Mobilne aplikacije u vrednovanju postignuća učenika

Rade, Ante

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

MOBILNE APLIKACIJE U VREDNOVANJU POSTIGNUĆA UČENIKA

Ante Rade

Split, Septembar 2019.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za Informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

MOBILNE APLIKACIJE U VREDNOVANJU POSTIGNUĆA UČENIKA

Ante Rade

SAŽETAK

Tema ovoga rada su mobilne aplikacije za vrednovanje studenata, te je u radu opisano kakav je to tip aplikacija, te što takva aplikacija sadrži. Neke od najpoznatijih aplikacija su i navedene te opisane po čemu su tako posebne i što ih čini najboljima. Navedeni su i osnovni korisnički zahtjevi koje ovakva aplikacija mora sadržavati, te je na primjeru jedne aplikacije objašnjena i implementacija tih zahtjeva. Aplikacija koja je izrađena i prikazana je najosnovnija aplikacija ovoga tipa. Na početku rada su navedene i neke statistike koje pokazuju važnost i učinkovitost aplikacija ovoga tipa, u današnjem modernom svijetu.

Ključne riječi: aplikacija, mobilna, student, vrednovanje

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

Rad sadrži: 51 stranica, 9 grafičkih prikaza, 0 tablica i 2 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **Dr. sc. Saša Mladenović,** *izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Neposredni voditelj:

Dr. sc. Goran Zaharija, *poslijedoktorand Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Ocjenjivači: **Dr. sc. Saša Mladenović,** *izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Dr. sc. Goran Zaharija, *poslijedoktorand Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Dr. sc. Divna Krpan, *viši predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: **Septembar 2019**.

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Computer Science
Ruđera Boškovića 33, 21000 Split, Croatia

MOBILE APPLICATION IN STUDENT EVALUATION

Ante Rade

ABSTRACT

The topic of this paper is mobile applications for student evaluation, and the paper describes what type of application is and what such application contains. Some of the most famous applications are also listed and described what makes them so special and what makes them the best. The basic user requirements are explained on the example of one application. An application that is made and displayed is the most basic application of this type. At the beginning of the paper, some statistics are presented that show importance and effectiveness of applications of this type in today's modern world.

Key words: application, student, evaluation, mobile

Thesis deposited in library of Faculty of science, University of Split

Thesis consists of 51 pages, 9 figures, 0 tables and 2 references

Original language: Croatian

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

Supervisor: **Goran Zaharija, Ph.D.** *Professor of Faculty of Science, University of Split*

Reviewers: **Saša Mladenović, Ph.D.** *Associate Professor of Faculty of Science, University of Split*

Goran Zaharija, Ph.D. *Professor of Faculty of Science, University of Split*

Divna Krpan, Ph.D. *Professor of Faculty of Science, University of Split*

Thesis accepted: **September 2019.**

IZJAVA

kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam završni rad s naslovom MOBILNE APLIKACIJE U VREDNOVANJU POSTIGNUĆA UČENIKA izradio samostalno pod voditeljstvom dr.sc.Saše Mladenovića. U radu sam primijenio metodologiju znanstvenoistraživačkog rada i koristio literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući naveo u diplomskom radu na uobičajen, standardan način citirao sam i povezao s fusnotama s korištenim bibliografskim jedinicama. Rad je pisan u duhu hrvatskog jezika.

Student

Ante Rade

Prazna stranica

Sadržaj

Uvod

U današnjem svijetu, svijetu tehnologije, gdje je Internet prisutan u svim aspektima ljudskoga života, usvajanje novih znanja je postalo dostupno svakome čovjeku. Trenutno postoji mnoštvo mrežnih (engl. *online)* platformi za učenje, na kojima se može naučiti razne jezike, nadopuniti znanje o mnoštvu prirodoslovnih znanosti. Tako postoje i platforme za učenje programiranja. Neke od njih su *Codeacademy, Coursera, edX, Udemy,* itd. Na svakoj od tih platformi postoji mnoštvo tečajeva za gotovo svaki programski jezik danas poznat. Sve navedene platforme spadaju u *web* platforme, no, postoje također i mobilne verzije platformi za učenje. Neke od poznatijih mobilnih aplikacija za učenje programiranja su *SoloLearn, ProgrammingHub, LearnProgramming,* itd. Cilj ovoga rada je napraviti jednu takvu mobilnu aplikaciju, koja bi pomogla studentima u učenju programiranja.

1. Online platforme za učenje

Tehnologija je modernizirala cijeli svijet, pa i predavanje i učenje. Lekcije i seminari se više ne moraju održavati u školama. Zato što pomoću moderne tehnologije predavači i studenti imaju digitalni alatni okvir – u rasponu od mobilnih uređaja preko virtualnih sustava učenja do internetskih tečajeva. Velika prednost *online* učenja je prikupljanje podataka za prepoznavanje prednosti i slabosti svakoga učenika. Na temelju toga platforma može prilagoditi svoj pristup i učiniti proces učenja jednostavnijim i učinkovitijim. Danas se *online* učenje sve više koristi. Kako od pojedinaca, tako i do velikih korporacija koje preko *online* tečajeva obučavaju svoje zaposlenike. Ovo su neke impresivne statistike internetskog učenja koje pokazuju kako industrija bilježi procvat. I to s dobrim razlogom.

- Predviđa se da će svjetsko tržište e-učenja u 2025. biti vrijedno 325 milijardi dolara (*Source: Forbes)*
    - Odgovarajuća brojka za 2014. bila je 165,36 milijardi dolara, pokazuju statistike e-učenja. To znači da će se tržište gotovo udvostručiti u jednom desetljeću. Za učenje putem interneta postoji velika potražnja. To je u osnovi, vrlo isplativo i povoljno za velike tvrtke.
- U 2017. godini otprilike 77% američkih korporacija koristilo je internetsko učenje *(Source: eLearning Industry)*
    - 1995. godine taj broj je iznosio samo 4%, što znači da su tvrtke relativno brzo prihvatile prednosti e-učenja.
- U 2017. godini 67% američkih tvrtki nudilo je mogućnosti učenja putem pametnih telefona (engl. *smartphone) (Source: eLogic Learning)*
    - Skoro svaki čovjek danas ima *smartphone.* Također postoji i mnoštvo aplikacija koje olakšavaju učenje, kao što pokazuju i statistike o mrežnom obrazovanju.

