SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I

INFORMACIJSKIH TEHNOLOGIJA OSIJEK (Title stil)

**Sveučilišni diplomski studij**

**ANALIZA MOGUĆNOSTI VIŠEPLATFORMSKOG NATIVNOG RAZVOJA MOBILNIH APLIKACIJA**

Diplomski rad

Ivan Štajcer

Osijek, 2022.

##### SADRŽAJ

[1. UVOD 1](#_Toc107480748)

[1.1. Zadatak diplomskog rada 2](#_Toc107480749)

[2. UVOD U PROBLEMATIKU 3](#_Toc107480750)

[2.1. Izazovi u razvoju aplikacije 3](#_Toc107480751)

[2.2. Trenutno stanje razvoja mobilnih aplikacija 3](#_Toc107480752)

[2.3. Primjeri aplikacija 3](#_Toc107480753)

[3. KORIŠTENE TEHNOLOGIJE I ALATI 4](#_Toc107480754)

[3.1. Funkcijski i nefunkcijski zahtijevi na aplikaciji 4](#_Toc107480755)

[3.2. Korištene tehnologije, alati i jezici 4](#_Toc107480756)

[3.2.1. Flutter 4](#_Toc107480757)

[3.2.1.1. Arhitektura 4](#_Toc107480758)

[3.2.1.2. Widgeti i trenutno stanje aplikacije 5](#_Toc107480759)

[3.2.1.3. Propagacija stanja 6](#_Toc107480760)

[3.2.1.4. Prikazivanje korisničkog sučelja 7](#_Toc107480761)

[3.2.1.5. Prikazivanje promjene stanja 10](#_Toc107480762)

[3.2.2. SwiftUI 11](#_Toc107480763)

[3.2.2.1. Razlika sa *UIKit* alatom 11](#_Toc107480764)

[3.2.2.2. Struktura SwiftUI aplikacije 12](#_Toc107480765)

[3.2.2.3. Prikazivanje korisničkog sučelja 13](#_Toc107480766)

[3.2.2.4. Obnavljanje stanja na ekranu 14](#_Toc107480767)

[3.3. Specifikacija arhitekture aplikacije 15](#_Toc107480768)

[3.4. Testiranje u višeplatformskom razvoju mobilnih aplikacija 15](#_Toc107480769)

[4. PROGRAMSKO RIJEŠENJE NATIVNE I VIŠEPLATFORMSKE APLIKACIJE 16](#_Toc107480770)

[4.1. Implementacija u Flutteru 16](#_Toc107480771)

[4.2. Implementacija u SwiftUI 16](#_Toc107480772)

[5. TESTIRANJE I ANALIZA PROGRAMSKOG RIJEŠENJA 17](#_Toc107480773)

[5.1. Testiranje aplikacije 17](#_Toc107480774)

[5.1.1. Nativna aplikacija 17](#_Toc107480775)

[5.1.2. Višeplatformska aplikacija 17](#_Toc107480776)

[5.2. Analiza rezultata 17](#_Toc107480777)

[5.3. Analiza implementacije i korisničkog iskustva 17](#_Toc107480778)

[6. ZAKLJUČAK 18](#_Toc107480779)

[LITERATURA 19](#_Toc107480780)

[SAŽETAK 20](#_Toc107480781)

[ABSTRACT 21](#_Toc107480782)

[ŽIVOTOPIS 22](#_Toc107480783)

[POPIS PRILOGA 23](#_Toc107480784)

