Usporedba Flutter i nativne iOS aplikacije

Forjan, Krešimir

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:880006

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

Usporedba Flutter i nativne iOS aplikacije

Diplomski rad

Krešimir Forjan

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 05.05.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Krešimir Forjan
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1119R, 13.10.2020.
OIB studenta:	35120480601
Mentor:	izv. prof. dr. sc. Mirko Köhler
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 1:	izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 2:	Miljenko Švarcmajer, mag. ing. comp.
Naslov diplomskog rada:	Usporedba Flutter i nativne iOS aplikacije
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	STUDENT: Forjan Krešimir U diplomskom radu potrebno je usporediti aplikacije razvijene u dva različita razvojna okvira. Aplikacije trebaju biti napisane pomoću Flutter-a i jednog od nativnih iOS rješenja. Potrebno je provesti vlastite i standardizirane testove, a rezultate prikazati u radu.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	05.05.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 30.05.2023.

Ime i prezime studenta:	Krešimir Forjan
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1119R, 13.10.2020.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Usporedba Flutter i nativne iOS aplikacije**

izrađen pod vodstvom mentora izv. prof. dr. sc. Mirko Köhler

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

S neprekidnim razvojem mobilnih uređaja i njihovom sve većom integracijom u svakodnevni život, kvaliteta mobilnih aplikacija postaje sve važnija. Zbog velikog broja aplikacija koje su dostupne korisnicima, važnu ulogu pri odabiru aplikacije koju će koristiti ima kvaliteta. Osim kvalitete, veliku važnost imaju brzina i jednostavnost samog procesa izrade aplikacije.

Brze promjene i inovacije u tehnološkom svijetu svakim danom otvaraju vrata novim idejama i mogućnostima kod razvoja aplikacija. U takvom okruženju, najveću prednost imaju pojedinci i tvrtke koje se uspijevaju prilagoditi novim trendovima i brzo realiziraju svoje ideje. Velik broj platformi dostupnih krajnjim korisnicima negativno je utjecao na vrijeme razvoja aplikacija pa se pojavila potreba za alatom koji će ubrzati taj proces. U posljednjem desetljeću pojavilo se nekoliko tehnologija koje pružaju mogućnost izrade aplikacija koje rade na više platformi, kao što su React Native, Xamarin, Ionic i Flutter. Takve tehnologije nastoje ubrzati razvoj aplikacija uz istovremeno očuvanje kvalitete i performansi.

Jedna od novijih tehnologija u ovom području je Flutter, koji se od svog objavljivanja pokazao kao vrlo kvalitetan alat i stekao veliku popularnost. Cilj ovog diplomskog rada je analizirati i usporediti Flutter aplikaciju s nativnom iOS aplikacijom, kako bi se utvrdile razlike u performansi i karakteristike oba pristupa. U sklopu rada, razvijena je aplikacija s jednakim funkcionalnostima za oba pristupa, kako bi se omogućila direktna usporedba i analiza performansi.

U drugom poglavlju analizirano je trenutno stanje nativnog iOS i Flutter pristupa u profesionalnom okruženju i navedeni su primjeri aplikacija za svaki pristup. Treće poglavlje daje detaljan uvid u tehnologije korištene za razvoj i ispitivanje aplikacija. U četvrtom poglavlju navedeni su zahtjevi koje aplikacije moraju ispunjavati i objašnjeni su najvažniji dijelovi programskog rješenja. Peto poglavlje uspoređuje oba pristupa razvoju na temelju njihovih općih karakteristika. U šestom poglavlju prikazani su rezultati izvršenih mjerenja performansi. Posljednje poglavlje sadrži zaključak.

1.1. Zadatak diplomskog rada

U diplomskom radu potrebno je usporediti aplikacije razvijene u dva različita razvojna okvira. Aplikacije trebaju biti napisane pomoću Flutter-a i jednog od nativnih iOS rješenja. Potrebno je provesti vlastite i standardizirane testove, a rezultate prikazati u radu.

2. PREGLED PODRUČJA TEME

U ovom poglavlju objašnjeni su nativni i višeplatformski pristup razvoju mobilnih aplikacija, uz navođenje prednosti i nedostataka oba pristupa te obrazloženja kada je koji pristup poželjniji. Pruža se uvid u stanje tržišta pomoću analize interesa zajednice kroz vrijeme i kroz primjere aplikacija za oba pristupa razvoju.

2.1. Nativni i višeplatformski pristup razvoju mobilnih aplikacija

Izrada aplikacije za određenu platformu koristeći tehnologije i alate koje platforma nudi naziva se nativni razvoj. Takvi alati su maksimalno optimizirani za platformu kojoj su namijenjeni pa zbog toga ovaj pristup osigurava optimalno korisničko iskustvo i maksimalne performanse. Jedna od stavki koja daje prednost nativnim aplikacijama kad su u pitanju performanse je direktan pristup hardverskim i softverskim mogućnostima platforme, dok višeplatformske aplikacije moraju izvršavati dodatne zadatke i međukorake da bi pristupile istim mogućnostima platforme. Osim performansi, nativne aplikacije su uspješnije u stvaranju korisničkog iskustva i sučelja koje se u potpunosti slaže sa smjernicama dizajna za određenu platformu i tako osigurava da će se izgled i ponašanje aplikacije uklopiti s ostatkom značajki na platformi [1].

Uz sve navedene prednosti, nativni razvoj ima svoje nedostatke. Razvoj aplikacija zahtijeva odvojeni tim za svaku platformu što znatno povećava troškove i vrijeme razvoja. Kod je razdvojen za svaku platformu, što povećava vjerojatnost za nedosljednostima i razlikama među aplikacijama.

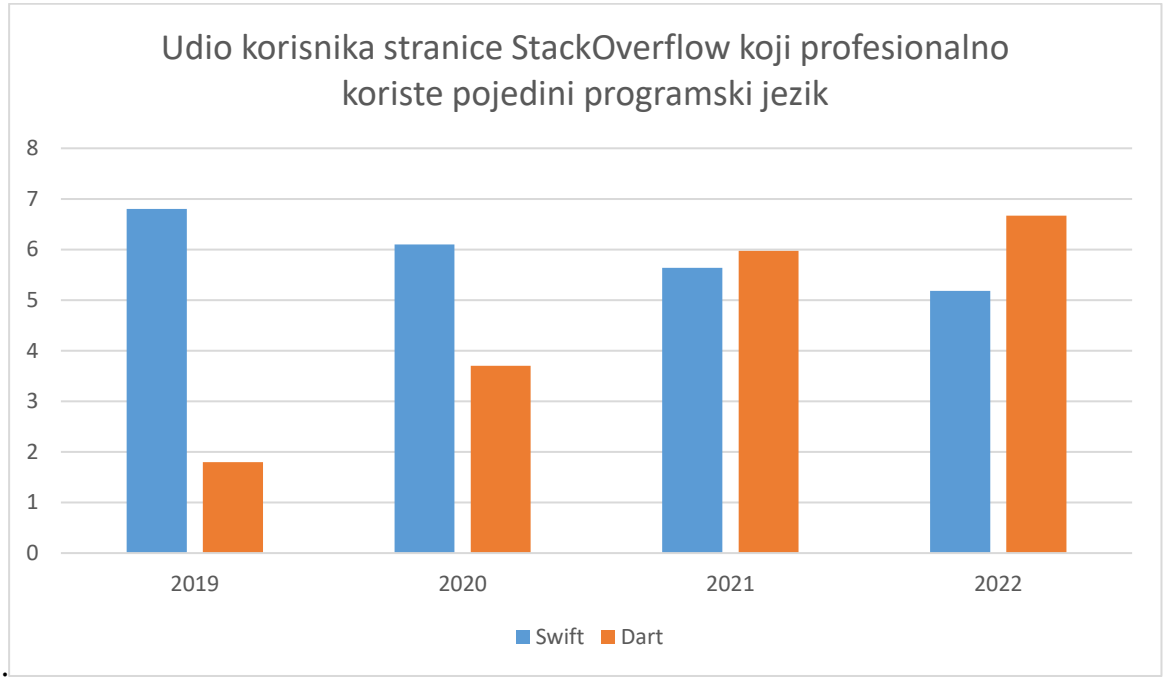
Višeplatformski razvoj rješava problem nativnog razvoja i zahtijeva manje vremena i resursa za proces izrade aplikacije, ali istovremeno negativno utječe na performanse. Razlike u performansi često su minimalne i neprimjetne korisnicima, a stvarna razlika bit će prikazana u nastavku rada.

Kod odabira pristupa važno je uzeti u obzir zahtjeve aplikacije. Ako je prioritet postići optimalne performanse i pružiti korisniku vrlo prilagođeno iskustvo, preporučeni izbor bio bi razvoj nativne aplikacije. Takav prioritet najčešće imaju grafički intenzivne aplikacije, aplikacije s vrlo specifičnim funkcionalnostima koje zahtijevaju visoku podršku alata jedinstvenih za platformu i aplikacije kojima je cilj imati fokus na jednu platformu. Višeplatformski razvoj preporučeni je izbor u slučaju da su brzina razvoja i smanjenje troškova veći prioritet od maksimalnih performansi i prilagođenosti aplikacije, jer jedan tim može u isto vrijeme razvijati aplikaciju za veći broj platformi. Ovaj pristup može pojednostaviti razvoj aplikacija kojima je važna distribucija na većem broju platformi i aplikacija s jednostavnim zahtjevima.

2.2. Stanje tržišta

Postoje različiti podatci koji bi mogli ukazati na učestalost korištenja i popularnost alata za razvoj aplikacija. Jedan od pouzdanijih pokazatelja su podatci prikupljeni anketama na platformi StackOverflow. Ankete se provode svake godine i korisnici, koji su u velikoj većini razvojni programeri, odgovaraju na niz pitanja o tehnologijama koje koriste ili žele koristiti u budućnosti.

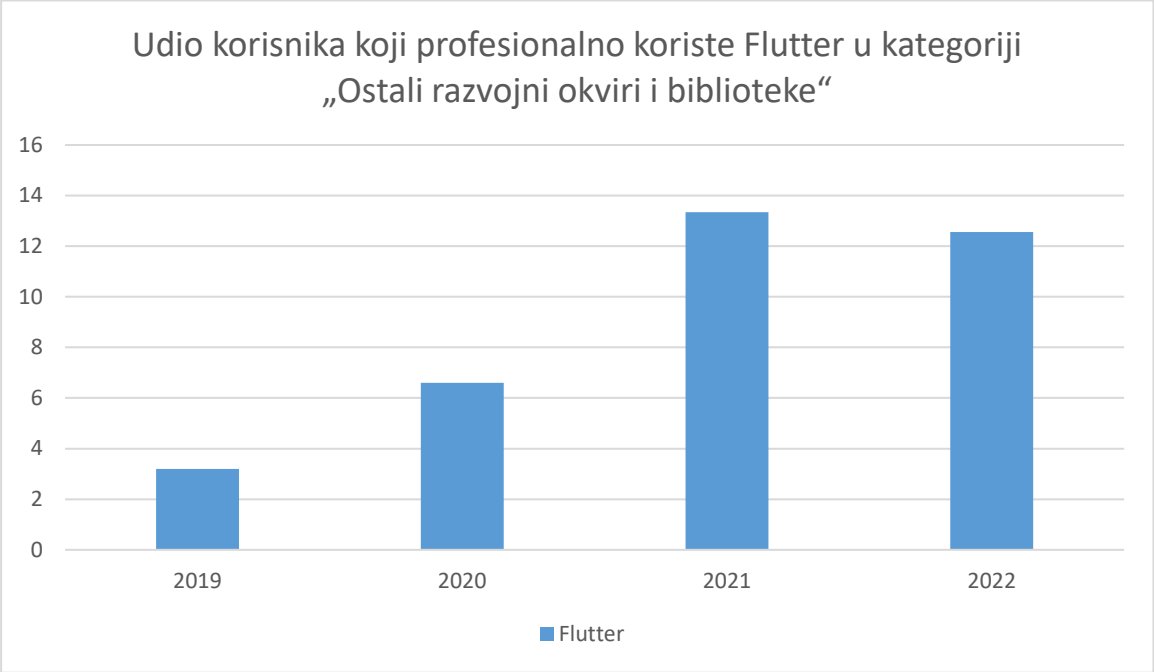
Slika 2.1. prikazuje udio sudionika u anketi na StackOverflowu koji koriste Swift i Dart u profesionalnom okruženju. Zastupljenost Swifta, primarnog jezika korištenog za razvoj nativnih iOS aplikacija, dobar je indikator popularnosti razvoja nativnih aplikacija za iOS, dok zastupljenost Darta, primarnog jezika za razvoj Flutter aplikacija, pokazuje približan broj korisnika Fluttera. Iz rezultata ankete moguće je zaključiti da je broj pojedinaca kojima je primarna profesionalna platforma Flutter naglo porastao kroz prve dvije godine postojanja Flutter, dok se posljednje godine provođenja ankete taj broj stabilizirao. Broj korisnika Swifta i nativnih iOS tehnologija za razvoj ostao je stabilan, s vrlo blagim padom u broju profesionalnih korisnika [2].



Slika 2.2. Udio korisnika koji profesionalno koriste pojedini programski jezik [2]

Porast broja korisnika Fluttera u profesionalnom okruženju može potvrditi i slika 2.2. Graf prikazuje postotak sudionika u anketama StackOverflowa koji profesionalno koriste Flutter kao

okvir za razvoj aplikacija. Moguće je primijetiti da se porast korisnika alata Flutter i programskog jezika Dart približno podudara svake godine [2].



