JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 346–351
Received: 27 June 2023
Accepted: 5 September 2023

A comparative analysis of resource use in the Flutter and React Native frameworks

Analiza porównawcza wykorzystania zasobów w szkieletach programistycznych Flutter oraz React Native

Mateusz Markowski*, Jakub Smołka

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

Abstract

The article describes a comparative analysis of the resource use efficiency in mobile applications developed using Flutter and React Native frameworks. The study consisted in creating two mobile apps with the same functionalities and then comparing usage of the following resources: Virtual Memory (VIRT), Shared Memory (SHR), Central Processing Unit (CPU), Resident Set Size (RES) and Memory (MEM). The functionality of the application consisted of actions on the GUI. The test was conducted on a Huawei P20 Lite smartphone, using the Android Debug Bridge (ADB) tool and custom scripts in the Bash shell language. The results do not clearly indicate which technology is more efficient.

Keywords: Flutter; React Native; efficiency; mobile applications

Streszczenie

Artykuł opisuje analizę porównawczą efektywności wykorzystania zasobów aplikacji mobilnych stworzonych za pomocą szkieletów programistycznych Flutter oraz React Native. Badanie polegało na stworzeniu dwóch aplikacji mobilnych z tymi samymi funkcjonalnościami, a następnie porównaniu wykorzystania następujących zasobów: Virtual Memory (VIRT), Shared Memory (SHR), Central Processing Unit (CPU), Resident Set Size (RES) oraz Memory (MEM). Na funkcjonalność aplikacji składały się działania na interfejsie graficznym użytkownika. Badanie zostało przeprowadzone na telefonie Huawei P20 Lite, za pomocą narzędzia Android Debug Bridge (ADB) oraz własnych skryptów w języku powłoki Bash. Wyniki nie wskazują jednoznacznie, która technologia jest wydajniejsza.

Słowa kluczowe: Flutter; React Native; wydajność; aplikacje mobilne

*Corresponding author
Email address: mateusz.markowski@pollub.edu.pl (M. Markowski)

1. Wstęp

Od wielu lat zaobserwować można trend wzrostowy na rynku urządzeń mobilnych. Dostawcy prześcigają się w dostarczaniu użytkownikom coraz bardziej funkcjonalnych smartfonów, tabletów, smartbandów czy innych tego typu urządzeń. Dominującym systemem operacyjnym jest Android jednak coraz więcej osób posiada także urządzenia z systemem iOS. Standardowo aplikacje dla systemu Android tworzone są w językach Java/Kotlin w środowisku Android Studio natomiast dla iOS z wykorzystaniem języka Swift oraz środowiska programistycznego XCode. Tworzenie aplikacji dla dwóch różnych platform wiąże się ze zwiększonymi kosztami, co wywołuje konieczność powołania dwóch zespołów programistów, którzy pisali tą samą aplikację w dwóch różnych technologiach. W związku z dużymi kosztami tworzenia aplikacji mobilnych firmy takie jak Meta czy Google zaczęły pracę nad własnymi technologiami umożliwiającymi programistom pisanie jednocześnie na obydwa systemy mobilne. Tak powstały dwa szkielety programistyczne Flutter oraz React Native, które pojawiły się na rynku odpowiednio w 2017 roku oraz 2015 roku.

Celem niniejszego artykułu jest porównanie wydajności aplikacji mobilnych stworzonych z wykorzystaniem szkieletów programistycznych React Native i Flutter. W niniejszym artykule skupiono się na porównaniu zużycia zasobów przez aplikacje stworzone z użyciem frameworków wieloplatformowych działających w systemie Android. Motywacją do zbadania tych parametrów był fakt, iż w dotychczasowych badaniach obu technologii skupiano się na zupełnie innych charakterystykach wydajności, takich jak czas renderowania klatek. Postawiono następującą hipotezę badawczą „Aplikacje mobilne tworzone z wykorzystaniem technologii Flutter są wydajniejsze od aplikacji mobilnych tworzonych z wykorzystaniem technologii React Native".

2. Przegląd literatury

W celu przeprowadzenia rzetelnych badań został wykonany przegląd literatury. W artykule [1] przeprowadzono badanie porównawcze wydajności aplikacji mobilnych z wykorzystaniem technologii natywnych Flutter, React Native, iOS oraz Android. Celem artykułu było zbadanie czy aplikacje między-platformowe są wydajniejsze od aplikacji natywnych. Na podstawie otrzymanych wyników autorzy określili, że technologie natywne są szybsze od technologii dedykowanych konkretnym systemom mobilnym. Na podstawie otrzymanych wyni-

ków dotyczących wydajności aplikacji autorzy nie byli w stanie stwierdzić, które technologie są wydajniejsze.

