aplikacją. Jedną z takich technologii jest framework Flutter, który pozwala na kompilację aplikacji na platformy takie jak

Windows, Linux, MacOS, Android, iOS lub web przy pomocy jednej bazy kodowej.

Celem niniejszego artykułu jest zbadanie czasu ładowania i odpowiedzi interfejsu graficznego platformy Flutter w celu sprawdzenia, czy aplikacje desktopowe oraz webowe są w takim samym stopniu wydajne. Zostanie to sprawdzone na podstawie przygotowanej aplikacji na potrzeby badań. W celu potwierdzenia hipotez, aplikacja zostanie uruchomiona i przeanalizowana na systemie Windows oraz jako aplikacja webowa uruchamiana na tym samym urządzeniu. Na tej podstawie postawione zostały następujące hipotezy:

H1. Platforma Flutter jest mniej wydajna w środowisku webowym w porównaniu do środowiska natywnego pod względem czasu ładowania widoków,
H2. Platforma Flutter jest mniej wydajna w środowisku webowym w porównaniu do środowiska natywnego pod względem czasu przebudowywania widoków.

2. Przegląd literatury

W środowisku naukowym istnieje wiele artykułów dotyczących porównania wydajnościowego aplikacji na różnych platformach docelowych. Można znaleźć także prace, które bezpośrednio porównują ze sobą framework Flutter z konkurencyjnymi rozwiązaniami pod względem wydajności i responsywności. Analiza poniższych artykułów pomóc może głębiej przyjrzeć się wydajności frameworku oraz dowiedzieć się czego można od niego oczekiwać odnosząc się do jego wydajności.

W pracy [4] zbadano wydajność graficzną platformy Flutter oraz zestawiono wyniki z podobnymi badaniami opierających się na testowaniu platformy Cordova. Skupiono się tam głównie na urządzeniach mobilnych - nie brano pod uwagę platform natywnych oraz webowych. Wyniki badań przerosły oczekiwania twórców i udowodniły przewagę Fluttera nad starszymi mobilnymi technologiami wieloplatformowymi.

Podobne badania wykonano w pracy [5]. Skupiono się tam nad porównaniem Fluttera oraz jego bezpośredniej konkurencji - Reacta Native. Wykazano wówczas, że nieznacznie mniej klatek animacji gubionych jest przez framework od firmy Facebook. Autor podkreśla jednak, że o ile różnica jest zauważalna, to nie jest ona znacząca.

Autor pracy [6] przeanalizował responsywność aplikacji natywnych i porównał ją z aplikacjami webowymi. Przy wykorzystaniu autorskich aplikacji wykonał badania, które wskazały, że lepszym wyborem przy tworzeniu aplikacji, która będzie intensywnie wykorzystywać zasoby, będzie platforma natywna.

Jednym z wniosków jednak było to, że aplikacje webowe, które korzystają z zewnętrznych treści, np. poprzez wykorzystywanie API, a nie takie, które same przeprowadzają obliczenia mogą być bliskie w wydajności do swoich natywnych odpowiedników.

W pracy [7] wykonano podobne badania. Skupiono się jednak tam nie tylko na aspektach wydajności interfejsu - badano także wydajność sieciową oraz wykorzystywanie baterii urządzenia. Autorzy pracy wykazali, że około 70% badanych aplikacji jest w każdym aspekcie bardziej wydajna w formie natywnej w porównaniu do swojego odpowiednika webowego.

W pracy [8] z 2020 roku autorka dokonała porównania aplikacji stworzonych na platformach natywnych oraz przy użyciu frameworka Flutter. Analizie zostały poddane dane zużycia procesora podczas korzystania z aplikacji. Wniosek, który został wyciągnięty przez autorkę z tych badań jest taki, że Flutter jest idealnym rozwiązaniem jeśli chodzi o tworzenie małych i średnich aplikacji z myślą o wydajności, ale przy dużych ustępuje rozwiązaniom natywnym, choć jak sama podkreśla, zauważalny jest potencjał w prześcignięciu wydajnościowym nawet przy większych aplikacjach.

