Usporedba nativnog i višeplatformskog razvoja mobilne aplikacije u iOS i React-Native tehnologijama

Jurišić, Blaž

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

Sveučilišni preddiplomski studij

USPOREDBA NATIVNOG I VIŠEPLATFORMSKOG RAZVOJA MOBILNE APLIKACIJE U IOS I REACT-NATIVE TEHNOLOGIJAMA

Završni rad

Blaž Jurišić

Osijek, 2018.

SADRŽAJ

1. UVOD

Razvoj programske podrške, od samih početaka, nastojao se pojednostaviti uvođenjem alata i tehnologija koje omogućuju razvoj programa koji ima mogućnost izvršavanja na više platformi. Neki od razloga težnje tom pristupu su jeftiniji razvoj, manji broj ljudi potreban za razvijanje jedinstvenog koda koji čini aplikacije na svim platformama, jedinstven programski jezik te nametanje tehnologije kao sveobuhvatne i samodostatne na tržištu. Unatoč navedenim težnjama, razvoj programskih rješenja specifičnih za pojedinu platformu prisutan je u suvremenom razvoju. Glavni razlozi tome su sigurnost i jednostavnost same implementacije te velika podrška tvrtki koje promoviraju takav razvoj.

Cilj ovog rada je opisati oba pristupa, istaknuti njihove prednosti i nedostatke, usporediti dvije, trenutno najpopularnije tehnologije za razvoj mobilnih aplikacija na iOS platformi te provesti mjerenja koja potvrđuju, odnosno osporavaju mišljenja različitih dijelova zajednice, kako razvojnih programera, tako i njihovih menadžera koji imaju čest problem u odabiru tehnologija potrebnih za realizaciju određenog programskog rješenja.

Poglavlje 2 opisuje metode razvoja programske podrške. U poglavlju 3 opisane su tehnologije iOS nativnog razvoja. U poglavlju 4, opisan je razvoj uz pomoć React Native tehnologije. U poglavlju 5 opisana je aplikacija koja je ostvarena u obje tehnologije, na kojoj je izvršena usporedba te napravljena mjerenja.

1.1. Zadatak završnog rada

U radu je potrebno opisati programska obilježja iOS i React-Native tehnologija, predložiti model vlastite mobilne aplikacije koja omogućuje izbor odredišta na temelju geolokacije i zadanih parametara, te usporediti programske jezike i paradigme u navedenim tehnologijama. Mobilnu aplikaciju potrebno je programski ostvariti u navedenim tehnologijama, te usporediti nativni i višeplatformski razvoj s gledišta jezika i jezične paradigme, okolina, alata i postignutih performansi.

2. METODE RAZVOJA

Kroz povijest, postojalo je više trendova i pristupa razvoju. Nastanak i kreacija novih metoda ovisila je o potrebama i zahtjevima programa te omjeru kompleksnosti postojećih alata i tehnologija. Metode su ponekad bile znatno različite, no često bile i vrlo slične, pa su konvencijom klasificirane u tri skupine:

- Nativni razvoj
- Višeplatformski razvoj
- Hibridni razvoj

I dalje je najzastupljeniji razvoj nativni razvoj, a ponajviše zbog jednostavnosti postizanja kvalitetne implementacije te podrške velikih tvrtki koje žele nametnuti svoju tehnologiju kao vodeću.

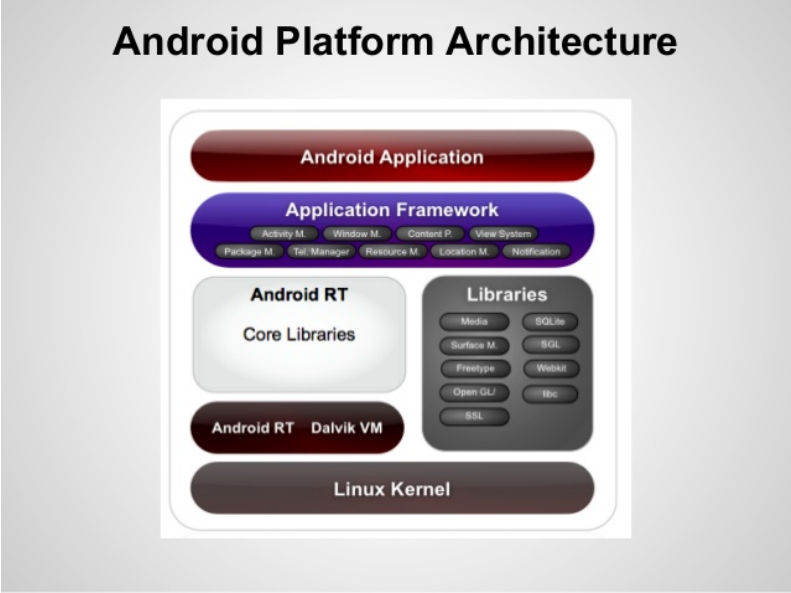
Hibridni je razvoj nastao posljednji, kao proizvod rapidnog razvoja web tehnologija korištenih u internetskim preglednicima. Rezultat hibridnog razvoja programa je programsko rješenje za svaku podržanu platformu pa se takav razvoj također može smatrati višeplatformskim, no način na koji hibridna rješenja funkcioniraju značajno se razlikuje od rješenja dobivena korištenjem tehnologija koje su svrstane u višeplatformski razvoj.

2.1. Nativni razvoj

Nativni razvoj označava razvoj koji je specifičan za jednu platformu, a karakterizira ga jedinstven ekosustav.

2.1.1. Definiranje platforme

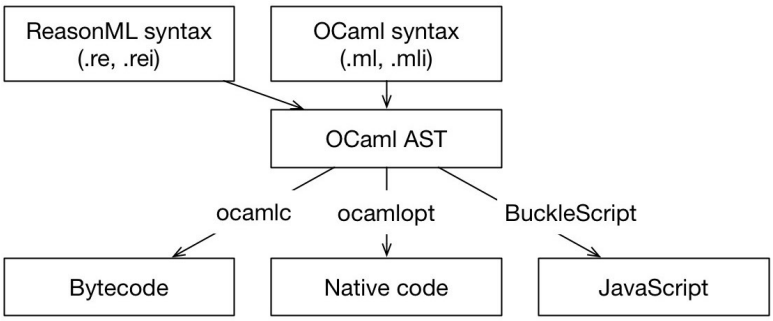
Pojam platforma, u svom užem smislu, označava sklopovlje (najčešće CPU) za koji je operacijski sustav, odnosno aplikacija namijenjena. U širem smislu, odnosno u kontekstu i domeni ovog rada, a tako i u kontekstu razvoja programskih rješenja, platforma označava uređaj (*sklopovlje*) i operacijski sustav kao okruženje u kojem se aplikacija izvodi. Također, platforma obuhvaća i kostur (eng. *framework*) te programska sučelja (eng. API - *application programming interface*) potrebna za komunikaciju sa sklopovljem i operacijskim sustavom. Na slici 2.1 [1] vidljiva je arhitektura Android platforme.



Sl. 2.1. *Arhitektura Android platforme* [1]

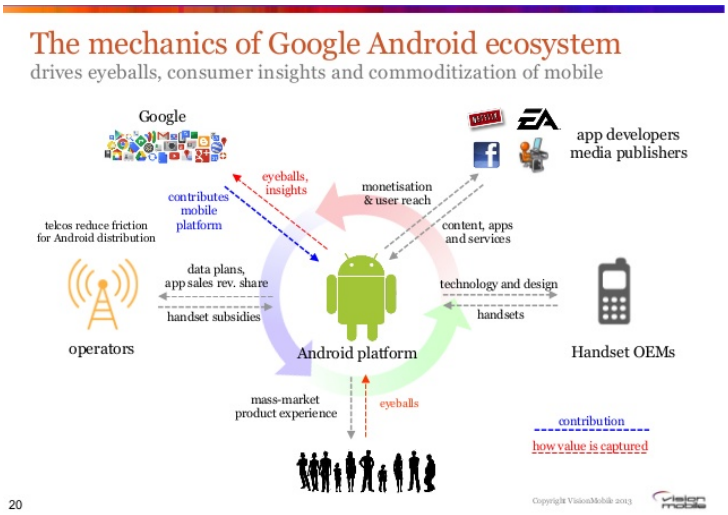
2.1.2. Ekosustav

Pojam ekosustav označava skup alata i tehnologija potrebnih za razvoj programa. Najčešće se odnosi na programski jezik i sustav za izvršavanje (engl. *build system*), alate, upravitelj paketima (*package manager*), razvojno okruženje (eng. IDE - *integrated development environment*) te standardne biblioteke, no dolaskom pametnih telefona, u kontekstu razvoja mobilnih aplikacija, često obuhvaća i razvojni (engl. *deployment*) sustav, trgovinu aplikacijama i ostale sustave koji su dio ciklusa razvoja i objave aplikacije na mobilni uređaj. Slika 2.2. [2] prikazuje primjer ekosustava kao što je Ocaml ekosustav.



Sl. 2.2. *OCaml ekosustav* [2]

U svojem najširem smislu, ekosustav, uz već navedeno, obuhvaća i krajnjeg korisnika, razvojne programere, pružatelje telekomunikacijskih usluga pa i marketing prostor. Slika 2.3 [3] prikazuje Android ekosustav u svojem najširem smislu.



Sl. 2.3. *Android ekosustav* [3]

2.2. Višeplatformski razvoj

Višeplatformski razvoj često se različito interpretira. Iako je krajnji rezultat jednak, postoji distinkcija među pristupima koji se nazivaju višeplatformskim.

2.2.1. Virtualni strojevi

Višeplatformski razvoj može se odnositi na aplikacije koje rade po principu "Prevedi jednom, pokreni svagdje" (eng. *Compile once, run everywhere*). Primjer takve aplikacije bi bila aplikacija pisana u programskom jeziku Java, gdje se Java kod prevodi u bytekod (eng. *bytecode*). Bytekod se potom izvodi u Java virtualnom stroju (eng. JVM - *Java virtual machine*), koji funkcionira kao virtualni procesor. Postoji više programskih jezika koji se prevode u Java bytekod (Clojure, Groovy, Kotlin, Scala, itd.), samim tim, programska rješenja razvijana uz pomoć navedenih jezika također se mogu pokretati u JVM. Također, postoje i drugi virtualni strojevi poput CLR, virtualnog stroja .NET okruženja.

2.2.2. Generatori koda

Drugi kontekst višeplatformskog razvoja predstavljaju aplikacije koje imaju jedinstvenu bazu koda (eng. *codebase*). Cilj ovakvog razvoja je napisati aplikaciju u jednom jeziku, a tehnologija i alati

u pozadini generiraju nativni kod posebno za svaku platformu. Takav pristup ima velike prednosti jer su performanse takvih aplikacija usporedive s nativnim aplikacijama. Također, automatizirajući dugi proces prilagodbe iz jednog ekosustava u više drugih ekosustava omogućuje timu razvojnih programera da pišu jedinstven jezik u jedinstvenom ekosustavu. Time se štedi na vremenu i količini programskog koda koji je potreban kako bi se aplikacija razvila na više platformi. U idealnim uvjetima, uz korištenje savršene tehnologije, ušteda vremena, koda i resursa iznosila bi

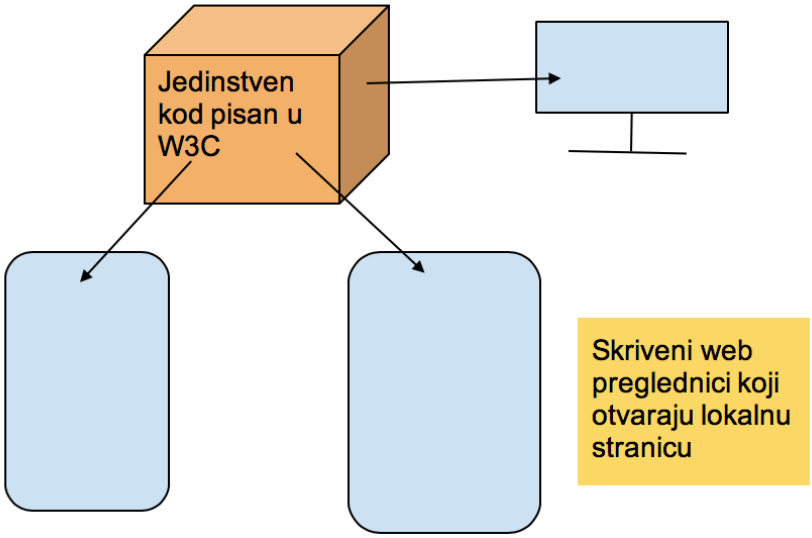
$$f(x,n) = \frac{x}{n} \tag{2-1}$$

gdje je *x* promatrani resurs, a *n* broj podržanih platformi. U stvarnim uvjetima, takva ušteda je nemoguća zbog manjkavosti tehnologija koje rade po principu jedinstvene baze koda. To je upravo glavni nedostatak ovakvog pristupa. Ne postoji savršena tehnologija koja bi iz jedinstvenog koda generirala nativni kod više platformi. Samim tim što tvrtke koje proizvode ekosustav ne stoje iza ovakvog pristupa, tvrtka treće strane konstantno kasne s novijim značajkama sustava.
Također je moguće da nove promjene na ekosustavu "slome" API i implementaciju takve tehnologije, pa je potrebno čekati zakrpe. U praksi, takvih slučajeva nema mnogo.