Slika 2.3. Udio korisnika koji koriste Flutter [2]

Prikazani podatci ukazuju na pomake u prioritetima tvrtki i pojedinaca kod razvoja mobilnih aplikacija. Napredak u kvaliteti višeplatformskih rješenja uzrokovao je nagli porast popularnosti Fluttera, dok je nativni iOS razvoj imao konstantan broj korisnika. Moguće je predvidjeti da će nativni iOS razvoj ostati relevantan i imati jednolik broj korisnika kao do sada, dok je promjene u relevantnosti Fluttera teško prognozirati zbog brzih promjena na području višeplatformskih tehnologija i čestog objavljivanja novih alata.

2.3. Primjer nativne iOS aplikacije – Apple Wallet

Mobilna aplikacija Apple Wallet služi za spremanje, upravljanje i korištenje digitalnih kartica za plaćanje, članskih kartica, ulaznica i drugih dokumenata. Tvrtka Apple razvija ju od 2012. godine, no aplikacija nosi naziv Apple Wallet tek od 2015. godine, kada je naziv promijenjen iz Apple Passbook [3]. Aplikacija je dostupna ekskluzivno za iOS platformu te je zbog toga napisana kao nativna iOS aplikacija kao i sve ostale ekskluzivne aplikacije koje Apple razvija za uređaje. Apple wallet ima odličnu integraciju ostalih usluga kao što je Apple Pay i sistemskih funkcija kao što su NFC tehnologija i biometrijska autentifikacija.

2.4. Primjer Flutter aplikacije – Google Pay

Google Pay je mobilna aplikacija tvrtke Google za plaćanje, praćenje troškova i upravljanje digitalnim verzijama kartica za plaćanje i ostalim dokumentima. Aplikaciju koristi više od 100 milijuna korisnika u više zemalja. Google Pay je prvotno napisan u programskom jeziku Kotlin kao nativna Android aplikacija i jeziku Swift kao nativna iOS aplikacija. 2019. godine tim zadužen za razvoj aplikacije odlučio je da će početi koristiti Flutter kao glavni alat za daljnji razvoj. Nakon prijelaza s nativnog pristupa razvoju na Flutter, Google Pay tim objavio je da su smanjili broj linija koda za 35% i napor inženjera za 70% [4].

3. TEHNOLOGIJE ZA RAZVOJ I TESTIRANJE MOBILNIH APLIKACIJA

Ovo poglavlje daje uvid u tehnologije korištene za izradu i ispitivanje aplikacije. Prikazane su njihove glavne funkcionalnosti, način na koji funkcioniraju i što sve koriste za uspješno izvođenje svog zadatka.

3.1. iOS nativne razvojne tehnologije

Nativnim iOS razvojem smatra se razvoj aplikacija koje su namijenjene specifično uređajima koji koriste iOS kao operacijski sustav. U takve uređaje spadaju iPhone, iPad i drugi proizvodi tvrtke Apple. Primarni programski jezik koji se koristi je Swift, prije kojeg Objectiv-C bio jedini jezik za razvoj.

Službeno integrirano razvojno okruženje za razvoj nativnih iOS aplikacija je Xcode. Osim za iOS, Xcode podržava i razvoj za ostale operacijske sustave tvrtke Apple, kao što su macOS, watchOS i tvOS [5]. Kao i većina modernih integriranih razvojnih okruženja, Xcode nudi razne alate koji pomažu kod razvoja. Među tim alatima su alati za praćenje performansi, alati za uređivanje koda i alati za uređivanje korisničkog sučelja.

3.1.1. Izvršavanje koda i pristup platformi

Kod razvoja nativnih iOS aplikacija Low Level Virtual Machine koristi se kao prevoditelj u strojni kod [6]. LLVM prevoditelj omogućuje optimizaciju koda za različite arhitekture procesora, kao što su ARM i x86, koje se koriste u Appleovim uređajima.

Renderiranje grafičkog sučelja izvodi se pomoću OpenGL ES tehnologije i Metal tehnologije [7]. OpenGL ES i Metal pružaju sučelje za renderiranje 2D i 3D grafike. Metal, koji je uveden 2014. godine, posebno je optimiziran za Appleove hardverske platforme, što rezultira dodatnim poboljšanjem performansi u odnosu na OpenGL ES.

Pristup nativnim funkcionalnostima platforme, poput kamere i GPS servisa, omogućen je kroz iOS razvojni set alata koji dolazi s Xcode integriranim razvojnim okruženjem. Razvoj nativnih iOS aplikacija omogućuje korištenje Appleovih platformskih tehnologija, poput Core Data, Core Graphics, Core Animation i mnogih drugih. Ove tehnologije nude funkcionalnosti koje olakšavaju razvoj i poboljšavaju performanse aplikacija na iOS platformi.

3.1.2. Swift programski jezik

Swift je objektno orijentirani programski jezik otvorenog koda koji je objavila tvrtka Apple 2014. godine. Stvoren je kao zamjena za Objectiv-C, koji je do tada bio korišten za razvoj na Appleovim platformama. Na Appleovim platformama može biti pokrenut unutar istog programa s C, C++ i Objective-C jezicima, zahvaljujući Objective-C runtime bibliotekama.[8]

3.1.3. SwftUI i izrada korisničkog sučelja

SwiftUI je moderni okvir za izradu korisničkih sučelja na Appleovim platformama. Uveden je 2019. godine kao zamjena za UIKit. SwiftUI nudi deklarativni pristup izradi korisničkog sučelja, što znači da se sučelje opisuje kroz niz deklaracija o izgledu i ponašanju komponenti.

SwiftUI se temelji na Swift jeziku i pruža jednostavan način za izradu korisničkog sučelja s poboljšanom čitljivošću koda, umjesto kompleksne hijerarhije klasa i objekata koja se koristi u UIKitu [9].

3.2. Flutter

Flutter je alat otvorenog koda za razvoj višeplatformskih aplikacija visoke kvalitete i performansi koji koristi Dart kao programski jezik [10]. Razvila ga je tvrtka Google je s ciljem stvaranja univerzalnog alata za razvoj aplikacija na velikom broju platforma kao što su Android, iOS, Windows, Linux, macOS i web aplikacije unutar internetskog preglednika. Prva stabilna verzija objavljena je 2018. godine, ali podrška za razvoj se smatrala stabilnom samo za Android i iOS, a za ostale platforme je postojala podrška u ranoj fazi, što nije bilo dovoljno za razvoj produkcijskih aplikacija [11]. Kroz naredne godine Google je nastavio s redovnim održavanjem i nadogradnjom te je u ožujku 2021. godine objavljen Flutter 2.0. koji je pružao stabilnu podršku za razvoj web aplikacija unutar internetskih preglednika. U svibnju 2022. godine objavljen je Flutter 3.0 koji je stabilnu podršku proširio na Windows, macOS i Linux aplikacije te time obuhvatio svih 6 najučestalijih platformi [12].

Preuzimanje i postavljanje Flutter SDK na osobno računalo daje mogućnost korištenje alata i biblioteka za višeplatformski razvoj pomoću Fluttera. Flutter SDK dostupan je na operacijskim sustavima: Windows, macOS, Linux i ChromeOS.

Uz Flutter SDK, koriste se različita integrirana razvojna okruženja za pisanje i testiranje aplikacija. Najčešći izbori za integrirana razvojna okruženja su: Android Studio, Visual Studio Code, Xcode i IntelliJ IDEA. Takva okruženja čine razvoj jednostavnijim, tako što nude razne ekstenzije kompatibilne s Flutterom i značajke koje olakšavaju pisanje koda.

3.2.1. Izvršavanje koda i pristup platformi

Izvođenje koda za vrijeme razvoja aplikacije u Flutteru vrši se pomoću Dart Virtual Machine tehnologije. Dart Virtual Machine omogućuje brzo i često ažuriranje korisničkog sučelja i logike na razvojnoj aplikaciji, bez potrebe za prevođenjem cjelokupnog Dart koda u strojni jezik. Ova mogućnost znatno smanjuje vrijeme utrošeno na nepotrebno čekanje kod razvoja.

U slučaju kreiranja produkcijske aplikacije za iOS platformu, Flutter koristi Low Level Virtual Machine (LLVM) prevoditelj za generiranje ARM koda koji se pokreće izravno na uređaju [13].

Grafičko korisničko sučelje generira 2D grafička biblioteka Skia. Skia je biblioteka otvorenog koda koja pruža sučelja za rad s grafikom na velikom broju platformi. Od 2005. godine razvija ju Google i koristi ju kao glavni grafički pogon za razne projekte među kojima su, osim Fluttera, Google Chrome, ChromeOS i Android [14].

Iako Flutter koristi Dart kod za izradu većinu značajki, ponekad je potreban pristup nativnim funkcionalnostima platforme, kao što su senzori, GPS i slične značajke. Na iOS platformi, za takve slučajeve koriste se platformski kanali za dvosmjernu komunikaciju Dart koda i nativnog Swift ili Objectiv-C koda.

3.2.2. Dart programski jezik

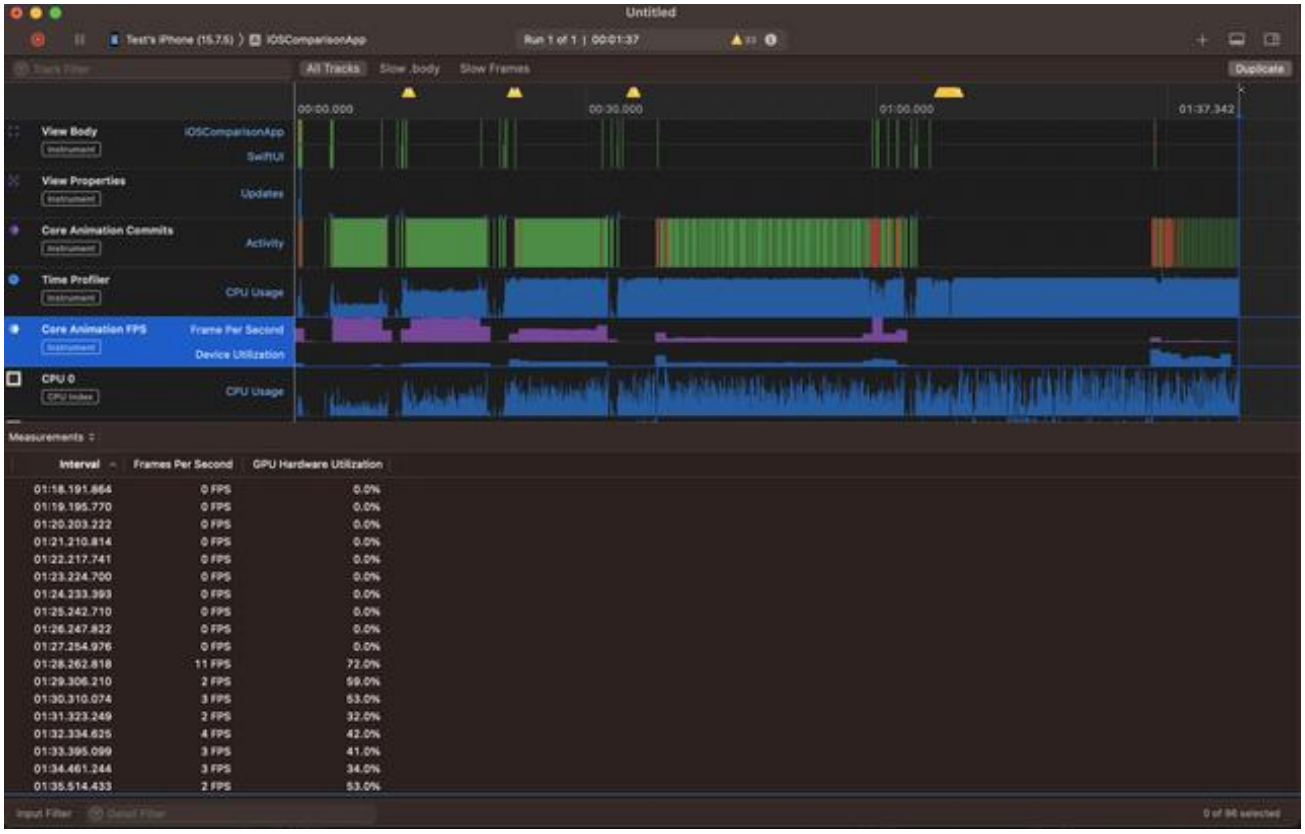
Dart je objektno orijentirani jezik sa sintaksom sličnom programskom jeziku C. Optimiziran je za rad na klijentskoj strani aplikacije i vrlo efikasan za izradu korisničkog sučelja. Dart je prvi put predstavljen 2011. godine na konferenciji GOTO u Danskoj [15]. Najčešće se koristi za razvoj višeplatformskih aplikacija u Flutteru, ali zbog mogućnosti za prevođenjem u JavaScript, moguća je i primjena u web razvoju.

3.2.3. Flutter widgeti i izrada korisničkog sučelja

Za izradu korisničkog sučelja Flutter koristi koncept „*widgeta*“ koji su temeljni blokovi grafičkog korisničkog sučelja. Slažu se hijerarhijski u strukturu drveta. Svaki element korisničkog sučelja u Flutteru je „*widget*“. Kreiraju se pomoću Dart jezika te nije potreban poseban odvojeni jezik za strukturiranje aplikacije. Flutter ima bogat skup prethodno izrađenih elemenata korisničkog sučelja koji se vrlo lako implementiraj, a izgledom potpuno poštuju smjernice nativnih platformi za koje su namijenjeni.

3.3. Xcode Instruments

Xcode Instruments su alati koji dolaze uz Xcode integrirano razvojno okruženje. Ovi alati dizajnirani su za analizu performansi na iOS, macOS, watchOS i tvOS uređajima. Xcode Instruments ima brojne mogućnosti za analizu i ispitivanja, kao što su profiliranje performansi, upravljanje memorijom, analiza energetske učinkovitosti, analiza grafičkih performansi i detekcija mrežnih problema [16]. Slika 3.1. prikazuje alat za profiliranje performansi.