# UVOD

Ovaj rad se bavi usporedbom izrade mobilne iOS aplikacije koristeći noviju višeplatformsku tehnologiju Flutter te implementacije identične mobilne aplikacije koristeći najnoviji nativni pristup izrade iOS aplikacija, *SwiftUI*. Cilj ovog rada je što bolje usporediti sve aspekte dvaju pristupa izrade mobilnih aplikacija. S obzirom na tržite i cijene izrada nativnih aplikacije zasebno, dati dobar pregled koliko se isplati raditi posebne nativne aplikacije umjesto odabira višeplatformkog rješenja. Kako bi se rad što više fokusirao na same alate izrade aplikacija, pri samoj izradi spomenutih aplikacija neće se koristiti komuniciranje sa serverom, radi varijabilnih promjena brzina interneta i teškog uspoređivanja dobivenih rezultata. Također, logika unutar aplikacije će se svesti na što niži nivo, kako bi izbacio moguće greške programera pri samom pisanju koda te stavio fokus na same alate. U radu će se usporediti sam postupak izrade mobilne aplikacije koristeći pojedine alate, brzina izrade, korištene arhitekture, rukovanje stanjima, komponente i slično. Također, uspoređivati će se tehničke performanse aplikacije, kao i korisničko iskustvo. U drugom poglavlju će se opisati korištene tehnologije pri izradi pojedinih aplikacija te način na koji prikazuju korisničko sučelje. Treće i četvrto poglavlje će detaljno opisati čitav proces implementacije pojedinih aplikacija, kako bi se što bolje u razmatranje rezultata uzela sama implementacija aplikacije. Peto poglavlja će usporediti sve prethodno navedene stavke napravljenih aplikacija te, uz uzimanje proces izrade i programera u obzir, dati pregled i opis dobivenih rezultata.

## Zadatak diplomskog rada

Uzimajući u obzir aktualne metodologije razvoja programske podrške, programske arhitekture, programske okvire, alate, jezike i mogućnosti testiranja, u diplomskom radu potrebno je opisati i analizirati mogućnosti i izazove nativnog i višeplatformskog razvoja mobilnih aplikacija. Poseban naglasak treba dati na razvoj za iOS platformu, te na mogućnosti nativnog višeplatformskog razvoja u alatu Flutter i pripadajuće tehnologije i jezike (Dart, koji se koristi u Flutteru te Swift, koji se koristi pri izradi iOS aplikacija). Na primjerima prikladnih projekata ispitnih mobilnih aplikacija s odgovarajućim funkcionalnostima i komponentama na korisničkoj i poslužiteljskoj strani pisanima u navedenim jezicima, potrebno je prikazati postupak nativnog i višeplatformskog nativnog razvoja alatom Flutter i obaviti testiranje programske podrške s naglaskom na testiranje performansi. Na temelju toga, potrebno je napraviti usporedbu i analizu pogodnosti navedenih pristupa u razvoju mobilnih aplikacija, dati osvrt na prikladnost programske arhitekture, te analizirati rezultate testiranja s gledišta postignutih performansi.

# UVOD U PROBLEMATIKU

## Izazovi u razvoju aplikacije

## Trenutno stanje razvoja mobilnih aplikacija

## Primjeri aplikacija

# KORIŠTENE TEHNOLOGIJE I ALATI

U ovom poglavlju je dan pregled korištenih tehnologija pri izradi mobilnih aplikacija.

## Funkcijski i nefunkcijski zahtijevi na aplikaciji

## Korištene tehnologije, alati i jezici

### Flutter

Flutter je višeplatformski alat za razvoj softvera koji omogućava korištenje istog koda na različitim operacijskim sustavima kao što su iOS i Android, uz dopuštanje aplikacijama da direktno komuniciraju sa servisima pojedine platforme [1]. Sama implementacija alata je otvorena i slobodna za izmjene od strane bilo koje osobe. Cilj alata je pružiti nativni osjećaj pri korištenju pojedinih platformi uz održavanje visokih performansi. Kao programski jezik koristi Dart. Dart je klijentski optimiziran jezik za razvoj aplikacija na bilo kojoj platformi uz pružanje značajki kao što su „*type saftey“* i „ *sound null saftey“* [2]. Pri višeplatformskom razvoju, Dart pruža prevođenje koda u strojni kod. To se kod nativnih platformi postiže pomoću virtualne mašine koja pruža „*just in time“* (JIT) prevodilac pri postupku izrade aplikacija, dok u produkcijskom okruženju se koristi „*ahead of time“* (AOT) prevodilac kako bi preveo kod u strojni jezik. Za internetske aplikacije koristi prevodilac za razvojno (*dartdevc*) i produkcijsko (*dart2js*) okruženje kako bi preveo kod napisan u Dart programskom jeziku u JavaScript.