Z kolei w pracy [9] autorzy porównali ze sobą pod względem wydajnościowym platformy Flutter oraz Xamarin, który także jest jednym z konkurencyjnych rozwiązań. Początkowo postawiona hipoteza mówiąca o tym, że Xamarin jako bardziej dojrzała technologia będzie średnio działać lepiej, ale badania potwierdziły że Flutter w większości aspektów był tak samo dobry, a często lepszy od swojego starszego konkurenta. Jedynym polem na którym Xamarin wypadł lepiej był test zużycia pamięci RAM.

W pracy [10] autor porównał ze sobą autorskie aplikacje na platformę Windows. Jedna z nich była napisana w technologii Flutter, druga zaś, korzystając z bardziej klasycznego rozwiązania w produkcji aplikacji desktopowych - w technologii .NET. Badania wydajnościowe, które przeprowadził wskazały, że aplikacje korzystające z frameworka do Google okazały się znacznie lepiej radzące sobie jeśli chodzi o rozruch programu i zużycie pamięci ale w renderingu obrazów przy użyciu CPU ustępują innym rozwiązaniom.

Dokonany przegląd literatury jednoznacznie wskazał, że Flutter został dość dobrze zbadany wydajnościowo w kontraście do swojej konkurencji na rynku cross-platformowych technologii, jak i do natywnych rozwiązań na różnorakich platformach. Wyniki badań i stojące za nimi wnioski przeważnie podkreślały że Flutter przerasta oczekiwania w kontekście wydajności i widoczna jest jego przewaga w tym aspekcie nad innymi rozwiązaniami.

Wiele wskazuje na to, że w środowisku brakuje prac, które bezpośrednio zajmują się porównaniem wydajnościowym wspomnianego produktu z nim samym z kontekście platform przez niego obsługiwanych.

Flutter, jako platforma do tworzenia aplikacji zarówno internetowych, mobilnych, jak i na komputery stacjonarne, może zachowywać się inaczej na tych samych komputerach w kontekście środowiska wykonawczego.

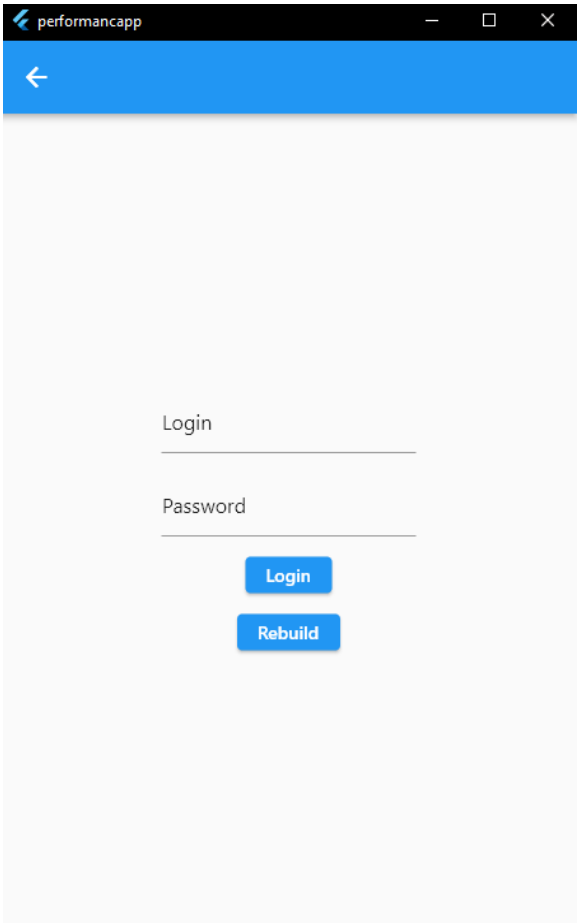
3. Implementacja aplikacji testowej

Na potrzeby badań wydajnościowych została wykonana aplikacja przy wykorzystaniu badanego frameworka.

