SVEUČILIŠTE U SPLITU

PRIRODOSLOVNO-MATEMATIČKI FAKULTET

DIPLOMSKI RAD

REACT NATIVE OKVIR U RAZVOJU MOBILNIH APLIKACIJA

Petar Meter

Mentor:

prof.dr.sc. Saša Mladenović

Split, rujan 2024.

Temeljna dokumentacijska kartica

Diplomski rad

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Ruđera Boškovića 33, 21000 Split, Hrvatska

REACT NATIVE OKVIR U RAZVOJU MOBILNIH APLIKACIJA

Petar Meter

SAŽETAK

Cilj ovog rada je prikaz izrade mobilne aplikacije FeelFoodApp. Rad započinje upoznavanjem sa vrstama mobilnih aplikacija te detaljnim objašnjenjem odabira strategije za izradu mobilne aplikacije. U daljnjem dijelu rada pojašnjavaju se vrste okvira za izradu mobilnih aplikacija više-platformskim pristupom razvoja te odabrani React Native okvir. Na kraju rada detaljno je analizirana izrada FeelFoodApp mobilne aplikacije na kojoj se temelji ovaj rad.

Ključne riječi: Razvoj mobilnih aplikacija, više-platformski pristup izrade, programski okviri, React Native

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 51 stranicu, 20 grafičkih prikaza i 29 literaturnih navoda. Izvornik je na hrvatskom jeziku

Mentor: **prof. dr. sc. Saša Mladenović,** *redoviti profesor, Prirodoslovno- matematičkog fakulteta u Splitu, Sveučilišta u Splitu*

Ocjenjivači: **Antonela Prnjak, mag. educ. inf.**, *asistentica Prirodoslovno- matematičkog fakulteta u Splitu, Sveučilište u Splitu*

Nina Jerković, mag. ing. comp., *asistentica Prirodoslovno- matematičkog fakulteta u Splitu, Sveučilište u Splitu*

Rad prihvaćen: **rujan, 2024.**

Basic documentation card

Thesis

University of Split

Faculty of Science

Department of Computer science

Ruđera Boškovića 33, 21000 Split, Hrvatska

The React Native Framework in Mobile Application Development

Petar Meter

ABSTRACT

Aim of this thesis is to present implementation of FeelFoodApp mobile application. Thesis begins with an introduction to the types of mobile applications and a detailed explanation of choosing a strategy for implementing a mobile application. In the further part of thesis, the types of frameworks for implementing mobile applications with a cross-platform development approach and the React Native framework are clarified. At the end of thesis, implementation of the FeelFoodApp mobile application, on which this paper is base, is analyzed in detail.

Key words: Mobile application development, cross-platform development, frameworks, React Native

Thesis consists of: 51 pages, 20 figures and 29 references

Original language: Croatian

Mentor: **Saša Mladenović, Ph.D.**
Full Professor of Faculty of Science, University of Split

Reviewers: **Antonela Prnjak, mag. educ. inf.**,
Assistant at the Faculty of Science and Matheatics, University of Split

Nina Jerković, mag. ing. comp.,
Assistant at the Faculty of Science and Matheatics, University of Split

Thesis accepted: **September, 2024**

IZJAVA

kojom izjavljujem, s punom materijalnom i moralnom odgovornošću, da sam diplomski rad s naslovom **REACT NATIVE OKVIR U RAZVOJU MOBILNIH APLIKACIJA** izradio samostalno pod voditeljstvom prof. dr. sc. Saše Mladenovića. Rad je izrađen samostalno, u njemu nema kopiranih ili prepisanih dijelova teksta tuđih radova, a da nisu propisno označeni kao citati s navedenim izvorom iz kojeg su preneseni.

Student

Petar Meter

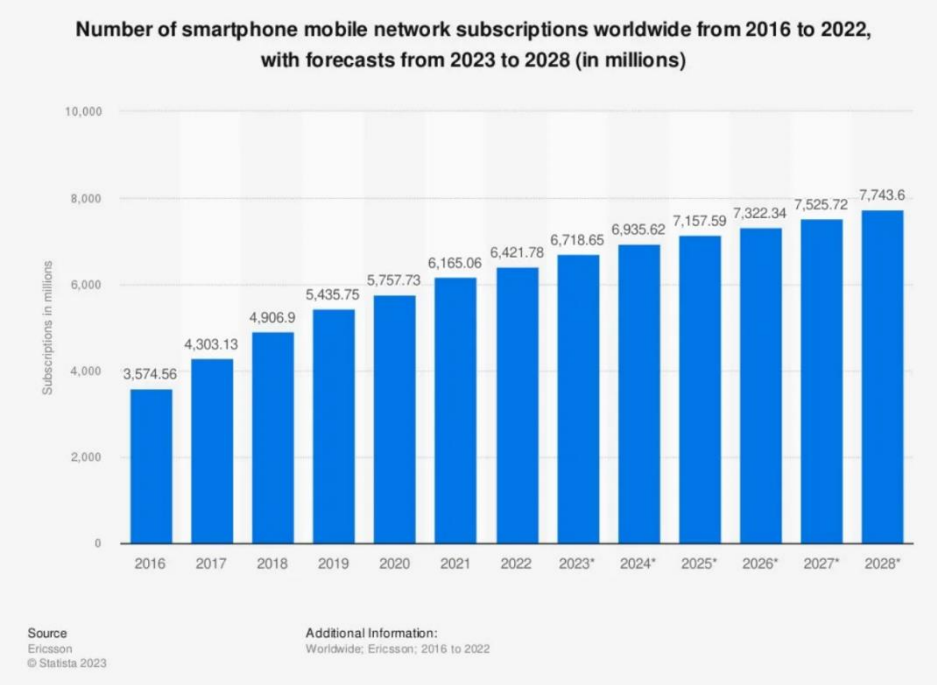
Sadržaj

1. UVOD

Pregledom istraživanja koja su provedena kroz nekoliko posljednjih godina može se utvrditi kako se čovjek sve više i više koristi tehnologijom u svakodnevnom životu. Tako se na svakodnevnoj razini čovjek služi mobilnim uređajima u prosjeku 4 sata dnevno. Samim time povećana je dostupnost raznih mobilnih aplikacija iz različitih područja. Mnoštvo mobilnih aplikacija uređuje čovjekovu svakodnevicu te oblikuje njegov način života. Osim pametnih telefona u mobilne uređaje još spadaju i tableti i pametni satovi te se iz tih razloga pred programere stavlja izazov kreiranja aplikacija koje će biti visoko funkcionalne bez obzira kojim mobilnim uređajem se korisnik služio. U ovom radu će biti opisano što je sve potrebno proučiti pri implementaciji mobilne aplikacije, koje vrste razvoja mobilnih aplikacija postoje zajedno sa njihovim prednostima i nedostatcima u odnosu na konkurente te koji su trenutačno najpopularniji alati za izradu mobilnih aplikacija. U konačnici će biti prikazana izrada projekta FeelFoodApp mobilne aplikacije te koje su njene funkcionalnosti i ideje za nadogradnju u budućnosti.

1.1. MOTIVACIJE

Trenutačno tržište mobilnih aplikacija svakim danom postaje sve veće i veće što ne čudi s obzirom da se pretpostavlja kako u svijetu trenutno postoji oko 6.84 milijardi aktivnih pametnih telefona. Što se tiče postotaka pretpostavlja se kako oko 86% svjetske populacije posjeduje pametni telefon. Eksperti u mobilnoj industriji predviđaju porast broja pametnih telefona i do 7.1 milijarde do kraja 2024. godine. Iz ovih razloga industrija mobilnih aplikacija privlači poslove iz različitih područja. U periodu od 2022. do 2027. godine očekuje se godišnji prirast prihoda trgovine mobilnih aplikacija od 8.83% te se očekuje da će vrijednost mobilnog tržišta biti oko 673.8 milijardi$ do 2027. U slici ispod vidimo porast u broju aktivnih pametnih telefona prema godinama. [1]



Slike 1. Broj aktivnih pametnih telefona po godinama

U idućem dijelu će se detaljnije objasniti što su to zapravo mobilne aplikacije i koje sve prednosti donose vlasnicima poslova te kako izgleda proces izrade mobilne aplikacije. Mobilne aplikacije su vrsta aplikacije koja se izvršava na mobilnom uređaju. U mobilne uređaje uz pametne telefone spadaju i mini računala koja se nazivaju tableti. Iako mobilne aplikacije imaju ograničenu softversku funkcionalnost i dalje uspijevaju u dostavljanju kvalitetnih usluga za široku populaciju korisnika. Mobilne aplikacije se najčešće dizajniraju za specifične mobilne operacijske sustave od kojih su najčešći Android i iOS. Kao najveće prednosti mobilnih aplikacija navode se: dostupnost u svakom trenutku, personalizacija, pristup bez interneta te primanje obavijesti. Jednom kada se aplikacija instalira na mobilni uređaj najčešće su njene funkcionalnosti i servisi dostupni korisniku u svakom trenutku gdje god se on nalazio. Svaki zasebni korisnik može koristiti aplikaciju kako on želi te je prilagoditi svojim potrebama. U slučaju da korisnik nema pristup internetu, većina aplikacija se može koristiti bez internetske povezanosti. Često mobilne aplikacije imaju mogućnost slanja obavijesti svojim korisnicima pomoću kojih ih obavještavaju o trenutnim važnim informacijama i događajima. [2]

Kao što imaju svoje prednosti mobilne aplikacije imaju i svoje nedostatke. Od glavnih nedostataka izdvajaju se ograničena funkcionalnost, ograničena kompatibilnost, sigurnosne prijetnje i ograničena sposobnost ažuriranja. Ograničena kompatibilnost predstavlja nedostatak iz razloga što su neke aplikacije kreirane za određene operacijske sustave te nisu dostupne na ostalima. Mobilne aplikacije često koriste osjetljive informacije te nemaju jednako razvijene razine sigurnosti kao računalne aplikacije. Kod nekih aplikacija postoji problem ograničene sposobnosti ažuriranja kada korisnici često moraju preuzimati novije verzije aplikacije kako bi mogli koristiti aplikaciju. Budući da su mobilne aplikacije jednostavnije za izradu i korištenje postoji nedostatak ograničene funkcionalnosti. Iz tog razloga mobilne aplikacije se najčešće kreiraju samo za specifične potrebe. Tako u današnje vrijeme prednjače aplikacije koje su kreirane za industriju video-igara, društvene mreže, aplikacije za kupovinu, edukacijske aplikacije, aplikacije za svakodnevni život i slično. U ovom radu prikazat će se izrada mobilne aplikacije unutar koje postoji prikaz različitih jela, ispis njihovih sastojaka te njihove hranjive tvari. Ova aplikacija će spadati u područje aplikacija za edukaciju.

Prema podatcima Svjetske Zdravstvene Organizacije u svijetu trenutno ima oko 1 milijarda pretilih ljudi. Od 1990. godine do 1. ožujka 2024. godine pretilost se kod odraslih ljudi udvostručila dok se kod djece učetverostručila. Kao najveći krivac smatra se današnji „brzi“ način života u kojem se najčešće konzumira hrana kojoj je potrebno najmanje vremena za pripremu.[3]

Motiv za izradu ovakve aplikacije je dostupnost izrade različitih jela, bila ona poznata ili nepoznata korisnicima, te pregled sastojaka za izradu. Uz ovakve usluge smatram da će korisnici aplikacije imati raznovrsniju prehranu što će im za rezultat dati bolje zdravstveno stanje.

2. PRIPREMA IZRADE MOBILNE APLIKACIJE

Izrada mobilne aplikacije bez unaprijed pripremljene strategije često može dovesti do neuspjeha. U današnjem tržištu mobilnih aplikacija mnoge jedinstvene ideje nisu dovoljne za uspjeh zbog prevelike konkurentnosti. Teško je ponuditi korisnicima uslugu koja će se istaknuti među sličnima dok je u isto vrijeme potrebno uložiti mnogo novca i vremena pri izradi aplikacije. Kako bi smanjili rizik od neuspjeha potrebno je pripremiti strategiju koja će dovesti do lakše izrade aplikacije. Ako programeri u početnim fazama ulože vrijeme kako bi razvili strategiju izrade aplikacije mogu dobiti značajne prednosti u odnosu na konkurenciju. [4]

2.1. IZRADA STRATEGIJE

Prvi korak koji je potrebno proći pri izradi strategije je upoznati tržište vlastite aplikacije. U ovom dijelu programeri moraju pronaći postojeće aplikacije na tržištu te dobiti uvid u njihove nedostatke. Budući da je tržište mobilnih aplikacija svakim danom sve konkurentnije dostupno je mnoštvo besplatnih aplikacija na različitim trgovinama aplikacijama kao što su Google Play Store i Apple App Store. Također, uz pomoć pregleda recenzija korisnika programeri mogu pronaći nedostatke trenutnih aplikacija te ih iskoristiti pri izradi vlastite aplikacije.