- E-učenje povećava stope zadržavanja učenja za između 25% i 60% *(Source: SHIFT)*
    - Pomoću e-učenja učenici imaju veću kontrolu nad procesom učenja. Štoviše, ako se dogodi da nešto zaborave, mogu uvijek ponovo pregledati materijal i podsjetiti se na zaboravljeno. Ovo otklanja veliki pritisak i omogućava im da se usredotoče na stvarno učenje.
- E-učenje je potaknulo povećanje prihoda za 42% američkih organizacija *(Source: eLearning Industry)*
    - Gotovo polovica američkih korporacija iskorištava prednosti e-učenja, pokazuju statistike e-učenja za 2019. godinu.
- IBM je nakon prelaska na e-učenje uštedio oko 200 milijuna dolara *(Source: SHIFT)*
    - Kada koriste *online* učenje, tvrtke mogu smanjiti troškove instruktora. Na taj način, smanjuju se i troškovi koji se odnose na putovanja, najam hotela i opremu. To je posebno važno kada zaposlenici žive u različitim gradovima, zemljama ili čak kontinentima.
- U 2016. godini, 81% studenata američkog sveučilišta složilo se da im digitalna tehnologija učenja pomaže u poboljšanju ocjena *(Source: Statista)*
    - Iako se od nastavnika očekuje da pruže dodatnu podršku studentima izvan nastave, oni nisu dostupni 24 sata dnevno. Mnogi studenti sada radije kontaktiraju sa svojim instruktorima putem e-pošte. Statistika e-učenja je sasvim jasna: Digitalna tehnologija učenja tu je da pomogne studentima kada god im je pomoć potrebna.

Zaključak svega je da je e-učenje budućnost. Industrija je u procvatu, kao što gore navedene statistike jasno pokazuju. Ona također ima potencijal revolucije u načinu prijenosa znanja. Zamislite samo mogućnosti.

U uvodu su navedene samo neke od mrežnih i mobilnih platformi za učenje. No, danas ih ima jako puno i svaka je posebna na svoj način, svaka pruža neki svoj način usvajanja znanja. No, prema *TechRadar-u*, među 5 najboljih *online* platformi za 2019. godinu, prva dva mjesta zauzele su *Docebo* i već spomenuta *Udemy* platforma [1].

1.1. Docebo mrežna platforma

Docebo je sustav za upravljanje učenjem (engl. *Learning Management System,* skraćeno LMS) koji pomaže organizirati, pratiti i distribuirati internetske tečajeve za formalno učenje, bilo za zaposlenike, klijente ili kupce. *Docebo* potiče suradnju omogućavajući zaposlenicima da postavljaju pitanja i dobivaju odgovore od relevantnih stručnjaka iz svoje organizacije. Učenici također mogu podijeliti vlastito znanje koje se može potvrditi stručnim pregledom i dijeliti između timova. Da bi administratori platforme učinkovito upravljali nedostatcima u vještinama i kompetencijama svoje organizacije, *Docebo Perform* alat omogućuje dodjeljivanje formalnih i neformalnih treninga onima kojima će možda trebati dodatne vještine u određenim područjima. Sustav je također prilagodljiv pa mu se može dati odgovarajući izgled i dojam prema marki neke organizacije.

Docebo ima korisničko sučelje (engl. *User Interface,* skraćeno UI) koje se jednostavno koristi i dolazi s širokim rasponom integracija treće strane, pa bi implementacija platforme u programima treninga neke organizacije trebala biti gladak proces.

1.2. Udemy mrežna platforma

Udemy je jako uspješna zajednica za nastavnike, studente i tvrtke. Sadrži velik broj tečajeva i uključuje alate za nadzor ponašanja korisnika te poslovne planove za tvrtke koji nisu jeftini. *Udemy* je jako poznato ime u svijetu *online* učenja i sada se može pohvaliti s preko 15 milijuna učenika kojima se prenosi više od 65 000 dostupnih tečajeva. Postoje dvije glavne opcije plana, Timski plan (engl. *Team plan)* i Poduzetnički plan (engl. *Enterprise plan)*. Timski plan dostupan je za 5-20 korisnika radi pružanja osnova i košta 240 dolara. Poduzetnički plan dolazi s dodatnim značajkama, te naravno, više košta. Oba plana su dostupna za besplatno probno razdoblje.

U međuvremenu, organizacije koje žele dugotrajno koristiti uslugu mogu se odlučiti za račun *Udemy for Business*, koji otključava više od 2500 tečajeva relevantnih za poslovanje osmišljenih za usavršavanje zaposlenika u određenim područjima. Može se izmjeriti i angažman u učenju kako bi se saznalo koji materijali najbolje rade za pojedine zaposlenike, sa uvidom u ponašanje korisnika i obrasce učenja.

Korištenje *smartphone-a* ili *tableta* za učenje jednostavno ima smisla. To je uređaj koji je gotovo uvijek uz čovjeka, što znači da se uz njega može učiti svugdje i postoji mnoštvo resursa za učenje. Neke su aplikacije bolje od drugih, ali sve postižu isti cilj, pomoći u učenju. Pet najboljih *Android* aplikacija za učenje u 2019. godini su: *Amazon Kindle, Coursera, Duolingo, Khan Academy* i *LinkedIn Learning*, prema članku [2].

1.3. Amazon Kindle

Amazon Kindle jedna je od tradicionalnijih aplikacija za učenje. Sadrži nevjerovatan broj referentnih vodiča, knjiga sa uputama, knjiga za samopomoć, udžbenika i još mnogo toga. Vrlo jednostavno se kupuju, preuzimaju i pročitaju. To je možda malo stara škola, ali ljudi uživaju u tome. U biti su knjige na aplikaciji jeftinije od fizičkih knjiga i *smartphone* ih može pohraniti jako puno.