Trenutno najpoznatije međuplatformske tehnologije (eng. *cross-platform technology)* za izradu klijentskih aplikacija su React Native, Qt i Xamarin.

2.3. Hibridni razvoj

Hibridni razvoj novi je pristup kojeg je stvorila zajednica web programera. Hibridni razvoj funkcionira tako da se na svakoj platformi nalazi nativni web preglednik koji učitava zajednički kod pisan u tehnologijama poput HTML, CSS, JavaScript koje se koriste za izradu web aplikacija. Korisnik aplikacije u pravilu ne bi trebao moći prepoznati hibridnu aplikaciju, no zbog slabijih performansi i sučelja koje nije karakteristično za platformu, korisnici često znaju prepoznati razliku između hibridne aplikacije i nativne. Slika 2.4 prikazuje graf koji opisuje način na koji funkcionira hibridni razvoj, odnosno hibridna aplikacija.



Sl. 2.4. *Graf koji pokazuje arhitekturu hibridne aplikacije*

Najpoznatije tehnologije za hibridni razvoj su PhoneGap, Electron, Cordova i drugi.

3. RAZVOJNE TEHNOLOGIJE IOS PLATFORME

Krajem 2007. godine, tvrtka Apple u prodaju je pustila mobilni uređaj iPhone i time pokrenula revoluciju mobilnih uređaja koji se zajedničkim imenom nazivaju "pametni uređaji".

iOS, prvobitno iPhoneOS, vrlo brzo je postao najbolji operacijski sustav za mobilne uređaje čija je trgovina aplikacijama najprofitabilnija na tržištu. Trenutno je drugi najčešće korišteni operacijski sustav, a izvršava se na više vrsta uređaja kao što su iPad i iPod. Na slici 3.1 [4] prikazan je napredak dizajna iOS operacijskog sustava od prve do devete verzije.

Sl. 3.1. *Evolucija iOS operacijskog sustava* [4]

3.1. iOS ekosustav

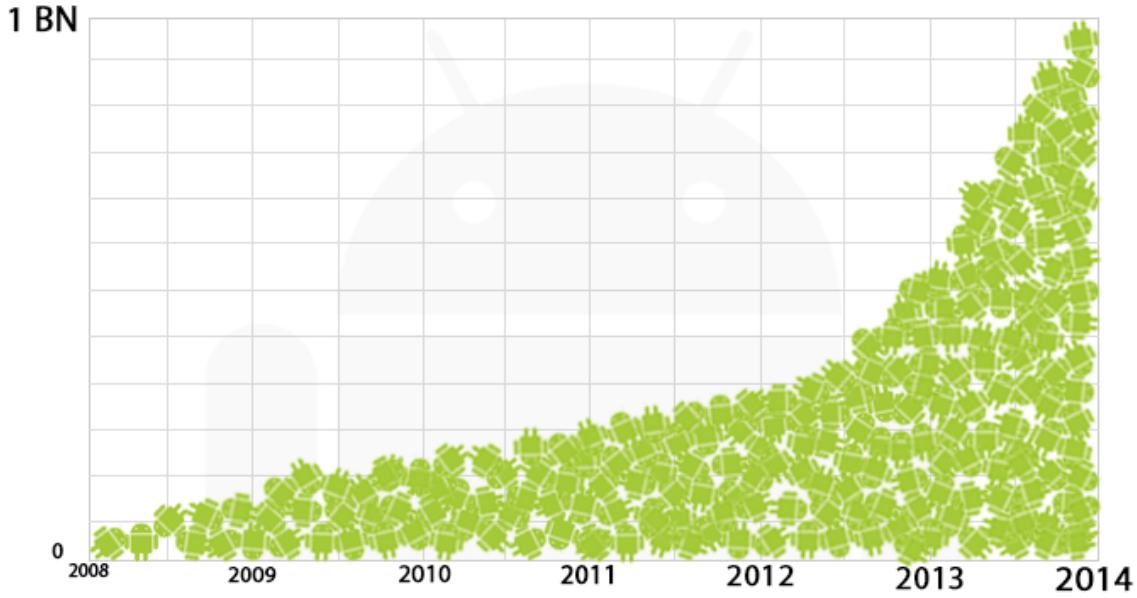
iOS ekosustav smatra se najboljim ekosustavom mobilnih uređaja. Za razliku od najvećeg konkurenta, Android ekosustava, iOS ima rješenje za vrlo bitne stavke kao što su fragmentacija, faktor prihvaćenosti te monetizacija trgovine. Također, razvojni alati su vrlo kvalitetni i dobro podržani od strane Apple-a, a zajednica je dovoljno velika i aktivno sudjeluje u unapređenju ekosustava.

3.1.1. Fragmentiranje

iOS, za razliku od Android operacijskog sustava [5] Apple nema problema s fragmentiranjem. Android operacijski sustav je program otvorenog koda (eng. *open source*), pa mnogi proizvođači treće strane razvijaju svoje inačice Android sustava koje često nisu kompatibilne sa

nadogradnjama službenog Android operacijskog sustava. Ponekad, takvo fragmentiranje vodi do neujednačenosti i problema čak i s osnovnim sigurnosnim nadogradnjama OS -a.

U 2014. godini, 20% svih uređaja koji pokreću Android, pokretalo je neslužbenu inačicu sustava. Zbog rasta tržišta mobilnih telefona, raste i udio neslužbenih inačica programa. Na slici 3.2 [6], vidljiv je porast neslužbenih Android inačica od 2008. do 2014. godine.



Sl.3.2. *Graf iz 2014. godine grafički prikazuje broj različitih inačica Android sustava. [6]*

Kod operacijskog sustava iOS nije otvoren, pa postoji samo jedna službena inačica programa koju održava i nadograđuje tvrtka Apple.

3.1.2. Čimbenik prihvaćenosti

Android operacijski sustav je besplatan i otvorenog koda, a prilagodljiv je velikom broju uređaja. Iako ideja operacijskog sustava kao sustava koji se može izvoditi na velikom broju uređaja različitih proizvođača te na uređajima različitih namjena zvuči odlično, implementacija takvog operacijskog sustava nailazi na velike probleme sa inačicama operacijskog sustava koje se izvode na različitim uređajima.

Faktor prihvaćenosti (engl. *adoption rate*) nam govori koliki postotak uređaja vrti određenu inačicu sustava. Stariji, odnosno uređaji slabijih perforemansi često ne mogu koristiti najnoviju

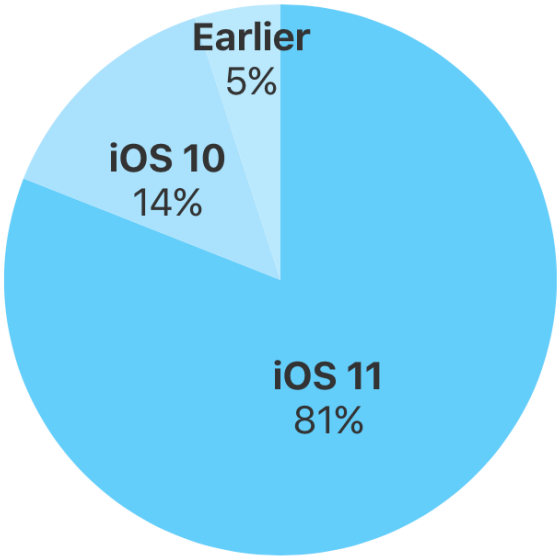
inačicu sustava, a zbog ranije navedene fragmentacije, mnogi uređaji nisu kompatibilni s novijim inačicama.

Najnoviju inačicu, Android Nougat, koristi samo 30.8% Android mobilnih telefona kao što je vidljivo na slici 3.3 [7]. Valjalo bi napomenuti kako je faktor prihvaćenosti Android OS-a iz godine u godinu ima blago pozitivni trend.

Android Name	Android Version	Usage Share
Nougat	7.0, 7.1	30.8%
Marshmallow	6.0	22.7%↓
Lollipop	5.0, 5.1	19.2%↓
Oreo	8.0, 8.1	14.6%↑
KitKat	4.4	8.6%↓
Jelly Bean	4.1.x, 4.2.x, 4.3.x	3.5%↓
Ice Cream Sandwich	4.0.3, 4.0.4	0.3%↓
Gingerbread	2.3.3 to 2.3.7	0.3%↑

Sl.3.3. *Faktor prihvaćenosti Android OS-a, rujan 2018.* [7]

S druge strane, iOS se izvodi isključivo na uređajima proizvedenim od strane tvrtke Apple. Zbog navedenog razloga, iOS ima izuzetno velik faktor prihvaćenosti. Najnoviju inačicu, iOS 11, pokreće 81% svih Apple uređaja.



Sl.3.4. *Faktor prihvaćenosti iOS operacijskog sustava, svibanj 2018. [7]*

3.1.3 Monetizacija

Iako Android OS koristi veći broj uređaja te Google Play trgovina broji mnogo veći broj preuzimanja aplikacija, monetizacija iOS App Store trgovine puno je uspješnija, što je i vidljivo na slici 3.5 [8].

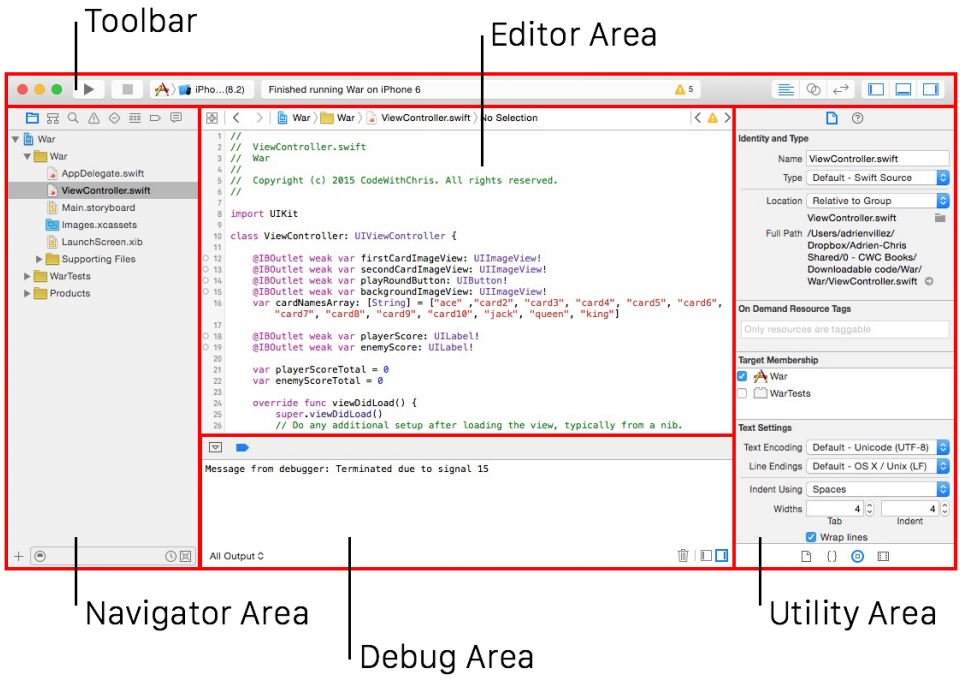
Sl.3.5. *Usporedba broja preuzimanja (lijevi graf) i zarade (desni graf), 2016. godina [8]*

3.2. Razvojno okruženje Xcode

Xcode je razvojno okruženje (engl. kratica *IDE*) kompatibilno s isključivo macOS operacijskim sustavom. Dizajniran od tvrtke Apple, a sadrži alate za razvoj macOS, iOS, watchOS i tvOS aplikacija. Prva inačica objavljena je 2003. godine, a zadnja stabilna inačica je 9.4 te je besplatna za preuzimanje u Mac App Store trgovini.

Xcode podržava ne veliki broj programskih jezika koji služe za razvoj navedenih aplikacija. Programski jezici koje Xcode podržava su: C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit (Rez) i Swift.

Xcode sučelje vrlo je intuitivno i kvalitetno dizajnirano. Ranije inačice Xcode-a nisu se mogle pohvaliti stabilnošću i performansama, no novije su znatno bolje i unaprijeđene. Samo sučelje sastoji se od pet osnovnih dijelova: alatna traka (engl. *Toolbar*), navigator (engl. *Navigator Area*), urednik koda (engl. *Editor Area*), traka sa značajkama (engl. *Utility Area*), traka sa alatom za detekciju grešaka (engl. *Debug Area*), a ti dijelovi naznačeni su na slici 3.6 [9].



Sl.3.6. *Naznačeni dijelovi sučelja razvojnog sučelja Xcode [9]*

3.3. Swift i Objective-C

Swift i Objective-C primarni su jezici za razvoj iOS aplikacija. Swift je novi jezik čija je svrha zamijeniti Objective-C koji dizajnom i funkcionalnostima zaostaje za suvremenim jezicima.

3.3.1. Objective-C

Objective-C je objektno orijentirani programski jezik generalne namjene koji jeziku C dodaje stil pisanja sličan onome u programskom jeziku Smalltalk. Nastao je u ranim 1980-tim, a bio je

primarni jezik tvrtke NeXT i njenog operacijskog sustava NeXTSTEP čiji su derivati operacijski sustavi macOS i iOS. Objective-C, prije nastanka programskog jezika Swift, bio je primarni jezik tvrtke Apple u kreaciji macOS i iOS operacijskih sustava te njihovih alata i programskih sučelja Cocoa i Cocoa Touch [10].