Nakon ovog koraka, potrebno je ustanoviti temeljnu svrhu aplikacije. Često je programerima ovaj dio najteži iz razloga što mnoštvo ideja trebaju svesti na samo jednu primarnu. Kada definiramo primarnu svrhu potrebno je objasniti usluge koje će aplikacija nuditi, definirati koji su potencijalni korisnici aplikacije te zbog čega će korisnici koristiti baš tu aplikaciju. Iz ovih razloga programeri mogu usmjeriti svoje resurse na usluge aplikacije koje će ispuniti očekivanja

određene skupine korisnika. Ako ovaj korak predstavlja nedoumice postoji mogućnost da takva aplikacija doživi neuspjeh.

U odabiru strategije potrebno je imati i par odluka što se tiče pristupa razvoja mobilne aplikacije. Programeri moraju odabrati između 4 vrste mobilnih aplikacija koje se razlikuju po pristupu razvoja: izvorne mobilne aplikacije, web mobilne aplikacije, aplikacije razvijene hibridnim pristupom i više-platformski razvijene mobilne aplikacije. Uspješnost ranije utvrđenih koraka ima veliku ulogu pri odabiru prikladnog razvoja za izradu aplikacije. Osnovne razlike između izvornih, web, hibridnih i više-platformski razvijenih aplikacija su načini izrade aplikacija te vrijeme potrebno za kreiranje aplikacije. Također, odabir između ovih vrsta aplikacija ovisit će za koju platformu želimo izraditi aplikaciju, Android ili iOS. U daljnjem dijelu rada detaljnije će se analizirati pristupi razvoja mobilnih aplikacija.

Pitanje koje programeri često postavljaju je koju ulogu će aplikacija imati u njihovom modelu poslovanja? Najčešće se želi zaraditi upravo pomoću same aplikacije. U svijetu poslovanja mobilnim aplikacijama najčešći modeli monetizacije su sljedeći:

1. Freemium aplikacije – Preuzimanje aplikacija je besplatno ali određenim uslugama se može pristupiti samo u slučaju plaćanja.
2. Plaćene aplikacije – Aplikaciju je potrebno kupiti u trgovini aplikacijama kako bi se mogla koristiti. U ovom slučaju potrebno je izgraditi jedinstvenu marketinšku strategiju kako bi aplikacija konkurirala sličnim aplikacijama koje su besplatne za korištenje.
3. Kupovina unutar aplikacije – Ovakve aplikacije najčešće nude prodaju određenih usluga unutar aplikacije.
4. Pretplate – Model aplikacija koji je sličan „freemium“ aplikacijama ali donosi stalni prihod korisnika.
5. Plaćene reklame – Najjednostavniji model kod kojeg korisnici nemaju nikakvih troškova, a prihodi dolaze od plaćenih reklama unutar aplikacije. U ovakvim aplikacijama programeri moraju osigurati dovoljno mjesta za reklame ali da one ne idu na štetu korisničkom iskustvu korištenja aplikacije.
6. Sponzorstva – Ovakav model se često koristi u trenutku kada već postoji solidna baza korisnika te aplikacija postane plaćena od strane određenih brandova.

Svaki od ovih modela ima svoje prednosti i nedostatke te se naknadno aplikacija može prebaciti na drugi model ali je važno za programere da razumiju osnove svakog modela kako bi njihova aplikacija bila što uspješnija.

U posljednjem koraku potrebno je upoznati resurse kojima se raspolaže pri izradi aplikacije. U ovaj korak spada odabir programskog jezika, izgled aplikacije i njene analize. Nakon analize prethodnog može se uvidjeti koliko vremena je potrebno za izradu aplikacije te raspolaže li se sa dovoljno resursa za izradu. Iako je potrebno uložiti vrijeme i trud u izradu strategije, programerima će izrada aplikacije ići puno jednostavnije te će imati manji rizik od neuspjeha. [4]

2.2. KRITERIJI ZA ODABIR PRISTUPA RAZVOJA MOBILNIH APLIKACIJA

Odabir ispravnog pristupa razvoja mobilnih aplikacija programerima nikada nije jednostavan. Budući da je svaka aplikacija jedinstvena programeri moraju uzeti u obzir sve zasebne dijelove te ih posložiti u konačni proizvod. Nakon izrade strategija za kreiranje mobilne aplikacije, programeri se susreću sa izazovom odabira ispravnog pristupa za razvoj mobilnih aplikacija. [5]

Kao što je navedeno u prijašnjem dijelu rada, postoje 4 vrste mobilnih aplikacija koje se razlikuju po pristupu razvoja: izvorne, web, hibridne i više platformski razvijene mobilne aplikacije. U daljnjem dijelu će se objasniti svaki od ovih pristupa. [6]

2.2.1. Izvorne mobilne aplikacije

Izvorne mobilne aplikacije koriste okvire za izradu aplikacija koji su specifično izrađeni za određeni operacijski sustav, najčešće Android koji je proizveden od tvrtke Google ili iOS proizveden od tvrtke Apple. Budući da su optimizirane za određene operacijske sustave ovakve aplikacije pružaju visoke perfomanse te nude programerima iskorištavanje svih resursa koje platforme nude što rezultira velikom pouzdanosti ovakvih aplikacija. [7,8]

Ovakve mobilne aplikacije imaju pristup integriranim okruženjima za razvoj što im omogućuje najbolje alate za izradu te slijedno tome imaju brze rezultate izvršavanja. Izvorne mobilne aplikacije se preuzimaju iz trgovina mobilnih aplikacija te se nakon instalacije izvršavaju na uređajima. Aplikacije koje su izrađene za iOS operacijski sustav se najčešće izrađuju uz pomoć programskih jezika Swift i Objective-C dok se aplikacije za Android uređaje izrađuju uz pomoć Jave ili Kotlina. Trgovina aplikacijama koja podržava iOS aplikacije naziva se App Store dok se Android aplikacije najčešće preuzimaju u Google Play Store trgovini aplikacijama. [5]

Kao najveće prednosti izvornih mobilnih aplikacija navodi se njihova pouzdanost i brzina. Ovakve aplikacije imaju dostupne sve značajke pametnog telefona budući da su izrađene i optimizirane za specifične operacijske sustave. Korištenje nekih izvornih mobilnih aplikacija je moguće i bez povezanosti sa Internetom. Aplikacije kreirane za određene operacijske sustave pružaju ugodnije korisničko iskustvo budući da su sučelja dizajnirana prema standardima određenog operacijskog sustava. Zbog toga korisnicima treba manje vremena pri savladavanju svih značajki aplikacija.

U najveće mane ovakvih aplikacija navode se višestruke kodne baze, veliki troškovi izrade te duže vrijeme potrebno za izradu. Ovakve aplikacije nisu u mogućnosti koristiti istu kodnu bazu budući da su razvijene za specifične operacijske sustave. Troškovi i vrijeme izrade se množe sa brojem platformi koje aplikacija želi podržavati. [9]

Neke od najpoznatijih aplikacija koje su razvijene ovakvim pristupom su: Google Maps, Apple Music, WhatsApp, Instagram i Snapchat.

2.2.2. Web mobilne aplikacije

Alternativni pristup kreiranju mobilnih aplikacija uz pomoću kojeg se može konkurirati mobilnim aplikacijama kreiranim izvornim pristupom je kreiranje Web mobilnih aplikacija. Web mobilne aplikacije su kreirane kako bi se izvršavale na internetskim preglednicima. Dakle, nijedna od komponenata aplikacije nije instalirana na mobilnom uređaju već se aplikacija dohvaća uz pomoć URL-a odnosno putanje do podataka aplikacije koja je instalirana na serveru. Kod web mobilnih aplikacija izrada je olakšana budući da se izrađuju uz pomoć HTML5 i CSS3 jezika za uređenje izgleda stranice, dok je od programskih jezika potrebno poznavati JavaScript koji je zadužen za funkcionalnost aplikacije. Ovakve aplikacije se dizajniraju kako bi imale slične usluge kao izvorne mobilne aplikacije ali se njihovo izvršavanje obavlja na pregledniku umjesto na samom mobilnom uređaju. Web mobilne aplikacije zahtijevaju stalnu povezanost sa Internetom budući da aplikacija nije instalirana na pametni telefon već njene usluge dohvaćamo uz pomoć preglednika. Zbog jedinstvenih servisa preglednika ovakva vrsta aplikacije je dostupna na svim uređajima koji imaju pristup pregledniku neovisno o njihovom operacijskom sustavu. Brzina aplikacije će ovisiti o brzini povezanosti sa Internetom. [10,11, 12]

Najpopularnije svjetske aplikacije najčešće imaju i svoju web aplikaciju kojoj se pristupa putem preglednika. Trenutno najpopularnije aplikacije od kojih se očekuje najveći porast korisnika u 2024. godini su: MakeMyTrip, Flipboard, Tinder, Uber i Pinterest.

2.2.3. Mobilne aplikacije izgrađene hibridnim pristupom

Pristup izrade mobilnih aplikacija koji predstavlja kombinaciju izvornih i web mobilnih aplikacija su aplikacije izgrađene hibridnim pristupom. Hibridne aplikacije imaju karakteristike web mobilnih aplikacija budući da su izrađene sličnim tehnologijama za izgled i funkcionalnost aplikacije dok ovakav pristup nudi usluge koje pružaju pametni telefoni, što kod web mobilnih aplikacija nije bio slučaj. [13]

Hibridne aplikacije su dostupne za preuzimanje putem trgovina mobilnih aplikacija te jednom kada se instaliraju spremne su za korištenje. Usluge aplikacije se dohvaćaju putem preglednika koji čini dodatni sloj između aplikacije i usluga mobilnih uređaja. Neke hibridne aplikacije se mogu pokrenuti i bez povezanosti sa Internetom a to ovisi o funkcionalnostima aplikacije. Najveće prednosti hibridnih aplikacija su: mogućnost korištenja neovisno o operacijskom sustavu uređaja, brži način izrade u usporedbi sa izvornim aplikacijama, jeftiniji troškovi zbog korištenja jednostavnijih tehnologija u usporedbi sa izvornim aplikacijama te mogućnost korištenja sa i bez povezanosti sa Internetom. [13, 14]

Mane koje najčešće susrećemo kod hibridnih aplikacija su: izgled aplikacije može varirati u ovisnosti o operacijskom sustavu, korisničko iskustvo može biti lošije u odnosu na ostale vrste mobilnih aplikacija te je potrebno dodatno testirati aplikacije kako bi ispravno funkcionirale na različitim uređajima.

