SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5115

VIŠEPLATFORMSKA MOBILNA APLIKACIJA ZA SUSTAV ZA TESTIRANJE STUDENATA EDGAR

Stjepan Petruša

Zagreb, lipanj 2017.

Zahvaljujem se zaposlenicima tvrtke Infinum za svu pomoć koju su mi pružali tijekom izrade aplikacije, te mentoru doc. dr. sc. Igoru Mekteroviću na uputama, smjernicama i pomoći za bolje razumijevanje sustava Edgar.

Sadržaj

Uvod

Testiranje znanja studenata najčešće se vrši u pisanom obliku. Kako bi se ubrzao proces rješavanja, ispitivanja i ispravljanja razvili su se razni web servisi koji omogućuju lako i brzo testiranje znanja kroz više raznih vrsta pitanja. Jedan od takvih servisa je i Edgar, sustav koji se razvio na Zavodu za primijenjeno računarstvo na Fakultetu elektrotehnike i računarstva u Zagrebu.

Sustav Edgar trenutno podržava ispitivanje studenata kroz pitanja s više ponuđenih odgovora i pitanja za testiranje SQL upita. Uz web platformu korisno je imati i mobilnu aplikaciju kako bi se studentima povećala pristupačnost zadacima. Za izradu mobilne aplikacije koristit će se tehnologije koje su dobro poznate u web svijetu, kao što je biblioteka React Native koja omogućuje izradu višeplatformske mobilne aplikacije koristeći jedan izvorni kod. Mobilne platforme koje ćemo pokriti ovim razvojem su Android i iOS, pošto one čine preko 99% globalnog tržišta[1][10].

Mobilna aplikacija, za razliku od postojeće web aplikacije, će pružati samo određen dio mogućnosti koje Edgar nudi. Tako će Edgar mobile omogućiti korisnicima prijavu, rješavanje testova koji se sastoje od pitanja s više ponuđenih odgovora, te pregled rezultata i točnih odgovora na istom testu. Za ostvarenje tih mogućnosti logika na mobilnoj aplikaciji mora biti istovjetna logici koja se koristi na web platformi.

[1] 2016. godine Android je imao 86.8%, a iOS 12.5% udjela na globalnom tržištu mobilnih operativnih sustava [10]

1 – **Tehnologija**

Najveća razlika između standardnog načina razvoja mobilnih aplikacija koji je opisan u poglavlju 2 i načina na koji je razvijena Edgar mobilna aplikacija je u samom programskom jeziku. Za razvoj se primarno koristio programski jezik JavaScript, što je dosta čudno s obzirom na to da je JavaScript namijenjen za izvršavanje u web pregledniku, a ne za pisanje nativnih mobilnih aplikacija. U sljedećim poglavljima kratko će biti opisan razvoj JavaScripta kao programskog jezika, te biblioteke koje omogućuju nestandardni način korištenja JavaScript jezika.

1.1 – **JavaScript**

Sve bržem razvoju interneta ‘90-ih godina prošlog stoljeća stvorila se potreba za lakšim razvojem interaktivnih dijelova web stranica koje su se do tada sastojale samo od HTML predloška. Skupina stručnjaka iz tvrtke Netscape Communications stvara prvu inačicu današnjeg JavaScripta pod nazivom LiveScript[2]. Do preimenovanja u JavaScript je došlo zbog sve veće popularnosti programskog jezika Java, te se tako htjelo privući što veći broj razvojnih inženjera. [11]

Slika 1.1 – JavaScript logotip [1]

[2] Jezik je prvo bio razvijan pod nazivom Mocha, no kasnije se promijenio naziv u LiveScript

Danas se JavaScript smatra jednim od popularnijih programskih jezika, te uz HTML i CSS čini osnovu svih web sustava. Sve do danas JavaScript je prošao kroz nekoliko iteracija promjena standarda prilagođavajući se potrebama razvojnih inžinjera, te dodajući nove i poboljšane funkcionalnosti. Najveće promjene su se dogodile 2015. godine kada je predstavljena nova verzija standarda pod nazivom ES2015[3].

Postoji više raznih verzija JavaScript interpretera. Neki od najpopularnijih su V8, SpiderMonkey i ChakraCore. V8 interpreter je toliko razvijen da je omogućio razvoj platforme Node.js, odnosno omogućio je izvršavanje JavaScript koda ne samo u web pregledniku, nego i na poslužitelju. Zato danas JavaScript možemo koristiti za razvoj web i poslužiteljskih, ali i mobilnih i desktop aplikacija.

Sve veća popularnost JavaScripta dovela je do razvoja velikog broja raznih biblioteka i paketa koji koriste mnoge potencijale JavaScript jezika. Veći dio biblioteka se koristi za manipulaciju HTML predloška kako bi se ostvario efekt besprekidnog prijelaza s jednog stanja aplikacije na drugo. (Aplikacija kreirana koristeći jednu od takvih biblioteka naziva se *one-page* aplikacija). Tako sve informacije potrebne za korištenje aplikacije se nalaze u JavaScript kodu i potrebno je učitati samo jedan HTML predložak i odgovarajući JavaScript kod koji onda dalje mijenja predložak i prikazuje informacije ovisno o trenutnom stanju aplikacije. Tako se komunikacija s poslužiteljem odvija samo kod inicijalnog učitavanja aplikacije, dok na svim drugim prijelazima aplikacija po potrebi obavlja AJAX pozive[4] na poslužitelj kako bi dohvatila podatke za prikaz.

Neke od popularnijih biblioteka su React, o kojem će biti više riječi u sljedećem poglavlju, Angular, Vue i Ember. Postoji još veliki broj biblioteka koje rješavaju isti problem, no rješavaju ga na drugačiji način, ili gledajući iz druge perspektive. Broj tih biblioteka je toliko velik da postoji pojam koji opisuje nemoć

[3] ES2015 je skraćenica za ECMAScript 2015. ECMAScript je specifikacija za standardizaciju JavaScripta [12]

[4] AJAX pozivi (eng. Asynchronous JavaScript and XML) opisuju asinkrone pozive iz JavaScript koda prema poslužitelju. Takvi pozivi su stekli veliku popularnost jer ne zahtijevaju od web preglednika da ponovno učita cijelu aplikaciju, nego da dohvati samo određeni resurs s kojime se lako manipulira unutar JavaScripta (najčešće se radi o resursima u JSON formatu) [13]

samog razvojnog inženjera da odabere “pravu” biblioteku, a naziva se “*JavaScript fatigue*”.

1.2 – **React**

React je trenutno najpopularnija biblioteka za izradu *one-page* web aplikacija. Tim programera unutar tvrtke Facebook je razvio React zbog potrebe za bibliotekom koja veoma brzo može manipulirati HTML predloškom[5]. To je ostvareno tako da se u memoriji nalazi cijela struktura HTML predloška u obliku stabla (*virtual DOM*), te se prvotno sve promjene vrše nad tom strukturom koja se kasnije sinkronizira sa stvarnom strukturom. Na taj način se mogu sinkronizirati samo dijelovi koji su mijenjani, a ne cijelo stablo, odnosno cijela struktura HTML elemenata.

React se temelji na dizajnu baziranom na komponentama. Tako se sva logika koja se piše unutar biblioteke nalazi u odvojenim, enkapsuliranim komponentama, te se kasnije te komponente međusobno povezuju. Tu je bitno napomenuti jedan od oblikovnih obrazaca ako se radi o dizajnu baziranom na komponentama, a to su postojanje dvije vrste komponenti: prezentacijske i smještajne komponente. Razlika je sljedeća:

- **Prezentacijske** – služe za prikaz stanja aplikacije, te koriste se za ostvarivanje interakcije sa samim korisnikom. One znaju prikazati određeno stanje, no ne znaju mijenjati to isto stanje. Pošto one ovise o parametrima koje prime, u većini slučajeva se mogu ponovno koristiti na više različitih mjesta u aplikaciji

- **Smještajne** – služe za dohvaćanje i manipulaciju podacima, te za organizaciju prezentacijskih komponenti. Pošto te komponente manipuliraju podacima imaju veoma specifičnu ulogu, te se ne mogu korisitit na drugim mjestima u aplikaciji.

[5] Manipuliranje HTML predloška iz JavaScript koda je jedno od sporijih dijelova web aplikacija zbog sporog pretraživanja elemenata, te zbog potrebe za ponovnim prevođenjem tog istom HTML predloška nakon obavljene promjene.

Komponente unutar React okruženja nalaze se unutar .jsx[6] datoteka. JSX datoteka opisuje standardnu JavaScript datoteku, no sadrži dodatne funkcionalnosti kako bi se olakšala izrada HTML predloška, točnije unutar JSX datoteka dozvoljeno je pisanje HTML elemenata. Kasnije React to formatira u normalan JavaScript kod, a svako spominjanje HTML elemenata zamijeni s `React.createElement` metodom. Promjer jedne takve datoteke je prikazan u programskom kodu 1.2.

```
import React from 'react';

class Greeting extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'world!'
    };
  }

  render() {
    return <h1>Hello {this.state.name}</h1>;
  }
}
```

Programski kod 1.2 – Jednostavna React komponenta

1.3 – **React Native**

React Native, za razliku od React biblioteke, koristi se za izradu mobilnih aplikacija. Način razvoja React web aplikacije i React Native mobilne aplikacije je dosta sličan, te poznavanje React biblioteke može mnogo pomoći u razvoju React Native aplikacije.

Trenutno na tržištu postoje biblioteke koje rade sličan posao, odnosno omogućuju pisanje mobilnih aplikacija koristeći JavaScript. Neki od primjera su Cordova i PhoneGap, no njihov rad se svodi na kreiranje rudimentarne mobilne

[6] .jsx ekstenzija označava JavaScript XML datoteku.

aplikacije koja sadrži jedan WebView[7] unutar kojega se izvršava napisani JavaScript kod. Takva aplikacija se dalje može razvijati kao nativna mobilna aplikacija i tako implementirati dodatne mogućnosti, no središnji dio aplikacije se i dalje svodi na prikaz jedne WebView komponente.

Još jedna dosta bitna razlika između React Native biblioteke i ostalih sličnih biblioteka je ta što React Native biblioteka promiče drugačiju filozofiju. Cilj React Native biblioteke je taj da razvojni inženjeri nauče jednu biblioteku i koristeći nju mogu pisati aplikacije ili za Android ili za iOS ili za obje platforme. Tako razvojni inženjer nije ograničen znanjem platforme, nego znanjem biblioteke. Android i iOS su podosta različite platforme i drugačiji način korištenja. Razlike u stilovima su opisane u poglavlju 4.3. React Native biblioteka pokušava objediniti obje platforme tako da se sama brine o različitostima između platformi. Razvojni inženjer mora na primjer samo navesti da želi postaviti *Button* element na dno ekrana, a React Native biblioteka će prikazati *Button* element različitih stilova i različitih postavki ovisno o očekivanom ponašanju na određenoj platformi. Cordova i ostale slične biblioteke nemaju takvu mogućnost unutar same biblioteke, nego očekuju od razvojnog inženjera da zna kakvo je očekivano ponašanje na pojedinoj platformi i takvo ponašanje sam implementira.