Dzięki niej, możliwe będzie zbadanie czasu budowy i przebudowy dostępnej w niej ekranów. Projekt oparty został na Flutterze w wersji 3.7.0 umożliwiającej kompilację kodu do postaci zarówno webowej jak i desktopowej.

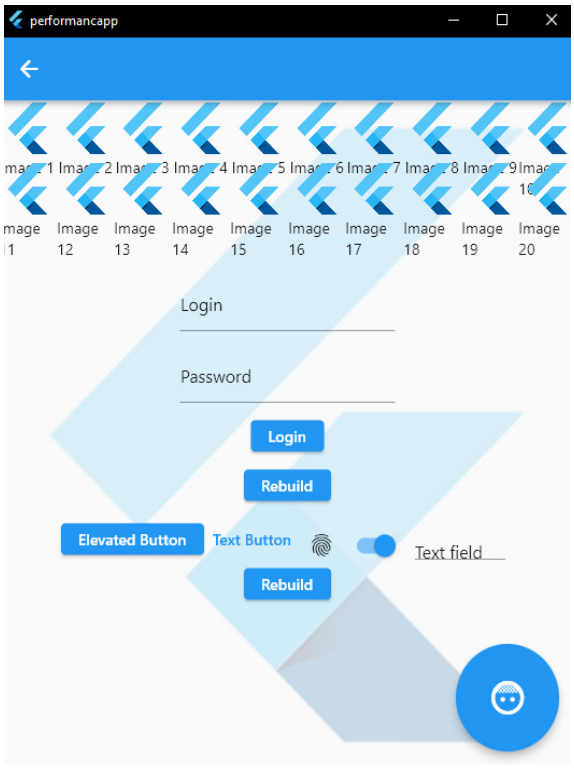
Aplikacja zawiera dwa ekrany, które będą poddane badaniom - ekran o mniejszym i większym poziomie skomplikowania. Dla upewnienia się że ekrany różnią się od siebie stopniem złożoności, a nie tylko są od siebie różne, zawierają one część wspólną – standardowy formularz logowania oraz przycisk, który pozwoli przebudować dany ekran na żądanie.

Ekran o mniejszej złożoności (Rysunek 1) został przygotowany zgodnie z wytycznymi optymalizacyjnymi dostępnymi w dokumentacji samego frameworku [11] i składa się tylko z formularza logowania oraz przycisku przebudowywania, wspomnianej wcześniej części wspólnej.



Rysunek 1: Interfejs ekranu o mniejszej złożoności.

Drugi z ekranów, który dostępny jest w aplikacji testowej, jest ten o wysokim skomplikowaniu (Rysunek 2). W jego skład, poza częścią wspólną, wchodzi szereg widgetów, których dodanie ma spowolnić czas budowy interfejsu graficznego. Wspomniane widgety zostały dobrane pod względem ich „ciężkości", co oznacza że powinny one mieć duży wpływ na szybkość budowy GUI. Przykładami takich widgetów mogą być Opacity lub ListView [12,13]. Znane są one ze swojej nieoptymalności i z reguły unikane są przy tworzeniu aplikacji komercyjnych.



Rysunek 2: Interfejs ekranu o większej złożoności.

4. Metodyka badawcza

Metodyka badawcza pozwalająca przeprowadzić badania dające odpowiedź na przedstawione wcześniej hipotezy składa się z ośmiu scenariuszy badawczych, które sprawdzą czas wstępnej budowy oraz późniejszej przebudowy graficznego interfejsu użytkownika przygotowanej aplikacji. Badania zostały przeprowadzone na dwóch ekranach i na dwóch platformach docelowych. Zależnie od scenariusza badawczego, zmierzony został czas wstępnej budowy ekranu po przejściu do niego z ekranu głównego, lub w przypadku drugiego scenariusza, czas przebudowy bo naciśnięciu odpowiedniego przycisku przebudowującego ekran.