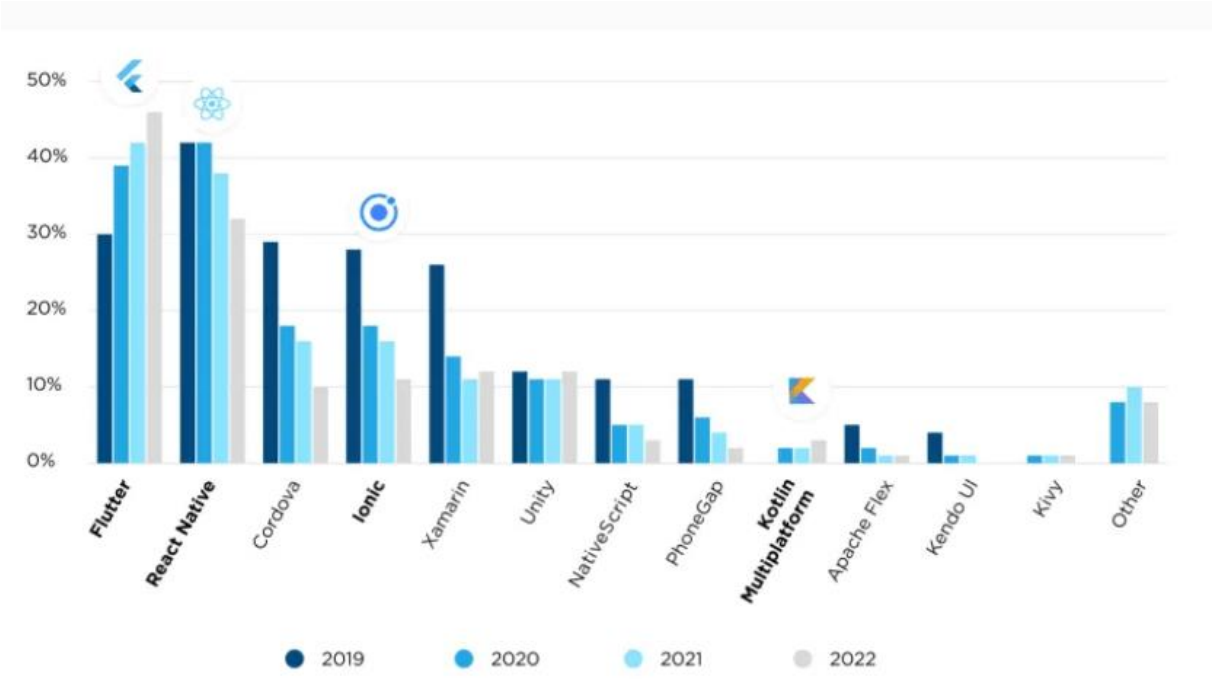
2.2.4. Više-platformski razvijene mobilne aplikacije

Pristup razvoja mobilnih aplikacija koji dozvoljava da se jedna mobilna aplikacija izvršava na više različitih operacijskih sustava naziva se više-platformski pristup. Uz pomoć ovog pristupa programeri mogu kreirati aplikacije te dijeliti izvorni kod kako bi se aplikacije mogle izvršavati na različitim operacijskim sustavima. [15] Dakle, aplikacija koja je kreirana može se izvršavati i na Android i na iOS operacijskom sustavu bez potrebe za pisanjem novog koda za svaki od operacijskih sustava. Uz pomoć okvira koji su dostupni unutar programskih jezika kao što su: JavaScript, Ruby i Java programeri kreiraju programska aplikacijska sučelja. Kod koji se koristi pri izradi sučelja se procesira i pretvara u izvorne aplikacije određenog operacijskog sustava na kojem će se aplikacija pokrenuti. Uz pomoć ovog postupka aplikacije se mogu izvršavati na

više različitih operacijskih sustava. [16] Najveće prednosti više-platformski razvijenih aplikacija su njihova brzina kreiranja i poboljšano korisničko iskustvo budući da se aplikacije izvršavaju kao izvorne te im je dostupna većina usluga uređaja na kojem se izvršavaju. Zbog dijeljenja koda, tvrtke će uštedjeti na vremenu i troškovima zaposlenja količine programera za izradu svoje aplikacije. Iako još uvijek ne mogu biti konkurentni izvornim mobilnim aplikacijama u smislu brzine izvođenja i samog korisničkog iskustva popularnost više-platformskih razvijenih aplikacija postaje sve veća. [17]

Ograničenja koja se javljaju kod više-platformski razvijenih mobilnih aplikacija se najčešće prikazuju u odnosu na izvorne mobilne aplikacije. Brzina izvođenja je manja u odnosu na izvorne mobilne aplikacije dok za iskorištavanje izvornih usluga uređaja programerima treba više truda. Što se tiče radne snage, na tržištu je teško pronaći programere koji imaju znanje u kreiranju više-platformski razvijenih aplikacija ali sve većim napretkom modernih tehnologija kao što su React Native i Kotlin ovaj nedostatak postaje sve manji. [17]

Trenutno najpopularniji okviri za izradu više-platformski razvijenih aplikacija u posljednjih par godina su: Flutter, React Native, Ionic i Kotlin Multiplatform. U daljnjem dijelu ukratko ćemo opisati najpopularnije okvire za razvoj više-platformskih aplikacija uz njihove prednosti i nedostatke. [18]



Slike 2. Najpopularniji okviri po godinama

Flutter je razvojni okvir kreiran od strane Google u 2017. godini. Od tada je postao najpopularniji okvir za izradu više-platformskih aplikacija. Aplikacije kreirane pomoću Flutter okvira se pišu u Dart programskom jeziku koji je poznat po svojoj efikasnosti te je zbog toga odličan izbor za izradu mobilnih aplikacija. Budući da koristi samo jedan izvorni kod za pokretanje na različitim operacijskim sustavima, Flutter smanjuje troškove i vrijeme izrade aplikacija. Kao najznačajnije svojstvo Fluttera se najčešće navodi funkcionalnost brzog osvježavanja. Tako programeri mogu vidjeti razlike u izgledu aplikacije nakon svake promjene koda bez ponovnog pokretanja aplikacije što ubrzava vrijeme izrade. Zbog mnoštva usluga koje Flutter posjeduje programeri mogu kreirati vizualno atraktivna korisnička sučelja. Procjenjuje se da će ovaj razvojni okvir u budućnosti napredovati zbog široke zajednice programera koji ga koriste što će za rezultat dati brojnije servise za unaprjeđenje korisničkog iskustva. Trenutno najpopularnije mobilne aplikacije izrađene pomoću Fluttera su: Google Ads, Hamilton i Reflectly.

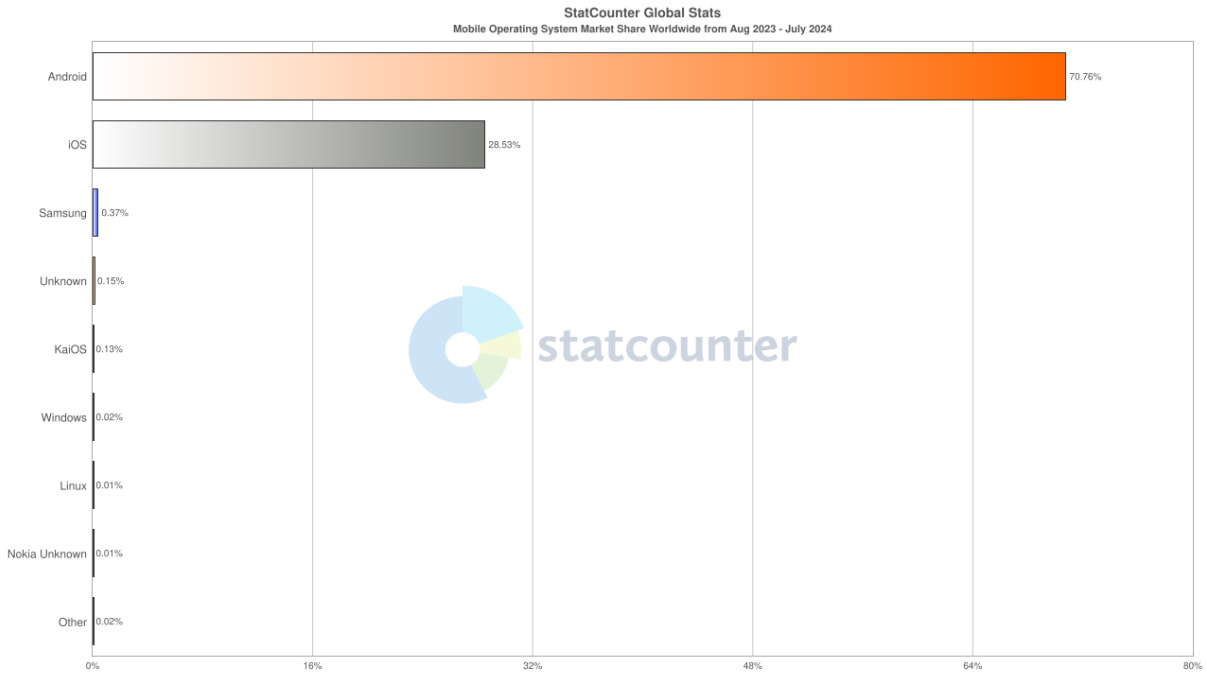
React Native je jedan od najpopularnijih okvira za izradu više-platformski razvijenih mobilnih aplikacija kreiran od strane Facebooka u 2015. godini. On je popularni okvir za izradu mobilnih aplikacija uz pomoć JavaScript programskog jezika te podržava izradu aplikacija za iOS i Android operacijske sustave. Trenutno jedne od najvećih mobilnih aplikacija su kreirane upravo ovim okvirom a to su: Instagram, Facebook i Skype. Budući da se uz pomoć jednog izvornog koda kreiraju aplikacije za dva operacijska sustava, uz pomoć React Nativea se štedi na vremenu i troškovima razvoja aplikacija. Popularnost React Nativea se može opravdati time što se on bazira na JavaScript biblioteci naziva React koja je već bila popularna u vrijeme izlaska React Native okvira. [18] Budući da je React Native okvir izabran za izradu mobilne aplikacije na kojoj se temelji ovaj rad o njemu će se pisati detaljnije u daljnjem dijelu rada.

Ionic je okvir za izradu više-platformskih aplikacija uz pomoć alata otvorenog koda napisanog u HTML-u, CSS-u i JavaScriptu. Glavni fokus ovog okvira je kreiranje interaktivnih korisničkih sučelja aplikacije. Ovaj okvir dozvoljava integraciju sa Angular, React i Vue okvirima te zbog toga omogućuje izradu visoko kvalitetnih više-platformskih aplikacija. Neke od najpoznatijih aplikacija izrađene pomoću Ionica su: T-Mobile, BBC i EA Games.

Kotlin Multiplatform je alat za kreiranje više-platformskih mobilnih aplikacija koji je objavljen u 2017. godini od strane JetBrainsa. On se temelji na ideji „Write Once Run Anywhere“ čija je glavna svrha da se jedan izvorni kod može koristiti za pokretanje aplikacija na različitim operacijskim sustavima. Kotlin podržava mnoštvo različitih operacijskih sustava kao što su: iOS, Android, Mac, Linux i Windows. Iako najmlađi od okvira njegova zajednica korisnika je svakim danom sve veća a to se može opravdati mnoštvom modernih usluga za izradu korisničkih sučelja koja su konkurentna izvornim mobilnim aplikacijama. Najpopularnije aplikacije koje su izrađene uz pomoć Kotlina su: Netflix, Bolt i McDonald's.[20]

3. ODABIR TEHNOLOGIJE

Prema trenutnim brojkama na tržištu mobilnih aplikacija prednjače aplikacije koje podržavaju iOS i Android operacijske sustave. Mobilne aplikacije koje podržavaju Android operacijski sustav zauzimaju oko 71% dok mobilne aplikacije koje podržavaju iOS operacijski sustav zauzimaju oko 28%. Dakle, ova dva operacijska sustava obuhvaćaju oko 99% tržišta mobilnim aplikacijama. [21]



Slike 3. Tržište mobilnih aplikacija prema operacijskim sustavima

Ovakve brojke je poželjno uzeti u obzir te se može zaključiti kako svako novo poslovanje želi obuhvatiti što širu skupinu korisnika. Kako bi privukli korisnike ova dva operacijska sustava moguće je: kreirati zasebnu mobilnu aplikaciju za svaki operacijski sustav ili kreirati više-platformski razvijenu mobilnu aplikaciju. Za mobilnu aplikaciju na kojoj se temelji ovaj rad odabran je više-platformski pristup razvoja mobilne aplikacije iz više razloga. Budući da će mobilna aplikacija FeelFoodApp biti nova na tržištu želi se obuhvatiti velika većina korisnika

