Mobilna aplikacija za upravljanje redovima čekanja

Duda, Alen

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:025906

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-18**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli

Odjel za informacijsko-komunikacijske tehnologije

ALEN DUDA

MOBILNA APLIKACIJA ZA UPRAVLJANJE REDOVIMA ČEKANJA

Diplomski rad

Pula, travanj 2018. godine

Sveučilište Jurja Dobrile u Puli

Odjel za informacijsko-komunikacijske tehnologije

ALEN DUDA

MOBILNA APLIKACIJA ZA UPRAVLJANJE REDOVIMA ČEKANJA

Diplomski rad

JMBAG: 0303012620, redoviti student

Studijski smjer: Informatika

Predmet: Izrada informatičkih projekata

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Siniša Sovilj

Pula, 06. travnja 2018. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Alen Duda, kandidat za magistra informatike ovime izjavljujem da je ovaj diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio seminarskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, 11.04.2018. godine

IZJAVA

o korištenju autorskog djela

Ja, Alen Duda, dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „Mobilna aplikacija za upravljanje redovima čekanja“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 11. travnja 2018. godine

Potpis

_____

SVEUČILIŠTE JURJA DOBRILE U PULI
ODJEL ZA INFORMACIJSKO-KOMUNIKACIJSKE TEHNOLOGIJE

Pula, 1. ožujka 2017.

DIPLOMSKI ZADATAK

Pristupnik:	**Alen Duda (0303012620)**
Studij:	Sveučilišni diplomski studij Informatike
Naslov (hrv.):	**Mobilna aplikacija za upravljanje redovima čekanja**
Naslov (eng.):	Queue managing mobile application
Opis zadatka:	Zadatak je razviti mobilnu Android aplikaciju za pregled redova čekanja u raznim ustanovama te mogućnost rezerviranja mjesta u redovima. Omogućiti korisnicima prijavu u aplikaciju stvaranjem posebnog računa ili postojećim Facebook ili Google korisničkim računom. Korisnika obavijestiti kad je uskoro na redu ovisno o njegovim postavkama. Istražiti tehnologije koje mobilnoj aplikaciji omogućavaju pristup i upravljanje podacima korištenjem REST API-a na udaljenom poslužitelju.

Zadatak uručen pristupniku: 1. ožujka 2017.
Rok za predaju rada: 1. veljače 2018.

Mentor:

Siniša Sovilj

_____

doc.dr.sc. Siniša Sovilj

Sadržaj

UVOD 1
1 NAČINI IZRADE MOBILNIH APLIKACIJA 2
1.1 Android Studio 3
1.2 Višeplatformni alternativni načini 5
1.2.1 Xamarin 5
1.2.2 React Native 6
1.2.3 Hibridne mobilne aplikacije 9
1.3 Zaključak istraživanja i odabir tehnologije 11
2 PLANIRANJE RAZVOJA APLIKACIJE 12
2.1 Interakcija i nužne mogućnosti aplikacije 12
2.2 Sučelje aplikacije 13
2.3 Shema baze podataka 18
3 POZADINSKI SUSTAV I ADMINISTRACIJSKO SUČELJE 20
3.1 Tehnologije i programski alati 20
3.1.1 Baza podataka 20
3.1.2 Pozadinski sustav 20
3.1.3 Administracijsko sučelje 22
3.1.4 Programski alati 22
3.1.5 Arhitektura sustava 23
3.2 REST sučelje 24
3.2.1 Korisnici 25
3.2.2 Ustanove 27
3.2.3 Redovi 28
3.2.4 Rezervacije 29
3.2.5 Stvaranje rezervacije (uzimanje broja) 30
3.2.6 Prelazak na sljedećeg korisnika u redu 31

3.3 Administracijsko sučelje (Web aplikacija) ... 33
4 MOBILNA APLIKACIJA ... 36
4.1 Tehnologije i alati ... 36
4.1.1 Osnovne komponente ... 36
4.1.2 Dodatne biblioteke ... 37
4.1.3 Obrazac dizajna ... 38
4.1.4 Izvođenje, testiranje i otklanjanje grešaka ... 38
4.2 Sučelje i interakcija ... 39
4.2.1 Navigacija ... 39
4.2.2 Prijava ... 40
4.2.3 Registracija ... 42
4.2.4 Popis ustanova ... 43
4.2.5 Detalji ustanove ... 44
4.2.6 Popis redova ... 46
4.2.7 Detalji o redu – rezerviranje broja ... 47
4.2.8 Popis rezervacija i prikaz obavijesti ... 48
4.2.9 Postavke profila ... 51
5 MOGUĆA POBOLJŠANJA ... 53
5.1 Dodatni podaci o ustanovama i redovima ... 53
5.2 Statistika i administracija ... 53
5.3 Firebase integracija ... 54
5.4 Korisničko iskustvo ... 54
ZAKLJUČAK ... 56
LITERATURA ... 57
POPIS SLIKA ... 59
POPIS PRILOGA ... 60
SAŽETAK ... 61

SUMMARY 61

UVOD

Nitko ne voli čekati u redu. Ipak, za neke je to neizbježan problem i zapreka prilikom obavljanja administrativnih dužnosti poput plaćanja računa, ugovaranja zdravstvenog osiguranja, izrade ili produženja osobne iskaznice, putovnice te, iz studentske perspektive, upisivanja više godine, preuzimanja potvrda i slično. Zbog radnog vremena nekih ustanova, ljudi često moraju izbivati s posla ili potrošiti vrijeme predviđeno za marendu i nadati se kako će izbjeći gužvu – samo kako bi uvidjeli da popriličan broj istomišljenika već čeka u redu ispred njih.

Čest razlog za početak učenja programiranja je strast za rješavanjem problema. U današnje vrijeme, kad postoje aplikacije za gotovo sve, nema razloga zašto ne bi postojao i sustav za uklanjanje redova čekanja. Upravo to je cilj ovog rada – stvaranje sustava koji uz pomoć mobilne aplikacije omogućava ljudima da izbjegnu dugotrajno čekanje u redu na način da u svako vrijeme i na bilo kojem mjestu mogu vidjeti koliko je ljudi u nekom redu te rezervirati svoje mjesto u njemu. Takav sustav u praksi može biti vrlo složen pa je za ovaj rad odabran relativno jednostavan pristup – izrada aplikacije kao dokaz koncepta.

Rad se, osim uvoda i zaključka, sastoji od pet poglavlja. U prvom poglavlju istražuju se najpopularniji načini izrade mobilnih aplikacija, važu se njihove prednosti i nedostaci te odabire najprikladniji za ovaj projekt. Drugo poglavlje bavi se konceptima izrade sustava i aplikacije – izrađen je osnovni prototip sučelja, ustanovljeni potrebni podaci i prikazi, predviđen način interakcije korisnika s aplikacijom te određena shema baze podataka na način da se sustav može izraditi korištenjem bilo koje tehnologije. Potom se u trećem poglavlju kreće s tehničkim pojedinostima izrade sustava – odabir baze podataka, izrada Web usluge i prateće Web aplikacije za administratore. Opisat će se odabrane tehnologije te funkcionalnosti koje se pomoću njih ostvaruju. Četvrto se poglavlje bavi izrađenom mobilnom aplikacijom kao glavnim instrumentom interakcije korisnika i sustava za upravljanje redovima čekanja. Opisat će se korišteni alati, postupak izrade Android aplikacije – korištene tehnike, metode, obrasci dizajna, komponente, pomoćne biblioteke i navesti izazovi koje predstavlja izrada ovakve aplikacije. Peto i završno poglavlje nabrojat će neka od mogućih poboljšanja koja ovakav sustav treba kako bi mogao zaživjeti i u praksi te omogućio ljudima da više vremena provedu s bližnjima, a manje u nepredvidivim redovima čekanja.

1 NAČINI IZRADE MOBILNIH APLIKACIJA

U ovom će se poglavlju istražiti koje su najpopularnije tehnologije i načini za izradu mobilnih aplikacija za Android mobilne uređaje. Budući da je glavni cilj ovog rada izrada upravo Android aplikacije, ovo istraživanje ima i svoj cilj – odabrati najprikladniju tehnologiju od svih proučenih te pomoću nje izraditi mobilnu aplikaciju.

Istraživanje se provodi proučavajući statistiku najkorištenijih biblioteka u Android aplikacijama (AppBrain, 2018) kako bi se pronašle najpopularnije alternativne tehnologije za izradu Android aplikacija, a potom se pronađene tehnologije treba proučiti pomoću kvalitetnijih materijala na platformama YouTube[1] i Udemy[2] kako bi se identificiralo one najzastupljenije i najtraženije – kako među predavačima, tako i među polaznicima tečajeva. Potom će se, proučavajući službenu dokumentaciju svake od odabranih tehnologija, istaknuti prednosti i nedostatke svake od njih kako bi se odabrala najprikladnija.

Tehnologije i načini za izradu Android aplikacija mogu se podijeliti u nekoliko kategorija:

- izvorno (engl. *native)* programiranje koristeći **Android Studio** rabeći programske jezike **Java** ili **Kotlin**
- korištenje dodatka **Xamarin** za Microsoft Visual Studio za višeplatformno programiranje uz pomoć jezika **C#**
- korištenje **JavaScript** programskog okvira (engl. *framework)* **React Native** za višeplatformno programiranje
- **hibridna** izrada korištenjem Web tehnologija (**HTML5**, **CSS3**, **JavaScript**) uz pomoć programskih okvira **Ionic** i **PhoneGap**

Postoji još tehnologija osim navedenih i teško ih je kategorizirati te će se stoga u daljnjem tekstu odabrati predstavnici najpopularnijih jer i unutar navedenih kategorija te tehnologije međusobno imaju alternative, a njihove detaljne razlike su izvan opsega ovog rada.

[1] Mrežna platforma za dijeljenje video sadržaja, dostupna na http://www.youtube.com/
[2] Mrežna platforma za dijeljenje tečajeva i predavanja o digitalnim temama u video formatu, dostupna na http://www.udemy.com/

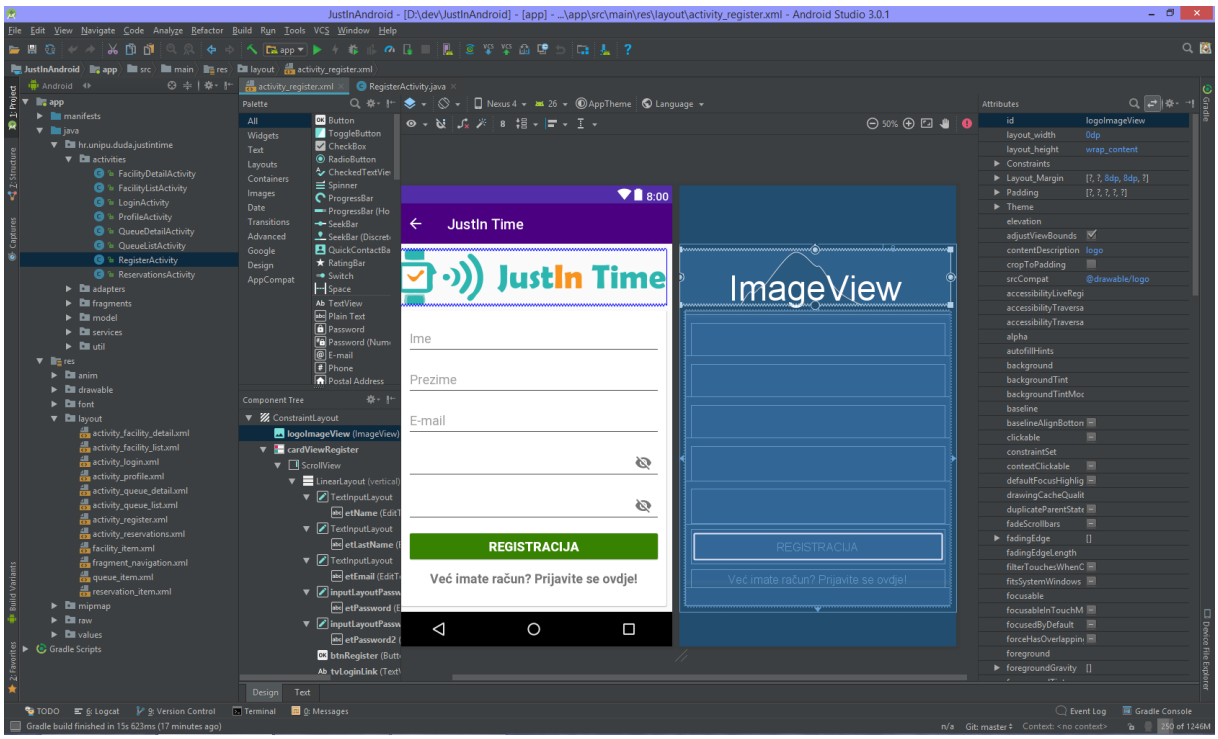
1.1 Android Studio

Prije istraživanja alternativnih načina za izradu Android mobilnih aplikacija, riječi mora biti o standardnom i najkorištenijem alatu, a to je Android Studio.

Nastao je i dalje se razvija kroz suradnju tvrtki Google i JetBrains, a temelji se na besplatnoj inačici Java integriranog razvojnog okruženja IntelliJ IDEA. Početkom 2015. godine zamijenio je Eclipse, koji je također Java integrirano razvojno okruženje, kao službeni alat za izradu Android aplikacija.

Glavne značajke (Duda, 2015):

- pregledno sučelje poznato Java programerima,
- moćan uređivač koda i dizajna aplikacije,
- pomoćni alati za testiranje i otklanjanje pogrešaka,
- emulator na kojemu se mogu pokrenuti aplikacije tijekom izrade bez potrebe za spajanjem fizičkog uređaja,
- mjerenje iskorištenosti resursa tijekom izvođenja (profiliranje) aplikacije,
- odlična integracija sa sustavima za verzioniranje koda



Slika 1: Android Studio.
Izvor: autor

Tijekom pisanja ovog rada objavljena je inačica 3.0 koja donosi mnogobrojne novosti i poboljšanja, od kojih su najzanimljivije podrška za programski jezik Kotlin i mogućnost korištenja nekih jezičnih značajki programskog jezika Java 8 (primjerice lambda[3] izraza), bez potrebe za korištenjem dodataka treće strane. (Google, 2017)

Mnoge su **prednosti** korištenja Android Studio-a za izradu mobilnih aplikacija, osim već navedenih značajki i potpunog Android razvojnog okruženja koje ga prati, programski jezik Java koji je prisutan desetljećima te je stoga široko rasprostranjen među programerima i najlakše je pronaći gotove primjere koda i ustanovljenih najboljih praksi iz raznih izvora te je dodatno poboljšana podrška uvođenjem Java 8 mogućnosti. Valja istaknuti da bi se, kako nema nikakvog prevođenja koda ni slojeva podrške poput Web preglednika, aplikacije izrađene metodom izvornog programiranja u pravilu trebale brže izvoditi od alternativnih načina. Također, osigurana je programska podrška za svo specifično sklopovlje koje Android mobilni uređaji posjeduju.

Još jedna od prednosti izvornog programiranja je to što sučelje aplikacije izgleda primjereno platformi za koju se izrađuje. Tako korisnik prilikom korištenja aplikacije ima poznato iskustvo koje očekuje i prepoznaje u ostalim aplikacijama dizajniranim za platformu. (Cruxlab, Inc., 2017)

Ne smije se zanemariti okretanje prema Kotlin jeziku koji uživa sve veću podršku i popularnost te postoji mogućnost da će s vremenom zamijeniti Javu kao najzastupljeniji programski jezik na Android platformi (slično kao što je Swift zamijenio Objective-C programski jezik za iOS platformu), a to može zahvaliti svojem modernom stilu pisanja (čista sintaksa, manje potrebnog koda u odnosu na ekvivalentne rezultate u Javi, izbjegavanje čestih pogrešaka nepostojećih pokazivača…) i potpunoj kompatibilnosti s postojećim Java kodom – moguće je istovremeno koristiti se obama jezicima u istom projektu te postojeći Java kod pretvoriti u Kotlin sintaksu izravno iz Android Studio-a. (Google, 2017)

Najveći **nedostatak** izvornog programiranja korištenjem Android Studio-a za izradu mobilnih aplikacija je to što se gotov proizvod može plasirati samo na Android mobilne uređaje – za druge platforme potrebno je stvoriti odgovarajuće aplikacije ispočetka

[3] anonimne funkcije koje smanjuju količinu koda potrebnog za izvršavanje određenih akcija, posebno korisne prilikom pisanja funkcija koje reagiraju na događaje unutar grafičkog korisničkog sučelja

korištenjem izvornog programiranja za platforme u pitanju (konkretno, program Visual Studio i jezik C# za Windows Phone platformu te program XCode i jezik Swift za iOS uređaje). Taj je nedostatak izraženiji kod poslovnih korisnika gdje se cijena projekta uvećava za svaku platformu koju treba podržati jer je često potrebno zaposliti više programera (ili timova programera) koji su specijalizirani za svoje mobilne platforme.

Upravo zbog tog razloga se sve češće poseže za alternativnim načinima izrade mobilnih aplikacija koji su po prirodi višeplatformni – cilj je jednom napisan kod plasirati na barem dvije najpopularnije platforme (Android i iOS) čime se smanjuju troškovi izrade, nadogradnje i daljnjeg održavanja aplikacija.

1.2 Višeplatformni alternativni načini

Kako bi se pokrilo što je moguće veći postotak tržišta, izrađene aplikacije moraju funkcionirati na najkorištenijim platformama. Ranije je pojašnjeno kakve troškove to može prouzrokovati, a zbog razlika u platformama nemoguće je ponovno iskoristiti postojeći kod.