Artykuł [2] opisuje porównanie wydajności dwóch aplikacji między platformowych. Autor tego badania skupił się na zmierzeniu liczby porzuconych ramek w określonym czasie przez szkielety programistyczne React Native oraz Flutter. Celem badania było sprawdzenie czy istnieje znacząca różnica pomiędzy wyżej wymienionymi technologiami. Na podstawie otrzymanych wyników stwierdzono, że szkielet programistyczny Flutter wypadł lepiej od szkieletu programistycznego React Native, pod względem liczby porzucanych ramek w aplikacjach mobilnych zawierających bardzo duże listy.

Wydajność aplikacji mobilnych można zmierzyć również z wykorzystaniem takich parametrów jak: zużycie pamięci, wykorzystanie CPU, zużycie baterii oraz FPS. W tym celu wykorzystano technologię Flutter oraz Native Mobile App [3]. Celem tego badania było odpowiedzenie na następujące pytanie: „Czy lepiej tworzyć jedną aplikację, która będzie kompatybilna z wieloma platformami, czy też wydajniej jest stworzyć wiele aplikacji dla różnych systemów?". Po przeprowadzeniu badań Autorzy stwierdzili, że rozmiar aplikacji natywnych jest niewiele mniejszy od rozmiaru aplikacji mobilnej stworzonej w technologii Flutter. Zużycie baterii jest na podobnym poziomie w obydwu technologiach. Jednak pod względem liczby wyświetlanych klatek na sekundę lepiej wypadł szkielet programistyczny Flutter.

Ważnym aspektem tworzenia aplikacji mobilnych jest również proces ich tworzenia oraz umiejętność korzystania z dokumentacji dostarczanej przez twórców. Na takie badanie zdecydowano się w artykule [4]. Gdzie Autorzy porównując technologię React Native oraz Flutter zdecydowali się przebadać obydwa szkielety wśród programistów. Dodatkowo porównano ich składnię, strukturę, wykorzystanie zmiennych, przepływ kontroli i danych oraz architekturę, na której są oparte wyżej wymienione technologie. W tym celu również zdecydowano się na przeszukanie dokumentacji każdego szkieletu programistycznego.

Po przeprowadzeniu badań autorzy uznali, że obie technologie Flutter oraz React Native posiadają wiele użytecznych aspektów. Technologia React Native była bardziej zaawansowana w porównaniu do technologii Flutter w obszarach: przygotowania środowiska programistycznego, rozwoju aplikacji internetowych, różnorodności dla komponentów gotowych do użycia oraz dostępnych zasobów. Szkielet programistyczny Flutter posiada trzystopniowy system testowy, który był bardziej przydatny pod względem rozmiaru aplikacji, zajmując mniej miejsca. Dodatkowo był szybszy od szkieletu programistycznego React Native w momencie pierwszego otwierania aplikacji. Podsumowując autorzy nie znaleźli żadnej wyróżniającej charakterystyki sugerującej, która technologia jest wydajniejsza.

W artykule [5] przedstawiono badanie dotyczące porównania aplikacji mobilnych stworzonych w szkielecie programistycznym Flutter z aplikacjami natywnymi. Badanie polegało na stworzeniu podobnych aplikacji o takich samych funkcjonalnościach w technologiach: Kotlin, Flutter oraz Swift. Celami tego badania było porównanie aplikacji pod względem wydajności procesora oraz sprawdzenie ilości kodu potrzebnego do wytworzenia dwóch identycznie działających aplikacji mobilnych. Z otrzymanych wyników wydajności procesora wynika, że nie ma prawie żadnej różnicy pomiędzy szkieletem programistycznym Flutter oraz technologiami iOS i Kotlin. Autor zaznacza, że w celu zweryfikowania otrzymanych wyników należy przeprowadzić dalsze testy, które potwierdzą otrzymane wyniki. Po porównaniu ilości kodu potrzebnego do stworzenia trzech podobnych aplikacji Autor zauważył, że w technologii Flutter wystarczył napisać 125 linii kodu. W technologii Swift 363 linie kodu. Natomiast w technologii Kotlin 217 linii kodu.