3.3.2. Swift

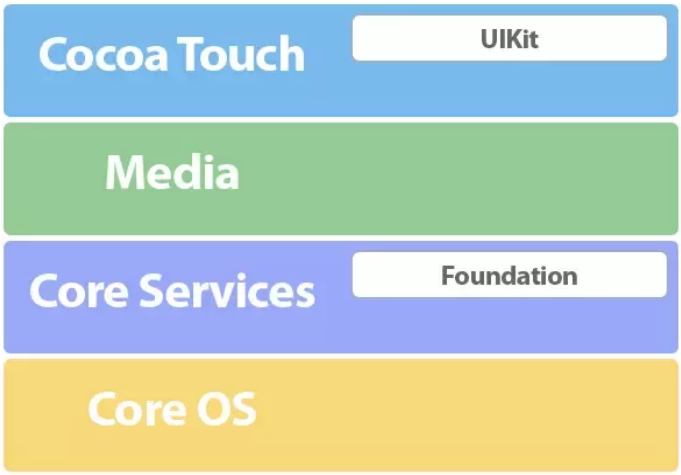
Swift je višeparadigmatski programski jezik opće namjene razvijen od tvrtke Apple. Prva stabilna inačica izdana je 2014. Godine. Glavni motiv programskog jezika Swift je zamijeniti Objective-C kao primarni jezik za razvoj programa na iOS, macOS, watchOS i tvOS platformama. Swift se također izvršava na Linux operacijskom sustavu. Swift je dizajniran za rad s programskim sučeljima Cocoa i Cocoa Touch te s postojećim Objective-C i C++ kodom pisanim za proizvode tvrtke Apple.

Izgrađen je na LLVM kompajleru čiji je kod također otvoren. Swift, kao i Objective-C, za optimizaciju memorije (engl. *Garbage collection*) koristi ARC (engl. *Automatic reference counting* - automatsko brojanje referenci), algoritam koji je vrlo brz, no zahtjeva eksplicitno navođenje slabih i jakih referenci radi eliminacije ciklusa referenci (engl. *Reference cycle*) koje ostali programski jezici rješavaju uvođenjem dodatnih algoritama u svoju optimizaciju memorije. Također, Swift ima automatsko inferiranje tipova (engl. *Automatic type inference*) po uzoru na jezike iz porodice ML (engl. *Meta Language*, Meta jezik) jezika [11].

3.4. Paradigma i arhitektura

3.4.1. Arhitektura iOS operacijskog sustava

Arhitektura iOS sustava sastoji se od četiri sloja: Cocoa Touch, Media, Core Services i Core OS što je vidljivo na slici 3.7 [12].



Sl.3.7. *Grafički prikaz arhitekturalnih slojeva iOS operacijskog sustava* [12]

Cocoa Touch

Cocoa Touch je sloj koji sadrži mnoštvo programskih okvira koji definiraju izgled aplikacije i korisničkog sučelja. Također, sloj pruža osnovnu infrastrukturu uključujući tehnologije kao što su sposobnost obavljanja više radnji (engl. *multitasking*), detekciju dodira, notifikacije (*push notifications*) i mnoge druge usluge sustava više razine.

Media

Sloj tehnologija koje upravljaju grafiku, audio i video, a namijenjene su kako bi ih razvojni programeri koristili pri implementaciji multimedijalnih iskustava unutar aplikacije.

Core Services

Usluge jezgre (engl. *Core Services*) je sloj koji sadrži esencijalne usluge sustava. Glavna usluga su *Core Foundation* i *Foundation Frameworks* koji definiraju osnovne tipove i alate koje sve aplikacije koriste. Također, sadrži tehnologije koje podržavaju značajke kao što su geolokacija, iCloud sustav, podrška za društvene mreže, povezivanje s mrežom i sl.

Core OS

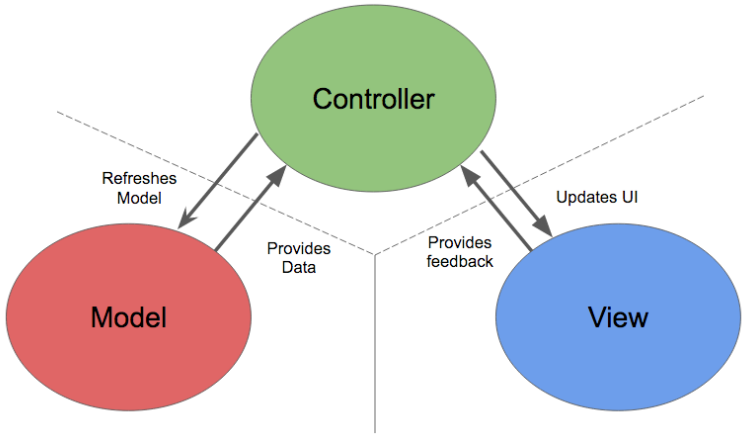
Core OS je sloj koji sadrži usluge kao što su sigurnost, autentikacija i bluetooth podrška.

3.4.2 Arhitektura iOS i macOS aplikacija

Arhitektura iOS i macOs aplikacija ovisi o razvojnom programeru, no najčešće, arhitekture iOS i macOS aplikacija koriste arhitekturalne uzorke MVC i MVVM. Od poznatijih arhitekturalnih uzoraka važno je spomenuti i MVP, Viper te Rx.

MVC

MVC je kratica od riječi *Model - View - Controller*. Arhitekturni uzorak MVC dijeli logiku aplikacije u tri različite skupine, model, pogled i upravljač. Na slici 3.8. [13] vidljiv je grafički prikaz MVC uzorka s dodatnim objašnjenjima.



Sl.3.8. *Grafički prikaz arhitekturalnog uzorka MVC.* [13]

Model je dio aplikacijske logike koji sadrži poslovnu logiku i obrađuje podatke. Često ga čine usluga i različitih uloga; komunikacija sa poslužiteljem, komuniciranje s lokalnom bazom i slično.

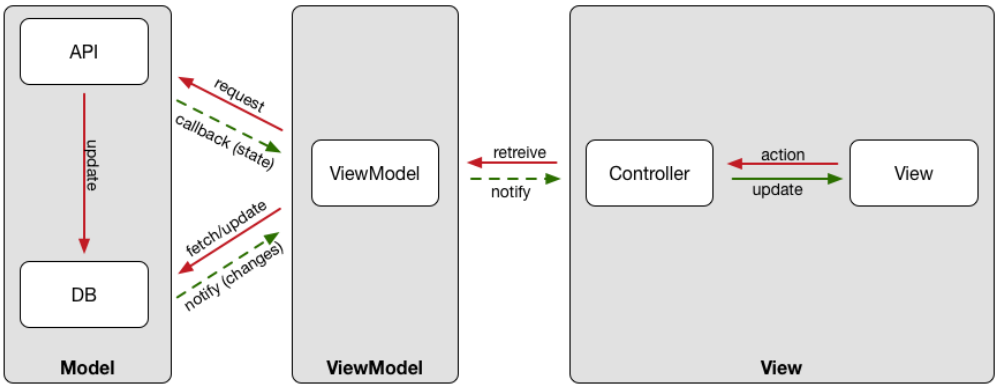
Pogled (engl. *View*) označava dio aplikacijske logike koji opisuje izgled, korisničko sučelje te interakciju korisnika sa sučeljem.

Upravljač (engl. *Controller*) označava dio aplikacijske logike koji "servira", odnosno prikazuje različite poglede korisničkog sučelja te ih popunjava podacima dobivenim iz modela. On je posrednik između pogleda i modela. Strogi MVC zabranjuje komunikaciju modela i pogleda te se svaka komunikacija vrši preko upravljača.

MVVM

MVVM, kratica od *Model - View - ViewModel* (engl. model, pogled, model u vlasništvu pogleda) je nadogradnja arhitekturalnog uzorka MVC. MVVM je uz MVC najpopularniji uzorak koji se koristi u većim i složenijim aplikacijama.
Ključna razlika MVVM i MVC je u tome što u MVVM arhitekturi upravljač nije zasebna struktura, već se smatra dijelom pogleda, a svaki pogled sadrži vlastiti model koji posjeduje stanje aplikacije.

Na taj način, glavni model nije opterećen globalnim stanjem aplikacije, a upravljač je rasterećen stanjem te sadrži isključivo logiku vezanu za korisničko sučelje (engl. UI - *User interface*). Na slici 3.9 [14], grafički je prikazan MVVM uzorak.



Sl.3.9. *Grafički prikaz MVVM arhitekturalnog uzorka* [14]

3.5. Alati iOS razvoja

3.5.1. iOS Simulator

iOS simulator služi za testiranje aplikacija tijekom razvoja. Prednosti testiranja aplikacija na simulatoru u odnosu na fizički uređaj su brzina i naravno činjenica da korisnik ne mora posjedovati, odnosno koristiti fizički uređaj. Simulator ima gotovo sve potrebne alate i funkcionalnosti pravog fizičkog uređaja te je većinu značajki moguće testirati koristeći simulator, bez fizičkog uređaja. Također, simulirati je moguće sve podržane Apple uređaje. Simulator nema podršku za kameru, programski okvir Metal, notifikacije i još nekolicinu značajki, pa je takve značajke potrebno testirati na fizičkom uređaju. Na slici 3.10 [15] vidljiv je prikaz iOS simulatora koji simulira uređaj iPhone X.

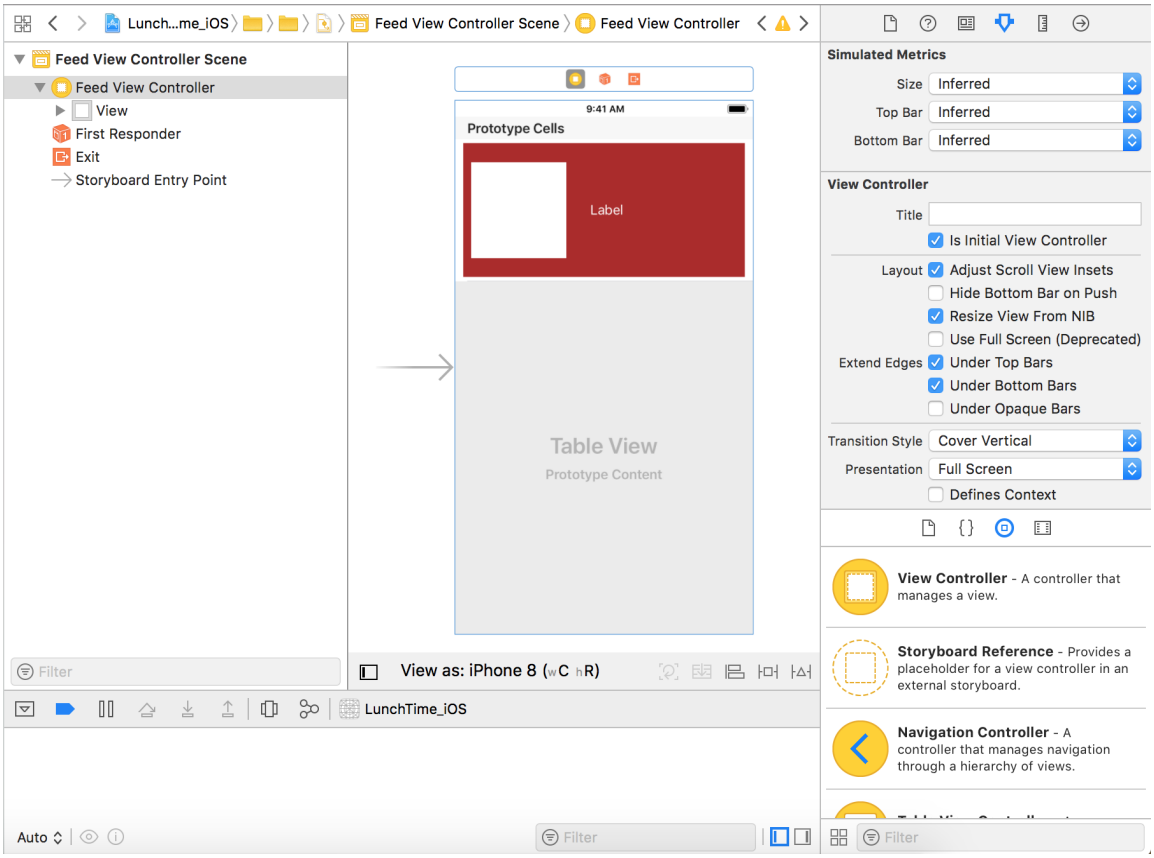
Sl.3.10. *iOS simulator* [15]

3.5.2. Cocoa Pods

Cocoa Pods je upravitelj biblioteka (engl. *dependency manager*) za Swift i Objective-C. Trenutno broji preko pedeset tisuća biblioteka i korišten je u više od tri milijuna aplikacija [16].

3.5.3. Interface Builder

Graditelj sučelja (engl. *Interface Builder*) ugrađen je u Xcode IDE. Olakšava izradu sučelja tako što omogućuje povuci i pusti (engl. *drag and drop*) funkcionalnost objektima na ekranu bez pisanja programskog koda. Prikazan je na slici 3.11.

