WYŻSZA SZKOŁA BANKOWA W GDAŃSKU
WYDZIAŁ INFORMATYKI I NOWYCH TECHNOLOGII

Krzysztof Gründemann
nr albumu 63052

Badanie progu wejścia i wydajności aplikacji mobilnych napisanych w technologiach Flutter i React Native

Praca magisterska na kierunku Informatyka

Praca napisana pod kierunkiem
dr Dariusza Kralewskiego

Gdańsk 2023

Spis treści

Spis treści 2

Wstęp 3

1. Wprowadzenie do technologii Flutter i React Native 5

1.1 Aplikacje natywne i wieloplatformowe 5

1.2 Próg wejścia do nauki technologii Flutter i React Native 10

1.3 Język Dart 11

1.4 Opis i historia technologii Flutter 14

1.5 Wprowadzenie do React 16

1.6 Opis i historia technologii React Native 17

2. Porównanie Flutter i React Native 20

2.1 Architektura i funkcjonalność 20

2.2 Narzędzia i środowisko pracy 25

2.3 Społeczność i wsparcie 26

3. Analiza porównawcza wybranych aplikacji mobilnych 30

3.1 Metodologia porównania 30

3.2 Charakterystyka wybranych aplikacji 31

3.3 Porównanie wyników osiąganych przez aplikacje stworzone w technologiach Flutter i React Native 35

Zakończenie 43

Spis wykorzystanych źródeł 44

Spis ilustracji 46

Spis wykresów 47

Załączniki 48

Wstęp

Temat pracy magisterskiej "Badanie progu wejścia i wydajności aplikacji mobilnych napisanych w technologiach Flutter i React Native" jest aktualny i ważny ze względu na rosnące znaczenie aplikacji mobilnych w dzisiejszym świecie. Obie technologie są popularne wśród programistów i coraz częściej wybierane do tworzenia aplikacji mobilnych. Uzasadnieniem podjęcia tego tematu jest chęć przeprowadzenia badań, które pozwolą na porównanie tych dwóch technologii i udzielenie odpowiedzi na pytanie, która z nich jest lepsza pod względem wydajności.

Hipotezą badawczą, która przyświeca pracy jest "Aplikacje stworzone z użyciem technologii Flutter będą posiadały lepsze parametry wydajności w porównaniu do aplikacji napisanych w React Native." W celu potwierdzenia lub obalenia tej hipotezy zostaną przeprowadzone badania porównawcze aplikacji napisanych w obu technologiach.

Pytania badawcze:

- Czy aplikacje stworzone z użyciem technologii Flutter rzeczywiście posiadają lepsze parametry wydajności w porównaniu do aplikacji napisanych w React Native?
- Jaki jest poziom trudności w tworzeniu aplikacji przy użyciu technologii Flutter i React Native?
- Jakie są wady i zalety zastosowania technologii Flutter i React Native?
- Jakie są główne różnice między aplikacjami stworzonymi z użyciem technologii Flutter i React Native pod względem wydajności?

Celem pracy magisterskiej jest porównanie wydajności aplikacji mobilnych napisanych w technologiach Flutter i React Native oraz odpowiedzi na powyższe pytania badawcze. Przeprowadzenie badań porównawczych pomiędzy Flutterem a React Native pozwoli na uzyskanie informacji, które będą przydatne dla programistów i firm zajmujących się tworzeniem aplikacji mobilnych, umożliwiając im wybór najlepszej technologii do ich projektów.

Wprowadzenie do technologii Flutter i React Native: W rozdziale pierwszym zostaną omówione technologie Flutter i React Native. W ramach tego rozdziału zostaną zaprezentowane podstawowe informacje dotyczące technologii wieloplatformowych oraz aplikacji natywnych. Ponadto, przedstawione zostaną kluczowe cechy języka Dart oraz opis i historia technologii Flutter oraz React Native.

Porównanie Flutter i React Native: W rozdziale drugim zostanie przeprowadzone porównanie technologii Flutter i React Native. W ramach tego rozdziału zostaną przedstawione cechy architektury i funkcjonalności obu technologii, narzędzia i środowisko pracy oraz społeczność i wsparcie. Porównanie tych elementów pozwoli na lepsze zrozumienie i porównanie obu technologii.

Analiza porównawcza wybranych aplikacji mobilnych: W rozdziale trzecim zostanie przeprowadzona analiza porównawcza wybranych aplikacji mobilnych stworzonych przy użyciu technologii Flutter i React Native. W ramach tego rozdziału zostanie przedstawiona metodologia porównania, a następnie przedstawione zostaną aplikacje mobilne napisane w obu technologiach. Na końcu zostaną zaprezentowane wyniki analizy porównawczej.

Zakończenie: W zakończeniu zostanie podsumowane porównanie technologii Flutter i React Native. W tym rozdziale zostaną podsumowane wyniki analizy porównawczej oraz przedstawione wnioski z badania. Ponadto, zostanie poruszona kwestia przyszłości i rozwoju tych technologii.

Spis wykorzystanych źródeł, spis ilustracji, spis wykresów i załączniki: Wszystkie te rozdziały zawierają dodatkowe informacje, które są potrzebne do zrozumienia pracy. W spisie wykorzystanych źródeł znajdują się informacje o źródłach, które były wykorzystane do pisania pracy. Spis ilustracji i wykresów zawierają numery stron, na których znajdują się ilustracje i wykresy, które są użyte w pracy. Załączniki zawierają kod aplikacji oraz wyniki testów.

1. Wprowadzenie do technologii Flutter i React Native

W dzisiejszych czasach aplikacje mobilne są coraz bardziej popularne i stanowią ważny element w życiu wielu ludzi. Jednym z najważniejszych wyborów, które muszą podjąć programiści tworzący aplikacje mobilne, jest wybór odpowiedniej technologii. W związku z tym, istnieje wiele narzędzi i frameworków, które pozwalają na łatwiejsze i szybsze tworzenie aplikacji mobilnych. W niniejszej pracy zostaną porównane dwie popularne technologie: Flutter i React Native, które pozwalają na tworzenie aplikacji wieloplatformowych. Rozdział pierwszy poświęcony jest wprowadzeniu do tych technologii, zwracając uwagę na kwestie związane z programowaniem aplikacji mobilnych. W podrozdziale 1.1 zostanie przedstawiona charakterystyka aplikacji natywnych i wieloplatformowych, co pozwoli czytelnikowi na zrozumienie różnic między nimi i dlaczego ważny jest wybór odpowiedniej technologii do tworzenia aplikacji mobilnych.

1.1 Aplikacje natywne i wieloplatformowe

W ostatnich latach rozwój technologii mobilnych przyspieszył w znacznym stopniu, a zarazem zwiększyła się potrzeba tworzenia aplikacji o wysokiej jakości. Istnieją dwa główne sposoby tworzenia aplikacji mobilnych: natywny i wieloplatformowy. Wybór między natywnym a wieloplatformowym sposobem jest jednym z najważniejszych decyzji w każdym projekcie aplikacji mobilnej. Ta jedna decyzja ma ogromne implikacje dla projektu aplikacji, technologii używanych do jej stworzenia i ostatecznie, dla użytkowników, którzy mogą do niej uzyskać dostęp.

Aplikacje mobilne natywne to aplikacje przeznaczone na system Android lub iOS. W zależności od systemu operacyjnego, na który tworzy się aplikacje, kod jest pisany w określonym języku programowania. Poniżej zostały wypisane technologie, które pozwalają napisanie aplikacji natywnej w zależności od systemu.

Aplikacje natywne na Androida są tworzone przy użyciu języków programowania Java lub Kotlin. Java był pierwotnie używany do tworzenia aplikacji na Androida. Google

wprowadził wsparcie dla Kotlin na Androida w 2017 roku.[1] Kotlin obsługuje programowanie obiektowe i funkcyjne, podczas gdy Java jest ograniczone do programowania obiektowego.

Aplikacje natywne dla iOS, korzystają z języków programowania Objective-C lub Swift. Objective-C jest nadzbiorem języka programowania C. Jest to pierwotnie język używany do tworzenia oprogramowania działającego na iOS. Apple wprowadził Swift w 2014 roku podczas swojego World Wide Developer Conference.[2] To potężny ogólnego przeznaczenia, język programowania wysokiego poziomu dla ekosystemu Apple. Apple twierdzi, że Swift jest 2,6 razy szybszy niż Objective-C[3], a składnia jest łatwiejsza do nauki.

Korzyści płynące z natywnych aplikacji mobilnych obejmują wiele aspektów, takich jak:

- Lepsza wydajność: najważniejszą zaletą aplikacji natywnych jest ich lepsza wydajność w porównaniu do rozwiązań wieloplatformowych lub hybrydowych. Przede wszystkim dlatego, że aplikacja bezpośrednio współdziała z natywnymi interfejsami API i nie zależy od oprogramowania pośredniego. Dzięki temu, dzięki pełnemu wsparciu sprzętowemu i systemowemu, aplikacje natywne są bardziej responsywne i wydajne. Ponadto, urządzenia, na których są one ładowane, przechowują wszystkie kluczowe dane, co przyspiesza ładowanie treści.Bezpieczeństwo: natywne aplikacje mogą poprawić bezpieczeństwo danych użytkownika. Mają dostęp do platformowych funkcji bezpieczeństwa.
- Udoskonalona jakość interakcji z użytkownikiem: każda platforma ma swoje własne wytyczne dotyczące interfejsu użytkownika (UI/UX), których programiści powinni przestrzegać. Dlatego, tworząc natywną aplikację mobilną, te standardy są dobrze przestrzegane, co prowadzi do spójnego wyglądu i odczucia zgodnego z systemem operacyjnym. Spójność natywnych aplikacji mobilnych zapewnia również bardziej

[1] (P. Dedio 2019) - https://android.com.pl/programowanie/182482-poczatki-programowania-java-kotlin/ [dostęp 20.02.2023]

[2] (Apple 2014) - https://developer.apple.com/swift/blog/?id=14 [dostęp 20.02.2023]

[3] (J.Wasiak 2022) - https://teamquest.pl/blog/2442_objective-c-czym-jest [dostęp 20.02.2023]

intuicyjną i interaktywną jakość interakcji z użytkownikiem, ponieważ ludzie są zaznajomieni z układami typowymi dla swojego systemu operacyjnego. Dlatego, gdy korzystają z Twojej aplikacji, czują się jakby korzystali z czegoś, co znają.

- Lepsze zabezpieczenia: kolejnym powodem, aby wybrać rozwój aplikacji natywnych, jest wyższy poziom bezpieczeństwa, jakie oferują. Wraz ze wzrostem obaw związanych z ochroną danych, rozwiązania oprogramowania we wszystkich branżach powinny sprawiać, że użytkownicy czują się bezpiecznie, udostępniając informacje cyfrowe.[4]

Niemniej jednak, natywne aplikacje mobilne mają też kilka wad, takich jak:

- Kosztowny rozwój: Programowanie używane w aplikacjach natywnych jest dość skomplikowane. W związku z tym, rozwój aplikacji natywnych wymaga większej pracy, co zwiększa czas i koszt rozwoju. Ponadto, z różnymi kodami dla różnych platform, rozwijanie aplikacji natywnych wymaga jeszcze więcej czasu i pieniędzy. Im więcej kosztuje rozwój aplikacji natywnej, tym wyższe będą koszty utrzymania.

- Brak elastyczności: Deweloperzy nie mają elastyczności co do wyboru platformy do tworzenia aplikacji natywnych. Deweloperzy muszą pisać kod oddzielnie dla jednej platformy w danym momencie, oddzielnie dla Androida i iOS. W większości przypadków, jeśli zatrudniasz deweloperów do swojego pomysłu na aplikację natywną, musisz zatrudnić dwie zespoły deweloperów - jeden dla Androida i jeden dla iOS.[5]

- Czasochłonny rozwój: Każda platforma, takie jak iOS i Android, wymaga oddzielnych kodów, co oznacza, że potrzeba więcej czasu, ponieważ odpowiada to pisaniu kodu dla dwóch różnych aplikacji.

- Brak możliwości powtórnego użycia kodu: Jest konieczne tworzenie i utrzymanie kodu w osobnych projektach dla każdego systemu operacyjnego.

---

[4](M. Carrington 2022) - https://www.velvetech.com/blog/native-mobile-app-development/ [dostęp 20.02.2023]

[5] (U. Khan 2022) https://clutch.co/app-developers/resources/pros-cons-native-apps [dostęp 20.02.2023]

Aplikacje mobilne wieloplatformowe (cross-platform) napisane są za pomocą jednego kodu źródłowego. Tworzy się je za pomocą cross-platformowych frameworków, które korzystają z zestawów SDK (Software Development Kit) dla danego systemu (zestawy SDK systemu Android i zestawy SDK systemu iOS) z ujednoliconego interfejsu API (application programming interface).