Niže opisane tehnologije imaju zadaću olakšati stvaranje aplikacija za više platformi tako da se koristi samo jedan sustav tehnologija (ovisno o okviru). Time se postiže velik broj prednosti (Cruxlab, Inc., 2017):

- programer ne mora znati dvije (ili više) različitih tehnologija kako bi stvorio jedno te isto sučelje i funkcionalnost, ali treba biti upoznat s nekim specifičnostima i ograničenjima svake platforme
- dijeljenje koda smanjuje količinu duplikata – poslovnu logiku i mrežne upite može se, primjerice, odvojiti u posebne module te se time ne gubi mogućnost stvaranja različitog sučelja za svaku platformu
- olakšan rad više programera na istom projektu – kako se zajednički dio koristi za više platformi, ne mogu se dogoditi razlike u mogućnostima aplikacije (ali se implementaciju može prilagoditi odabranoj platformi)

1.2.1 Xamarin

Xamarin je programski okvir za izradu višeplatformnih mobilnih aplikacija koji koristi programski jezik C#. Dio je integriranog razvojnog sučelja Visual Studio tvrtke Microsoft. Može zamijeniti izvorne alate u potpunosti ili samo djelomično. Primjerice,

sučelje aplikacije može se izraditi koristeći Android Studio dizajner ili Xamarin Studio alate, a logički se dio koda napiše koristeći C# umjesto Jave ili Swifta. Kod koji ne ovisi o sučelju slobodno se može dijeliti između Android i iOS projekata. Za istovremenu, višeplatformnu izradu sučelja može se koristiti Xamarin.Forms koji dijelove sučelja zamjenjuje izvornim izgledom ciljne platforme.

Ne mogu se koristiti jezične biblioteke specifične za određenu platformu (na koje su izvorni programeri već navikli), već je potrebno koristiti višeplatformne C# biblioteke koje možda nemaju sve mogućnosti kao i izvorne. Ipak, postoje poveznice na izvorne biblioteke Android i iOS platformi koje je moguće koristiti, a to su najčešće one za specifično sklopovlje. Podrška ide tako daleko da se nastoji zadržati ista imena funkcija u povezanim bibliotekama. Dodatne poteškoće se mogu javiti zbog razlika u traženju dozvola od korisnika, budući da se razlikuju u opsegu i izgledu sučelja upita pa to obično treba pojedinačno prilagoditi svakoj platformi.

C# se ne prevodi u izvorni kod platforme, već u međukod (engl. *bytecode)* kojeg se mora prevesti u izvorni kod tijekom (JIT[4]) ili prije samog izvođenja (AOT[5]) aplikacije na uređaju. Performanse su vrlo blizu izvornim aplikacijama i sučelje izgledom odgovara platformi. (Cruxlab, Inc., 2017)

S gledišta Android programera, programski jezik C# vrlo nalikuje na Javu što bi u pravilu trebalo olakšati i ubrzati prilagodbu na Xamarin. Međutim, višeplatformni dijelovi nemaju previše dodirnih točaka s Android Studiom i zahtijevaju učenje ispočetka da bi se učinkovito koristili. Proces izrade aplikacije najsličniji je izvornom programiranju, kao i brzina izvođenja, izgled sučelja te podrška i integracija u izvorne operacijske sustave.

1.2.2 React Native

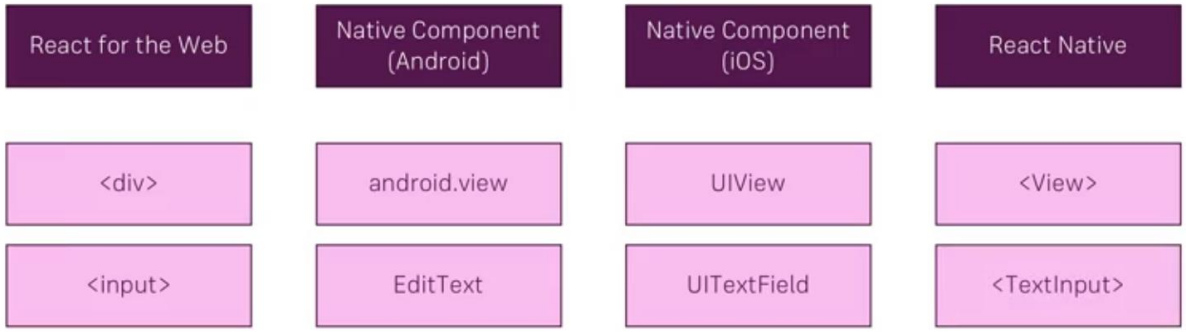
React Native je JavaScript programski okvir za izradu višeplatformnih mobilnih aplikacija korištenjem slične metodologije kao i okvir kojeg nasljeđuje – **React** JavaScript biblioteka koja se vrlo često koristi za izradu sučelja (engl. *frontend)* Web aplikacija. Budući da koristi isti jezik i metodologiju, vrlo je pristupačan programerima koji su specijalizirani ili imaju iskustva s Web tehnologijama.

Programiranjem koristeći React Native ne stvara se Web ni hibridna aplikacija,

[4] engl. *just in time* – točno na vrijeme
[5] engl. *ahead of time* – prije vremena

već prava mobilna aplikacija koja se ne razlikuje od izvorne pisane u Javi ili Swiftu. Koriste se isti sastavni dijelovi kao kod izvornog programiranja, samo što se grade koristeći JavaScript programski jezik. Podržane platforme su Android i iOS. Koristi se za izradu nekih od najpopularnijih mobilnih aplikacija, kao što su Facebook, Instagram, Airbnb, Skype i druge. (Facebook, 2018)

Za razliku od React biblioteke, koja koristi mješavinu JavaScript i HTML jezika za prikaz korisničkog sučelja nazvanu JSX, za mobilne se aplikacije ne smije koristiti HTML oznake kao komponente jer ne postoji DOM[6] kao na Web stranici. Umjesto toga koriste se posebne React Native komponente koje služe kao adapter između JavaScripta i izvornog koda platforme (slično kao što se Xamarin.Forms prevodi u izvorne komponente korisničkog sučelja ciljne platforme, op. a.). JavaScript logika se, s druge strane, ne prevodi u izvorni kod, već se pokreće prilikom izvođenja aplikacije. (Schwarzmüller, 2017)



Slika 2: Usporedba odgovarajućih Web, Android, iOS i React Native komponenti. Izvor: (Schwarzmüller, 2017)

React filozofija prenosi se na React Native tako što korisničko sučelje postaje funkcija trenutnog stanja, što je ključan element React biblioteke. Prilikom promjene stanja, React brine o osvježavanju dijelova sučelja koji ovise o toj promjeni. Tako se kompleksnim sučeljima može lakše upravljati jer se logika koja osvježava sučelje ne nalazi na više različitih mjesta u kodu. (Cruxlab, Inc., 2017)

Nažalost, to sučelje nema izvorni izgled platforme na kojoj se izvodi. Potrebno je pojedinačno ručno stilizirati komponente. Dostupan je ograničen broj osnovnih komponenti i alati za rad s responzivnošću[7] korisničkog sučelja, ali aplikacije nisu

[6] engl. Document Object Model, struktura podataka slična stablu od koje je sačinjena svaka Web stranica
[7] reagiranje aplikacije na promjenu orijentacije uređaja, različite veličine zaslona i sl.

responzivne same po sebi. Kao i s React bibliotekom, često se u praksi koriste dodaci treće strane za stiliziranje, gotove komponente i responzivne dizajne sučelja. Također, većina JavaScript biblioteka koje ne ovise o Web pregledniku ni DOM-u bi u pravilu trebala u potpunosti funkcionirati. Podrška za specifično sklopovlje i integracija s operacijskim sustavom uređaja je u stalnom razvoju te je iz dana u dan sve bolja. (Schwarzmüller, 2017)

React Native pruža jedinstvenu mogućnost pisanja dijela aplikacije koristeći izvorno programiranje za ciljnu platformu. Na taj se način dijelovi koje je teže prilagoditi React sustavu mogu napisati koristeći Javu ili Swift.

Dodatna je prednost mogućnost trenutnog osvježavanja aplikacije nakon svake izmjene, koja je vrlo poznata Web programerima. Umjesto da se cjelokupan projekt mora prevesti i aplikacija reinstalirati na uređaju (što, ovisno o veličini i kompleksnosti projekta, može trajati i nekoliko minuta), u već pokrenutoj aplikaciji se osvježi samo dio koji je promijenjen u kodu, bez da se gubi stanje unutar aplikacije. Time se tijekom razvoja može uštediti mnogo vremena, naročito kod malih i čestih promjena koje se javljaju prilikom detaljne prilagodbe dizajna aplikacije. (Cruxlab, Inc., 2017)

Donja slika prikazuje primjer korištenja React Native programskog okvira za prikaz popisa ustanova i redova unutar svake od njih, uz minimalno stiliziranje. Izgled aplikacije je gotovo identičan na Android i iOS platformama.

```
import React, { Component } from 'react';
import { AppRegistry, SectionList, StyleSheet, Text, View } from 'react
  -native';

export default class SectionListBasics extends Component {
  render() {
    return (
      <View style={styles.container}>
        <SectionList
          sections={[
            {title: 'FET', data: ['Referada', 'ISVU služba']},
            {title: 'MUP', data: ['Osobne iskaznice', 'Putovnice',
              'Oružje']},
          ]}
          renderItem={({item}) => <Text style={styles.item}>{item}</Text>}
          renderSectionHeader={({section}) => <Text style={styles
            .sectionHeader}>{section.title}</Text>}
          keyExtractor={(item, index) => index}
        />
      </View>
    );
  }
}
```

```
2:34
FET
Referada
ISVU služba
MUP
Osobne iskaznice
Putovnice
Oružje
```

Slika 3: Primjer stvaranja popisa korištenjem React Native okvira.
Izvor: https://facebook.github.io/react-native/docs/using-a-listview.html

React Native se čini odličan izbor za izradu višeplatformnih mobilnih aplikacija za programere koji dolaze iz Web svijeta gdje je React jedna od najpopularnijih biblioteka za izradu korisničkih sučelja. Za izvorne programere koji žele početi višeplatformni razvoj aplikacija (i nisu koristili React) nije najbolji izbor zbog svoje specifične sintakse i drukčije metodologije.

1.2.3 Hibridne mobilne aplikacije

Popularizacijom pametnih mobilnih uređaja u posljednjem desetljeću dolazi do nužnosti prilagodbe klasičnih Web stranica od velikih računalnih zaslona prema mobilnim i tabletnim uređajima. Tako se javlja responzivni Web dizajn u kojem Web stranica izgleda različito (i primjereno) uređaju na kojem se prikazuje. Jedan od suvremenih pristupa je i prvenstveno izrada za mobilne uređaje (engl. *mobile first)* gdje se prvo određuje što će se i kako prikazati na mobilnim uređajima, a tek kasnije se gradi sučelje za veće zaslone. Na taj se način želi prikazati samo najbitniji sadržaj na najmanjim zaslonima čime se štedi vrijeme i povećava korisnikova percepcija korisnosti Web stranice, bez da mu se ograniči količina dostupnog sadržaja (ostatak sadržaja je dostupan pomoću raznih poveznica, a najčešće je moguće pristupiti i inačici Web stranice za stolna računala).

Kako je Web jedna od najpopularnijih platformi, logično je da ima mnogo programera koji se bave karakterističnim tehnologijama te platforme – HTML jezikom za opis sadržaja, CSS stilskim jezikom koji upravlja izgledom i razmještajem Web stranica te već spomenutim JavaScriptom za poslovnu logiku i dinamičku interakciju korisnika sa sučeljem. Da bi se Web stranica smatrala Web aplikacijom, mora imati i pozadinsku logiku na poslužitelju, primjerice korištenjem jezika PHP, Python, Ruby ili JavaScript (uz pomoć Node.js[8] izvedbenog okvira). Ovako stvorene Web aplikacije rade na svim uređajima, ali upravo zbog toga nisu prilagođene ni jednoj mobilnoj platformi – kako izgledom sučelja, tako ni performansama ni mogućnošću korištenja specifičnog sklopovlja platforme na kojoj se prikazuju (ograničene su mogućnostima Web preglednika uređaja).

Prirodno je da se javila potreba za korištenjem široko rasprostranjenih Web tehnologija za izradu mobilnih aplikacija, odnosno za pretvaranje postojećih Web

[8] sustav za izvođenje JavaScript programskog jezika na poslužitelju (kojeg je inače moguće izvoditi samo unutar Web preglednika) korištenjem JavaScript prevoditelja preglednika Chrome tvrtke Google

aplikacija i mobilnih Web stranica u mobilne aplikacije koje izgledaju prirodnije na ciljnoj platformi i bolje se integriraju sa sučeljem, operacijskim sustavom i sklopovljem.

Ranije spomenuta React biblioteka je jedan od najpopularnijih načina za izradu sučelja Web aplikacija. Još jedan vrlo popularan način je Angular JavaScript programski okvir. Upravo korištenjem Angular okvira moguće je pretvoriti postojeće Web aplikacije u **hibridne** mobilne aplikacije. Za to je potrebno koristiti programski okvir **Ionic** koji sadrži elemente sučelja koji izgledom odgovaraju izvornim elementima mobilnih aplikacija na ciljnim platformama. Angular programski okvir može se, ali i ne mora koristiti – po želji se može koristiti samo elemente sučelja. Ionic prati Angular pa tako postoje inačice za Angular i Angular 2 koje su odgovarajuće nazvane Ionic i Ionic 2.

Za pokretanje Ionic aplikacije na mobilnom uređaju koristi se tehnologija **Cordova** ili **PhoneGap**. PhoneGap je distribucija tehnologije Cordova te je pomoću nje moguće unutar prozora preglednika na mobilnom uređaju izvoditi Web stranicu. Kombinacijom tehnologija Ionic i Cordova dobije se mobilna aplikacija koja se sastoji samo od prikaza preglednika unutar kojeg se može nalaziti Web stranica odnosno aplikacija koja se izgledom ne razlikuje od izvorne aplikacije platforme na kojoj se izvodi. Takvu aplikaciju moguće je distribuirati kao izvornu na Android, iOS i Windows Phone platforme. PhoneGap se može koristiti i bez Ionic-a, međutim tad je potrebno ručno stilizirati uobičajene Web elemente sučelja ili koristiti dodatke koji to olakšavaju. Pomoću Cordova dodataka omogućava interakciju s mobilnim uređajem kroz programsko sučelje pa se tako može pristupiti kameri, imeniku, lokacijskim uslugama, pohrani i ostalim mogućnostima. (Traversy, 2017)

Budući da se za prikaz sučelja koriste HTML elementi stilizirani da izgledaju što je moguće sličnije izvornim elementima platforme – za što je potreban okvir Web preglednika – hibridne mobilne aplikacije imaju nešto lošije performanse u usporedbi s ostalim navedenim načinima za izradu mobilnih aplikacija. Također može doći do problema koji je prisutan na Web-u od njegovog početka – različit izgled aplikacije na različitim uređajima. Srećom, prisutno je još nešto karakteristično za Web programiranje, a to je relativno brzo osvježavanje i pregled nakon učinjenih izmjena u kodu. (Cruxlab, Inc., 2017)

Stvaranje hibridnih mobilnih aplikacija je zanimljiv kompromis između boljih

performansi kakve pružaju ostali načini izrade mobilnih aplikacija i modernog razvojnog procesa. Hibridne aplikacije izgledom sučelja odgovaraju ciljnoj platformi i djeluju kao odličan izbor za Web programere koji imaju iskustva s korištenjem Angular programskog okvira, a žele se okušati u izradi mobilnih aplikacija.

1.3 Zaključak istraživanja i odabir tehnologije

U ovom poglavlju su proučene najpopularnije tehnologije za izradu mobilnih aplikacija te njihove karakteristike. Na temelju istraživanja i prethodnog iskustva odabrat će se najprikladnija za izradu Android mobilne aplikacije za upravljanje redovima čekanja.

Android Studio ima najbolju podršku i najzastupljeniji je od svih spomenutih metoda izrade mobilnih aplikacija. Najlakše će biti pronaći dodatne biblioteke, primjere koda i najbolje prakse iz postojećih projekata otvorenog koda. Njegov glavni nedostatak – stvaranje aplikacije samo za jednu platformu – nije bitan za ovaj rad, budući da će se izrađivati samo Android mobilna aplikacija.

Višeplatformni načini izrade razlikuju se tehnologijama i performansama i svaki od njih ima neku svoju ciljanu skupinu programera koji već znaju koristiti potrebne tehnologije – Xamarin je najprimjereniji za C# programere s iskustvom rada u tehnologijama tvrtke Microsoft, dok su React Native i hibridne mobilne aplikacije bolji izbor za Web programere koji su radili s programskim okvirima React i Angular. Hibridne aplikacije imaju nekoliko nedostataka koji bi mogli stvoriti probleme u izradi aplikacije za upravljanje redovima čekanja, konkretno otežano izvođenje koda u pozadini i najlošije performanse od svih proučenih alternativa.

Alternativni načini izrade Android aplikacija imaju prednost kad se izrađuje višeplatformne aplikacije i ako se već zna programirati u navedenim JavaScript okvirima, odnosno ako već postoji Web aplikacija koju se želi prenijeti na mobilnu platformu.