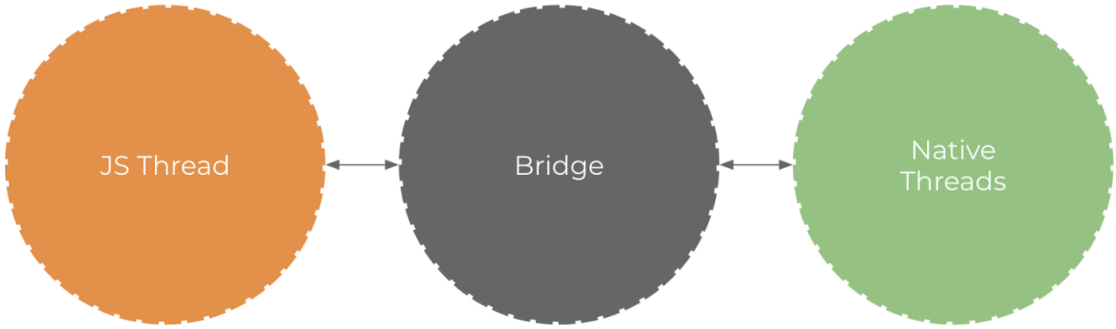


Sl.3.11. *Sučelje InterfaceBuilder-a*

4. REACT NATIVE RAZVOJ

React Native je tehnologija kreirana od tvrtke Facebook. Cilj tehnologije je kreirati mobilne aplikacije za Android i iOS uz pomoć React tehnologije. Za razliku od ostalih tehnologija namijenjenih za višeplatformski razvoj koje koriste skriveni web preglednik, produkt React Native tehnologije su nativne aplikacije visokih performansi.

Primarni jezik React Native tehnologije je JavaScript, pa je sama izvedba stapanja dvaju ekosustava vrlo kompleksna. Arhitektura React Nativa rješava ovaj problem na sljedeći način: umjesto prevođenja JavaScript koda u svaki nativni jezik, React Native koristi nativne prevoditelje koji prevode nativni kod u binarni, a JavaScript uz pomoć mosta (engl. *bridge*) napisanog u programskom jeziku C++ daje instrukcije u JSON formatu koje se stavljaju u red (engl. *queue*) te se po danim uputama kreiraju komponente nativnog jezika. Slika 4.1 [17] prikazuje komunikaciju nativnih dretvi sa JavaScript dretvom putem mosta.



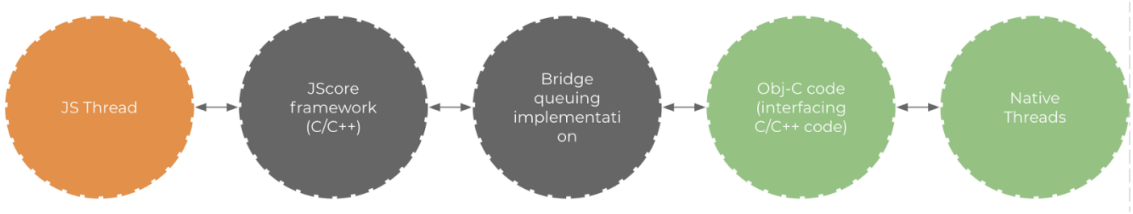
Sl.4.1. *Grafički prikaz komunikacije nativnih dretvi s JavaScript dretvom putem mosta* [17]

Most funkcionira u oba smjera (engl. *full duplex*) na gotovo identičan način na koji funkcioniraju brokeri poruka (engl. *message brokers*) te na taj način, JavaScript komunicira sa slušateljima događaja (engl. *event listeners*) [18]. Slika 4.2 prikazuje poruke poslane mostu od strane JavaScript dretve.

```
JS->N : UIManager.createView([2,"RCTView",1,{"flex":1,"pointerEvents":"box-none"}])
JS->N : UIManager.createView([3,"RCTView",1,{"pointerEvents":"box-none","flex":1}])
JS->N : UIManager.createView([4,"RCTView",1,{"flex":1,"justifyContent":"center","alignItems":"center","backgroundColor":4294311167}])
JS->N : UIManager.createView([5,"RCTText",1,{"accessible":true,"allowFontScaling":true,"ellipsizeMode":"tail","disabled":false}])
JS->N : UIManager.createView([6,"RCTRawText",1,{"text":"Hello world"}])
JS->N : UIManager.setChildren([5,[6]])
JS->N : UIManager.setChildren([4,[5]])
JS->N : UIManager.setChildren([3,[4]])
JS->N : UIManager.createView([7,"RCTView",1,{"position":"absolute"}])
JS->N : UIManager.setChildren([2,[3,7]])
JS->N : UIManager.setChildren([1,[2]])
```

Sl.4.2. *Primjer poruka i instrukcija poslanih putem mosta u smjeru JavaScript - nativni kod [17]*

Komunikacija između mosta (koji je napisan u programskom jeziku C++) i nativnog koda je jednostavnija pošto je programski jezik Objective-C ekstenzija programskog jezika C te dva jezika mogu komunicirati nativno. Na Android platformi, komunikacija se odvija uz pomoć Java Native Interface. Na slici 4.3 [17] prikazana je komunikacija nativne i JavaScript dretve s mostom.



Sl.4.3. *Grafički prikaz komunikacije nativnih i JavaScript dretve s mostom [17]*

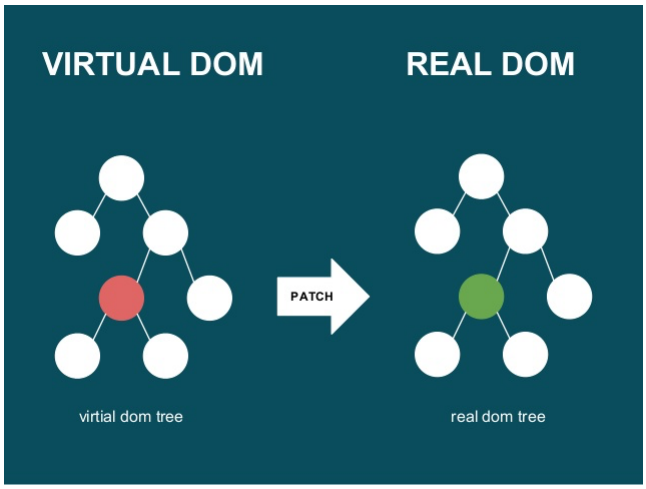
4.1. Tehnologija React

Prema [19], React je JavaScript biblioteka koja služi za izradu korisničkih sučelja. Održavana je od strane tvrtke Facebook i zajednice individualnih razvojnih programera te tvrtka. Svrha Reacta je izrada SPA, odnosno web aplikacija koje se izvode na isključivo jednoj stranici (engl. *Single Page App*)

Kreator React-a je Jordan Walke, programski inženjer u tvrtki Facebook. React je inspiriran XHP-om, programskim okvirom za PHP koji služi za izradu HTML komponenti. Prvi put je korišten 2011. godine na Facebook stranici, a kod mu je otvoren u svibnju 2013.

React je dizajniran na način da je svaka funkcionalnost, odnosno dio sučelja, modulariziran unutar komponenti. Komponente se potom slažu i jednosmjerno povezuju podacima (engl. *Unidirectional data binding*).

React, kao i svaka druga JavaScript UI biblioteka manipulira DOM (*Document Object Model*), no različit je po tome što React kreira i virtualni DOM koji se izvršava u RAM memoriji, pa pri osvježavanju sučelja naprednim algoritmima uspoređuje virtualni i stvarni DOM. Napredni algoritmi prolaze oba stabla te ih uspoređuju, pa ažuriraju stvarni DOM u minimalnom broju instrukcija što uvelike povećava performanse aplikacije. Na slici 4.4 [19] prikazana su oba DOM stabla te njihova usporedba prije osvježavanja stranice.



Sl.4.4. *Umjesto da pri ažuriranju browser renderira cijelo stablo, on ažurira samo promijenjeni dio [19]*

4.2. Node.JS ekosustav

Node.JS je višeplatformsko JavaScript okruženje otvorenog koda koje se izvodi tijekom pokretanja (engl. *run-time*). Kroz povijest, JavaScript se koristio primarno za klijentske skripte koje su se izvršavale na klijentu, odnosno unutar pretraživača.

Node.js omogućuje korištenje JavaScript programskog jezika za pisanje alata komandnog prozora (engl. *Command Line tools*) te za pisanje skripti koje se izvršavaju na poslužitelju (engl. *server-side scripting*). Skripte na poslužitelju kreiraju sadržaj prije nego je stranica poslana klijentu.

NPM (*Node Package Manager*) je upravitelj biblioteka koji je integriran u Node.js. Trenutno je najveći upravitelj biblioteka na svijetu i broji preko 700 000 biblioteka [20].

4.3. JavaScript

JavaScript je visoko apstraktni, skriptni interpreterski jezik koji je poznat kao jedan od tri tehnologije koje koristi World Wide Web standard. Jezik se karakterizira kao dinamički, *weakly-typed*, prototipno orijentiran te više paradigmatski jer podržava različite paradigme kao što su funkcijska, reaktivna, imperativna, i objektno orijentirana. U početku je dizajniran za izvođenje u pregledniku, ali zbog velikog zanimanja zajednice, kreirana su i okruženja za poslužitelj (engl. *server*). Novi standard jezika zove se ECMAScript, no zajednica je zadržala stari naziv [21].

4.4. ReasonML

Novi programski jezik, odnosno dijalekt programskog jezika OCaml. Kreiran kako bi zamijenio JavaScript na klijentskoj strani. Sintaksa, zbog faktora prihvaćenosti, uvelike podsjeća na JavaScript, no AST - apstraktno sintaksno stablo (engl. *Abstract syntax tree*) je AST jezika OCaml.

Kompatibilnost i komunikacija između JavaScripta i OCaml ekosustava vrši se preko BuckleScript prevoditelja koji Ocaml prevodi u JavaScript kod koji je čitljiv čovjeku. [22]

4.5. Upravljanje stanjem

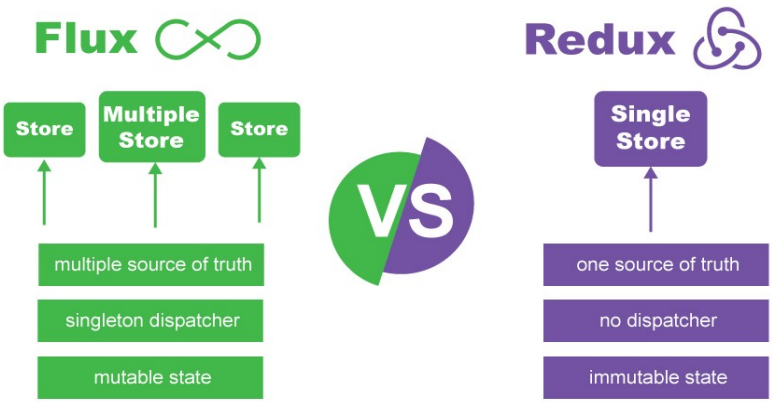
Često, u velikim SPA aplikacijama, arhitektura postaje izuzetno kompleksna, pa standardni arhitekturani uzorci kao što su MVC i MVVM nisu dovoljni. Navedeni problem riješen je alatima koji automatiziraju prosljeđivanje podataka i upravljaju stanjem aplikacije. Neki od poznatih takvih alata su: Redux, Flux, MobX i drugi.

4.5.1. Flux

Facebookov alat koji je vrlo sličan MVVM arhitekturalnom uzorku. Nedavno je napušten zbog Redux alata koji je napredniji i jednostavniji za korištenje [23].

4.5.2. Redux

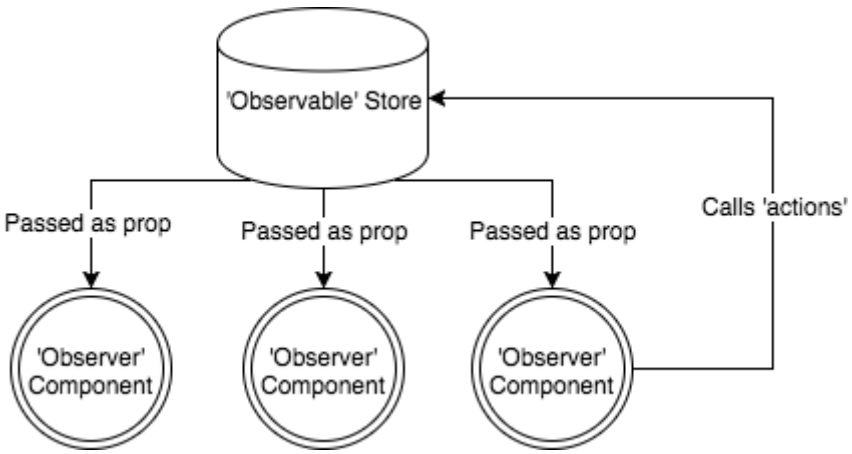
Slično kao i Flux, Redux stanje aplikacije drži u spremištu (engl. *store*). Unutar UI sloja pozivaju se akcije koje potom ažuriraju spremište te potom osvježe UI sloj s novim podacima. Autor Redux-a je Dan Abramov, a inspiraciju za Redux vidio je u Elm arhitekturi [24]. Slika 4.5 prikazuje usporedbu arhitekture Flux i Redux alata.