Przykłady popularnych frameworków wieloplatformowych obejmują:

- React Native od Meta. Wykorzystuje JavaScript jako język programowania,
- Flutter od Google. Wykorzystuje Dart jako język programowania,
- Xamarin firmy Microsoft. Wykorzystuje C# i XAML jako język programowania.

Aby zapewnić użytkownikom jak najlepsze wrażenia, programiści wieloplatformowych aplikacji mobilnych często korzystają z natywnych elementów interfejsu użytkownika, które są specyficzne dla każdego systemu operacyjnego. Dzięki temu aplikacja działa na różnych platformach, a jednocześnie zachowuje spójność i wygląd podobny do aplikacji natywnych, co może poprawić doświadczenie użytkownika i zwiększyć zaangażowanie.

Znane przykłady wieloplatformowych aplikacji mobilnych to:

- Instagram, Skype, Walmart, Uber Eats i Airbnb (React Native),[6]
- Google Ads, Groupon i eBay (Flutter).[7]

Tworzenie aplikacji mobilnych dla wielu platform za pomocą wspólnego kodu źródłowego może przynieść następujące korzyści:

[6] (A. Cińcio 2022) - https://nofluffjobs.com/blog/react-native-developer-czym-sie-zajmuje-i-jak-nim-zostac/ [dostęp 20.02.2023]

[7] (P. Chmielewska, A. Trachim, G. Smith 2022) - https://itcraftapps.com/blog/19-apps-built-with-flutter-framework/ [dostęp 20.02.2023]

- Oszczędność czasu i kosztów: tworzenie jednej wersji kodu źródłowego dla wielu platform zamiast tworzenia osobnych wersji dla każdej z nich może znacznie skrócić czas i obniżyć koszty.
- Usprawnienie procesu testowania: testowanie jednej wersji kodu źródłowego dla wszystkich platform jest prostsze niż testowanie kilku osobnych wersji.
- Łatwiejsza aktualizacja i utrzymanie: aktualizowanie jednej wersji kodu źródłowego dla wszystkich platform jest prostsze niż aktualizowanie kilku osobnych wersji.

Tworzenie aplikacji mobilnych na wiele platform rozwiązuje pewne problemy, z jakimi zmagają się twórcy aplikacji natywnych. Pomimo to, taki sposób tworzenia aplikacji niesie ze sobą swoje własne wady:

- Większy rozmiar pliku: Aplikacje cross-platform zwykle są większe niż aplikacje natywne. Powodem jest to, że aplikacje cross-platform muszą działać na wielu różnych urządzeniach i wymagają dodatkowych, abstrakcyjnych procesów działania, co jest wynikiem konieczności zastosowania bibliotek/fragmentów kodu niezbędnych do realizacji określonych funkcjonalności. Choć różnice te nie są zauważalne obecnie, to jednak stanowią wadę takiego rozwiązania.
- Mniejsza wydajność i niezawodność: Aplikacje cross-platform mają gorszą wydajność od natywnych, głównie ze względu na dodatkową warstwę obliczeniową związaną z koniecznością komunikacji z platformowymi usługami. Cross-platformowe frameworki często wykorzystują własny runtime, który odpowiada za komunikację z usługami platformowymi, co zwiększa obciążenie procesora i skutkuje niższą wydajnością. Dodatkowo, cross-platformowe frameworki nie zawsze oferują pełne wsparcie dla wszystkich funkcji dostępnych na danej platformie, co może również prowadzić do niższej wydajności.
- Możliwe niezgodności projektowe na dwóch platformach: Kolejnym problemem może być to, że istnieją pewne różnice w interfejsach użytkownika i doświadczeniu użytkownika na różnych platformach. Może to wynikać zarówno z różnych wymagań

dla projektowania interfejsów użytkownika na różnych platformach, jak i z konkretnych ograniczeń tych platform.[8]

Konkludując, wybór między natywnym i wieloplatformowym podejściem do tworzenia aplikacji mobilnych zależy od wielu czynników, takich jak potrzeby biznesowe, budżet, harmonogram i cel projektu. Dla projektów wymagających maksymalnej wydajności i bezpieczeństwa aplikacji, natywne aplikacje są najlepszym rozwiązaniem. Dla projektów, które wymagają szybkiego dostarczenia aplikacji na różne platformy, aplikacje wieloplatformowe mogą być lepszym rozwiązaniem ze względu na korzyści płynące z dzielenia kodu i czasu produkcji.

1.2 Próg wejścia do nauki technologii Flutter i React Native

Wybór technologii do tworzenia aplikacji mobilnych nie jest łatwym zadaniem, zwłaszcza dla początkujących programistów, którzy dopiero zaczynają swoją przygodę z programowaniem aplikacji mobilnych. Dlatego ważne jest, aby przed podjęciem decyzji o wyborze odpowiedniej technologii, znać progi wejścia do nauki danej technologii. Warto podkreślić, że wybór odpowiedniej technologii do tworzenia aplikacji mobilnych jest kluczowy dla sukcesu projektu, dlatego ważne jest, aby poznać progi wejścia do nauki danej technologii.

W przypadku technologii Flutter i React Native, prog wejścia jest różny, ponieważ wykorzystują one różne języki programowania i środowiska. Flutter wykorzystuje język programowania Dart, który został stworzony przez Google, a React Native wykorzystuje język JavaScript. Z tego powodu, aby nauczyć się korzystać z tych technologii, warto najpierw poznać podstawy tych języków programowania.

Początkujący programiści powinni rozpocząć naukę technologii Flutter od podstaw, czyli od nauki języka Dart. Na początku warto zapoznać się z podstawami języka, takimi jak składnia, zmienne, typy danych oraz struktury kontrolne. Następnie można przejść do nauki widgetów, które stanowią podstawę interfejsu użytkownika w technologii Flutter.

---

[8](A. Safonov) 2023 - https://merehead.com/blog/cross-platform-native-mobile-app-development/ [dostęp 20.02.2023]

W przypadku React Native, początkujący programiści powinni najpierw poznać język JavaScript oraz bibliotekę React, na której opiera się ta technologia. JavaScript jest językiem popularnym i stosowanym w wielu projektach internetowych, więc warto zacząć naukę od podstaw języka i zrozumienia jego działania. Następnie można przejść do nauki biblioteki React, która umożliwia tworzenie interfejsu użytkownika w React Native.

Podsumowując, prog wejścia do nauki technologii Flutter i React Native jest różny i zależy od znajomości języków programowania i bibliotek, na których opierają się te technologie. Dlatego warto zacząć naukę od podstaw, aby lepiej zrozumieć działanie tych technologii i móc efektywnie tworzyć aplikacje mobilne.

1.3 Język Dart

Aby efektywnie pracować z frameworkiem Flutter, programiści powinni zrozumieć język Dart. Ważne jest, aby znać historię tego języka, sposób działania społeczności programistów, jego atuty oraz powód, dla którego został wybrany jako język do tworzenia aplikacji w Flutterze.

Dart to język programowania stworzony przez Google. Został zaprezentowany po raz pierwszy w 2011 roku i od tego czasu nieustannie się rozwija. W 2013 roku ukazała się jego stabilna wersja, a w 2018 roku została wprowadzona znacząca aktualizacja w postaci Dart 2.0.[9] Początkowo, Dart został stworzony z myślą o tworzeniu stron internetowych i miał zostać alternatywą dla JavaScript. Obecnie jego zastosowanie skupia się głównie na rozwoju aplikacji mobilnych oraz na użyciu w frameworku Flutter.

Dart posiada wiele przydatnych funkcji takich jak: obsługa asynchroniczności z klasami takimi jak Future[10], generyki, wyrażenia regularne czy wsparcie dla programowania

[9] (D. Bolton 2019) - https://www.dice.com/career-advice/fall-rise-dart-google-javascript-killer [dostęp 20.02.2023]

[10](K. Walrath 2019) - https://medium.com/dartlang/dart-asynchronous-programming-futures-96937f831137 [dostęp 20.02.2023]

funkcyjnego. Posiada także bibliotekę standardową, która zawiera wiele przydatnych narzędzi i funkcji, takich jak kolekcje, modele danych, parsowanie JSON czy obsługę sieci.

Kolejną ważną zaletą języka Dart jest jego skalowalność. Może on być używany zarówno do tworzenia prostych aplikacji jednoekranowych, jak i skomplikowanych projektów z wieloma ekranami i funkcjonalnościami. Dzięki wykorzystaniu biblioteki AngularDart, kod może być łatwo podzielony na moduły, co ułatwia jego utrzymanie i rozwój.

Oprócz aplikacji webowych i mobilnych, Dart może być również używany do tworzenia aplikacji na inne platformy, takie jak np. smart TV[11] czy desktopowe aplikacje. Dzięki wykorzystaniu Fluttera, kod napisany w Dart jest kompilowany do natywnego kodu, co pozwala na optymalizację wydajności aplikacji.

Dart posiada zarówno możliwość kompilacji AOT (Ahead-of-Time) jak i JIT (Just-In-Time). AOT kompilacja pozwala na przyspieszenie uruchamiania aplikacji na urządzeniach końcowych, natomiast JIT kompilacja pozwala na dynamiczne generowanie kodu podczas działania aplikacji, co pozwala na lepszą optymalizację i szybsze dostosowywanie się do różnych warunków wykonania. Jest również wyposażony w Garbage Collection, co pozwala na automatyczne zarządzanie pamięcią, co zmniejsza ilość kodu potrzebnego do napisania aplikacji. Kolejnym ważnym narzędziem jest Observatory, który służy do debugowania i profilowania aplikacji. Observatory umożliwia developerom śledzenie działania aplikacji w czasie rzeczywistym, zbieranie informacji o wykorzystaniu pamięci i CPU, a także identyfikowanie problemów z wydajnością. Narzędzie pozwala również na zdalne debugowanie aplikacji z poziomu przeglądarki. Observatory umożliwia również dostęp do informacji o stanie aplikacji oraz umożliwia wykonywanie operacji na rzecz aplikacji bezpośrednio z poziomu narzędzia.

Dodatkowo Dart posiada wsparcie dla programowania obiektowego, co pozwala na łatwiejsze zarządzanie kodem oraz rozwój aplikacji. Jedną z głównych zalet języka Dart jest jego czytelność oraz prostota. Początkowo celem Darta miało być zastąpienie JavaScriptu.

[11] (A. Denisov 2022) - https://medium.com/flutter-community/flutter-for-apple-tv-756fcd5e8113 [dostęp 20.02.2023]

Natomiast teraz Dart stał się głównie narzędziem, które wykorzystuje się w aplikacjach mobilnych.

SDK Darta (Software Development Kit) zawiera kompleksowy zestaw narzędzi, które ułatwiają pracę developerów podczas tworzenia aplikacji z wykorzystaniem frameworka Flutter. Zestaw ten obejmuje m.in.:

- narzędzia do statycznej analizy kodu za pomocą wbudowanego analizatora,
- kompilatory Dart JIT i AOT,
- narzędzia do profilowania, debugowania i logowania za pomocą Dart DevTools oraz Observatory.[12]

W trakcie pisania lub debugowania kodu aplikacji pisanej za pomocą Fluttera używa się domyślnej metody kompilacji JIT. Przy tej metodzie kod Dart jest kompilowany do kodu maszynowego w czasie uruchamiania aplikacji. Jest to bardziej elastyczne rozwiązanie, ponieważ pozwala na dynamiczne zmiany kodu w trakcie działania aplikacji, ale może powodować pewne opóźnienia w działaniu aplikacji. Dlatego przy budowie finalnej wersji kod kompiluje się za pomocą AOT. W tej metodzie kod Dart jest kompilowany do kodu maszynowego przed uruchomieniem aplikacji. To rozwiązanie pozwala na szybsze działanie aplikacji, ponieważ nie ma potrzeby kompilowania kodu w trakcie działania aplikacji. Jednak nie pozwala na dynamiczne zmiany kodu.

Podsumowując, język Dart to wydajny i skalowalny język programowania, który został stworzony z myślą o tworzeniu aplikacji mobilnych i webowych. Posiada wiele zalet, takich jak obsługa asynchroniczności, wsparcie dla programowania funkcyjnego oraz bibliotekę standardową z wieloma przydatnymi narzędziami. Jego czytelność i prostota sprawiają, że jest on łatwy w użyciu dla programistów, a narzędzia dostępne w SDK ułatwiają pracę i rozwój aplikacji. Znajomość języka Dart jest niezbędna dla programistów pracujących z frameworkiem Flutter i pozwala na tworzenie wydajnych i nowoczesnych aplikacji mobilnych.