React Native kreira kod u nativnom jeziku za navedenu platformu. Od tuda i "Native" u nazivu biblioteke. Trenutno React Native podržava izradu aplikacije za mobilne operativne sustave Android i iOS, što znači da React Native ima podršku za pretvorbom vlastitog JavaScript koda u Java i Objective-C kodove prikladne za pokretanje na mobilnom uređaju.

Jedna od značajnih razlika u razvoju React i React Native aplikacije je ta što se u React Native aplikaciji više ne pišu HTML predlošci za komponente. Mobilne platforme ne koriste HTML za prikaz elemenata, već imaju vlastite XML elemente koji se mogu koristiti. Zato se unutar React Native koda neće nikada nalaziti elementi kao što su `<div />`, `<h1></h1>`, `<input />` ili `<header />`. React Native pruža veliki broj nativnih komponenti kroz svoje sučelje, te ih je

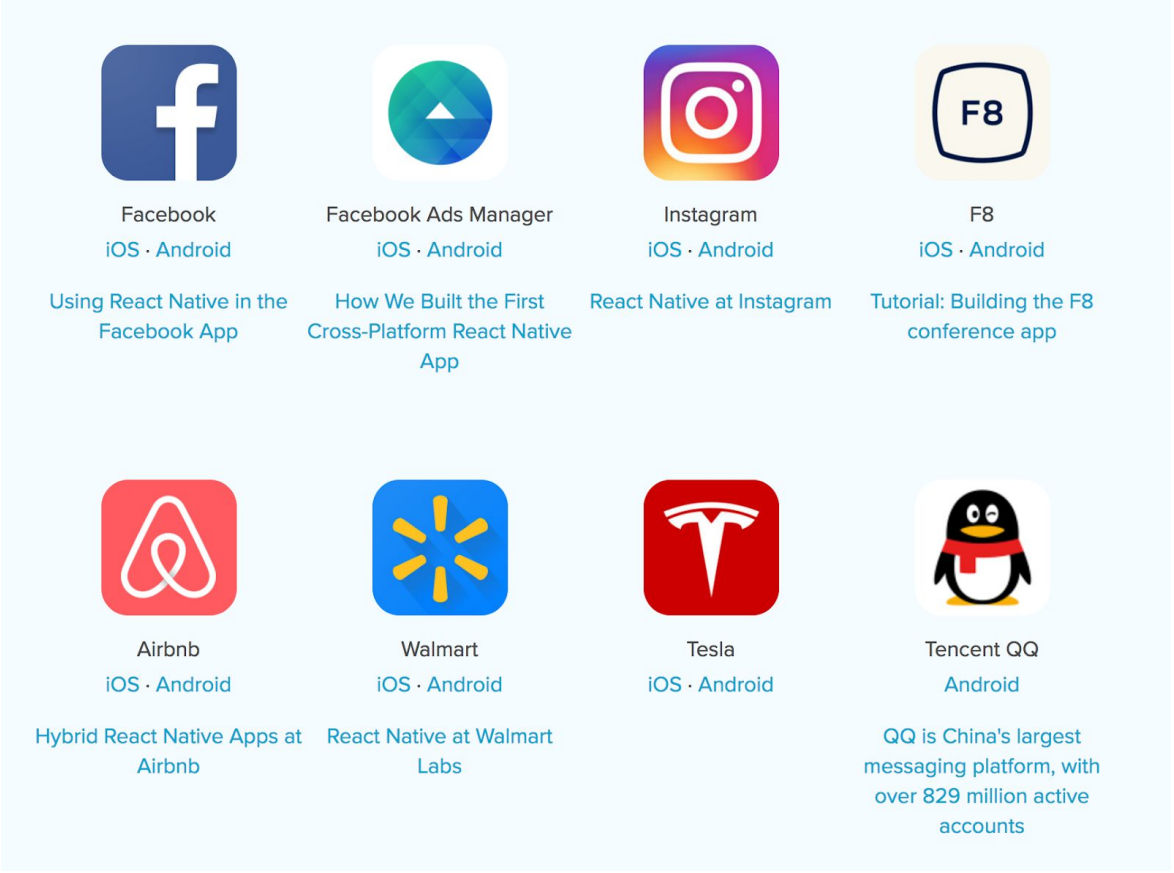
[7] WebView komponenta u nativnim mobilnim aplikacijama se koristi za renderiranje, odnosno prikaz web stranice. Ima istu funkciju kao i osnovni web preglednik.

moguće lako pronaći unutar React Native dokumentacije [14]. Također postoje brojne komponente koje su specifične za određeni mobilni operativni sustav. Takve komponente su navedene u React Native sučelju sa sufiksom “Android” ili “IOS”. Tako se lako može vidjeti koje komponente će biti vidljive samo na Android ili samo na iOS uređajima.

Pošto u React Native aplikaciji sve komponente koje se koriste su ili komponente koje su dostupne kroz samu React Native biblioteku ili su komponente koje je sam razvojni inženjer izradio nema potrebe za .jsx prefixom kao što je to slučaj u React aplikacijama. Zato se React Native projekti sastoje samo od .js datoteka[8].

React Native se može činiti kao dosta novi koncept izrade mobilnih aplikacija, i mnogi bi odustali od takvog načina zbog nedovoljnog broja uspješnih aplikacija u produkciji. No zanimljiva činjenica je da su jedne od najpopularnijih aplikacija izrađene upravo koristeći React Native. Kao primjer su Facebook, AirBnB, Instagram i Tesla aplikacija, što pokazuje samu zrelost biblioteke.

[8] Odnosi se na datoteke koje opisuju komponente u aplikaciji. Naravno svaka aplikacija će uz njih imati i mnogo drugih vrsta datoteka za multimedijske sadržaje ili konfiguraciju projekta.



Slika 1.3 – Popis nekih React Native aplikacija [15]

2 – **Razlike u razvojnom procesu**

2.1 – **Prednosti**

Jedna od najvećih prednosti izrade mobilne aplikacije pomoću React Native biblioteke je vrijeme prevođenja tijekom razvoja same aplikacije. Prilikom izrade aplikacije u jednom od nativnih jezika (Java, Objective-C ili Swift) svaki put je potrebno prevesti kod cijele aplikacije kako bi se ona mogla pokrenuti i vidjeti rezultat. Taj proces može potrajati nekoliko minuta, ovisno o veličini same aplikacije. U React Native aplikaciji to vrijeme je smanjeno na samo nekoliko sekundi jer za razliku od nativnih metoda, React Native prati promjene koje su se dogodile u cijelom projektu, te nakon promjene prevodi samo onaj dio koda koji se promijenio.

Ako uz mobilnu aplikaciju postoji i web aplikacija tada postoji mogućnost za korištenjem već napisanih modula za opis zajedničke logike. Za primjer se može uzeti aplikacija za računanje kamata na štednju. Web i mobilna aplikacija će na vlastiti način imati implementirane forme koje korisnik može ispuniti, no sama logika za računanje kamata se može nalaziti u jednom JavaScript modulu koji se koristi na obje platforme. Tako aplikacije postaju lakše za održavati jer promjenom logike za računanje kamata potrebno je mijenjati samo jedan, umjesto više modula.

Veoma slična prednost se vidi i prilikom razvoja samo mobilne aplikacije, ali za više različitih platformi. Kroz React Native sučelje piše se jedan zajednički kod koji se kasnije prevodi u nativni kod za više mobilnih platformi. Logika je opet zajednička za sve platforme i potrebno je manje angažmana prilikom održavanja same aplikacije. Također izrada aplikacija koristeći React Native je mnogo jeftinija od zasebnog razvoja Android i iOS aplikacije [3].

2.2 – **Nedostaci**

Najveći nedostatak React Native biblioteke je taj što ograničava razvojnog inženjera da koristi samo komponente koje su dostupne kroz biblioteku. Ako se želi dodati komponenta koja koristi neka druga nativna sučelja pojedine mobilne platforme, a ne postoji u React Native biblioteci, potrebno je pisati vlastite nativne komponente što onda unosi Java i Objective-C kod u postojeću React Native aplikaciju. Jedan način kako ublažiti ovaj nedostatak je taj da se sve komponente koje je potrebno pisati u nativnom kodu pišu neovisno o samoj aplikaciji (najbolje izvan same aplikacije, pa se samo učita pojedina komponenta u postojeći projekt). No i dalje je onda potrebno imati razvojne inženjere koji su upoznati s Java, Objective-C i JavaScript programskim jezicima.

Drugi nedostatak je veoma teško pronalaženje grešaka. Trenutno jedini način za zaustavljanjem izvođenja programa na određenoj liniji koda je koristeći alate za nativni razvoj mobilnih aplikacija (Android Studio za Android i XCode za iOS aplikacije). Također ti alati su potrebni ako se želi pratiti koje web zahtjeve je aplikacija uputila prema nekom web poslužitelju.

3 – Sustav za testiranje studenata Edgar

3.1 – O sustavu

Sustav za testiranje studenata Edgar razvijen je na Fakultetu elektrotehnike i računarstva, te se koristi na mnogim kolegijima kako bi se brzo i lako pomoću raznih vrsta pitanja ispitalo znanje studenata o određenoj nastavnoj jedinici. Autentifikacija u sustav se vrši preko AAI@EduHr sustava kako bi se svim studentima olakšao pristup. Središnji dio je Node.js aplikacija koja brine o cijelom sustavu. Dio aplikacije vezan uz samo testiranje studenata, prikaz testa, i slanje odabranih odgovora je Angular aplikacija koje se pokreće u korisnikovom web pregledniku kada korisnik želi pristupiti pisanju određenog testa. Kao baza podataka koristi se Postgresql[9] baza podataka, te za praćenje aktivnosti i sjednice korisnika postoji dodatna MongoDB[10] baza podataka.