#### Arhitektura

Flutter je dizajniran kao višeslojni sustav sastavljen od više međusobno neovisnih biblioteka, pri kojoj svaka ovisi o sloju na kojem se nalazi. Svaki sloj je osmišljen kao nezavisni te ga je moguće totalno zamijeniti sa nekim drugim [3].

Graphical user interface

Description automatically generated

Sl. 2.1. Slojevi arhitekture Flutter-a , izvor: https://docs.flutter.dev/resources/architectural-overview

Svaki operacijski sustav komunicira sa „*Embedder*“ slojem te se on razlikuje ovisno o kojoj se platformi radi. Njegova svrha je komunikacija sa operacijskim sustavom te pakiranje aplikacija na način da je operacijski sustav ne razlikuje od nativne aplikacije. Pri tome, moguće je uvesti Flutter aplikaciju unutar već postojeće aplikacije kao modul. Također, on komunicira sa Flutter mašinom napisanom u C/C++ programskom jeziku, koja predstavlja centralni sloj. U njoj se nalaze implementacije središnjeg programskog sučelja Fluttera. Na vrhu sloja se nalazi samo programsko okruženje Fluttera koje se koristi pri razvoju aplikacija. Komunikacija sa mašinom je omogućena pomoću „*dart:ui*“ biblioteke sastavljene od Dart klasa, koje služe kao adapteri klasama napisanim u C/C++ programskom jeziku iz same mašine.

#### Widgeti i trenutno stanje aplikacije

Flutter čitavo korisničko sučelje gardi pomoću widgeta. Widget predstavlja osnovnu jedinicu za izgradnju korisničkog sučelja te je implementiran kao nepromjenjiva klasa, koja pomoću svoje „*build()“* metode opisuje kako trenutno izgleda korisničko sučelje s obzirom na trenutno stanje unutar aplikacije [4]. Više različitih widgeta formiraju sučelje koristeći kompoziciju te tim stvaraju trenutnu sliku aplikacije s obzirom na trenutno stanje. Svaki widget od roditelja dobiva „*context“*, preko čega prima informacije o tome gdje se nalazi unutar formiranog stabla. Prilikom promjene stanja, kao na primjer pri reakciji na interakciju korisnika sa aplikacijom, pojedini widget unutar hijerarhije se može zamijeniti sa drugim te time reflektirati promijenjeno stanje unutar aplikacije.

Diagram

Description automatically generated

Sl. 2.2. Hijerarhija widget-a ,

izvor: https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple

Postoje dvije vrste widgeta:

* „*Stateless widget*“
* „*Stateful widget“*

„*Stateless widget“* ne sadrži nikakvo stanje koje je podložno promjenama. Kao na primjer „*Text“* koji služi za prikazivanje teksta na ekran. „*Stateful widget“* sadrži neki atribut koji je podložan promjeni, koja se može pojaviti usred, na primjer, interakcije korisnika s aplikacijom ili drugih faktora. Trivijalni primjer bi bio početna aplikacija dana pri postavljanju svakog novog projekta, koja sadrži gumb te tekst koji prikazuje koliko je puta gumb pritisnut. Takav widget mijenja stanje, ali s obzirom da widgeti sami po sebi nisu promjenjivi, spremaju takva stanja unutar posebne „*State“* klase. Ta klasa sadrži „*build“* metodu, koja treba biti pozvana nakon svake promjene stanja kako bi se ona reflektirala na zaslonu. Pozivajući metodu „*setState“*, Flutter dobiva informaciju o razlici trenutnog stanja widgeta i onog prikazanog na ekranu te poziva „*build“* methodu „*State“* klase.