[12] A. Biessek, Flutter i Dart 2 dla początkujących, Helion S.A., Gliwice 2021, s. 29

1.4 Opis i historia technologii Flutter

Flutter to framework o otwartym kodzie źródłowym który pozwala na tworzenie aplikacji mobilnych, desktopowych i webowych. Został stworzony przez Google i pierwszy raz zapowiedziany w 2015 roku podczas konferencji Dart Developer Summit. Od tego czasu jest ciągle rozwijany. Pierwsze wydanie Fluttera miało kodową nazwę „Sky" i było kompatybilne tylko z systemem operacyjnym Android. Został przedstawiony jako ulepszona wersja kilku poprzednich eksperymentów przeprowadzonych przez Google, których celem było stworzenie czegoś lepszego dla telefonów komórkowych, jeśli chodzi o komfort i rozwój dla użytkownika, z naciskiem na renderowanie z wysoką wydajnością.[13]

Flutter jest oparty na języku programowania Dart, który również został stworzony przez Google. Jego architektura opiera się w głównej mierze na użyciu widżetów, które pozwalają na zbudowanie całego interfejsu użytkownika za pomocą tzw. drzewka widżetów. Aplikacja jest głównym widżetem, a poszczególne ekrany są pierwszymi rozgałęzieniami. Framework umożliwia tworzenie własnych widżetów lub na korzystanie z gotowych widżetów, które wyglądają tak samo jak ich natywne odpowiednik.

Flutter oparty jest na silniku renderowania Skia, który jest biblioteką renderującą grafikę 2D.[14] Flutter używa Skia do rysowania interfejsów użytkownika, a także do wykonywania animacji i efektów wizualnych. Silnik renderowania Skia został wybrany przez zespół Flutter z kilku powodów. Po pierwsze, Skia jest bardzo szybki i wydajny, co jest kluczowe dla interfejsów użytkownika, które muszą działać płynnie, bez względu na to, jak skomplikowane są. Po drugie, Skia jest bardzo elastyczny i łatwo dostosowuje się do różnych platform i urządzeń.

Od momentu powstania, framework Flutter miał na celu zapewnić użytkownikom lepsze doświadczenie poprzez wydajne działanie aplikacji. Jednak to nie jedyny cel tego narzędzia. Również skupiono się na rozwiązaniu pewnych problemów związanych z

[13] A. Biessek, Flutter i Dart 2 dla początkujących, Helion S.A., Gliwice 2021, s. 96

[14] (Google 2023) - https://skia.org/ [dostęp 20.02.2023]

tworzeniem aplikacji mobilnych na różne platformy. Przykłady problemów, na które odpowiedzią ma być Flutter:

- Hot-reload: platforma umożliwia programistom obserwowanie zastosowanych zmian w czasie rzeczywistym, pomaga to zaoszczędzić czas, co skutkuje większą wydajnością i produktywnością. Hot-reload Fluttera zapewnia przewagę nad podobnymi funkcjami konkurencji, umożliwiając programistom wstrzymanie wykonywania kodu, wprowadzanie zmian w kodzie i kontynuowanie kodowania z tego samego miejsca, co skutkuje dużą szybkością.
- Dostosowywanie interfejsu użytkownika: Flutter oferuje biblioteki i narzędzia do tworzenia skórek interfejsu użytkownika, co pozwala na łatwe dostosowywanie wyglądu aplikacji do różnych urządzeń i rozdzielczości.
- Tworzenie aplikacji animowanych: Flutter oferuje biblioteki i narzędzia do tworzenia animacji, co pozwala na łatwe tworzenie atrakcyjnie wyglądających aplikacji.

Niemniej jednak, korzystanie z Fluttera może wiązać się z pewnymi wadami, takimi jak:

- Ograniczone wsparcie dla bibliotek zewnętrznych: Chociaż Flutter ma rosnącą liczbę dostępnych pakietów, może brakować wsparcia dla niektórych ważnych bibliotek, które są dostępne w innych frameworkach.
- Złożoność: Architektura i drzewo widżetów Flutter może sprawić, że będzie trudniejsze dla programistów pracować z nim, zwłaszcza dla tych, którzy nie są zaznajomieni z tym frameworkiem.
- Wielkość aplikacji: Wielkość aplikacji opracowywanych za pomocą Flutter może być większa niż aplikacji natywnej, co może być problemem dla użytkowników z ograniczoną przestrzenią na przechowywanie.
- Ograniczone wsparcie dla starszych urządzeń: Chociaż Flutter jest zaprojektowany tak, aby działał zarówno na nowszych, jak i starszych urządzeniach, może działać gorzej na starszych urządzeniach z mniej wydajnym sprzętem.

- Wymaga znajomości języka Dart – Choć Flutter jest stosunkowo prosty, wymaga od programistów najpierw nauczenia się Darta. Wymagając dodatkowego etapu nauki, Flutter może zwiększyć czas i koszt jakiegokolwiek projektu.

Konkludując, Flutter to popularny framework pozwalający na tworzenie wydajnych i atrakcyjnych aplikacji mobilnych, desktopowych i webowych, który ma wiele zalet, ale może także stanowić wyzwanie dla mniej doświadczonych programistów ze względu na swoją specyficzną architekturę.

1.5 Wprowadzenie do React

React to biblioteka JavaScript stworzona przez Facebook, która umożliwia tworzenie interaktywnych aplikacji webowych. Na konferencji JSConf w 2013 roku, zapowiedziano, że jest biblioteką typu open source, co oznacza, że udostępnioną ją dla każdego i może być swobodnie modyfikowana. Dzięki temu React dołączył do grona innych popularnych bibliotek do tworzenia interfejsu użytkownika, takich jak jQuery, Angular, Dojo czy Meteor.[15]

W React, składniki są podstawowymi jednostkami tworzenia interfejsu użytkownika. Każdy składnik jest odpowiedzialny za renderowanie określonej części interfejsu, co umożliwia łatwą i elastyczną implementację zmian. Dzięki temu można łatwo testować, debugować i ulepszać każdy składnik niezależnie od innych, co prowadzi do bardziej modularnego i elastycznego projektu.

React korzysta z wirtualnego DOM (Document Object Model), co pozwala na optymalizację renderowania. Virtual DOM to reprezentacja pamięciowa DOM-u, która jest używana przez React do określenia, które elementy interfejsu trzeba zaktualizować. Dzięki temu React może zwiększyć wydajność aplikacji, ponieważ nie trzeba od razu aktualizować całego DOM-u po każdej zmianie.[16]

---

[15] A. Banks E. Porcello. React od podstaw. Nowoczesne wzorce tworzenia aplikacji. Gliwice: Helion S.A., 2021. s.14

[16] S. Stoyan. React w działaniu. Tworzenie aplikacji internetowych. Gliwice: HELION S.A., 2021. s.79

Aplikacje w React tworzone są za pomocą JavaScript oraz rozszerzenia do JS jakim jest JSX. Dzięki JSX, programiści mogą pisać kod HYML wprost w kodzie JS, a React automatycznie konwertuje ten kod na odpowiednie elementy natywne dla platformy, co pozwala na łatwe i naturalne tworzenie interfejsu użytkownika. JSX jest bardzo przyjazny dla programistów, którzy mają doświadczenie w tworzeniu aplikacji HTML/CSS/JavaScript, a także umożliwia przejrzyste tworzenie składników interfejsu użytkownika w React.

1.6 Opis i historia technologii React Native

React Native to framework o otwartym kodzie źródłowym stworzony przez Facebooka, który pozwala na pisanie aplikacji mobilnych renderujących się natywnie dla systemów iOS i Android przy użyciu języka JavaScript i biblioteki React.[17]

Rozwój z JavaScript zapewnia możliwość dzielenia się kodem źródłowym z aplikacjami webowymi React. W rezultacie ci sami programiści mogą pracować zarówno nad aplikacjami webowymi, jak i mobilnymi, ponieważ technologie są bardzo podobne. Takie rozwiązanie jeszcze nie jest idealnie stabilne, ale możliwość dzielenia się kodem niezależnym od interfejsu użytkownika jest nadal korzystne. Skraca to nie tylko czas rozwoju, ale może również poprawić spójność logiki biznesowej aplikacji między wszystkimi obsługiwanymi platformami.

Framework ten oparty jest na koncepcji "learn once, write anywhere", co oznacza, że po opanowaniu technologii React, programiści mogą tworzyć aplikacje dla różnych platform bez konieczności opanowywania nowych języków i narzędzi. React Native pozwala na tworzenie interfejsu użytkownika za pomocą komponentów, które są renderowane natywnie na urządzeniu końcowym, co zapewnia wysoką wydajność i płynność działania aplikacji.[18]

React Native udostępnia także narzędzia do debugowania i testowania aplikacji, co ułatwia proces tworzenia oprogramowania. Framework ten jest również wspierany przez

[17] B. Eisenman. React Native. Tworzenie aplikacji mobilnych w języku JavaScript. Wydanie II. Gliwice: Helion S.A., 2018. s.13

[18] (J. Budny 2019) https://www.netguru.com/blog/react-native-pros-and-cons [dostęp 20.02.2023]

społeczność, która dostarcza wiele gotowych rozwiązań i bibliotek, co ułatwia tworzenie aplikacji.

Zalety korzystania z React-Native:

- Używa JavaScript: React Native używa popularnego języka programowania JavaScript, znanego wielu programistom. Niektórzy programiści używają nawet TypeScript, który jest podzbiorem JavaScript.
- Duża społeczność: Tworzenie aplikacji z React Native daje dostęp do dużej społeczności. Wśród wielu programistów, którzy korzystają z tego frameworku, jeśli ktoś napotka trudność, istnieje duża szansa, że znajdzie kogoś, kto miał podobny problem i znalazł rozwiązanie.
- Szybsza nauka dla programistów React: Większość programistów z doświadczeniem w React nie powinna mieć trudności w tworzeniu aplikacji React Native, i odwrotnie. Wynika to z faktu, że wiele pomysłów i modułów w obu systemach pokrywa się. W związku z tym programiści, którzy są zaznajomieni z jednym z tych dwóch środowisk, będą potrzebowali mniej czasu na naukę, gdy podejdą do drugiego.

Oprócz wielu zalet korzystanie z React Native łączy się też z kilkoma wadami, m.in.:

- Problemy z aktualizacją: Każda aktualizacja React Native wprowadza dalsze ulepszenia, więc zalecane jest korzystanie z zaktualizowanej wersji. Jednak nie zawsze można całkowicie polegać na automatycznej aktualizacji: mozna napotkać nieoczekiwane problemy przy dostosowywaniu do nowej wersji.
- Ograniczenia w zakresie narzędzi i bibliotek: React Native ma mniej narzędzi i bibliotek niż natywne rozwiązania dla iOS i Android.
- Ograniczenia w zakresie rozmiarów pliku: aplikacje napisane w React Native są zwykle większe niż te napisane natywnie, co może być problematyczne dla niektórych użytkowników.

Podsumowując, React Native pozwala na tworzenie aplikacji mobilnych, które wyglądają i działają jak aplikacje napisane w językach natywnych dla systemów operacyjnych iOS i Android. React Native umożliwia tworzenie aplikacji z interfejsem użytkownika

renderowanym natywnie, co zapewnia płynność i wydajność działania aplikacji. Dzięki temu, mimo że React Native wykorzystuje język JavaScript i bibliotekę React, aplikacje, które z niego korzystają, mają charakter natywny. Warto jednak pamiętać, że nie wszystkie funkcje systemowe mogą być dostępne przez React Native, dlatego w niektórych przypadkach tworzenie aplikacji natywnych może być konieczne.

2. Porównanie Flutter i React Native

W rozdziale drugim dokładnie omówiono porównanie tych dwóch technologii, w celu zwiększenia wiedzy na temat ich cech architektury, funkcjonalności, narzędzi, środowiska pracy oraz społeczności i wsparcia.

Architektura to kluczowy element w procesie tworzenia aplikacji mobilnych. Oba frameworki opierają się na deklaratywnym podejściu, co oznacza, że programista deklaruje, co aplikacja powinna robić, a nie jak to powinno być zrobione. Funkcjonalności są kolejnym aspektem, który zostanie porównany. Flutter i React Native oferują podobne funkcjonalności, ale z nieznacznymi różnicami. Narzędzia i środowisko pracy to kolejny element, który zostanie omówiony. Flutter i React Native mają różne narzędzia i środowiska pracy.