Artykuł [6] przedstawia porównanie zalet oraz wad aplikacji między-platformowych oraz natywnych. Badanie polegało na porównaniu wielkości projektów, w których wykorzystywane są technologie: React Native, Java, Kotlin, Flutter. Zbadano popularność technologii, do tego celu posłużono się danymi z Google Trends. Przedstawiono także liczbę ofert pracy dla technologii: Android, iOS, Flutter oraz React Native. Celem pracy było przeanalizowanie zalet i wad natywnych aplikacji mobilnych oraz aplikacji między-platformowych. Po przeanalizowaniu wszystkich dostępnych danych Autorzy uzyskali wyniki, z których wynika, że nie da się udzielić jednoznacznej odpowiedzi, które technologie są lepsze natywne czy między-platformowe. Wynika to z faktu, iż każda z wyżej wymienionych technologii posiada własne wymagania. Dodatkowo każdy programista posiada inny poziom wiedzy oraz preferencji dotyczących szkieletów programistycznych.

3. Metoda badawcza

W celu zbadania wydajności aplikacji opracowano scenariusz badawczy, którego zadaniem było zbadanie następujących parametrów aplikacji mobilnej: Virtual Memory (VIRT), Shared Memory (SHR), Central Processing Unit (CPU), Resident Set Size (RES), Memory (MEM). Parametr VIRT określa całkowitą ilość pamięci wirtualnej, która jest używana przez aplikację. Jest to pamięć, która dostępna jest dla procesu. Może obejmować zarówno pamięć fizyczną jak i pamięć dostępną na dysku. Wartość tego parametru mierzona jest w gigabajtach. Parametr SHR określa ilość pamięci współdzielonej przez procesy. Może być ona używana przez różne wątki i procesy w tym samym czasie. Wartość tego parametru mierzona jest w megabajtach. CPU to parametr odnoszący się do wykorzystania procesora przez aplikację. Wyrazić ją można jako procentowy udział czasu procesora przeznaczonego na wykonanie kodu aplikacji. RES określa ilość obecnie zajmowanej pamięci RAM przez aplikację, mierzona jest w megabajtach. Parametr MEM odnosi się do ogólnego wykorzystania pamięci przez aplikację. Może obejmować pamięć wirtualną jak i pamięć fizyczną. Mierzony jest w procentach [7].

Tabela 1 przedstawia scenariusz badawczy, który opracowano w celu przeprowadzenia badań wydajności aplikacji mobilnych. Grupa badawcza składała się z 15 studentów. Przed wykonaniem badania scenariusz został wyjaśniony osobom badanym. Następnie badani zostali poproszeni o wykonanie opisanych czynności. Maksymalny czas badania ustalono na 10 minut.

Do przeprowadzenia badania wykorzystano telefon marki Huawei P20 Lite ANE-LX1 z pamięcią wbudowaną 64 GB oraz pamięcią RAM 4 GB. System operacyjny to Android 9. Telefon wyposażony był w procesor Hisilicon Kirin 659, posiadający 8 rdzeni z taktowanie zegara 2,36 GHz.

Wydajność aplikacji została zmierzona za pomocą narzędzia Android Debug Bridge (ADB) oraz własnych skryptów stworzonych w języku programowania Python oraz języku powłoki Bash.

W momencie gdy badany był gotowy do wykonania scenariusza badawczego uruchamiany był jeden ze specjalnych skryptów powłoki Bash stworzonych odpowiednio dla szkieletu programistycznego React Native oraz Flutter. Skrypt (Listing 1) działał w następujący.

1. Sprawdzał czy od momentu uruchomienia upłynął czas 10 minut.
2. Jeśli warunek był fałszywy oczekiwał, aż użytkownik kliknie w dowolne miejsce na ekranie aplikacji.
3. W momencie wykrycia dotyku zbierał informacje o parametrach VIRT, SHR, RES, CPU, MEM oraz TIME.
4. Zebrane dane wysyłał do pliku z rozszerzeniem .csv.
5. Powtarzanie punktów 2 - 4 do momentu zakończenia badania, czyli zamknięcia aplikacji lub przekroczenia 10 minut od rozpoczęcia badania.
6. Wyliczenie średniej wartości zebranych parametrów oraz zapisanie wyników do pliku z rozszerzeniem .txt.
7. Uruchomienie skryptu napisanego w języku programowania Python.