#### Propagacija stanja

Kako je widget klasa, moguće mu je predati podatke pomoću konstruktora te pomoću tih podataka implementirati „*build“* metodu. Ovaj pristup je primjenjiv dok se aplikacija sastoji od svega par widget-a. Kako se dodaju ove funkcionalnosti i implementiraju zasloni, postupak prosljeđivanja podataka kroz konstruktore postaje vrlo težak. Stoga, kao rješenje Flutter pruža treći tip widgeta, „*InheritedWidget“*. „*InheritedWidget“* omogućuje jednostavan pristup podacima svim widgetima koji se nalaze ispod unutar hijerarhije stabla.

Diagram

Description automatically generated

Sl. 2.3. „*InheritedWidget“*,

izvor: https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple

Ako „*ExamWidget“* treba podatke od „*StudentState“* widgeta, moguće ih je dobiti preko „*context”* parametra, kojeg svaki widget pruža kao argument svoje „*build“* metode. Prilikom toga, dobiti će stanje od najbližeg widgeta koji se nalazi iznad unutar hijerarhije stabla i ima tip „*StudentState“*. Ova metodologija propagiranja stanja unutar aplikacije se koristi od samog Flutter okruženja te je osnova mnogih biblioteka napravljenih od strane programera u svrhu upravljanja stanjem unutar aplikacije.

#### Prikazivanje korisničkog sučelja

Nativne aplikacije, kao ona primjer u Androidu koje se pišu u Java programskom jeziku, zajedno sa svojim bibliotekama pružaju komponente koje predstavljaju što se prikazuje na ekranu. Ti elementi se koriste od strane *Skia* programa koji upravlja grafičkom karticom te ovisno o primljenim komponentama govori grafičkoj kartici što da prikaže na ekranu. Obično, višeplatformske tehnologije iskorištavaju ovo već postojeću implementaciju nativnih tehnologija te jednostavno fasada preko njih koja komunicira s obje platforme. Flutter je drugačiji po tome što totalno zamjenjuje tu već postojeću funkcionalnost nativnih tehnologija sa svojom. Time se postiže bolja i brža komunicira sa *Skia* programom te ima bolju kontrolu. Tome što se prikazuje na korisničkom sučelju.

Diagram

Description automatically generated

Sl. 2.4. Komuniciranje sa GPU

Flutter kao rješenje za određivanje što se prikazuje na korisničkom sučelju, koristi kompozicijsku strukturu stabla, i to tri različite vrsta stabla. [5]

* Stablo widgeta („*Widget tree“*)
* Stablo elemenata („*Element tree“*)
* Stablo prikaza („*Render tree“*)

Stablo widgeta predstavlja upravu onu kompoziciju widgeta koju programer napiše unutar koda. Ono predstavlja deklarativan pristup izricanja kako treba izgledati korisničko sučelje ovisno o trenutnom stanju aplikacije. Postupak prikazivanja korisničkog sučelja, započinje pozivanjem „*build“* fazom, unutar koje se izvršavaju „*build“* metode pojedinih widgeta. Iako je widget sam po sebi nepromjenjiv („*immutable“*), tokom izvođenja „*build“* metode Flutter može ubaciti nove widgete u stablo, čija uloga može biti skrivena unutar nekih od widgeta koje Flutter pruža. Tokom izvođenja „*build“* faze, Flutter. Preslikava stablo widgeta u stablo elemenata u odnosu jedan na prema jedan. Stalbo elemenata predstavlja stvarni opis komponente korisničkog sučelja za pojedini widget zajedno sa njegovim stanjem, ako je „*StatefulWidget“*, povezuje stablo widgeta i stablo prikaza te upravlja stablima. Svaki element zna koji mu je pripadajući widget iz stabla widgeta, tako što zna koji je njegov tip te koji je njegov ključ, ako postoji. Ključ služi kao jedinstveni identifikator koji se može predati svakom widgetu, kako bi element unutar stabla elemenata mogao bolje odrediti sa kojim je widgetom povezan.