Slika 3.1. Xcode Instruments alat za profiliranje

4. DIZAJN I RAZVOJ TESTNE APLIKACIJE

U ovom poglavlju prikazani su zahtjevi koje testne aplikacije moraju ispunjavati da bi se omogućila objektivna i ispravna usporedba performansi. Objašnjeni su najvažniji dijelovi koda za izvođenje danih zahtjeva. Prikazan je izvorni kod za iste funkcionalnosti kod oba pristupa, što pomaže u isticanju potencijalnih razlika u logici izvornog koda i olakšava razumijevanje načina implementaciji zahtjeva u oba slučaja.

4.1. Definiranje zahtjeva testnih aplikacija

Prije početka razvoja testnih aplikacija, potrebno je definirati zahtjeve koje će obje aplikacije morati ispuniti. Definicijom zahtjeva osigurava se točnost rezultata usporedbe i analize, jer će obje aplikacije biti razvijene prema istim kriterijima, a to omogućuje objektivnu usporedbu performansi na svakoj platformi.

Testne aplikacije moraju imati jednostavnu navigaciju i strukturu, kako bi pronalazak odgovarajućih testnih zaslona bio što brži i efikasniji. Početni zaslon je prvi zaslon vidljiv pri otvaranju aplikacije. Sastoji se od izbornika s pet poveznica, od kojih svaka vodi na zaslon s posebnom testnom značajkom. Svaka poveznica sastoji se od tipke koja korisnika preusmjerava na novi zaslon i naslova testne značajke. U slučaju da značajka ima podesiv intenzitet izvođenja, poveznica uključuje i klizač za određivanje intenziteta. Intenzitet može biti u rasponu od 1 do 5.

Prvi testni zaslon treba omogućiti testiranje skalabilnosti korisničkog sučelja. Ovaj testni zaslon sadrži elemente korisničkog sučelja koji se naizmjence prikazuju. Četiri elementa koja se uzastopno izmjenjuju na prikazu su: gumb, ikona, tekst i polje za unos teksta. Broj elemenata prikazanih na zaslon ovisi o parametru intenziteta. Kako bi se osiguralo testiranje zaslona s vrlo malim i vrlo velikim brojem elemenata korisničkog sučelja, pogodno je eksponencijalno povećavati broj elemenata s linearnim rastom intenziteta. Za ovaj testni primjer, najefikasnijom se pokazala funkcija:

$$brojElemenata = 7^{intenzitet}$$

Sljedeći zaslon pri pokretanju započinje animaciju rotacije kružnih elementa korisničkog sučelja oko centra rotacije. Broj kružnih elemenata koji su animirani moguće je podesiti pomoću parametra intenziteta. Intenzitet u ovom slučaju povećava broj elemenata koji se rotiraju i tako stvara zahtjevniju i za izvođenje težu animaciju, što olakšava testiranje performansi aplikacije u

većem broju okolnosti. Formula koja je korištena za određivanje broja elemenata korisničkog sučelja u odnosu na intenzitet je:

$$brojElemenata = (10 - intenzitet) * 3^{intenzitet+2}$$

Ova formula osigurava da animacija neće postati prezahtjevna za izvođenje, a u isto vrijeme osigurava da će promjena intenziteta izazvati vidne promjene u performansi.

Također, potrebno je implementirati zaslon koji će izvoditi matematičke operacije i mjeriti vrijeme potrebno za njihovo izvođenje. Ovaj zaslon sadrži tipku koja pokreće izvođenje operacija zbrajanja, dijeljenja i množenja. Kao i u dosadašnjim primjerima, broj operacija koje će se izvesti ovisi o intenzitetu. Formula koja se koristi za određivanje broja operacija koje će se izvesti, glasi:

$$brojOperacija = 3 * 10^{intenzitet} * 10000$$

Zbrajanje, dijeljenje i množenje izvode se u svakom koraku zadatka te se je zato na početku formule koeficijent 3. Ostatak formule prikazuje koliko će iteracija zadatka biti izvedeno. Kod ovog testnog primjera intenzitet igra veliku ulogu, jer je zadatak vrlo jednostavan, ali maksimalan broj od tri milijarde iteracija ovaj zadatak čine zahtjevnim.

Zaslon za upravljanje lokalnom bazom podataka je još jedan zaslon koji je potrebno implementirati. Služi za mjerenje performansi aplikacije kod pristupa lokalnoj memoriji. Potrebno je koristiti istu vrstu lokalne baze podataka zbog ispravnosti mjerenja te je zato odabrana baza podataka koja se temelji na SQLite biblioteci. Ovaj zaslon treba pružiti opciju pisanja i čitanja iz baze podataka te opciju brisanja svih podataka kako bi se omogućilo testiranje na praznoj bazi podataka kod izvođenja mjerenja. Količina podataka koji se unose i čitaju iz baze podataka ovise o parametru intenziteta. Intenzitet određuje količinu podataka za pisanje i čitanje na temelju formule:

$$brojPodataka = 10^{intenzitet-1}$$

Posljednji ekran bavi se nativnim značajkama platforme poput GPS-a i kamere. Zaslon sadrži dvije tipke od kojih jedna pokreće kameru, a druga dohvaća trenutnu lokaciju korisnika.

4.2. Razvoj nativne iOS aplikacije

Prije početka razvoja značajki unutar aplikacije potrebno je stvoriti novi SwiftUI projekt pomoću izbornika u razvojnom okruženju Xcode. Kreiranjem novog projekta, Xcode u zadani direktorij dodaje sve potrebne datoteke za daljnji razvoj.

4.2.1. Početni zaslon

Kako bi početni zaslon ispunio navedene zahtjeve za jednostavnost potrebno je stvoriti listu poveznica, što je moguće postići dodavanjem elemenata „*ScrollView*“ i „*VStack*“ te unutar njih 5 elemenata „*MainItem*“ čije je korištenje prikazano na slici 4.1. „*MainItem*“ kao parametar prima funkciju „*onStart*“ koja se izvodi pri pokretanju zaslona. Unutar funkcije koja se predaje kao parametar izvodi se kod koji kreira objekt zaslona i predaje mu intenzitet. Zatim se kreira upravljač korisničkog sučelja koji omogućuje suradnju SwiftUI i UIKit pogleda. Ostatak funkcije zadužen je za otvaranje novog prozora koji sadrži novi pogled i animaciju tog prozora.

```
MainItem(title: "Skalabilnost korisničkog sučelja", onStart: { intensity in
    let contentView = ScalabilityView(intensity: Int(pow(7, sliderValueScalability)))
    let hostingController = UIHostingController(rootView: contentView)
    if let windowScene = UIApplication.shared.connectedScenes.first as? UIWindowScene {
        windowScene.windows.first?.rootViewController?.present(
        hostingController,
        animated: true,
        completion: nil)
    }
}, showIntensitySlider: true, intensity: $sliderValueScalability)
.frame(width: max(geometry.size.width - 24, 0))
```

Slika 4.1. SwiftUI implementacija elementa „*MainItem*“

4.2.2. Učitavanje elemenata korisničkog sučelja

Zaslon s ovom značajkom zadužen je za prikaz većeg broja elemenata korisničkog sučelja. Taj zadatak ispunjava funkcija „*buildInterfaceElements*“ sa slike 4.2. Ova funkcija ima pristup parametru intenziteta te zatraženim brojem iteracija u petlji stvara odgovarajući broj elemenata. Kod svake iteracije novi element dodaje se u listu elemenata, koja će kod kasnijeg izvođenja služiti kao referenca na zatražene elemente. Modularnim dijeljenjem trenutnog broja iteracije određuje se element koji će biti dodan u listu elemenata.

```
private func buildInterfaceElements() -> [AnyView] {
        var elements: [AnyView] = []

        for i in 0..<intensity {
            if i % 4 == 0 {
                elements.append(
                    AnyView(
                        Button(action: {}) {
                            Text("Gumb \(i)")···
                        }
                        .padding(4)
                    )
                )
            } else if i % 4 == 1 {···
            } else if i % 4 == 2 {···
            } else {···
            }
        }
        return elements
    }
```

Slika 4.2. SwiftUI implementacija „*buildInterfaceElements*“ funkcije

Da bi se kroz kreirane elemente moglo efektivno listati, lista elemenata se predaje pogledu „*ScrollView*“.

4.2.3. Animiranje korisničkog sučelja

Zaslon na kojem je animirana rotacija elemenata, animaciju postiže tako da petljom iscrtava zatražen broj elemenata, postavlja u zajednički kontejner te im zadaje efekt rotacije metodom „*rotationEffect*“ koja prima parametar „*animationValue*“. Navedeni parametar ima vrijednost na kojem se temelji rotacija. Svaki krug ima posebno postavljenu boju, poziciju, veličinu i rotacijsku vrijednost. Kod koji je odgovoran za animiranje prikazan je na slici 4.3.

```
struct CircleAnimation: View {
    @Binding var animationValue: CGFloat
    let intensity: Int

    var body: some View {
        GeometryReader { geometry in
            ZStack {
                ForEach(0 ..< intensity, id: \.self) { i in
                    let color = i % 2 == 0 ? Color.red : Color.yellow···

                    Circle()
                        .fill(color)
                        .frame(width: 20, height: 20)
                        .offset(x: xPos - geometry.size.width / 2, y: yPos - geometry.size.height / 2)
                        .rotationEffect(.radians(Double(animationValue)),
                        anchor: UnitPoint(x: 0.5, y: 0.5 + 100 / geometry.size.height))
                }
            }
            .frame(width: geometry.size.width, height: geometry.size.height)
        }
    }
}
```

Slika 4.3. SwiftUI implementacija animacije

4.2.4. Izvođenje računskih operacija

Na slici 4.4. prikazan je kod koji se kreće izvoditi na zahtjev korisnika, pritiskom na gumb unutar zaslona za izvođenje računskih operacija. Funkcija „*calculate*“ kao parametar prima željeni intenzitet koji određuje koliko puta će se izvesti operacije unutar petlje. Funkcija za računanje je vrlo jednostavna, no jednostavnost znatno olakšava identičnu implementaciju na obje platforme, što čini usporedbu točnijom i objektivnijom.

```swift
func calculate(_ n: Int) -> Double {
    var temp = 0.0

    let valueToAdd = 2.0
    let divisor = 4.0
    let multiplier = 2.0

    for _ in 0..<n {
        temp += valueToAdd
        temp /= divisor
        temp *= multiplier
    }

    return temp
}
```

Slika 4.4. Swift implementacija funkcije „*calculate*“

4.2.5. Spremanje i čitanje iz lokalne baze podataka

Zaslon za spremanje i čitanje iz lokalne baze podataka zahtjeva njenu implementaciju. Za implementaciju korištena je GRDB baza podataka koja u pozadini koristi SQLite [17]. Kod pokretanja aplikacije stvara se instanca klase „*Database*“ kao singleton. Singleton je razvojni obrazac koji osigurava da će samo jedna instanca klase biti stvorena, što znači da niti jedna dodatna instanca klase „*Database*“ neće biti stvorena nakon pokretanja aplikacije. Klasa baze podataka ima definirane metode za pisanje i čitanje podataka te za postavljanje baze podataka. Unutar tih metoda koriste se funkcije biblioteke GDBR za kreiranje baze podataka, kreiranje tablica, pisanje i čitanje iz baze podataka. Kod za implementaciju navedenih metoda nalazi se na slici 4.5.

```swift
private func setupDatabase() {
    try? dbQueue.write { db in
        try db.create(table: User.databaseTableName, ifNotExists: true) { table in
            table.autoIncrementedPrimaryKey("id")
            table.column("name", .text).notNull()
            table.column("age", .integer).notNull()
        }
    }
}

func saveUser(_ user: User) {
    try? dbQueue.write { db in
        var newUser = user
        newUser.id = nil
        try newUser.insert(db)
    }
}

func getAllUsers() -> [User] {
    return try! dbQueue.read { db in
        return try User.fetchAll(db)
    }
}
```

Slika 4.5. Swift implementacija baze podataka

4.2.6. Korištenje nativnih značajki

Za pristup nativnim značajkama unutar razvojnog okruženja Xcode potrebno je dodati dopuštenja za željene nativne značajke. *Info.plist* je datoteka unutar koje je moguće pronaći sva dopuštenja koja aplikacija zahtjeva od korisnika za uspješno izvođenje. Nakon dodjeljivanja dopuštenja, korištenje nativnih značajki unutar nativne iOS aplikacije je vrlo jednostavno jer nisu potrebne dodatne biblioteke i alati, kao što je to slučaj kod višeplatformskih alata. Na slici 4.6. prikazan je kod koji je dovoljan za pristup lokaciji i kameri iz nativne iOS aplikacije.

```swift
locationManager.requestLocation()

let picker = UIImagePickerController()
picker.sourceType = .camera
picker.delegate = context.coordinator
```

Slika 4.6. Swift implementacija GPS i kamera servisa

4.3. Razvoj Flutter aplikacije

Za početak rada s alatom Flutter potrebno je započeti novi projekt na željenoj lokaciji naredbom u „*flutter create*“. Ova naredba kreira direktorije i datoteke koje su Flutteru potrebne za rad. Aplikacija započinje u main.dart datoteci te se grana kroz hijerarhijsko stablo elemenata korisničkog sučelja.