1.4. Coursera

Coursera je internetska škola koja sadrži razne lekcije i časove koji se mogu polagati. Sadrži više od 1000 tečajeva, od matematike, preko znanosti, pa čak i do tehnologije i programiranja. Nastava ima predavanja, zadatke i video sadržaj. Završetkom tečaja se dobije potvrda o završetku koja može biti jako korisna pri zapošljavanju. Neki tečajevi su besplatni, no, većina ih se mora platiti. Aplikacija je izvrsna kombinacija stare škole i modernog učenja.

To su samo neke aplikacije za online učenje. Postoji ih mnogo više, te ne samo za *Android* nego i za *iOS* i *Windows* operacijske sustave. Cilj ovog rada je napraviti takvu jednu aplikaciju. U sljedećem poglavlju su detaljno opisani korisnički zahtjevi za aplikaciju ovog tipa.

2. Korisnički zahtjevi aplikacije

Kod razvoja svake programske podrške potrebno je odrediti korisničke zahtjeve prije nego se krene u razvoj same programske podrške. Korisnički zahtjevi za razvoj ovakvoga tipa programske podrške su općeniti za gotovo sve aplikacije ovakvoga tipa. Obuhvaćaju tri glavne točke:

1. *Backend*
2. Sučelje (engl. *Frontend)*
3. Responsivnost aplikacije

2.1. Korisnički zahtjevi za *Backend*

Kod programske podrške, *Backend* predstavlja pozadinski dio podrške, koji korisniku nije vidljiv, dio koji uključuje bazu podataka i server koji komunicira s tom bazom. Kod aplikacija ovakvoga tipa, na *Backendu* se najčešće nalaze podaci o korisnicima aplikacije, te tečajevi koje će korisnici polagati. U podatke o korisnicima, osim njihovih osnovnih podataka, spadaju i podaci o njihovim uspjesima prilikom učenja. To se odnosi na vrednovanje njihovog znanja prilikom korištenja aplikacije. Dakle, potrebno je realizirati bazu podataka koja će sadržavati sve navedene podatke. Naravno, treba omogućiti korisnicima registraciju i prijavu, te na taj način spriječiti neautorizirane korisnike od korištenja aplikacije. *Backend* također mora jako brzo funkcionirati tako da bi se korisnički podaci i podaci o tečajevima mogli jako brzo dohvaćati te da bi se s njima na *Frontendu* mogla vršiti interakcija.

2.2. Zahtjevi za sučelje aplikacije

Suprotno od *Backenda*, *Frontend* predstavlja sami izgled aplikacije, onaj dio aplikacije koji korisnik vidi i s kojim vrši interakciju. Dakle, *Frontend* predstavlja prikaz svih podataka koji se nalaze na *Backendu* te interakciju s tim podacima. Za taj dio je važno korisniku omogućiti registraciju i prijavu, kako bi mogao koristiti sve funkcionalnosti aplikacije.

Nakon logiranja, korisniku moraju biti dostupni njegovi podaci, u koje spadaju i postignuća te ocjene koje su zaslužene na određenim kursevima u aplikaciji. Također, svakom korisniku moraju biti dostupni svi kursevi, sa svim poglavljima i zadacima, koje može interaktivno izvršavati, uz prikaz točnog odgovora. Korisniku mora biti omogućeno i ponovno polaganje kursa, nakon 30 dana od zadnjeg polaganja. Korisnik također mora moći vidjeti detalje svakoga kursa, te statistiku. Statistika kursa se odnosi na brojke u smislu koliko korisnika je polagalo kurs, te koliko ih je uspješno, odnosno, neuspješno završilo kurs. Naravno, uz sve navedeno, potrebno je izgled aplikacije dizajnirati na način koji će korisnicima biti primamljiv, oku ugodan.

2.3. Responsivnost aplikacije

Responsivnost (engl. *Responsivenes*), kao pojam prvi put je uveo web dizajner i programer Ethan Marcotte u svojoj knjizi „Responsive Web Design“. Responzivni dizajni reagiraju na promjene u širini podešavanjem položaja dizajnerskih elemenata koji se uklapaju u raspoloživi prostor. Responzivna web stranica prikazuje sadržaj na temelju dostupnog prostora u pregledniku. Ako otvorimo responzivno web mjesto na radnoj površini, a zatim promjenimo veličinu prozora preglednika, sadržaj će se dinamički pomaknuti kako bi se (barem teoretski) najbolje organizirao prostor u pregledniku. Na mobilnim uređajima je ovaj postupak automatski; web mjesto provjerava raspoloživi prostor, a zatim se predstavlja u idealnom rasporedu. Responzivni dizajn je izravan. Budući da je fluidan, korisnici mogu pristupiti internetskom svijetu i uživati u onoliko toga na svom ručnom uređaju, kao što bi to imali i na masivnom monitoru. Da bi to funkcioniralo, responzivni dizajn zahtjeva vrlo dobru konceptualizaciju web mjesta i poznavanje zahtjeva korisnika. No, kakvu ulogu responzivnost ima u ovoj aplikaciji? Ova aplikacija je nativna mobilna aplikacija. Kako danas postoji mnoštvo mobilnih uređaja sa različitim dimenzijama, responzivnost omogućuje prikaz podataka koji je najbolji za svaki taj uređaj, ovisno o njegovim dimenzijama. Više o tome kako je realizirana responzivnost u ovoj aplikaciji, bit će razjašnjeno u sljedećem poglavlju, u kojem je detaljno opisana realizacija svih korisničkih zahtjeva.