Chart, bubble chart

Description automatically generated

Sl. 2.5. Stablo widgeta i stablo elemenata,

izvor: https://docs.flutter.dev/resources/architectural-overview

Svaki element unutar stabla elemnata je jedan od dva osnovna tipa elemenata:

* Element komponente („*ComponentElement“*)
* Element prikaza („*RenderObjectElement“*)

Element komponente služi kao spremink drugim elementima, dok element prikaza služe kao poveznica između pojedinog widgeta unutar stabla widgeta i objekta prikaza unutar stabla prikaza.

Stablo prikaza se sastoji od različitih objekata prikaza kao na primjer „RenderImage*“* ili „RenderParagraph*“*, koji nasljeđuju od osnovnog tipa „RenderObject*“*. Svaki objekt sadrži informacije o tome kako pozicionirati i nacrtati pojedini widget na korisničkom sučelju. Sastavljeno je na principu da svaki objekt zna samo tko su mu djeca unutar stabla te koja su njihova ograničenja s obzirom na veličinu. Koristeći taj princip, Flutter prolazi korz stablo koristeći „deth-first*“* iteraciju stabla. Prilikom posjete, svaki objekt preda dijeci njihova dimenzijska ograničenja. Dijete pojedinog objekta, s obzirom na predana ograničenja, odgovori roditelju koje će mu biti dimenzije. Koristeći ovu tehniku, moguće je proći kroz stablo u linearnom vremenu (*O(n)*). Nakon završetka prolaska kroz stablo, određene su dimenzije svakog objekta te su spremni za prikaz na korisničkom sučelju.

Chart, bubble chart

Description automatically generated

Sl. 2.5. Stabla Fluttera,

izvor: https://docs.flutter.dev/resources/architectural-overview

#### Prikazivanje promjene stanja

Iako widget nije promjenjiv, element unutar stabla elemenata je. Flutter koristi ovo u svoju prednost prilikom prikaza novog stanja na korisničkom sučelju. Kada se stablo widgeta promjeni, kako bi se indiciralo novo stanje aplikacije, Flutter ne gradi posve novo stablo elemenata. Ako je moguće, Flutter iskorištava već postojeće elemente unutar stabla. Svaki element zna sa kojim je widgetom povezan s obzirom na njegov tip i ključ. Ako se widget zamjeni sa widgetom istog tipa, i istog ključa (ako postoji ključ), element se ponovo iskorištava i povezuje sa novim widgetom. Jedina stavka koja se mijenja, su nove informacije iz novonastalog widgeta. To mogu biti informacije, na primjer, veličini teksta, boji itd. Time, element prosljeđuje potrebne informacije objektu prikaza unutar stabla prikaza, koji se također ponovo koristi. Postupak pozicioniranja widgeta na ekranu se ponavlja te se oni oslikavaju na korisničkom sučelju koristeći nove upute o prikazu na ekran.

### SwiftUI

SwiftUI je novi Apple-ov alat za razvoj korisničkih sučelja za iOS, macOS, tvOS i watchOS. Omogućen je za korištenje prvi puta od 2019 godine od kada ga Apple unaprjeđuje iz godine i godinu [6]. Kao programski jezik koristi se Swift. Swift je programski jezik namijenjen za opću namjenu pri uzimanju modernog pristupa kod sigurnosti, performansi i metodologijama dobrog dizajna [7]. Swift je namjenjen da zamjeni *Objective-C* programski jezik te se koristi za razvoj svih aplikacija na Apple platformama.