Sl.4.5. *Usporedba Flux-a i Redux-a* [24]

4.5.3. MobX

MobX također sprema aplikacije u spremište, ali funkcionira pomoću promatrač - objekt arhitekturalnom uzorku (engl. *observer - observable pattern*). Noviji je i manje je robustan od prethodno navedenih te je upravo to uzrok nagle popularnosti i velikog broja aplikacija koje ga koriste [25]. Slika 4.6 prikazuje arhitekturu MobX alata.



Sl.4.6. *Arhitektura MobX-a* [25]

4.6. Alati kao oslonac

Za razliku od nativnog razvoja, React Native ovisi o mnoštvu različitih alata često održavanih od različitih tvrtki. Cilj većine alata je omogućiti što jednostavniji proces postavljanja projekta te što

jednostavniji proces objavljivanja projekta. React Native ima 3 stupnja automatizacije razvoja: Expo klijent, CRNA, CRNA s izbačenim postavkama.

4.6.1. Expo klijent

Expo klijent je projekt treće strane koji je najpopularniji izbor pri odabiru alata i okruženja u React Native zajednici. Expo automatizira cijeli proces objave aplikacije, od testnog poslužitelja do trgovine. Također, nudi mogućnosti kao što su besplatni poslužitelji i OTA (engl. *over the air*) ažuriranja koja su praktična jer ne zahtijevaju dugački proces ažuriranja na trgovine [26]. Expo također nudi JavaScript biblioteke koje pokrivaju približno 90% mogućnosti koje nativne platforme imaju, bez da programer mora pisati nativni kod. Programski okvir zove se ExpoKit.

4.6.2. CRNA

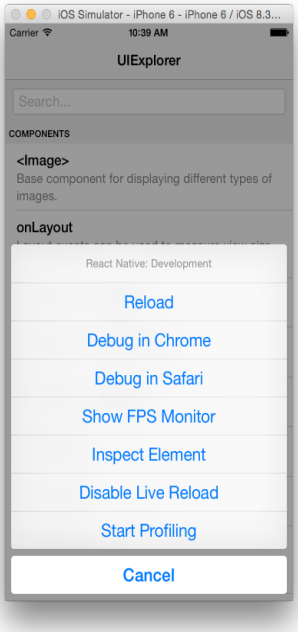
CRNA je kratica koja stoji za: "stvori React Native aplikaciju" (engl. *Create React Native App*). Održavana je od strane zajednice i nudi kompletno automatizirane postavke i alate koji su potrebni za izradu aplikacije. Također, u ovom načinu rada, moguće je koristiti ExpoKit, no proces objave mora se implementirati ručno.

4.6.3. CRNA s izbačenim postavkama

Ukoliko aplikacija zahtijeva nativni kod, miješanje nativnog i JavaScript koda ili tim inženjera želi koristiti React Native u samo jednom od dijelova aplikacije, tada može izbaciti dodatne postavke u kojima se kreiraju Android i iOS nativna okruženja u kojima je moguće pisati nativni kod.

4.6.4. Ostali alati

React Native poznat je po kvalitetnim razvojnim alatima koje nativni razvoj ne može omogućiti. Samo neki od njih su: brzo osvježavanje (engl. *hot reloading*), ispitivač UI elemenata (engl. *element inspector*) koji je napravljen po uzoru na istoimeni alat u web razvoju, udaljeno otklanjanje pogreški (engl. *remote debugging*) i ostali. Na slici 4.7 prikazani su alati koji su ugrađeni u React Native unutar iOS simulatora.



Sl.4.7. *Prikaz navedenih alata u iOS simulatoru*

5. MJERENJA I USPOREDBE IOS I REACT NATIVE TEHNOLOGIJA

React Native nastao je kao pokušaj boljeg rješenja za izradu iOS aplikacija. Facebook je tek kasnije uveo podršku za Android. Koliko je React Native bolje rješenje od iOS-a tema je mnogih rasprava i debata unutar zajednice, kako programera, tako i menadžera i investitora. Mnoge rasprave uglavnom koriste subjektivne argumente, pa je ideja usporedbe donijeti konkretan sud potkrepljen mjerenjima i činjenicama.

5.1. Postavke mjerenja i usporedbe

Dvije tehnologije, u daljnjem tekstu, kvalitativno su uspoređene su po sljedećim kriterijima:

- Opseg značajki koje je moguće izvesti
- Ponovna upotrebljivost koda (engl. *reuseability*)
- Implementacija programskih sučelja (API) treće strane
- Komunikacija sa poslužiteljem
- Stabilnost ekosustava
- Programski jezik
- Arhitektura i dizajn
- Vrijeme potrebno za izradu
- Cijena izrade
- Alati
- Objavljivanje

Kvantitativna usporedba, odnosno mjerenja, obavljena su po sljedećim kriterijima:

- CPU opterećenje
- GPU opterećenje
- zauzeće memorije

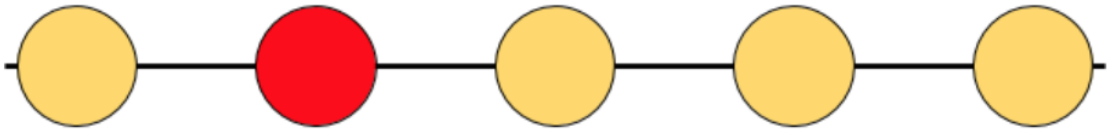
Kriteriji su uspoređeni i izmjereni na dvije aplikacije koje izgledaju isto te imaju isti skup značajki. Jedna aplikacija izrađena je nativno, a druga uz pomoć React Native tehnologije.

Niti u jednoj aplikaciji nisu korištene UI biblioteke treće strane radi izbjegavanja greški u kvalitativnoj usporedbi te mjerenjima.

5.2. Tehnika kvalitativne usporedbe

Usporedba po navedenim kriterijima kvantizirana je po Likertovoj skali. Umjesto postavljanja jedne od tehnologija kao referentne i uspoređivanja svakog segmenta od negativne prema pozitivnoj vrijednosti, skala je modificirana na sljedeći način: ukoliko su dana rješenja u obje tehnologije približno jednako kvalitetna, ocjena usporedbe biti će "Usporedivo".

Ukoliko jedna od tehnologija bolje rješava dani problem, ocjena će biti "Bolje". Ukoliko se dani problem jednostavno rješava u jednoj od tehnologija, a s poteškoćama, ili ne postoji rješenje za problem u drugoj tehnologiji, ocjena usporedbe biti će "znatno bolje". Na slici 5.1. prikazana je opisana skala.



znatno bolje (iOS)	Bolje (iOS)	usporedivo	Bolje (RN)	znatno bolje (RN)

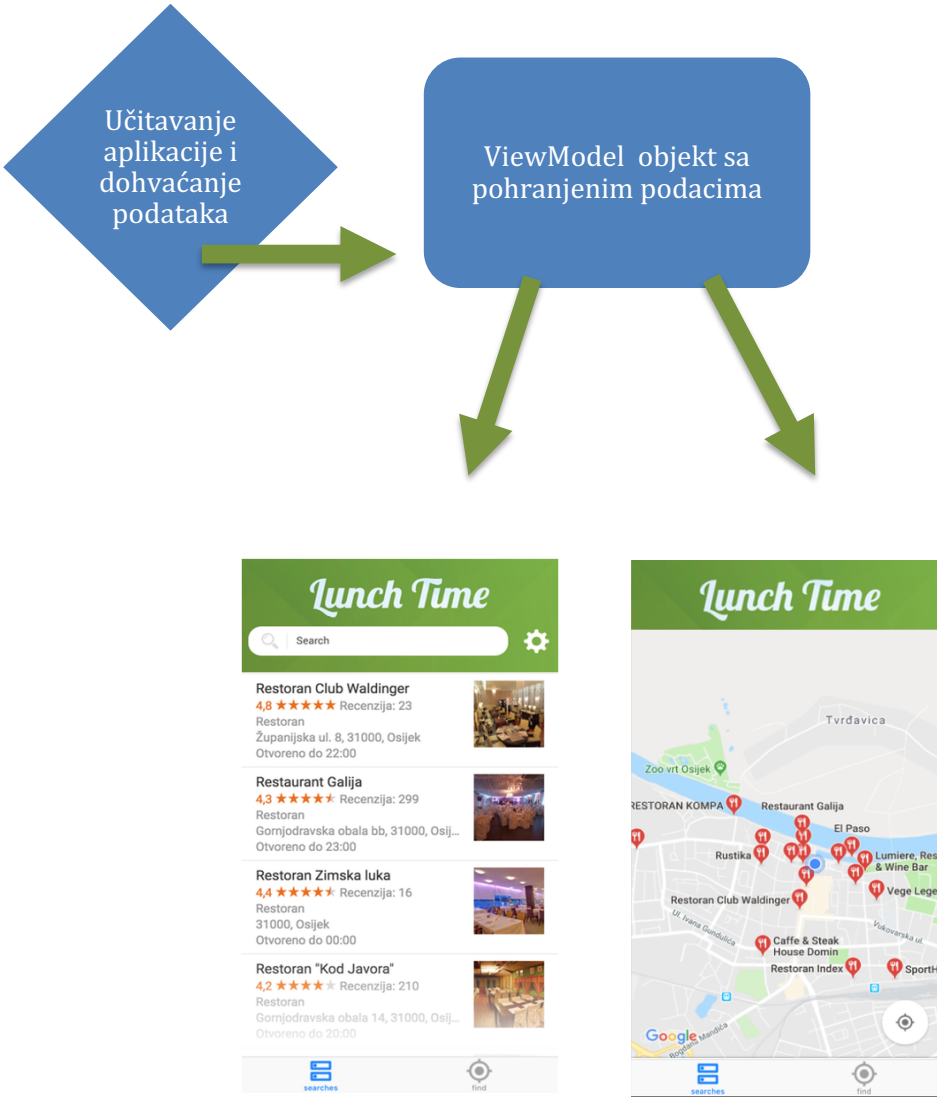
Sl.5.1 *Grafički prikaz skale po kojoj je izvršena usporedba*

5.3. Primjer aplikacije na kojoj je izvršena usporedba

Aplikacija koja je testirana zove se *LunchTime*. Sačinjena je od tri prikaza od kojih svaki ima drugačiju zadaću. Popis restorana s dinamičkom listom, Google karta te modalni prikaz postavki.

Dana tri prikaza odabrana su s namjerom mjerenja brzine, pošto su tri, gore navedene značajke, vrlo važne za usporedbu jer svaka od njih zahtjeva velike optimizacije unutar tehnologije. Na taj način, vidjet će se koja tehnologija bolje optimizirana, odnosno ima bolje performanse.

Pri učitavanju prvog prikaza, dinamičke liste koja prikazuje popis restorana, izvršava se zahtjev koji sa Google poslužitelja dohvaća popis restorana koji su sortirani s obzirom na blizinu, recenzije posjetitelja i radno vrijeme. Dohvaćena lista pohranjuje se u ViewModel objekt, odnosno MobX spremište koje prosljeđuje podatke dinamičkoj listi te prikazu karte, kao što je vidljivo na slici 5.2.



Sl.5.2. *Grafički prikaz toka aplikacije*

Pri odabiru prikaza karte u navigaciji, od liste koja sadrži podatke o restoranima, kreira se nova lista objekata koji sadrže dva podatka: koordinate te ime restorana. Na temelju nove liste, na karti se iscrtavaju POI ikonice (engl. Point of interest). Također, u novu listu moguće je proslijediti dodatan sadržaj koji se može prikazati pritiskom na pojedinu ikonicu.

5.3.1. iOS inačica aplikacije

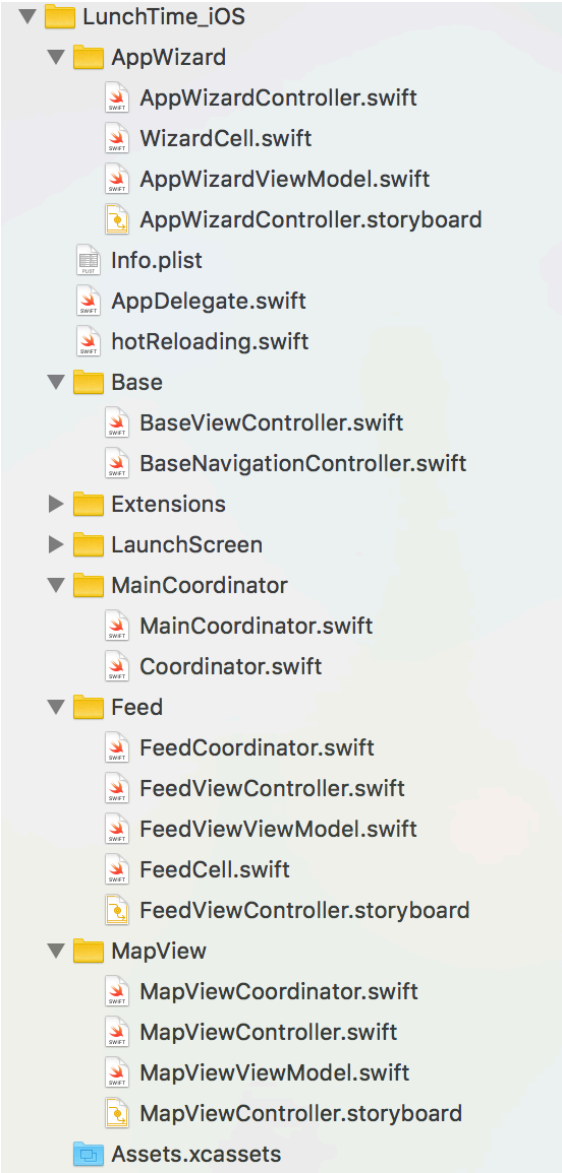
iOS inačica aplikacije izrađena je u Xcode 9. Arhitektura korištena u aplikaciji je MVVM + C, gdje slovo C označava navigacijske objekte (eng. *coordinators*). Svaki prikaz stavljen je u odvojen *Storyboard* (dio *InterfaceBuilder* alata), a navigaciju između njih vrši navigacijski objekt. Za komunikaciju s vanjskim sučeljima (API) korištena je biblioteka *Alamofire*.