3. Realizacija zahtjeva

3.1. Realizacija *Backend*-a

Kako je u prethodnom poglavlju u korisničkim zahtjevima za *Backend* navedeno da je potrebno imati podatke o korisnicima i podatke o svim kursevima, vrlo je jasno kako bi trebala izgledati baza podataka. Za *Backend*, odnosno, za bazu podataka, korištena je *Real Time* baza podataka koju nudi *Firebase* platforma. *Firebase* je korišten zato što je vrlo jednostavan za korištenje i integriranje u aplikaciju. Vrlo je jednostavno spremati podatke u bazu i dohvaćati ih. *Firebase* je *Backend-as-a-Service* (skraćeno BaaS) *online* platforma koja služi kao podrška programerima u razvoju aplikacija. BaaS je model tzv. oblak usluge (engl. *cloud service*) u kojem programeri zaobilaze samostalno razvijanje *Backend-a*, te se orijentiraju samo na izgled aplikacije, bila to mobilna ili mrežna aplikacija. Dakle, *Firebase* praktički omogućuje cijeli *Backend* za razvoj programske podrške. Sadrži bazu podataka, *cloud* funkcije, spremište podataka (kao što su slike, videa, itd.), organiziranu autentikaciju koja može biti funkcionalna na različite načine, te mnoge druge funkcionalnosti koje čine *Backend.* Razlika *Firebase-a* od drugih baza podataka je da većina baza podataka zahtjeva HTTP zahtjev (engl. *Hyper Text Transfer Protocol request*) kako bi se pristupilo podacima iz baze, dok se na *Firebase* spajanje na bazu vrši pomoću *WebSocket-a,* koji su znatno brži od klasičnih HTTP zahtjeva. *Firebase* je danas jedna od najpoznatijih *Google Cloud* platformi za *Backend*, te je jako puno korištena u mnogim aplikacijama. *Real Time* baza podataka, koja je korištena u razvoju ove aplikacije, je nerelacijska baza podataka, u kojoj se podaci spremaju u JSON (engl. *Javascript Object Notation*) formatu i sinkroniziraju se u stvarnom vremenu sa svim povezanim klijentima. Kada se izrađuje aplikacija za više platformi sa *Firebase-ovim* SDK-ovima (engl. *Software Development Kit*) za *Android*, *iOS* i *JavaScript*, svi *Firebase-ovi* klijenti dijele jednu instancu *Real Time* baze podataka i automatski se primaju ažuriranja s najnovijim podacima. Dakle, podaci koji su spremljeni u bazi podataka za ovu aplikaciju, su podaci o svim kursevima i korisnicima aplikacije. Na sljedećoj stranici se nalazi slika sa prikazom baze podataka.

Slika 3.1 Struktura baze podataka

Dakle, ova baza podataka sadrži podatke s nazivom „tutorials“ i „users“. Vidimo da je „tutorials“ niz podataka, odnosno, niz kurseva, od kojih svaki ima svoj naziv, opis, naslovnu sliku i lekcije koje sadrži. Svaka od tih lekcija je jedno poglavlje kursa koja također ima svoj naziv, opis i interaktivne vježbe koje korisnik treba proći.

Međutim, vidimo da je i svaki kurs, kao i svaki korisnik, spremljen pod nekim nizom *string-ova* (*string* je naziv za niz znakova koji se koristi u mnogim programskim jezicima). To je jedinstveni ključ koji *Firebase* sam generira i na takav način omogućuje jedinstvenost podataka. Gore u samom vrhu, `learncode-efd41` je sam naziv baze podataka. Drugi dio naziva koji je crticom odvojen od prvoga je također jedinstveni string koji *Firebase* generira, radi jedinstvenosti baze podataka. Dakle, dobili smo strukturu baze podataka, no, pitanje je kako dohvaćati podatke, te kako u bazu spremati i ažurirati već postojeće podatke.

To *Firebase* omogućuje na vrlo jednostavan način, slanjem zahtjeva na poveznicu (engl. *link*) koju *Firebase* sam generira i koji sadrži jedinstveni naziv baze podataka unutar linka. Na generirani link se nadovezuju pristupne točke (engl. *endpoints*) koje omogućuju dohvaćanje svakog podatka unutar baze, i onog najudaljenijeg u JSON stablu. Na taj način može se primjerice, dohvatiti bilo koju vježbu bilo kojeg kursa ili bilo koji podatak o bilo kojem korisniku.

Alternativa *Firebase-u* je *Mongo* i još neke druge poznate platforme, međutim *Firebase* je odabran zbog dobrih iskustava sa navedenim.

3.2. Realizacija *Frontend-a*

Nakon što je baza podataka definirana, treba početi raditi na samoj aplikaciji. Tehnologija korištena u razvoju aplikacije naziva se *React Native*. *React Native* je razvijen od strane *Facebook-a* 2015. godine, te je od tada pa do danas korišten u mnogim svjetski poznatim aplikacijama, kao što su *Instagram, Skype* i mnoge druge. Dakle, *React Native* je tehnologija razvoja *nativnih* (engl. native) *mobilnih aplikacija* koji je baziran na modernom *JavaScript-u*. *Nativna mobilna aplikacija* je aplikacija za *smartphone* uređaje koja je kodirana u određenom programskom jeziku, kao što je *Objektno orijentirani C* za *iOS* ili *Java* za operativne sustave *Android. Nativne,* ili izvorne mobilne aplikacije pružaju brze performanse i visok stupanj pouzdanosti. Također imaju pristup raznim uređajima *smartphone-a,* poput kamere, galerije slika, lokacije, adresara i mnogih drugih. Pored toga, korisnici mogu koristiti neke aplikacije i bez internetske veze. Međutim, ovu vrstu aplikacije je jako skupo razviti jer je vezana za samo jednu vrstu operativnog sustava, što prisiljava tvrtku koja kreira aplikaciju da izrađuje višestruku verziju koja radi i na drugim platformama. Većina današnjih mobilnih igara za *smartphone* su *nativne* mobilne aplikacije.

Nakon svega navedenog se postavljaju vrlo očita pitanja: Gdje se tu uklapa *React Native,* koji je baziran na *JavaScript-u*? Na koji način *JavaScript* može napraviti *nativnu* mobilnu aplikaciju koja radi na *Javi* ili na *Objektivnom C-u*? U pitanju se upravo nalazi i odgovor. *React Native* je tehnologija koja dopušta kodiranje aplikacije u *JavaScriptu,* te nakon toga, izgrađivanje, odnosno *kompajliranje* te aplikacije u *Javu* ili *Objektni C,* kako bi ona savršeno funkcionirala na *Androidu* ili *iOS-u.*