4.3.1. Početni zaslon

Početni zaslon izgleda i ponaša se isto kao SwiftUI verzija zaslona. Na slici 4.7. prikazan je kod za navigaciju na drugi zaslon. Kod je puno jednostavniji nego SwiftUI verzija istog koda. To je zbog jednostavnosti navigacije u Flutteru. Za navigaciju na drugi zaslon dovoljan je jedan poziv funkcije „*push*“ kojoj se kao parametar predaje željeni zaslon. Zaslon kao parametar prima intenzitet kojim korisnik upravlja putem klizača unutar „*MainScreenItem*“ elementa. Kod promjene vrijednosti klizača varijabla „*value*“ mijenja svoju vrijednost i time direktno utječe na intenzitet koji će biti proslijeđen zaslonu.

```
MainScreenItem(
  title: 'Skalabilnost korisničkog sučelja',
  onStart: (value) {
    Navigator.of(context).push(
      CupertinoPageRoute(
        builder: (context) => ScalabilityScreen(
          intensity: pow(7, value).toInt(),
        ),
      ),
    );
  },
),
```

Slika 4.7. Flutter implementacija elementa „*MainItem*“

4.3.2. Učitavanje elemenata korisničkog sučelja

Kod Flutter verzije zaslona za učitavanje korisničkog sučelja logika kreiranja liste elemenata ostaje potpuno ista kao i u SwiftUI verziji. U listu „elements“ dodaju se novi elementi svakom iteracijom petlje. „*CupertinoButton*“ korišten je kao alternativa za „*Button*“ element na SwiftUI aplikaciji. Flutter ima dva stila za svoje elemente korisničkog sučelja: Cupertino i Material. Cupertino je stil prilagođen nativnom iOS okruženju, dok je Material prilagođen Android okruženju te su zbog toga

svi elementi specifični za iOS označeni prefiksom „*Cupertino*“. Slika 4.8. prikazuje kod za implementaciju zaslona za učitavanje elemenata u Flutteru.

```
List<Widget> _buildInterfaceElements() {
  List<Widget> elements = [];
  for (int i = 0; i < widget.intensity; i++) {
    if (i % 4 == 0) {
      elements.add(
        Container(
          margin: const EdgeInsets.all(4.0),
          child: CupertinoButton(
            padding: const EdgeInsets.symmetric(horizontal: 12),
            onPressed: () {},
            color: CupertinoColors.activeBlue,
            child: Text('Gumb $i'),
          ),
        ),
      );
    } else if (i % 4 == 1) {…
    } else if (i % 4 == 2) {…
    } else {…
    }
  }
  return elements;
}
```

Slika 4.8. Flutter implementacija „*buildInterfaceElements*“ funkcije

4.3.3. Animiranje korisničkog sučelja

Postavljanje elemenata korisničkog sučelja koji će biti animirani na zaslon u Flutter aplikaciji ima vrlo sličan pristup kao i SwiftUI inačica. Elementi se iterativno generiraju i pozicioniraju na zaslon. Razlika se primjećuje tek kod načina na koji se elementi animiraju. Flutter nudi dodatni element „*AnimatedBuilder*“ koji olakšava rad s animacijama. Kao parametre prima animaciju koju će element izvoditi i sam element koji će biti animiran. Primjer implementacije „AnimatedBuilder“ elementa prikazan je na slici 4.9.

```
Stack(
  children: List.generate(widget.intensity, (i) {
    final angle = 2 * pi / widget.intensity * i;
    final xPos = 100 * cos(angle);
    final yPos = 100 * sin(angle);

    return AnimatedBuilder(
      animation: _animations[i],
      builder: (BuildContext context, Widget? child) {
        return Transform.rotate(
          angle: _animations[i].value,
          child: Transform.translate(
            offset: Offset(xPos, yPos),
            child: child,
          ),
        );
      },
      child: CircleAvatar(
        backgroundColor: i % 2 == 0 ? Colors.red : Colors.yellow,
        radius: 10,
      ),
    );
  }),
)
```

Slika 4.9. Flutter implementacija elementa „*AnimationBuilder*“

„*AnimatedBuilder*“ kao jedan od parametara prima animaciju. U ovom primjeru predana mu je animacija iz liste koja se kreira pri izvođenju „*initState*“ funkcije. „*initState*“ se izvodi na početku životnog ciklusa Flutter elementa koji ima promjenjivo unutarnje stanje. Elementi s promjenjivim unutarnjim stanjem u mogućnosti su prilagoditi svoj izgled trenutnom unutarnjem stanju i u trenutku promjene stanja prilagoditi svoj izgled novom stanju. Kod inicijalizacije stanja ovog elementa, kreiraju se „*Tween*“ objekti koji sadrže informacije o početnom i krajnjem stanju jedne animacije. Pozivanjem „*animate*“ metode nad objektom „*Tween*“, kao rezultat dobije se objekt tipa „*Animation*“ koji sadrži podatke o animaciji između početnog i krajnjeg položaja bivšeg „*Tween*“ objekta. Kod za inicijalizaciju stanja i kreiranje animacija u Flutteru vidljiv je na slici 4.10.

```dart
@override
  void initState() {
    super.initState();
    for (int i = 0; i < widget.intensity; i++) {
      final animationController = AnimationController(
        duration: const Duration(milliseconds: 2000),
        vsync: this,
      )..repeat();
      final animation =
          Tween<double>(begin: 0, end: 2 * pi).animate(animationController);

      _animationControllers.add(animationController);
      _animations.add(animation);
    }
  }
```

Slika 4.10. Flutter implementacija animacije

4.3.4. Izvođenje računskih operacija

Implementacija funkcije za izvođenje računskih operacija u Flutter inačici aplikacije je gotovo identična implementaciji u SwiftUI inačici. Za to je zaslužna jednostavnost implementacije. Na slici 4.11. prikazan je kod za implementaciju funkcije „*calculate*“, koja vrši matematičke operacije na za to previđenom zaslonu.

```dart
double _calculate(int n) {
    double temp = 0;
    for (int i = 0; i < n; i++) {
      temp = temp + 2;
      temp = temp / 4;
      temp = temp * 2;
    }
    return temp;
  }
```

Slika 4.11. Dart implementacija funkcije „*calculate*“

4.3.5. Spremanje i čitanje iz lokalne baze podataka

Kod implementacije baze podataka u Flutter inačici aplikacije bilo je vrlo važno koristiti bazu podataka baziranu na SQLite, kako bi na usporedbu performansi utjecalo što manje vanjskih faktora. Korištena je biblioteka Moor. Moor je biblioteka koja generacijom koda i pružanjem gotovih funkcija za pristup lokalnoj memoriji pojednostavljuje implementaciju baza podataka [18]. Na slici 4.12. nalazi se implementacije metoda za pristup i mijenjanje podataka te konstruktor baze podataka koji osim kreiranja objekta, služi i za inicijalizaciju u lokalnoj memoriji.

```
@UseMoor(tables: [Users])
class Database extends _$Database {
  Database()
      : super(
          FlutterQueryExecutor.inDatabaseFolder(
            path: 'db.sqlite',
            logStatements: true,
          ),
        );

  @override
  int get schemaVersion => 1;

  Future<List<User>> getAllUsers() => select(users).get();

  Future<void> saveUser(UsersCompanion user) {
    return into(users).insert(user);
  }
```

Slika 4.12. Dart implementacija baze podataka

4.3.6. Korištenje nativnih značajki

Nativnim značajkama iz Flutter aplikacije pristup je omogućen uz dodatne biblioteke. Za pristup GPS podatcima, korištena je biblioteka „*Geolocator*“, a za pristup kameri „*Image picker*“. Nakon njihovog dodavanja među zavisnosti projekta, potrebno je, kao i za SwiftUI, unutar *Info.plist* datoteke zatražiti dopuštenja koja aplikacija zahtijeva za rad. Kod na slici 4.13. dovoljan je za pokretanje nativnih značajki uz sve navedene korake.

```
Geolocator.getCurrentPosition();
ImagePicker().pickImage(source: ImageSource.camera);
```

Slika 4.13. Flutter implementacija GPS i kamera servisa

5. OPĆA USPOREDBA NATIVNOG iOS I FLUTTER RAZVOJA

U ovom poglavlju uspoređuju se opće karakteristike razvoja nativnih iOS i Flutter aplikacija. Uspoređene su neke od najvažnijih karakteristika pri odabiru tehnologije koja će biti korištena za izvođenje projekta.

5.1. Održivost i skalabilnost

Prilikom odabira tehnologije za razvoj aplikacija, ključni faktori koje treba razmotriti su održivost i skalabilnost. Održivost se odnosi na sposobnost ažuriranja i unapređivanja aplikacije tijekom vremena, dok skalabilnost znači mogućnost rasta aplikacije kako bi zadovoljila potrebe korisnika i poslovnih zahtjeva.

Nativni iOS razvoj nudi visoku razinu održivosti zahvaljujući čvrstoj podršci koju Apple pruža razvojnim programerima i njihovim alatima. Korištenjem nativnih alata, kao što su Swift i UIKit, programeri mogu iskoristiti sve prednosti iOS operativnog sustava i njegovih ažuriranja, što omogućuje jednostavno održavanje i poboljšanje aplikacije s vremenom. Kada je riječ o skalabilnosti, nativne iOS aplikacije mogu se prilagoditi promjenama u poslovanju ili povećanom broju korisnika, ali mogu zahtijevati dodatno ulaganje vremena i resursa kako bi se osigurala usklađenost s drugim platformama.

S druge strane, Flutter nudi solidnu održivost zahvaljujući jedinstvenom kodu koji se može koristiti na različitim platformama. Programeri koji koriste Flutter mogu se osloniti na Dart jezik i bogat set elemenata korisničkog sučelja kako bi brzo razvili i održavali svoje aplikacije. Iako je Flutter relativno nova tehnologija, Google ulaže napore kako bi poboljšao platformu i osigurao njenu održivost u budućnosti. U pogledu skalabilnosti, Flutter omogućuje jednostavan rast aplikacija na više platforma, smanjujući potrebu za dodatnim resursima koji bi inače bili potrebni za zasebni razvoj aplikacija za iOS i Android.

5.2. Vrijeme i cijena razvoja

Prije početka razvoja aplikacije, ključna pitanja koja je potrebno razmotriti su vrijeme i cijena razvoja. Na ove faktore značajno utječe opseg projekta.

Ako je opseg projekta samo na razvoj nativne iOS aplikacije, razlika u vremenu i cijeni između nativnog pristupa i Fluttera možda neće biti značajna. Međutim, ako projekt uključuje razvoj na drugim platformama, poput Androida, Flutter je u prednosti u pogledu smanjenju vremena i troškova razvoja zahvaljujući jedinstvenom kodu za sve platforme [10].

Također, Flutter ima efikasan i jednostavan pristup izradi korisničkog sučelja, što rezultira bržom izradom koncepta i minimalnog održivog proizvoda, a klijenti dobivaju brz uvid u napredak projekta i mogućnost testiranja funkcionalnosti u ranoj fazi razvoja.

S druge strane, nativni razvoj zahtijeva više resursa, što može dovesti do većih troškova i dužeg vremena razvoja. Nativne aplikacije imaju prednost u pogledu performansi i kvalitete, pružajući korisnicima najbolje moguće iskustvo na određenoj platformi. Nativne aplikacije mogu bolje iskoristiti mogućnosti hardvera i operativnog sustava, što rezultira boljom optimizacijom i prilagodbom.

Prema nekim procjenama, Flutter razvoj u prosijeku može smanjiti troškove razvoja za približno 30% u odnosu na razvoj dviju zasebnih nativnih aplikacija [19].

5.3. Zajednica i podrška

Zajednica koja se bavi nativnim iOS razvojem postoji već duže vrijeme i zbog toga je velika i stabilna. Na internetu je dostupan velik broj materijala i savjeta koji pomažu u učenju i razvoju nativnih iOS aplikacija. Ova zajednica je vrlo stabilna zbog činjenice da utjecaj tvrtke Apple kontinuirano raste i to stvara dodatne potrebe za nativnim razvojnim programerima. Uz to, podrška i opširna dokumentacija koju Apple pruža Swift zajednici, dodatno pozitivno utječe na njenu veličinu i stabilnost. Najbolja mjesta za komunikaciju s nativnom iOS zajednicom su Apple Developer forum ili GitHub, gdje se održavaju i razvijaju velik broj biblioteka koje se mogu koristiti u nativnim aplikacijama. Te biblioteke su dostupne na platformama poput CocoaPods i Carthage.