Stablo podataka (Sl 5.3) iOS aplikacije strukturirano je na način da svaki prikaz u svojoj odgovarajućoj mapi sadrži sljedeće datoteke:

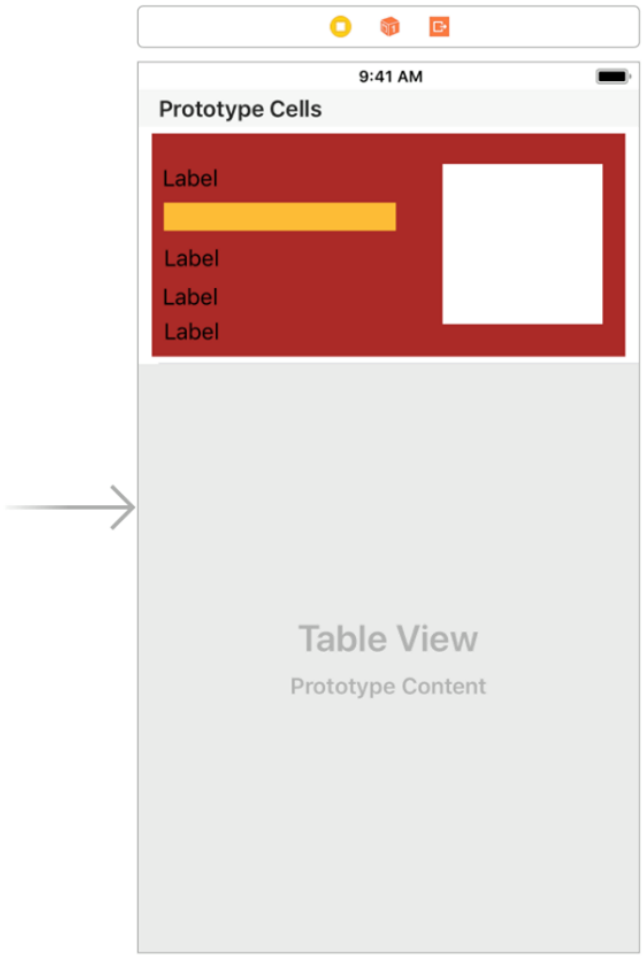
- *ViewController.storyboard*
- *ViewController.swift*
- *ViewModel.swift*
- *Coordinator.swift*

ViewController.storyboard je datoteka koji se otvara uz pomoć *InterfaceBuilder* alata koji je dio *Xcode* okruženja. U njemu je opisan prikaz, njegove glavne komponente te njihov omjer i položaj na zaslonu. Slika 5.4 prikazuje prikaz .storyboard datoteke u InterfaceBuilder sučelju.

U opisanoj aplikaciji, u *.storyboard* datoteci opisuje se okvir i položaj komponenti, a ostatak UI logike, kao što su boja i oblik teksta, vizualni efekti i dizajn, nalazi se u datoteci *ViewController.swift.*



Sl.5.3. *Stablo podataka iOS aplikacije*



Sl.5.4. *Prikaz .storyboard datoteke u InterfaceBuilder sučelju*

ViewController.swift datoteka sadrži UI logiku pripadajućeg prikaza. Također, moguće je cijelu UI logiku pisati programatski, u *ViewController.swift*, bez korištenja *.storyboard* datoteke. *ViewController.swift* povezan je s komponentama .storyboard datoteke putem ispusta (eng. *Outlet*) kao što je vidljivo na slici 5.5.

```
final class MapViewController: BaseViewController {

    //ispust
    @IBOutlet weak var mapView: GMSMapView!

    //referenca modela
    var viewModel: MapViewModel!
```

Sl.5.5. *Ispust i referenca na pripadajući model*

ViewController.swift, po MVVM uzorku, „vlasnik“ je *ViewModela* (Sl. 5.5), odnosno sadrži referencu na objekt *ViewModel*. Samim tim, zbog ranije opisanog ARC algoritma, dealociranjem kontroler objekta, dealocira se i objekt pripadajućeg modela pošto broj jakih referenci na objekt modela postaje nula.

ViewModel.swift datoteka sadrži poslovnu logiku specifičnu za pripadajući prikaz. Ukoliko je arhitektura zahtjevnija, moguće je apstrahirati objekt modela pomoću javnog sučelja. Opisana aplikacija ne sadrži dva ili više identičnih prikaza s različitom poslovnom logikom, pa navedena apstrakcija nije implementirana. Objekt kontrolera ima pristup svim javnim metodama objekta modela te je na taj način izvršeno odjeljivanje zadaća (engl. *decoupling*). Na slici 5.6 prikazan je programski kod datoteke ViewModel.swift.

```
1  //
2  //  MapViewViewModel.swift
3  //  LunchTime_iOS
4  //
5  //  Created by Blaz on 15/08/2018.
6  //  Copyright © 2018 Blaz. All rights reserved.
7  //
8
9
10 import UIKit
11
12
13
14 class MapViewViewModel {
15
16     //servisi
17     let networkingService: Alamofire
18
19     //Stanje, popis pronadenih lokacija
20     var places:[Place]! = []
21
22     //Konstruktor
23     init(networkingService: NetworkingServiceProtocol) {
24         self.networkingService = networkingService
25     }
26
27     func loadPlaces(_ location: String, radius: Int) -> [Place] {
28
29         var params = [
30             "location" : location,
31             "radius": radius
32             ] as [String : Any]
33
34         networkingService.makeRequest(with: Routes.placesURL, method: .get, parameters: params,
               encoding: URLEncoding.default, completion: { places in
35             self.places = places
36         })
37
38     }
39 }
40
```

Sl. 5.6. *Implementacija modela koji pripada prikazu koji sadrži kartu*

Više modela često imaju jednake zadaće, primjerice dohvaćanje podataka s mreže, pa se takve funkcionalnosti dodatno modulariziraju u modele koji su zajednički svim prikazima. Takvi modeli često se nazivaju uslugama (engl. *services*). Takve se usluge uvijek apstrahiraju javnim sučeljima jer uglavnom ovise o bibliotekama i kodu treće strane (engl. *dependencies*) koji bi u praksi morao biti brzo i lako zamjenjiv. Primjer modela usluge vidljiv je na slici 5.7.

```
//
//  NetworkingServiceProtocol.swift
//  iOS-AppTemplate
//
//  Created by Blaž Jurišić on 17/05/2017.
//  Copyright © 2017 Blaž Jurišić. All rights reserved.
//
import Foundation
import Alamofire

protocol NetworkingServiceProtocol: class {

}

extension NetworkingServiceProtocol {


    internal func makeRequest(with route: String = "", method: HTTPMethod = .get, parameters: Parameters, encoding:
        ParameterEncoding = URLEncoding.default, headers customHeaders: HTTPHeaders? = nil, completion: @escaping
        (DataResponse<Any>)-> Void) {

        let path = AppConfig.baseUrl + route
        let headers = prepareHeaders(headers: customHeaders)

        let req = request(path, method: method, parameters: nil, encoding: encoding, headers: headers)
            .responseString { response in
                print(response)
            }
            .responseJSON { response in

                completion(response)
        }
        print(req)
    }
}
```

Sl.5.7. *Primjer javnog sučelja za komunikaciju s mrežom te usluga koji implementira sučelje i koristi biblioteku Alamofire*

Coordinator.swift je datoteka koja sadrži implementaciju klase koordinatorskog objekta. Koordinator objekt zadužen je za navigaciju prikaza sličnih zadaća. Također, koordinatorski objekt stvara kontroler i pripadajući model te pridružuje model kontroleru (Sl. 5.8). Pri stvaranju pripadajućeg modela, konstruktoru modela predaje usluge potrebne za rad. U konkretnom primjeru (Sl. 5.7) *MapViewModel* šalje upit na mrežu pa mu koordinator predaje instancu mrežne usluge.

```swift
//
//  MapViewCoordinator.swift
//  LunchTime_iOS
//
//  Created by Blaz on 17/08/2018.
//  Copyright © 2018 Blaz. All rights reserved.
//

import UIKit

class MapViewCoordinator: Coordinator {

    private var navigationController = BaseNavigationController()

    @discardableResult
    func start() -> UIViewController {

        //instanca kontrolera
        let vc = MapViewController.instance()

        //instanca modela
        let viewModel = MapViewViewModel(ServiceLocator.networkingService)

        //dodjela reference modela kontroleru
        vc.viewModel = viewModel

        //prikaz se dodaje u navigaciju
        navigationController.viewControllers = [vc]
        vc.navigationBarDisplayMode = .always
        return navigationController

    }

}
```

Sl.5.8. *Primjer klase MapViewCoordinator objekta*

5.3.2. React Native inačica aplikacije

React native inačica aplikacije izrađena je uz pomoć *Expo* klijenta. Uz *JavaScript*, korišten je i programski jezik - alat *Flow*. Stanje je upravljano *MobX* upraviteljem stanja. Za komunikaciju s vanjskim sučeljima (API) korištena je biblioteka *Axios*. Arhitektura *React* aplikacije, zbog suvremenog dizajna, puno je slobodnija i manje definirana. *React* komponente čiji se primjeri nalaze na slikama 5.9 i 5.10 formiraju prikaz što je ekvivalent kontroleru u iOS aplikaciji.

```
const buttonComponent = (props) => (

    < Button
        rounded
        onPress={() => nekaMetoda()}
        style={{
            marginTop: 20,
            width: 150,
            height: 35,
            backgroundColor: '#3f51b5',
            justifyContent: 'center',
            alignSelf: 'center'
        }}
    >
        <Text> Naslov </Text>
    </Button >
)
```

Sl. 5.9. *Primjer komponente koja definira tipku na sučelju*

```
const modalComponent = (props) => (
    <Modal
        transparent={true}
        onRequestClose={() => nekaMetoda()}
        visible
    >

    <Text> Ovaj tekst prikazan je na modalnom prikazu</Text>
    </Modal>
)
```

Sl.5.10. *Primjer komponente koja definira modalni prikaz*

MobX upravlja stanjem, a akcije ažuriranja stanja. U slučaju da akcije iz više spremišta vrše iste funkcionalnosti (kao kod primjera *Alamofire* usluge u iOS aplikaciji), usluge se mogu dodatno modularizirati. Opisana aplikacija ima samo jedno spremište. Postoje tri nužna dekoratora (dekorator je drugačija sintaksa za funkciju u inačici budućeg ECMA standarda) koje *Mobx* pruža.

- Varijabla koji se može promatrati (engl. *observable*) se postavlja na vrijednost koja predstavlja stanje i potrebna je informacija promatračima (engl. *observers*) o njenoj promjeni.

- Promatrač(engl. *observer*) označava React komponentu koja prima obavijesti o promjenama te na promjenu poziva funkciju *render*() koja osvježava prikaz novim podacima.
- Akcija *(*engl. *action)* označava akcije, metode koje ažuriraju stanje.