Do określenia dokładnych czasów wykorzystane były narzędzia dostarczone, w przypadku aplikacji desktopowej, przez sam framework, a w przypadku aplikacji webowej, przez przeglądarkę internetową. Narzędzia o których mowa to odpowiednio: Dart DevTools oraz narzędzia deweloperskie do badania wydajności Google Chrome. Informacje w nich zawarte pokazały ile czasu zajmują poszczególne akcje wykonywane na aplikacji – w tym przebudowa i wstępna budowa ekranów.

Wszystkie próby badawcze były realizowane na następującym systemie:

- Procesor- Intel i7-4790k;
- Karta graficzna - Nvidia GeForce GTX 1070;
- RAM - DDR3 16 GB 1600 Mhz;

- Dysk - SSD Samsung EVO 1TB;
- System operacyjny - Windows 10 22H2.

Przeprowadzone testy były prowadzone w warunkach, której umożliwiły jak najbardziej dokładne pomiary, tj. bez aplikacji działających w tle i z odłączonym połączeniem internetowym. Dla większej dokładności wyników badania powtórzone po 20 razy dla każdego z przypadków. Odstęp czasowy pomiędzy każdym testem wynosił 5 minut.

5. Wyniki badań

Wyniki badań wydajnościowych zostały przedstawione w Tabelach 1,2,3 i 4. Pierwszym z badanych przypadków był czas pierwszej budowy ekranów.

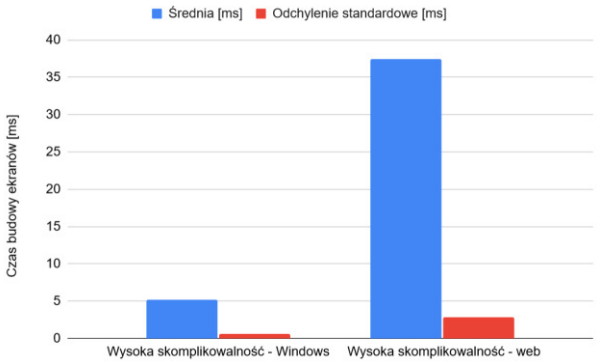
Tabela 1: Średni czas budowy ekranu o niskim skomplikowaniu na badanych platformach

Liczba prób: 20	Średni czas uruchomienia [ms] ± odchylenie standardowe
Windows	**1,87 ± 0,162**
Web	**21,47 ± 2,268**

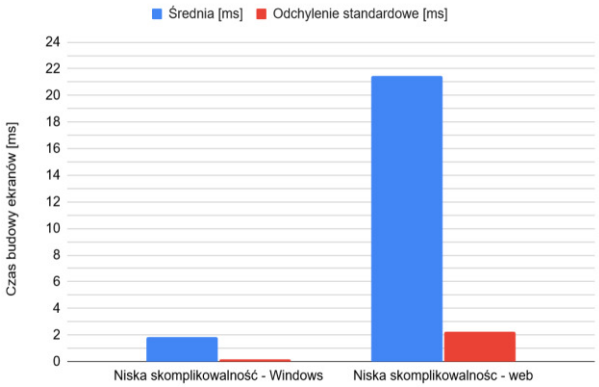
Tabela 2: Średni czas budowy ekranu o wysokim skomplikowaniu na badanych platformach

Liczba prób: 20	Średni czas uruchomienia [ms] ± odchylenie standardowe
Windows	**5,18 ± 0,598**
Web	**37,45 ± 2,873**

Zestawienie wyników badań wydajnościowych dla budowy ekranów dla poszczególnych widoków i platform zostało przedstawione na Rysunkach 3 i 4.



Rysunek 3: Testy wydajnościowe budowy ekranu o wysokim skomplikowaniu na badanych platformach.



Rysunek 4: Testy wydajnościowe budowy ekranu o niskim skomplikowaniu na badanych platformach.

Drugim z badanych przypadków był czas przebudowy każdego z ekranów.