Kako je za ovaj rad cilj izraditi samo Android mobilnu aplikaciju, a Web aplikacija ne postoji te zbog prethodnog osobnog i fakultetskog iskustva s tim načinom, koristit će se **Android Studio** s **Java** programskim jezikom uz mogućnost eksperimentiranja s Kotlinom za **izvorno programiranje**.

2 PLANIRANJE RAZVOJA APLIKACIJE

U ovom će se poglavlju opisati interakcija korisnika sa sustavom za upravljanje redovima čekanja putem aplikacije, utvrdit će se nužne mogućnosti koje aplikacija mora imati i stvorit će se prototip sučelja po kojem se može izraditi mobilna aplikacija. Iz toga će se prepoznati nužne podatke koje treba spremiti u bazu podataka.

2.1 Interakcija i nužne mogućnosti aplikacije

Osnovna ideja aplikacije je mogućnost korisnika da pomoću nje može rezervirati mjesto u redu bez da nužno fizički bude prisutan u ustanovi tijekom čekanja. Stoga aplikacija mora imati **popis ustanova** i **redova** unutar svake od njih te nekoliko osnovnih **detalja o ustanovama** – adresu, telefon i adresu e-pošte. Odabirom reda treba vidjeti njegove detalje – trenutno **stanje** i mogućnost **rezerviranja** mjesta u tom redu uzimanjem **broja**. Korisnik mora moći vidjeti popis svojih **rezervacija** kako bi u svakom trenutku znao kad se bliži njegov red. Kako ne bi morao stalno provjeravati unutar aplikacije, poželjno je uvesti mogućnost primanja zvučne **obavijesti** prilikom promjene stanja u redu. Korisnik također treba moći **otkazati** rezervaciju. To su nužne mogućnosti mobilne aplikacije.

Kako bi mobilna aplikacija funkcionirala i to istovremeno za veći broj korisnika, nužno je stvoriti i pozadinski sustav (engl. *backend)* na poslužitelju koji će pružati navedene mogućnosti – mobilna aplikacija će biti sučelje pomoću kojeg korisnik komunicira s poslužiteljem. Time se šire nužne mogućnosti aplikacije i sustava – kako postoji više korisnika, mora postojati način identifikacije i evidencije korisnika. Stoga u nužne mogućnosti treba dodati **registraciju** novih korisnika i **prijavu** postojećih. Podaci o korisnicima, ustanovama i redovima moraju biti trajno pohranjeni korištenjem baze podataka.

Za upravljanje redom čekanja potrebno je izraditi dodatno sučelje za administratore, najbolje u obliku Web aplikacije koja također komunicira s poslužiteljem. Pomoću tog sučelja treba biti moguće vidjeti pojedinosti korisnika (klijenata) u svakom redu čekanja, označiti završetak rada s trenutnim klijentom i tako omogućiti sljedećem korisniku dolazak na red te mogućnost zatvaranja odabranog reda (primjerice na kraju radnog vremena). Također treba omogućiti administratorima **dodavanje** i **uređivanje** ustanova i redova te njihovo eventualno **brisanje**. Budući da su to osjetljive radnje, trebalo bi uvesti zaštitu prilikom pokušaja njihovog izvršavanja

tako da se ograniče samo na autorizirane korisnike – one koje imaju **administratorska prava**. Idealno bi bilo odrediti odgovorne osobe za svaku ustanovu i red i ograničiti takve radnje samo na njih, međutim to bi dodatno zakompliciralo sustav kojem je cilj biti dokaz koncepta – mogućnosti za poboljšanje sustava opisat će se kasnije u radu.

2.2 Sučelje aplikacije

Na temelju ranije navedenih nužnih mogućnosti potrebno je izraditi prototip sučelja mobilne aplikacije. Izrada sučelja u Android Studio-u može započeti i bez prototipa, ali ovim se postupkom to znatno olakšava i manje je vjerojatno da će se neki element zaboraviti dodati ako se istovremeno ne mora voditi računa o imenovanju i programskoj logici. Prototip sučelja izrađen je uz pomoć besplatne inačice Web alata NinjaMock[9].

Zaslon za prijavu korisnika trebao bi biti vrlo jednostavan – omogućiti korisniku da upiše korisničko ime i lozinku te gumb koji provjerava unesene podatke. Također treba sadržavati poveznicu na zaslon registracije. Navigacija unutar aplikacije trebala bi biti stalno na istom mjestu i sadržavati poveznice na glavne zaslone aplikacije – popis ustanova, popis rezervacija te postavke korisnika (profil).

Slika 4: Prototip sučelja zaslona prijave.
Izvor: autor

[9] Dostupno na https://ninjamock.com/

Zaslon registracije treba sadržavati polja za unos svih korisnikovih podataka – ime, prezime, adresu e-pošte te polja za unos i potvrdu lozinke. Kao i na zaslonu prijave, dodirom gumba trebaju se prenijeti uneseni podaci na poslužitelj gdje se provjeravaju i obrađuju. Dobro je i da sadrži poveznicu za povratak na zaslon prijave.

Slika 5: Prototip sučelja zaslona registracije.
Izvor: autor

Korisnik treba, neovisno je li prijavljen u aplikaciju, moći vidjeti popis ustanova, detalje o njima, redove unutar njih i stanje u redovima koji ga zanimaju. Na popisu ustanova bi trebala stajati poveznica na zaslon informacija o ustanovi.

Slika 6: Prototip sučelja zaslona popisa ustanova.
Izvor: autor

Zaslon detalja o ustanovi treba sadržavati osnovne podatke o odabranoj ustanovi – ime, adresu, telefonski broj, adresu e-pošte i kartu lokacije ustanove. Također, sa zaslona detalja treba moći navigirati do zaslona popisa redova odabrane ustanove. Poželjno bi bilo omogućiti korisniku zvanje ustanove i slanje e-pošte dodirom na željeni detalj.

Slika 7: Prototip sučelja zaslona detalja ustanove.
Izvor: autor

Odabirom ustanove treba se prikazati zaslon s popisom svih njenih redova, a kraj svakog od njih korisnik bi trebao vidjeti broj kojeg se u ovom trenutku poslužuje.

Slika 8: Prototip sučelja zaslona popisa redova.
Izvor: autor

Odabirom reda korisnik treba vidjeti naziv reda i ustanove, broj koji se trenutno poslužuje i mogućnost rezerviranja mjesta u redu s prikazom očekivanog broja.

Slika 9: Prototip sučelja zaslona detalja reda.
Izvor: autor

Rezerviranjem mjesta u redu korisniku treba prikazati zaslon popisa rezervacija. Na njemu trebaju biti detalji svih korisnikovih rezervacija – ustanova, red, korisnikov broj u redu i trenutni broj koji se poslužuje. Poželjno bi bilo korisniku pružiti predviđeno vrijeme čekanja te istaknuti kad je upravo on osoba koja je upravo na redu. Također je potrebno pružiti korisniku mogućnost otkazivanja rezervacije na ovom zaslonu.

Slika 10: Prototip sučelja zaslona popisa rezervacija.
Izvor: autor

Zaslon postavki profila korisnika treba omogućiti korisniku izmjenu svojih podataka te postavki o željenim načinima obavještavanja – želi li čuti zvučni signal prilikom promjene u redovima gdje ima rezervaciju te želi li primiti tzv. *push*[10] obavijest kad je došao njegov broj na red. Osim gumba za spremanje izmjena, trebao bi sadržavati gumb za odjavu korisnika iz aplikacije. Tim postupkom može omogućiti drugom korisniku na istom uređaju prijavu u sustav, ali svaki korisnik treba imati svoje podatke koji su pohranjeni na poslužitelju (u bazi podataka).

Slika 11: Prototip sučelja zaslona postavki profila.
Izvor: autor

2.3 Shema baze podataka

Po ranijim opisima aplikacije i interakcije može se predvidjeti potrebni podaci koji će se spremati u bazu podataka. U ovom trenutku nije bitno koja će se točno baza podataka koristiti, samo je bitno definirati neophodne podatke. Svaka tablica/dokument treba imati **ID** koji će biti primarni ključ, pa se to više neće isticati. Također, u implementaciji će se koristiti engleski nazivi za tablice/dokumente i polja i moguće je da će se razlikovati od ove sheme.

- **Korisnik** – ime, prezime, e-mail, lozinka, razina_prava
- **Ustanova** – naziv, adresa, e-mail, telefon

[10] poruka koju poslužitelj šalje uređaju u pravom trenutku, bez potrebe za stalnim slanjem upita s uređaja na poslužitelj čime se omogućuje visoka razina interaktivnosti i trenutna obaviještenost korisnika

- **Red** – ime, id_ustanove, trenutni_broj
- **Rezervacija** – id_korisnika, id_reda, broj, vrijeme_rezervacije

Iz gornje sheme mogu se prepoznati veze (pomoću vanjskih i primarnih ključeva) – jedna ustanova sadrži više redova, dok rezervacija služi za razrješavanje veze **više na više** između reda i korisnika.

3 POZADINSKI SUSTAV I ADMINISTRACIJSKO SUČELJE

Na temelju prethodnog istraživanja i planiranja razvoja, izrađen je (relativno) jednostavan sustav za upravljanje redovima čekanja koji se sastoji od baze podataka, pozadinskog sustava, administracijskog sučelja i mobilne aplikacije. U ovom i sljedećem poglavlju će se navesti korištene tehnologije, usluge i alati te, nešto detaljnije, postupci i metode izrade ovog sustava.

3.1 Tehnologije i programski alati

3.1.1 Baza podataka

Za bazu podataka odabrana je **MongoDB** dokumentna baza podataka. Podaci se, za razliku od relacijskih baza podataka, spremaju u fleksibilne dokumente koji su vrlo nalik JSON[11] objektima. Besplatna je i otvorenog koda, nije ograničena shemom, upiti su jednostavni i brzi te ima podršku među svim popularnijim programskim jezicima. Među ostalima, koriste je tvrtke Adobe, Amazon, Bosch, Cisco, eBay i VMWare. (MongoDB, 2018)

Kako bi baza podataka bila dostupna u svako vrijeme i na svakom mjestu, potrebno ju je prenijeti na udaljeni poslužitelj. To se obavilo na samom početku razvoja koristeći jednu od besplatnih opcija **mLab**[12] usluge dijeljenja baze podataka u oblaku (engl. d*atabase-as-a-service)* za MongoDB bazu. Na taj način razvoj i testiranje sustava nisu ni u jednom trenutku bili ograničeni na jedan uređaj ni stolno računalo, što je dobar početak za sustav koji predviđa postojanje mnogo različitih korisnika i uređaja istodobno.

3.1.2 Pozadinski sustav

Za izradu pozadinskog sustava rabljen je **JavaScript** programski jezik uz pomoć već spomenutog **Node.js** sustava za izvođenje. Budući da se na sustav treba povezivati i s mobilne i s Web aplikacije, stvoreno je **REST**[13] programsko sučelje korištenjem **Express.js**[14] JavaScript programskog okvira, što će detaljnije biti opisano

[11] JavaScript Object Notation – standardiziran način zapisivanja podataka u obliku ključ – vrijednost koji, zbog svoje široke rasprostranjenosti, često služi kao međukorak za prijenos podataka između sustava koji su pisani različitim programskim jezicima
[12] dostupno na https://mlab.com/
[13] Representational State Transfer
[14] dostupno na https://expressjs.com/

kasnije u radu. Za olakšano spajanje na MongoDB bazu korišten je **Mongoose.js**[15] programski okvir za objektno modeliranje podataka – postupak u kojem se koriste elementi objektno orijentiranog programiranja (konkretno klase) za stvaranje sheme baze podataka uz olakšano čitanje, pisanje i validaciju. Kako se lozinke korisnika ne bi pohranjivale nezaštićene, korištena je biblioteka **bcrypt.js**[16]. Kako zbog prirode REST programskog sučelja nije moguće spremati stanje o prijavi korisnika na poslužitelju, korišten je okvir za autorizaciju **passport.js**[17] uz pomoć strategije **JWT**[18]. Na taj način prilikom prijave u sustav klijentska aplikacija dobije autorizacijski token (koji je zapravo dugački niz alfanumeričkih znakova šifriran određenim ključem i kodiran korištenjem Base64[19] metode) i mora se poslati prilikom poziva zaštićenim dijelovima pozadinskog sučelja kako bi se izvršila autentikacija i autorizacija. Uz nekoliko pomoćnih i, za ovaj projekt, manje bitnih biblioteka, korištena je i **Firebase**[20] platforma. Platforma Firebase tvrtke Google ima mnoge mogućnosti (uključujući pohranu, autentikaciju, baze podataka i analitiku) koje uvelike olakšavaju izradu pozadinskih sustava, ali za ovaj sustav korišten je samo **Firebase Cloud Messaging** za slanje push obavijesti.

Zbog istih razloga navedenih za bazu podataka – i pozadinski sustav bi trebao biti dostupan u svako vrijeme sa svakog mjesta. Za tu zadaću korištena je usluga platforme u oblaku (engl. *platform-as-a-service*) **Heroku**[21]. Besplatni model ima neka ograničenja, od kojih se najčešće susreće neaktivnost aplikacije ukoliko joj se ne pristupi trideset minuta te se ona mora ponovno pokrenuti (što je stvaralo probleme tijekom izrade mobilne aplikacije). Razlog tome je ograničena količina vremena tijekom kojeg besplatne aplikacije mogu biti pokrenute, a to je petsto minuta u jednom mjesecu. Na taj način Heroku nastoji uštediti korištenje vremena tijekom neaktivnosti. Prijenos na udaljeni poslužitelj vršio se automatski uz pomoć **Git** sustava za verzioniranje koda spojenog na mrežnu uslugu **GitHub**[22].

[15] dostupno na http://mongoosejs.com/
[16] dostupno na https://www.npmjs.com/package/bcryptjs
[17] dostupno na http://www.passportjs.org/
[18] JavaScript Web Token
[19] sustav za pretvorbu raznih podataka u tekstualni oblik prevođenjem u prikaz kojem je baza broj 64
[20] dostupno na https://firebase.google.com/
[21] dostupno na https://www.heroku.com/
[22] dostupno na https://github.com/

3.1.3 Administracijsko sučelje

Za izradu administracijskog sučelja – koje je u suštini Web aplikacija – korištena je (očekivana) kombinacija tehnologija **HTML5**, **CSS3** i **JavaScript**.

Za izgled sučelja korišten je responzivni okvir za izradu sučelja **Materialize**[23]. Time se postiže da izgled aplikacije nalikuje na Android sučelje te odgovarajući razmještaj na različitim uređajima, neovisno jesu li mobilni uređaji ili stolna računala.

Aplikacijska logika i općenito ponašanje aplikacije pod kontrolom je JavaScript programskog okvira za sučelja **Vue.js 2**[24]. Olakšava izradu Web aplikacija na moderan i modularan način korištenjem najnovijih tehnologija i sposobnost pokretanja cijele Web aplikacije na jednoj Web stranici. Alternativa je ranije spomenutim tehnologijama za izradu sučelja (React i Angular).

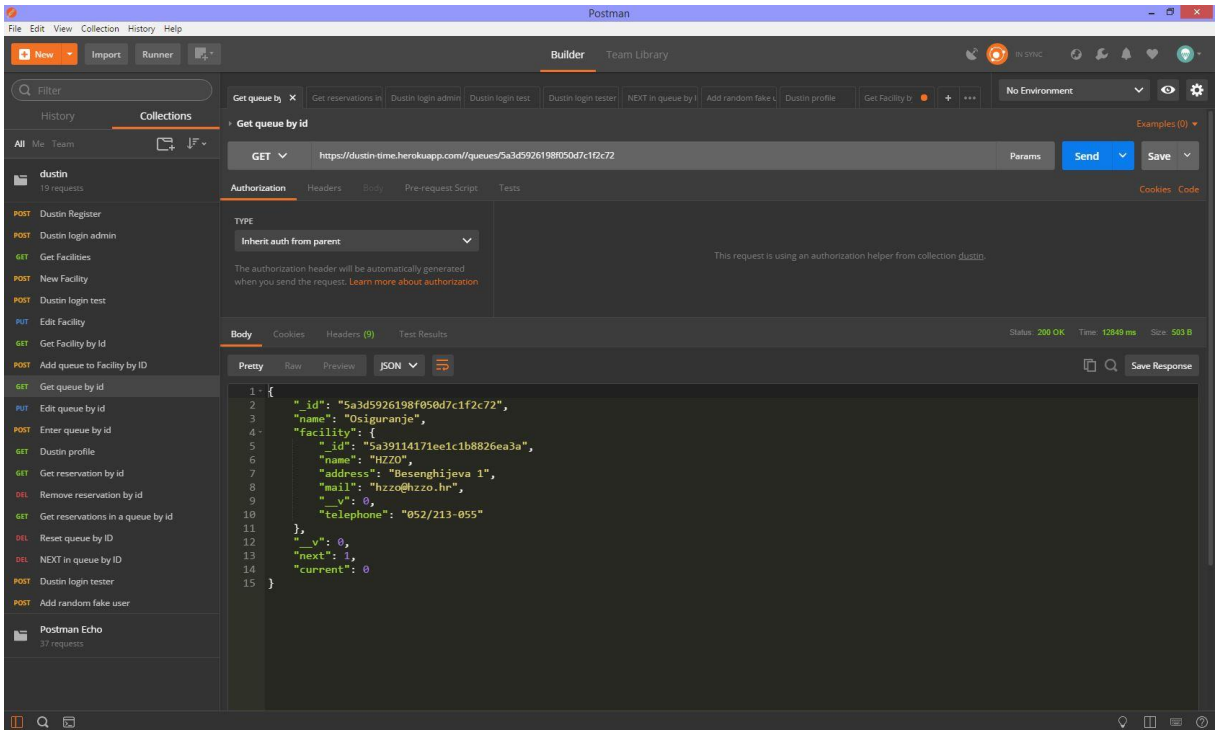
Za slanje REST poziva korišten je HTTP[25] klijent **Axios**[26]. Jednostavan je za korištenje na klijentu i na poslužitelju i podržava mnoge moderne mogućnosti i načine programiranja.