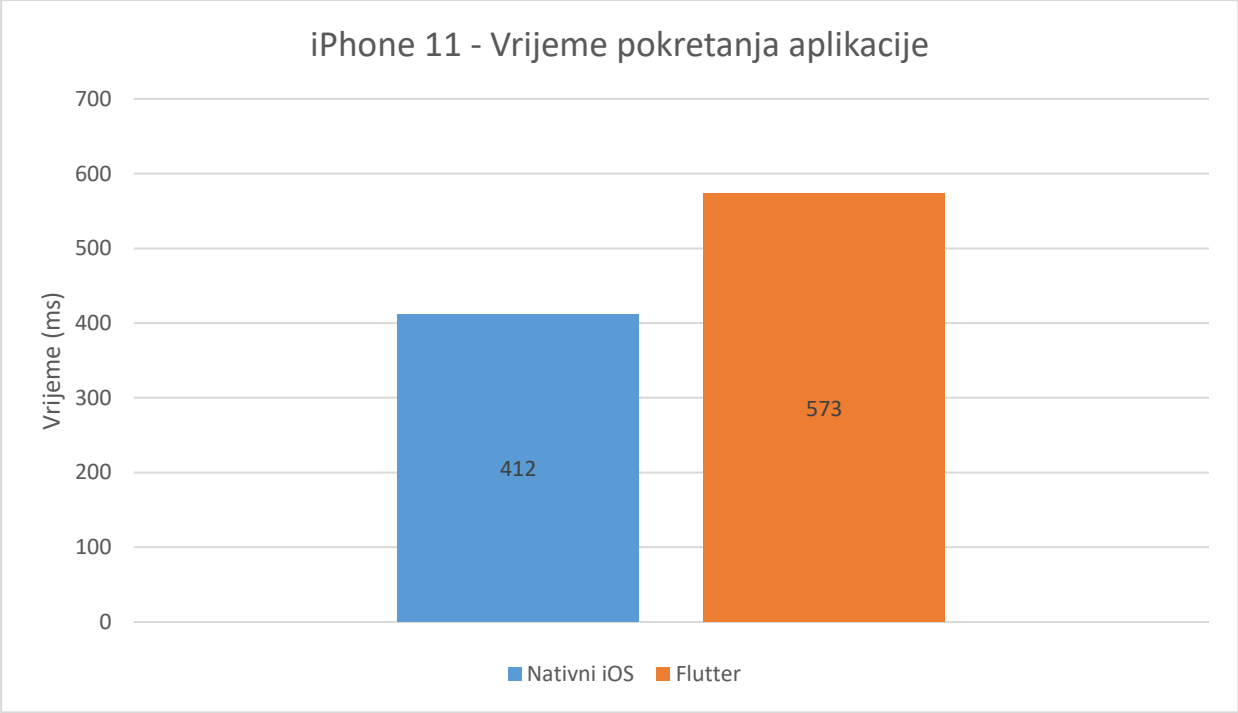
S druge strane, Flutter zajednica je manja, jer je kao zajednica mlađa, no brzo raste i korisnici postaju sve aktivniji, što je uzrokovalo statistikom da je po broju postavljenih i odgovorenih pitanja na StackOverflowu Flutter prestigao nativni iOS razvoj. Zajednica neprestano razvija nove biblioteke koje se objavljuju na pub.dev. Google često organizira događaje koji služe kao dodatna edukacija za korisnike Fluttera, a ujedno pružaju zabavan način za učenje novih stvari. Flutter tim je vrlo angažiran na svojim društvenim mrežama, gdje redovito objavljuju novosti i zanimljivosti iz svijeta Fluttera. To dodatno potiče rast zajednice i pruža korisnicima priliku da se upoznaju s najnovijim trendovima i mogućnostima koje Flutter nudi.

6. USPOREDBA PERFORMANSI NATIVNE iOS I FLUTTER APLIKACIJE

U ovom poglavlju grafički se prikazuju i analiziraju rezultati testova performansi nativne iOS i Flutter aplikacije, provedenih na uređajima iPhone 7 i iPhone 11. Rezultati testova su prosječni rezultati nakon izvođenja većeg broja testova za svaku stavku.

6.1. Vrijeme pokretanja aplikacije

Vrijeme pokretanja nativne iOS aplikacije, koje iznosi 412 ms, i vrijeme pokretanja Flutter aplikacije, koje iznosi 573 ms, ne pokazuju značajnu razliku. Nativna iOS aplikacija pokreće se nešto brže, otprilike 150 ms, zbog toga što Flutter aplikacija zahtijeva pokretanje dodatnih alata prilikom izvođenja koda. Međutim, ova razlika u vremenu pokretanja dovoljno je mala da ne bi negativno utjecala na korisničko iskustvo. U praksi, većina korisnika neće primijetiti tu razliku. Rezultati mjerenja prikazani su na slici 6.1.

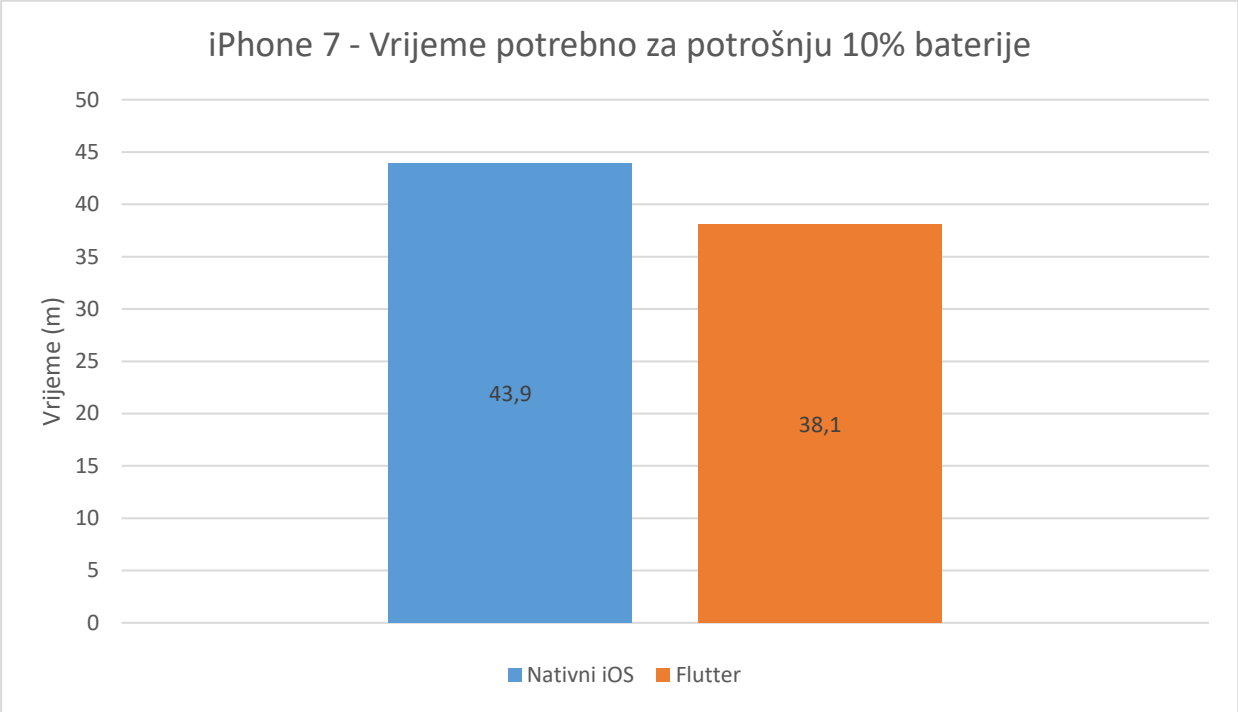


Slika 6.1. Vrijeme pokretanje aplikacije

6.2. Potrošnja baterije

Vrijeme potrebno za potrošnju 10% baterije nativne iOS aplikacije u mirnom stanju, uz smanjenu potrošnju energije na vanjske proces, iznosi 43 minute i 54 sekunde, dok Flutter aplikacija u istim uvjetima troši 10% baterije za 38 minuta i 6 sekundi. Ovi rezultati su grafički prikazani na slici

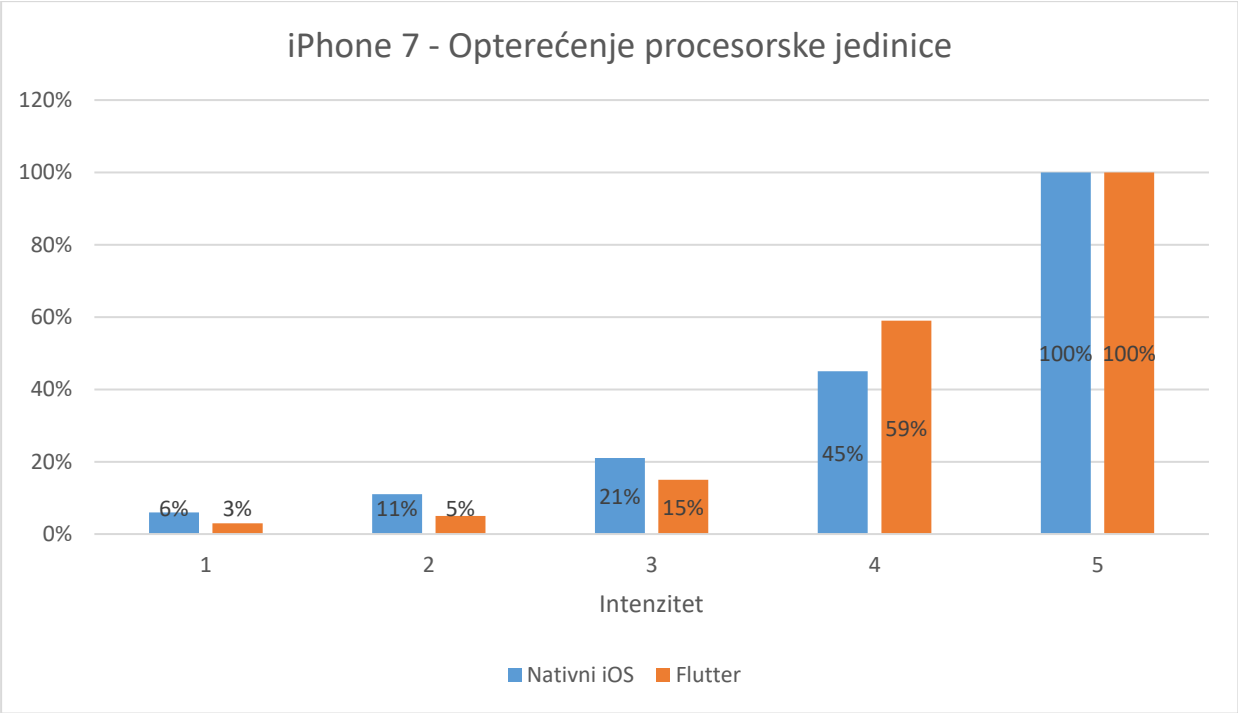
6.2. Ova razlika od približno 5 minuta uglavnom je posljedica korištenja Skia grafičkog pogona u Flutteru, koji možda nije potpuno optimiziran za iOS platformu i stoga može trošiti nešto više energije. Osim toga, Flutter koristi Dart Virtual Machine i druge komponente koje također mogu utjecati na potrošnju energije. Iako je razlika u potrošnji energije između nativne i Flutter aplikacije u ovom slučaju relativno mala, može postati značajnija u složenijim aplikacijama s intenzivnijim radom na grafičkom sučelju ili s više interakcije s nativnim funkcionalnostima platforme.



Slika 6.2. Vrijeme potrošnje 10% baterije

6.3. Opterećenje centralne procesorske jedinice

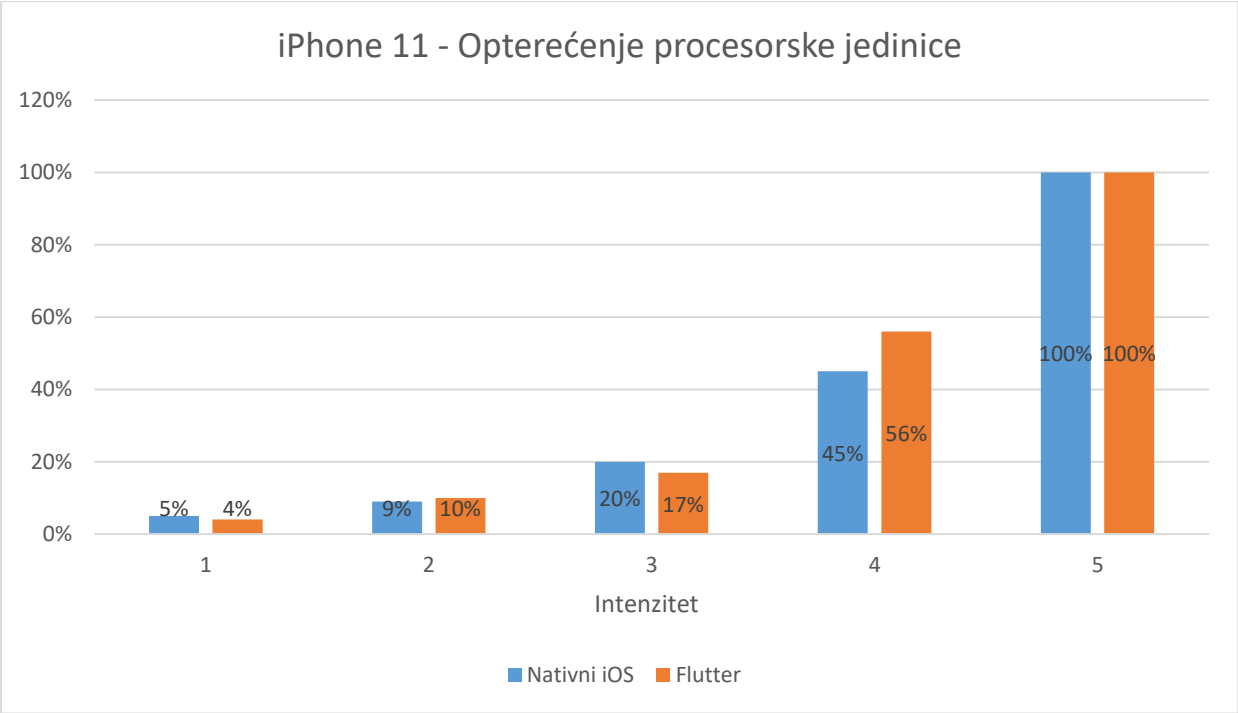
Mjerenje opterećenja centralne procesorske jedinice provedeno je na 5 različitih intenziteta zadatka. Intenzitet opterećenja varira od 1 do 5, gdje je intenzitet 1 jednak 300 000 operacija, a svaki idući intenzitet je 10 puta veći od prethodnog, sve do intenziteta 5 koji iznosi 3 000 000 000 operacija. U tablici 6.1. i tablici 6.2. prikazane su vrijednosti u mikrosekundama (µs) za različite razine opterećenja, a na slici 6.3. i slici 6.4. grafički su prikazana opterećenja centralne procesorske jedinice za vrijeme izvođenja zadatka za uređaje iPhone7 i iPhone 11.