Tabela 1: Scenariusz badawczy stworzony dla testowanych aplikacji

Lp.	Opis czynności	Oczekiwany wynik
1.	Wpisanie tekstu o długości 255 znaków.	Wpisano ciąg o długości 255 znaków.
2.	Skasowanie ciągu znaków z wykorzystaniem wirtualnej klawiatury.	Usunięto ciągu o długości 255 znaków.
3.	Wpisanie ciągu o długości 255 znaków.	Wpisano ciąg o długości 255 znaków.
4.	Skasowanie ciągu o długości 255 znaków z wykorzystaniem przycisku.	Usunięto ciąg o długości 255 znaków.
5.	25- krotne wciśnięcie przycisku “+”/	Stan licznika zmienił wartość z 0 na 25.
6.	25-krotne wciśnięcie przycisku “-”.	Stan licznika zmienił wartość z 25 na 0.
7.	Wciśnięcie przycisku przekierowującego do drugiego ekranu	Wyświetlany jest ekran z przewijaną listą elementów.
8.	Przewinięcie listy na sam dół.	Wyświetlany jest ostatni element z listy.
9.	Przewinięcie listy na samą górę.	Wyświetlany jest pierwszy element z listy.
10.	Wciśnięcie przycisku sortowanie.	Wyświetlana jest posortowana lista.
11.	Przewinięcie listy na sam dół.	Wyświetlany jest ostatni element z listy.
12.	Przewinięcie listy na samą górę.	Wyświetlany jest pierwszy element z listy.
13.	Wciśnięcie drugiego przycisku sortowania.	Wyświetlana jest posortowana lista w innej kolejności niż poprzednim razem.
14.	Przewinięcie listy na sam dół.	Wyświetlany jest ostatni element z posortowanej listy.
15.	Przewinięcie listy na samą górę	Wyświetlany jest pierwszy element z posortowanej listy.

Na Listingu 2 przedstawiono fragment kodu odpowiedzialnego za uruchomienie skryptu w języku programowania Python. Skrypt ten działał w następujący sposób, pobierał dane z pliku csv, w którym zapisywane były dane dla badanych parametrów: SHR, VIRT, RES, MEM oraz CPU. Następnie na ich podstawie tworzył wykres zależność ich wykorzystania w trakcie konkretnego badania. Fragment skryptu w języku Python przedstawiono na Listingu 3. Przedstawiono na nim ścieżkę z jakiej skrypt pobierał dane (w tym wypadku dla technologii Flutter) oraz w jaki sposób pobierał dane wykorzystywane na wykresie.

Listing 1: Fragment kodu skryptu w języku powłoki Bash sprawdzający kiedy należy zakończyć badanie wraz ze zbieraniem wartości

```
current_time=$(date +%s)

if [ $((current_time - start_time)) -ge 500 ] then
  break

fi

if  adb shell dumpsys window windows | grep -E
'mCurrentFocus|mFocusedApp' | grep
com.example.mgr_app_flutter >/dev/null  then

result=$(adb shell top -n 1 -d 1 | grep
com.example.mgr+ | awk '{
printf"%s,%s,%s,%s,%.2f,%s,%s,%s,%s,%.2f,%s,%s\
n",

$1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12 }')

if [ -n "$result" ]; then

echo $result >>
~/Desktop/wyniki/wyniki_flutter/dane_usrednione/dane
_szczegolowe_$1.txt

counter=$((counter+1))
```

Listing 2: Fragment kodu skryptu w języku powłoki Bash wywołujący w tle skrypt w języku programowania Python

```
#Uruchomienie skryptu w Pythonie w tle

nohup  python flutter.py user_$1.csv &

echo "Badanie zostało zakończone!!!!!!!!"
```

Listing 3: Fragmentu kodu w języku Python odpowiedzialny za wczytanie danych z pliku

```
path = '~/Desktop/wyniki/wyniki_flutter/dane_python/'

filename = path + sys.argv[1]

values = ["VIRT", "RES", "SHR", "CPU", "MEM",
"TIME"]

df = pd.read_csv(filename, header=None, names=values)

fig1, ax1 = plt.subplots()
```

4. Wyniki

Na Rysunku 5 oraz Rysunku 6 przedstawiono zrzuty ekranu prezentujące stworzoną aplikację z wykorzystaniem technologii Flutter oraz języka programowania Dart. Rysunek 7 oraz Rysunek 8 przedstawiają aplikację stworzoną w technologii React Native stworzoną przy użyciu języka programowania Java Script. W Tabeli 2 oraz w Tabeli 3 przedstawiono wyniki, które zebrano podczas prowadzonych badań.