Dakle, kodiranje se odvija u *JavaScriptu,* koji komunicira sa *nativnim* komponentama (*Javom* na *Androidu*, *Objektnim C-om* na *iOS-u* ili *C#* na *Windowsu*). Komunikacija se odvija putem takozvanog mosta (engl. *bridge*). Ukoliko, u bilo koje vrijeme razvoja aplikacije, navedena komunikacija oslabi ili uspori, neki djelovi koda se mogu implementirati u *Javi, Objektnom C-u* ili *C#-u,* kako bi se pokrenuli iz izvornog koda, a ne komunikacijom putem *JavaScripta.* Dakle, na taj način funkcionira *React Native.* Alternative *React Native-u* su već navedeni *Java* u *Android Studiu,* te *Vue Native,* koji je jako sličan *React Native-u* jer se također bazira na *JavaScriptu.*

Tijekom razvoja aplikacije, korištene su razne biblioteke, odnosno, paketi koji nisu direktno ugrađeni u *React Native,* nego ih je potrebno posebno instalirati i ugraditi u *nativni kod.* Takvi paketi se stručno nazivaju *biblioteke trećega reda* (engl. *Third-party-libraries*). Kako je u razvoju aplikacije korištena najnovija verzija *React Native-a* (0.60.5), koja vanjske pakete direktno ugrađuje u *nativni kod* samom instalacijom paketa, taj dio je vrlo jedostavan. Instalacija paketa se izvršava naredbom `npm install <naziv_paketa>`.

Npm je kratica za *Node Package Manager,* a sam *npm* je najveći svjetski registar paketa, koji sadrži preko 800 000 paketnih implementacija. Također je *open-source* (otvoren registar) i potpuno besplatan te ga mnoge organizacije koriste u implementaciji svojih programskih podrški.

Dakle, to bi bilo nešto o tehnologiji koja je korištena u samom razvoju aplikacije, slijedi detaljan opis realizacije najvažnijih korisničkih zahtjeva u aplikaciji.

U prethodnom poglavlju, u korisničkim zahtjevima je navedeno nekoliko bitnih stvari. To su:

- Autentikacija korisnika
- Pregled korisničkih postavki te sustav ocjenjivanja
- Prikaz svih kurseva te njihovih detalja
- Interaktivno sučelje za pisanje koda u svrhu odgovaranja na zadatke kurseva

Zadnja točka uključuje i način na koji su spremljena pitanja unutar *Firebase-ove* baze podataka, te kako su ta pitanja dohvaćena i pretvorena u interaktivni kod.

3.2.1. Autentikacija i navigacija

Dakle, ono što je već više puta spomenuto i što je jako bitno u velikoj većini aplikacija ovakvoga tipa, je sama registracija korisnika te njihovo *prijavljivanje.* U bazi podataka na *Firebase-u* su spremani podaci o svakome korisniku, njegovo ime, prezime, email, korisničko ime, lozinka te mnogi drugi podaci. Svaki put kada se netko novi registrira kao korisnik, novi entitet u tablici „users“ se stvara i što je jako bitno, stvara se pod jedinstvenim ključem, kojeg generira *Firebase.* Dakle, na početnoj stranici aplikacije se nalazi forma koju korisnik mora popuniti ukoliko se želi registrirati. Kada popuni podatke i klikne na dugme za registraciju, šalje se zahtjev na *Firebase.* Provjerava se da li postoji već takav korisnik, korisnik sa istim korisničkim imenom i lozinkom. Ako postoji jedan takav korisnik, zahtjev se odbija i korisnik se ne može registrirati. Ispisuje se poruka da podaci nisu valjani te da mora unijeti nove podatke. Kada se korisnik uspješno registrira, preusmjeren je na zaslon za *prijavu.* Na zaslonu za *prijavu* je potrebno unijeti korisničko ime i lozinku koja je stvorena tijekom registracije. Tu se također provjerava poklapanje tih dvaju podataka. Ako se podaci poklapaju, korisnik je ulogiran i aplikacija se preusmjerava na glavni zaslon na kojem se nalaze svi kursevi. To je dakle kratak opis procedure koju svaki korisnik mora proći da bi mogao koristiti sve funkcionalnosti aplikacije. Naravno, kada je korisnik jednom registriran, svaki sljedeći put je potrebno samo *prijavljivanje.* No, zavirimo sada malo u kod, te što je sve korišteno u implementaciji ove procedure. Za slanje zahtjeva na *Firebase* je korištena ugrađena *JavaScript* funkcija „fetch“, koja zahtjeva dva parametra: jedan je URL na koji se šalje zahtjev, URL koji je generiran od *Firebase-a* te se može vrlo lako pronaći na *Firebase-ovom webu,* a drugi je objekt koji sadrži samu konfiguraciju zahtjeva (kojeg tipa je zahtjev: *GET, UPDATE, POST,* itd.) te podatke koji se šalju unutar samoga tijela zahtjeva.

Na sljedećoj stranici se nalazi jedan od primjera komunikacije sa bazom podataka putem „fetch“ funkcije.

```
export const registrateUser = (userData) => {
    return dispatch => {
        const user = {
            firstName: userData.firstName.value,
            lastName: userData.lastName.value,
            age: userData.age.value,
            username: userData.username.value,
            city: userData.city.value,
            country: userData.country.value,
            email: userData.email.value,
            password: userData.password.value
        }
        fetch(`${apiURL}/users.json`, {
            method: 'POST',
            body: JSON.stringify(user)
        }).catch(error => {
            alert('Something went wrong, please try again');
        }).then(res => {
            return res.json();
        }).then(response => {
            dispatch({
                type: REGISTRATE,
                user: user
            })
        })
    }}
```

Kod 3.1 – *Redux* metoda za registriranje korisnika

Ovaj dio koda prestavlja funkciju „registrateUser“ koja šalje zahtjev na *Firebase* sa korisničkim podacima za registraciju. Dakle, ova funkcija sprema nove korisničke podatke u bazu, ako su podaci potpuno ispravni. Većina ovoga koda je poznata svim programerima koji su upoznati sa *JavaScriptom*, no, što predstavlja „dispatch“ u navedenom kodu? *Dispatch* metoda je dio tzv *Redux* paketa. *Redux* paket je zapravo okruženje (engl. *framework*) koji služi za pohranjivanje podataka na globalnoj razini, koji mogu biti dostupni u svakoj komponenti *React Native-a*. *Redux* se generalno sastoji od globalnog *state-a* (skladišta podataka), *akcija*, *reducera* i *dispatch* metode. U ovom slučaju, kod registriranja korisnika, *dispatch* metoda pokreće akciju za spremanje korisnikovih podataka u globalno skladište podataka (*state*).