Slika 6.3. Opterećenje centralne procesorske jedinice uređaja iPhone 7

Tablica 6.1. Vrijeme izvođenja zadatka za uređaj iPhone 7

Intenzitet	*1*	*2*	*3*	*4*	*5*
Nativni iOS	2029 µs	16214 µs	78776 µs	486072 µs	5106969 µs
Flutter	1112 µs	10982 µs	114729 µs	1088291 µs	10867524 µs



Slika 6.4. Opterećenje centralne procesorske jedinice uređaja iPhone 11

Tablica 6.2. Vrijeme izvođenja zadatka za uređaj iPhone 11

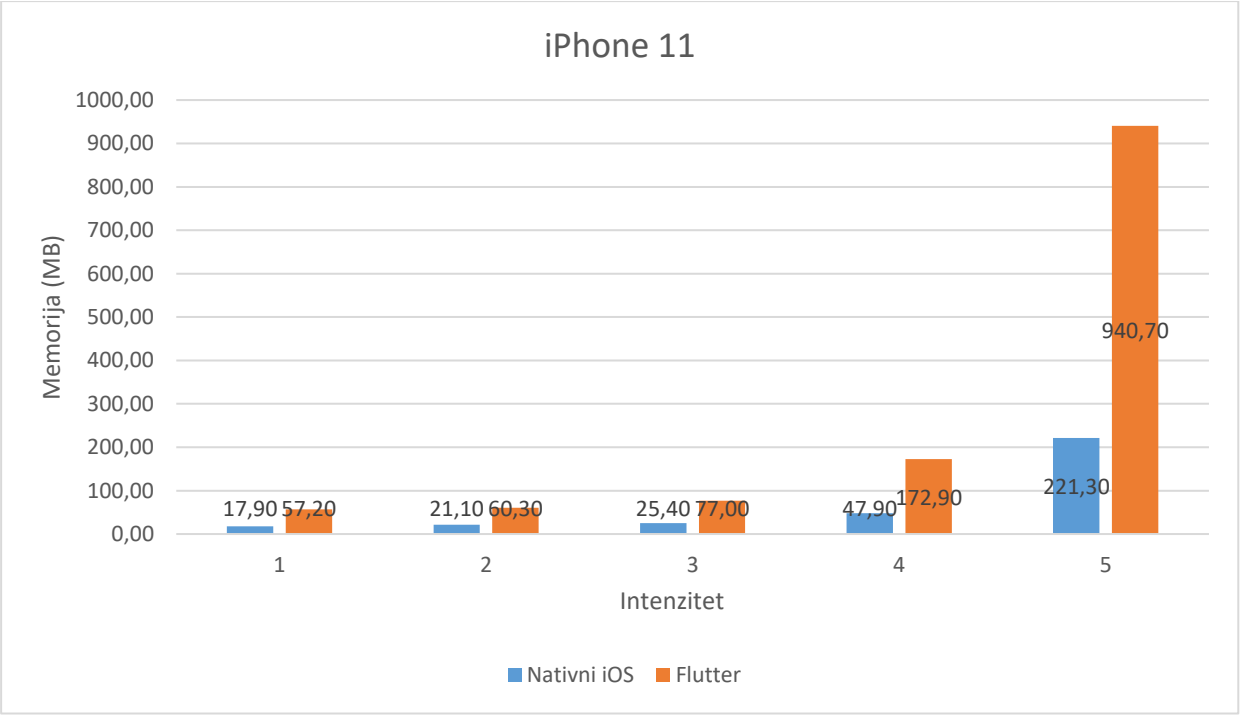
Intenzitet	*1*	*2*	*3*	*4*	*5*
Nativni iOS	1788 µs	13168 µs	77697 µs	418350 µs	3767581 µs
Flutter	2520 µs	19211 µs	104235 µs	694448 µs	6548937 µs

Ovi podatci pokazuju da postoji razlika u opterećenju procesorske jedinice između nativnih iOS aplikacija i Flutter aplikacija. U većini slučajeva, Flutter aplikacije pokazuju veće opterećenje procesne jedinice, posebno na višim razinama opterećenja.

Vrijeme izvođenja zadatka znatno se razlikuje tek kod jako zahtjevnih zadataka, dok je na manjim zadatcima ta razlika neprimjetna i ne utječe negativno na korisničko iskustvo. To znači da za većinu svakodnevnih aplikacija, Flutter može pružiti slične performanse kao nativne iOS aplikacije, dok za zahtjevnije zadatke, nativne iOS aplikacije mogu imati prednost u pogledu učinkovitosti i opterećenja procesorske jedinice.

6.4. Zauzeće radne memorije

Analiza zauzeća memorije nativne iOS aplikacije i Flutter aplikacije izvršena je na zaslonu za skalabilnost korisničkog sučelja. Intenzitet 1 značio je 7 elemenata korisničkog sučelja, intenzitet 2 predstavlja 7^2 elemenata, i tako dalje do intenziteta 5 koji predstavlja 7^5 elemenata. Rezultati mjerenja prikazani na grafu:



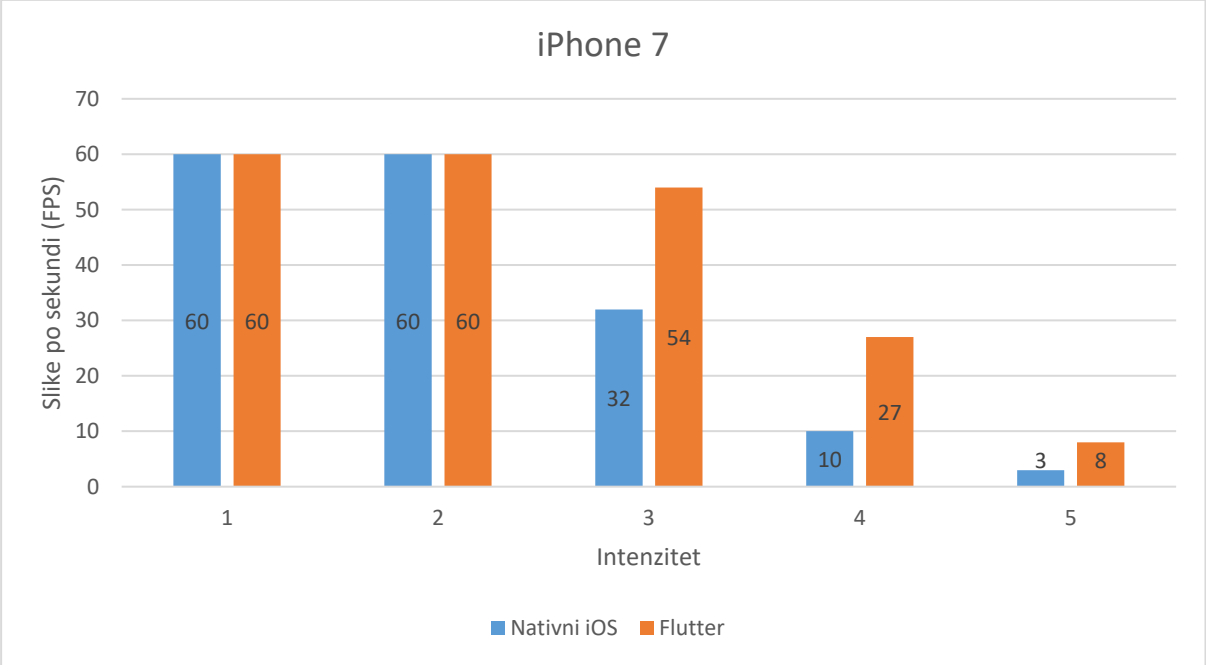
Slika 6.5. Zauzeće radne memorije kod prikaza elemenata korisničkog sučelja

Iz slike 6.5. moguće je zaključiti da Flutter aplikacije zauzimaju znatno više memorije u usporedbi s nativnim iOS aplikacijama kada je u pitanju korisničko sučelje. Ovaj trend postaje izraženiji kako se povećava broj elemenata korisničkog sučelja na zaslonu. Također, izmjereno je povećano korištenje memorije u stanju mirovanja kod Flutter aplikacije koje je iznosilo 52 MB, a kod nativne aplikacije 18 MB.

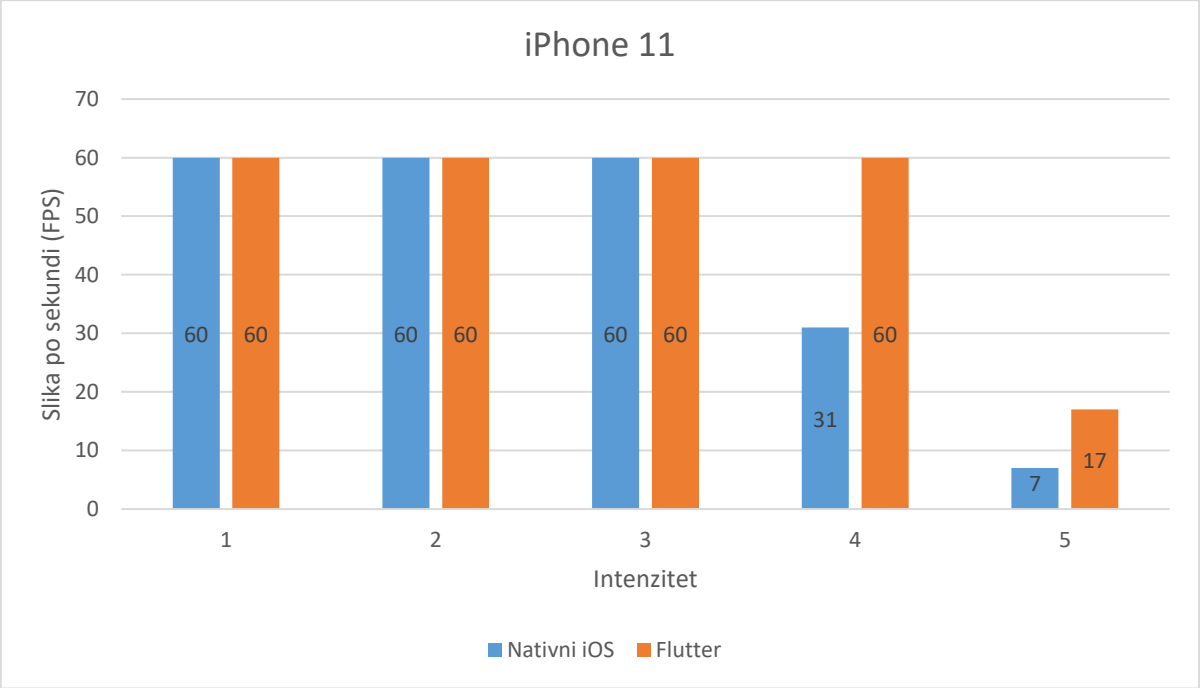
Važno je napomenuti da povišena memorija ne mora nužno ukazivati na lošije korisničko iskustvo, posebno u situacijama s manjim brojem elemenata. Međutim, za aplikacije s velikim brojem elemenata i kompleksnijim sučeljima, veće zauzeće memorije može utjecati na performanse uređaja i korisničko iskustvo. Kod takvih aplikacija, nativni iOS pristup može pružiti bolje rezultate.

6.5. Slike po sekundi (FPS) u animacijama

Testiranje performansi animacija izvodi se s intenzitetima opterećenja od 1 do 5. Broj animiranih objekata može biti između 243 za intenzitet 1 i 10935 za intenzitet 5. Rezultati analize prikazani su na slici 6.6. za uređaj iPhone 7 i slici 6.7. za uređaj iPhone 11.



Slika 6.6. Slike po sekundi za vrijeme animacije za uređaj iPhone 7



Slika 6.7. . Slike po sekundi za vrijeme animacije za uređaj iPhone 11

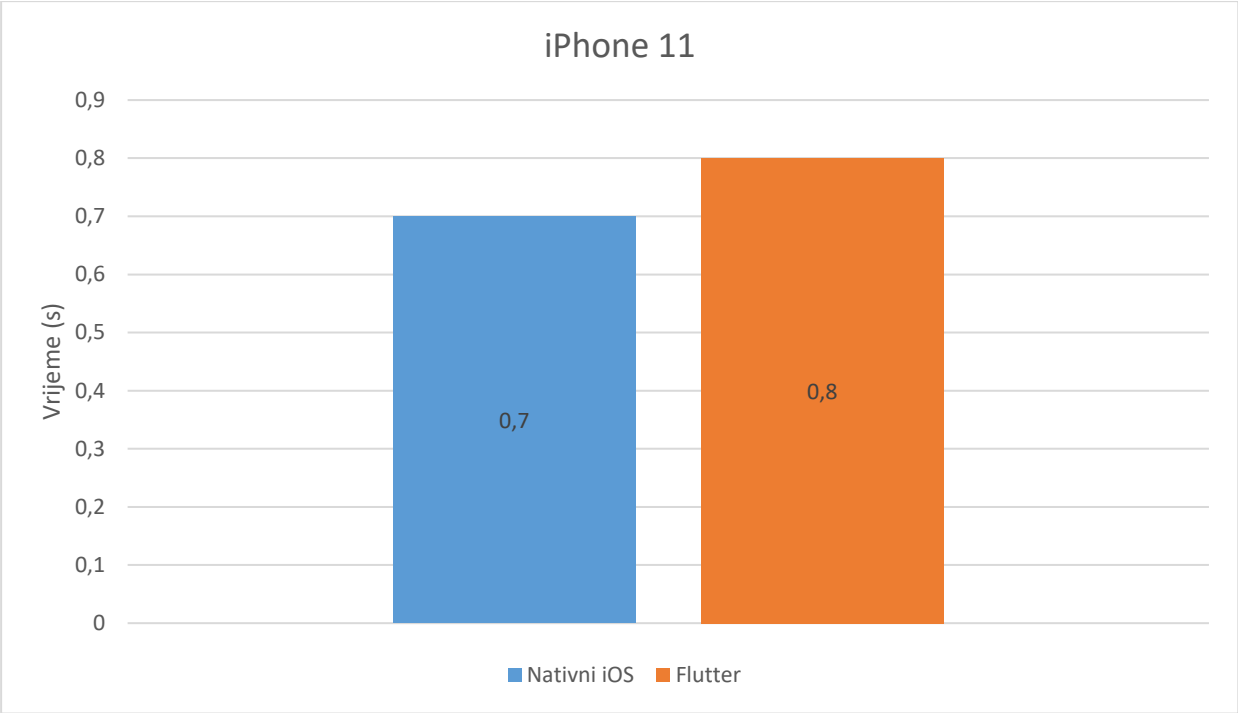
Rezultati pokazuju da Flutter ima bolje performanse u pogledu animacija, posebno pri višim intenzitetima opterećenja. Razlog za ovakve rezultate je način na koji Flutter optimizira animacije. U testiranju, svaki rotirajući krug prikazan je kao zaseban objekt korisničkog sučelja u stogu objekata. Stvaranje i upravljanje velikim brojem objekata na jednom zaslonu pokazalo se kao procesorski vrlo zahtjevan zadatak te su zbog toga animacije bile prikazane s manje slika po sekundi pri porastu broja elemenata.