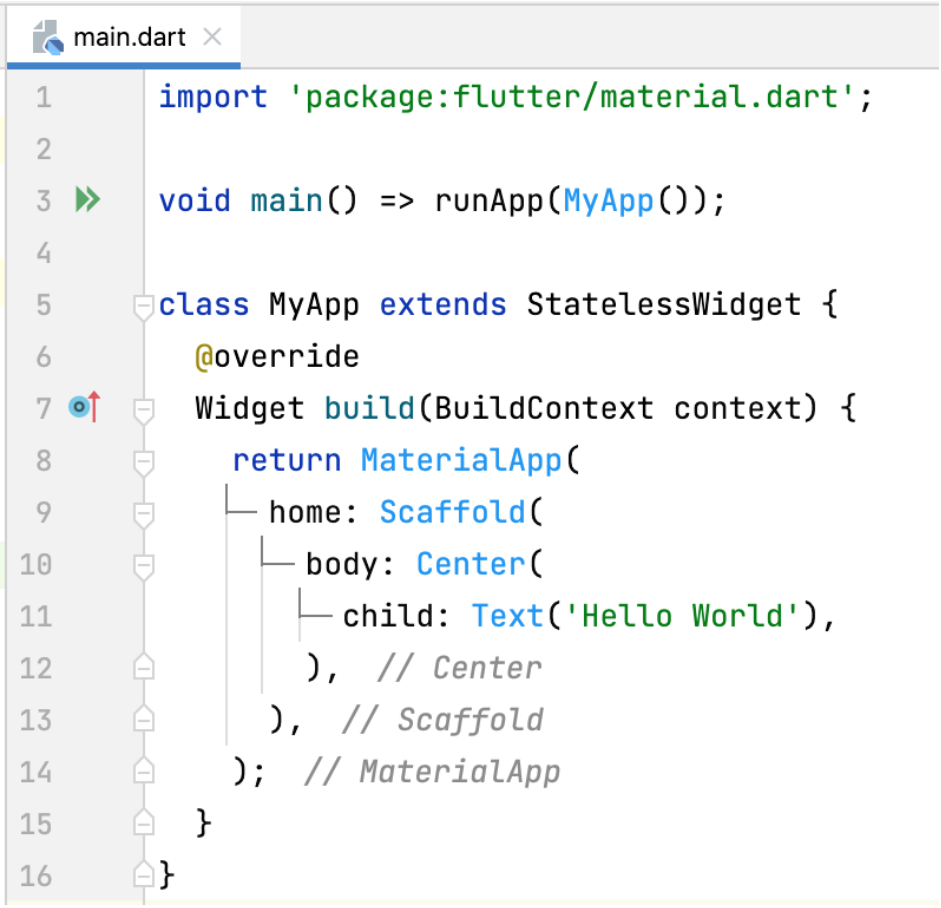
Społeczność i wsparcie to kolejny ważny aspekt w kontekście porównania Flutter i React Native. Flutter jest stosunkowo nową technologią, ale posiada już bardzo silną społeczność deweloperów i wiele narzędzi i bibliotek do tworzenia aplikacji mobilnych. React Native z kolei ma już dłuższą historię i posiada dużą społeczność deweloperów, co oznacza, że istnieje wiele narzędzi, bibliotek i dokumentacji, co ułatwia pracę programistom.

2.1 Architektura i funkcjonalność

Architektura i funkcjonalność to jeden z kluczowych aspektów, które warto porównać między technologiami Flutter i React Native. Oba frameworki wykorzystują podobne podejście do tworzenia aplikacji mobilnych, ale różnią się w kilku kwestiach.

Flutter używa pojęcia "widget" do budowy aplikacji. Widgety są elementami, które opisują, jak ma wyglądać aplikacja i jak ma działać. Wszystko w aplikacji jest widgetem, od największych elementów, takich jak ekran, do najmniejszych, jak tekst. Poniżej przedstawiono dwie proste aplikacje, których jedynym zadaniem było wyświetlenie napisu "Hello World" na środku ekranu.

Ilustracja 1 Kod prostej aplikacji Flutter, która wyświetli napis „Hello World” na ekranie[19]



Na ilustracji 1 przedstawiono prosty kod aplikacji napisanej we Flutterze, która wyświetli na środku ekranu napis „Hello World”. Poniżej opisano poszczególne elementy kodu:

- import 'package: flutter/material.dart'; - linia ta importuje pakiet Flutter Material, który zawiera gotowe komponenty interfejsu użytkownika i style, które można wykorzystać w aplikacji.
- void main () => runApp (MyApp ()); - to jest punkt wejścia programu. main jest funkcją, która jest uruchamiana jako pierwsza i jest główną funkcją aplikacji. runApp jest funkcją, która uruchamia aplikację Flutter i przyjmuje jako argument klasę MyApp, która jest naszą główną klasą aplikacji.

[19] Źródło: Opracowanie własne

- class MyApp extends StatelessWidget - jest to klasa aplikacji, która rozszerza klasę StatelessWidget. Widgety bez stanu (ang. stateless) są widżetami, które nie zmieniają swojego stanu i są renderowane tylko raz.
- @override - to jest dekorator, który oznacza, że ta metoda jest nadpisywana z nadklasy. Widget build (BuildContext context) - jest to metoda, która jest wywoływana, gdy następuje renderowanie widgetu. Zwraca drzewo widgetów, które określają wygląd aplikacji.
- return MaterialApp - ta linia zwraca nową instancję klasy MaterialApp, która jest głównym widgetem aplikacji i jest odpowiedzialna za konfigurację i ustawienie aplikacji.
- title: 'Hello, World!' - jest to właściwość, która określa tytuł aplikacji.
- home: Scaffold - jest to właściwość, która określa domyślny widżet na ekranie. W tym przypadku jest to Scaffold, który jest bazowym widżetem Material Design i zawiera wiele właściwości, takich jak górny pasek, dolny pasek i ciało.
- body: Center- jest to właściwość Scaffold, która określa główną treść aplikacji. W tym przypadku jest to widżet Center, który wyrównuje swoje dziecko w pionie i poziomie na środku ekranu.
- child: Text ('Hello, World!') - jest to właściwość Center, która określa dziecko tego widżetu. W tym przypadku jest to widżet Text, który wyświetla tekst "Hello, World".

React Native natomiast, używa pojęcia "komponentów" do budowy aplikacji. Komponenty opisują, jak ma wyglądać i działać część aplikacji. Komponenty są napisane za pomocą JavaScript i React, a następnie są tłumaczone na kod natywny dla każdej platformy. React Native posiada również bibliotekę gotowych komponentów, które mogą być używane do tworzenia interfejsu użytkownika. Poniżej przedstawiono kod aplikacji:

Ilustracja 2 Kod prostej aplikacji React Native, która wyświetli napis „Hello World” na ekranie[20]

```
JS App.js M ×
HelloWorld > JS App.js > ...
1  import React from "react";
2  import { View, Text } from "react-native";
3
4  const App = () => {
5    return (
6      <View style={{ flex: 1, alignItems: "center", justifyContent: "center" }}>
7        <Text>Hello World</Text>
8      </View>
9    );
10 };
11
12 export default App;
13
```

Ilustracja 2 przedstawia analogiczny kod do ilustracji 1 napisany w React Native. Poniżej opisano poszczególne linijki kodu:

- import React from 'react'; - linia ta importuje bibliotekę React.
- import { View, Text, } from 'react-native'; - linia ta importuje kilka komponentów z biblioteki React Native, które będą wykorzystywane w aplikacji.
- const App = () => - jest to funkcja strzałkowa, która zwraca drzewo komponentów React, które opisują wygląd aplikacji.
- <View style= …> - jest to komponent View, który jest odpowiednikiem Scaffold z Flutter. Jego właściwość style określa styl tego komponentu.
- <Text style= ...>Hello, World</Text> - jest to komponent Text, który wyświetla tekst "Hello, World!". Jego właściwość style określa styl tekstu.

[20] Źródło: Opracowanie własne

- export default App; - linia ta eksportuje komponent App jako główny komponent aplikacji.

Rozmiar pliku aplikacji "Hello World" w najbardziej zoptymalizowanych wersjach dla obu frameworków przedstawia się następująco:

- Flutter: 5,6 MB,
- React Native: 8 MB.[21]

Warto zaznaczyć, że rozmiar pliku aplikacji zależy od wielu czynników, takich jak używane biblioteki, funkcjonalności, konfiguracje i wiele innych. Dlatego też, mimo że te wersje aplikacji "Hello World" dla Flutter i React Native są zoptymalizowane, różnice w ich rozmiarze mogą wynikać z różnych decyzji projektowych lub architektonicznych. Niemniej jednak, obie platformy oferują narzędzia i metody, które pozwalają na redukcję rozmiaru aplikacji oraz optymalizację jej wydajności.

Oprócz architektury, Flutter i React Native oferują wiele podobnych i różniących się funkcjonalności, które mogą mieć wpływ na wybór jednej z technologii do tworzenia aplikacji mobilnych.

W przypadku Flutter, jedną z najważniejszych funkcjonalności jest hot reload, co pozwala na szybkie i łatwe testowanie zmian w kodzie bez potrzeby ponownej kompilacji aplikacji. Hot reload jest funkcjonalnością, która umożliwia programistom na bieżąco wprowadzanie zmian w kodzie źródłowym aplikacji i natychmiastowe sprawdzenie efektów wprowadzonych zmian.[22] Dzięki temu można szybko eksperymentować i poprawiać aplikację, co znacznie skraca czas potrzebny na rozwój aplikacji. W przypadku tworzenia aplikacji z użyciem React Native, istnieje podobna funkcjonalność o nazwie "Fast Refresh", która umożliwia szybkie i płynne testowanie zmian w kodzie źródłowym aplikacji bez potrzeby ponownej kompilacji.

---

[21] (Saigon Technology 2023) - https://saigontechnology.com/blog/flutter-in-comparison-with-react-native-and-xamarin [dostęp 20.02.2023]

[22] (K. Nahotko 2022) - https://boringowl.io/tag/flutter [dostęp 20.02.2023]

Flutter oferuje również wiele gotowych widgetów, takich jak przyciski, pola tekstowe czy listy, co ułatwia tworzenie interfejsów użytkownika. Dzięki Dart, deweloperzy mogą również korzystać z typów statycznych i asynchronicznych funkcji, co może przyspieszyć rozwój aplikacji.

Z drugiej strony, React Native oferuje wiele narzędzi i bibliotek, które pozwalają na łatwe tworzenie aplikacji mobilnych, takich jak Redux do zarządzania stanem aplikacji czy React Navigation do obsługi nawigacji między ekranami. React Native również umożliwia korzystanie z wielu gotowych komponentów, co ułatwia tworzenie interfejsów użytkownika.

Podsumowując, zarówno Flutter, jak i React Native oferują podobne podejście do tworzenia aplikacji mobilnych, ale różnią się w kilku kwestiach, m.in. w sposobie budowania aplikacji, gdzie Flutter wykorzystuje pojęcie "widget", a React Native "komponenty". Flutter i React Native oferują również różne biblioteki gotowych elementów do tworzenia interfejsu użytkownika. W każdym przypadku można wykorzystać wiele gotowych rozwiązań, ale także napisać własne komponenty/widgety dostosowane do specyfikacji projektu.

2.2 Narzędzia i środowisko pracy

Przy wyborze technologii do tworzenia aplikacji mobilnych, istotnym czynnikiem jest również dostępność odpowiednich narzędzi oraz wygodne środowisko pracy.

Flutter udostępnia narzędzie o nazwie Flutter SDK, które zawiera zestaw narzędzi do tworzenia aplikacji mobilnych. W skład Flutter SDK wchodzą m.in. kompilator języka Dart, biblioteki standardowe oraz narzędzia programistyczne, takie jak narzędzie do testowania, debugowania i budowania aplikacji. Flutter oferuje również interaktywny konsolowy interfejs deweloperski (CLI), który ułatwia tworzenie i zarządzanie projektami.

W przypadku React Native, narzędzia dostarczane są przez pakiet o nazwie React Native CLI. Pakiet ten zawiera narzędzia umożliwiające tworzenie, uruchamianie i testowanie aplikacji, takie jak bundler Metro, interaktywny CLI oraz narzędzia do budowania aplikacji.

Flutter i React Native mają różne wymagania dotyczące środowiska pracy. W przypadku Flutter, wymagane jest zainstalowanie środowiska Dart i Flutter SDK. Flutter można uruchamiać na wielu platformach, takich jak Windows, macOS i Linux. Aplikacje Flutter

można tworzyć w różnych edytorach, takich jak Android Studio ,Visual Studio Code, IntelliJ IDEA i inne.