Na slici 5.11 prikazan je programski kod MobX spremišta.

```
export default class GooglePlacesStore {
    // stanje
    @observable placesState: Array<Marker> = []
    @observable isErrorPresent: boolean = false
    @observable region: string = ''

    constructor() {
        autorun(() => console.log(this.places))
    }

    //metode za dohvacanje
    @computed get places(): Array<Marker> {
        return this.placesState
    }

    @computed placeWithId(id: string): ?Marker {
        return this.placesState.find(place => place.id === id)
    }

    // akcije
    @action requestPlaces(location: string, radius: number) {
        axios.get(Constants.PlacesUrl, {
            params: {
                location: location,
                radius: radius
            }
        })
            .then(response => this.placesState = response.data)
            .catch(error => this.isErrorPresent = true)
    }
}
```

Sl.5.11. *Implementacija Mobx spremišta*

Implementacija karte jednostavnija je od implementacije nativnim putem. Opisana aplikacija koristi *ExpoKit* biblioteku koja sadržava gotovu komponentu karte. Karti se predaje trenutna lokacija uređaja, a POI oznake predaju se kao lista djece (eng. *children*) komponente karta. (Sl. 5.12).

```
export const mapComponent = (props) => (
  <MapView
      region={store.region}
      onRegionChange={region => store.region = region}
  >
      {store.places.map(place => (
          <Marker
              coordinate={place.latlng}
              title={place.title}
              description={place.description}
          />
      ))}
  </MapView>
)
```

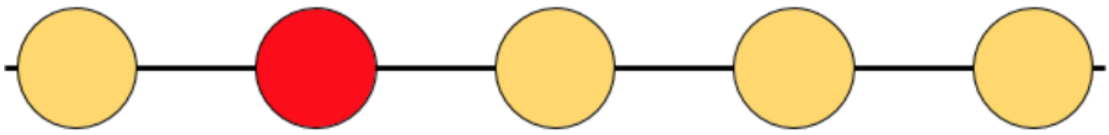
Sl.5.12. *Implementacija karte u React Native aplikaciji*

5.4. Rezultati usporedbe

5.4.1. Kvalitativna usporedba

Opseg značajki koje je moguće izvesti

React Native s vremenom podržava sve više značajki iOS sustava. Trenutno, ExpoKit podržava većinu iOS funkcionalnosti poput ArKit-a i notifikacija. Svega nekolicina značajki nije podržana, a njih je tada potrebno implementirati nativno, izbacivanjem postavki. Na slici 5.13. prikazana je skala sa ocjenom opsega značajki.



znatno bolje (iOS)	Bolje (iOS)	usporedivo	Bolje (RN)	znatno bolje (RN)

Sl.5.13 *Skala koja prikazuje ocjenu opsega značajki*

Ponovna upotrebljivost koda (engl. *reuseability*)

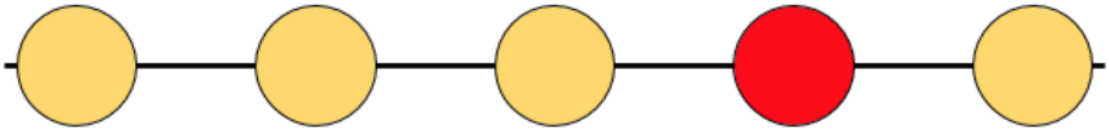
UI logika iOS-a teško se može ponovno koristiti. React Native je puno fleksibilniji i moguće je pisati bolji i kvalitetniji UI kod koji je spreman za ponovno korištenje u drugim projektima. Na slici 5.13. prikazana je skala sa ocjenom ponovne upotrebljivosti.

znatno bolje (iOS)	Bolje (iOS)	usporedivo	Bolje (RN)	znatno bolje (RN)

Sl.5.14 *Skala koja prikazuje ocjenu ponovne upotrebljivosti koda*

Implementacija programskih sučelja (API) treće strane

React Native, zahvaljujući NPM-u, ima puno više biblioteka i sučelja koje je moguće implementirati. Na slici 5.15. prikazana je skala sa ocjenom jednostavnosti implementacije programskih sučelja treće strane.

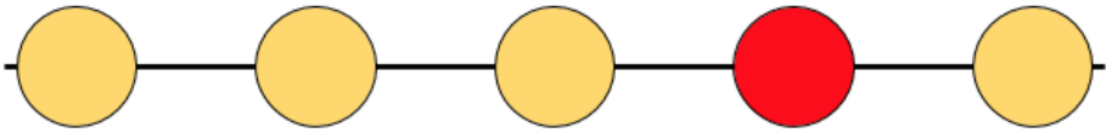


znatno bolje (iOS)	Bolje (iOS)	usporedivo	Bolje (RN)	znatno bolje (RN)

Sl.5.15 *Skala koja prikazuje ocjenu jednostavnosti implementacije programskih sučelja treće strane*

Komunikacija s poslužiteljem

React Native kao primarni jezik koristi JavaScript. Pošto većina komunikacije putem mreže koristi JSON standard (*JavaScript Object Notation*), JavaScript ne mora vršiti serijalizaciju odnosno deserijalizaciju JSON-a, već JSON prepoznaje kao nativni objekt. Na slici 5.16. prikazana je skala sa ocjenom jednostavnosti komunikacije sa poslužiteljem.

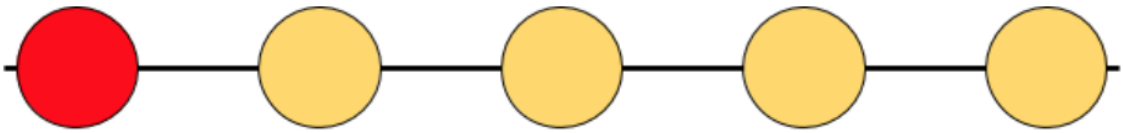


znatno bolje (iOS)	Bolje (iOS)	usporedivo	Bolje (RN)	znatno bolje (RN)

Sl.5.16 *Skala koja prikazuje ocjenu jednostavnosti komunikacije sa poslužiteljem*

Stabilnost ekosustava

iOS nativni razvoj pruža znatno stabilniji ekosustav. React Native je u prošlosti imao velikih problema sa stabilnošću ekosustava za razvoj. Na slici 5.17. prikazana je skala sa ocjenom stabilnosti ekosustava.

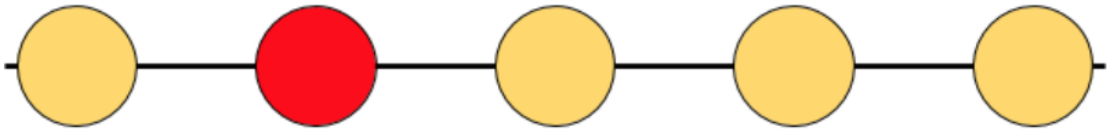


znatno bolje (iOS)	Bolje (iOS)	usporedivo	Bolje (RN)	znatno bolje (RN)

Sl.5.17. *Skala koja prikazuje ocjenu stablinosti ekosustava*

Programski jezik

Programski jezik Swift noviji je i ima puno više značajki od programskog jezika JavaScript. Na slici 5.18. prikazana je skala sa ocjenom programskog jezika koji se koristi u promatranim tehnologijama.

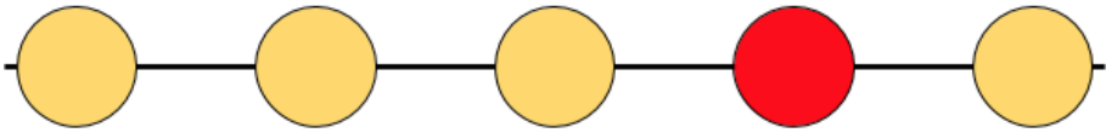


znatno bolje (iOS)	Bolje (iOS)	usporedivo	Bolje (RN)	znatno bolje (RN)

Sl.5.18 *Skala koja prikazuje ocjenu programskog jezika koji se koristi u promatranim tehnologijama*

Arhitektura i dizajn

React je puno bolje dizajniran od UIKit-a. Komponente puno bolje modulariziraju kod, a jednosmjerno slanje podataka u kombinaciji s podjelom komponenti na komponente sa stanjem i bez, superioran je arhitekturalni dizajn naspram dizajna UIKit-a. Na slici 5.19. prikazana je skala sa ocjenom arhitekture i dizajna promatranih tehnologija.

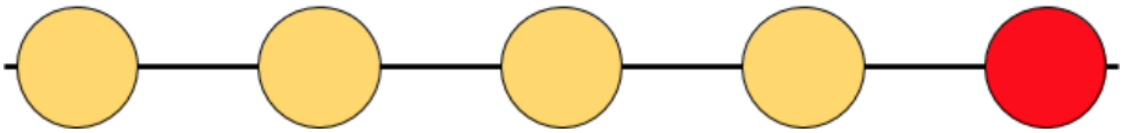


znatno bolje (iOS)	Bolje (iOS)	usporedivo	Bolje (RN)	znatno bolje (RN)

Sl.5.19 *Skala koja prikazuje ocjenu arhitekture i dizajna promatranih tehnologija*

Vrijeme potrebno za izradu i cijena izrade

React Native zahtjeva nešto manje vremena za razvoj od nativnog iOS-a. Uzmemo li u obzir da React Native pokriva i Android platformu, React Native je puno brži i jeftiniji. Na slici 5.20. prikazana je skala sa ocjenom cijene i vremena potrebnog za izradu.

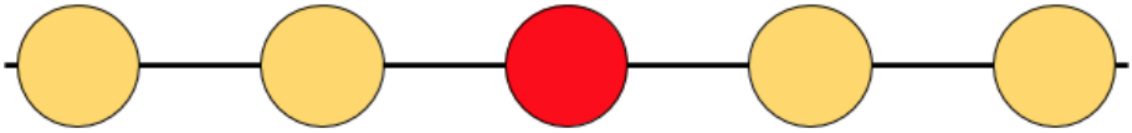


znatno bolje (iOS)	Bolje (iOS)	usporedivo	Bolje (RN)	znatno bolje (RN)

Sl.5.20. *Skala koja prikazuje ocjenu cijene i vremena potrebnog za izradu*

Alati

React Native ima bolje alate za razvoj, dok iOS ima bolje alate za otklanjanje grešaka. Na slici 5.21. prikazana je skala sa ocjenom alata promatranih tehnologija.



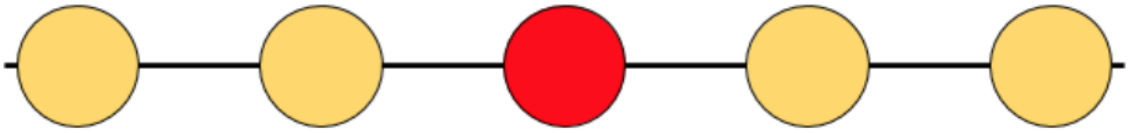
znatno bolje (iOS)	Bolje (iOS)	usporedivo	Bolje (RN)	znatno bolje (RN)

Sl.5.21. *Skala koja prikazuje ocjenu alata promatranih tehnologija*

Objavljivanje

Expo klijent maksimalno automatizira objavljivanje aplikacija. OTA ažuriranja rade bez dodatnih konfiguracija. Također, Expo klijent nudi otvaranje i dijeljenje aplikacije bez potrebe za certifikatima platforme. Apple je krajem 2017. godine onemogućio takvo dijeljenje, pa je ta mogućnost, bez zaobilaznih rješenja, moguća samo na Androidu. S druge strane, sve mogućnosti Expo klijenta moguće je implementirati sa alatima treće strane poput alata Code Push i Fast Lane

tako da nativni razvoj po tom pitanju ne zaostaje izuzev vremena potrebnog za implementaciju navedenih alata. Na slici 5.22. prikazana je skala sa ocjenom jednostavnosti objavljivanja.



znatno bolje (iOS)	Bolje (iOS)	usporedivo	Bolje (RN)	znatno bolje (RN)

Sl.5.22. *Skala koja prikazuje ocjenu jednostavnosti objavljivanja*

5.4.1. Kvantitativna usporedba (mjerenja)

Mjerenja su izvršena za tri stavke: CPU opterećenje, GPU opterećenje i zauzeće memorije, a rezultati su prikazani u tablicama 5.1. i 5.2. te na grafovima koji su prikazani na slikama 5.23, 5.24. i 5.25.

Alati korišteni za mjerenje stavki iOS nativne aplikacije

Apple instrumenti (engl. *Apple instruments*).
Korištena su tri alata:

- CPU mjerač (engl. *Time Profiler Tool*)
- GPU mjerač (engl. *Core Animation Tool*)
- Mjerač zauzeća memorije (engl. *Allocations Tool*)

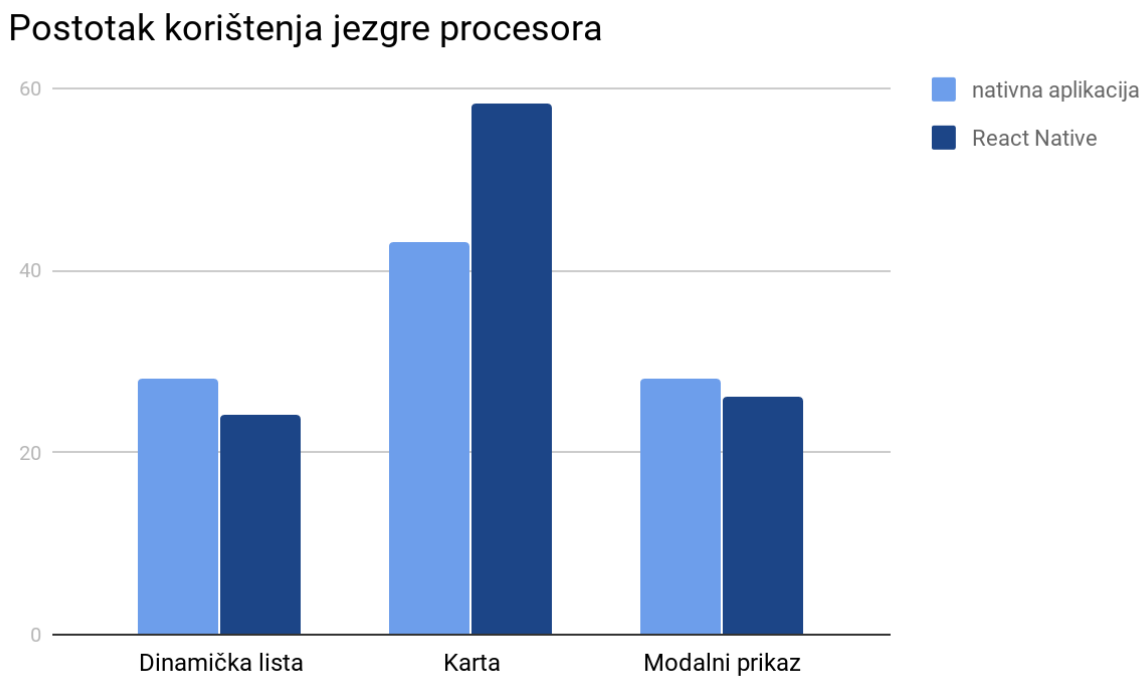
Alati korišteni za mjerenje stavki iOS React Native aplikacije