#### Razlika sa *UIKit* alatom

Iako se SwiftUI razlikuje od UIKit alata, u pozadini koristi elemente iz UIKit alata kako bi prikazao elemente korisničkog sučelja. Za razliku od *UIKit* alata, koji se koristi za izradu aplikacija još od 2008. godine, SwiftUI koristi deklerativan pristup pri izradi aplikacije te podržava više platformi od strane Apple firme [8]. Unutar UIKit alata, programer stvara pojedine prikaze („views*“*) te ih povezuje kako bi stvorio hijerarhiju prikaza koje skupa čine korisničko sučelje. Prilikom promjena na ekranu, potrebno je ponovno kalkuliranje veličina i ograničenja pojedinih prikaza te moguće dodavanje i brisanje prikaza iz hijerarhije. Potrebno je aktivno praćenje i detektiranje promjena te aktivno ažuriranje promijenjenih prikaza. Kontrola prikaza, njihovo stvaranje i brisanje se izvodi na eksplicitni način te se razina kompleksnosti može dosta povećati u kratkom vremenu. *SwiftUI* pruža programsko sučelje pomoću kojeg programer deklarira kako korisničko sučelje treba izgledati. *SwiftUI* koristi te informacije kako bi prikazao ispravno korisničko sučelje. Time je korisničko sučelje prikazano funkcija trenutnog stanja aplikacije. Promjenom stanja aplikacije, promjena se reflektira na ekranu. Ovaj deklarativni pristup pri stvaranju prikaza na ekranu ne zahtijeva ručno stvaranje i brisanje prikaza na ekran od strane programera, što smanjuje kompleksnost i mogućnost grešaka u kodu. *SwiftUI* se oslanja na mogućnosti samog *Swift* programskog jezika, što zahtjeva promjenu jezika kako bi se unaprijedio. Ovo je vrlo bitna stavka kod odabira alatka u kojem će programer izraditi aplikaciju. Animacije su važan dio svake aplikacije, sa SwiftUI pruža lakše rukovanje sa animacijama te dosta animacija i prijelaza se dobije direktno od samog alat. Animacije u UIKit alatu nisu jednostavne za izvest te se moraju eksplicitno navesti kako se izvode. Kako bi dobio najbolje elemente SwiftUI alata, mora koristiti noviju verziju operacijskog sustava na uređaju za kojeg se aplikacija radi. Sa iOS 16 verzijom, SwiftUI je napredovao, ali potrebno je podržavati starije verzije operacijskog sustava kako bi se pokrili svi ciljani korisnici. SwiftUI je dostupan od iOS 13 verzije, ali nije spreman za pravu komercijalnu upotrebu u toj početnoj verziji alata. Stoga, mnogi programeri koriste UIKit kako bi pokrili sve ciljane verzije operacijskih sustava korisnika. No, s vremenom minimalna ciljana verzija se diže te, sa boljim verzijama SwiftUI alata, postaje bolja opcija pri odabiru alata za izradu korisničkih sučelja.

|  |  |  |
| --- | --- | --- |
|  | UIKit | SwiftUI |
| Pristup rada | Imperativan | Deklarativan |
| Minimalna iOS verzija | iOS 9 | iOS 13 |
| Rad s alatom | 14 godina | 3 godine |
| Pokriva sve scenarije | da | ne |
| Brz proces izrade aplikacije | ne | da |
| Interface Builder podržan | da | ne |
| Podržava više platformi | ne | da |
| Rad s animacijama | težak | lagan |
| Podržava widget-e | ne | da |
| Podržava Live Preview | ne | da |