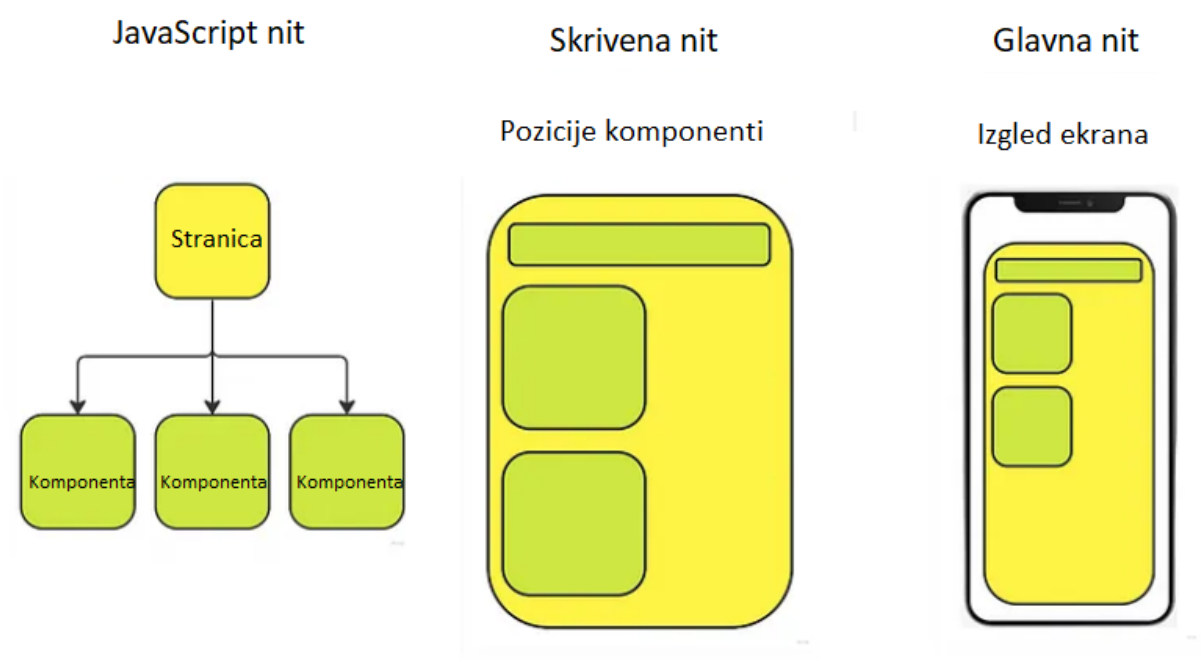
što više-platformski razvoj omogućuje. Sve šira zajednica programera koristi više-platformski razvoj izrade aplikacija te se samim time svakim danom unaprjeđuju programski okviri i alati za razvoj aplikacija. Također, zbog veće zajednice korisnika ovakvog pristupa alati za uređenje izgleda korisničkog sučelja aplikacije nude kvalitetnije funkcionalnosti te ostvaruju zadovoljstvo korisnika, kako Androida tako i iOS operacijskog sustava. Uz dijeljenje koda za izradu aplikacija za više operacijskih sustava istovremeno štedi se na vremenu razvoja te su aplikacije u kratkom roku dostupne na tržištu.

Nakon odabira pristupa za razvoj mobilne aplikacije potrebno je odabrati okvir za izradu aplikacije. Kod više-platformski razvijenih aplikacija najpopularniji okviri su: Flutter, React Native, Ionic i Kotlin Multiplatform. Svaki od ovih programskih okvira ima vlastite prednosti i nedostatke. Razumljivo je da su trenutno najpopularniji okviri Flutter i React Native budući da imaju podršku od strane Googlea i Facebooka. Pri izradi mobilne aplikacije na kojoj se temelji ovaj rad odabran je React Native programski okvir. [22]

3.1. REACT NATIVE

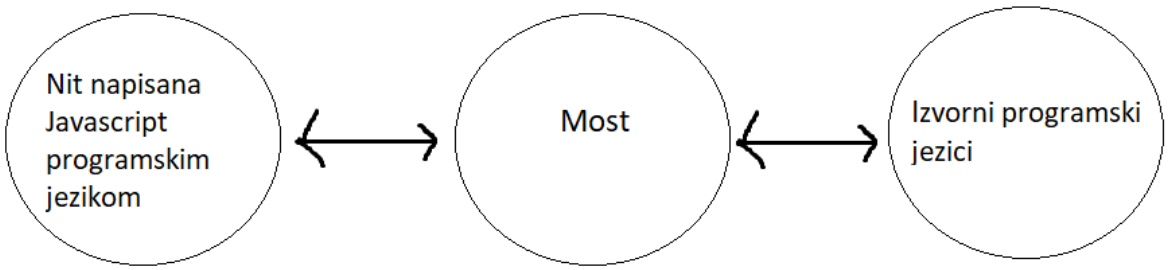
U prethodnim poglavljima su obrađene osnovne informacije o React Native okviru za više-platformski razvoj mobilnih aplikacija. Koristeći React Native kreiramo izvorne aplikacije za dva mobilna operacijska sustava, iOS i Android, uz pomoć jedinstvene kodne baze. Ono što je specifično za React Native i po mnogim programerima označeno kao njegova najveća prednost u odnosu na ostale okvire za razvoj više-platformskih aplikacija je njegova arhitektura.

Kada se priča o arhitekturi React Native-a ona se sastoji od osnovne 3 niti: JavaScript nit, skrivena nit i glavna nit. JavaScript nit je zaslužna za pokretanje cijelog koda, skrivena nit kreira izvorne stilove i pozicionira komponente aplikacije dok glavna nit prikazuje korisničko sučelje te osluškuje događaje. Sav kod napisan u JavaScript niti se interpretira od dva React Native Engine-a a to su JavaScriptCore ili Hermes. Uz pomoć skrivene niti, izvorni kod pomoću kojeg kreiramo ekran aplikacije se transformira u format kojeg podržavaju iOS i Android operacijski sustavi. U konačnici glavna nit je zaslužna za prikaz korisničkog sučelja te upravljanje korisničkim događajima putem usluga pametnih telefona.



Slike 4. Niti u React Native-u

Za komunikaciju unutar React Native arhitekture je zaslužan takozvani *most. Most* je koncept kreiran od Reactovog razvojnog tima kako bi omogućio komunikaciju unutar aplikacije između dva potpuno različita jezika. Komunikacija unutar *mosta* se odvija asinkrono i u oba smjera. Funkcionira kao red poruka unutar kojeg su poruke spremljene u JSON formatu zbog efikasnosti. *Most* prima poruke od niti koje su napisane u JavaScript programskom jeziku te ih prevodi u izvorne jezike za iOS i Android operacijske sustave (Objective-C za iOS i Java za Android). Ono što je važno naglasiti je da *most* sam po sebi nije zasebna nit već je posrednik u komunikaciji između JavaScript niti i modula koji su pokrenuti na glavnoj niti.[23]



Slike 5.Komunikacija u React Native-u

3.1.1 Prikaz osnovnih komponenti

U React Native okviru postoji mnoštvo komponenti pomoću kojih se kreira korisničko sučelje aplikacije bez naknadnog instaliranja dodatnih biblioteka. Osnovne komponente, koje će se detaljno obraditi u idućim poglavljima, su: *View, Text, ScrollView, TextInput, Image, Button* i *FlatList* budući da su najpopularnije korištene komponente pri izradi korisničkog sučelja pomoću React Native okvira.

3.1.1.1. View komponenta

View ili pogled je osnovna komponenta pri izradi rasporeda svakog korisničkog sučelja a u React Nativeu se označava kao <View>. Na Android i iOS operacijskim sustavima ova je komponenta također postojana ali sa različitim imenima. Tako se na iOS-u naziva <UIView> a za Android njen naziv je <ViewGroup>. Zbog svoje mogućnosti da može biti ugniježđen unutar drugih pogleda te može imati različite komponente unutar sebe najčešće se koristi za kreiranje rasporeda korisničkog sučelja aplikacije.

Primjer više ugniježđenih <View> komponenti u React Nativeu bi izgledao ovako:

```
<View style={containerStyle}>
    <View style={childView1}>
        <View style={childView2}>
        </View>
    </View>
</View>
```

3.1.1.2. Text komponenta

Komponenta koja se kod React Nativea koristi za prikaz tekstualnog sadržaja je komponenta <Text>. Podržava mogućnost uređivanja i ugnježđivanja unutar ostalih komponenti te postoji mogućnost događaja povezanih sa dodirom. Kod iOS operacijskih sustava ova komponenta ima naziv <UITextView> a kod Androida je poznatija kao <TextView>. Ova komponenta kao djecu može primati konstante i varijable tekstualnog oblika koje se nazivaju *string* ili druge <Text> komponente. Najčešće se koristi za prikaz više različitih stilova tekstualnog sadržaja.

Primjer korištenja <Text> komponente u React Nativeu je sljedeći:

```
<Text style={italicText}>
    Marko voli igrati nogomet.
    <Text style={boldText>
    Najdraži su mu treninzi petkom.
    </Text>
</Text>
```

3.1.1.3. ScrollView komponenta

Komponenta koja se koristi za povlačenje sadržaja ekrana u React Nativeu naziva se <ScrollView>. Ova komponenta se može ugnijezditi u ostale komponente. Postoji mogućnost vodoravnog i okomitog povlačenja a ono se kreira uz pomoć za to specifičnih svojstava. Mana <ScrollViewa> je što sve komponente koje su ugniježdene u ovu osvježava odjednom što znači da u slučaju velikih količina podataka imamo sporije učitavanje stranice. Kod iOS operacijskog sustava ova komponenta je poznatija pod nazivom <UIScrollView> dok se kod Androida naziva <ScrollView>.

Primjer korištenja <ScrollView> komponente u ReactNativeu je sljedeći:

```
<ScrollView>
    <Text> Marko voli igrati nogomet. </Text>
    <Text> Najdraži treninzi su mu petkom.</Text>
</ScrollView>
```

3.1.1.4. TextInput komponenta

Komponenta koja se koristi pri unosu teksta u aplikaciju naziva se <TextInput>. Ova komponenta pruža razne mogućnosti kao što su: ispravljač teksta, skrivanje lozinke te podrški za različite vrste tipkovnica. Kod iOS operacijskog sustava ova komponenta se naziva <UITextField> dok je kod Androida poznata kao <EditText>. Ova komponenta se može ugnježđivati unutar različitih komponenti ali ne može primati druge komponente. Najpopularniji način korištenja je uz pomoć događaja *onChangeText* kada se tekst sprema u JavaScript varijablu.

Primjer korištenja <TextInput> komponente u React Nativeu je sljedeći:

```
<View>
    <Text input onChangeText={onChangeText}
    placeholder="Molimo unesite tekst:"/>
</View>
```

3.1.1.5. Image komponenta

Komponenta pomoću koje se prikazuju slike na korisničkim sučeljima aplikacije u React Nativeu je <Image> komponenta. Ova komponenta podržava ugnježđivanje u ostale komponente ali ne prima nijednu drugu komponentu kao svoju. Nudi mogućnost umetanja vlastitih slika kao i snimanje kamerom mobilnog uređaja. Kod iOS operacijskog sustava ova komponenta je poznata kao <UIImageView> dok se kod Androida naziva <ImageView>.

Primjer korištenja <Image> komponente kod React Native je sljedeći:

```
<View>
    <Image
        style={base64Image}
        source={{uri: https://reactnative.dev/img/homepage/devices.png}}
    />
</View>
```

3.1.1.6. Button komponenta

Najjednostavnija komponenta za prikazivanje gumba u React Nativeu se naziva <Button> komponenta. Zbog svoje jednostavnosti ova komponenta za svaki od operacijskih sustava ima strogo prilagođeni stil izgleda kako bi korisnici aplikacije na tom operacijskom sustav imali osjećaj izvornosti. Primarni razlog korištenja komponenti za prikaz gumba je aktiviranje određene radnje ili događaja koji će potaknuti promjene na korisničkom sučelju. U <Button> komponenti je stoga dostupna funkcija koja će se izvesti nakon što korisnik pritisne gumb a naziva se *onPress.*

Primjer <Button> komponente u React Nativeu je sljedeći:

```
<Button
        onPress={funkcijaZaIzvrsavanje}
        title="Gumb"
/>
```

3.1.1.7. FlatList komponenta

Komponenta koja se u React Nativeu također koristi za prikaz uz mogućnost povlačenja naziva se <FlatList> komponenta. Ova komponenta ima mogućnost da bude ugniježđena u ostale komponente kao i da prima druge komponente. Glavna razlika u <FlatList> i <ScrollView> komponentama je ta što <ScrollView> kod učitavanja većih podataka ima sporije učitavanje. Iz tog razloga kreirana je <FlatList> komponenta koja funkcionira na principu brisanja komponenti koje se pomiču daleko od zaslona te samim time ušteđuje vrijeme obrade i memoriju. Podatci se u ovu komponentu ugnježđuju u obliku JavaScript liste podataka te određivanjem izgleda komponente za prikaz tih podataka.