Edgar omogućuje kreiranje novih testova i pitanja za određene cjeline na kolegiju, kontrolu korisnika, reviziju riješenih testova i još mnogo drugih funkcionalnosti. U mobilnoj aplikaciji je cilj implementirati sljedeće funkcionalnosti:

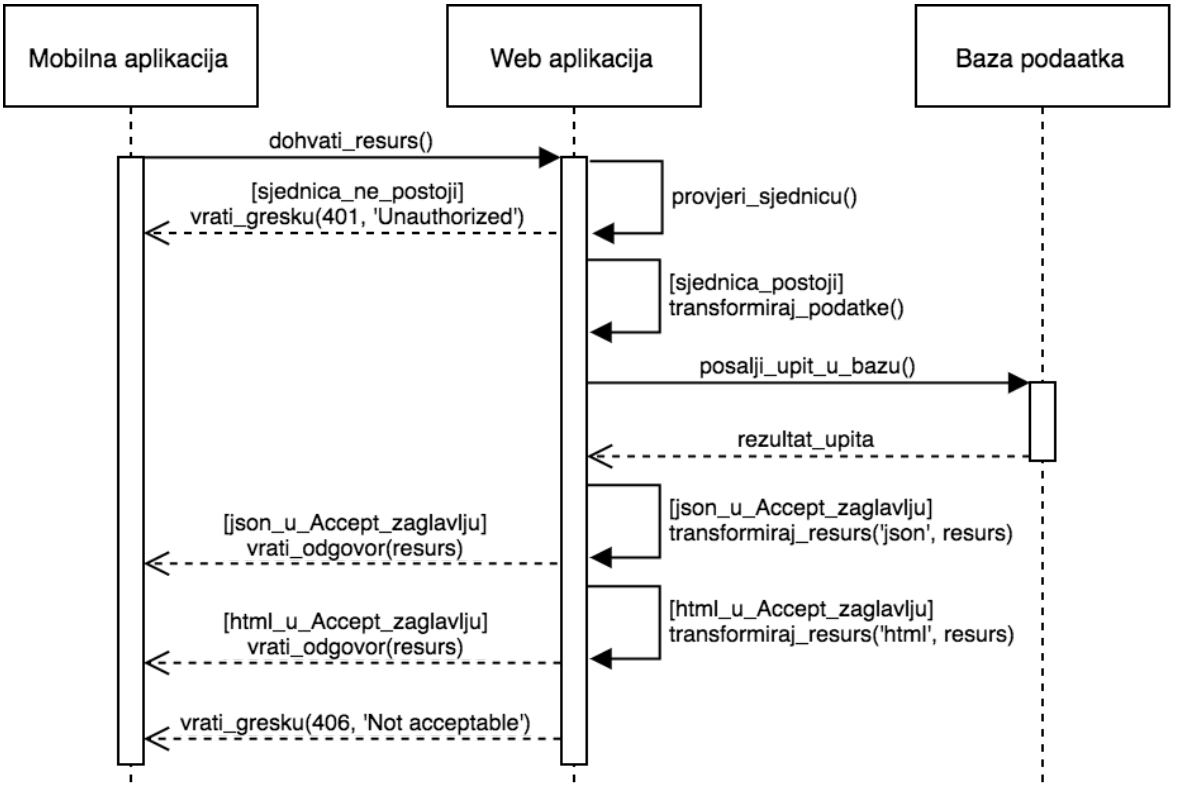
- Autentifikacija korisnika
- Pristupanje određenom ispitu
- Navigacija i odgovaranje na pitanja
- Predaja ispita s odgovorima na ocjenjivanje
- Prikaz ostvarenih bodova
- Prikaz odabranih odgovora s prikazom točnih odgovora

[9] Postgresql baza podataka je jedna od snažnijih SQL baza podataka koja je ujedino i baza podataka otvorenog koda [16]
[10] MongoDB baza podataka je jedna od bržih, skalabilnijih i najčešće korištenih NoSQL baza podataka [17]

3.2 – **Prilagodba za mobilnu aplikaciju**

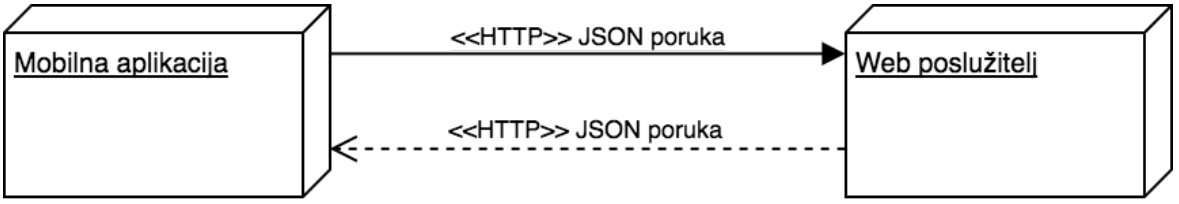
Kako bi se mobilna aplikacija lakše povezala na postojeću arhitekturu potrebno je izmijeniti nekoliko dijelova aplikacije. Najveće promjene su se odnosile na format odgovora s poslužitelja. Nakon uspješne prijave sustav je kao odgovor slao HTML kod koji se onda prikazao u web pregledniku. Ta vrsta odgovora nije standardna za mobilnu aplikaciju koja kao odgovor očekuje JSON format (*JavaScript Object Notation*). Jednostavno rješenje ovog problema je korištenje *Accept* zaglavlja u samome zahtjevu prema poslužitelju. *Accept* zaglavlje se koristi kako bi se poslužitelju indiciralo koje sve formate odgovora sustav koji je poslao zahtjev prihvaća. Ukoliko poslužitelj nije u mogućnosti vratiti odgovor u tom formatu dužan je odgovoriti statusnim kodom 406 (*Not Acceptable*). Web preglednik automatski postavlja *Accept* zaglavlje na vrijednost "*/*" što označava da prihvaća odgovor u bilo kojem formatu. Mobilna aplikacija zato u svakom zahtjevu na poslužitelj postavlja *Accept* zaglavlje na vrijednost "application/json". Poslužitelj na početku svakog zahtjeva prvo provjeri *Accept* zaglavlje, formatira odgovor i šalje natrag pošiljatelju.

Na dijagramu 3.1 je prikazana povezanost između mobilne aplikacije, web aplikacije i baze podataka, te operacije koje se odvijaju tijekom dohvaćanja određenog resursa.



Dijagram 3.1 – Komunikacija između mobilne i web aplikacije

Na dijagramu 3.2 je prikazana povezanost između mobilne aplikacije i web poslužitelja. Lako se uoči kako se sva komunikacija odvija preko HTTP protokola sa sadržajem u JSON obliku.



Dijagram 3.2 – Povezanost mobilne aplikacije i web poslužitelja

Mobilna aplikacija ostvaruje samo određeni podskup funkcionalnosti web aplikacije, pa je potrebno samo određene dijelove koda promijeniti da ovisno o *Accept* zaglavlju šalju različiti format odgovora. Sve krajnje točke na poslužitelju koje su izmijenjene, te detaljniji opisi svih promjena su opisani u poglavlju 6.

Sustav Edgar za verzioniranje i kontrolu koristi Git sustav tako da je za svaku od navedenih promjena bilo potrebno kreirati novu granu i trenutno aktivne grane (master) i tamo pohraniti sve promjene. Sustav Edgar se također nalazi u GitLab sustavu koji omogućuje interaktivno spajanje jedne grane u drugu koristeći *Merge Request* (Zahtjev za spajanjem) funkcionalnost. Tako je moguće vidjeti sve promjene i lakše uočiti potencijalne probleme koji mogu nastati spajanjem jedne grane u drugu.

4 – Struktura mobilne aplikacije

Datotečna struktura mobilne aplikacije koja je izrađena veoma je slična datotečnoj strukturi web aplikacije rađene uz pomoć React biblioteke. Struktura je prikazana u isječku 4.1.

```
edgar-mobile
├ android
│  └ ...
│
├ ios
│  └ ...
│
├ components
│  ├ forms
│  │  └ ...
│  └ ...
│
├ config
│  └ ...
│
├ screens
│  └ ...
│
├ services
│  └ ...
│
├ store
│  └ ...
│
├ styles
│  └ ...
│
├ index.js
├ package.json
├ README.md
└ ...
```

Isječak 4.1 – Prikaz datotečne strukture projekta

4.1 – Prezentacijske i smještajne komponente

Sve komponente koje se koriste u aplikaciji nalaze se u dvije mape: `components/` i `screens/`. Kao što je navedeno u poglavlju 1.2 u aplikaciji bi trebale postojati dvije vrste komponenti: prezentacijske i smještajne komponente. Smještajne komponente je veoma lako prepoznati po tome što se one nalaze u `screens/` mapi. U `components/` mapi imamo dvije vrste komponenti, prezentacijske i navigacijske komponente.

Navigacijske komponente se dosad nisu spominjale jer je njihova jedina funkcija da određuju koja smještajna komponenta se treba u određenom trenutku prikazati na zaslonu korisnika. Ako bi u aplikaciji imali neki napredniji sustav navigacije, ili ako bi koristili neki modul koji se brine o navigaciji unutar aplikacije (*router* modul), tada ne bi imali potrebu za navigacijskim komponentama. Navigacijske komponente koje se koriste su: *App* i *AppNavigation*. Koristeći podatke u skladištu stanja navigacijske komponente prikazuju različite smještajne komponente. Tako je izvedena potpuna navigacija između različitih zaslona.

U programskom kodu 4.2 se nalazi primjer koji prikazuje isječak App komponente, točnije sadržaj *render* metode[11]. Lako se uoči na danom primjeru kako je jedina uloga *App* komponente prikazivanje različitih komponenti ovisno o informacijama u skladištu stanja. Ako je test predan na ocjenjivanje prikazuje se *ReviewScreen* smještajna komponenta, u slučaju da je test spreman za pisanje prikazuje se *TestScreen*, a u suprotnom prikazuje se *MainScreen* smještajna komponenta koja omogućava korisniku početak pisanja testa. Slična logika se nalazi i u *AppNavigation* komponenti koja se brine o prikazu komponenti ovisno o postojanju korisničke sjednice, te kao skladište stanja koristi skladište *sessionStore*, dok *App* komponenta koristi *testStore* skladište.

[11] *render* metoda opisuje izlaz iz komponente

```
if (this.props.testStore.isSubmitted) {
  return (
    <ReviewScreen />
  );
}

if (this.props.testStore.testReady) {
  return (
    <TestScreen />
  );
}

return (
  <MainScreen />
);
```

Programski kod 4.2 – Isječak sadržaja *App* komponente

4.2 – Servisi

Za izradu mnogih aplikacija često se stvara potreba za dodatnim servisima koji olakšavaju proces izrade same aplikacije, te uz to pomažu u smanjivanju pojave identičnog koda na više mjesta u aplikaciji.

Jedini servisi koji se u ovoj aplikaciji koriste su *request* i *logger* servisi, odnosno moduli.

Request servis služi kao spojnik između ove mobilne aplikacije i poslužitelja na kojem se nalazi Edgar web aplikacija s kojega se dohvaćaju podaci. *Request* modul pruža kroz svoje sučelje dvije metode, jednu za slanje GET zahtjeva, te jednu za slanje POST zahtjeva. Također njezina zadaća je da postavi odgovarajuća zaglavlja opisana u poglavlju 3.2.

Logger servis se koristi za slanje dodatnih informacija na web poslužitelj. Logger šalje zahtjeve prilikom važnih akcija u aplikaciji (dohvat ispita, dohvat pitanja, promjena odgovora, ...). Također se koristi za slanje informacije o nazočnosti korisnika (*heartbeat* signal) svake minute.