3.1.4 Programski alati

Korišteni programski alati:

- **WebStorm**[27] razvojno okruženje za JavaScript tvrtke JetBrains koje ima vrlo slično sučelje alatu Android Studio i besplatno je za korištenje u obrazovne svrhe, korišteno za pozadinski sustav i administracijsko sučelje
- **Visual Studio Code**[28] uređivač koda tvrtke Microsoft, idealan za manje izmjene, korišten pretežno za administracijsko sučelje
- **Postman**[29] sustav alata za olakšanu izradu, testiranje i otklanjanje grešaka REST programskih sučelja, gotovo nezamjenjiv prije izrade grafičkog korisničkog sučelja

[23] dostupno na http://materializecss.com/
[24] dostupno na https://vuejs.org/
[25] Hypertext Transfer Protocol
[26] dostupno na https://www.npmjs.com/package/axios
[27] dostupno na https://www.jetbrains.com/webstorm/
[28] dostupno na https://code.visualstudio.com/
[29] dostupno na https://www.getpostman.com/



Slika 12: Testiranje REST sučelja pomoću programa Postman.
Izvor: autor

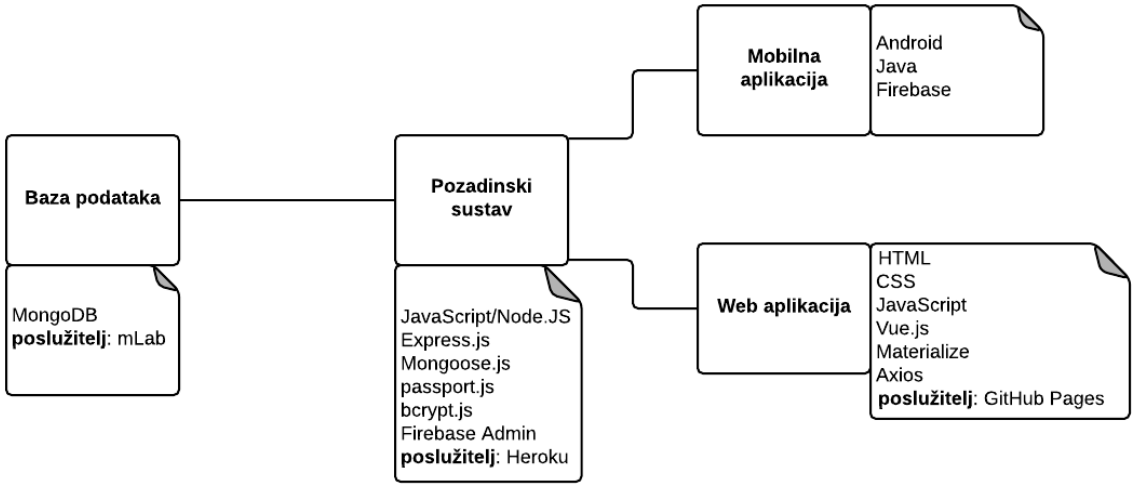
3.1.5 Arhitektura sustava

Već je navedeno kako se ovaj sustav za upravljanje redovima čekanja sastoji od nekoliko glavnih komponenti – mobilne Android aplikacije koja služi kao klijent za korisnike sustava, Web aplikacije koja je administracijsko sučelje za upravljanje ustanovama i redovima, baze podataka koja trajno pohranjuje podatke o korisnicima, redovima, ustanovama i rezervacijama te pozadinskog sustava koji povezuje bazu sa sučeljima mobilne i Web aplikacije.

Osim mobilne aplikacije koja se izvodi na mobilnom uređaju korisnika, ostale komponente sustava moraju biti dostupne u svakom trenutku – što znači da se moraju izvoditi na udaljenim poslužiteljima kako bi mogle međusobno komunicirati. Za tu svrhu korištene su besplatne inačice mrežnih usluga, neke od kojih su već navedene ranije u radu tijekom navođenja i opisa tehnologija.

Središte sustava za upravljanje redovima čekanja je upravo pozadinski sustav koji se izvodi na platformi Heroku. Taj sustav pruža univerzalno sučelje za međusobnu komunikaciju ostalih komponenti sustava. Baza podataka poslužuje se s platforme mLab. Web aplikacija se poslužuje pomoću usluge **GitHub Pages**, koji omogućava besplatno posluživanje Web sadržaja u sklopu GitHub mrežne usluge za verzioniranje koda. Na taj je način izbjegnuta potreba za dodatnim pružateljem usluga udaljenog

poslužitelja jer je GitHub usluga već korištena za pohranu i verzioniranje – ne samo Web aplikacije, već svih komponenti sustava. Ipak, zbog specifičnog načina izrade Vue.js Web aplikacije, bilo je potrebno izvršiti određene prilagodbe, budući da GitHub Pages omogućava posluživanje samo Web stranica koje se u potpunosti mogu izvoditi u Web pregledniku, odnosno posebne mape koja sadrži programsku dokumentaciju. Upravo je ta činjenica iskorištena za izvođenje Web aplikacije – prije prijenosa izmjena na GitHub, Vue.js aplikacija se prevodi u oblik koji se može izvoditi u Web pregledniku i rezultat se prebacuje u mapu predviđenu za dokumentaciju.



Slika 13: Prikaz komponenti sustava i korištenih tehnologija.
Izvor: autor

3.2 REST sučelje

Zadaća pozadinskog sustava je povezati bazu podataka u kojoj su spremljeni svi podaci sustava s klijentskim aplikacijama koje trebaju čitati i upravljati tim podacima uz poštovanje aplikacijske logike i ograničenja sustava. U većini slučajeva kad je potrebno pristupiti podacima koristeći različite klijentske tehnologije (primjerice mobilnu i Web aplikaciju), uobičajena je praksa stvoriti REST programsko sučelje. To je način pristupanja mrežnim resursima korištenjem HTTP protokola i odgovarajućih **ruta** koje označavaju željeni resurs. Na taj je način moguće čitati, pisati, uređivati i brisati[30] sadržaje korištenjem **metoda** (koje se zbog naziva nekad nazivaju i glagolima) – GET, POST, PUT i DELETE (iako ima i drugih, ovo su najkorištenije). Poslužitelj najčešće odgovara podacima u JSON obliku i korištenjem unaprijed određenih brojčanih kodova

[30] engl. CRUD – *create, read, update, delete*

o uspjehu upita po kojima klijentska aplikacija može utvrditi je li sve prošlo očekivano ili je došlo do pogreške (i kakve). Izrada REST sučelja nije standardizirana – kao i mnoge stvari u programiranju, moguće je isti učinak postići na različite načine. (Devslopes, 2017)

Izrada REST sučelja korištenjem Express.js programskog okvira je poprilično izravna, u najjednostavnijem slučaju potrebno je definirati metodu i rutu te programski kod koji se izvršava u tom slučaju. Dobra je praksa odvojiti rute u posebne datoteke – po jedna datoteka za svaku osnovnu rutu. Rute najčešće odgovaraju resursima kojima se želi pristupiti, u slučaju sustava za upravljanje redovima čekanja to će biti korisnici, ustanove, redovi i rezervacije – prateći najbolje prakse, na engleskom jeziku. Zbog kompatibilnosti s drugim sustavima, za razmjenu podataka koristi se JSON oblik zapisa podataka, dok neke rute mogu čitati parametre iz URL[31]-a, što se označava znakom dvotočja prije imena parametra u ruti.

3.2.1 Korisnici

Rute koje upravljaju korisnicima započinju s ***/users***. Omogućuju registraciju, prijavu, provjeru i izmjenu korisničkog profila.

Registracija korisnika obavlja se slanjem **POST** upita na /**register** rutu s priloženim podacima o korisniku (ime, prezime, e-mail i lozinka). Potom se provjerava postoji li već korisnik s istim e-mailom u sustavu, u tom se slučaju prekida registracija i javlja greška s kodom 409 koja označava konflikt. Ukoliko e-mail nije zauzet, nastavlja se proces registracije korisnika zapisom podataka u bazu, dok se lozinku šifrira prije spremanja koristeći bcrypt. Obavlja se i pomoćna validacija podataka na bazi korištenjem Mongoose okvira – provjerava se prisutnost svih podataka, uklanja se mogući višak razmaka te valjanost adrese e-pošte. Prilikom zapisa automatski se postavlja uloga korisnika (samo je jedan administrator sustava).

[31] Uniform Resource Locator

```
// User Schema
const UserSchema = mongoose.Schema({
    firstName: {
        type: String,
        required: true,
        trim: true
    },
    lastName: {
        type: String,
        required: true,
        trim: true
    },
    mail: {
        type: String,
        required: true,
        alias: 'username',
        trim: true,
        validate: {
            validator: validator.isEmail,
            message: '{VALUE} is not a valid email',
            isAsync: false
        },

    },
    password: {
        type: String,
        required: true,
    },
    role: {
        type: String,
        default: 'user',
    },
});
```

Slika 14: Mongoose shema resursa korisnici.
Izvor: autor

Prijava korisnika također se obavlja slanjem **POST** upita, međutim na **/login** rutu s korisničkim imenom (e-mail) i lozinkom u tijelu upita. Provjerava se postojanje korisnika te se, ukoliko je korisnik pronađen, uspoređuje priloženu lozinku (koju se prethodno šifrira na isti način kao kod registracije) s onom spremljenom u bazi. Ako se lozinke podudaraju, vraća se odgovor s podacima korisnika, popisom njegovih rezervacija i JWT tokenom koji se koristi prilikom pristupanja zaštićenim rutama unutar sustava, a sadrži podatke o korisniku koji tada služe za identifikaciju. Upravo je token najvažniji dio prijave jer se, zbog prirode REST sučelja, podaci o prijavljenim korisnicima ne spremaju na poslužitelju, već je potrebno pohraniti token na strani klijenta i priložiti ga uz upite zaštićenim rutama.

Dvije zaštićene rute vezane za korisnike omogućuju pregled te izmjenu podataka o korisniku slanjem upita na **/me** rutu. Pregled podataka i svih rezervacija korisnika obavlja se **GET** upitom, dok se za izmjenu profila šalje **PUT** upit zajedno s podacima koji se žele izmijeniti. Kao što je već spomenuto, za pristup ovim rutama potrebno je upitu priložiti autorizacijsko zaglavlje s tokenom pomoću kojeg sustav identificira

prijavljenog korisnika prilikom čitanja i mijenjanja podataka. Passport okvir provjerava valjanost tokena, a ukoliko je došlo do promjene adrese e-pošte, prije spremanja provjerava se zauzetost kako ne bi došlo do konflikta.

3.2.2 Ustanove

Rute koje upravljaju ustanovama započinju s **/facilities**. Omogućuju dodavanje, čitanje i izmjenu podataka o ustanovama te dodavanje novog reda odabranoj ustanovi.

```
// Facility Schema
const FacilitySchema = mongoose.Schema({
    name: {
        type: String,
        required: true,
        trim: true
    },
    address: {
        type: String,
        required: true,
        trim: true
    },
    mail: {
        type: String,
        required: true,
        trim: true,
        validate: {
            validator: validator.isEmail,
            message: '{VALUE} is not a valid email',
            isAsync: false
        }
    },
    telephone: {
        type: String,
        required: true,
        trim: true
    }
});
```

Slika 15: Mongoose shema resursa ustanove.
Izvor: autor

Dohvaćanje popisa svih ustanova obavlja se slanjem **GET** upita na **/** (korijensku) rutu ustanova. Tako se dobiju podaci o svim ustanovama (naziv, adresa, telefon, e-mail), ali ne i popis redova unutar njih.

Dodavanje ustanove vrši se slanjem **POST** upita na korijensku rutu s priloženim podacima o ustanovi i tokenom prijavljenog administratora.

Za detaljnije informacije o jednoj ustanovi potrebno je priložiti **id** ustanove – slanjem **GET** upita na **/:id** rutu. Tada se dobiju podaci o odabranoj ustanovi i popis redova koji joj pripadaju (uključujući detalje o svakom redu). Slanjem **PUT** upita na istu rutu

moguće je izmijeniti podatke o ustanovi, ali samo ako je u zaglavlju priložen token prijavljenog administratora.

Slanjem **POST** upita na ***/*:id** rutu dodaje se novi red odabranoj ustanovi. Potrebno je priložiti ime reda i administracijski token, a red se dodaje u bazu povezan za ustanovu koristeći njen id pročitan iz rute.

3.2.3 Redovi

Rute koje upravljaju redovima započinju s ***/*queues**. Nude mogućnosti čitanja i uređivanja podataka određenog reda, resetiranja reda na zadane postavke te, najvažnije, slanje signala za prelazak na sljedećeg korisnika u tom redu. Kako su ovo sigurnosno **osjetljive rute**, sve osim čitanja su **zaštićene** i potrebno je priložiti autorizacijsko zaglavlje s tokenom prijavljenog administratora sustava kako bi funkcionirale. U odnosu na prototip, shema redova sadrži još jedno polje – sljedeći broj koji se treba izdati korisniku prilikom rezervacije. Zadana vrijednost trenutnog broja je nula, dok je prvi sljedeći broj jedan.

```
// Queue Schema
const QueueSchema = mongoose.Schema({
    name: {
        type: String,
        required: true,
        trim: true
    },
    facility: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Facility',
        required: true
    },
    current: {
        type: Number,
        default: 0,
        min: 0
    },
    next: {
        type: Number,
        default: 1,
        min: 1
    }

});
```

Slika 16: Mongoose shema resursa redovi.
Izvor: autor

Čitanje i uređivanje određenog reda obavljaju se slanjem upita na istu rutu - **/:id**. **GET** upit vraća podatke o redu i ustanovi kojoj pripada. **PUT** upit omogućuje izmjenu podataka o redu.

Vraćanje reda na zadane postavke briše sve rezervacije unutar odabranog reda i mijenja vrijednosti trenutnog broja na nulu, a sljedećeg na jedan. To se obavlja slanjem **DELETE** upita na rutu **/:id/reset**.

Mogućnost prelaska na sljedećeg korisnika najkompleksniji je dio cijelog sustava i teško ju je objasniti bez da se prethodno objasni proces uzimanja broja (stvaranja rezervacije) – koji je sam po sebi također poprilično kompleksan pa će se te dvije mogućnosti pojasniti u zasebnim potpoglavljima.

3.2.4 Rezervacije

Rute koje upravljaju rezervacijama započinju s **/reservations**. Omogućuju čitanje jedne određene rezervacije, svih rezervacija u određenom redu, stvaranje nove rezervacije te njeno brisanje.

```
// Reservation Schema
const ReservationSchema = mongoose.Schema({
    user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        required: true
    },
    queue: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Queue',
        required: true
    },
    time: {
        type: Date,
        default: Date.now
    },
    number: {
        type: Number,
        required: true,
        min: 1
    }
});
```

Slika 17: Mongoose shema resursa rezervacije.
Izvor: autor

Čitanje jedne rezervacije obavlja se slanjem **GET** upita na **/:id** rutu. Time se

dohvaćaju svi povezani podaci – korisnik (vlasnik rezervacije), red i ustanova u kojoj se nalazi rezervacija, vrijeme stvaranja rezervacije te redni broj korisnika.

Moguće je pročitati sve rezervacije u određenom redu slanjem **GET** upita na **/queue/:id** rutu, gdje id predstavlja id željenog reda. Tako se dobije popis rezervacija s popunjenim detaljima o korisniku.

Otkazivanje rezervacije obavlja se slanjem **DELETE** upita na **/:id** rutu. Pritom je nužno priložiti autorizacijski token čime se ograničava mogućnost otkazivanja na vlasnika rezervacije i administratora sustava.

3.2.5 Stvaranje rezervacije (uzimanje broja)

Temeljna značajka sustava za upravljanje redovima čekanja je mogućnost korisnika da rezerviraju mjesto u redu, drugim riječima da uzmu (virtualni) broj pomoću kojeg mogu pretpostaviti kad su na redu. Svaki red sprema u bazu podataka dva broja – trenutni broj koji se poslužuje i broj koji će se izdati sljedećem korisniku prilikom stvaranja rezervacije.

Uzimanje broja obavlja se slanjem **POST** upita na rutu **/reservations/:id**, gdje :id parametar označava id reda u kojem korisnik želi stvoriti rezervaciju, uz priložen autorizacijski token. Pomoću kombinacije podataka o korisniku i id-a reda prvo se vrši provjera je li korisnik već u tom redu, u tom se slučaju radnja prekida uz slanje poruke o konfliktu.

Čitanjem podataka o odabranom redu iz baze priprema se nova rezervacija – broj korisnika u redu postavlja se na vrijednost sljedećeg broja zapisanog u bazi podataka. Rezervacija se također puni id-om korisnika i reda te time postaje potpuna (vrijeme rezervacije se automatski stvara prilikom zapisivanja u bazu podataka).

Ključna radnja obavlja se istovremeno sa stvaranjem rezervacije – potrebno je povećati spremljeni sljedeći broj unutar reda za jedan, kako bi sljedeće uzimanje broja proteklo uspješno.

Odgovor sustava sadrži sve podatke o rezervaciji – id korisnika, podatke reda i ustanove kojoj pripada te korisnikov broj kojeg je upravo rezervirao. U slučaju greške, na upit se odgovara kodom greške 500 (unutarnja greška sustava) te svim dostupnim podacima o greški.