Navedena akcija se obrađuje u tzv. *reduceru,* koji podatke o korisniku sprema u globalni *state.* Sljedeći dio koda prikazuje implementaciju *reducera* (točnije jedan dio koji je odgovoran za spremanje registriranog korisnika u *state*).

```
const reducer = (state = initialState, action) => {
    switch(action.type) {
        case REGISTRATE: {
            return {
                ...state,
                userData: action.user
            }
        }
}
```

Kod 3.2 – *Redux reducer*

Aplikacija sadrži više različitih akcija te *reducera* koji spremaju podatke koje akcije šalju u *state.* Ovaj dio spremanja u *state* je jako bitan jer su ti podaci potrebni za prikazivanje korisničkih postavki (podataka) koji su navedeni među korisničkim zahtjevima. Da bi se podaci spremljeni u *Redux state* mogli dohvaćati u komponentama *React Native-a,* potreban je još jedan paket, tzv. `react-redux`, koji spaja *React Native* sa *Redux-om.* Ovaj paket sadrži „Provider" komponentu koja u sebi sadrži cijeli *state*. Da bi podaci bili dostupni u svim komponentama, potrebno je obuhvatiti cijeli kod aplikacije unutar navedene „Provider" komponente. To je implementirano sljedećim dijelom koda:

```
class App extends Component {
  render() {
    return (
      <Provider store={store}>
        <Navigator />
      </Provider>
    )
  }
}
```

Kod 3.3 – Uvoz *Redux store-a* unutar cijele aplikacije

„App“ komponenta je tzv. korijenska (engl. *root*) komponenta aplikacije (korijen cijeloga stabla komponenti), iz koje se granaju sve ostale komponente, tj. cijeli kod. No, ovdje vidimo i „Navigator“ komponentu. Što ona predstavlja? „Navigator“ komponenta sadrži navigaciju unutar cijele aplikacije. Za njeno kreiranje korišten je paket `react-navigation`.

U mrežnim (*web*) aplikacijama navigacija je realizirana pomoću URL-a, no kod *nativnih mobilnih aplikacija* ne postoji URL. Postoje zasloni (engl. *screen*) koji se mogu pojavljivati na zaslonu *smartphone-a*. Tu na snagu dolazi prethodno navedeni paket, koji omogućuje prikaz bilo kojeg *screena*. Navigacija, odnosno, „Navigator“ komponenta je definirana sljedećim djelom koda:

```
const Navigator = createStackNavigator({
    Login: LoginScreen,
    Register: RegisterScreen,
    Tutorials: TutorialsScreen,
    TutorialDetails: TutorialDetailsScreen,
    Exercise: ExerciseScreen
})

const ProfileNavigator = createStackNavigator({
    Profile: ProfileScreen
})

const LogoutNavigator = createStackNavigator({
    Logout: LogoutScreen
})

const MainNavigator = createDrawerNavigator({
    Navigator: {
        screen: Navigator,
        navigationOptions: {
            drawerLabel: 'Home'
        }
    },
    Profile: ProfileNavigator,
    Logout: LogoutNavigator
}
```

Kod 3.4 – Implementiranje navigacije unutar aplikacije

Unutar `react-navigation` paketa korištene su dvije bitne funkcije. To su „createStackNavigator“ i „createDrawerNavigator“. Prva funkcija stvara običnu navigacijsku komponentu, u koju se zasloni spremaju pod jedinstvenim imenom (kao što je u kodu „LoginScreen“ spremljen pod imenom „Login“). Navigacija na neki od zaslona se tada izvršava u komponenti koja definira zaslon, pomoću metoda unutar „navigation“ objekta koji se nalazi unutar „props-a“ te komponente. Sve komponente (*screenovi*) koji su navedeni u „Navigator“-u imaju „navigation“ objekt u svojim „props“-ovima, te se navigacija može vršiti samo na one *screenove* koji su registrirani unutar funkcije „createStackNavigator“. Metoda koja vrši navigaciju je „navigate“, metoda unutar već spomenutog „navigation“ objekta:

```
this.props.navigation.navigate({routeName: 'Register'})
```

Ovdje vidimo kako metoda „navigate“ prebacuje aplikaciju na „Register“ *screen,* koji je definiran pod tim jedinstvenim nazivom, gore u spomenutoj „Navigator“ komponenti. Druga metoda `react-navigation` paketa, „createDrawerNavigator“, služi za stvaranje izbornika, te navigacije putem njega. Komponente koje su kreirane unutar „MainNavigator“ komponente se nalaze u izborniku koji će se pritiskom na „MENU“ dugme, izvlačiti sa desne strane. Opcije izbornika su definirane sljedećim djelom koda:

```
{
    drawerPosition: 'right',
    drawerBackgroundColor: '#b3d9ff',
    contentOptions: {
        activeTintColor: 'white',
        activeBackgroundColor: '#4a148c',
        labelStyle: {
            fontFamily: 'open-sans',
            fontSize: 20,
            fontWeight: 'bold'
        }
    }
```

Kod 3.5 – Opcije izbornika

Ovaj dio koda je objekt sa opcijama izbornika koji se dodaje kao drugi argument metodi „createDrawerNavigator“. U izborniku se nalaze dugmad koja vode do svakoga *screena* koji je naveden u definiciji „MainNavigator“ komponente.