Primjer korištenja <FlatList> komponente u React Nativeu je sljedeći:

```
<FlatList
        data={lista_podataka}
        keyExtractor={item=>item.id}
        renderItem={(item) =>(
                <View>
                        <Text>{item.title}</Text>
                </View>
)}/>
```

3.1.2 JSX

Ekstenzija koja omogućuje pisanje HTML oznaka unutar Reacta naziva se JSX. Prije JSX ekstenzije izgled aplikacija se sastojao od HTML-a za pisanje oznaka, CSS-a za izgled stranice te JavaScripta za logiku aplikacije i sva tri jezika su najčešće pisana u zasebnim datotekama. Sa svakodnevnim napredovanjem aplikacija te izlaskom Reacta, komponente se pišu zajedno u jednoj datoteci te spajaju pisanje oznaka i logičkog dijela aplikacije. Pisanje JSX-a je vrlo slično pisanju HTML oznaka ali u ovom slučaju oznake mogu sadržavati dinamične informacije čije se stanje mijenja pomoću JavaScripta. JSX sintaksa spaja pisanje oznaka i pisanje JavaScript koda unutar vitičastih zagrada. Iz tog razloga moguće je povezati korisničko sučelje aplikacije sa logikom aplikacije. Korištenjem JSX-a se proces izrade aplikacije pojednostavljuje te smanjuje mogućnost pogrešaka i upozorenja. [24]

Primjer JSX koda:

```
        let  ime = 'Petar';
        const element = <Text> Moje ime je {ime} </Text>
```

Pomoću JSX-a se uz pisanje varijabli mogu kreirati i JavaScript metode.

Primjer korištenja JSX-a sa JavaScript metodama:

```
function pomnoziBrojeve(brojevi){
        return brojevi.prvi_broj * brojevi.drugi_broj
}
function dohvatiRezultat(brojevi){
        if(brojevi){
                return <Text> Rezultat je {pomnoziBrojeve(brojevi)} </Text>
        }
}
```

Važnost korištenja JSX a u React Nativeu će se prikazati u idućem poglavlju unutar kojeg se prikazuje pisanje vlastitih komponenti.

3.1.3. Izrada vlastitih komponenti

Kao što se ranije pisalo prilikom detaljnog opisivanja osnovnih komponenti, većina osnovnih komponenti daje mogućnost ugnježđivanja što predstavlja osnovni koncept pri izradi mobilnih aplikacija uz pomoć React Nativea. Vlastite komponente daju mogućnost višestruke uporabe jedne komponente prilikom daljnje izrade korisničkog sučelja što u konačnici smanjuje potrebu za pisanjem već napisanog koda.

Primjer izrade vlastite komponente:

```
const Osoba = () => {
        return(
                <View>
                        <Text>Moje ime je {props.ime}</Text>
```

```
            </View>
        );
}
const App = () => {
        return(
            <View>
                <Osoba ime = 'Petar'/>
                <Osoba ime = 'Mate'/>
            </View>
        );
}
```

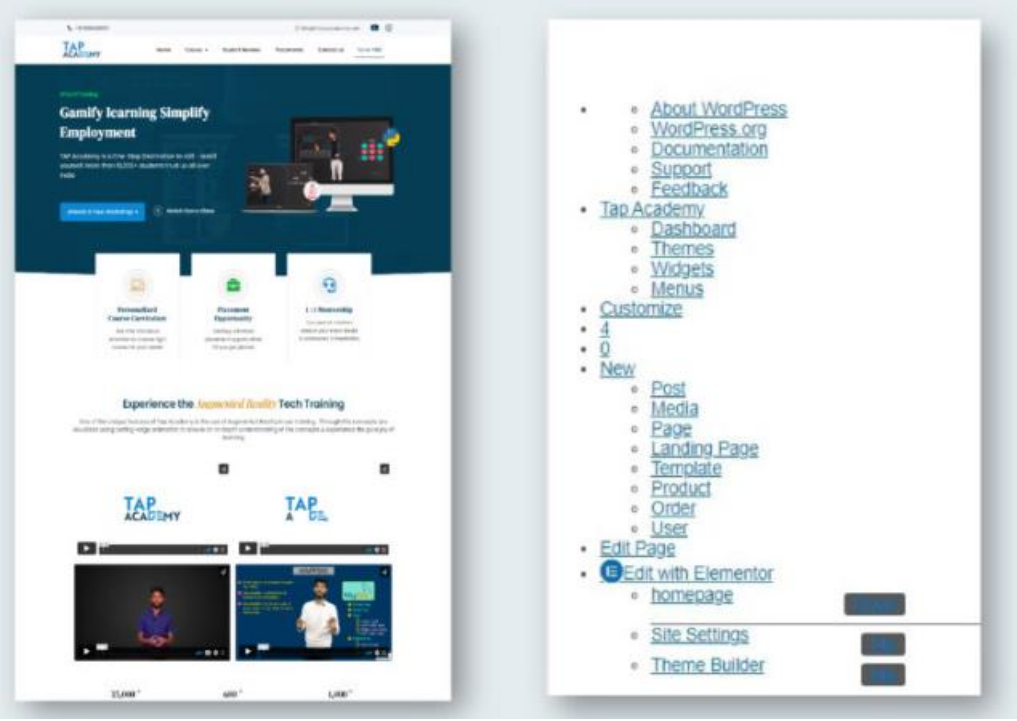
U primjeru izrade vlastite komponente postoje dvije komponente. Prva komponenta ima naziv Osoba te se sastoji od osnovnih komponenti: <View> i <Text>. Druga komponenta se sastoji od osnovne komponente <View> te dvije ugniježđene komponente <Osoba>. Izraz *props* omogućuje korištenje svojstava komponenti roditelja. Tako se u ovom primjeru od roditeljske komponente prima ime osobe koje se naknadno opisuje. Može se primijetiti kako prilikom svakog novog pozivanja vlastite kreirane komponente <Osoba> ime osobe se mijenja. Pošto *props* predstavlja JavaScript objekt, svojstva komponenata mogu biti različiti te se prilikom prosljeđivanja može proslijediti različiti sadržaj. Iz tog razloga React Native daje mogućnost organizacije logike korisničkog sučelja te jednostavniju izradu.

3.2. CSS

Cascading Sytle Sheets ili skraćeno CSS je jezik za stilsko oblikovanje izgleda elemenata stranice koji su napisani HTML-om. Kreiran je 1996. godine te je široko korišten u programerskoj zajednici što mu doprinosi svakodnevni razvoj. Pomoću CSS stilskog jezika

može se oblikovati dizajn i raspored elemenata koji su prikazani na internetskoj stranici. Drugim riječima, oblikuje se sve ono što korisnik vidi kada se stranica koristi. Ovaj jezik se najčešće povezuje sa HTML-om. HTML jezik je zadužen za kreiranje sadržaja stranice koji uključuje tekstove, linkove, slike i slično ali ne određuje raspored i dizajn kojim će sve to biti prikazano na stranici. Uz pomoć CSS-a programeri mogu ispuniti korisničke zahtjeve te učiniti da stranice izgledaju što ljepše i ugodnije za korištenje. HTML i CSS se u pravilu pišu u zasebnim datotekama kako bi olakšali kreiranje velikih internetskih stranica. [25]

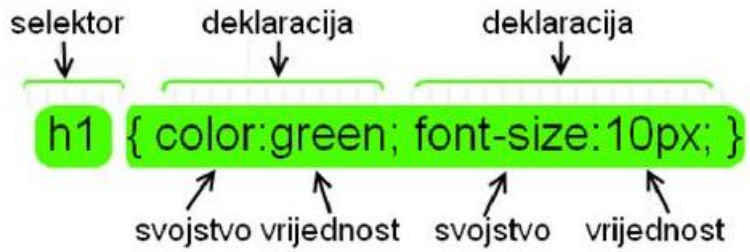
CSS dozvoljava dijeljenje koda te se tako jedan kreiran stil može koristiti na više elemenata unutar više stranica smanjujući vrijeme izrade te mogućnost pogrešaka. Budući da je CSS trenutno najpopularniji jezik za oblikovanje stranica svakodnevno se proširuju njegove mogućnosti što dovodi do ispunjavanja raznih korisničkih zahtjeva. Njegova jednostavna sintaksa omogućuje i manje iskusnim programerima brzo savladavanje jezikom. Na sljedećoj slici prikazat će se izgled stranice sa i bez uređivanja uz pomoć CSS jezika.



Slike 6.Izgled stranice uređene CSS-om i bez CSS-a

3.2.1. Sintaksa pisanja CSS-a

Kod pisanja CSS-a postoji par osnovnih komponenti: selektori, deklaracije, svojstva i vrijednosti. Pravila pisanja uvijek započinju sa selektorom. Uz pomoć selektora se odabire koji se element napisan HTML-om želi stilizirati. Unutar vitičastih zagrada se pišu deklaracije, odnosno označava se koje svojstvo želimo uređivati te prema kojim vrijednostima. Najčešća svojstva koja se uređuju su boja, veličina, font, oblik i lokacija elementa. Iako su ovo najčešća svojstva kod CSS jezika postoji još mnoštvo svojstava koja se svakodnevno ažuriraju. Vrijednosti svojstava mogu varirati; tako se za boje koriste riječi koje označavaju boju dok se može koristiti i kod u heksadekadskom obliku (npr. #3300FF) te RGB vrijednosti koje pišemo na slijedeći način rgb(0,0,0). Pri dodjeljivanju vrijednosti svojstvima koja uređuju oblik elemenata koriste se pikseli ili se pišu vrijednosti u obliku postotka.



Slike 7.Primjer koda u CSS-u

3.2.2. TailWind

TailWind je okvir niske razine koji je prilagođen CSS stilskom jeziku. Zbog svoje jednostavnosti i brzine uređivanja trenutno je jedan od najpopularnijih okvira izrađenih za CSS. Uz pomoć TailWind okvira elementi se stiliziraju uz pisanje koda direktno uz HTML. Sintaksa TailWinda se razlikuje od CSS-a u tome što se kod TailWinda koriste kratice za svojstva koja se stiliziraju. Najveća prednost TailWinda u odnosu na ostale stilske okvire je korištenje takozvanih *utility* klasa. Uz pomoć *utility* klasa uređuje se svaka zasebna komponenta nakon

njene deklaracije. Većina klasa ima slična svojstva te se stilovi mogu koristiti na više različitih klasa. [27]

Primjer pisanja koda TailWindom je slijedeći:

* *

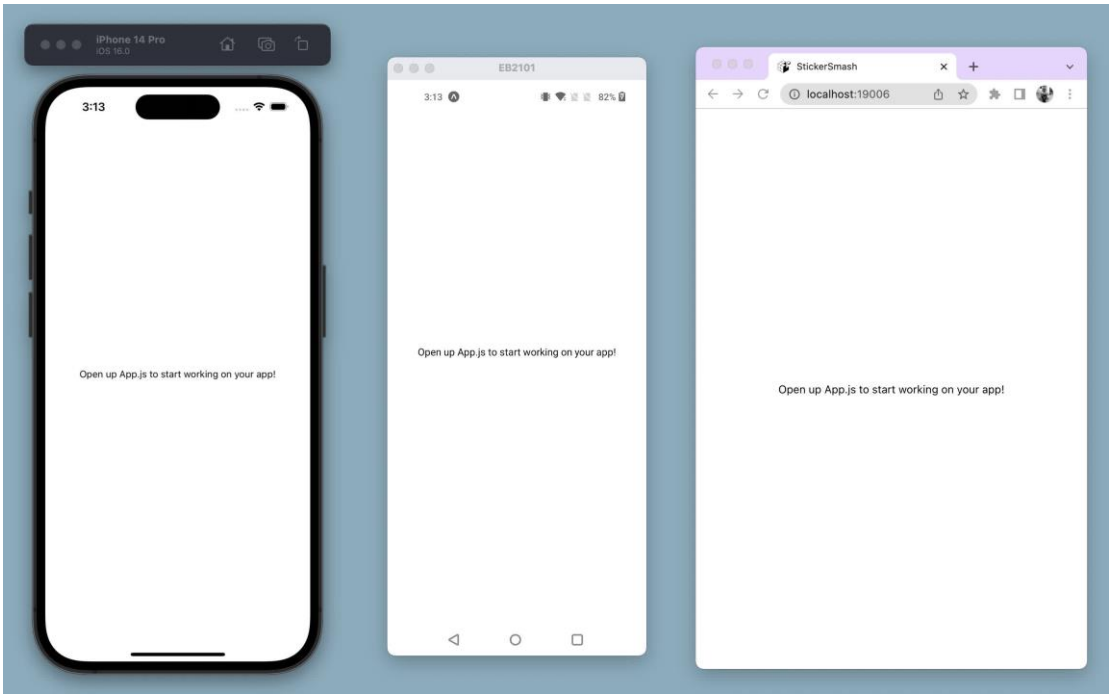
3.3. VISUAL STUDIO CODE

Visual Studio Code je uređivač teksta koji podržava različite programske jezike i operacijske sustave te je iz tih razloga jedan od najpopularnijih alata među programerima. Posjeduje mnoštvo različitih alata i ugrađenih biblioteka. Kreiran je od strane Microsofta te je besplatan pa je kao takav pogodan za početnike. Najčešće se koristi za kreiranje internetskih stranica te mobilnih i web aplikacija. Uz pomoć ugrađenih biblioteka jednostavno se mogu otklanjati pogreške u hodu te pruža mogućnost predviđanja željenog koda čime se dobiva na brzini implementacije.