4.3 – Stilovi

Za razliku od web aplikacija gdje za stiliziranje izgleda aplikacije koristimo CSS u slučaju React Native aplikacije to nije moguće jer se sav kod mora moći prevesti u Java ili Objective-C kod[12]. Zato za stiliziranje React Native komponenti koristimo ***StyleSheet* klasu** koja se nalazi u React Native biblioteci. Primjer uporabe je prikazan u programskom kodu 4.3.

```
const styles = StyleSheet.create({
  loading: {
    flex: 1,
    backgroundColor: colors.bgPrimary,
    flexDirection: 'column',
    alignSelf: 'stretch',
    alignItems: 'center',
    justifyContent: 'center'
  }
});
```

Programski kod 4.3 – Primjer instanciranja StyleSheet klase

Gledajući u navedeni primjer uočljivo je kako je kod veoma sličan CSS kodu, što olakšava stiliziranje ako je poznat rad u CSS programskom jeziku. Najveća razlika je što nije moguće postavljati stilove jedan unutar drugoga, odnosno kaskadirati stilove kao što je moguće u CSS kodu.

Dobra praksa prilikom izrade web aplikacija je da se sve boje koje se koriste referenciraju preko varijabli, te da se vrijednosti tih varijabli postavlja u jednoj datoteci. Slična praksa je primjenjiva i prilikom izrade React Native aplikacije, samo što ovdje imamo jednu JavaScript datoteku koja se nalazi u `styles/` mapi i unutar nje su referencirane sve korištene boje. Sadržaj te datoteke je u programskom kodu 4.4.

Još jedna dobra praksa je da se odvoje varijable koje se koriste za postavljanje boje teksta, pozadina i obruba. Takva organizacija je veoma korisna

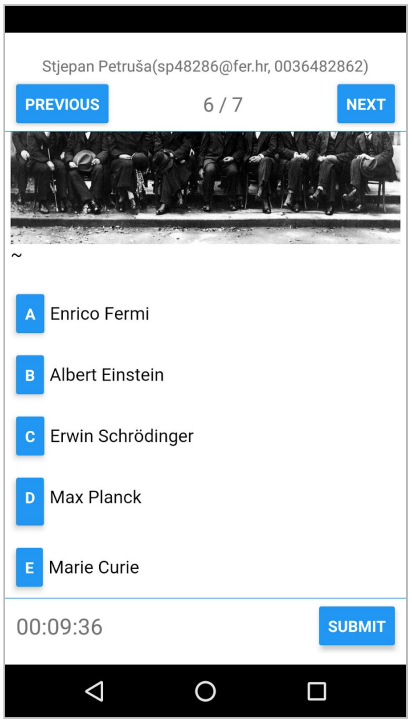
[12] Isti je razlog zašto u strukturiranju komponenti ne možemo koristiti standardne HTML elemente

ako se u jednom trenutku promijeni određena boja koja se koristi u aplikaciji, pa ju je potrebno promijeniti samo na jednom mjestu.

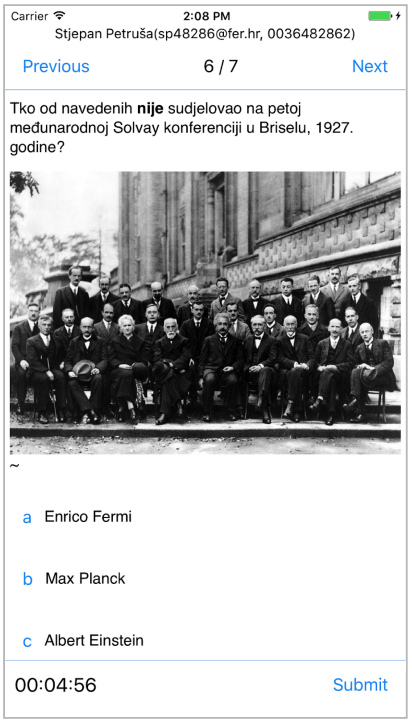
```
const baseWhite = '#FFFFFF';
const baseBlue = '#2196F3';
const baseGreen = '#4CAF50';

export default {
  // BACKGROUND
  bgPrimary: baseBlue,
  bgLight: baseWhite,

  // TEXT
  textPrimary: baseBlue,
  textPrimaryInverted: baseWhite,
  textAffirmative: baseGreen,

  // BORDER
  borderPrimary: baseBlue,
  borderPrimaryInverted: baseWhite
};
```

Programski kod 4.4 – Sadržaj colors.js datoteke

Dosta je važno napomenuti razlike u stilovima nativnih komponenti u pojedinom operacijskom sustavu. Najveću razliku možemo uočiti u *Button* komponenti. Na slikama 4.5 i 4.6 prikazani su zasloni aplikacije za ispunjavanje ispita. Oba prikaza koriste istu konfiguraciju komponenti i stilova. Mnoge komponente unutar react Native biblioteke imaju različite stilove ovisno o platformi kako bi se komponenta izgledom prilagodila ostatku operacijskog sustava.



Slika 4.5 – Prikaz ispita na Android uređaju



Slika 4.6 – Prikaz ispita na iOS uređaju

4.4 – Upravljanje modulima

Izrada jedne kompleksnije aplikacije nije moguća bez korištenja raznih modula koji pomažu u razvoju cijele aplikacije. Korištenjem modula može se ubrzati cijeli razvojni proces. Kako bi se olakšalo praćenje verzija instaliranih modula koriste se razni alati za njihovo upravljanje. Najpopularniji alat za upravljanje modulima pisanih u JavaScript programskom jeziku je **npm** (*Node Package Manager*). On u svome registru sadrži više od 450 000 modula [7] koji su besplatni za korištenje.

Umjesto npm-a u ovom projektu koristit će se **Yarn** alat. Yarn je veoma sličan npm alatu i koristi isti registar kao npm, no na drugačiji način pohranjuje module u projektu. Također Yarn generira i posebnu datoteku s nazivom yarn.lock u kojoj se nalaze informacije o verziji svakog modula i podmodula koji se koristi,

tako da svi razvoji inženjeri koji rade na istom projektu imaju instaliranu istu verziju.

Autori React Native modula su i autori Yarn alata, pa se zato uz React Native predlaže i korištenje Yarn alata za upravljanjem svih modula.

Još jedna dosta značajna datoteka kada se spominje upravljanje modulima je i **package.json**. U toj datoteci se nalazi popis svih potrebnih modula za rad s aplikacijom (*dependencies* atribut). Postoji još i popis svih modula koji se koriste za pomoć tijekom razvojnog procesa i oni se mogu pronaći pod *devDependencies* atributom.

Jedan od najkorisnijih dijelova package.json datoteke je i *scripts* dio u kojem se nalaze korisne funkcije za pokretanje ili testiranje same aplikacije. U ovoj aplikaciji veoma su korisne skripte za pokretanje aplikacije na različitim mobilnim platformama.

5 – **Upravljanje stanjem aplikacije**

5.1 – **Reaktivnost**

Kako bi se shvatila potreba za kvalitetnim upravljanjem stanjem u ovoj aplikaciji, potrebno je objasniti svojstvo reaktivnosti React Native biblioteke. Ovo svojstvo nije specifično samo za React Native, nego se koristi i u mnogim drugim bibliotekama kao što su React i Vue.js. Uloga, odnosno zadatak jedne takve biblioteke je da generira HTML kod u slučaju web stranice, ili nativan kod za mobilnu platformu u slučaju rada s React Native bibliotekom. Te biblioteke znaju kako prikazati komponente koje se koriste u samoj aplikaciji i veoma su dobre u tome. Ovdje se lako primjećuje kako su sve takve biblioteke rađene po pravilu "*Do only one thing and do it good*" ("Radi samo jednu stvar i radi su dobro") što zaključujemo gledajući kratko vrijeme prevođenja.

Pojedine komponente imaju ograničenu domenu primjene, te bi se trebale koristiti kao gradivni materijal aplikacije. Kako bi se to ostvarilo komponente moraju biti veoma fleksibilne[13]. Fleksibilnost mogu ostvariti tako da se veoma malo logike nalazi u samoj komponenti, a većinu podataka koji su joj potrebni dobiva preko ulaznih parametara[14].

Kako komponente mogu primati veliki broj parametara komponenta se mora brinuti o tome da promijeni svoj izlaz kada se promijeni i neki od ulaznih parametara. Svojstvo da se statička komponenta odazove na određenu promjenu ulaznih parametara nazivamo **reaktivnost**.

5.2 – **Skladišta stanja**

Kako većina komponenti sve podatke koje im trebaju primaju kao ulazni podatak tada moramo negdje spremati sve te podatke kako bi ih mogli koristiti na više mjesta u aplikaciji. Za spremanje podataka koriste se **skladišta stanja** (*state*

[13] Najviše se odnosi na prezentacijsku vrstu komponenti opisane u poglavlju 1.2
[14] Ulazni podaci se komponentama predaju kao atributi u HTML kodu.

store). Aplikacije najčešće imaju samo jedno skladište stanja koje se onda može sačinjavati od više manjih dijelova, točnije modula. Svaki dio se brine za odvojeni dio aplikacije i u pravilu oni bi trebali biti neovisni jedni o drugome. Postoje različite biblioteke koje rade apstrakciju skladišta kako bi se olakšalo njihovo korištenje. Dvije trenutno popularnije biblioteke su Redux i Mobx čije će korištenje biti objašnjeno u poglavljima 5.3 i 5.4.

Često se nameće pitanje o tome koji sve podaci bi se trebali nalaziti u skladištu stanja, a koje podatke treba komponenta sadržavati u vlastitom internom stanju. Najbolja organizacija je takva da svi podaci koji ovise o trenutnom prikazu, točnije izgledu aplikacije trebaju biti u skladištu. To na primjer može uključivati sljedeće informacije:

- Elementi i aktivan element u navigacijskoj traci
- Lista elemenata koji se prikazuju
- Prikaz ekrana za učitavanje podataka
- Podaci unutar tablice
- …

Dok u internom skladištu stanja komponente bi se trebale nalaziti informacije o sadržaju polja za unos koji se nalazi u toj komponenti ili podaci koji su privremeni i ne bi trebali biti prikazani prilikom ponovnog prikazivanja iste komponente.

Kako su sve komponente reaktivne, određena promjena podataka unutar skladišta stanja dovest će do promjene izlaza komponente.

5.3 – **Redux**

Redux biblioteka je odličan primjer za skladište stanja pošto je veoma ekspresivna i ima točno određenu formu. Cijelo skladište stanja sastoji se od više povezanih dijelova, te ima dosta restrikcija kojih se treba pridržavati.

Jedna od najvažnijih restrikcija je ta da može postojati samo jedno skladište stanja, te se ono ne smije nikada modificirati. Naravno moguće je koristeći pomoćne funkcije napraviti strukturu koja se sastoji od više skladišta, no i dalje će

sve te funkcije u konačnici proizvesti samo jedno skladište stanja. Razlog zbog kojega nije dozvoljeno modificirati podatke u trenutnom stanju je taj da biblioteka lakše može odrediti koje točno vrijednosti treba promijeniti i tako smanjiti broj komponenti koje će reagirati na promjenu podataka.