W Tabeli 2 zaprezentowane dane dotyczą aplikacji mobilnej stworzonej z wykorzystaniem szkieletu programistycznego Flutter. Natomiast w Tabeli 3 znajdują się dane dotyczące aplikacji mobilnej stworzonej z wykorzystaniem szkieletu programistycznego React Native.

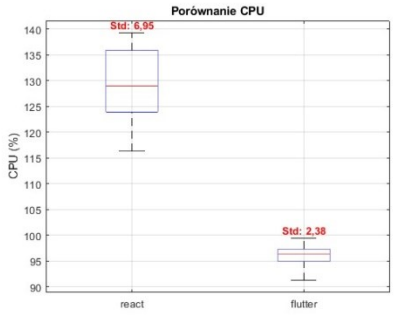
Każdy z wierszy przedstawia średnią wartość parametrów: VIRT, RES, SHR, CPU, MEM oraz czas trwania pojedynczego badania, który został zarejestrowany dla pojedynczego uczestnika badania. Na Rysunku 1 przedstawiono porównanie wykorzystania CPU dla obu technologii. Zaznaczono na nim także odchylenie standardowe dla badanego parametru, które wynosi odpowiednio około 7 % dla technologii React Native oraz około 2,4 % dla technologii Flutter. Na Rysunku 2 przedstawiono porównanie wykorzystania pamięci wirtualnej (parametr VIRT) dla technologii React Native oraz Flutter. Zmierzone odchylenie standardowe dla tego parametru jest bardzo bliskie 0 GB. Na Rysunku 3 zaprezentowano porównanie wykorzystania pamięci RAM (parametr RES) oraz pamięci współdzielonej (parametr SHR) dla technologii React Native i Flutter. Odchylenie standardowe dla parametru RES wynosi odpowiednio około 19 MB dla technologii React Native oraz około 35,5 MB dla technologii Flutter. Z kolei odchylenie standardowe dla parametru SHR dla technologii React Native wynosi około 9,6 MB, dla technologii Flutter jest to około 19,3 MB. Na Rysunku 4 przedstawiono porównanie całkowitego wykorzystania pamięci (parametr MEM). Odchylenie standardowe dla technologii React Native wynosi około 0,5 %. Natomiast dla technologii Flutter odchylenie standardowe wynosi około 0,9 %. Pomiędzy obiema badanymi technologiami największa różnica dotyczy wykorzystania przez system Android wirtualnej pamięci opisanej w tabelach jako parametr VIRT. W wartościach liczbowych to około 13 GB na korzyść technologii Flutter. Dane dotyczące wykorzystania pamięci fizycznej przez system Android prezentują się na zbliżonym poziomie i wynoszą około 5 %. Przy porównaniu wykorzystania czasu procesora przez aplikację korzystniej wypadła technologia Flutter. Średnia wartość zużycia wynosiła około 97 %. Technologia React Native wykorzystywała około 130 % czasu procesora (oznacza to zajmowanie więcej niż jednego rdzenia). Całkowity rozmiar pamięci współdzielonej przez aplikację dla szkieletu programistycznego Flutter oscylował w granicach od 50 MB do 120 MB. Dla aplikacji stworzonej z wykorzystaniem szkieletu programistycznego React Native wartość tej pamięci oscylowała w okolicy 75 MB. Wartość pamięci fizycznej określonej przez parametr RES dla aplikacji mobilnej stworzonej w technologii Flutter nie przekracza wartości 300 MB. Natomiast ten sam parametr dla technologii React Native nie przekracza wartości 170 MB.

Tabela 2: Wyniki wykorzystania zasobów VIRT, RES, SHR, CPU oraz MEM dla aplikacji mobilnej stworzonej w technologii Flutter