3.4. EXPO

Expo je programski okvir koji pomaže pri kreiranju React Native mobilnih aplikacija. Sadrži brojne biblioteke i izvorne servise koji olakšavaju kreiranje aplikacija. Koristi ga široka zajednica programera budući da je besplatan te je u izravnoj povezanosti sa React Native timom što mu omogućuje dostupnost najnovijih usluga koje React Native nudi.

Unutar Expo korisničkog sučelja programerima je dozvoljeno implementiranje same aplikacije bez da mijenjaju izvorni kod. Također dostupan je izgled aplikacije na iOS i Android operacijskom sustavu u svakom trenutku. Uz pomoć Expo Go aplikacije koju je moguće preuzeti i instalirati na mobilni uređaj dostupno je povezati aplikaciju koju izrađujemo sa našim mobilnim uređajem. [28]



Slike 8.Prikazi korisničkih sučelja kroz Expo

4. NAČIN IZRADE APLIKACIJE

Za praktični dio rada prikazat ćemo mobilnu aplikaciju naziva FeelFoodApp unutar koje će korisnicima biti dostupan prikaz različitih jela iz različitih zemalja, te osnovne informacije o jelima kao što su: priprema i sastojci koji sačinjavaju jelo. Jela će biti poredana po kategorijama te će se na odabir prikazati različita jela iste kategorije. Za izradu mobilne aplikacije odabran

je više-platformski pristup kroz React Native razvojni okvir. Pomoću ovog pristupa aplikacija će biti kompatibilna kako sa iOS tako i sa Android operacijskim sustavom.

4.1. KORIŠTENE BIBLIOTEKE I KOMPONENTE

U sljedećem poglavlju detaljnije će se objasniti korištene biblioteke pri implementaciji FeelFoodApp mobilne aplikacije.

4.1.1. React biblioteka

React biblioteka je namijenjena za kreiranje modernih korisničkih sučelja od zasebnih komponenti. Pomoću različitih funkcija programerima je dostupna mogućnost primanja i slanja podataka kroz komponente aplikacije. Komponente se osvježavaju u trenutku promjene stanja informacija. Iz ove biblioteke u projektu su korištene metode 'useState' i 'useEffect'.

Pomoću 'useState' metode možemo kreirati funkcionalne komponente pomoću korištenja stanja komponente. Ova metoda se sastoji od inicijalizacije stanja, ažuriranja stanja i stanja renderiranja. U inicijalizaciji stanja kreiramo inicijalnu vrijednost stanja komponente koja može biti različitog tipa podatka. U ažuriranju stanja pomoću interakcija korisnika stanje komponente se mijenja te pokreće stanje renderiranja. Kad je pokrenuto stanje renderiranja tada se korisničko sučelje ažurira te ispisuje nove informacije stanja komponenti sa trenutnim vrijednostima.

Primjer korištenja 'useState' metode je slijedeći:

```
const [godine, postaviGodine] = useState(28);
```

U primjeru koda iznad je deklarirana varijabla *godine* sa inicijalnom vrijednosti 28. Kada želimo promijeniti vrijednost ove varijable pozvati će se *postaviGodine* te će kao vrijednost primiti novi broj godina.

Metoda koju koristimo kada želimo izvršiti određeni dio logike koda nakon renderiranja komponente naziva se 'useEffect'. Vrijednost koja se šalje kao prvi argument se mijenja nakon svakog renderiranja komponente, a o drugoj vrijednosti ovisi hoće li se 'useEffect' izvršiti. U slučaju da šaljemo prazan niz kao drugu vrijednost 'useEffect' će se izvršiti nakon prvog renderiranja.

4.1.2.React-native osnovna biblioteka

U implementaciji projekta iz osnovne 'react-native' biblioteke korištene su slijedeće komponente: <View>, <Text>, <Pressable>, <Image>, <TouchableOpacity>, <StatusBar>, <Button>. Neke od ovih komponenti su već detaljnije objašnjene u prijašnjim poglavljima.

<Pressable> komponenta služi za detekciju korisnikovih interakcija pritiskanja bilo mišem, tipkovnicom ili preko zaslona. Ova komponenta može ugnijezditi sve ostale komponente te se detekcija dodira vrši u trenutku dodirivanja komponenti koje se nalaze unutar <Pressable> komponente. Iako detekcija pritiskanja nekada nije pouzdana ova komponenta posjeduje različite funkcije uz pomoću kojih se sprječava očitavanje slučajnih pritisaka po zaslonu.

<TouchableOpacity> je komponenta slična <Pressable> komponenti te najčešće služi za detekciju korisnikovih interakcija pritiskanja zaslona. Glavna razlika između ovih komponenti je korištenje <TouchableOpacity> komponente pri jednostavnijim interakcijama dok <Pressable> komponenta nudi mogućnosti povratnih informacija prema korisniku u trenutku njegove interakcije.

<StatusBar> komponenta služi za kontrolu trake informacija aplikacije. Ova traka se najčešće nalazi na vrhu zaslona te prikazuje vrijeme, razinu baterije, povezanost sa Internetom i slično.

4.1.3. React-native-responsive-screen biblioteka

Biblioteka 'react-native-responsive-screen' je malena biblioteka koja nudi dvije metode za korištenje React Native programerima pri izradi mobilnih aplikacija. Najčešće se koristi za raspored elemenata korisničkog sučelja. Dvije metode koje ova biblioteka nudi su: 'widthPercentageToDP' i 'heightPercentageToDP'. One se koriste kako bi aplikacija detektirala na kojem uređaju je pokrenuta te pomoću toga odredila visinu i širinu zaslona. Nakon toga pri kreiranju elemenata zaslona možemo iskoristiti rezultat ovih dviju metoda u obliku postotka kako bi korisničko sučelje izgledalo isto na bilo kojem uređaju. Uz pomoć ove metode korisničko sučelje će biti jednako na kojem god uređaju se aplikacija pokrenila.

4.1.4.React-native-reanimated biblioteka

Biblioteka kreirana od strane Software Mansion tima se naziva 'react-native-reanimated'. Ova biblioteka nudi mnoštvo različitih animacija pri kreiranju korisničkog sučelja aplikacije. Animacije kreirane ovom bibliotekom najčešće imaju i do 120 fps-a. Kod React Native-a se svaka komponenta odmah pojavi na ekranu pri učitavanju svakog zaslona. U ovom projektu korištene animirane komponente FadeIn i BounceIn koje sadrže mnoštvo modifikacija učitavanja komponenti.

4.1.5.React-native-seoul biblioteka

Komponenta koja je korištena iz 'react-native-seoul' biblioteke se naziva <Masonry-list> komponenta. Ova komponenta ima najsličnija svojstva <FlatList> komponenti te prikazuje listu <View> komponenti jednu ispod druge uz mnoštvo različitih mogućnosti prikaza.

4.1.6.React-navigation biblioteka

Biblioteka koja se naziva 'react-navigation' služi za implementaciju navigacije aplikacije. Budući da se mobilne aplikacije rijetko kreiraju sa samo jednim zaslonom ova biblioteka nudi tranziciju između različitih zaslona. Ova biblioteka se sastoji od mnogobrojnih različitih komponenti. U ovom projektu su se koristile komponente <NavigationContainer> te <Stack.Navigator>.

<NavigationContainer> komponenta je odgovorna za upravljanje stanjima aplikacije te povezivanje okruženja u aplikaciji.

<Stack.Navigator> komponenta služi za tranziciju između zaslona aplikacije postavljajući svaki novi zaslon na vrh stoga. Ova komponenta je slična i u iOS i Android operacijskom sustavu dok se njene animacije mogu modificirati prema potrebama korisnika.

4.1.7.React-native-heroicons biblioteka

Biblioteka koju je kreirao Steve Schoger a objavljena je od strane Tailwind Labs tvrtke naziva se 'react-native-heroicons'. Pomoću ove biblioteke najčešće se kreiraju komponente koje imaju izgled različitih ikona. Ikona koja je korištena u ovom projektu naziva se 'ChevronLeftIcon' a njena svrha je povratak na prijašnji zaslon uz interakciju korisnikovog pritiska.

4.1.8. Axios biblioteka

Biblioteka pomoću koje se dohvaćaju API zahtjevi u ovom projektu naziva se 'axios'. API je naziv sa mehanizme koji omogućavaju komunikaciju između dva različita softverska proizvoda koristeći se unaprijed definiranim protokolima. Ova komunikacija se odvija na principu zahtjeva i odgovora. Aplikacija koja šalje zahtjev se najčešće naziva klijent dok se aplikacija koja šalje odgovor naziva server. Npr. zavod za vremensku prognozu sadrži svakodnevne nove informacije o vremenskim prognozama te one mogu biti poslane mobilnoj aplikaciji za vremensku prognozu pomoću API-ja. [29]

Uz pomoć Axios biblioteke i njenih metoda podatci koji su dohvaćeni sa Interneta su prikazani u mobilnoj aplikaciji.

5. IZGLED I FUNKCIONALNOST FEELFOODAPP MOBILNE APLIKACIJE

U ovom poglavlju bit će prikazan izgled zaslona unutar mobilne aplikacije FeelFoodApp te funkcionalnosti aplikacije. Razvojno okruženje unutar kojeg je implementirana aplikacija je Visual Studio Code. FeelFoodApp je mobilna aplikacija kreirana više-platformskim pristupom razvoja te se u budućnosti planira nalaziti na mobilnom tržištu. Za model monetizacije izabrana je *Freemium* vrsta mobilne aplikacije što bi značilo da će aplikacija biti dostupna za besplatno preuzimanje ali će se neke funkcionalnosti morati kupiti. Za testiranje aplikacije korištena je aplikacija ExpoGo koja nudi povezanost sa Visual Studio Code razvojnim okruženjem.

FeelFoodApp mobilna aplikacija se sastoji od 3 zaslona: 'Welcome', 'Home' i 'Details'. Svaki od ovih zaslona će biti detaljno objašnjen uz prikazan kod implementacije.

5.1. NAVIGACIJA APLIKACIJE

Za navigaciju aplikacije korištena je <NavigationContainer> komponenta i Stack.Screem metoda pomoću kojih tranzicija između zaslona funkcionira na primjeru stoga. Početni zaslon je inicijaliziran a slijedeći zasloni idu na vrh stoga.

Primjer koda za navigaciju aplikacije prikazan je na slijedećoj slici:

```
const Stack = createNativeStackNavigator();

function AppNavigation(){
    return(
        <NavigationContainer>
            <Stack.Navigator initialRouteName="Welcome" screenOptions={{headerShown:false}}>
                <Stack.Screen name="Welcome" component={Welcome}/>
                <Stack.Screen name="Home" component={Home}/>
                <Stack.Screen name="Details" component={Details}/>
            </Stack.Navigator>
        </NavigationContainer>
    )
}
```

Slike 9.Implementacija navigacije aplikacije

5.2. POČETNI ZASLON 'WELCOME'

Pri pokretanju aplikacije pojavit će se prvi zaslon dobrodošlice. Na zaslonu dobrodošlice prikazan je logo aplikacije, njen naziv, te poruka za korisnika. Prijelaz sa zaslona dobrodošlice na slijedeći zaslon se radi pritiskom na tipku s natpisom 'Let's get started.'

Slike 10.Prikaz zaslona dobrodošlice

Izgled zaslona je uređen uz pomoć Tailwind koda. Animacija koja se koristi na početnoj stranici je povećanje loga aplikacije u trenutku pokretanja.