W przypadku React Native, wymagane jest zainstalowanie Node.js, pakietu npm oraz React Native CLI. Środowisko pracy React Native działa na różnych platformach, takich jak Windows, macOS i Linux. Aplikacje React Native można tworzyć w różnych edytorach, takich jak Visual Studio Code, Atom, Sublime Text i inne.

Podsumowując, zarówno Flutter, jak i React Native oferują zestaw narzędzi oraz wygodne środowisko pracy. Różnią się jednak wymaganiami dotyczącymi środowiska pracy oraz narzędziami, które są dostępne w pakietach narzędziowych.

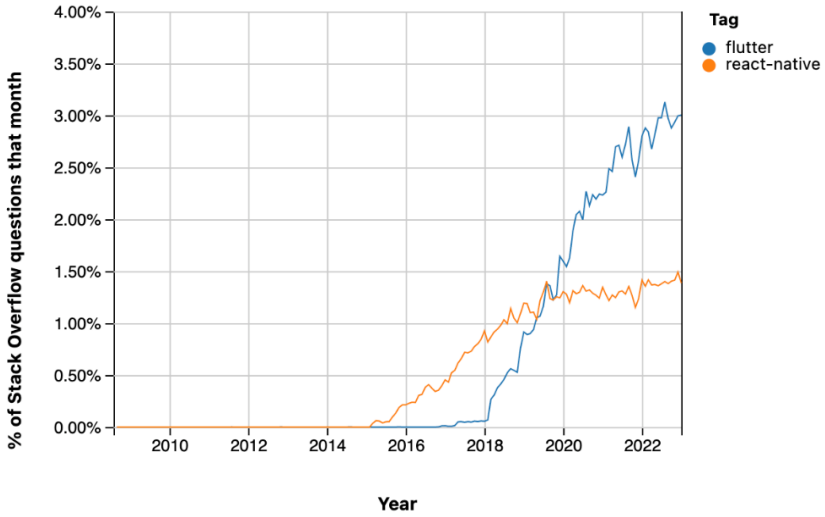
2.3 Społeczność i wsparcie

W ramach tej pracy magisterskiej przeprowadzono porównanie popularności obu frameworków, tj. Flutter i React, za pomocą narzędzi Stack Overflow Trends i Google Trends.

Stack Overflow to największe i najbardziej uznane społecznościowe miejsce dla programistów, którzy szukają pomocy w rozwiązywaniu problemów i dzielenia się wiedzą. Stack Overflow Trends to narzędzie, które analizuje i wyświetla dane dotyczące popularności różnych technologii i języków programowania.

Wyniki Stack Overflow Trends wskazują, że w ostatnim czasie Flutter staje się coraz bardziej popularny w porównaniu z React Native. W ciągu ostatnich kilku lat liczba pytań związanych z Flutter wzrosła znacznie, a liczba pytań związanych z React Native pozostała stosunkowo stała.

Ilustracja 3 Procentowa ilość zapytań odnośnie fluttera i react-native na serwisie Stack Overflow[23]

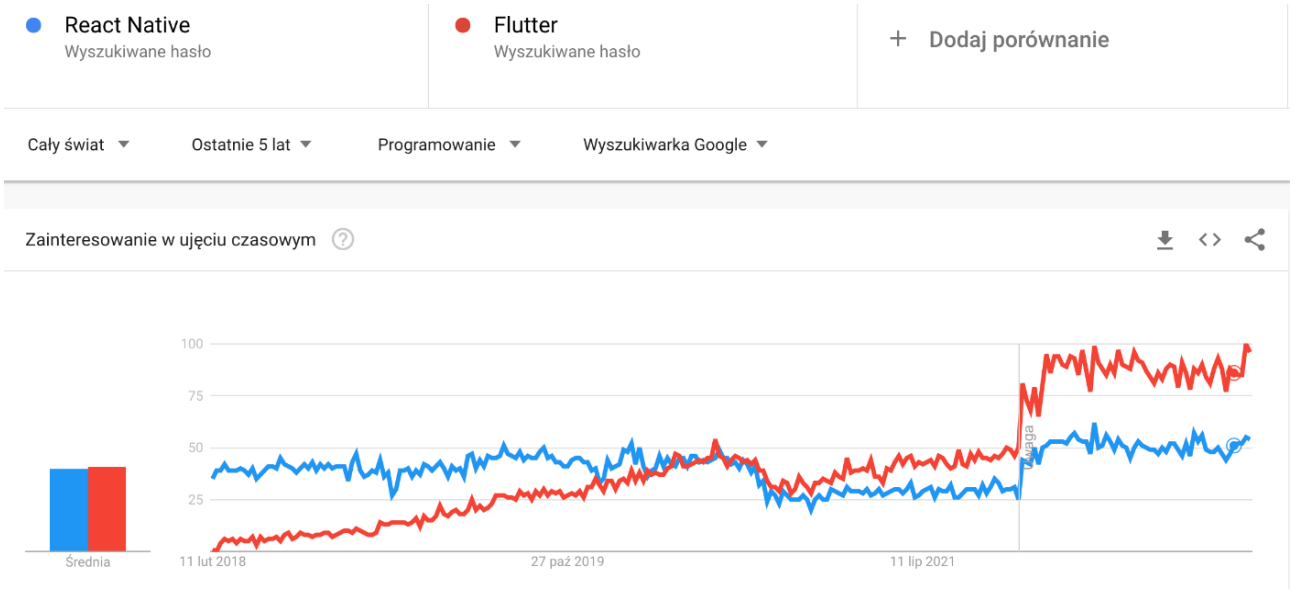


Na ilustracji numer 3 przedstawiono wykres, który obrazuje procentowe ilości zapytań z poszczególnych miesięcy w ciągu ostatnich lat. Można zauważyć, że aktualnie popularność Fluttera przewyższa popularność React Native na tym serwisie. Takie wyniki pokazują, że coraz więcej programistów decyduje się na korzystanie z technologii Flutter, a jednocześnie liczba osób korzystających z React Native nie wzrosła w takim samym stopniu.

Google Trends to narzędzie udostępniane przez Google, które pozwala na analizowanie popularności konkretnych fraz wyszukiwanych w Google przez użytkowników z różnych regionów i w różnym czasie. Google Trends pozwala na monitorowanie trendów w wyszukiwanych i umożliwia sprawdzenie, jak zmieniają się preferencje użytkowników w związku z konkretnymi tematami.

[23] (Stack Overflow 2023) <https://insights.stackoverflow.com/trends?tags=flutter%2Creact-native&fbclid=IwAR1OVOK4t30GT8zplW_CymJ34lwj4KgPkRwtei0hrhSkuVvvLq8W6MRNgf8> [dostęp 20.02.2023]

Ilustracja 4 Zainteresowanie danymi technologiami w ujęciu czasowym na podstawie Google Trends[24]



Na podstawie ilustracji numer 4 można zauważyć, że popularność wyszukiwań w Google dotyczących Fluttera zaczęła rosnąć od 2017 roku i w ciągu ostatnich kilku lat systematycznie zyskuje na popularności. Z kolei wyszukiwania dotyczące React Native osiągnęły szczyt popularności w 2018 roku i od tamtej pory stopniowo tracą na znaczeniu. Można z tego wnioskować, że coraz więcej programistów szuka informacji na temat Fluttera i zainteresowanie tą technologią będzie prawdopodobnie rosnąć w kolejnych latach. Z drugiej strony, popularność React Native wydaje się spadać, co może sugerować, że programiści szukają innych rozwiązań do tworzenia aplikacji mobilnych.

Dostępność wsparcia ma duże znaczenie dla programistów, którzy w trakcie pracy nad aplikacjami mobilnymi napotykają na różne problemy i wyzwania.

W przypadku Flutter, Google udostępnia kompleksową dokumentację[25], która zawiera szczegółowe instrukcje dotyczące korzystania z różnych funkcjonalności i narzędzi. W ramach wsparcia dla deweloperów, Google organizuje również różnego rodzaju szkolenia, spotkania i

[24] (Google Trends 2023) - <https://trends.google.com/trends/explore?cat=31&date=today%205-y&q=React%20Native,Flutter> [dostęp 20.02.2023]

[25] (Google 2023) - https://docs.flutter.dev/ [dostęp 20.02.203]

konferencje, które pozwalają na zdobycie wiedzy i wymianę doświadczeń. Na uwagę zasługuje także społeczność Fluttera, która jest bardzo aktywna i tworzy wiele narzędzi oraz bibliotek, które ułatwiają pracę z tą technologią. Wsparcie społeczności Fluttera pozwala na szybkie rozwiązanie wielu problemów oraz na łatwiejsze dostosowanie aplikacji do indywidualnych potrzeb.

W przypadku React Native, również dostępna jest bogata dokumentacja[26], która jest stale aktualizowana i rozwijana. Wsparcie programistów zapewniają również różnego rodzaju szkolenia i konferencje organizowane przez Facebooka, twórców tej technologii. Społeczność React Native jest również bardzo aktywna i tworzy liczne narzędzia oraz biblioteki, które ułatwiają pracę z tą technologią. Istotne jest również to, że wiele aplikacji mobilnych opartych na React Native jest na bieżąco aktualizowanych i rozwijanych, co przekłada się na dostępność nowych narzędzi i funkcjonalności dla deweloperów.

Podsumowując, zarówno Flutter jak i React Native oferują bogate wsparcie dla programistów, w postaci dokumentacji, szkoleń, konferencji oraz aktywnych społeczności deweloperów. Dostępność takiego wsparcia pozwala na szybsze i bardziej efektywne tworzenie aplikacji mobilnych.

[26] (Meta Platforms 2023) - https://reactnative.dev/docs/getting-started [dostęp 20.02.203]

3. Analiza porównawcza wybranych aplikacji mobilnych

Rozdział trzeci niniejszej pracy skupia się na analizie porównawczej wybranych aplikacji mobilnych zaimplementowanych przy użyciu dwóch popularnych technologii: Flutter i React Native. W ramach tego rozdziału zostanie dokładnie omówiona metodologia porównawcza, która pozwoli na wykonanie obiektywnej analizy, a także zostaną przedstawione przykładowe aplikacje mobilne zrealizowane w obu technologiach.

Przy porównywaniu aplikacji mobilnych stworzonych w różnych technologiach, istotne jest zapewnienie obiektywności analizy. W tym celu, w pierwszej kolejności zostanie dokładnie przedstawiona metodyka porównania, która umożliwi wykrycie różnic pomiędzy aplikacjami stworzonymi w technologiach Flutter i React Native.

Po przedstawieniu metodyki porównania, w pracy zostaną omówione przykładowe aplikacje mobilne, które zostały stworzone w obu technologiach W końcowej części rozdziału zostaną przedstawione wyniki analizy porównawczej, które pozwolą na określenie, która z technologii - Flutter czy React Native - sprawdziła się lepiej w kontekście projektowania aplikacji mobilnych.

3.1 Metodologia porównania

Jednym z kluczowych aspektów jakości aplikacji mobilnej jest jej wydajność. Wydajność aplikacji mobilnej jest to zdolność aplikacji do działania płynnie i bezproblemowo na urządzeniach mobilnych. W celu zapewnienia wysokiej jakości aplikacji mobilnej, ważne jest, aby projektować i testować aplikacje zgodnie z dobrymi praktykami i metodologiami pomiaru wydajności.

Metodologia pomiaru wydajności aplikacji mobilnych zależy od wielu czynników, takich jak typ aplikacji, platforma, urządzenie mobilne, sieć i środowisko. W ogólności, do pomiaru wydajności aplikacji mobilnych wykorzystuje się różne metryki, takie jak czas ładowania aplikacji, czas reakcji na interakcję użytkownika, zużycie baterii, zużycie pamięci, zużycie danych sieciowych, szybkość przetwarzania danych itp.

Istnieją różne narzędzia i techniki, które pomagają w pomiarze wydajności aplikacji mobilnych. Jednym z popularnych narzędzi jest Profiler, wbudowane narzędzie w środowisko programistyczne, które pozwala na monitorowanie wydajności aplikacji podczas jej działania.

Innym narzędziem jest testowanie wydajnościowe, które polega na symulowaniu obciążenia aplikacji, aby sprawdzić jej wydajność w różnych warunkach.

W celu porównania wydajności dwóch omawianych frameworków, przeprowadzono badanie na podobnych aplikacjach. Na potrzeby badania napisano dwie aplikacje mobilne, które wizualnie i funkcjonalnie były jak najbardziej zbliżone do siebie. Jedna aplikacja została napisana przy użyciu frameworka React Native, natomiast druga przy użyciu frameworka Flutter.