Važno je napomenuti da je moguće optimizirati animacije u oba pristupa. Kod implementacije koja koristi „*Canvas*“ u iOS i Flutter aplikaciji, animacije su bile znatno otpornije na povećan broj elemenata. Ovaj pristup zaobilazi korištenje zasebnih objekata i umjesto toga crta sve na jednom platnu, nad kojim se vrši animacija rotacije. To smanjuje broj objekata korisničkog sučelja na samo jedan pa obje aplikacije uspješno izvode animaciju na 60 slika po sekundi u svim slučajevima, no vrijeme pokretanja animacije je i dalje bilo povišeno zbog potrebe za iscrtavanjem elemenata na platno.

Metal je također važan alat za optimizaciji iOS aplikacija. Metal je grafički i računalni API razvijen od strane tvrtke Apple, koji omogućuje bolji pristup hardverskim resursima za grafičko i računalno intenzivne operacije. Korištenje Metala može pomoći u povećanju performansi nativnih iOS aplikacija, posebno kod grafički intenzivnih zadataka kao što su animacije.

6.6. Pokretanje prozora korisničkog sučelja

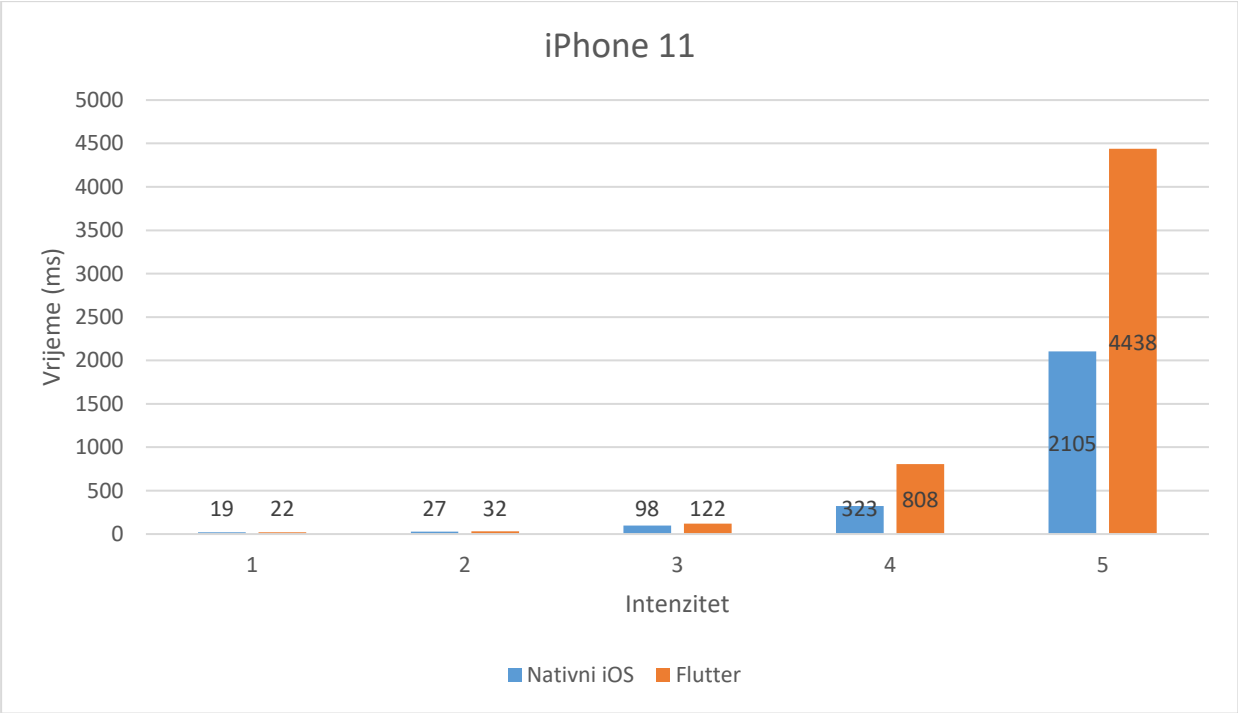
Rezultati vremenskog mjerenja pokretanja prozora korisničkog sučelja za nativne iOS aplikacije i Flutter aplikacije pokazuju vrlo slične vrijednosti. Nativna iOS aplikacija zahtijeva 0,7 sekundi za pokretanje prozora, dok Flutter aplikacija zahtijeva 0,8 sekundi. Ova razlika u vremenu pokretanja je zanemariva i neprimjetna za korisnika. Rezultati mjerenja prikazani su na slici 6.8.



Slika 6.8. Vrijeme pokretanja prozora korisničkog sučelja

6.7. Učitavanje elemenata korisničkog sučelja

Mjerenje vremena potrebnog za učitavanje elemenata korisničkog sučelja izvedeno je za više intenziteta izvođenja zadatka, od intenziteta 1 koji predstavlja 7 elemenata do intenziteta 5 koji predstavlja 7^5 elemenata. Vrijeme koje je izmjereno ne uključuje animaciju otvaranja prozora, već samo učitavanje elemenata. Rezultati su prikazani na slici 6.9.



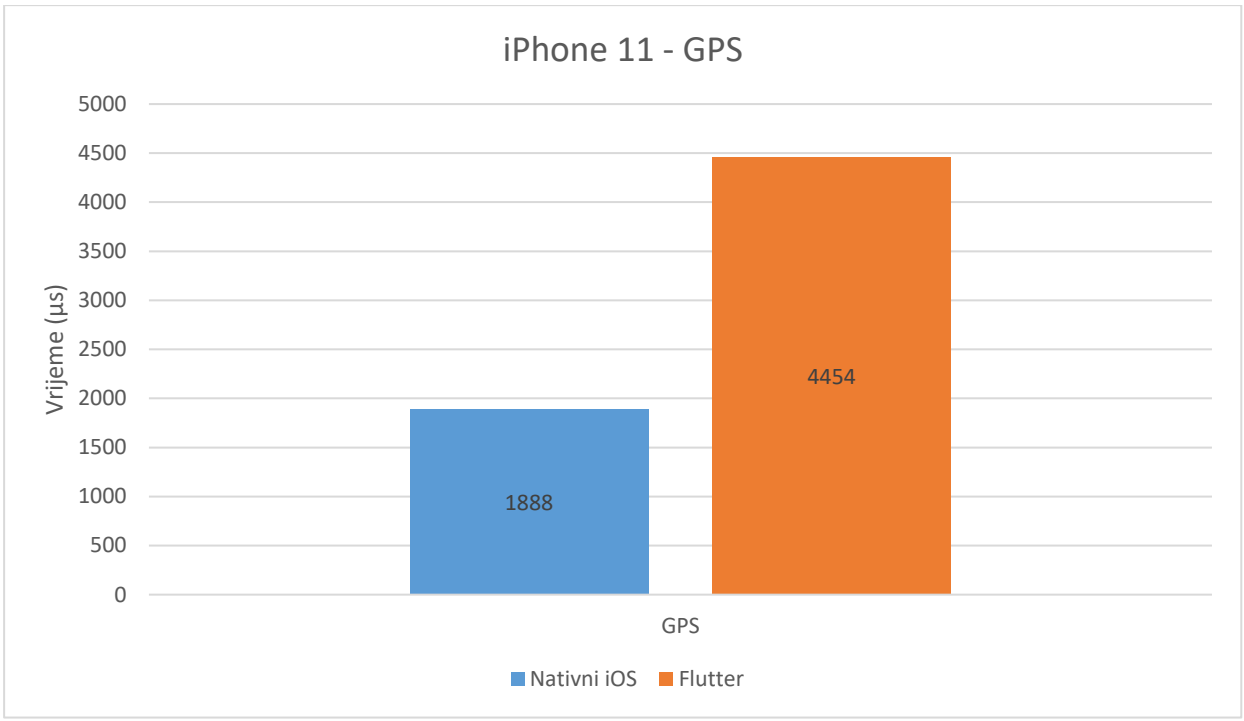
Slika 6.9. Vrijeme učitavanja elemenata korisničkog sučelja

Iz rezultata je vidljivo da nativna iOS aplikacija ima bolje performanse u pogledu učitavanja elemenata korisničkog sučelja u usporedbi s Flutter aplikacijom. Razlika u performansama postaje izraženija s povećanjem broja elemenata.

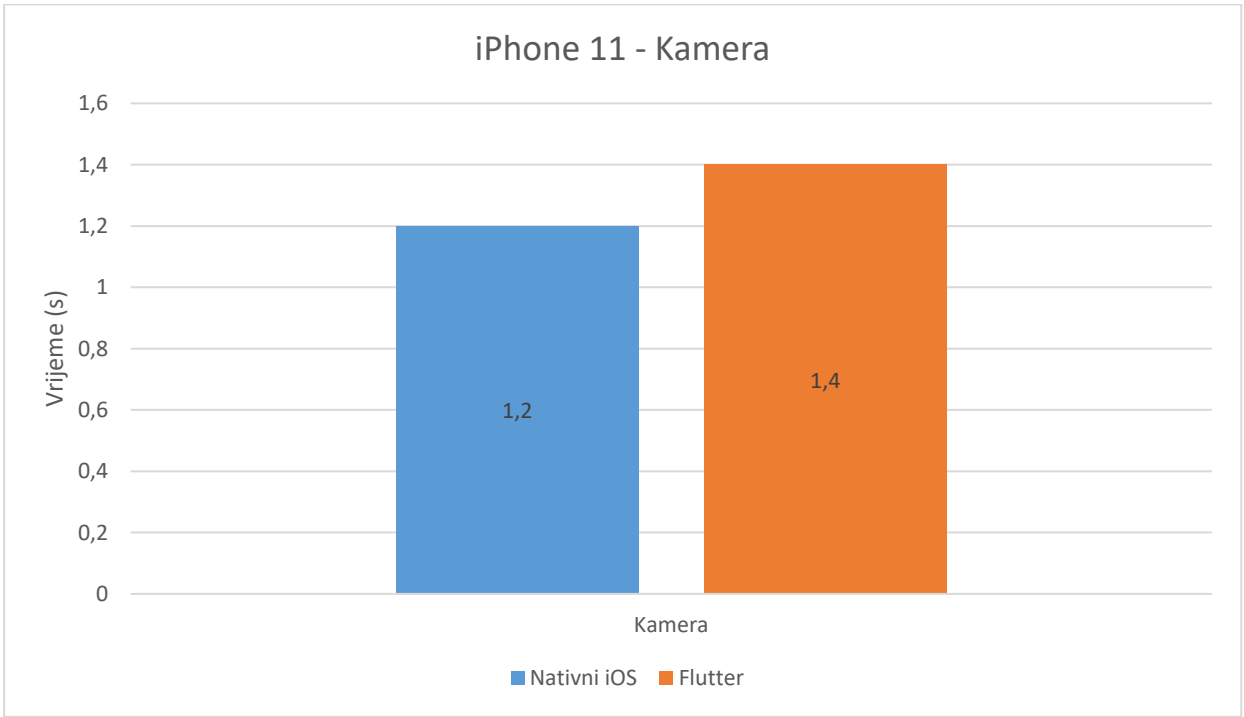
U većini slučajeva aplikacije ne sadrže tako velik broj elemenata korisničkog sučelja koji se prikazuju u isto vrijeme, a čak i kada sadrže velik broj elemenata u listi, postoje metode za modularno učitavanje elemenata koje dijele zadatak na manje zadatke učitavanja. Zbog toga razlike u performansama između ova dva pristupa neće imati značajan utjecaj na korisničko iskustvo u većini aplikacija.

6.8. Vrijeme pokretanja nativnih značajki

Slika 6.10. prikazuje rezultate mjerenja vremena za pristup GPS servisu. Iako nativna aplikacija brže pristupa GPS usluzi, u oba slučaja pristup geolokaciji je iznimno brz, a razlika u vremenu pristupa iznosi samo 3 milisekunde. Ovako mala razlika korisniku je neprimjetna i ne utječe na korisničko iskustvo kod korištenja aplikacije. Slika 6.11. prikazuje rezultate mjerenja vremena za pristup kameri uređaja. U ovom slučaju nativna iOS opet pokazuje bolje performanse, ali su one također zanemarive u odnosu na ukupno vrijeme mjerenja. Ovakve razlike su korisniku neprimjetne i ne utječu na kvalitetu aplikacije.