Dakle, do sada je objašnjeno kako se spremaju podaci o korisnicima na globalnoj razini te kako aplikacija prelazi sa jednog *screena* na drugi. No, pojasnimo sustav autentikacije. Nakon što se korisnik registrira, aplikacija ga preusmjerava na „Login“ *screen.* Tu se korisnik, nakon što se prijavi, prebacuje na glavnu stranicu gdje se nalaze kursevi. Tu je ponovno korištena „Navigator“ komponenta, ali na malo drugačiji način:

```
this.props.navigation.replace('Tutorials').
```

Ovdje je korištena metoda „replace“, koja za razliku od metode „navigate“, u *screenu* koji je odredište, ne ostavlja u gornjem lijevom kutu strelicu za povratak unatrag. Kod prelaska iz „Login“ *screena* u „Registration“ je to bilo potrebno, pa je i ostavljeno, no, ovde to ne treba. Kada se korisnik jednom prijavi, „Login“ *screen* više nije potreban osim u slučaju da se korisnik odjavi (pritiskom na dugme „logout“ u izborniku). No, pitanje se postavlja, što ako korisnik potpuno iziđe iz aplikacije, pritiskom na mali kvadratić na dnu *smartphone-a.*

Taj problem je riješen pomoću „AsyncStorage“-a, odnosno, pomoću `async-storage`

paketa koji je omogućen službeno od *React Native* zajednice. Zadaća „AsyncStorage“-a je vrlo jednostavna, on omogućuje dugotrajno spremanje podataka. Da bi se podaci izbrisali, potrebno ih je ručno izbrisati unutar koda. Na taj način je omogućena trajna autentikacija korisnika. Dakle, nakon što se korisnik uspješno prijavi, prije prelaska na *screen* sa kursevima, u „AsyncStorage“ se sprema jedinstveni „id“ korisnika koji je generiran od strane *Firebase-a.* To je implementirano sljedećim dijelom koda:

```
await this.props.login(loginData)

        if (this.props.userData !== null) {
            await AsyncStorage.setItem('@uuid', this.props.us
erData.uuid);
            if (this.props.userData.tutorials) {
                await AsyncStorage.setItem('@tutorials', JSON
.stringify(this.props.userData.tutorials))
            }
            this.props.navigation.replace('Tutorials')
            }
```

Kod 3.6 – Implementiranje prijave korisnika

Dakle, kada korisnik unese podatke i pritisne dugme za prijavu, izvršava se prethodni kod. Metoda „login“ je *Redux* metoda koja šalje zahtjev na *Firebase* i koja sprema odgovor u *Redux state*. To se može vidjeti u sljedećoj implementaciji:

```
export const loginUser = (loginData) => {
    return async (dispatch, getState) => {
        const stateData = getState()
        const usersList = stateData.users.usersList

        const user = usersList.find((usr) => {
            return usr.username === loginData.username && usr
.password === loginData.password;
        })

        if (user) {
            dispatch({
                type: LOGIN,
                user: user
            })
        }
        else {
            alert('Incorect login data. Please try again');
            dispatch({
                type: LOGIN,
                user: null
            })
        }
    }
}
```

Kod 3.7 – *Redux* metoda za prijavljivanje korisnika

No, kada pogledamo malo bolje, ovdje nešto ne štima. Ovdje se metoda naziva „loginUser“, a u pozivu „login“. Da bismo to pojasnili, prvo moramo objasniti kako koristimo *Redux* metode unutar *React Native* komponente. Objasnimo to na ovom primjeru. Potrebno je uvesti (engl. *import*) dvije stvari: „connect“ metodu iz `react-redux` paketa te samu „loginUser“ metodu.

Uvođenje se implementira sljedećim dijelom koda:

```
import { connect } from 'react-redux'
import { loginUser } from '../../store/actions/auth'
```

Dalje, što je potrebno jest spremiti „loginUser“ metodu unutar „props“-a komponente. To je implementirano sljedećim kodom:

```
const mapStateToProps = state => {
    return {
        userData: state.auth.userData
    }
}

const mapDispatchToProps = dispatch => {
    return {
        fetchAllUsers: () => dispatch(fetchUsers()),
        fetchUser: (uuid) => dispatch(fetchUserData(uuid)),
        login: (loginData) => dispatch(loginUser(loginData))
    }
}

export default connect(mapStateToProps, mapDispatchToProps)(L
oginScreen);
```

Kod 3.8 – Uvoz *Redux* metoda unutar *React Native* komponenti

Ovdje se vidi da je komponenta „LoginScreen“ izvezena (engl. *export*) kao argument metode „connect“, koja je metoda višega reda, koja vraća metodu, koja na kraju vraća komponentu kao *React Native* komponentu sa dodatnim *Redux* podacima i metodama. Ovdje se vidi da je osim metode za prijavu, spremljena metoda za dohvaćanje podataka korisnika i metoda za dohvaćanje podataka svih korisnika. Ti podaci su potrebni kod samoga prijavljivanja korisnika (kod provjere da li postoji korisnik s unesenim podacima i da li se korisničko ime i lozinka podudaraju). U funkciji „mapStateToProps“ spremamo korisničke podatke koje imamo spremljene nakon logiranja u *Redux state.* Sada te podatke imamo spremljene i u „props“ komponente „LoginScreen“. Pomoću tih podataka je provjeravano da li je korisnik prijavljen. Dakle, na taj način funkcionira prijava korisnika. Kada je korisnik prijavljen i njegov jedinstveni identifikator spremljen u „AsyncStorage“, korisnik je prijavljen sve dok se ne odjavi.

Ako korisnik izađe iz aplikacije i ponovno otvori aplikaciju, korisnika će se odmah preusmjeriti na *screen* sa kursevima. To je implementirano sljedećim kodom:

```
async componentDidMount() {
        await this.props.fetchAllUsers();
        const userUuid = await AsyncStorage.getItem('@uuid');

        if (userUuid !== null) {
            await this.props.fetchUser(userUuid)
            this.props.navigation.replace('Tutorials')
        }
    }
```

Kod 3.9 – Preusmjeravanje prijavljenog korisnika

„componentDidMount“ je metoda životnog ciklusa (engl. *lifecycle method*) *React-a,* odnosno, *React Native-a,* koja se izvršava nakon što je prikazan *screen* općenito (u ovom slučaju „Login“ *screen*). Dakle, prvo što se izvršava je dohvaćanje svih korisnika jer je to potrebno kod provjere prijavljivanja, no ovdje to nije potrebno. Dakle, prvo se dohvati jedinstveni identifikator korisnika iz „AsyncStorage“-a. Ako on postoji dohvate se kompletni podaci o korisniku putem *Redux* metode koja je objašnjena prethodnim kodom, i nakon toga se korisnika preusmjeri na *screen* sa kursevima.