3.2.6 Prelazak na sljedećeg korisnika u redu

Kako bi sustav za upravljanje redovima čekanja mogao pravilno funkcionirati, mora postojati način za obavještavanje korisnika o promjeni trenutnog broja. Korisnici moraju u svakom trenutku moći provjeriti koji je trenutni broj i usporediti ga sa svojim. Međutim, takav način već postoji s fizičkim uzimanjem brojeva, iako je za nijansu lakše uzeti virtualni broj i provjeravati stanje preko mobilnog uređaja u odnosu na obavezno fizičko prisustvo u ustanovi gdje se papirnati listić s brojem uspoređuje s digitalnim brojem na panou. Stoga je zadaća opcije prelaska na sljedećeg korisnika dvojaka – izmijeniti trenutni broj u zapisu reda te poslati push obavijest korisnicima u tom redu korištenjem Firebase platforme.

Kako bi prešao na sljedećeg korisnika u redu, prijavljeni administrator sustava (prilaganjem autorizacijskog tokena) treba poslati **DELETE** upit na rutu **/queues/:id/next**, gdje :id parametar predstavlja id reda kojim se upravlja. U tom se trenutku iz baze čitaju podaci o tom redu i programska se logika grana ovisno o trenutnom stanju reda.

Slanje push obavijesti i poruka moguće je na više načina – može se poslati obavijest ciljano (jednom korisniku ili uređaju), određenom podskupu svih korisnika ili na zahtjev korisnicima koji su pretplaćeni na određenu temu. Upravo zadnja opcija se koristi u ovom sustavu – prilikom rezerviranja mjesta u redu, korisnici se pretplaćuju na „temu“ kojoj je šifra id reda. Na taj način može se istovremeno poslati obavijest o promjeni stanja u redu svim korisnicima u njemu, bez potrebe za dodatnim zapisivanjem podataka o uređaju s kojim je korisnik prijavljen u sustav. Iz tog se razloga prije utvrđivanja stanja u redu prvo priprema slanje grupne poruke kojoj je tema id tog reda, dok sadržaj poruke ovisi o stanju reda.

Utvrđivanje stanja reda započinje uspoređivanjem vrijednosti trenutnog i sljedećeg broja u redu. Ukoliko je trenutni broj nula i sljedeći jedan, smatra se da je **red prazan** i da nije moguće prijeći na sljedećeg korisnika pa se javlja poruka o greški.

U slučaju da je trenutni broj nula – bez da je istovremeno sljedeći jedan – smatra se da je **red zatvoren** te ga ova operacija otvara. Iz baze podataka se učitava prva rezervacija u tom redu te se trenutni broj u redu postavlja na vrijednost broja učitane rezervacije. Potom se stvara sadržaj push poruke – id reda i njegov trenutni broj, a uspješnim slanjem te poruke završava se prelazak na sljedećeg korisnika – čija se

rezervacija vraća kao odgovor na upit. Ako se tijekom operacije dogodi greška – vraća se odgovor o neuspješnom prelasku na sljedećeg korisnika i pratećim razlogom (što vrijedi za sve slučajeve).

Ako se utvrdi da ni trenutni ni sljedeći broj nisu na zadanim vrijednostima, smatra se da je **red otvoren** i funkcionalan – uspješno se već prelazilo na sljedećeg korisnika u redu. Tada je potrebno pronaći sve rezervacije koje pripadaju tom redu te provjeriti njihov broj – postoji li samo jedna ili ih je više. Za razliku od prethodna dva slučaja, u otvorenom redu može se javiti nekoliko alternativnih situacija i sustav treba reagirati različito na svaku od njih.

Najjednostavnija i vjerojatno najčešća situacija je ukoliko postoji **više od jedne rezervacije** u trenutnom redu. Tada se provjerava podudaranje trenutnog broja u redu s prvom pronađenom rezervacijom. Ukoliko su brojevi jednaki, smatra se da je prva rezervacija trenutno aktivan korisnik te da je upravo završio s radom (jer je administrator odabrao opciju prelaska na sljedećeg korisnika). Stoga se ta rezervacija briše iz baze i učitava sljedeća po redu, čiji broj postaje trenutno aktivan broj u redu čekanja. Uspješnom izmjenom trenutnog broja u redu i slanjem push poruke s id-om reda i njegovim (novim) trenutnim brojem završava prijelaz na sljedećeg korisnika.

Zbog mogućnosti korisnika da otkažu rezervaciju te ukoliko nema više od jednog korisnika u redu prilikom pokušaja prelaska na sljedećeg, moguća je pojava situacije pri kojoj **prva pronađena rezervacija u redu nije trenutno aktivna**. Taj se slučaj manifestira kad se broj prve pronađene rezervacije i trenutnog broja u redu ne poklapaju. Bez te provjere, prvi korisnik koji uđe u red bio bi preskočen prilikom pokušaja prelaska na sljedećeg korisnika. Ova se situacija rješava postavljanjem rezervacije tog korisnika kao aktivne (mijenjanjem trenutnog broja u redu na vrijednost iz rezervacije) i slanjem push poruke kao i u ostalim slučajevima, s tim da u ovom slučaju nema brisanja rezervacija iz baze podataka.

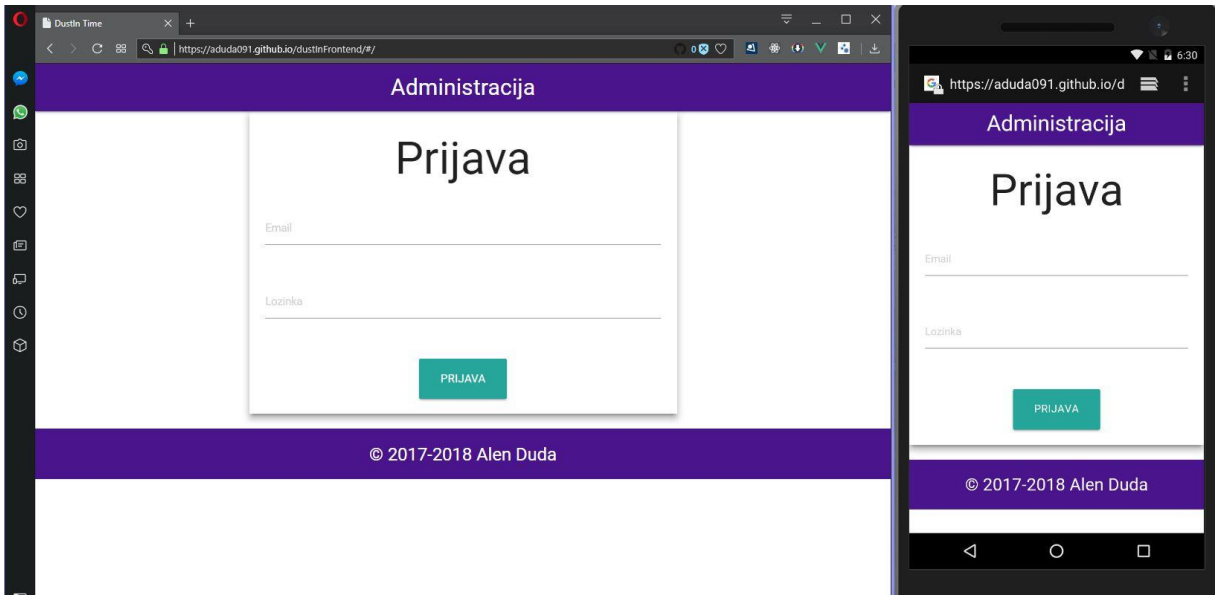
Posljednja prepoznata situacija u sustavu je postojanje **samo jednog korisnika u redu**. Do nje dolazi u slučaju kad je pronađena samo jedna rezervacija – nema drugih korisnika nakon trenutnog. Rješava se na sličan način kao i prethodno navedena situacija – postavljanjem trenutno aktivnog broja u redu na vrijednost pronađene rezervacije bez njenog brisanja iz baze podataka. Dodatna je razlika postojanje dodatne poruke u odgovoru na upit koja upozorava administratora da nakon trenutnog

korisnika trenutno nema drugih, što je moguće iskoristiti prilikom izrade grafičkog sučelja. Kako se u ovom slučaju ne briše rezervacija, ostaje na korisniku da izađe iz reda kada završi s radom (što mu aplikacija daje do znanja pomoću sučelja).

3.3 Administracijsko sučelje (Web aplikacija)

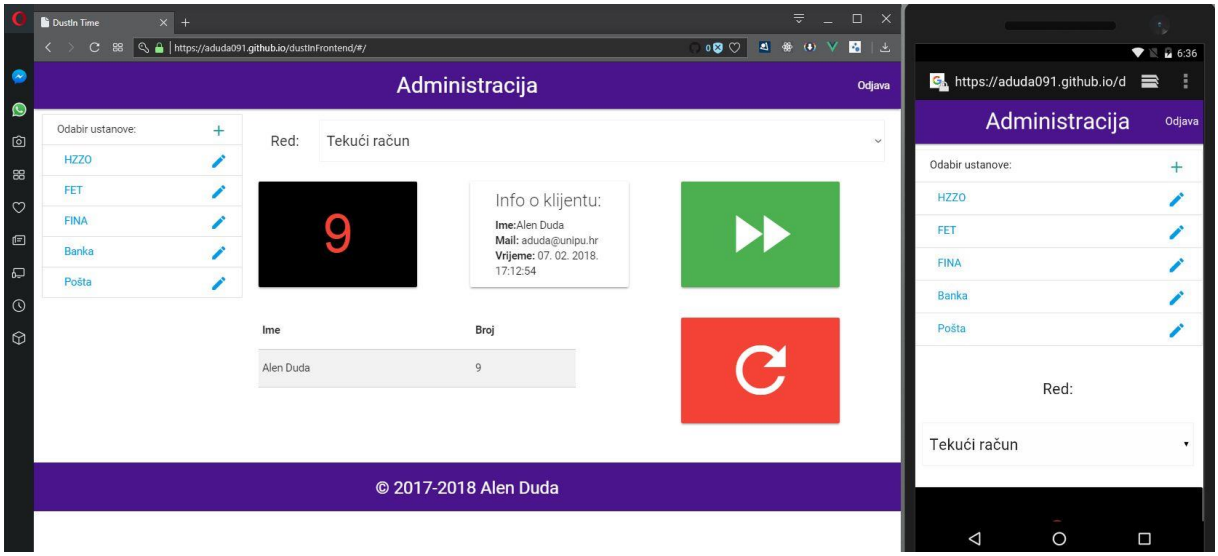
Kako bi administrator mogao upravljati sustavom, izrađeno je grafičko sučelje u obliku Web aplikacije koja se izvodi unutar jedne Web stranice (engl. *single-page application*). Omogućuje administratoru da, nakon uspješne prijave u sustav, dodaje i mijenja podatke ustanova te redova unutar njih. Također pruža pregled korisnika unutar svakog reda i daje mogućnost prelaska na sljedećeg korisnika te vraćanja reda na zadane postavke. Zahvaljujući korištenju tehnologija koje su navedene ranije u poglavlju, osiguran je ispravan prikaz Web aplikacije na stolnim i mobilnim uređajima.

Prilikom otvaranja Web aplikacije, administratora dočekuje forma za unos podataka za prijavu – adrese e-pošte i lozinke. Ti se podaci šalju pozadinskom sustavu na provjeru, gdje se utvrđuje ispravnost kombinacije te se odobrava daljnji pristup aplikaciji jedino ukoliko je u bazi pronađen korisnik koji odgovara unesenim podacima s ulogom administratora.



Slika 18: Zaslon prijave administracijskog sučelja.
Izvor: autor

Uspješnom prijavom administratorovi se podaci pohranjuju u lokalnu pohranu Web preglednika kako bi se njegov token mogao priložiti **asinkronim** HTTP zahtjevima zaštićenim dijelovima pozadinskog sustava uz pomoć **axios** klijenta. Istovremeno se administratora preusmjerava na zaslon pregleda ustanova, redova i korisnika. To je središnji dio administracijskog sučelja i sadrži popis ustanova s poveznicama za njihovo uređivanje i stvaranje nove ustanove. Odabirom ustanove prikazuje se padajući izbornik s popisom pripadajućih redova. Za odabrani red prikazan je njegov trenutni broj i popis korisnika koji u njemu čekaju. Također se prikazuju podaci o korisniku koji je upravo na redu za posluživanje. U slučaju da je odabrani red čekanja prazan, umjesto ostalih podataka prikazat će se samo odgovarajuća poruka i gumb za vraćanje reda na zadane postavke. Podaci o korisnicima osvježavaju se svakih nekoliko sekundi asinkronim slanjem odgovarajućeg zahtjeva prema pozadinskom sustavu.

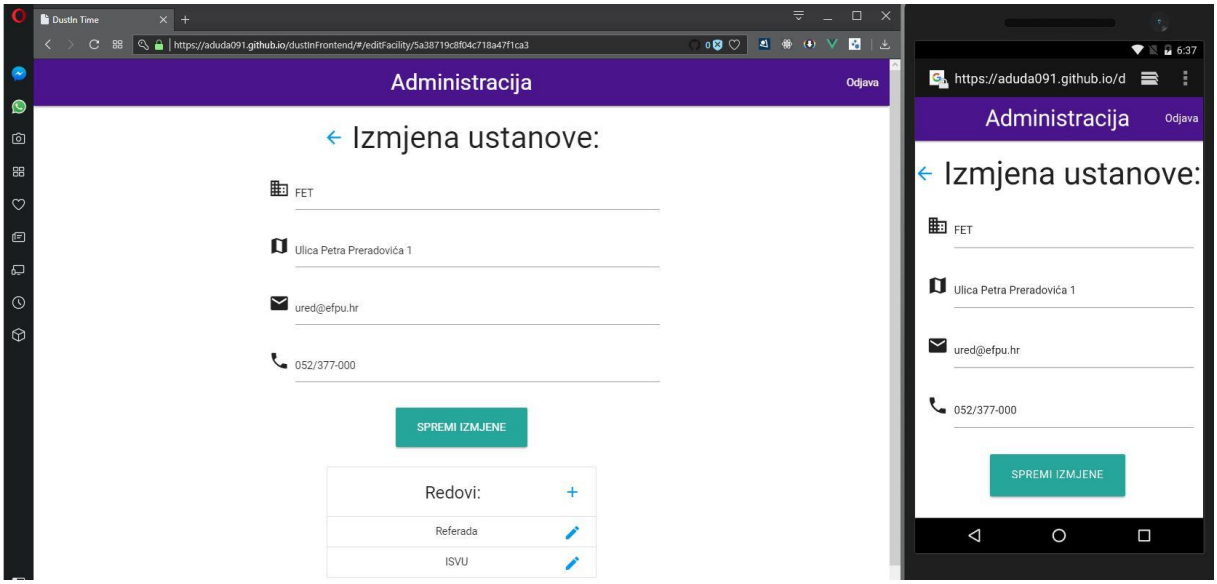


Slika 19: Zaslon pregleda administracijskog sučelja.
Izvor: autor

Administrator može odabirom zelenog gumba sa strelicama udesno poslati signal pozadinskom sustavu za **prelazak na sljedećeg korisnika**, dok odabirom crvenog gumba s kružnom strelicom može pokrenuti **vraćanje reda** na zadane postavke sa svim odgovarajućim posljedicama koje su opisane ranije u radu.

Prilikom izrade aplikacija koje rade s dodavanjem i uređivanjem podataka, često (očekivano) dolazi do poklapanja u grafičkom sučelju kod polja za unos i izmjenu podataka, budući da su u pitanju isti podaci. Tako je slučaj i ovdje – polja za unos nove i izmjenu postojeće ustanove su gotovo u potpunosti ista i sastoje se od polja za unos

naziva, adrese, e-pošte i telefonskog broja ustanove. Jedina razlika javlja se u sučelju izmjene ustanove, na dnu koje je smješten dio pomoću kojeg administrator može dodavati nove i mijenjati imena postojećih redova. Kako je jedini nužni podatak za red njegovo ime (id ustanove se čita iz URL-a kao parametar), zbog jednostavnije izrade umjesto izrade posebne HTML forme korištena je ugrađena JavaScript funkcija *prompt()* koja prikazuje jedno grafičko polje za unos u iskočnom dijalogu preglednika.



Slika 20: Zaslon mijenjanja podataka ustanove.
Izvor: autor

Nakon završetka rada, administrator može odabrati opciju odjave koja se nalazi u gornjem desnom rubu Web aplikacije. Tada se prazni lokalna pohrana koja sadržava autorizacijski token i administratora se preusmjerava natrag na zaslon prijave.

4 MOBILNA APLIKACIJA

Kako bi korisnici mogli koristiti sustav za upravljanje redovima čekanja, izrađena je mobilna Android aplikacija kao glavni način interakcije sa sustavom. Prateći službenu dokumentaciju i uz dijelove izrađene po uzoru na postojeće primjere projekata dostupne na ranije spomenutim platformama YouTube i Udemy, aplikacija se tijekom procesa izrade kontinuirano, kroz više iteracija, nadograđivala novijim tehnologijama, modernijim bibliotekama te općenito programerskim praksama i obrascima.

Izrada je počela nakon stvorenog prototipa sučelja, paralelno s razvojem pozadinskog sustava. Na taj su se način mogle utvrđivati i implementirati potrebne mogućnosti čim su bile prepoznate tijekom izrade sučelja. U međuvremenu su korišteni izmišljeni, ručno upisani (engl. *hardcoded*) podaci kako bi se prilikom izvođenja mogla dobiti bolja reprezentacija očekivanog izgleda sučelja.