W badaniu wydajności dwóch frameworków React Native i Flutter, użyto napisanego benchmarku. Benchmark jest to test wydajności, który pozwala na porównanie wydajności różnych technologii i narzędzi. W przypadku badania frameworków, benchmark pozwalał na określenie, który framework oferuje lepszą wydajność w konkretnych scenariuszach.

Do przeprowadzenia testu wydajności użyto napisanej funkcji. Funkcja ta przyjmowała jako argumenty testową funkcję oraz dane testowe, na których była ona uruchamiana. Ponadto, funkcja pozwalała na określenie liczby uruchomień testu. W wyniku każdego uruchomienia testowej funkcji, zbierane były czasy wykonywania i zapisywane do tablicy. Następnie, na podstawie zebranych danych, obliczane były wartości statystyczne, takie jak maksymalna, minimalna i średnia wartość czasu wykonywania, a także odchylenie standardowe.

Warto dodać, że w przypadku testów wydajności w React Native wykorzystano funkcję performance.now(), która dostarcza informacji o czasie wykonywania kodu w milisekundach. Natomiast w Flutter użyto klasy Stopwatch, która pozwala na pomiar czasu z dokładnością do mikrosekund. Natomiast przedstawione wyniki zostały przekonwertowane w przypadku React Native z milisekund na mikrosekundy.

3.2 Charakterystyka wybranych aplikacji

W tym podrozdziale przedstawione zostaną dwie aplikacje mobilne napisane w dwóch różnych technologiach: Flutter oraz React Native, z wykorzystaniem różnych algorytmów.

Pierwszą z omawianych aplikacji jest ta napisana w Flutterze. W klasie MyApp, która jest rozszerzeniem StatefulWidget, zdefiniowana jest metoda build, zwracająca widget MaterialApp. Ten widget implementuje standardowe wzorce projektowe Material Design i definiuje podstawowe funkcjonalności aplikacji.

W obrębie MaterialApp, zdefiniowane są wiersze nawigacji, w których nazwy poszczególnych ekranów są mapowane na odpowiadające im widgety. Każdy z tych widgetów jest widoczny po wybraniu określonej ścieżki nawigacji. Wiersz nawigacji jest definiowany przez routes i zawiera parę klucz-wartość, gdzie klucz to nazwa ekranu, a wartość to funkcja, która zwraca ekran.

W pliku main.dart, podczas wywołania runApp(MyApp()), inicjalizowany jest widget MyApp, który tworzy MaterialApp z zdefiniowanymi ścieżkami nawigacji dla poszczególnych ekranów. Domyślna ścieżka "/" jest ustawiona jako initialRoute, a po uruchomieniu aplikacji widoczny jest ekran HomeScreen. Gdy użytkownik wybierze inną ścieżkę nawigacji, widoczny jest odpowiadający mu ekran.

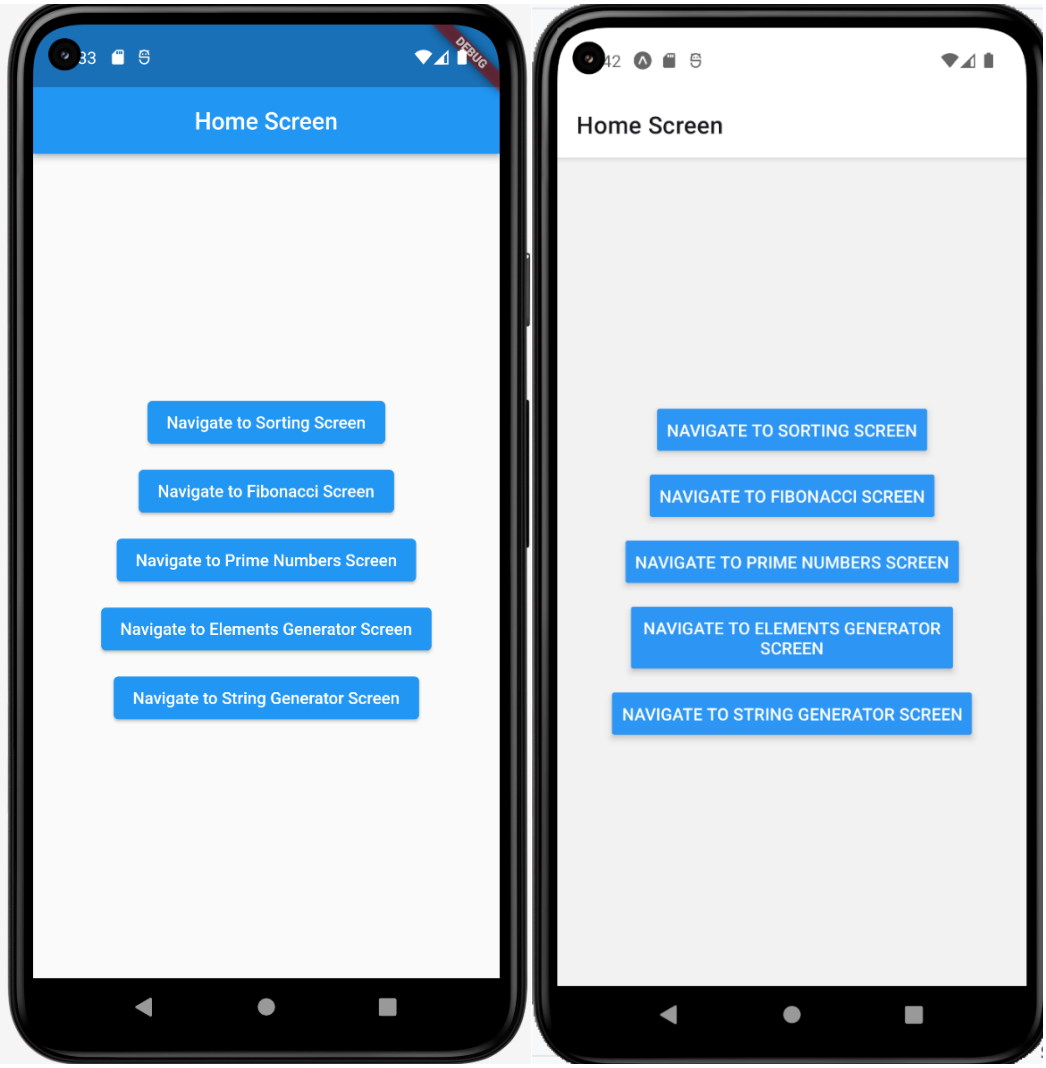
W aplikacji Flutter zdefiniowany jest Stack Navigator, który pozwala na nawigację między różnymi ekranami. W Stack Navigator zdefiniowane są poszczególne ekrany jako Stack.Screen, każdy mapowany na swoją nazwę używaną w nawigacji oraz na odpowiadający mu komponent.

Drugą z omawianych aplikacji jest ta napisana w technologii React Native. W pliku App.js, importowane są komponenty NavigationContainer oraz createNativeStackNavigator z biblioteki @react-navigation/native-stack. NavigationContainer to komponent zawierający całą logikę nawigacji dla aplikacji, a createNativeStackNavigator umożliwia nawigację między różnymi ekranami.

W funkcji App() tworzony jest NavigationContainer zdefiniowanym Stack Navigator zawierającym poszczególne ekrany jako Stack.Screen, każdy mapowany na swoją nazwę używaną w nawigacji oraz na odpowiadający mu komponent.

Podczas wyświetlania aplikacji, widoczny jest pierwszy ekran zdefiniowany jako domyślny w Stack.Navigator. Gdy użytkownik wybierze inną ścieżkę nawigacji, widoczny jest odpowiadający mu ekran. Obie aplikacje posiadają różnorodne ekrany, z których każdy umożliwia użytkownikom różne rodzaje interakcji. Poniżej przedstawiono wygląd ekranów początkowych aplikacji obu aplikacji.

Ilustracja 5 Wygląd ekranów głównych obu aplikacji (lewy ekran aplikacja napisana za pomoca Fluttera, prawy ekran aplikacja napisana za pomocą React Native).[27]



Poniżej przedstawione zostały opisy poszczególnych ekranów:

- Sorting Screen - ekran odpowiedzialny za interfejs użytkownika umożliwiający generowanie losowych danych, sortowanie ich wybranymi algorytmami sortowania oraz wyświetlanie wyników wraz z czasami wykonania. Zawiera pola tekstowe do wprowadzania liczby elementów i liczby iteracji dla testów wydajnościowych oraz przyciski do generowania danych, sortowania elementów za pomocą algorytmów sortowania (Insertion Sort, Bubble Sort, Quick Sort) oraz do czyszczenia

[27] Źródło: Opracowanie własne

wyświetlonych danych. W sekcji wyników wyświetlana jest informacja o czasie wykonania oraz posortowane dane.

- Fibonacci Screen – ekran umożliwia użytkownikowi wprowadzenie liczby oraz liczby powtórzeń testu, a następnie wykonanie testu wydajnościowego i obliczenie liczby Fibonacciego dla wprowadzonej liczby. Wyniki testu wyświetlane są na ekranie, wraz z obliczoną liczbą Fibonacciego.
- Prime Screen - reprezentuje ekran aplikacji, który pozwala użytkownikowi na wygenerowanie i przetestowanie danej liczby liczb pierwszych. Komponent składa się z dwóch pól tekstowych, w których użytkownik może wprowadzić liczbę elementów i liczbę testów, a także przycisku "Generate Prime Numbers", który generuje i testuje liczbę pierwszych na podstawie wprowadzonych danych. Po naciśnięciu przycisku "Generate Prime Numbers" wyniki testów i wygenerowane liczby pierwsze są wyświetlane na ekranie w polach tekstowych.
- Elements Generator Screen - umożliwia użytkownikowi wygenerowanie listy elementów w zależności od wyboru (przyciski, wiersze lub tabele). Użytkownik może ustawić ilość elementów, które mają zostać wygenerowane, a także wykonać test wydajności, określając liczbę testów, które mają zostać wykonane. W zależności od wybranego trybu, elementy są generowane i wyświetlane na ekranie.
- String Generator Screen - reprezentuje ekran aplikacji mobilnej, który umożliwia użytkownikowi generowanie losowych ciągów znaków o zadanej długości oraz wykonywanie testu, aby sprawdzić wydajność funkcji odwracającej ciągi znaków. Użytkownik może wprowadzić długość ciągu oraz liczbę prób testowych do wykonania, Na ekranie wyświetlają się wyniki testów oraz wygenerowany ciąg.

Ekrany w obu aplikacjach mają podobne funkcjonalności. W celu porównania wydajności obu aplikacji, przeprowadzono testy wydajnościowe każdego z ekranów. Wszystkie testy wydajnościowe zostały przeprowadzone na wirtualnym urządzeniu Pixel 5, wykorzystując Android Studio. Wirtualne urządzenie w Android Studio to narzędzie, które pozwala na uruchomienie wirtualnego środowiska Androida na komputerze, bez potrzeby posiadania fizycznego urządzenia. Za pomoca emulatorów można przeprowadzać testy aplikacji na różnych wersjach systemu Android i różnych konfiguracjach urządzeń bez

konieczności posiadania ich fizycznie. Wyniki tych testów zostaną przedstawione w kolejnym podrozdziale.

3.3 Porównanie wyników osiąganych przez aplikacje stworzone w technologiach Flutter i React Native

W niniejszym rozdziale dokonamy porównania wyników osiąganych przez aplikacje stworzone w technologiach Flutter i React Native. Pierwszy test polegał na porównaniu trzech algorytmów sortowania: Quick Sort (Szybkie sortowanie), Bubble Sort (Sortowanie bąbelkowe) i Insertion Sort (Sortowanie przez wstawianie).