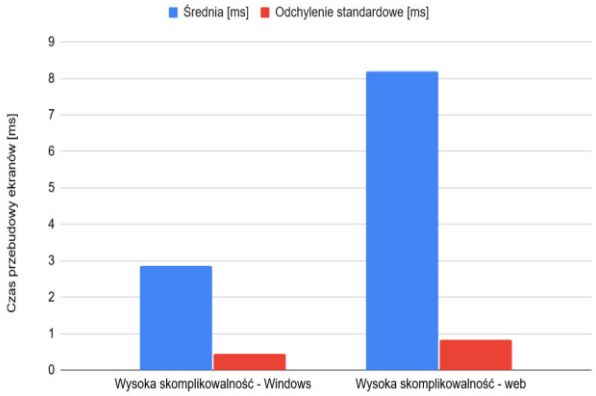
Tabela 3: Średni czas przebudowy ekranu o niskim skomplikowaniu na badanych platformach

Liczba prób: 20	Średni czas przebudowy [ms] ± odchylenie standardowe
Windows	**1,84 ± 0,303**
Web	**4,33 ± 0,889**

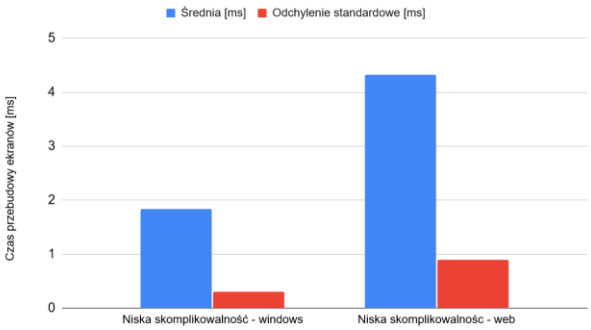
Tabela 4: Średni czas przebudowy ekranu o wysokim skomplikowaniu na badanych platformach

Liczba prób: 20	Średni czas przebudowy [ms] ± odchylenie standardowe
Windows	**2,86 ± 0,458**
Web	**8,2 ± 0,827**

Zestawienie wyników badań wydajnościowych dla przebudowy ekranów dla poszczególnych widoków i platform zostało przedstawione na Rysunkach 5 i 6.



Rysunek 5: Testy wydajnościowe przebudowy ekranu o wysokim skomplikowaniu na badanych platformach.



Rysunek 6: Testy wydajnościowe przebudowy ekranu o niskim skomplikowaniu na badanych platformach.

6. Wnioski

W przedstawionym artykule dokonano porównania wydajnościowego platformy Flutter w środowisku natywnym oraz webowym. Na potrzeby przeprowadzonych badań stworzono aplikację testową w której skład wchodziły ekrany o różnym typie skomplikowania. Wykonane badania polegały na zmierzeniu czasów wstępnej budowy oraz późniejszej przebudowy przygotowanych ekranów.

W przypadku badań obejmujących pomiar czasów wstępnej budowy ekranów, różnica w szybkości budowy zarówno ekranu o wyższym jak i niższym poziomie skomplikowania na badanych platformach była bardzo dobrze widoczna. Różnica ta była bardziej zauważalna w przypadku testów ekranu o niższym skomplikowaniu. Framework budował ten widok średnio 11,5 razy dłużej w środowisku webowym w porównaniu do natywnego. Nieznacznie mniejsza różnica widoczna była w przypadku ekranu o większym skomplikowaniu. Mowa tu o średnio nieco ponad 7-krotnie dłuższym czasie budowy na platformie webowej.

Podobną sytuację można zauważyć było w przypadku pomiarów czasów przebudowy ekranów, lecz zauważona różnica nie była już aż tak duża. Średni czas przebudowy ekranu o niższym skomplikowaniu był o ponad 2 razy większy w środowisku webowym w porównaniu do desktopowego. Jeśli chodzi o ekran o większym skomplikowaniu, tutaj różnica była nieznacznie większa – średnio niespełna 3 krotnie dłużej widok przebudowywał się na platformie webowej.