4.1 Tehnologije i alati

Već u prvom poglavlju zaključeno je kako će se za mobilnu aplikaciju koristiti programski jezik **Java** i alat **Android Studio**. Zbog toga je većina korištenih tehnologija uključena u skup alata za razvoj Android aplikacija (engl. *software development kit, SDK*) koje prate Android Studio. U načelu je to Java programski okvir koji se sastoji od klasa, sučelja i drugih načela objektno orijentiranog programiranja koje omogućuju stvaranje i korištenje elemenata grafičkog sučelja te upravljanje i reagiranje na događaje koji se javljaju prilikom interakcije i izvođenja aplikacije.

4.1.1 Osnovne komponente

Sučelje Android aplikacije gradi se koristeći **aktivnosti** i **fragmente** koji, uz prateće Java i **XML**[32] datoteke, predstavljaju dio aplikacije kojeg korisnik može vidjeti, to jest zaslone (i dijelove zaslona) aplikacije. XML datoteke omogućuju tekstualno i vizualno slaganje elemenata sučelja koji se u Android programiranju nazivaju **Widgets**. Zbog zadaće upravljanja razmještajem Widgeta i njihovog korijenskog elementa, XML datoteke se ponekad nazivaju i **Layout**[33] datotekama. Postoje različiti načini razmještaja i mogu se međusobno kombinirati kako bi se nastojalo stvoriti sučelje slično prototipu. Navigacija između aktivnosti obavlja se pomoću **namjera** (engl. *intents*) koje često služe i kao prijenosnici manje količine informacija s jednog zaslona

[32] eXtensible Markup Language
[33] hrv. *razmještaj*

na drugi. Za radnje koje se trebaju izvoditi u pozadini neovisno o grafičkom sučelju koriste se **servisi**. Univerzalni naziv za svaki od navedenih (i neke koji nisu navedeni jer nisu korišteni) dijelova je komponenta. (Duda, 2015)

4.1.2 Dodatne biblioteke

Kako bi se mogle koristiti funkcionalnosti i elementi sučelja koji nisu već uključeni u Android SDK, potrebno je uključiti dodatne biblioteke koje to omogućuju. Za ovaj projekt koristile su se samo originalne Android biblioteke koje održava tvrtka Google, što osigurava dobru podršku i za starije verzije Android operacijskih sustava te, očekivano, visoku razinu dokumentiranosti.

Većina korištenih biblioteka spada u **Android Support Library** (engl. biblioteka podrške), koja je sama po sebi nekad bila skup svih uključenih biblioteka. U današnje vrijeme razvila se u cjelokupan sustav zasebnih biblioteka koje su preporučene i gotovo nužne prilikom izrade Android aplikacija. Osnovna prednost je olakšavanje programiranja za više verzija Android operacijskog sustava tako da se omogući korištenje najmodernijih tehnologija na novijim verzijama, dok za starije verzije postoje odgovarajuće alternative na koje se oslanja kako bi se zadržala podjednaka razina funkcionalnosti. Također, uključivanjem samo zasebnih biblioteka koje su potrebne uvelike se smanjuje veličina aplikacije. (Google, 2018)

Korištene Support biblioteke su:

- ***AppCompat*** – podrška za aktivnosti s akcijskom pločom i općenito **Material Design** izgledom korisničkog sučelja (koji je zastupljen u svim modernim aplikacijama i proizvodima tvrtke Google)
- ***Design*** – dodatna podrška za Material Design
- ***ConstraintLayout*** – omogućuje fleksibilni razmještaj i kontrolu veličine elemenata grafičkog sučelja te time olakšava izradu sučelja za različite veličine ekrana, korišten kao korijenski element za sve zaslone
- ***RecyclerView*** – omogućuje efikasniji prikaz veće količine podataka u obliku popisa, preporučena zamjena za manje efikasan ListView prikaz
- ***CardView*** – podrška za prikaz podataka unutar tzv. kartica, koje su jedan od osnovnih elemenata Material Design načela izrade sučelja

Kako bi se uvelike olakšalo i ubrzalo slanje asinkronih HTTP zahtjeva na REST sučelje pozadinskog sustava, korištena je **Google Volley** biblioteka. Sadrži sve

potrebno za slanje odgovarajućih zahtjeva, prilaganje autorizacijskog zaglavlja, obrađivanje dobivenih rezultata i upravljanje greškama. Izvođenje zahtjeva je na zasebnoj procesorskoj niti (što ne blokira sučelje), dok se upravljanje odgovorom izvodi na glavnoj niti (što omogućava ažuriranje sučelja ovisno o odgovoru). Sva komunikacija i razmjena podataka sa pozadinskim sustavom odrađena je korištenjem Volley biblioteke, dok su podaci u već spomenutom JSON formatu.

Za bolju kompatibilnost i komunikaciju za pozadinskim sustavom prilikom promjene trenutnog broja u redu korištena je **Firebase Cloud Messaging** biblioteka. Kombinacijom Firebase Admin biblioteke na pozadinskom sustavu i Firebase poslužitelja aplikacija instantno može reagirati na promjene broja u redu i o tome obavijestiti korisnika. Alternativa Firebase pristupu bilo bi opetovano slanje zahtjeva svakih nekoliko sekundi, što bi negativno utjecalo na performanse i potrošnju baterije mobilnog uređaja.

4.1.3 Obrazac dizajna

Kako bi se olakšalo čitanje i spremanje podataka unutar aplikacije te osiguralo da ti podaci uvijek dolaze iz istog izvora, korišten je **singleton** obrazac dizajna – način programiranja u kojem određeni objekt može imati samo jednu instancu za vrijeme izvođenja. U Android programiranju se to vrši nasljeđivanjem Application klase čime se dobiva pristup kontekstu cjelokupne aplikacije. Kontekst je bitan zbog čitanja i pisanja u lokalnu pohranu uređaja što se koristi za spremanje podataka prijavljenog korisnika. Uobičajena je praksa takvu klasu nazvati Controller (engl. *kontrolor*) jer se najčešće koristi za upravljanje podacima u cijeloj aplikaciji. Osim podacima o korisniku, singleton upravlja i njegovim rezervacijama te reagira na promjene u redovima javljajući korisniku, ako je tako postavljeno, **zvučnim signalom** ili prikazom **push obavijesti**. Prilikom odjave, iz lokalne pohrane uređaja brišu se korisnikovi podaci te se iz memorije prazni popis njegovih rezervacija.

4.1.4 Izvođenje, testiranje i otklanjanje grešaka

Budući da je Android operacijski sustav namijenjen za mobilne uređaje, a za razvoj se koristi stolno računalo, aplikaciju je prije izvođenja potrebno prevesti i prenijeti na odgovarajući Android uređaj gdje se može izvoditi. Android Studio omogućava, osim samog prevođenja, pokretanje i praćenje izvođenja aplikacije na **fizičkim** i **virtualnim** Android uređajima (**emulatorima**). Zbog specifične problematike redova čekanja koja

iziskuje postojanje više od jednog korisnika u redu, tijekom izrade aplikacije korišteni su i fizički i virtualni uređaji, često **istovremeno** kako bi se simulirao funkcionalni red čekanja. Osim emulatora koji dolazi uz Android SDK, korišten je i **Microsoft Visual Studio Android Emulator** u trenucima izrade aplikacije na računalu s AMD procesorom koji nije kompatibilan s ugrađenim emulatorom (kao ni s naprednim mogućnostima za instantno izvođenje aplikacije nakon manjih promjena bez potrebe za ponovnim potpunim prevođenjem).

Testiranje se obavljalo ručno, vizualnim pregledavanjem tijekom izvođenja aplikacije, bez korištenja posebnih alata ni biblioteka koje su za to namijenjene. Kako bi se olakšalo testiranje i otklanjanje grešaka (engl. *debugging*), korištene su ugrađene metode za ispis željenih varijabli u konzolu Android Studio-a (**LogCat**), što je sustav za bilježenje poruka i pogrešaka koje generira Android Studio, Android sustav i pokrenute aplikacije. (Duda, 2015)

4.2 Sučelje i interakcija

Sučelje mobilne aplikacije sastoji se, kao što je navedeno prilikom izrade prototipa sučelja, od osam zaslona (aktivnosti), a to su:

- prijava (početni zaslon)
- registracija
- popis ustanova
- detalji ustanove
- popis redova odabrane ustanove
- detalji odabranog reda (uzimanje broja/rezervacije)
- popis rezervacija
- postavke profila

4.2.1 Navigacija

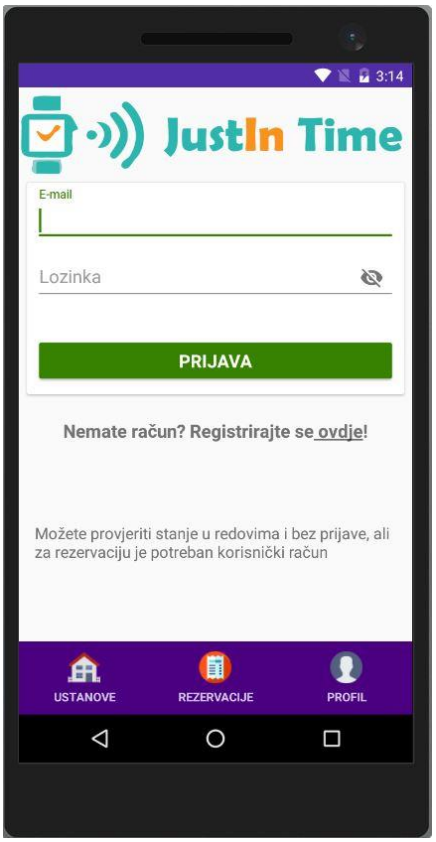
Pri dnu svakog zaslona (osim registracije, zbog bolje preglednosti svih polja za unos) nalazi se jednostavna traka za navigaciju s poveznicama na popis ustanova, popis rezervacija i postavke profila. Budući da se pojavljuje na gotovo svim zaslonima, navigacija je implementirana pomoću fragmenta koji se sastoji od vodoravnog linearnog razmještaja s tri gumba, od kojih svaki ima ikonu iznad teksta.

Zbog korištenja fragmenta, moguće je koristiti dodatnu logiku za prikaz navigacije

što omogućuje nekoliko izmjena prikaza navigacije ovisno o stanju aplikacije. Ako je korisnik trenutno na jednom od zaslona koji su navedeni u navigaciji, tekst odabrane poveznice prikazuje se drugom bojom. Budući da se unutar fragmenta može saznati koja je trenutna aktivnost, uvedena je provjera kako korisnik opetovanim odabirom iste poveznice ne bi pokretao ciljanu aktivnost iznova. Također, navigacija prema popisu rezervacija i postavkama profila moguća je samo prijavljenim korisnicima, što će fragment javiti skočnom porukom pri dnu zaslona (korištenjem Snackbar widgeta) ako je korisnik na zaslonu prijave, dok će ga u protivnom preusmjeriti na taj zaslon.

4.2.2 Prijava

Zaslon prijave početni je zaslon aplikacije. Glavni su mu dijelovi kartica (CardView) s poljima za unos korisničkog imena i lozinke s gumbom za slanje podataka te poveznica na zaslon registracije. Prateći Material Design načela, polja za unos podataka omeđena su TextInputLayout widgetom koji olakšava prikaz naljepnica, informacija i pogrešaka unosa u skladu s drugim Android aplikacijama. Također pruža korisniku odabir vidljivosti lozinke tijekom upisivanja, što može olakšati pristup aplikaciji.



Slika 21: Zaslon prijave mobilne aplikacije.
Izvor: autor

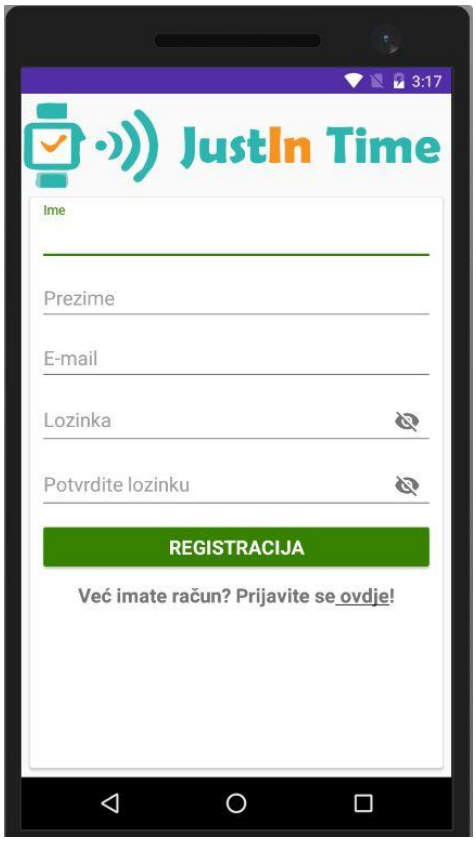
Glavna zadaća aktivnosti prijave je, očekivano, omogućiti korisniku prijavu u aplikaciju. Korisnik svoje podatke upisuje u za to predviđena polja za unos i potom odabire gumb „Prijava“. Tada kreće **validacija** unesenih podataka – polja ne smiju biti prazna i adresa e-pošte mora biti ispravnog oblika. U slučaju greške, korisniku se prikazuju poruke ispod problematičnih polja za unos. Validacija i naprednija polja za unos javljaju se i na drugim zaslonima (na sličan način) gdje se od korisnika traži unos podataka. Ukoliko je validacija podataka protekla uspješno, na zaslonu se prikazuje skočni dijalog o napretku (ProgressDialog) te se korisnički podaci u JSON obliku prilažu **Volley** POST zahtjevu za prijavu pozadinskom sustavu. Za razliku od većine ostalih Volley zahtjeva u aplikaciji, ovaj ima posebno **produljeno trajanje** od najviše trideset sekundi (uobičajeno je deset) kako bi se pozadinskom sustavu na Heroku platformi pružilo dovoljno vremena za pokretanje prije javljanja greške zbog nedostupnosti. Drugim riječima – ovaj zahtjev služi za „buđenje“ pozadinskog sustava. U slučaju pogreške korisniku se javlja poruka pomoću skočnog dijaloga (AlertDialog). Ako su potvrđeni podaci na pozadinskom sustavu, aplikaciji se vraćaju podaci o prijavljenom korisniku koje se sprema u lokalnu pohranu uređaja pomoću singletona. Kako bi se olakšao rad s podacima korisnika, stvorena je pomoćna klasa za **model** njegovih podataka, što je također česta praksa i odgovara poljima u bazi podataka. Nakon pohrane podataka, provjerava se ima li korisnik postojećih rezervacija. Korisnika se skočnim dijalogom obavijesti o uspješnoj prijavi te ga se potom preusmjeri na popis ustanova, odnosno na popis rezervacija (ako ih ima).

Budući da je ovaj zaslon početni, još jedna zadaća mu je provjeriti stanje prijavljenosti korisnika u aplikaciju odmah pri pokretanju. Ukoliko je već prijavljen, preskače se prikaz zaslona prijave te ga se preusmjerava na popis ustanova.

Zaslon prijave ima izravnu poveznicu na zaslon registracije, a nakon uspješne registracije korisnika se vraća na prijavu. Kako bi se poboljšalo korisničko iskustvo, u namjeru za povratak na prijavu prilaže se korisničko ime i lozinka kako bi se automatski popunila polja za unos podataka. Tako se izbjegava potreba da korisnik mora dvaput zaredom unositi iste podatke.

4.2.3 Registracija

Zaslon registracije sastoji se od polja za unos i potvrdu korisničkih podataka, gumba za slanje tih podataka te poveznice za prelazak na zaslon prijave. Svi elementi su smješteni unutar kartice unutar koje je ugniježđen ScrollView način razmještaja kako bi im korisnik, u slučaju ekrana manjih dimenzija, jednostavnim prelaskom prsta mogao pristupiti. Kako bi se potreba za tim nastojala izbjeći, ovaj zaslon nema uobičajenu navigaciju pri dnu.



Slika 22: Zaslon registracije mobilne aplikacije.
Izvor: autor

Polja za unos funkcioniraju gotovo identično kao na zaslonu prijave prije slanja zahtjeva prema pozadinskom sustavu – jedina je razlika dodatak polja za ime, prezime i potvrdu lozinke, koja također prolaze validaciju kako bi se osiguralo da ne bi bila prazna te da se obje lozinke podudaraju. Jednako tako se na isti način šalje POST zahtjev s unesenim podacima, tijekom kojeg je prikazan dijalog napretka. U slučaju neuspješne registracije, korisnika se obavještava o greški, posebice ako je pokušao unijeti adresu e-pošte koja je već zauzeta. Ako je sve proteklo u redu, korisniku se zahvaljuje na registraciji skočnim dijalogom. Nakon zatvaranja dijaloga, korisnika se preusmjerava na zaslon prijave uz prijenos njegovih podataka za prijavu.