Implementacija animacije se nalazi na slici ispod:

```
<Animated.View className="bg-white/10 rounded-full" style={{padding:ring2padding}}>
    <Animated.View className="bg-white rounded-full" style={{padding:ring1padding}}>
        <Image source={require('../assets/logo.jpg')}
            style={{width:hp(17),height:hp(17)}}/>
    </Animated.View>
</Animated.View>
```

Slike 11.Prikaz koda za implementaciju animacije učitavanja loga

Prelazak na drugi zaslon je implementiran uz pomoć <Button> komponente i metoda iz biblioteka za navigaciju.

Prikaz <Button> komponente za prelazak na iduću stranicu je na slijedećoj slici:

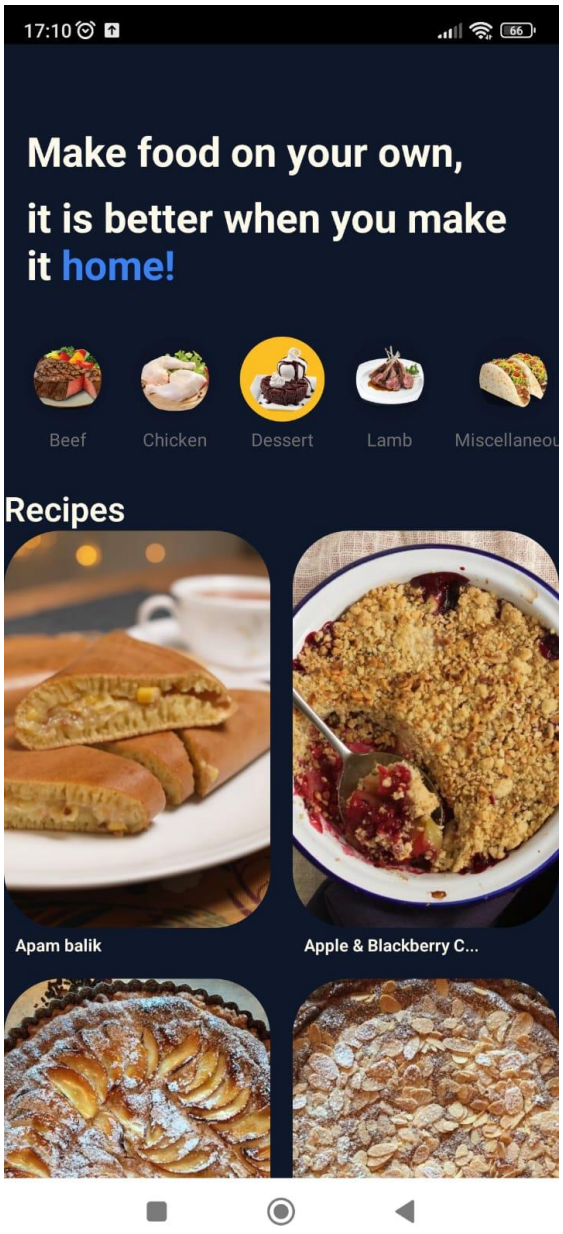
```
<View className="flex items-center space-y-2">
<Button
    classname="bg-blue-950 text-white/30"
    title="Let's get started"
    onPress={()=> navigation.navigate("Home")}
/>
</View>
```

Slike 12.Prikaz implementacije koda prelaska na zaslon 'Home'

5.3. ZASLON 'HOME'

Unutar 'Home' zaslona nalaze se tekstualni motivi na vrhu ekrana a zatim su korisniku ponuđene kategorije jela. Ispod kategorije jela korisniku su dostupne slike jela uz njihov naziv. U trenutku kada korisnik pritisne određenu kategoriju učitavaju se slike i nazivi jela iz odabrane kategorije. Za prijelaz na slijedeći zaslon potrebno je odabrati jelo čije detalje korisnik želi istražiti.

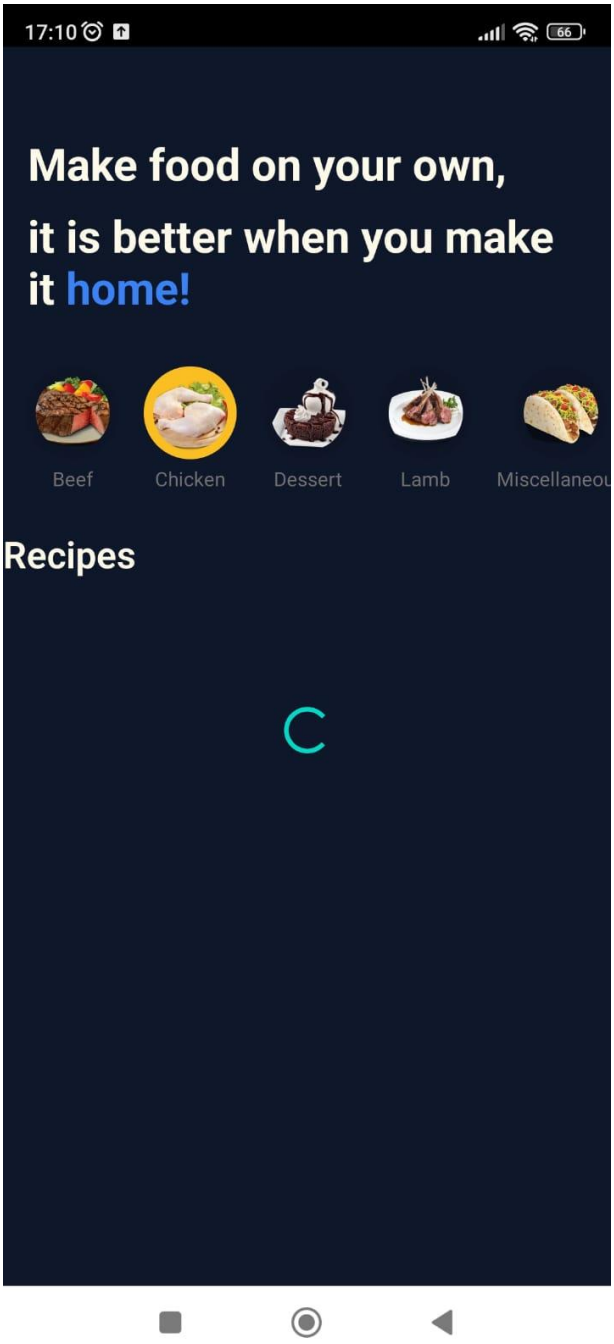
Primjer izgleda zaslona je na slijedećoj slici:



Slike 13.Izgled zaslona 'Home'

Pri promjeni kategorije koju je korisnik odabrao prikazuje se ikona za učitavanje jela.

Primjer učitavanja je na slijedećoj slici:



Slike 14.Prikaz učitavanja jela

Dohvaćanje dostupnih kategorija i jela se obavlja uz pomoć zahtjeva preko TheMealDB API-ja. U slanju zahtjeva korištena je Axios biblioteka.

Primjer implementacije dohvaćanje kategorija je na slijedećoj slici:

```
const getCategories = async()=>{
    try{
        const response = await axios.get(`https://themealdb.com/api/json/v1/1/categories.php`);
        //console.log('got categories:',response.data);
        if(response&&response.data){
            setCategory(response.data.categories);
        }
    }catch(err){
        console.log('error:',err.message);
    }

}
```

Slike 15.Dohvaćanje dostupnih kategorija

Kao što se u prijašnjim dijelovima rada pisalo u React Native-u je dostupno kreiranje vlastitih komponenti koje će programerima biti dostupne kroz cijelo implementiranje aplikacije. Tako se u zaslonu 'Home' koriste dvije vlastite komponente: <Categories> i <Recipes>. Komponenta <Categories> služi za prikaz dostupnih kategorija jela. Ona kategorija koja je trenutno odabrana će se po prikazu razlikovati od ostalih u boji pozadine.

Primjer implementacije komponente <Categories> je slijedeći:

```
export default function Categories({categories,activeCategory,handleChangeCategory}){
    return(
        <Animated.View entering={BounceIn.duration(500).springify()}>
            <ScrollView
            horizontal
            showsHorizontalScrollIndicator={false}
            className="space-x-4"
            contentContainerStyle={{paddingHorizontal: 15}}>
            {
                categories.map((cat,index)=>{
                    let isActive = cat.strCategory==activeCategory;
                    let activeButtonClass = isActive? ' bg-amber-400': ' bg-black/10';
                    return(
                        <TouchableOpacity
                            key={index}
                            onPress={()=> handleChangeCategory(cat.strCategory)}
                            className="flex items-center space-y-1">
                            <View className={"rounded-full p-[6px]"+activeButtonClass}>
                                <Image
                                    source={{uri:cat.strCategoryThumb}}
                                    style={{width:hp(6),height:hp(6)}}
                                    className="rounded-full"/>
                            </View>
                            <Text className=" text-neutral-500" style={{fontSize:hp(1.6)}}>
                                {cat.strCategory}
                            </Text>
                        </TouchableOpacity>
                    )
                })
            }
            </ScrollView>
        </Animated.View>
    )
}
```

Slike 16.Implementacija <Categories> komponente

Unutar <Recipes> komponente jela odabrane kategorije se prikazuju u obliku liste unutar koje se nalaze slike i nazivi jela. Za ovakav prikaz koristi se <MasonryList> komponenta.

Primjer implementacije <Recipes> komponente je na slijedećoj slici:

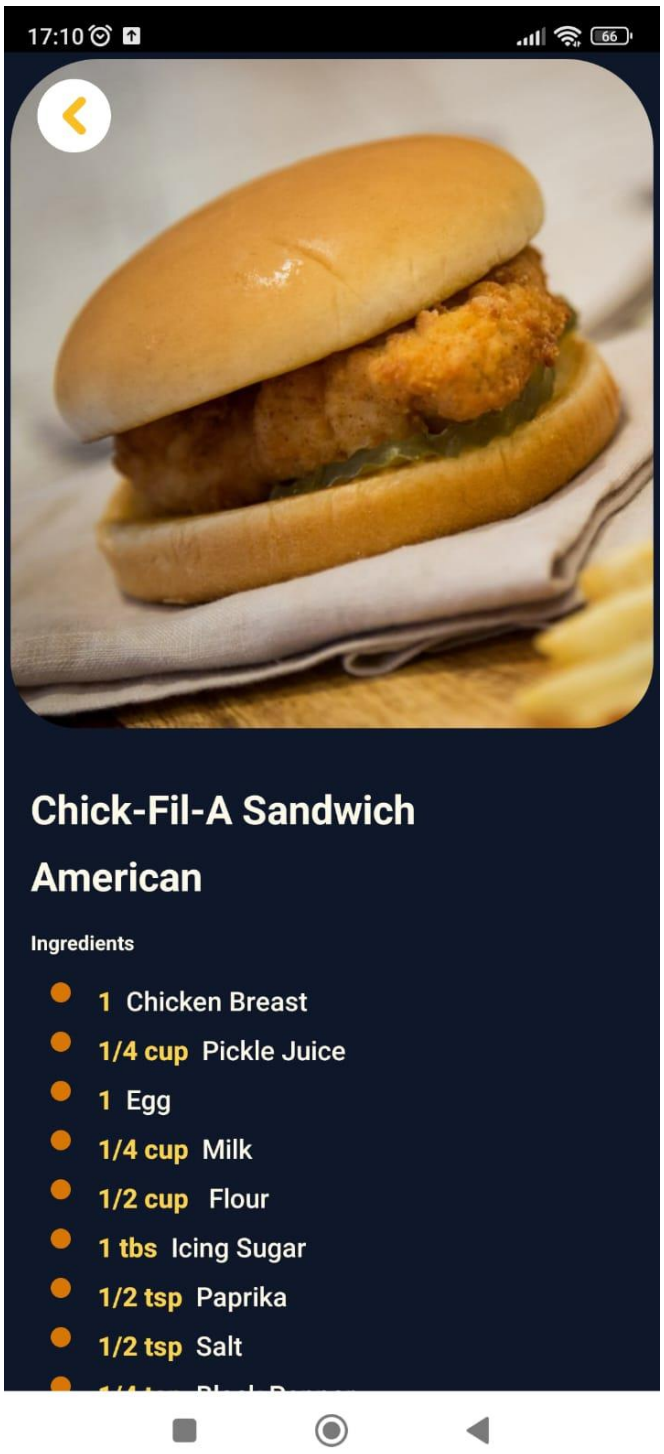
```
export default function Recipes({categories, meals}){
    Loading
    const navigation=useNavigation();
    return(
        <View clasName="mx-4 space-y-3">
            <Text style={{fontSize:hp(3)}} className="font-semibold text-amber-50">Recipes</Text>
            <View>
                {
                    categories.length==0 || meals.length==0 ? (
                        <Loading size="large" className="mt-20"/>
                    ): (
                        <MasonryList
                            data={meals}
                            keyExtractor={(item)=>item.idMeal}
                            numColumns={2}
                            showsVerticalScrollIndicator={false}
                            renderItem={({item,i}) => <RecipeCard item={item} index={i} navigation={navigation}/>}
                            //refreshing
                            ///onRefresh
                            onEndReachedThreshold={0.1}
                            //onEndReached={()=>loadNext(ITEM_CNT)}
                        />
                    )
                }

            </View>
        </View>

    )
}
```