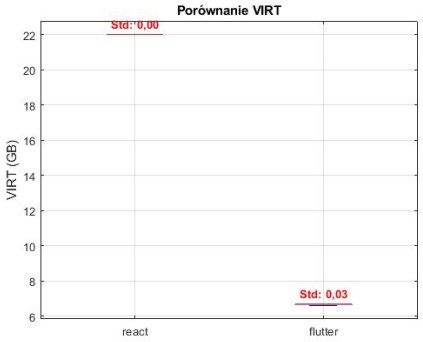
Numer badania	Średnia wartość VIRT [GB]	Średnia wartość RES [MB]	Średnia wartość SHR [MB]	Średnia wartość CPU [%]	Średnia wartość MEM [%]
1.	6,63	251	110	96,43	6,60
2.	6,62	268	113	96,65	7,04
3.	6,63	162	56	91,75	4,25
4.	6,65	159	56	95,19	4,15
5.	6,64	175	59	98,53	4,57
6.	6,67	157	58	97,49	4,12
7.	6,70	160	56	91,30	4,18
8.	6,70	160	56	96,29	4,18
9.	6,70	157	56	99,47	4,10
10.	6,70	179	58	97,42	4,68
11.	6,70	181	60	97,20	4,74
12.	6,70	185	58	94,82	4,84
13.	6,70	152	54	95,91	3,97
14.	6,70	148	55	96,50	3,87
15.	6,70	157	57	92,43	4,10

Tabela 3: Wyniki wykorzystania zasobów VIRT, RES, SHR, CPU oraz MEM dla aplikacji mobilnej stworzonej w technologii React Native

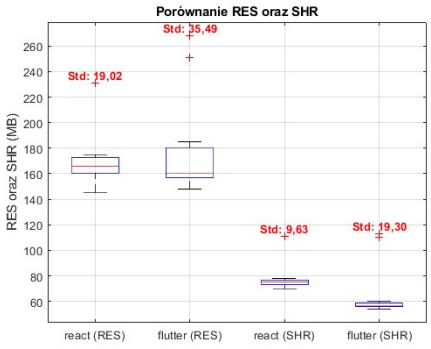
Numer badania	Średnia wartość VIRT [GB]	Średnia wartość RES [MB]	Średnia wartość SHR [MB]	Średnia wartość CPU [%]	Średnia wartość MEM [%]
1.	6,63	251	110	96,43	6,60
2.	6,62	268	113	96,65	7,04
3.	6,63	162	56	91,75	4,25
4.	6,65	159	56	95,19	4,15
5.	6,64	175	59	98,53	4,57
6.	6,67	157	58	97,49	4,12
7.	6,70	160	56	91,30	4,18
8.	6,70	160	56	96,29	4,18
9.	6,70	157	56	99,47	4,10
10.	6,70	179	58	97,42	4,68
11.	6,70	181	60	97,20	4,74
12.	6,70	185	58	94,82	4,84
13.	6,70	152	54	95,91	3,97
14.	6,70	148	55	96,50	3,87
15.	6,70	157	57	92,43	4,10



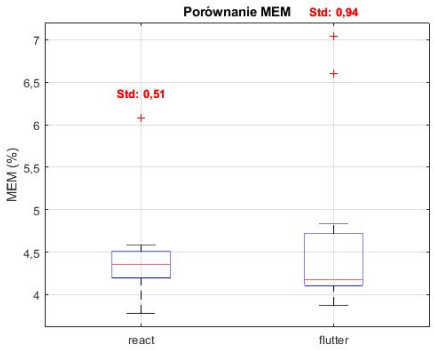
Rysunek 1: Porównanie wykorzystania parametru CPU dla technologii React Native oraz Flutter.



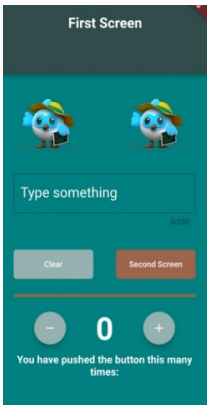
Rysunek 2: Porównanie wykorzystania pamięci wirtualnej (parametr VIRT) dla technologii React Native oraz Flutter.



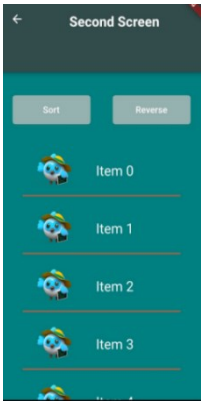
Rysunek 3: Porównanie wykorzystania pamięci RAM (parametr RES) oraz pamięci współdzielonej (parametr SHR) dla technologii React Native oraz Flutter.



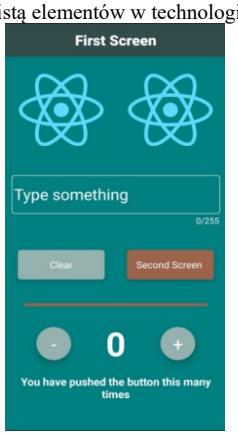
Rysunek 4: Porównanie całkowitego wykorzystania pamięci (parametr MEM) dla technologii React Native oraz Flutter.