Svaka promjena koja se želi pohraniti u skladište mora proći kroz **reduktor funkciju** (*reducer*). Svaki reduktor kao parametre prima trenutno stanje i akciju koja je pokrenula taj reduktor, i vraća novo stanje i pritom ne modificira ulazne podatke. To se najčešće izvršava tako da se napravi kopija trenutnog stanja i ono se zatim modificira i vraća kao izlaz iz reduktora. Biblioteka će koristiti izlaz reduktor funkcije za analizu promjena u skladištu stanja. Također sve reduktor funkcije moraju biti čiste (*pure*) što znači da osim što ne smiju modificirati ulazne parametre, ne smiju vršiti ikakve asinkrone akcije. Ako se asinkrono mijenja vrijednost stanja biblioteka neće moći reagirati na promjenu. Primjer jedne reduktor funkcije prikazan je u programskom kodu 5.1

```
function todoApp(state = initialState, action) {
  switch (action.type) {
    case 'SET_VISIBILITY_FILTER':
      return Object.assign({}, state, {
        visibilityFilter: action.filter
      });
    default:
      return state
  }
}
```

Programski kod 5.1 – Primjer jednostavne reduktor funkcije

U većini aplikacija najčešće će postojati samo jedna reduktor funkcija koja će ovisno o akciji koja ju je pozvala vratiti drugačiji rezultat[15]. Ako se reduktor pokrene s akcijom na koju ne zna kako da odgovori, reduktor je dužan vratiti stanje koje je primio kao parametar, ili ako to stanje nije predano, dužan je vratiti inicijalno stanje aplikacije.

[15] Ukoliko se koriste pomoćne funkcije reduktor se može organizirati tako da se sastoji od više međusobno povezanih dijelova.

Primjer jednog poziva reduktor funkcije prikazan je u programskom kodu 5.2

```
store.dispatch({type: 'SET_VISIBILITY_FILTER', filter: '42'});
```

Programski kod 5.2 – Primjer poziva reduktor funkcije

Ako je potrebno izvršiti asinkronu akciju tada najčešće ta akcija pokreće reduktor više puta. Kao primjer može se prikazati dohvat podataka s poslužitelja. U tom slučaju će biti potrebno pozvati reduktor prije početka zahtjeva kako bi se stanje promijenilo tako da je spremno prihvatiti nove podatke, te će biti potrebno pozvati reduktor funkciju još prilikom uspješne dohvate podataka s podacima kao jedan od parametara i kod neuspjele dohvate podataka s informacijom o grešci.

Restrikcije koje Redux biblioteka postavlja stvara veliku količinu koda koji se duplicira ili je veoma sličan, te tako postaje veoma teško pratiti sve akcije koje mogu pokrenuti reduktor funkciju. Redux je zato pogodno rješenje za manje i jednostavnije aplikacije, ili aplikacije s veoma dobrom organizacijom skladišta stanja.

5.4 – **Mobx**

Mobx je također jedna od biblioteka za upravljanje stanjem aplikacije, no ona taj zadatak rješava na potpuno drugačiji način za razliku od Redux biblioteke. Mobx biblioteka je mnogo fleksibilnija i uz puno manju količinu koda može se dobiti jednak rezultat kao što bi se moglo koristeći Redux biblioteku.

Mobx biblioteka dozvoljava korištenje većeg broja skladišta podataka koji su međusobno neovisni, te nema toliki broj restrikcija kao što je to slučaj s Redux bibliotekom. Skladište podataka čini jedna klasa koja se instancira samo jednom (*singleton*) i sadrži sve podatke koji bi se trebali nalaziti u skladištu zajedno s metodama (koje mogu biti i asinkrone) koje izravno mijenjaju stanje skladišta.

Kako bi se objasnio rad Mobx biblioteke potrebno je prvo objasniti pojam dekoratera u JavaScript programskom jeziku. **Dekorateri** su funkcije sa

specijaliziranom namjenom. Koriste se za modificiranje ili za dodavanje dodatnih informacija na klase, njezine atribute ili metode. Dekorateri su trenutno u drugoj fazi postupka uvrštenja novih funkcionalnosti u JavaScript standard. Dekoratere je jednostavno prepoznati po znaku "`@`" ispred imena dekoratera, a primjer korištenja se nalazi u programskom kodu 5.3. Naravno korištenje dekoratera nije obavezno pošto se ne nalaze u standardu JavaScript programskog jezika. Kao zamjena moguće je korištenje funkcijskog oblika dekoratera.

```
import {observable} from 'mobx';

class OrderLine {
    @observable price = 0;
    @observable amount = 1;

    @computed get total() {
        return this.price * this.amount;
    }
}
```

Programski kod 5.3 – Primjer korištenja dekoratora [4]

Nekoliko važnih dekoratera se koristi kada se radi s Mobx bibliotekom:

- **@observable** – postavlja se ispred atributa u skladištu čija vrijednost se može u nekom trenutku promijeniti i tako promijeniti stanje aplikacije. (`OrderLine.price` i `OrderLine.amount` u primjeru 5.3)
- **@computed** – predstavlja vrijednosti koje nemaju izravnu vrijednost, ali se mogu izračunati korištenjem drugih vrijednosti iz iste klase skladišta. (`OrderLine.total` u primjeru 5.3)
- **@action** – opisuje metodu unutar klase skladišta koja mijenja stanje aplikacije (na sinkroni ili asinkroni način)
- **@observer** – koristi se nad komponentama koje trebaju reagirati na promjene unutar navedenog skladišta

Kako Mobx biblioteka radi jednako dobro sa sinkronim i asinkronim akcijama, te je jednostavnija za korištenje, tijekom izrade Edgar mobilne aplikacije Redux skladište stanja je zamijenjeno s Mobx skladištem stanja. Tako je

programski kod postao čišći uz manje ponavljanja, te jednostavniji za pronalazak grešaka.

Kako bi se omogućilo korištenje dekoratera potrebno je dodati i *transform-decorators-legacy* dodatak za Babel modul [6]. Funkcija Babel modula je da prevodi JavaScript kod pisan u novijem standardu JavaScript jezika u kod pisan u nekom starijem standardu[16].

[16] U ovom slučaju stariji standard se odnosi na standard na kojem je zasnovan node.js prevoditelj na računalu.

6 – **Zasloni i funkcionalnosti**

Premda mobilna aplikacija pruža samo određeni podskup funkcionalnosti koje pruža web-aplikacija, mobilna aplikacija je dovoljno kompleksna da se funkcionalnosti mogu podijeliti na četiri veće cjeline. Svaka cjelina je opisana u svome poglavlju koji uključuju prikaze zaslona aplikacije, opis promjena koje su bile potrebne obaviti unutar web-aplikacije i detaljniji opis implementiranih funkcionalnosti.

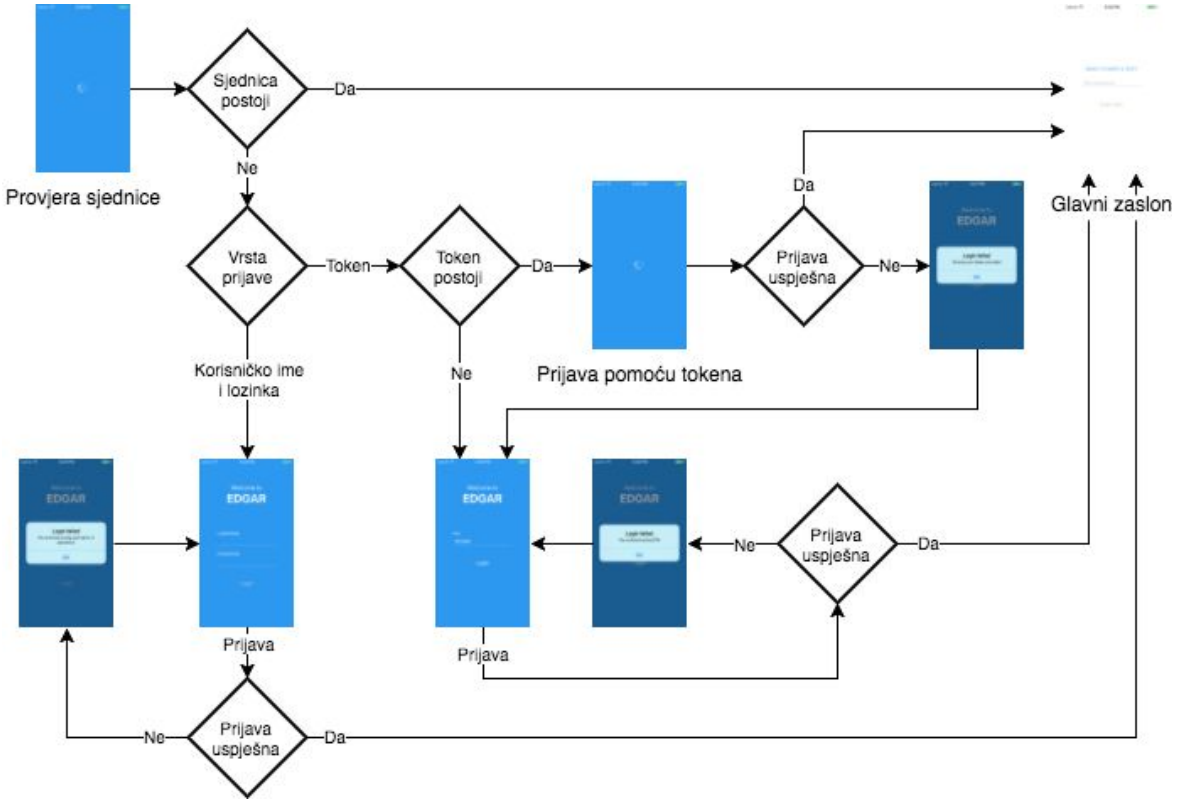
6.1 – **Autentifikacija**

Autentifikacija u mobilnoj aplikaciji se planirala vršiti pomoću jedinstvenog PIN broja i odgovarajućeg token ključa. Kako bi se to ostvarilo bilo je potrebno modificirati logiku pristupnih točaka na postojećoj web-aplikaciji. Kako je to dosta veliki zahvat prvo se implementirala prijava pomoću korisničkog imena i lozinke, a kasnije dodala mogućnost prijave pomoću PIN broja.

Sva logika i funkcionalnosti koje su obuhvaćene kroz autentifikaciju prikazani su na dijagramu 6.1. Navedeni dijagram opisuje sva stanja aplikacije u slučaju kada korisnik nije prijavljen u sustav i sve greške koje se mogu dogoditi.

Kao što je objašnjeno u poglavlju 5.2, sve informacije o stanju aplikacije nalaze se u odvojenom skladištu stanja pod nazivom *SessionStore*. Navedeno skladište stanja pohranjuje informaciju o trenutno aktivnom korisniku. Uz to posjeduje i metode za prijavu koji se također koriste u ovom dijelu aplikacije.

Promjena vrste prijave se ne može mijenjati unutar aplikacije, odnosno moguća je prijava samo pomoću PIN/token kombinacije ili pomoću kombinacije korisničkog imena i lozinke. U konfiguracijskoj datoteci nalazi se zastavica koja mijenja vrstu prijave, te se ona mora postaviti prije nego se izgradi nova verzija aplikacije. Tako se smanjuje prostor za pogreškom i zlouporabom korisničkih podataka prilikom prijave.