Slika 6.10. Vrijeme pokretanja GPS servisa



Slika 6.11. Vrijeme pokretanja kamere

6.9. Pristup lokalnoj bazi podataka

U ovoj analizi uspoređene su performanse pristupa lokalnoj bazi podataka u nativnoj iOS aplikaciji i Flutter aplikaciji. Testirane su performanse za različite količine podataka, od 1 do 10000 za intenzitete od 1 do 5. Rezultati su prikazani u tablici 6.3.

Tablica 6.2. Vrijeme izvođenja zadatka za uređaj iPhone 11

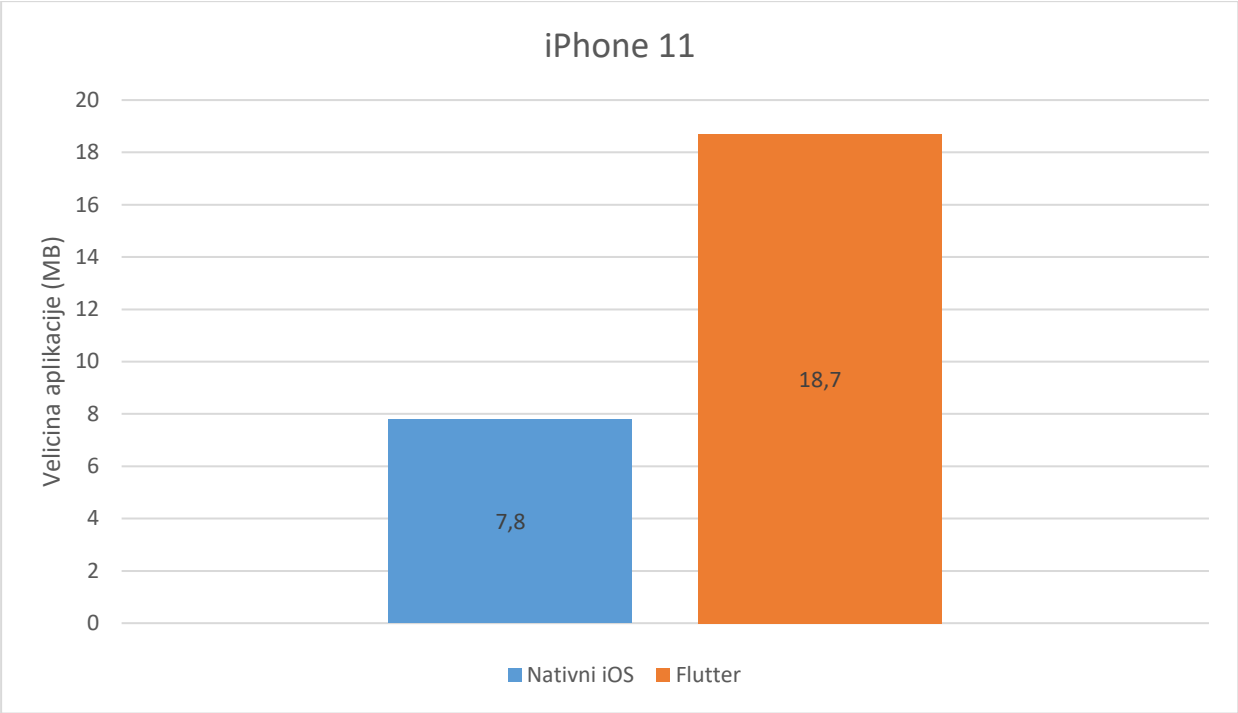
Intenzitet	*1*	*2*	*3*	*4*	*5*
Nativni iOS	16 ms	57 ms	372 ms	3503 ms	34943 ms
Flutter	20 ms	66 ms	378 ms	3741 ms	36730 ms

Iz rezultata je vidljivo da nativna iOS aplikacija ima nešto bolje performanse pristupa lokalnoj bazi podataka u usporedbi s Flutter aplikacijom. Razlika u performansama povećava se s povećanjem količine podataka, iako ostaje relativno mala u odnosu na vrijeme potrebno za izvođenje operacija.

U praksi, razlike u performansama između ova dva pristupa možda neće imati značajan utjecaj na korisničko iskustvo u većinu slučajeva, posebno ako se radi o manjim količinama podataka. Međutim, za aplikacije koje zahtijevaju brz i efikasan pristup velikim količinama podataka, nativni iOS pristup može pružiti nešto bolje performanse.

6.10. Veličina aplikacije

Usporedba zauzeća memorije uređaja vršila se nakon instalacije svake aplikacije u postavkama uređaja. Rezultati mjerenja vidljivi su na slici 6.12.



Slika 6.12.

Usporedba veličine nativne iOS aplikacije i veličine Flutter aplikacije, ukazuje na razliku u veličini paketa. Iz mjerenja vidljivo je da veličina nativne iOS aplikacije iznosi 7,8 MB, dok veličina Flutter aplikacije iznosi 18,7 MB. Ova veća veličina Flutter aplikacija rezultat je dodatnih resursa i biblioteka koje se koriste za podršku višeplatformskom razvoju.

Iako postoji razlika u veličini aplikacije, važno je napomenuti da ova razlika u ne utječe značajno na performanse ili korisničko iskustvo. Suvremeni uređaji imaju dovoljno prostora za pohranu, a visoke brzine internetskih veza omogućuju brzo preuzimanje i instalaciju aplikacija.

6.11. Vrijeme instalacije

Uspoređujući nativne iOS aplikacije i Flutter aplikacije, nije moguće primijetiti značajne razlike u vremenu potrebnom za instalaciju. Obje vrste aplikacija nude brzo i efikasno iskustvo instalacije za korisnike. Nema značajne razlike u vremenu instalacije zbog načina na koji se aplikacije pakiraju i distribuiraju.

Jedini mogući uzrok razlike u brzini instalacije može biti varijacija u veličini aplikacije, ali čak ni to ne može značajno utjecati na vrijeme instalacije.

7. ZAKLJUČAK

U ovom diplomskom radu proučavane su prednosti i nedostaci nativnog i višeplatformskog razvoja mobilnih aplikacija s naglaskom na iOS platformu. Rad se posebno fokusirao na usporedbu performansi aplikacija razvijenih uz nativni pristup koristeći SwiftUI i višeplatformski pristup koristeći Flutter.

Nativni iOS pristup pokazao je minornu prednost u performansama pri normalnim korisničkim uvjetima i primjetno bolje performanse kod nesvakidašnjih zahtjevnijih zadataka. U većini slučajeva aplikacije nemaju potrebu za rad s toliko zahtjevnim zadatcima pa ove razlike ne utječu značajno na korisničko iskustvo. Utvrđene razlike u performansi su neprimjetne za krajnjeg korisnika u većini slučajeva.

Kroz vrijeme će performanse Flutter konstantno napredovati u odnosu na nativne iOS aplikacije, koje postižu maksimalne performanse, zbog činjenice da Flutter razvojni tim kontinuirano radi na optimizaciji performansi i poboljšanju korisničkog iskustva. Osim toga, vrijedno je spomenuti da će performanse samih uređaja napredovati kroz vrijeme što će dodatno smanjiti razlike u izvođenju nativnih i višeplatformskih aplikacija.

Oba pristupa pokazala su se kao validna rješenja za razvoj aplikacija. Flutter se istaknuo kao izvrstan alat za razvoj višeplatformskih aplikacija. Omogućuje značajne uštede u vremenu i troškovima pri razvoju, a uz to, postiže performanse slične nativnoj aplikaciji. Ako aplikacija zahtijeva vrhunske performanse kod zahtjevnijih zadataka, nativni pristup je i dalje najbolje rješenje. Odluka o načinu razvoja ovisi o potrebama projekta i potrebna je detaljna analiza zahtjeva projekta za pravilan odabir.

LITERATURA

[1] M. Levin, *Designing Multi-Device Experiences: An Ecosystem Approach to User Experiences across Devices*. 2014.

[2] „Stack Overflow Insights - Developer Hiring, Marketing, and User Research“. https://insights.stackoverflow.com/survey (pristupljeno 29. travanj 2023.).

[3] „Apple replaces Passbook with Wallet as Apple Pay expands to U.K. | Macworld“. https://www.macworld.com/article/225653/apple-replaces-passbook-with-wallet-as-apple-pay-expands-to-u-k.html (pristupljeno 29. travanj 2023.).

[4] „Flutter Showcase |Google Pay“. https://flutter.dev/showcase/google-pay (pristupljeno 29. travanj 2023.).

[5] J. Hoffman, *Mastering Swift 5.3*, 6. 2020.

[6] „Home - Swift (programming language) - Research Guides at University of Cincinnati“. https://guides.libraries.uc.edu/swift (pristupljeno 29. travanj 2023.).

[7] „Mixing Metal and OpenGL Rendering in a View | Apple Developer Documentation“. https://developer.apple.com/documentation/metal/metal_sample_code_library/mixing_metal_and_opengl_rendering_in_a_view (pristupljeno 29. travanj 2023.).

[8] „Swift.org - The Swift Linux Port“. https://www.swift.org/blog/swift-linux-port/ (pristupljeno 29. travanj 2023.).

[9] „SwiftUI Overview - Xcode - Apple Developer“. https://developer.apple.com/xcode/swiftui/ (pristupljeno 29. travanj 2023.).

[10] E. Windmill, *Flutter in Action*. 2020.

[11] „Google Developers Blog: Flutter: the first UI platform designed for ambient computing“. https://developers.googleblog.com/2019/12/flutter-ui-ambient-computing.html (pristupljeno 29. travanj 2023.).

[12] „Flutter development framework now stable across platforms • The Register“. https://www.theregister.com/2022/05/11/google_io_flutter_crossplatform_app/ (pristupljeno 29. travanj 2023.).

[13] „Flutter - FAQ: How does Flutter run my code on iOS?“ https://docs.flutter.dev/resources/faq#run-ios (pristupljeno 29. travanj 2023.).

[14] „Skia“. https://skia.org/ (pristupljeno 29. travanj 2023.).

[15] „Presentations -> Opening Keynote: Dart, a new programming language for structured web programming“. http://gotocon.com/aarhus-2011/presentation/Opening%20Keynote:%20Dart,%20a%20new%20programming%20language%20for%20structured%20web%20programming (pristupljeno 29. travanj 2023.).

[16] „Instruments Overview - Instruments Help“. https://help.apple.com/instruments/mac/current/#/dev7b09c84f5 (pristupljeno 03. svibanj 2023.).

[17] „groue/GRDB.swift: A toolkit for SQLite databases, with a focus on application development“. https://github.com/groue/GRDB.swift (pristupljeno 29. travanj 2023.).

[18] „moor - Dart API docs“. https://pub.dev/documentation/moor/latest/ (pristupljeno 29. travanj 2023.).

[19] „Flutter vs Native Comparison - Surf“. https://surf.dev/flutter-vs-native/ (pristupljeno 29. travanj 2023.).

SAŽETAK

Ovaj diplomski rad bavi se analizom i usporedbom nativnog iOS i višeplatformskog Flutter pristupa razvoju mobilnih aplikacija. Cilj rada je pronaći i analizirati razlike u performansama oba pristupa i odrediti njihov utjecaj na korisničko iskustvo. Kako bi se postigla objektivna usporedba, razvijene su dvije aplikacije identičnih funkcionalnosti, od kojih je jedna razvijena nativnim iOS pristupom, a druga Flutter pristupom. Analiza se najviše temelji na usporedbi performansi obje platforme. Provedeni su brojni testovi kao što su: analiza opterećenja centralne procesorske jedinice, analiza zauzeća memorije i analiza potrošnje baterije. Ovaj rad pruža uvid u prednosti i nedostatke oba pristupa te može pomoći pri odabiru odgovarajuće tehnologije za razvoj mobilnih aplikacija.

Ključne riječi: analiza, Flutter, iOS, performanse, usporedba, Xcode Instruments

ABSTRACT

Comparison of Flutter and native iOS applications

This thesis deals with the analysis and comparison of native iOS and cross-platform Flutter approaches to mobile application development. The goal of the paper is to find and analyze the differences in the performance of both approaches and determine their impact on the user experience. In order to achieve an objective comparison, two applications with identical functionalities were developed, one of which was developed with a native iOS approach, and the other with a Flutter approach. The analysis is mostly based on a comparison of the performance of both platforms. Numerous tests were carried out, such as: analysis of the load on the central processing unit, analysis of memory usage and analysis of battery consumption. This paper provides an insight into the advantages and disadvantages of both approaches and can help in choosing the appropriate technology for mobile application development.

Key words: analysis, comparison, Flutter, iOS, performance, Xcode Instruments

ŽIVOTOPIS

Krešimir Forjan rođen je 28. lipnja 1998. godine u Osijeku u Hrvatskoj. Pohađao je Osnovnu školu Bilje u Bilju. Nakon završene osnovne škole upisuje III. Gimnaziju Osijek. Za vrijeme srednjoškolskog obrazovanja bavi se veslanjem i sudjeluje na juniorskom svjetskom i europskom prvenstvu. Maturirao je 2017. godine. Iste godine upisuje preddiplomski studij Računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Preddiplomski studij završava 2020. godine i iste godine upisuje diplomski studij na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Za vrijeme diplomskog studija započinje karijeru kao Flutter developer i radi na raznim internacionalnim projektima. Treći semestar na diplomskom studiju provodi na razmjeni studenata u Švedskoj.

_____

Krešimir Forjan

PRILOZI

DVD

- izvorni kod obje aplikacije
- pisani rad u .docx i .pdf formatu