Slike 17.Prikaz implementacije <Recipes> komponente

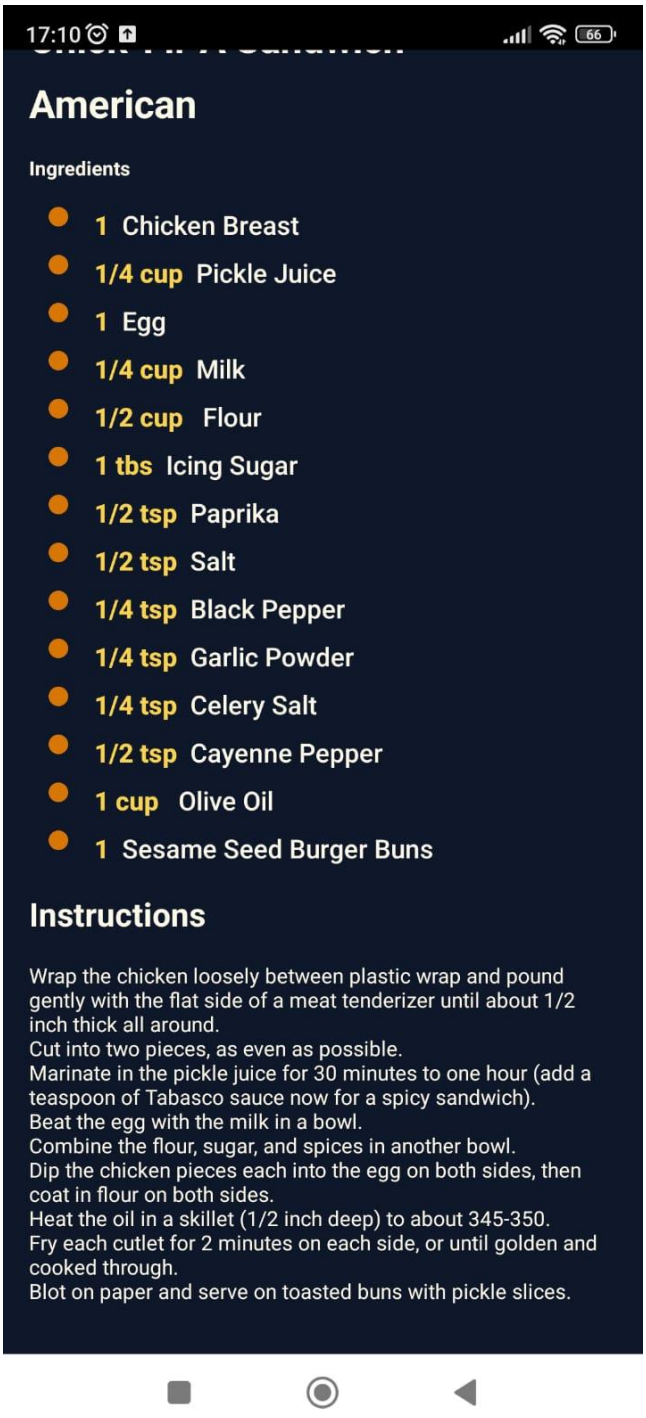
5.4. ZASLON 'DETAILS'

Nakon što je korisnik odabrao jelo čiji način pripreme želi istražiti otvara se zaslon 'Details'. Unutar ovog zaslona korisniku je dostupna velika slika jela, njegov naziv, zemlja podrijetla, sastojci te način pripreme ovog jela. U gornjem lijevom kutu je dostupna strelica za povratak na prijašnji zaslon.

Izgled zaslona 'Details' je na slijedećoj slici:



Slike 18.Izgled zaslona 'Details'



Slike 19.Prikaz instrukcija unutar zaslona 'Details'

Za dohvaćanje željenog jela koristi se ID jela te se šalje zahtjev sa željenim ID-om prema TheMealDB API-ju koji taj zahtjev obrađuje te vraća informacije željenog jela. Naknadno se sastojci obrađuju u obliku niza te ispisuju.

Primjer implementacije metoda za dohvaćanje jela te dohvaćanje niza sastojaka je na slijedećoj slici:

```
const getMeal = async(id)=>{
    try{
        const response = await axios.get(`https://themealdb.com/api/json/v1/1/lookup.php?i=${id}`);
        //console.log('got recipes:',response.data);
        if(response&&response.data){
            setMeal(response.data.meals[0]);
        }
    }catch(err){
        console.log('error:',err.message);
    }

}

const ingredientsIndex=(meal)=>{
    if(!meal) return[];
    let indexes=[];
    for(let i = 1;i<=20;i++){
        if(meal['strIngredient'+i]){
            indexes.push(i);
        }
    }
    return indexes;
}
```

Slike 20.Implementacija metoda za dohvaćanje jela i niza sastojaka

6. ZAKLJUČAK

Veliki porast dostupnosti mobilnih uređaja te njihovo korištenje u svakodnevnom životu dovelo je do povećanja tržišta mobilnih aplikacija. Većina velikih tvrtki i proizvođača želi biti aktualna te ponuditi svoje proizvode uz pomoć mobilnih aplikacija. Sve ovo stavlja veliki izazov za programere mobilnih aplikacija budući da na tržištu postoji veliki broj različitih uređaja sa različitom kvalitetom. Najpopularniji operacijski sustavi kod mobilnih uređaja su Android i iOS te programeri često moraju odlučiti za koji će sustav implementirati mobilnu aplikaciju. Napretkom tehnologije i pristupa izrade mobilnih aplikacija programeri imaju mogućnost kreiranja jedne mobilne aplikacije koja će biti kompatibilna sa više operacijskih sustava čime uštеđuju vrijeme i resurse potrebne za izradu. Svaka od ovih tehnologija ima zasebne prednosti i nedostatke te je odluka odabira na samom programeru. Pristup koji omogućuje kompatibilnost mobilne aplikacije na više operacijskih sustava naziva se više-platformski pristup. Najpopularniji okviri za ovakav razvoj aplikacija su React Native i Flutter. Veliku popularnost React Native može zahvaliti i JavaScript programskom jeziku koji je u programerskom svijetu vrlo popularan te je zbog toga logičniji izbor od Fluttera. Iako mobilne aplikacije kreirane više-platformskim pristupom nisu jednako dobre kao i izvorne aplikacije vjeruje se da će one i u budućnosti privlačiti pažnju programerima.

Tijekom izrade rada može se izdvojiti zaključak kako odabrano React Native okruženje može biti primjereno za poučavanje zbog toga što se temelji na poučavanju od konkretnog prema apstraktnom.

7. LITERATURA

1. How many people own smartphones? (2024-2029)

https://whatsthebigdata.com/smartphone-stats/#google_vignette

2. Holzer, Adrian & Ondrus, Jan. (2011). Mobile Application Market: A Developer's Perspective. Telematics and Informatics. 28. 22-31. 10.1016/j.tele.2010.05.006.

3. Global malnourishment: 1 in 8 people are obese

https://www.dw.com/en/global-malnourishment-1-in-8-people-are-obese/a-68407538

4. Mushtaq, Ziema & Kirmani, Mudasir & Andrabi, Syed Mohsin & Wahid, Abdul. (2018). Mobile Application Development: Issues and Challenges.

5. M. K. Khachouch, A. Korchi, Y. Lakhrissi and A. Moumen, "Framework Choice Criteria for Mobile Application Development," 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), Istanbul, Turkey, 2020, pp. 1-5, doi: 10.1109/ICECCE49384.2020.9179434.

6. Lachgar, Mohamed & Abdelmounaim, Abdali. (2017). Decision Framework for Mobile Development Methods. International Journal of Advanced Computer Science and Applications. 8. 10.14569/IJACSA.2017.080215.

7. Zarichuk, Oleksii. (2023). Comparative analysis of frameworks for mobile application development: Native, hybrid, or cross-platform solutions. Вісник Черкаського державного технологічного університету. 28. 19-27. 10.62660/2306-4412.4.2023.19-27.

8. Lachgar, Mohamed & Abdelmounaim, Abdali. (2015). Survey of mobile development approaches.

9. Lewis, S., Dunn, M. (2019). Native Mobile Development: A Cross-Reference for IOS and Android. Sjedinjene Američke Države: O'Reilly Media.

10. D. Sin, E. Lawson and K. Kannoorpatti, "Mobile Web Apps - The Non-programmer's Alternative to Native Applications," 2012 5th International Conference on Human System Interactions, Perth, WA, Australia, 2012, pp. 8-15, doi: 10.1109/HSI.2012.11.

11. Andre Charland and Brian LeRoux. 2011. Mobile Application Development: Web vs. Native: Web apps are cheaper to develop and deploy than native apps, but can they match the native user experience? Queue 9, 4 (April 2011), 20–28. https://doi.org/10.1145/1966989.1968203

12. N. Serrano, J. Hernantes and G. Gallardo, "Mobile Web Apps," in IEEE Software, vol. 30, no. 5, pp. 22-27, Sept.-Oct. 2013, doi: 10.1109/MS.2013.111.

13. Huynh, M., Ghimire, P., & Truong, D. (2017). Hybrid app approach: Could it mark the end of native app domination? Issues in Informing Science and Information Technology Education, 14, 49-65. http://www.informingscience.org/Publications/3723

14. Denko, Blaž & Pecnik, Spela & Fister jr, Iztok. (2021). A Comprehensive Comparison of Hybrid Mobile Application Development Frameworks. International Journal of Security and Privacy in Pervasive Computing. 13. 78-90. 10.4018/IJSPPC.2021010105.

15. Hartmann, Gustavo, Geoff Stead, and Asi DeGani. "Cross-platform mobile development." Mobile Learning Environment, Cambridge 16.9 (2011): 158-171.

16. Amatya, Suyesh & Kurti, Arianit. (2014). Cross-Platform Mobile Development: Challenges and Opportunities. 10.1007/978-3-319-01466-1_21.

17. Sommer, Andreas; Krusche, Stephan (2013): Evaluation of cross-platform frameworks for mobile applications. Software Engineering 2013 - Workshopband. Bonn: Gesellschaft für Informatik e.V.. PISSN: 1617-5468. ISBN: 978-3-88579-609-1. pp. 363-376

18. Adam Boduch(2017): React and React Native, PACKT PUBLISHING PACKT PUBLISHING, ISBN: 9781786465658

19. 4 Most Popular Cross-Platform App Development Frameworks for 2024

https://www.thedroidsonroids.com/blog/top-cross-platform-app-development-frameworks

20. Kotlin Multiplatform

https://kotlinlang.org/docs/multiplatform.html

21. Mobile Operating System Market Share Worldwide

https://gs.statcounter.com/os-market-share/mobile/worldwide

22. Wu, W. (2018). React Native vs Flutter, Cross-platforms mobile application frameworks.

23. Paul, A., Nalwaya, A. (2019). React Native for Mobile Development: Harness the Power of React Native to Create Stunning IOS and Android Applications. Njemačka: Apress.

24. Writing Markup with JSX

https://react.dev/learn/writing-markup-with-jsx

25. Cascading Style Sheets home page

https://www.w3.org/Style/CSS/Overview.en.html

26. Tap Academy

https://thetapacademy.com/

27. Tailwind CSS

https://tailwindcss.com/

28. Expo

https://docs.expo.dev/

29. M Biehl. "API Architecture The Big Picture For Building APIs". ISBN-13: 978-1508676645 ISBN-10: 150867664X

8. POPIS SLIKA