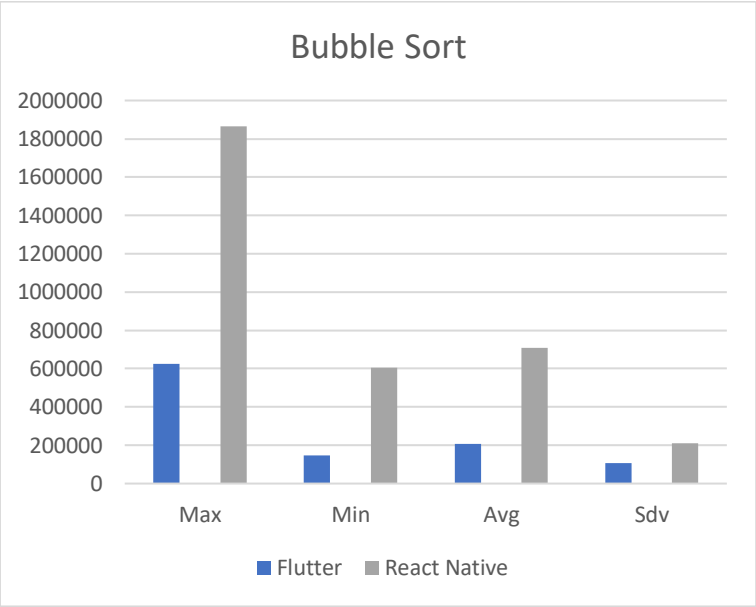
Quick Sort to algorytm szybkiego sortowania. Charakteryzuje się on tym, że w każdej iteracji wybiera jeden element (zwany pivotem) z listy i dzieli pozostałe elementy na dwie podlisty - mniejsze i większe od pivota. Następnie rekurencyjnie sortuje każdą z tych podlist. Jest to najbardziej efektywna metoda sortowania pośród trzech wymienionych.

Bubble Sort to algorytm sortowania bąbelkowego. Polega on na porównywaniu każdej pary kolejnych elementów i zamianie ich miejscami, jeśli nie są w dobrej kolejności. Ten proces jest powtarzany aż do momentu, gdy lista zostanie w pełni posortowana. Sortowanie bąbelkowe jest stosunkowo wolniejszym algorytmem od szybkiego sortowania.

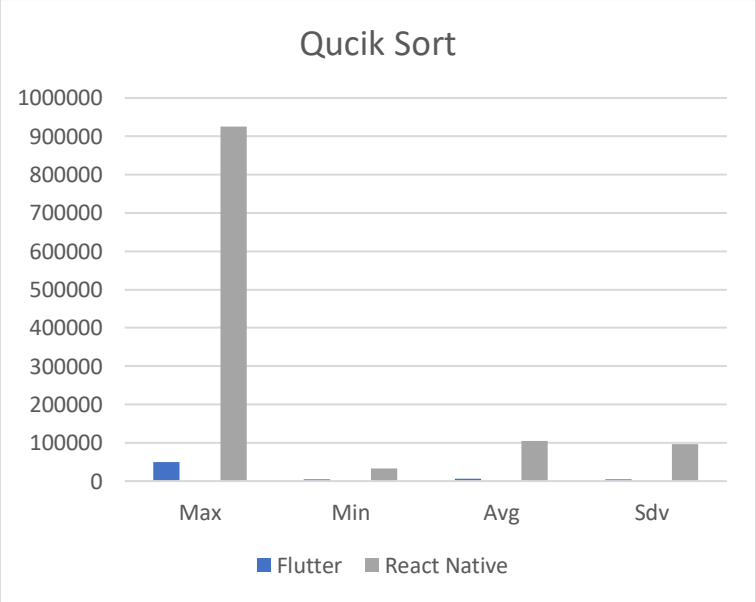
Insertion Sort to algorytm sortowania przez wstawianie. Polega on na porównywaniu każdego elementu z poprzednimi elementami listy i wstawianiu go w odpowiednie miejsce w liście, aby zachować porządek. Ten sposób sortowania jest mniej efektywny im więcej elementów jest do posortowania.

Poniżej przedstawione są wykresy prezentujące minimalny i maksymalny czas wykonania testów, średnią arytmetyczną oraz odchylenie standardowe czasu wykonania testów. Wartości czasu wyrażone są w mikrosekundach.

Wykres 1 Porównanie czasów wykonania sortowania Bubble Sort dla 10000 liczb wykonanych w 100 testach na platformach Flutter i React Native, wyrażonych w mikrosekundach.[28]



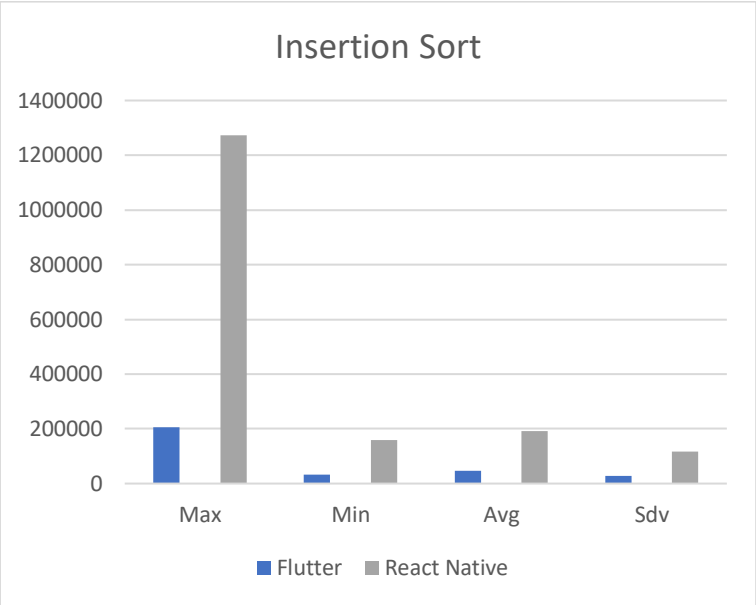
Wykres 2 Porównanie czasów wykonania sortowania Quick Sort dla 10000 liczb wykonanych w 100 testach na platformach Flutter i React Native, wyrażonych w mikrosekundach.[29]



[28] Źródło: Opracowanie własne

[29] Źródło: Opracowanie własne

Wykres 3 Porównanie czasów wykonania sortowania Insertion Sort dla 10000 liczb wykonanych w 100 testach na platformach Flutter i React Native, wyrażonych w mikrosekundach.[30]



Na podstawie wyników przedstawionych na wykrach można zauważyć, że w przypadku wszystkich trzech algorytmów sortowania, Flutter uzyskał lepsze wyniki od React Native.

W przypadku algorytmu Quick Sort, Flutter osiągnął średni czas wykonania około 104368 mikrosekund, podczas gdy w React Native było to około 5562 mikrosekund.

Podobnie, w przypadku algorytmów Bubble Sort i Insertion Sort, Flutter osiągnął lepsze wyniki niż React Native, co sugeruje, że Flutter jest bardziej wydajny niż React Native w przypadku tych algorytmów sortowania.

Drugi test polegał na porównaniu wydajności generowania ciągu Fibonacciego. Ciąg Fibonacciego to sekwencja liczb naturalnych, w której każda liczba poza dwoma pierwszymi jest sumą dwóch poprzednich: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, itd.

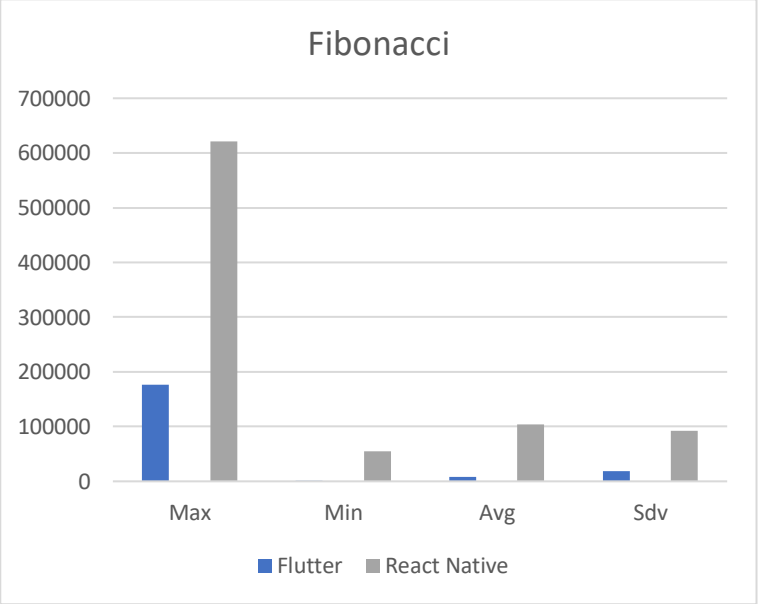
Generowanie ciągu Fibonacciego jest dobrym przykładem algorytmu rekurencyjnego. W przypadku generowania w sposób rekurencyjny, dla danego n, trzeba wygenerować dwa

[30] Źródło: Opracowanie własne

kolejne wyrazy ciągu Fibonacciego dla n-1 i n-2, a następnie dodać je do siebie, aby uzyskać n-ty wyraz ciągu.

Najważniejszą różnicą między aplikacjami jest sposób, w jaki operacje na dużych liczbach są wykonywane. W kodzie napisanym za pomocą React Native jest wykorzystywana biblioteka big-integer, która zapewnia możliwość wykonywania operacji na dużych liczbach. W kodzie napisanym za pomocą Fluttera natomiast, jest wykorzystywana klasa BigInt, która jest już wbudowana w język Dart. Wykres poniżej pokazuje wartości w mikrosekundach.

Wykres 4 Porównanie czasów wykonania obliczeń ciągu Fibonacciego dla 10000 liczby, przeprowadzone w 100 testach na platformach Flutter i React Native, wyrażonych w mikrosekundach.[31]



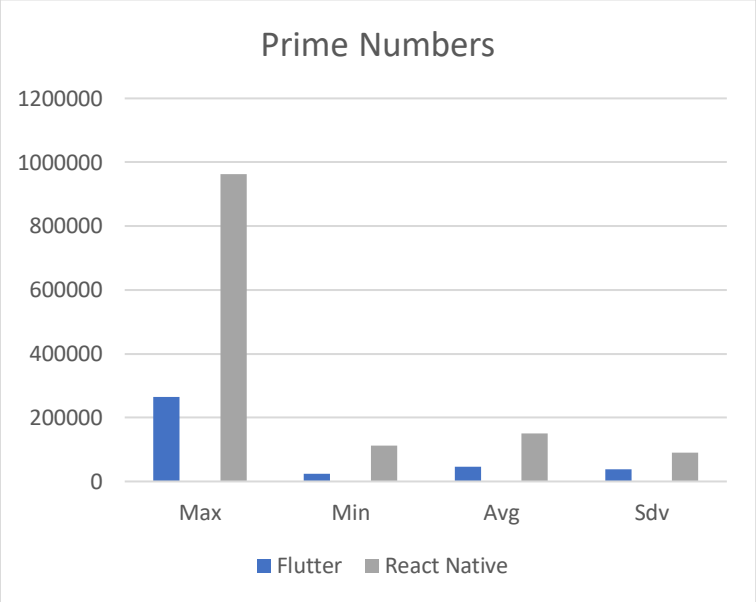
Na podstawie wyników przedstawionych na wykresie można zauważyć, że implementacja w języku Flutter wykonywała obliczenia ciągu Fibonacciego szybciej niż implementacja w języku React Native. Średni czas wykonania obliczeń w Flutter wynosił 7617,15 mikrosekund, podczas gdy w React Native był to 103945,52 mikrosekund. Różnica ta

[31] Źródło: Opracowanie własne

wynosiła około 13,6 razy. Zestawienie wartości minimalnej, maksymalnej i standardowego odchylenia pokazuje, że wyniki testów dla obu implementacji były dość stabilne.

Trzeci test polegał na porównaniu czasu generowania liczb pierwszych. W obu kodach, program iteruje przez liczby, sprawdzając każdą z nich, aby sprawdzić, czy jest pierwsza. Wykres przedstawiony poniżej przedstawia czas w mikrosekundach.