Ugrađeni alat - monitor performansi (engl. *Pref. monitor*)

CPU

Tab 5.1. *Mjerenja postotka korištenja jezgre procesora*

	nativna aplikacija	React Native
Dinamička lista	28.22	24.14
Karta	43.08	58.26
Modalni prikaz	28.12	26.2

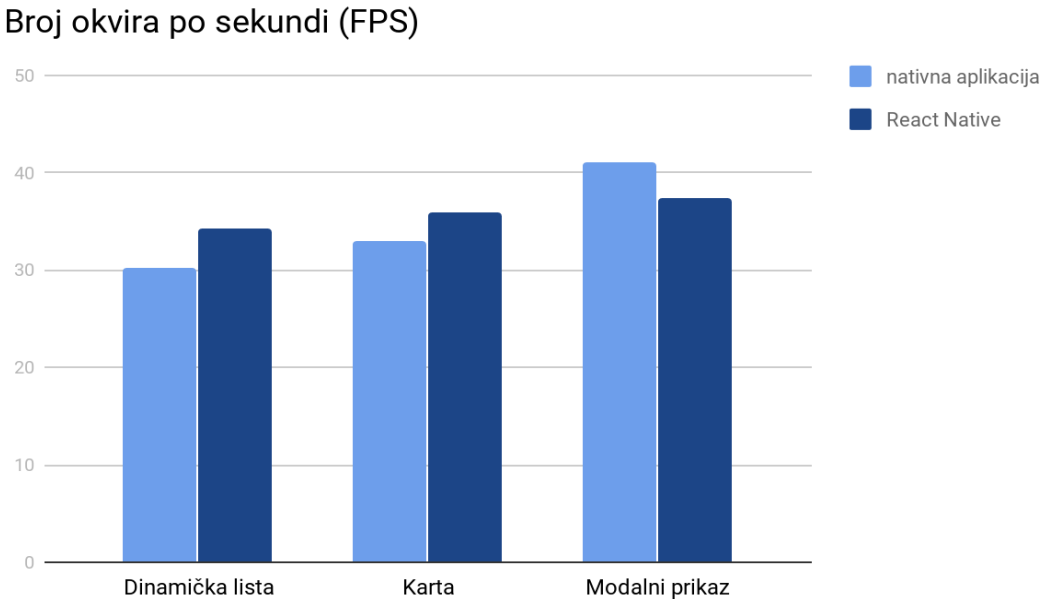


Sl.5.23. *Graf koji prikazuje korištenje procesora u nativnoj i React Native aplikaciji*

GPU

Tab 5.2. *Mjerenja broja okvira po sekundi (FPS)*

	nativna aplikacija	React Native
Dinamička lista	30.15	34.18
Karta	33	36
Modalni prikaz	41	37.4

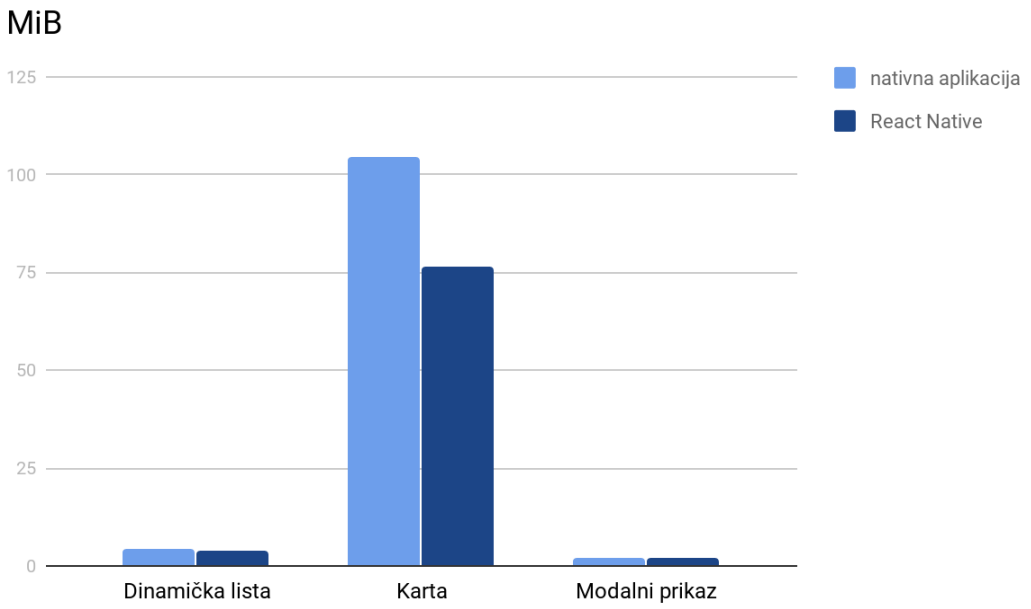


Sl.5.24. *Graf koji prikazuje korištenje GPU jedinice u nativnoj i React Native aplikaciji*

Memorija

Tab 5.3. *Mjerenja zauzeća memorije sikazane u MiB*

	nativna aplikacija	React Native
Dinamička lista	4.51	4.1
Karta	104.37	76.42
Modalni prikaz	2.09	2.14



Sl.5.25. *Graf koji prikazuje zauzeće memorije u nativnoj i React Native aplikaciji*

6. ZAKLJUČAK

Analizirajući činjenice, rezultate usporedbe i mjerenja, dalo bi se zaključiti kako je React Native ozbiljna tehnologija čija je prednost, nad nativnim razvojem, brzina i cijena razvoja, mogućnosti UI sloja, jedinstvena baza koda te alati i NPM ekosustav. Performanse React Native-a su u najmanju ruku usporedive s performansama nativnog razvoja kada je riječ o iOS platformi, dok su performanse nešto slabije na Android platformi. [27]. React Native platforma, u praksi nije savršena. Ekosustav nije jednako stabilan kao i ekosustav nativnog razvoja, no s vremenom se vidi golemi napredak te se u idućih godinu dana očekuju nadogradnje koje bi riješile sve probleme koji se tiču nestabilnosti. [28]

React Native razvoj bit će još većih performansi i bolji dolaskom ReasonML-a u prvu stabilnu fazu. React Native je vrlo moderna i napredna tehnologija koja bi mogla zauzimati sve veći udjel u razvoju mobilnih aplikacija. Također, valja napomenuti kako React Native nije optimalan u svim programerskim timovima. Velike tvrtke koje imaju puno programera s manjkom iskustva, preferiraju nativni razvoj zbog nepredvidivosti i kompleksnosti React Native ekosustava, dok manji timovi vještijih programera preferiraju React Native zbog brzine, cijene te kvalitete koda.
Nativni razvoj uvijek će imati podršku tvrtka koje su vlasnici platformi, a svi ostali alati kasnit će s implementacijom najnovijih značajki.
Iz kvalitativne analize toga i toga, te kvantitativne analize toga i toga u radu, može se zaključiti da jasnog pobjednika nema, a odabir tehnologije uvelike ovisi o okruženju, poslovnom modelu tvrtke te zahtjevima aplikacije.

LITERATURA

[1] Android platform structure, dostupno na: https://www.slideshare.net/chintal75/android-platform-architecture-24627455 (20.9.2018.)

[2] What is ReasonML?, dostupno na: http://reasonmlhub.com/exploring-reasonml/ch_about-reasonml.html (20.9.2018.)

[3] VisionMobile - Business models of mobile ecosystems, dostupno na: https://www.slideshare.net/andreasc/vision-mobile-digital-winners (20.9.2018.)

[4] B.Hein, The Evolution Of The iOS Home Screens, dostupno na; https://www.cultofmac.com/231655/the-evolution-of-the-ios-home-screens-image/ (20.9.2018.)

[5] H. Michael, Android forks are now 20% of the ecosystem. What is Google's plan?, 2014. dostupno na: https://www.phonearena.com/news/Android-forks-are-now-20-of-the-ecosystem.-What-is-Googles-plan_id59003 (13.9.2018.)

[6] https://i-cdn.phonearena.com/images/articles/132206-thumb/android-billion.png (20.9.2018.)

[7] Most Popular Android Versions In September 2018, 2018., dostupno na: https://fossbytes.com/most-popular-android-versions-always-updated/ (13.9.2018.)

[8] Y. Heideser, iOS torches Android when it comes to developer profits, dostupno na: https://bgr.com/2016/07/20/ios-vs-android-developers-profits-app-store-google-play/ (13.9.2018.)

[9] News and updates, Xcode, dostupno na: https://developer.apple.com/news/releases/?id=06132018b (13.9.2018.)

[10] Sight,Amit, A Brief History of Mac OS X, 2012.

[11] J. Timmer, 2014, A fast look at Swift, Apple's new programming language, Ars Technica. Condé Nast

[12] iOS From Scratch With Swift, dostupno na; http://allprowebdesigns.com/tag/mobile [20.9.2018.)

[13] Creating a Unit Converter App, dostupno na: https://androidkennel.org/unit-converter-android-tutorial/ (20.9.2018.)

[14] https://raw.githubusercontent.com/blkbrds/arch-mvvm-ios/1.1.0/images/interactor.png (20.9.2018.)

[15] iOS Simulator on Steroids: Tips & Tricks in Xcode 9, dostupno na: https://www.appcoda.com/ios-simulator-tips-tricks/ (20.9.2018.)

[16] CocoaPods, dostupno na: https://cocoapods.org/ (13.9.2018.)

[17] M.Frachet, Understanding the React Native bridge concept, dostupno na: https://hackernoon.com/understanding-react-native-bridge-concept-e9526066ddb8 (20.9.2018.)

[18] M. Frachet, Understanding the React Native bridge concept, dostupno na: https://hackernoon.com/understanding-react-native-bridge-concept-e9526066ddb8 (13.9.2018.)

[19] B. Fisher, How was the idea to develop React conceived and how many people worked on developing it and implementing it at Facebook?, dostupno na: https://www.quora.com/React-JS-Library/How-was-the-idea-to-develop-React-conceived-and-how-many-people-worked-on-developing-it-and-implementing-it-at-Facebook/answer/Bill-Fisher-17 (13.9.2018.)

[20] The Node.js, dostupno na: https://foundation.nodejs.org/ (13.9.2018.)

[21] Standard ECMA-262, dostupno na: https://www.ecma international.org/publications/standards/Ecma-262.htm (13.9.2018.)

[22] Reason, dostupno na: https://reasonml.github.io/ (13.9.2018.)

[23] D. Lillie, Flux vs. Redux: A Comparison, 2017., dostupno na: https://medium.com/@dakota.lillie/flux-vs-redux-a-comparison-bbd5000d5111 (13.9.2018.)

[24] D.Lillie, Flux vs. Redux: A Comparison, dostupno na: https://medium.com/@dakota.lillie/flux-vs-redux-a-comparison-bbd5000d5111 (20.9.2018.)

[25] MobX, dostupno na; https://mobx.js.org (13.9.2018.)

[26] Expo, dostupno na: ovdje stavi https://expo.io/ (13.9.2018.)

[27] D.S. Ganguly, Top Pros & Cons Comparison: React Native VS. Kotlin, 2018., dostupno na: https://hackernoon.com/top-pros-cons-comparison-react-native-vs-kotlin-2a0dfd1df3e3 (13.9.2018.)

[28] D. Ramel, 5 Years in, React Native Facing Re-Architecture, 2018, dostupno na: https://adtmag.com/articles/2018/06/19/react-native-revamp.aspx (13.9.2018.)

SAŽETAK

Ovaj rad bavi se usporedbom dva različita pristupa razvoju programske podrške. U radu se konkretno opisuje razlika između nativnog i višeplatformskog razvoja programske podrške na primjeru izrade aplikacije za iOS operacijski sustav. Aplikacija izrađena nativnim iOS razvojem uspoređuje se s aplikacijom izrađenom u tehnologiji React Native. Tehnologijama se mjere performanse kako bi se pokazalo da višeplatformski razvoj uz pomoć tehnologije React Native može konkurirati nativnom iOS razvoju.

Ključne riječi: iOS, iOS test performansi, React Native, React Native test performansi, Swift i JavaScript interoperabilnost, višeplatformski razvoj.

NATIVE AND CROSS-PLATFORM DEVELOPMENT COMPARISON ON EXAMPLE OF IOS AND REACT NATIVE

This paper deals with the comparison of two different approaches to the development of software support. The paper describes the difference between native and cross-platform development of software support on the example of creating an application for iOS operating system. The native iOS development application is compared to the application developed in React Native technology. Their performances are measured in order to show that multi-platform development of React Native technology can compete with native iOS development.

Keywords: cross-platform development, iOS, iOS benchmark, React Native, React Native benchmark, Swift and JavaScript interop.

ŽIVOTOPIS

Blaž Jurišić rođen je 31. rujna 1996. godine u Osijeku. Završio je Osnovnu školu Retfala i Matematičku gimnaziju u Osijeku. Trenutno pohađa preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Radi dvije godine kao iOS developer i godinu dana kao JavaScript developer.

PRILOZI (na CD-u)

Prilog 1. Dokument i pdf završnog rada

Prilog 2. Programski kod razvijenih aplikacija