Tablica 2.1. Usporedba SwiftUI i UIKit alata

#### Struktura SwiftUI aplikacije

Struktura SwiftUI aplikacije se iskazuje na deklarativan način, kao što se iskazuju i prikazi na ekranu. Početna točka svake aplikacije kreirati tip koji nasljeđuje „App*“* protokol te ga je potrebno naznačiti sa „@main*“* ključnom riječi koja pruža osnovnu implementaciju „*main“* funkcije. Ovaj protokol predstavlja strukturu i ponašanje aplikacije [9]. „App*“* protokol zahtijeva implementiranje „body*“* izračunatog atributa koji definira sadržaj aplikacije. Tijelo aplikacije je sastavljeno od scena te je svaka scena sastavljena od kompozicije različitih prikaza. Kako je svaki element struktura, a ne klasa, poboljšavaju se performanse pri izvođenju aplikacije. Strukture su unutar Swift programskom jezika „Value type*“* tipovi podataka, što znači da su tretirane kao cijeli broj bilo koja druga vrijednost te ih je moguće predavati s jednog mjesta na drugi bez razmišljanja o stvaranju preljeva memorije ili održavanju pokazivača na njih. Svaki prikaz mora naslijediti „View*“* protokol, koji također nalaže implementaciju „body*“* izračunatog atributa koji mora vratiti točno jedan tip prikaza [10]. Korisničko sučelje se tvori kompozicijom više različitih prikaza, kako bi prikazali trenutno stanje aplikacije. Moguće je razdvojiti posebne dijelove korisničkog sučelje u manje prikaze. Time se postiže veća jasnoća koda, moguće je ponovno korištenje istog prikaza na više različitih mjesta unutar koda te se smanjuje razina kompleksnosti i time se smanjuje broj grešaka. Kako bi se pojedini prikaz konfigurirao na točno određeni način, SwiftUI pruža razne modifikatore za modificiranje izgleda prikaza. Oni predstavljaju metoda nad prikazima koje kao povratnu vrijednost vraćaju novi prikaz te se time mogu vezati pri pozivanju jedan iza drugoga.

Icon

Description automatically generated

Sl. 2.6. Struktura SwiftUI aplikacije

#### Prikazivanje korisničkog sučelja

SwiftUI, kao i u UIKit alatu, je apstrakcija iznad petlje događaja koja je odgovorna za slanja poruka kodu koji ja zaslužan za korisničko sučelje. Zauzvrat kod odgovoran za korisničko sučelje može odrediti koji se dijelovi ekrana trebaju ažurirati. Unutar SwiftUI alata petlja je skrivena te nije potrebno direktno komunicirati sa niti znati da ona postoji [11]. Na svim Apple platformama, nalazi se instanca petlje događaja „*CFRunLoop“* koja dolazi iz „Core Foundation*“* biblioteke te je prisutna na svim alatima za izradu korisničkog sučelja još od Ma OS X 10.0. Petlja događaja je dio infrastrukture povezane sa nitima. Njen zadataka unutar te infrastrukture je da rezervira poslove koje se trebaju napraviti i organizira nadolazeće događaje. Ova petlja efektivno daje niti posao kada posla ima te ju stavlja u stanje spavanja kada nema posla za obradu [12]. Svi događaji koji dolaze unutar petlje događaja su obrađeni različitim redom, ovisno o tipu događaja. Postoje četiri različita tipa nadolazećih događaja.

* Tip 0. Događaji koji direktno komuniciraju sa „*CFRunLoop“* instancom, kao na primjer dodiri na ekranu.
* Tip 1. Događaj sinkroniziranje ažuriranja ekrana i ponovnog prikazivanja na ekranu te neki asinkroni događaji za. Dohvaćanje podataka sa interneta.
* Mjerači vremena.
* Glavni red za obrađivanje. Bilo kakav kod asociran sa ovim redom.

Osim navedenih ulaznih događaja, moguće je dodati slušatelje na instancu „*CFRunLoop“* koji budu obaviješteni kada petlja dosegne određeni dio [13]. Animacije se vrte na glavnoj niti, ali ako glavna nit zablokira, moguće je da će animacije i dalje raditi. Razlog tome je što se animacije, iako vrte na glavnoj niti, ne vrte na istom procesu. Aplikacija pomoću biblioteke „*Core Animation“* prikazuje svoje animacije na i prikaze na ekran, ali ih prikazuje na više različitih slojeva koji se nazivaju „*CALayer“*. Pomoću kompozitora aplikacija spaja više različitih slojeva te ih prikazuje na ekran, dok „*Core Animation“* komunicira sa serverom za crtanje prikaza na ekran kako bi mu dao informaciju o tome što treba nacrtati i animirati.