Przeprowadzone badania i uzyskane na ich podstawie wyniki pozwalają potwierdzić wcześniej postawione hipotezy H1 oraz H2. Warto zaznaczyć jednak, że różnice w czasie budowy i przebudowy ekranów były tym mniejsze, im bardziej skomplikowane były widoki. Niewykluczone jest, że w przypadku aplikacji komercyjnych, zwłaszcza takich, które zawierają skomplikowane struktury widgetów, różnica ta mogłaby być jeszcze mniej zauważalna, ale w celu potwierdzenia tego, potrzebne by były dodatkowe badania.

Ważnym do dodania jest także fakt, że o ile procentowo różnice pomiędzy czasami na różnych platformach są znaczące, to realne wartości dla przeciętnego użytkownika mogą być niezauważalne lub pomijane w odbiorze responsywności danej strony, czy też aplikacji.

Literatura

[1] A. M. Aladwani, IT project uncertainty, planning and success: An empirical investigation from Kuwait, Information Technology & People 15 (3) (2002) 210-226, https://doi.org/10.1108/09593840210444755.

[2] N. Shevtsiv, A. Striuk, Cross platform development vs native development, CEUR Workshop Proceedings 2832 (2021) 75-83, https://doi.org/10.31812/123456789%2F4428.

[3] L. Dagne, Flutter for cross-platform App and SDK development, Bachelor Thesis at Metropolia University of Applied Sciences, Helsinki, 2019, https://www.theseus.fi/bitstream/handle/10024/172866/Lukas%20Dagne%20Thesis.pdf.

[4] T. Tran, Flutter Native Performance and Expressive UI/UX, Bachelor Thesis at Metropolia University of Applied Sciences, Helsinki, 2022, https://www.theseus.fi/bitstream/handle/10024/336980/Thanh_Tran.pdf?sequence=2.

[5] J. Jagiełło, Performance comparison between react native and flutter, Bachelor Thesis at Umeå University, Umeå, 2019, https://www.diva-portal.org/smash/get/diva2:1349917/FULLTEXT01.pdf.

[6] A. Charland, B. Leroux, Mobile application development: web vs. native, Communications of the ACM, 54 (5) (2011) 49-53, https://doi.org/10.1145/1941487.1941504.

[7] Y. Ma, X. Liu, Y. Liu, Y. Liu, G. Huang, A tale of two fashions: An empirical study on the performance of native apps and web apps on android, IEEE Transactions on Mobile Computing 17 (5) (2018) 990-1003, https://doi.org/10.1109/TMC.2017.2756633.

[8] M. Olsson, A Comparison of Performance and Looks Between Flutter and Native Applications: When to prefer Flutter over native in mobile application development, Bachelor Thesis at Blekinge Institute of Technology, Karlskrona, 2020, https://www.diva-portal.org/smash/get/diva2:1442804/FULLTEXT01.pdf.

[9] Y. Rasmusson-Wright, S. Hedlund, Cross-platform Frameworks Comparison: Android Applications in a Cross-platform Environment, Xamarin Vs Flutter, Bachelor Thesis at Blekinge Institute of Technology, Karlskrona, 2021, https://www.diva-portal.org/smash/get/diva2:1568490/FULLTEXT01.pdf.

[10] S. Zindl, Flutter on Windows Desktop: a use case based study, Bachelor Thesis at Universität Stuttgart, Stuttgart, 2021, http://dx.doi.org/10.18419/opus-11723.

[11] Oficjalna dokumentacja Flutter - najlepsze praktyki, https://docs.flutter.dev/perf/best-practices, [19.05.2023].

[12] Gskinner, https://blog.gskinner.com/archives/2022/09/flutter-rendering-optimization-tips.html, [19.05.2023].

[13] Strona z wytycznymi Flutter cookbook, https://docs.flutter.dev/cookbook/lists/long-list, [19.05.2023].