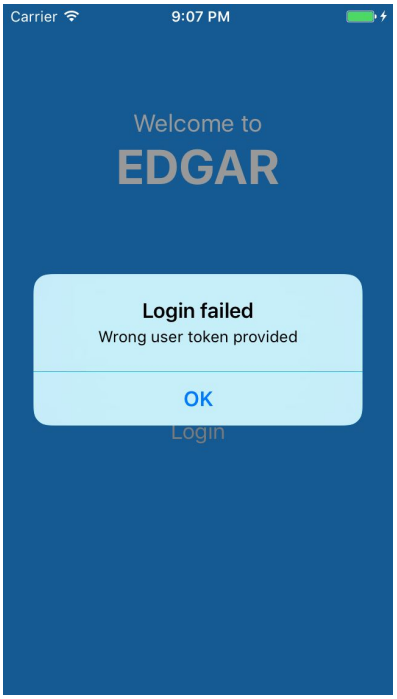


Dijagram 6.1 – Stanja aplikacije tijekom autentifikacije

6.1.1 – **Provjera postojanja korisničke sjednice**

Prilikom pokretanja aplikacije, potrebno je provjeriti postoji li već aktivna korisnička sjednica. Kako bi to bilo ostvarivo potrebno je dodati pristupnu točku na web poslužitelju koja će vraćati informacije o trenutno prijavljenom korisniku. Ostvarena pristupna točka nalazi se na adresi ***/auth/profile***. Ako korisnička sjednica ne postoji kao odgovor vraća se statusni kod 401 *Unauthorized*.

Ako je tijekom izrade verzije aplikacije bila uključena prijava pomoću PIN/token kombinacije aplikacija će dodatno u svoje interno skladište pogledati postoji li informacija o token ključu. Ako postoji, taj token ključ će biti korišten za ponovnu prijavu. Prijava pomoću token ključa je također dodana kao nova pristupna točka na web poslužitelj koja se nalazi na adresi ***/auth/token-login***. Navedena pristupna točka kao parametar prima token ključ i izvršava autentifikaciju korisnika ovisno o primljenom token ključu. U slučaju da token više ne vrijedi prikazuje se greška sa slike 6.2.

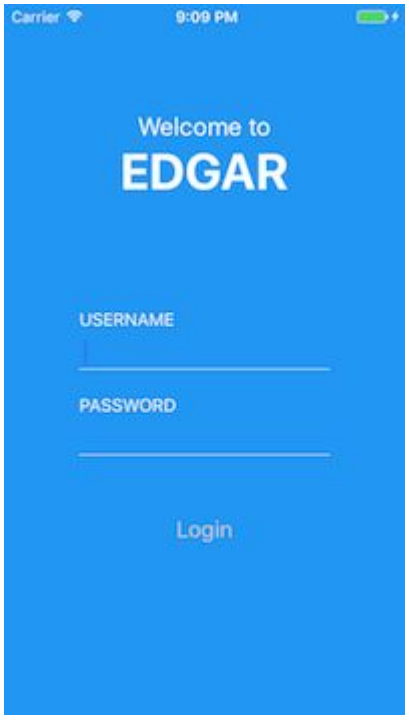


Slika 6.2 – Prikaz greške prilikom prijave koristeći token ključ

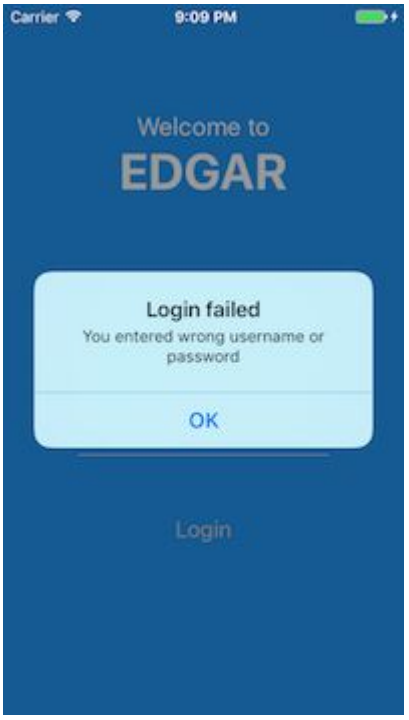
6.1.2 – **Autentifikacija pomoću korisničkog imena i lozinke**

Prvotna verzija autentifikacije je bila izrađena tako da podržava prijavu pomoću korisničkog imena i lozinke. Ova vrsta prijave se prva ostvarila jer na web poslužitelju nije bilo potrebno mijenjati nikakvu logiku osim vrste odgovora opisano u poglavlju 3.2. Ova vrsta prijave je dosta jednostavna i od korisnika očekuje samo unos korisničkog imena i lozinke isto kao što se radi i u web-aplikaciji. web-aplikacija zatim vrši upite prema AAI@EduHr sustavu i ostvaruje autentifikaciju korisnika.

Izgled zaslona za prijavu je prikazan na slici 6.3. Ako dođe do greške prilikom prijave prikazuje se odgovarajuća poruka prikazana na slici 6.4.



Slika 6.3 – Prijava pomoću korisničkog imena i lozinke



Slika 6.4 – Greška prilikom prijave

6.1.3 – **Autentifikacija pomoću jednokratnog PIN broja**

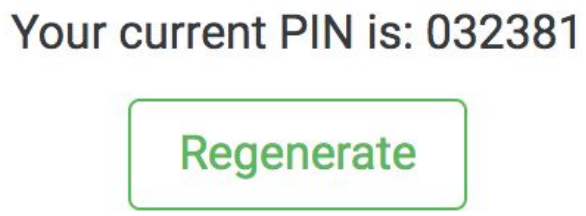
Autentifikacija pomoću PIN broja i token ključa je kompliciranija i zahtjeva mnogo više promjena na web poslužitelju nego autentifikacija koristeći korisničko ime i lozinku, te se zato ova vrsta autentifikacije implementirala tek kasnije.

Sustav autentifikacije je dosta jednostavan, iako zahtjeva dosta promjena kako bi se ostvarile sve funkcionalnosti. Unutar baze podataka na web poslužitelju potrebno je dodati novu tablicu podataka koja će sadržavati informacije o studentu, vremenu kreiranja token ključa, vremenu do kada generirani PIN broj vrijedi, token ključu, PIN broju, te informacija je li navedeni token ključ aktiviran.

Token ključ se generira kao jedinstveni ključ od 50 znakova. Tako dva studenta neće imati isti token ključ, a duljina od 50 znakova omogućava dodatnu zaštitu od zlouporabe. PIN broj se slično generira kao niz od 6 numeričkih znakova. Vrijeme do kada PIN broj vrijedi je konfigurabilno i postavljeno na razdoblje od 1 sat nakon što se taj PIN broj generira. Tako povećavamo sigurnost

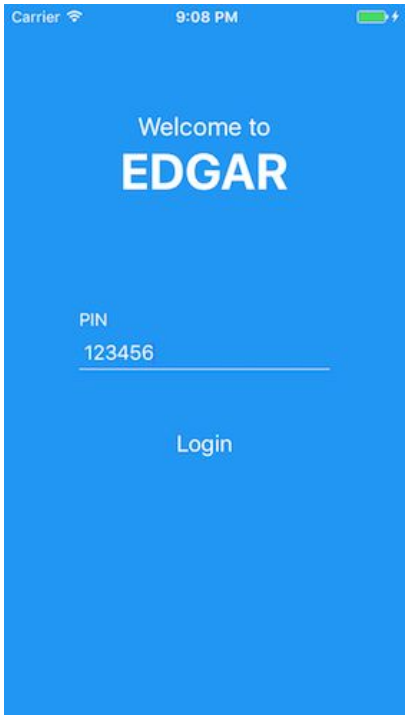
aplikacije jer svi neiskorišteni PIN brojevi nakon sat vremena postaju nevažeći i ne mogu se zlouporabiti.

Kako bi korisnici mogli saznati svoj PIN broj potrebno je i u web aplikaciji dodati novu stranicu s informacijom o trenutnom PIN broju, te mogućnosti za ponovnim generiranjem drugog PIN broja. Nova stranica je dodana na adresi ***/token/view***. Stranici se može lako pristupiti klikom na *Preferences* padajući izbornik i odabirom *App Token* opcije. Isječak izgleda stranice je prikazan na slici 6.5. Navedenoj stranici mogu pristupiti samo prijavljeni korisnici.

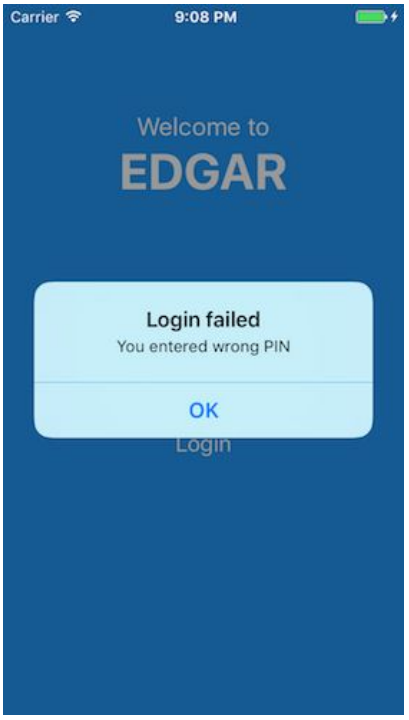


Slika 6.5 – Isječak stranice za generiranje PIN broja

Nakon što se generira novi PIN broj, on se može iskoristiti unutar sljedećih sat vremena za prijavu u mobilnu aplikaciju. Izgled prijave kroz mobilnu aplikaciju prikazan je na slici 6.6. Ako je PIN broj koji se unese netočan prikazuje se greška sa slike 6.7.



Slika 6.6 – Prijava koristeći PIN broj



Slika 6.7 – Greška prilikom prijave

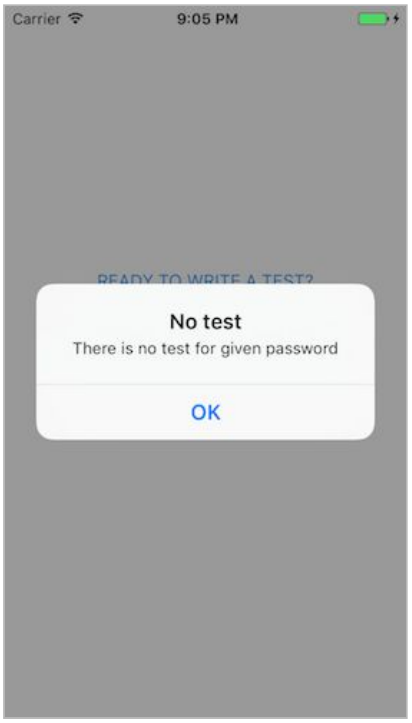
Prijava pomoću PIN broja se izvršava u dva dijela. U prvom dijelu se za upisani PIN broj dohvaća odgovarajući token ključ. Pristupnu točku za dohvatom token ključa za odgovarajući PIN je također potrebno implementirati. Pristupna točka se nalazi na adresi ***/auth/token-register*** i ona provjerava u bazi podataka postoji li redak s primljenim PIN brojem kojemu nije isteklo trajanje. Ako postoji takav token ključ, on se vraća kao odgovor, dok u suprotnoj situaciji poslužitelj odgovara statusnim kodom 404 *Not Found* (Slika 6.7). Ako je razmjena PIN broja i token ključa uspješno obavljena mobilna aplikacija sprema dobiveni token ključ u interno spremište podataka i šalje još jedan zahtjev na poslužitelj za prijavu pomoću dobivenog token ključa na adresu ***/auth/token-login*** (Ista adresa se koristi i za prijavu kod pokretanja aplikacije opisano u poglavlju 6.1.1).