4.2.4 Popis ustanova

Popis ustanova jedan je od tri zaslona u ovoj aplikaciji kojima je cilj prikazati popis stavaka. Sva tri zaslona popisa funkcioniraju na sličan način, razlikuju se po tome što i kako prikazuju te po tome kako reagiraju na dodir. Kako bi se u Androidu prikazao popis, nužno je implementirati sljedeće:

- widget koji sadrži popis (RecyclerView) unutar aktivnosti,
- XML datoteku koja definira izgled svake pojedine stavke unutar popisa,
- Java datoteku koja definira model podataka stavke,
- Java strukturu podataka (ArrayList) koja sadrži stavke u obliku objekata,
- adapter – Java klasu pomoću koje se povezuje model podataka s izgledom stavke što se iskorištava za svaki element strukture podataka i u kojoj se definira programska logika prilikom svake korisnikove interakcije s popisom

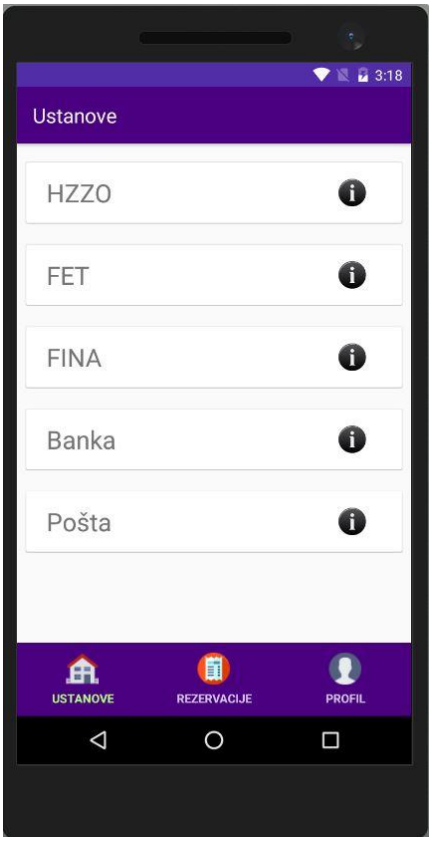
Navedeni postupak proveden je prilikom izrade popisa ustanova, redova i rezervacija. Kako bi korisnik mogao lakše i intuitivnije osvježiti popise, svaki RecyclerView je ugniježđen unutar SwipeRefreshLayout widgeta koji, uz odgovarajuću logiku unutar svake aktivnosti, prilikom pokušaja povlačenja popisa prema dolje ponovno dohvaća podatke i ažurira detalje popisa – što je intuitivno jer je implementirano u velikom broju popularnih Android aplikacija.

Dohvaćanje podataka o ustanovama obavlja se slanjem Volley GET zahtjeva pozadinskom sustavu. Ovaj Volley zahtjev još je jedan kojem je produženo trajanje prije javljanja greške, budući da je zaslon popisa ustanova potencijalan početni zaslon aplikacije ukoliko je korisnik već prijavljen u nju i mora biti u mogućnosti „probuditi“ pozadinski sustav, kao i zaslon prijave. Dohvaćeni podaci u obliku polja pretvaraju se iz JSON formata u listu pomoću odgovarajućih biblioteka i stvorenog modela podataka. Ta se lista prosljeđuje adapteru koji dalje određuje prikaz popisa. Ovaj se postupak ponavlja prilikom povlačenja za osvježavanje te prilikom vraćanja na ovaj zaslon.

Svaka stavka na prikazu popisa ustanova sastoji se od kartice koja sadrži naziv ustanove (koji nakon dodira korisnika vodi na popis redova) i gumb koji služi kao poveznica na zaslon detalja te ustanove. Logika za obje akcije implementirana je unutar adaptera tako što se stvara namjera kojoj se prilaže dodatak – id odabrane ustanove.

Na početku razvoja aplikacije korišten je standardni ListView widget pomoću kojeg

je moguće dobiti jednake rezultate, barem što se izgleda tiče. Službena dokumentacija ipak zbog boljih performansi predlaže korištenje RecyclerView-a prilikom rada s velikom ili nepoznatom količinom podataka, što je čest slučaj kod dohvaćanja podataka s Interneta.



Slika 23: Popis ustanova mobilne aplikacije.
Izvor: autor

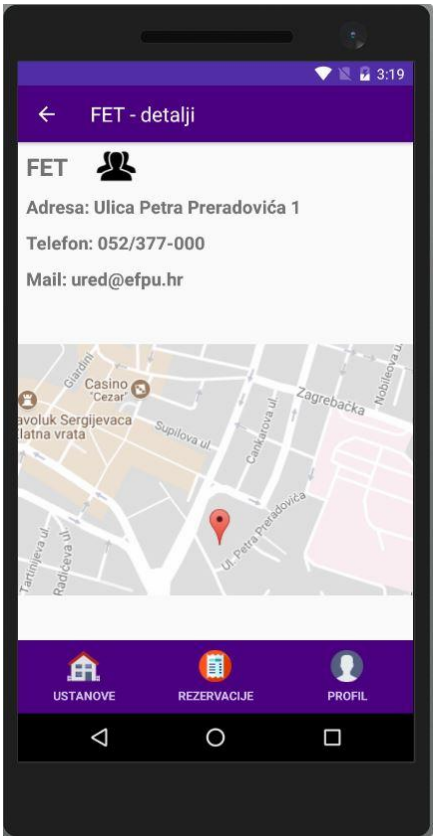
4.2.5 Detalji ustanove

Zaslon detalja ustanove pruža korisniku mogućnost pregleda podataka o odabranoj ustanovi. To su naziv, adresa, broj telefona i adresa e-pošte. S ovog zaslona korisnik može prijeći i na popis redova odabrane ustanove pomoću poveznice kraj naziva ustanove.

Dodir na broj telefona omogućava poziv ustanovi tako da se otvori sučelje za upućivanje poziva mobilnog uređaja i automatski popuni polje za unos broja. Na isti način moguće je, dodirom na adresu e-pošte, započeti sastavljanje poruke kojoj se popuni polje za unos naslova e-pošte.

Kako je poznata adresa ustanove, dodana je mogućnost uvida u njenu lokaciju pomoću Google karte. Prikazana je točna pozicija ustanove i bliža okolica putem

satelitskog prikaza. Dugi dodir na kartu omogućuje korisniku biranje naprednih mogućnosti (poput navigacije i uličnog prikaza) u posebnoj aplikaciji Google karata.



Slika 24: Zaslon detalja ustanove mobilne aplikacije.
Izvor: autor

Budući da se na ovaj zaslon dolazi s popisa ustanova, prilikom pokretanja se iz namjere mora pročitati id odabrane ustanove. Taj se id koristi za slanje Volley zahtjeva pozadinskom sustavu koji dohvaća sve potrebne podatke o ustanovi. Podaci se spremaju u objekt klase ustanova kako bi se olakšao rad s njima. Tada se elementi sučelja za prikaz teksta postavljaju na dohvaćene podatke i prikazuje se do tad nevidljiva poveznica na popis redova. Dodiri na telefonski broj i adresu e-pošte pokreću namjere koje upravljaju željenim radnjama, a prilažu im se traženi podaci. Ukoliko se pokreće popis redova, putem namjere se prosljeđuje id odabrane ustanove.

Za prikaz karte korišten je WebView okvir kojem je određena Web adresa (URL) kombinacijom poznate adrese za statični prikaz Google karata u obliku slika i modificirane adrese ustanove kojoj se dodaje poštanski broj i grad ustanove kako bi se lokacija uspješno prepoznala. Na ovaj je način izbjegnuta potreba za korištenjem dodatnih biblioteka za prikaz karata unutar aplikacije, a korisnik ipak može vidjeti (i pomicati) kartu lokacije ustanove. Dugi dodir na kartu putem namjere otvara aplikaciju

Google karte koja omogućuje sve podržane radnje s lokacijom, a korisnika se za tu mogućnost obavještava pomoću SnackBar prikaza pri dnu zaslona.

4.2.6 Popis redova

Na zaslon popisa redova korisnik dolazi odabirom željene ustanove s prethodnog popisa. Popis redova se dohvaća slanjem Volley GET zahtjeva pozadinskom sustavu uz prilaganje id-a odabrane ustanove (kojeg se pročita iz namjere). Postupak izrade prikaza popisa redova je vrlo sličan popisu ustanova – razlikuju se samo u modelu podataka, funkcionalnosti adaptera i izgledu pojedine stavke. Osim naziva reda, prikazan je broj koji se upravo poslužuje. Za taj je broj odabrana crna pozadina i digitalni izgled pisma crvene boje, što nastoji imitirati fizičke elektronske ploče s brojevima kakve se mogu pronaći u bankama i ostalim ustanovama koje su bile inspiracija za ovu aplikaciju. Adapter, osim ispravnog prikaza svake stavke, osigurava da dodir na željeni red prikaže zaslon detalja o redu kojem se potrebni podaci prenose pomoću namjere.

Slika 25: Zaslon popisa redova mobilne aplikacije.
Izvor: autor

4.2.7 Detalji o redu – rezerviranje broja

Nakon odabira reda, korisniku se na ovom zaslonu, osim imena tog reda i ustanove kojoj pripada, prikazuje broj koji se trenutno poslužuje u redu – slično kao i na popisu redova. U slučaju da je trenutni broj nula, korisnika se obavještava kako je u tijeku čekanje na službenika koji treba otvoriti red. Glavninu zaslona zauzima stilizirano dugme u obliku papirića s brojem kakvog se može pronaći kod fizičkih redomata koje služi za rezervaciju mjesta u redu – uzimanje broja.

Slika 26: Zaslon detalja o redu i uzimanje broja.
Izvor: autor

Prilikom prvog prikaza ovog zaslona iz namjere se čitaju podaci o odabranom redu te se prikazuju na predviđenim mjestima na zaslonu. Istovremeno se šalje zahtjev za novim podacima kako bi bili ažurni u slučaju da je u međuvremenu došlo do promjene u redu. Dodir na stilizirani gumb za uzimanje broja priprema Volley POST zahtjev pozadinskom sustavu za stvaranje nove rezervacije kojem se prilaže autorizacijski token prijavljenog korisnika. Kako pozadinski sustav može javiti greške prilikom pokušaja ulaska u red neprijavljenog korisnika te pokušaja ponovnog ulaska u red u kojem prijavljeni korisnik već ima rezervaciju, te se greške izričito javljaju korisniku skočnim dijalogom te se korisnika preusmjerava na zaslon prijave (ako nije prijavljen)

ili na zaslon popisa rezervacija (ako je već u tom redu).

Ukoliko nije došlo do pogreške i rezervacija je uspješno stvorena, poziva se instanca Firebase usluge unutar aplikacije i prosljeđuje joj se id reda kako bi se uređaj pretplatio na praćenje događaja koji se tiču tog reda. Nakon jedne sekunde (što odgovara završetku animacije) korisnika se preusmjerava na zaslon popisa rezervacija.

4.2.8 Popis rezervacija i prikaz obavijesti

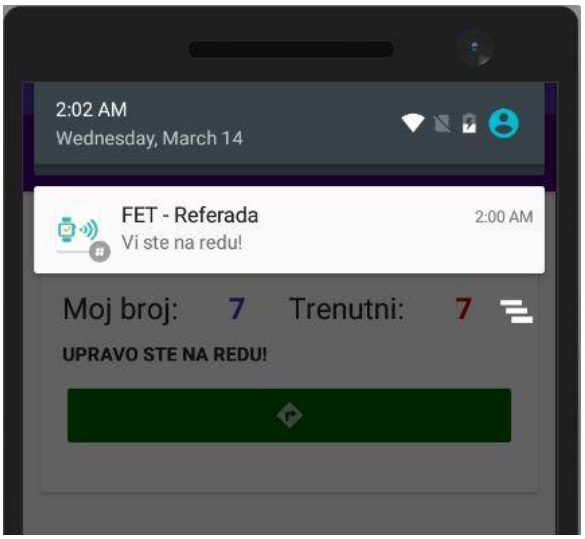
Popis rezervacija je treći u nizu zaslona (uz popis ustanova i redova) kojem je zadaća prikaz popisa te je, u suštini, izrađen na isti način – uz pomoć adaptera i posebnog modela za rezervacije. Razlikuje od ostalih popisa zbog dizajna svake stavke rezervacije, budući da je potrebno prikazati više detalja. Svaka stavka sadrži naziv ustanove i reda, korisnikov rezervirani i trenutni broj u tom redu, grubu procjenu preostalog vremena čekanja te gumba za otkazivanje rezervacije ili potvrde završetka rada. Popis je, kao i ostale, moguće ručno osvježiti povlačenjem prema dolje.

Slika 27: Zaslon popisa rezervacija mobilne aplikacije.
Izvor: autor

Dohvaćanje rezervacija obavlja se slanjem Volley GET zahtjeva pozadinskom sustavu uz prilaganje korisnikovog autorizacijskog tokena. Prije prevođenja JSON podataka provjerava se postoje li uopće rezervacije trenutnog korisnika, u kojem slučaju se umjesto popisa prikazuje (prethodno skriveni) tekstovni prikaz s natpisom „Nemate rezervacija“. Pronađene rezervacije se uz pomoć modela za ustanove, redove i rezervacije pohranjuju u listu te se pomoću Firebase usluge redom pretplaćuje korisnika na dohvaćanje novosti za red iz trenutno učitane rezervacije (koju se šalje singletonu na daljnju analizu). Lista rezervacija se prosljeđuje adapteru čija je zadaća uspješno prikazati podatke – osim postavljanja vrijednosti teksta, računa se procjena vremena čekanja množenjem razlike između korisnikovog i trenutnog broja s pet minuta. Ukoliko se ti brojevi podudaraju, prikaz se mijenja tako što se umjesto predviđenog vremena čekanja prikazuje podebljana poruka „Upravo ste na redu!“ te se boja gumba za otkazivanje rezervacije mijenja u zelenu kako bi bolje privukla pozornost korisnika, zajedno s drukčijom ikonom gumba. Također se razlikuje poruka koja se prikazuje pritiskom na gumb za otkazivanje – iako nema razlike prilikom slanja Volley DELETE zahtjeva pozadinskom sustavu s korisnikovim autorizacijskim tokenom. Nakon uspješnog otkazivanja rezervacije, korisniku se prikazuje kratka poruka o uspjehu te se ponovno pokreće aktivnost popisa rezervacija kako bi se osvježila. Istovremeno se otkazuje primanje daljnjih novosti o redu iz te rezervacije korištenjem Firebase usluge.

Kako bi se mogle pratiti promjene stanja u rezervacijama, unutar singletona aplikacije stvorene su posebna metoda i struktura podataka baš za tu namjenu. Primajući pojedini objekt klase rezervacija prilikom učitavanja svih rezervacija (unutar aktivnosti popisa rezervacija) ili prilikom dohvaćanja push novosti kroz Firebase uslugu (koja prima gotovo iste podatke kao i aktivnost, samo ne u JSON zapisu), ova metoda osigurava ispravno upravljanje rezervacijama s jednog mjesta. Za njihovo spremanje koristi se struktura podataka tipa Map kojoj je ključ id reda, a vrijednost objekt klase rezervacija. Proslijeđena rezervacija unutar metode se analizira tako da joj se najprije pročita id reda, pokuša pronaći postojeća rezervacija s istim ključem te se naposljetku svakako dodaje u mapu. Ukoliko nije pronađena postojeća rezervacija, nema potrebe za daljnjom usporedbom u ovoj iteraciji. S druge strane, ako je već bila pohranjena rezervacija s istim ključem, potrebna je daljnja analiza usporedbom trenutnog i korisnikovog broja, brojeva unutar postojeće i nove rezervacije te korisničkih postavki

o zvučnim i push obavijestima. Tako se u slučaju nepoklapanja trenutnih brojeva unutar rezervacija podrazumijeva kako je red napredovao i da korisnik nije upravo na redu – ukoliko je omogućeno, zvučni signal će obavijestiti korisnika o toj promjeni u redu. Ako je korisnik upravo na redu, poklapat će se njegov i trenutni broj. Prije prikazivanja push obavijesti vrši se još jedna usporedba nove i stare rezervacije kako se ne bi ručnim osvježavanjem popisa rezervacija ponavljala push obavijest. Push obavijest sastoji se od ikone aplikacije, manje ikone sa simbolom broja („#“), naziva ustanove i reda te poruke „Vi ste na redu!“. Boja i zvuk obavijesti prepuštene su zadanim postavkama korisnikovog uređaja. Prije prikaza obavijesti otkazuje se pretplata na daljnje novosti korištenjem Firebase usluge. Dodir na obavijest je istovremeno uklanja sa zaslona te prikazuje zaslon popisa rezervacija. Budući da naknadno osvježavanje rezervacija primanjem push novosti obavlja Firebase usluga, to omogućava korisniku da normalno koristi uređaj dok čeka u redu – ne mora imati ovu aplikaciju u prvom planu, već može primjerice pregledavati slike, društvene mreže ili jednostavno držati zaslon ugašenim.



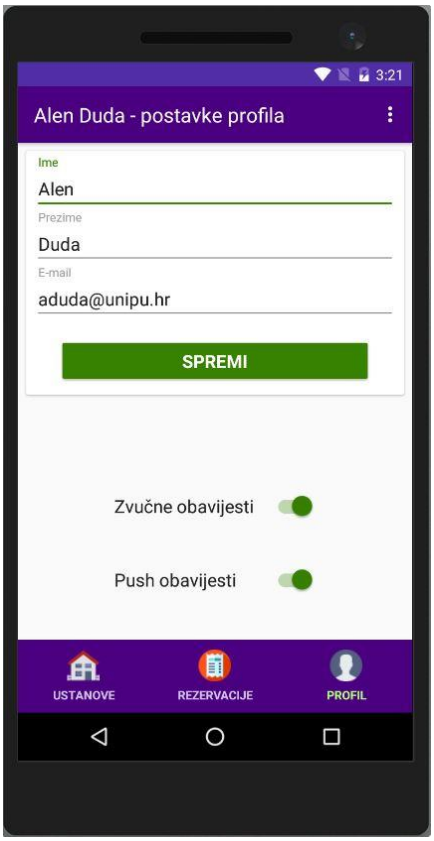
Slika 28: Push obavijest na mobilnom uređaju.
Izvor: autor

4.2.9 Postavke profila

Na zaslonu postavki profila korisnik može promijeniti podatke svog računa – ime, prezime i adresu e-pošte. Osim tih podataka, koji se trajno spremaju u sustavu, korisnik može na trenutnom uređaju odrediti želi li primati zvučne i push obavijesti tijekom promjene u redu. Za ovaj zaslon stvoren je i izbornik pomoću kojeg se korisnik može odjaviti iz aplikacije.