#### Obnavljanje stanja na ekranu

Kada se neko stanje unutar aplikacije promjeni, potrebno je reflektirati tu promjenu na ekranu tako što se mjenjaju pojedini dijelovi prikaza kao što su boja ili tekst. Nakon napravljenih promjena na pojedinim slojevima, potrebno je komunicirati tu promjenu serveru za crtanje te prikazati ih. Aplikacija ne može znati kada su sve promjene gotove i spremne za crtanje na ekran. Za signaliziranje završetka promjena koristi se transakcija, instanca „*CATransaction“*. Transakcija se mora pokrenuti kada se promjene dese i završiti kada su promjene gotove. Pomoću ovog mehanizma aplikacija zna da su sve promjene gotove za trenutni sloj te se pomoću servera za crtanje prikazuju na ekran [14]. Transakcije se mogu manualno stvoriti i završiti te su osnova unutar svake Apple, pa tako i SwiftUI aplikacije. Tokom izrade aplikacija, programer ne komunicira direktno sa transakcijama, iako to može učiniti uz pomoć „*Core Animation“* biblioteke, već se transakcija automatski pokrene dolaskom promjena i završi na kraju jednog obilaska petlje događaja.

Text, chat or text message

Description automatically generated

Sl. 2.6. SwiftUI render loop, izvor: https://rensbr.eu/blog/swiftui-render-loop/#:~:text=Just%20like%20UIKit%20%2C%20SwiftUI%20is,render%20loop%20of%20an%20application.

## Specifikacija arhitekture aplikacije

## Testiranje u višeplatformskom razvoju mobilnih aplikacija

# PROGRAMSKO RIJEŠENJE NATIVNE I VIŠEPLATFORMSKE APLIKACIJE

## Implementacija u Flutteru

## Implementacija u SwiftUI

# TESTIRANJE I ANALIZA PROGRAMSKOG RIJEŠENJA

## Testiranje aplikacije

### Nativna aplikacija

### Višeplatformska aplikacija

## Analiza rezultata

## Analiza implementacije i korisničkog iskustva

# ZAKLJUČAK

# LITERATURA

[1] Flutter, https://flutter.dev/

[2] Dart, https://dart.dev/overview#platform

[3] Flutter docs, arhitectular overview, https://docs.flutter.dev/resources/architectural-overview

[4] Geeks for geeks, https://www.geeksforgeeks.org/what-is-widgets-in-flutter/

[5] Flutter docs, inside Flutter, https://docs.flutter.dev/resources/inside-flutter

[6] Introducing SwiftUI, https://developer.apple.com/tutorials/swiftui

[7] About Swift, https://www.swift.org/about/

[8] What is SwiftUI, https://cocoacasts.com/swiftui-fundamentals-what-is-swiftui#:~:text=SwiftUI%20is%20Apple's%20brand%20new,is%20a%20cross%2Dplatform%20framework.

[9] App Structure and Behavior, https://developer.apple.com/documentation/swiftui/app-structure-and-behavior

[10] SwiftUI View, https://developer.apple.com/documentation/swiftui/view/

[11] Apple documentation, Run Loop, https://developer.apple.com/documentation/foundation/runloop

[12] The SwiftUI render loop, https://rensbr.eu/blog/swiftui-render-loop/#:~:text=Just%20like%20UIKit%20%2C%20SwiftUI%20is,render%20loop%20of%20an%20application.

[13]. Run Loops, https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/Multithreading/RunLoopManagement/RunLoopManagement.html

[14] Animations in SwiftUI: Get to Know Transactions, https://betterprogramming.pub/animation-in-swiftui-get-to-know-transactions-7cd57cfb299f

# SAŽETAK

# ABSTRACT

# ŽIVOTOPIS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Potpis autora

# POPIS PRILOGA