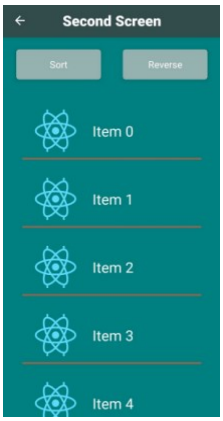
Rysunek 5: Zrzut ekranu przedstawiający główny ekran aplikacji w technologii Flutter.



Rysunek 6: Zrzut ekranu przedstawiający ekran aplikacji z przewijaną listą elementów w technologii Flutter.



Rysunek 7: Zrzut ekranu przedstawiający ekran główny aplikacji stworzony w technologii React Native.



Rysunek 8: Zrzut ekranu przedstawiający ekran aplikacji z przewijaną listą elementów w technologii React Native act Native.

5. Wnioski

Celem artykułu było porównanie wydajności aplikacji mobilnych tworzonych z wykorzystaniem technologii React Native oraz Flutter. W postawionej hipotezie badawczej założono, że efektywniejsze aplikacje mobilne tworzone są w technologii Flutter niż w technologii React Native. Po przeprowadzeniu badań i porównaniu badanych parametrów wyniki prezentują się następująco. Dla parametru MEM, dotyczącego ogólnego wykorzystania pamięci przez aplikacje obie technologie wypadły podobnie. Średnie zużycie wynosi około 4,5 %. Dla parametru VIRT, który odnosi się do całkowitej ilości pamięci wirtualnej przez aplikację lepiej wypadła technologia Flutter. Wykorzystuje ona tylko około 7 GB pamięci wirtualnej telefonu. Natomiast technologia React Native wykorzystuje 22 GB tej pamięci. Porównując procentowy udział czasu procesora przeznaczony na wykonanie kodu aplikacji również lepiej wypadał technologia Flutter. Po porównaniu wyników dotyczących ilości pamięci współdzielonej przez aplikację korzystniej wypadła technologia React Native. Również dla parametru określającego rozmiar pamięci RAM zajmowany przez aplikację lepiej wypadł szkielet programistyczny React Native. Jednak różnice między dwoma wyżej wymienionymi parametrami RES oraz SHR nie są duże i wynoszą około 20 MB. Dla obu parametrów SHR i RES najprawdopodobniej związane jest to z większą aktywnością innej aplikacji zainstalowanej na telefonie w momencie przeprowadzania badania z technologią Flutter. Jednak aby dokładniej potwierdzić ten scenariusz należałoby zwiększyć liczbę pobranych próbek badawczych.

Podsumowując po porównaniu pięciu parametrów: VIRT, SHR, RES, MEM oraz CPU nie da się jednoznacznie potwierdzić postawionej hipotezy badawczej świadczącej o lepszej wydajności technologii Flutter niż technologii React Native. W tym celu należałoby wykonać bardziej szczegółowe badania. Przykładowo można porównać inne czynniki m.in. interakcję technologię z zewnętrzną bazą danych, czas przetwarzań żądań HTTP czy wiele więcej inny danych.

Literatura

[1] L.P. Barros, F. Medeiros, E. Moraes, A. F. Júnior, Analyzing the Performance of Apps Developed by using Cross-Platform and Native Technologies, SEKE (2020) 186-191.

[2] J. Jagiełło, Performance comparison between React Native and Flutter, Bachelor thesis Umeå University, Umeå, 2019.

[3] H. Hussain, K. Khan, F. Farooqui, Q. A. Arain, I. F. Siddiqui, Comparative Study of Android Native and Flutter App Development, Memory 47 (2021) 36-37.

[4] E. Gucuoglu, A. B. Ustun, N. Seyhan, Comparison of Flutter and React Native Platforms, Journal of Internet Applications and Management 12 (2) (2021) 129-143, https://doi.org/10.34231/iuyd.888243.

[5] 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 Blekinge Institute of Technology, Karlskrona, 2020.

[6] N. A. Shevtsiv, A. M. Striuk, Cross platform development vs native development, CEUR Workshop Proceedings, 2021.

[7] Opis komendy top w systemie operacyjnym Linux, https://www.man7.org/linux/manpages/man1/top.1.html, [01.06.2022].