Polja za unos korisnikovih podataka funkcioniraju i izgledaju identično kao na zaslonu na registraciju, uključujući i njihovu validaciju prije slanja pozadinskom sustavu. Validacija i slanje podataka obavlja se nakon dodira gumba za spremanje podataka što šalje Volley PUT zahtjev pozadinskom sustavu s priloženim unesenim podacima i autorizacijskim tokenom korisnika. Tijekom izvršavanja zahtjeva privremeno se onemogućava gumb za spremanje i prikazuje animirani kružni indikator o napretku. U slučaju pogreške korisniku se javlja skočnim dijalogom o nastalom problemu. Ukoliko nema pogreške, potrebno je ažurirati podatke korisnika unutar aplikacija, za što je zaslužna posebna metoda u singleton klasi. Tada se korisnika skočnim dijalogom obavještava kako su podaci uspješno spremljeni.

Singleton klasa sprema i postavke o obavijestima pomoću važećih metoda koje se pozivaju prilikom mijenjanja tih postavki. Ukoliko korisnik ne utječe na te postavke, zadano je da su obje opcije uključene. Izbornik s opcijom za odjavu također pokreće metodu u singleton klasi koja uklanja sve spremljene podatke o korisniku unutar lokalne pohrane. Zatim prolazi kroz ranije spomenutu mapu s rezervacijama čitajući id-ove redova i redom otkazuje pretplate na dohvaćanje push novosti za svaki od njih pomoću Firebase usluge. Tada se prazni mapa s rezervacijama iz memorije, a korisnika preusmjerava na zaslon prijave.



Slika 29: Zaslon postavki profila mobilne aplikacije.
Izvor: autor

5 MOGUĆA POBOLJŠANJA

Stvaranje aplikacija je iterativan proces u kojem uvijek ima mjesta za poboljšanja, što se može tumačiti kao da taj proces nikad ne prestaje. Ipak, u određenom trenutku se mora odrediti što je minimum kojeg se može smatrati dovoljno dobrim za početak kako bi se počelo koristiti proizvod ili, u slučaju aplikacije opisane u ovom radu, moglo krenuti pisati o njoj. U ovom poglavlju navest će se neke ideje za poboljšanja koja bi mogla unaprijediti ne samo mobilnu aplikaciju, već cjelokupan sustav za upravljanje redovima čekanja.

5.1 Dodatni podaci o ustanovama i redovima

Trenutno se u bazi podataka nalaze samo ustanove i redovi koji im pripadaju. Ustanove bi trebalo razvrstati po **kategorijama** kako bi se, primjerice, lakše pronašla željena banka. Istovremeno se mogu dodati **gradovi** u kojima se nalaze pojedine ustanove. Korištenjem **lokacijskih usluga** mobilnog uređaja izbjegla bi se potreba da korisnik ručno bira grad, već bi se automatski odabrao najbliži (uz zadržavanje mogućnosti ručnog biranja).

Za ustanove i redove može se dodati kratki **opis** i **fotografija**, što bi moglo olakšati snalaženje korisnika u nepoznatom gradu. Opis može sadržavati kratku povijest ustanove te popis usluga koje se mogu obaviti u pojedinim redovima.

5.2 Statistika i administracija

Trenutno se rezervacije u redovima pokreću jednostavnim brisanjem postojećih što onemogućuje vođenje ikakve statistike. Rezervacije već spremaju datum i vrijeme stvaranja (koje se zasad koristi samo za sortiranje) pa bi se jednostavnim dodavanjem **vremena završetka** rezervacije omogućio **pregled povijesti** rezervacija unutar svakog reda (aktivne rezervacije bile bi one koje još nemaju vrijeme završetka). Računanjem razlike početka i kraja rezervacije moglo bi se izračunati vrijeme čekanja svake osobe, što se može koristiti i za **preciznije predviđanje** vremena čekanja.

U sustav bi se moglo dodati **administratore zadužene** za određene redove i tako ograničiti upravljanje tim redovima samo na ovlaštene osobe. Istovremeno, korisnicima bi se mogla ponuditi opcija **odabira željene radnje** u redu (npr. plaćanje računa, podizanje novca s tekućeg računa). Ti se podaci mogu evidentirati u rezervaciji što bi unaprijedilo statističko praćenje učinkovitosti i predviđanje vremena čekanja.

Bitan podatak stvarnih ustanova je njihovo **radno vrijeme**, uključujući **pauze** za marende. Svaki red bi trebao moći imati evidentirano radno vrijeme za svaki dan u tjednu, uz dodanu mogućnost zaduženog administratora da **privremeno zatvori red** i tako onemogući daljnje rezervacije. Zakazane pauze trebaju se uzeti u obzir prilikom predviđanja vremena čekanja.

5.3 Firebase integracija

Ranije u radu navedene su neke mogućnosti Firebase platforme. Korištenjem nekih od njih mogao bi se poboljšati način rada cijelog sustava.

Firebase omogućava praćenje stvaranje korisničkih računa na razne načine. Ta bi se činjenica mogla iskoristiti kako bi se korisnicima omogućila prijava u aplikaciju korištenjem **postojećih računa društvenih mreža** (Facebook, Twitter…) i Google računa kojeg gotovo svi Android korisnici već posjeduju.

Baza podataka u **realnom vremenu** jedna je od najkorištenijih značajki Firebase platforme. Korištenjem nje umjesto konvencionalnih baza korisnici bi mogli bez ručnog osvježavanja pratiti promjene brojeva u redovima u trenutku kad se promijene.

5.4 Korisničko iskustvo

Ako je mobilna aplikacija jedini način korisničke interakcije sa sustavom, ona mora biti vizualno atraktivna i pristupačna, što je najbolje prepustiti profesionalnom **dizajneru** prije puštanja aplikacije u uporabu.

Kako bi se povećao potencijalni broj korisnika, osim mobilne aplikacije moglo bi se stvoriti i **mobilnu Web aplikaciju** (korištenjem neke od tehnologija opisanih ranije u radu), budući da je vjerojatnije da će korisnici koristiti Web aplikaciju koju **nije potrebno instalirati** na uređaj.

Jedno od najjednostavnijih poboljšanja je dodavanje podrške za više jezika. **Prijevod sučelja** olakšao bi rad turistima te je vrlo dobra praksa prilikom izrade Android aplikacija.

Ukoliko se poveća broj ustanova i redova, nužno je dodati **mogućnost pretrage** koja obuhvaća naziv, opis i lokaciju. Tako bi, primjerice, korisnik mogao pronaći sva mjesta gdje može platiti račune – banka, pošta i FINA i odabrati najbližu ustanovu ili onu u kojoj je najkraće vrijeme čekanja (uz mogućnost sortiranja).

Trenutno se lokacija ustanove koristi za osnovni prikaz karte. To se može proširiti

boljom integracijom Google karata s uključenim **računanjem rute** do odabrane ustanove unutar aplikacije.

Izvještavanje korisnika o **greškama** trebalo bi biti **detaljnije** i **preciznije** uz **savjet** kako otkloniti grešku – npr. ako uređaj nije spojen na Internet, objasniti kako je za uspješan rad aplikacije nužna stalna veza na Internet. Istovremeno treba uložiti više truda u robusnost aplikacije kako bi se izbjegle greške.

Korisnik bi trebao moći **birati zvuk obavijesti**, primjerice za svaku ustanovu ili red drukčiji zvuk može omogućiti korisniku da zna u kojem se redu promijenio broj bez gledanja u zaslon mobilnog uređaja.

Sigurnost sustava se uvijek može poboljšati, jedan od načina je **skraćivanje vijeka trajanja autorizacijskog tokena** korisnika uz mogućnost zahtijevanja dugotrajnijeg iz aplikacije (tzv. osvježavanje tokena). Odjavom iz aplikacije token bi trebao prestati vrijediti. Korisniku bi trebalo omogućiti **mijenjanje lozinke** iz aplikacije te zatražiti stvaranje nove uz slanje poruke o potvrdi ukoliko je stara zaboravljena.

Ukoliko se u **mobilnu aplikaciju** prijavi **administrator**, može mu se omogućiti **upravljanje redovima** pomoću posebnog dijela sučelja kojem običnim korisnici nemaju pristup. Ovo bi svakako zahtijevalo višu razinu sigurnosti i povećalo bi veličinu aplikacije pa je moguća i druga opcija – stvaranje posebne aplikacije samo za administraciju.

ZAKLJUČAK

Dugotrajno čekanje u redovima je ponekad ozbiljan problem. U prethodnim poglavljima prikazan je cjelokupan postupak i metodologija pokušaja njegovog rješavanja pomoću informatičko-komunikacijskih tehnologija i programiranja. Prvo se prepoznao problem, istražili su se i usporedili mogući tehnički načini za rješavanje problema te je odabran onaj koji se smatrao najprikladnijim obzirom na autorovo prethodno iskustvo i znanje. Potom je krenula izrada prototipa sučelja, dizajn baze podataka i način interakcije korisnika s aplikacijom uz praćenje dokumentacije i načelno dobrih praksi razvoja. Po navedenim specifikacijama krenuli su se primjenjivati elementi izrade Android aplikacija gdje su prepoznati izazovi u tehničkoj implementaciji. Daljnjim istraživanjem različitih dostupnih izvora uspjelo se savladati izazove ili ih u potpunosti zaobići promjenom načina izvedbe.

Rezultat rada je osnovni sustav za upravljanje redovima čekanja koji je daleko od potpunog proizvoda i može poslužiti kao početak za daljnji razvoj. Neke od ideja za poboljšanja navedene su u radu. Nakon postepene implementacije nekih od navedenih mogućnosti, moglo bi se početi razmišljati o praktičnom uvođenju sustava u neke ustanove kao ograničeni pokus. Prikupljanjem podataka od korisnika i administratora prepoznale bi se slabosti i otkrili dodatni načini za poboljšanje sustava kojem je cilj ukloniti potrebu za fizičkim čekanjem u redu i tako sačuvati najvažniji resurs – vrijeme.

Nažalost, u stvarnosti će uvijek postojati ustanove i redovi u kojima nije moguće u potpunosti implementirati ovakav sustav. To najviše dolazi do izražaja zbog toga što nemaju sve osobe (pogotovo starije dobi) nekakav pametni mobilni uređaj kojim mogu rezervirati mjesto u redu. Stoga je potrebno dodatno istraživanje i, osim ranije navedenih poboljšanja, dodatak sustavu koji bi osobama omogućio rezerviranje mjesta u redu na licu mjesta (primjerice, korištenjem fizičkih redomata kakvi već postoje). Takav sustav morao bi se moći integrirati i s postojećim fizičkim digitalnim pločama koje prikazuju brojeve, što bi mogao biti zanimljiv projekt za neki budući istraživački rad s teoretskim i praktičkim dijelovima.

LITERATURA

a) INTERNET IZVORI

1. AppBrain, 2018. *Android app frameworks.* [Mrežno]
   Available at: https://www.appbrain.com/stats/libraries/tag/app-framework
   [Pokušaj pristupa 14 01 2018].
2. Cruxlab, Inc., 2017. *Xamarin vs Ionic vs React Native: differences under the hood.* [Mrežno]
   Available at: https://medium.com/swlh/xamarin-vs-ionic-vs-react-native-differences-under-the-hood-6b9cc3d2c826
   [Pokušaj pristupa 17 01 2018].
3. Devslopes, 2017. *Beginner API development in Node, Express, ES6 & MongoDB.* [Mrežno]
   Available at: https://www.udemy.com/api-development/learn/v4/content
   [Pokušaj pristupa 26 01 2018].
4. Facebook, 2018. *React Native.* [Mrežno]
   Available at: https://facebook.github.io/react-native/
   [Pokušaj pristupa 18 01 2018].
5. Google, 2017. *Android Studio Release Notes.* [Mrežno]
   Available at: https://developer.android.com/studio/releases/index.html
   [Pokušaj pristupa 14 01 2018].
6. Google, 2017. *Kotlin and Android.* [Mrežno]
   Available at: https://developer.android.com/kotlin/
   [Pokušaj pristupa 14 01 2018].
7. Google, 2018. *Support Library - Android Developers.* [Mrežno]
   Available at: https://developer.android.com/topic/libraries/support-library/index.html
   [Pokušaj pristupa 19 02 2018].
8. MongoDB, 2018. *What is MongoDB.* [Mrežno]
   Available at: https://www.mongodb.com/what-is-mongodb
   [Pokušaj pristupa 23 01 2018].
9. Schwarzmüller, M., 2017. *React Native - The Practical Guide.* [Mrežno]

Available at: https://www.udemy.com/react-native-the-practical-guide/learn/v4/content
[Pokušaj pristupa 18 01 2018].

10. Traversy, B., 2017. *Learn Best Five Mobile App Frameworks By Building Projects.* [Mrežno]
Available at: https://www.udemy.com/learn-best-five-mobile-app-frameworks-by-building-projects/learn/v4/content
[Pokušaj pristupa 01 19 2018].

b) OSTALI IZVORI

1. Duda, A., 2015. *Programiranje za Android - završni rad.* 1. ur. Pula: Sveučilište Jurja Dobrile u Puli.

POPIS SLIKA

Slika 1: Android Studio. 3
Slika 2: Usporedba odgovarajućih Web, Android, iOS i React Native komponenti. 7
Slika 3: Primjer stvaranja popisa korištenjem React Native okvira. 8
Slika 4: Prototip sučelja zaslona prijave. 13
Slika 5: Prototip sučelja zaslona registracije. 14
Slika 6: Prototip sučelja zaslona popisa ustanova. 15
Slika 7: Prototip sučelja zaslona detalja ustanove. 16
Slika 8: Prototip sučelja zaslona popisa redova. 16
Slika 9: Prototip sučelja zaslona detalja reda. 17
Slika 10: Prototip sučelja zaslona popisa rezervacija. 17
Slika 11: Prototip sučelja zaslona postavki profila. 18
Slika 12: Testiranje REST sučelja pomoću programa Postman. 23
Slika 13: Prikaz komponenti sustava i korištenih tehnologija. 24
Slika 14: Mongoose shema resursa korisnici. 26
Slika 15: Mongoose shema resursa ustanove. 27
Slika 16: Mongoose shema resursa redovi. 28
Slika 17: Mongoose shema resursa rezervacije. 29
Slika 18: Zaslon prijave administracijskog sučelja. 33
Slika 19: Zaslon pregleda administracijskog sučelja. 34
Slika 20: Zaslon mijenjanja podataka ustanove. 35
Slika 21: Zaslon prijave mobilne aplikacije. 40
Slika 22: Zaslon registracije mobilne aplikacije. 42
Slika 23: Popis ustanova mobilne aplikacije. 44
Slika 24: Zaslon detalja ustanove mobilne aplikacije. 45
Slika 25: Zaslon popisa redova mobilne aplikacije. 46
Slika 26: Zaslon detalja o redu i uzimanje broja. 47
Slika 27: Zaslon popisa rezervacija mobilne aplikacije. 48
Slika 28: Push obavijest na mobilnom uređaju. 50
Slika 29: Zaslon postavki profila mobilne aplikacije. 52

POPIS PRILOGA

1. Izvorni kod mobilne aplikacije
2. Izvorni kod pozadinskog sustava
3. Izvorni kod Web aplikacije

SAŽETAK

U ovom radu istražili su se različiti načini izrade mobilnih aplikacija s naglaskom na Android operacijski sustav. Cilj istraživanja je bio odabir najprikladnijeg načina izrade kako bi se stvorila mobilna aplikacija za upravljanje redovima čekanja. Opisan je postupak izrade aplikacije – njenog izgleda i funkcionalnosti krenuvši od prototipa sučelja, potpornih sustava te, naposljetku, same Android aplikacije korištenjem Android Studio razvojnog okruženja. Potporni sustav sastoji se od pozadinskog sustava koji upravlja bazom podataka i Web aplikacije koja služi kao administracijsko sučelje. Aplikacija omogućava registraciju i prijavu korisnika, pregled popisa ustanova i redova unutar njih, pregled detalja svake ustanove, rezerviranje mjesta u redu te primanje obavijesti prilikom promjena u tom redu. Izrađeni sustav ima puno mjesta za napredak, a neke od ideja za poboljšanja su opisane na kraju rada.

Ključne riječi: redovi čekanja, Android, mobilna aplikacija, Web aplikacija, pozadinski sustav

SUMMARY

In order to find the most appropriate method of developing a mobile application, this paper started by researching the most popular and widespread ones, focus being on the Android operating system. Following the research, development started on a queue-managing mobile Android application – documenting the whole process from start to finish. This includes user interface mockup, application behavior, support systems and the application development itself, using Android Studio integrated development environment. Support systems are composed of a database and backend system which manages it and a Web application that enables system administration. The application enables user registration and login, browsing lists of facilities and queues within them, inspecting facility details, reserving spots within queues and receiving notifications upon changes in the respective queue. The system has much room for improvements, some of the ideas have been mentioned at the end of the paper.

Keywords: queues, Android, mobile application, Web application, backend