6.2 – **Zahtjev za pisanjem ispita**

Nakon uspješne prijave prikazuje se zaslon s kojega je moguće pokrenuti pisanje ispita. Ekran je veoma sličan ekranu za pisanje ispita u web aplikaciji, te

od korisnika traži jedino lozinku ispita. Izgled ekrana je prikazan na slici 6.8. Ako za unesenu lozinku ne postoji ispit prikazuje se greška sa slike 6.9.

Slika 6.8 – Početni zaslon za pisanje ispita

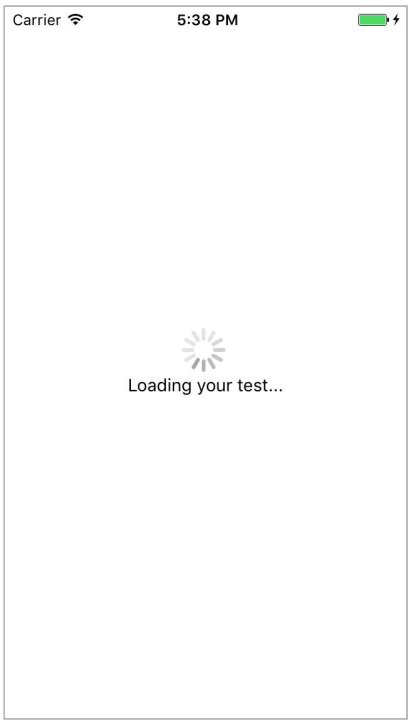


Slika 6.9 – Greška prilikom dohvaćanja instance ispita

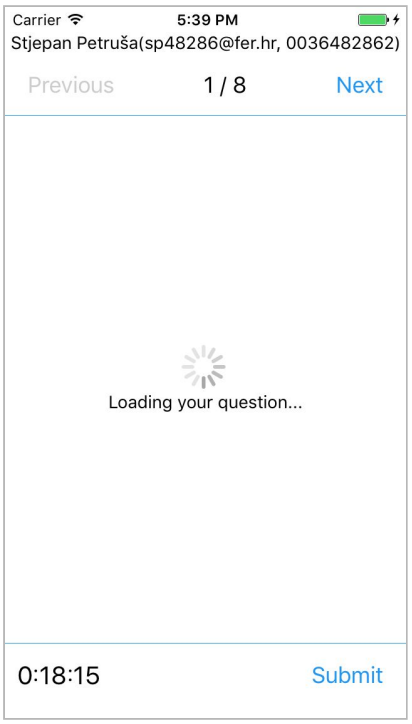
Na poslužitelju osim formata odgovora nije bilo potrebno mijenjati logiku. Kao odgovor za novi ispit se vraća ID nove ili već postojeće instance ispita za koji se kasnije dohvaćaju pitanja. ID instance ispita se također sprema unutar *testStore* skladišta stanja.

6.3 – **Pisanje i predaja ispita**

Nakon dohvata ID vrijednosti instance ispita mobilna aplikacija dohvaća podatke o ispitu (Slika 6.10). Za dohvat podataka koristi se pristupna točka na adresi ***/test/instance***. Svi podaci se spremaju u testStore skladište podataka, te se ti podaci koriste za prikaz ispita. Nakon dohvata ispita dohvaća se prvo pitanje (Slika 6.11) koje se zatim prikazuje korisniku (Slika 6.12).



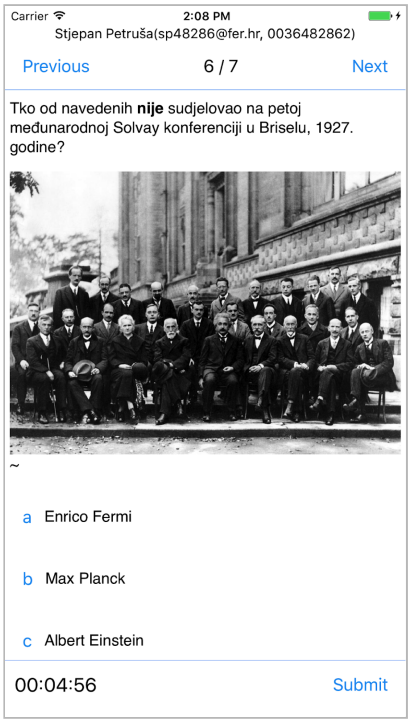
Slika 6.10 – Učitavanje ispita



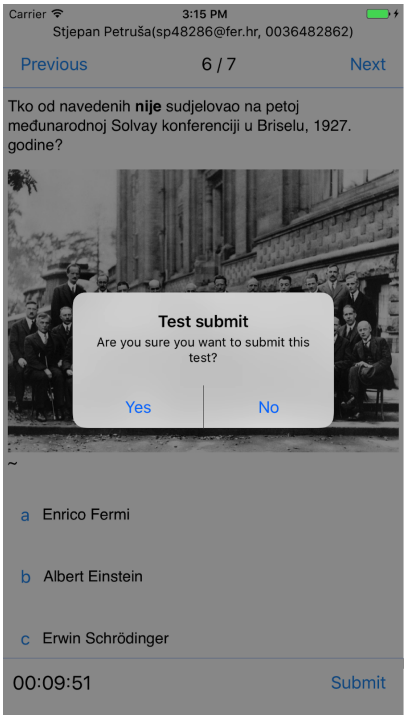
Slika 6.11 – Učitavanje trenutnog pitanja

Zaslon ispita se sastoji od više dijelova:

- Zaglavlje – sadrži informacije o studentu koji piše trenutni ispit, oznaku trenutnog pitanja i ukupnog broja pitanja. Sa svake strane se dodatno nalaze i gumbi za navigaciju između pitanja.
- Središnji dio – sadrži informacije o trenutnom pitanju zajedno s odgovorima
- Podnožje – sadrži informaciju o vremenu preostalom za pisanje ispita i gumb za predaju ispita.



Slika 6.12 – Prikaz pitanja ispita



Slika 6.13 – Upit o predaji ispita

Pritiskom na jedan od navigacijskih linkova u zaglavlju mijenja se broj trenutno aktivnog pitanja. Ako se informacije o pitanju ne nalazi u *testStorage* skladištu stanja, aplikacija pokreće dohvat pitanja s web poslužitelja (Slika 6.11). U suprotnom slučaju se pitanje odmah prikaže bez ponovnog učitavanja (Slika 6.12).

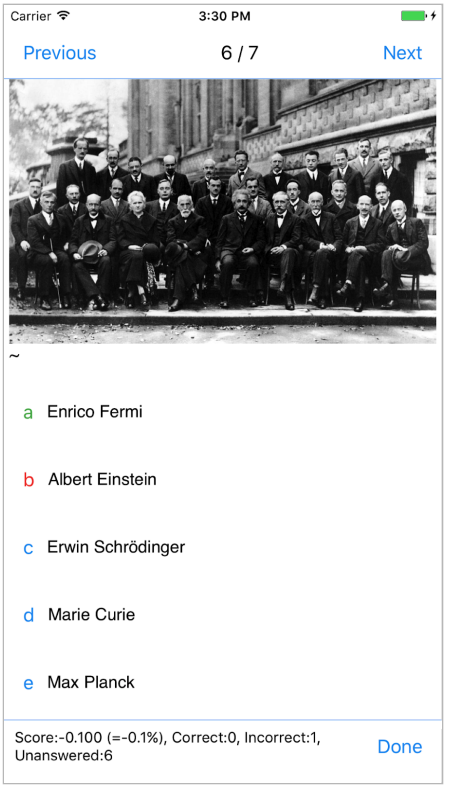
Mobilna aplikacija trenutno podržava samo pitanja s ponuđenim odgovorima. Tekst pitanja i odgovori su u HTML formatu, te je zbog toga za tekst pitanja i odgovora potrebno prikazati WebView komponentu kojoj se preda HTML kod kojeg treba prikazati zajedno s CSS kodom kako bi svi stilovi bili ujednačeni. Također kako bi se slike ispravno prikazivale potrebno je dodati i *baseUrl* parametar pošto su svi linkovi na slike u dobivenom kodu relativnog oblika. *BaseUrl* parametar se nalazi u konfiguracijskoj datoteci pa ga je lako mijenjati.

Odabir odgovora se vrši klikom na slovo ispred svakog odgovora. Odabran odgovor je označen zelenom bojom.

Ispit se može predati na ocjenjivanje pritiskom na *Submit* gumb u podnožju zaslona. Aplikacija će još jednom tražiti potvrdu slanja ispita na ocjenjivanje (Slika 6.13).

6.4 – **Pregled rezultata ispita**

Nakon što se ispit preda na ocjenjivanje moguće je vidjeti rezultat ispita, te točne odgovore za svako pitanje. Cjelokupna logika vezana uz pregled rezultata ispita je veoma slična logici za pisanje ispita.



Slika 6.14 – Prikaz rezultata ispita

Vizualno sve osim podnožja je identično zaslonu za pisanje ispita. U podnožju pregleda rezultata se nalazi informacija o broju bodova ostvarenom na ispitu zajedno s brojem točno i netočno odgovorenih pitanja i brojem neodgovorenih pitanja. Pritiskom na gumb *Done* završava se pregled rezultata ispita i korisnika se vraća na zaslon za ponovno pokretanje ispita (Slika 6.8).