Wykres 5 Porównanie czasów wykonywania testu generowania 10000 liczb pierwszych wykonanego 100 razy na platformach Flutter i React Native, wyrażonych w mikrosekundach.[32]



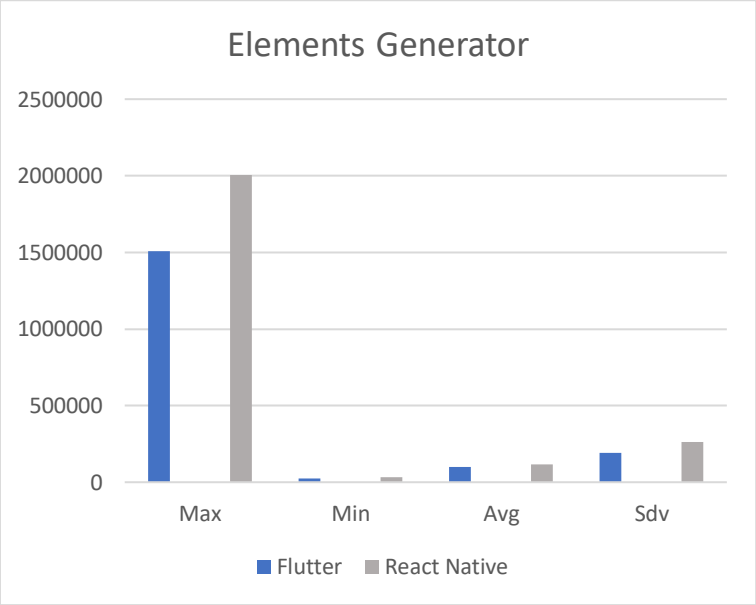
Porównanie wyników testów dla generowania liczb pierwszych pokazuje, że kolejny raz Flutter był szybszy niż React Native. Średni czas wykonania testu dla Fluttera wynosił około 46,4 milisekund, podczas gdy dla React Native wynosił około 149,7 milisekund. Maksymalny czas wykonania testu dla React Native wyniósł 963,1 milisekundy, podczas gdy maksymalny czas dla Fluttera wyniósł 264,8 milisekundy. Minimalny czas wykonania testu dla Fluttera

[32] Źródło: Opracowanie własne

wyniósł 23,9 milisekundy, podczas gdy minimalny czas dla React Native wyniósł 112,9 milisekundy. Odchylenie standardowe dla React Native wyniosło 90,9 milisekund, podczas gdy dla Fluttera wyniosło 37,6 milisekundy.

Zadanie wykonywane w czwartym teście polegało na porównaniu czasów generowania przycisków, wierszy lub tabel w zależności od wyboru użytkownika. Test ten był przeprowadzany analogicznie do poprzednich testów, czyli dla każdej implementacji uruchamiano kod testowy sto razy dla 10 0000 elementów, a następnie zbierano czasy wykonania i obliczano wartości maksymalne, minimalne, średnie oraz odchylenie standardowe. Poniżej wykres obrazujący wyniki:

Wykres 6 Porównanie czasów wykonywania testu generowania 10000 przycisków wykonanego 100 razy na platformach Flutter i React Native, wyrażonych w mikrosekundach.[33]



W teście generowania 10 000 przycisków, oba frameworki uzyskały wyniki, które są na podobnym poziomie, ale Flutter osiągnął lepsze wyniki niż React Native. Średni czas generowania przycisków w Flutterze wynosił około 100500 mikrosekund, podczas gdy w React

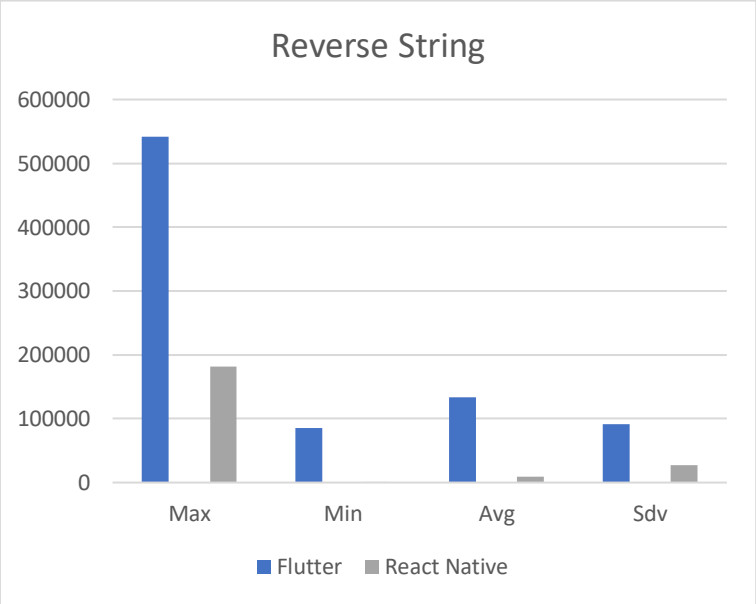
[33] Źródło: Opracowanie własne

Native wynosił około 117500 mikrosekund. Oba średnie czasy są jednak dość zbliżone, więc ciężko wskazać jednoznacznie, który framework jest lepszy w tym przypadku.

Warto zauważyć, że odchylenie standardowe czasów wykonania testów w obu frameworkach jest dość duże, co może sugerować, że czas wykonania testów może w zależności od niektórych czynników znacznie się różnić.

Ostatni test polegał na porównywaniu czasu odwracania łańcucha znaków o określonej liczbie znaków wskazanej przez użytkownika. Test ten był przeprowadzony analogicznie do poprzednich, czyli 100 razy dla każdej liczby znaków, a wyniki były mierzone w mikrosekundach. Wyniki przedstawiono na wykresie:

Wykres 7 Porównanie czasu testu odwracania stringa z 20000 znakami. Test przeprowadzony został 100 razy na platformach Flutter i React Native, wyrażonych w mikrosekundach.[34]



Średni czas odwracania łańcucha znaku w React Native wynosił 9589,05 mikrosekund, a w Flutter wynosił 133802 mikrosekundy.

[34] Źródło: Opracowanie własne

W przypadku operacji odwracania łańcuchów znaków, React Native wykazał znacznie lepszą wydajność niż Flutter, co sugeruje, że React Native może być bardziej efektywny w operacjach związanych z przetwarzaniem tekstu.

Warto jednak pamiętać, że testy wydajnościowe zawsze powinny być traktowane jako jedynie wskazówki, a nie jednoznaczne dowody na to, który framework jest lepszy. Wyniki testów mogą się różnić w zależności od wielu czynników, takich jak sprzęt, system operacyjny czy konfiguracja urządzenia testowego. Ponadto, różne implementacje aplikacji mogą prowadzić do różnych wyników testów wydajnościowych.

Zakończenie

Podsumowując, w ramach pracy magisterskiej dokonano porównania technologii Flutter i React Native pod względem wydajności. W pierwszej części pracy omówiono obie technologie, w tym język Dart oraz opisano ich historię. Następnie dokonano porównania architektury, funkcjonalności, narzędzi i społeczności obu technologii. W ostatniej części pracy przeprowadzono analizę porównawczą aplikacji mobilnych napisanych w Flutter i React Native.

Badania wykazały, że aplikacje stworzone w technologii Flutter charakteryzowały się w większości przypadków lepszymi wynikami pod względem wydajności. Poziom trudności w tworzeniu aplikacji przy użyciu obu technologii okazał się porównywalny, a wady i zalety każdej z nich zostały szczegółowo omówione. Różnice między aplikacjami stworzonymi w Flutter i React Native zostały przeanalizowane.

Wnioski z przeprowadzonych badań mogą pomóc programistom i przedsiębiorstwom w wyborze odpowiedniej technologii do tworzenia aplikacji mobilnych. Warto zaznaczyć, że wybór technologii zależy od konkretnych potrzeb i wymagań projektu, a nie tylko od parametrów wydajnościowych.

Spis wykorzystanych źródeł

Bibliografia

1. Biessek. Flutter i Dart 2 dla początkujących. Gliwice: Helion S.A., 2021.
2. Banks, E. Porcello. React od podstaw. Nowoczesne wzorce tworzenia aplikacji. Gliwice: Helion S.A., 2021.
3. Eisenman. React Native. Tworzenie aplikacji mobilnych w języku JavaScript. Wydanie II. Gliwice: Helion S.A., 2018.
4. S. Stoyan. React w działaniu. Tworzenie aplikacji internetowych. Gliwice: HELION S.A., 2021.
5. (A. Cińcio 2022) - <https://nofluffjobs.com/blog/react-native-developer-czym-sie-zajmuje-i-jak-nim-zostac/> [dostęp 20.02.203].
6. (A. Denisov 2022) - <https://medium.com/flutter-community/flutter-for-apple-tv-756fcd5e8113> [dostęp 20.02.2023]
7. (A. Safonov) 2023 - <https://merehead.com/blog/cross-platform-native-mobile-app-development/> [dostęp 20.02.2023]
8. (Apple 2014) - <https://developer.apple.com/swift/blog/?id=14> [dostęp 20.02.203].
9. (D. Bolton 2019) - https://www.dice.com/career-advice/fall-rise-dart-google-javascript-killer [dostęp 20.02.2023]
10. (Google 2023) - <https://docs.flutter.dev/> [dostęp 20.02.203].
11. (Google 2023) - <https://skia.org/> [dostęp 20.02.2023]
12. (Google Trends 2023) - <https://trends.google.com/trends/explore?cat=31&date=today%205-y&q=React%20Native,Flutter> [dostęp 20.02.2023].
13. (J. Budny 2019) https://www.netguru.com/blog/react-native-pros-and-cons
14. (J. Wasiak 2022) - <https://teamquest.pl/blog/2442_objective-c-czym-jest> 04 04 2022.
15. (K. Nahotko 2022) - https://boringowl.io/tag/flutter [dostęp 20.02.2023]
16. (K. Walrath 2019) - https://medium.com/dartlang/dart-asynchronous-programming-futures-96937f831137 [dostęp 20.02.2023]

17. (M. Carrington 2022) - <https://www.velvetech.com/blog/native-mobile-app-development/> [dostęp 20.02.2023]
18. (Meta Platforms 2023) - <https://reactnative.dev/docs/getting-started> [dostęp 20.02.203].
19. (P. Dedio 2019) - <https://android.com.pl/programowanie/182482-poczatki-programowania-java-kotlin/> [dostęp 20.02.2023]
20. (P. Chmielewska, A. Trachim, G. Smith 2022) - <https://itcraftapps.com/blog/19-apps-built-with-flutter-framework/> [dostęp 20.02.2023].
21. (Saigon Technology 2023) - <https://saigontechnology.com/blog/flutter-in-comparison-with-react-native-and-xamarin> [dostęp 20.02.2023].
22. (Stack Overflow 2023) - <https://insights.stackoverflow.com/trends?tags=flutter%2Creact-native&fbclid=IwAR1OVOK4t30GT8zplW_CymJ34lwj4KgPkRwtei0hrhSkuVvvLq8W6MRNgf8> [dostęp 20.02.2023].
23. (U. Khan 2022) https://clutch.co/app-developers/resources/pros-cons-native-apps [dostęp 20.02.2023]

Spis ilustracji

Ilustracja 1 Kod prostej aplikacji Flutter, która wyświetli napis „Hello World” na ekranie....21

Ilustracja 2 Kod prostej aplikacji React Native, która wyświetli napis „Hello World” na ekranie 23

Ilustracja 3 Procentowa ilość zapytań odnośnie fluttera i react-native na serwisie Stack Overflow 27

Ilustracja 4 Zainteresowanie danymi technologiami w ujęciu czasowym na podstawie Google Trends 28

Ilustracja 5 Wygląd ekranów głównych obu aplikacji (lewy ekran aplikacja napisana za pomoca Fluttera, prawy ekran aplikacja napisana za pomocą React Native). 33

Spis wykresów

Wykres 1 Porównanie czasów wykonania sortowania Bubble Sort dla 10000 liczb wykonanych w 100 testach na platformach Flutter i React Native, wyrażonych w mikrosekundach.....36

Wykres 2 Porównanie czasów wykonania sortowania Quick Sort dla 10000 liczb wykonanych w 100 testach na platformach Flutter i React Native, wyrażonych w mikrosekundach.....36

Wykres 3 Porównanie czasów wykonania sortowania Insertion Sort dla 10000 liczb wykonanych w 100 testach na platformach Flutter i React Native, wyrażonych w mikrosekundach. 37

Wykres 4 Porównanie czasów wykonania obliczeń ciągu Fibonacciego dla 10000 liczby, przeprowadzone w 100 testach na platformach Flutter i React Native, wyrażonych w mikrosekundach. 38

Wykres 5 Porównanie czasów wykonywania testu generowania 10000 liczb pierwszych wykonanego 100 razy na platformach Flutter i React Native, wyrażonych w mikrosekundach. 39

Wykres 6 Porównanie czasów wykonywania testu generowania 10000 przycisków wykonanego 100 razy na platformach Flutter i React Native, wyrażonych w mikrosekundach. 40

Wykres 7 Porównanie czasu testu odwracania stringa z 20000 znakami. Test przeprowadzony został 100 razy na platformach Flutter i React Native, wyrażonych w mikrosekundach.....41

Załączniki

1. Aplikacja „Hello World”, napisana za pomocą Fluttera
   https://github.com/kgrundemann/helloworldflutter
2. Aplikacja „Hello World”, napisana za pomocą React Native
   https://github.com/kgrundemann/HelloWorldRN
3. Aplikacja testująca, napisana za pomocą React Native
   https://github.com/kgrundemann/SortingApp-rn
4. Aplikacja testująca, napisana za pomocą Fluttera-
   https://github.com/kgrundemann/SortingApp-flutter