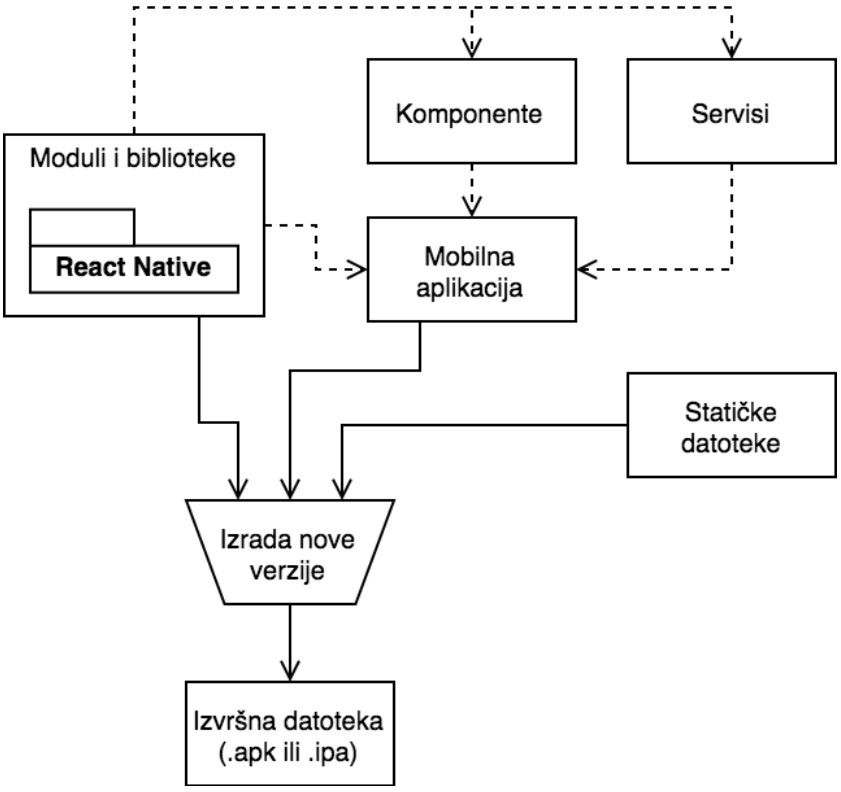
Na slici 6.14 može se vidjeti prikaz ekrana za pregled rezultata ispita. Točan odgovor je označen zelenom bojom, a korisnikov odgovor, ako je različit od točnog odgovora, crvenom bojom.

Kako dio aplikacije za pisanje ispita koristi *testStore* skladište stanja za pohranu stanja aplikacije, tako i dio za pregled rezultata ispita ima skladište stanja *reviewStore* koje sadrži informacije o rezultatima ispita.

Za pisanje ispita i za pregled rezultata ispita nije bilo potrebno ništa mijenjati unutar web aplikacije jer su sve pristupne točke bile prilagođene JSON načinu odgovora zbog korištenja dodatne Angular aplikacije.

7 – Objavljivanje aplikacije

Izdavanje ili objavljivanje nove verzije aplikacije je dosta slično standardnom načinu izgradnje nove verzije aplikacije. Na dijagramu 7.1 su prikazani dijelovi koji se koriste za izradu nove verzije aplikacije. Alat koji ovisi o platformi za koju se izrađuje nova verzija uzima sav aplikacijski kod (koji koristi servise i komponente koje smo napisali), te sve dodatne module i biblioteke koji se koriste u aplikaciji zajedno s React Native bibliotekom i transformira ih u izvršnu datoteku (u .apk ili .ipa formatu). Također se u proces izrade mogu i uključiti statičke datoteka kao što su slike, ikone, fontovi i ostali materijali potrebni u aplikaciji.



Dijagram 7.1 – Postupak izrade nove verzije aplikacije

7.1 – **Android**

Za izradu Android verzije aplikacije potrebno je imati instaliran *Android Studio* program pomoću kojega se mogu instalirati svi potrebni alati i verzije SDK paketa. Izrada nove verzije je veoma jednostavna i može se ostaviti koristeći *gradle* alat. Naredba za pokretanje izrade nove verzije je: `./gradlew assembleRelease`, te ju je potrebno pokrenuti iz `android/` mape. Nakon završetka instalacijska datoteka će se nalaziti na navedenoj adresi: `android/app/build/outputs/apk/app-release.apk`.

7.2 – **iOS**

Za izradu iOS verzije aplikacije potrebno je imati instaliran *XCode* program koji je moguće instalirati samo na računalima s macOS (OSX) operativnim sustavom. Također je za izradu potreban *Apple Developer* korisnički račun [19]. Pomoću *XCode* programa potrebno je otvoriti `ios/` mapu unutar ovog projekta, te pokrenuti izgradnju nove verzije aplikacije. Za testiranje aplikacije može se koristiti *testFlight* sustav [20] pomoću kojega je moguće samo određenim korisnicima omogućiti korištenje testne verzije aplikacije.

Zaključak

Kroz ovaj rad prikazan je rad jednog sustava za testiranje studenata napisan kao Node.js aplikacija. Prikazana je jednostavnost prilagodbe postojeće kompleksne aplikacije za rad s mobilnom aplikacijom, te je opisan uvedena nova funkcionalnost prijave pomoću PIN broja i token ključa.

Izrađena je mobilna aplikacija koja je u potpunosti napisana u programskim jezicima koji su namijenjeni radu na web stranicama. Prikazane su prednosti i fleksibilnost, ali i ograničenja jednog takvog sustava.

Kroz mobilnu aplikaciju implementirano je nekoliko važnih funkcionalnosti koje nudi web aplikacija, a neke od najvažnijih su:

- Mogućnost jednostavne prijave pomoću PIN broja
- Pisanje ispita s pitanjima s ponuđenim odgovorima
- Predaja ispita na ocjenjivanje
- Pregled rezultata ispita

React Native je odlična biblioteka za izradu mobilnih aplikacija ako postoji predznanje JavaScript programskog jezika i poznavanje paradigme web platforme. Znanje nativnih programskih jezika, Java i Objective-C, će također pomoću u razvoju cjelokupne aplikacije. Budućnost React Native biblioteke je dosta svijetla i može ubrzati razvoj manjih i jednostavnijih aplikacija. Za kompleksnije aplikacije potrebno je prvo dobro proučiti sve vrline i mane razvoja koristeći React Native biblioteku. Podrška novih programskih jezika koji nativno podržavaju izradu mobilnih aplikacija (Swift za iOS i Kotlin za Android) su motivator za daljnji razvoj React Native biblioteke.

Pojava progresivnih web aplikacija može smanjiti udio mobilnih aplikacija jer one objedinjuju i web i mobilnu platformu što u konačnici može još više ubrzati razvojni proces, smanjiti mjesto za pogreške i pojeftiniti troškove održavanja. Progresivne web aplikacije još uvijek se razvijaju, no taj ubrzani razvoj je već i danas vidljiv kako velike kompanije sve lakše prisvajaju takav način razvoja.

Progresivne web aplikacije već podržavaju postavljanjem ikone na početni zaslon korisnikovog uređaja, *fullscreen* način rada, integraciju s mikrofonom i zvučnikom, mobilno plaćanje, slanje nativnih obavijesti, te još mnogo drugih funkcionalnosti.

Nativne mobilne aplikacije sigurno neće nestati, ali bez alata za brži razvoj kao što je React Native cijena i brzina njihove izrade neće biti usporediva s cijenom i brzinom izrade progresivnih web aplikacija.

Literatura

[1] Mmww2.png – https://i.stack.imgur.com/Mmww2.png, svibanj 2017

[2] Showcase – https://facebook.github.io/react-native/showcase.html, svibanj 2017

[3] Tomislav Car – 27.08.2016. – *Android development is 30% more expensive than iOS. And we have the numbers to prove it!* – https://infinum.co/the-capsized-eight/android-development-is-30-percent-more-expensive-than-ios, svibanj 2017

[4] Dokumentacija @observable dekoratora – https://mobx.js.org/refguide/observable-decorator.html, svibanj 2017

[5] Decorators – 04.01.2017. – https://tc39.github.io/proposal-decorators/, svibanj 2017

[6] Babel.js – https://babeljs.io/, svibanj 2017

[7] Modulecounts – http://www.modulecounts.com/, svibanj 2017

[8] Node Package Manager – https://www.npmjs.com/, svibanj 2017

[9] Yarn Package Manager – https://yarnpkg.com/en/, svibanj 2017

[10] Smartphone market share – http://www.idc.com/promo/smartphone-market-share/os, lipanj 2017

[11] JavaScript Wiki stranica – https://en.wikipedia.org/wiki/JavaScript, svibanj 2017

[12] ECMAScript Wiki stranica – https://en.wikipedia.org/wiki/ECMAScript, svibanj 2017

[13] Wiki stranica o AJAX upitima – https://en.wikipedia.org/wiki/Ajax_(programming), svibanj 2017

[14] Dokumentacija za React Native biblioteku – https://facebook.github.io/react-native/docs/getting-started, svibanj 2017

[15] Primjeri aplikacija rađenih pomoću React Native biblioteke – https://facebook.github.io/react-native/showcase.html, svibanj 2017

[16] Službena stranica Postgresql baze podataka – https://www.postgresql.org/, svibanj 2017

[17] Službena stranica MongoDB baze podataka – https://www.mongodb.com/, svibanj 2017

[18] Službena stranica AAI@Edu sustava – http://www.aaiedu.hr/, svibanj 2017

[19] Službena informacije o Apple Developer računu – https://developer.apple.com/account/, lipanj 2017

[20] Službene stranice Apple testFlight sustava – https://developer.apple.com/testflight/, lipanj 2017

[21] Opis spremanja podataka u interno spremište – Sandro Machado – 24.08.2016. – https://stackoverflow.com/questions/39028755/save-sensitive-data-in-react-native, lipanj 2017

[22] Jonathan Z. White – 25.04.2017. – *CSS in JavaScript: The future of component-based styling* – https://medium.freecodecamp.com/css-in-javascript-the-future-of-component-based-styling-70b161a79a32, svibanj 2017

[23] Progresivne web aplikacije – https://developers.google.com/web/progressive-web-apps/, lipanj 2017

[24] Način rada React Native biblioteke – Tadeu Zagallo – 14.08.2015. – https://tadeuzagallo.com/blog/react-native-bridge/, lipanj 2017

[25] Dobre i loše strane React Native biblioteke –Vytenis Narusis – 28.04.2016. – https://www.devbridge.com/articles/pros-cons-of-react-native-crash-course/, lipanj 2017

Višeplatformska mobilna aplikacija za sustav za testiranje studenata Edgar

Kako bi se olakšalo studentima rješavanje ispita kroz sustav za testiranje *Edgar* izrađena je mobilna aplikacija za dvije najveće mobilne platforme: Android i iOS. Proučen je trenutni web sustav *Edgar*, te su učinjene preinake potrebne za rad s mobilnom platformom. Te preinake uključuju i novi način autentifikacije pomoći jedinstvenog PIN broja i token ključa. Mobilna aplikacija je izrađena koristeći React Native biblioteku. Opisana je razlika između standardnog načina razvoja nativnih mobilnih aplikacija s razvojem koristeći web tehnologije. Osnovne funkcije *Edgar* sustava su implementirane unutar mobilne aplikacije.

Ključne riječi: Mobilna aplikacija, sustav za testiranje studenata, Edgar, React Native, web tehnologije, Android, iOS.

Multi-platform Mobile Application for the Edgar student testing system

To allow students easier access to system for automated testing *Edgar*, mobile application was created for two largest mobile platforms Android and iOS. Current system *Edgar* was studied and modifications were made to allow connection with mobile application.Those modifications include new type of authentication using PIN number and token key. Mobile application for created using React Native library. Difference between standard development of mobile applications and development using only web technologies was described. Some features of *Edgar* system were implemented in mobile application.

Keywords: Mobile application, student testing system, Edgar, React Native, web technologies, Android, iOS.