Dakle, u ovom potpoglavlju je detaljno objašnjena autentikacija korisnika te navigacija unutar aplikacije. U sljedećem poglavlju ćemo se osvrnuti na implementaciju interaktivnog sučelja za pisanje koda. Ostale točke koje su navedene u korisničkim zahtjevima će biti objašnjene u četvrtom poglavlju gdje ćemo vizualno proći kroz sve funkcionalnosti aplikacije. U ovom poglavlju se više okrećemo nekim težim, problematičnijim implementacijama. Stoga krenimo na interaktivno sučelje.

3.2.2. Interaktivno sučelje za pisanje koda

Kao što je već spomenuto, najbitniji dio ove aplikacije je interaktivno sučelje, kroz koje korisnik testira svoje znanje iz kurseva koji su mu ponuđeni. Sada ćemo opisati kako uopće to sučelje funkcionira. Dakle, rekli smo, da nakon što se korisnik prijavi, aplikacija ga preusmjerava na glavni *screen* na kojem se nalaze svi kursevi. U sljedećem poglavlju ćemo prikazati izgled tog *screena*. Za sada možemo samo spomenuti da se ti kursevi dohvaćaju pomoću *Redux* metode koja navedene kurseve sprema u globalni *state* aplikacije u *Redux-u*. Ta metoda se poziva nakon što se korisniku otvori *screen* sa kursevima. To omogućuje sljedeći kod:

```
import { FETCH_TUTORIALS, UPDATE_TUTORIALS } from './actionTy
pes'
import { apiURL } from '../../utils/Connection'

export const fetchTutorials = () => {
    return dispatch => {
        fetch(`${apiURL}/tutorials.json`)
            .catch(error => console.log(error))
            .then(res => res.json())
            .then(response => {
                let tutorials = [];
                for(let key in response) {
                    const tutorial = response[key]
                    tutorial.uid = key
                    tutorials.push(tutorial)
                }

                dispatch({
                    type: FETCH_TUTORIALS,
                    tutorials: tutorials
                })
            })
    }
}
```

Kod 3.10 – *Redux* metoda za dohvačanje svih kurseva

Već smo na primjeru registracije i prijavljivanja pojasnili kako funkcioniraju *Redux* metode te kako se one koriste u *React Native* komponentama. Dakle, na potpuno isti način se dohvaćaju kursevi. U *Redux middleware-u* koji omogućuje paket `redux-thunk`, prvo se izvršava zahtjev za dohvaćanje kurseva koji se šalje na *Firebase* a zatim „dispatch" metoda pokreće akciju za spremanje tih kurseva u *Redux state,* odnosno *store.* Akcija se obrađuje u *reducer-u,* te su svi kursevi spremljeni u *Redux state.* No, vratimo se malo na paket `redux-thunk`. To je naime paket trećega reda koji omogućuje izvršavanje asinkronog koda unutar funkcija za pokretanje akcija. Naime, u općenitom *Redux-u* to nije moguće, jer se „dispatch" metoda odmah poziva nakon što se pokrene funkcija za dohvaćanje kurseva u nekoj od komponenti. No, pomoću *redux-thunk-a* prvo se dohvaćaju kursevi sa *Firebase-a,* te nakon što su uspješno dohvaćeni, pokreće se akcija za njihovo spremanje.

Nakon što su kursevi dohvaćeni, vidi se njihov prikaz. Pritiskom na bilo koji od kurseva (točnije na njihovu naslovnu sliku) otvara se posebni *screen* sa detaljima o kursu na koji se pritisnulo. To ćemo također prikazati u sljedećem poglavlju. Korisnik, nakon što se informira o kursu, može započeti sa polaganjem kada to želi, pritiskom na dugme koje otvara novi *screen* („ExerciseScreen"), na kojem se nalaze interaktivne vježbe kursa, koje korisnik izvršava putem interaktivnog sučelja u kojem može slobodno pisati svoj kod. Korisnik u svakoj vježbi ima natuknice kako bi trebao izgledati jedan dio koda, te na taj način ne mora samostalno pisati cijeli kod, što olakšava vježbu.

Interaktivno sučelje je jednostavno realizirano pomoću „TextInput" komponente, koja je sastavni dio *React Native-a.* Sljedeći dio koda predstavlja implementaciju interaktivnog sučelja:

```
<TextInput
    multiline={true}
    numberOfLines={2}
    style={styles.input}
    value={this.state.userAnswer}
    onChangeText={this.codeHandler}/>
```

Kod 3.11 – Implementacija interaktivnog sučelja

Ovaj dio koda izgleda vrlo jednostavno, od obične komponente napravljen je ekvivalent *textarea* komponenti u *HTML-u,* u koji se može unositi što god korisnik želi.

Taj dio je jednostavan, no teži dio je pratiti korisnikovo polaganje kursa, listanje jedne vježbe za drugom, te mijenjanje poglavlja kursa. Također je potrebno napraviti da ako korisnik iziđe iz aplikacije a ostao je prijavljen, kada se ponovno vrati u aplikaciju na neki kurs koji je već počeo polagati, aplikacija ga mora vratiti na vježbu na kojoj je stao. Sve te podatke aplikacija prati i sprema pomoću već poznatog „AsyncStorage"-a. Već je spomenut kod autentikacije, gdje se sprema jedinstveni identifikator korisnika kako bi korisnik ostao stalno prijavljen, dok god se ne odjavi. Ovdje je na potpuno isti način korišten „AsyncStorage". Dok je korisnik prijavljen u aplikaciju, svi podaci o kursevima koje je korisnik polagao se čuvaju unutar „AsyncStorage"-a. Naime, kada se korisnik tek prijavi, ti podaci se dobiju iz baze podataka skupa sa svim ostalim podacima o korisniku. Dok je korisnik prijavljen i nastavlja sa svojim kursevima, podaci se nadopunjuju unutar *storage-a,* a tek kada se korisnik odjavi, nadopunjeni podaci se spremaju u bazu podataka, kako bi se mogli dohvatiti pri ponovnom prijavljivanju korisnika, a brišu se iz „AsyncStorage"-a zajedno sa korisnikovim identifikatorom. Sljedeći kod prikazuje podatke o korisnikovom kursu koje spremamo u „AsyncStorage" i bazu podataka:

```
uid: tutorial.uid,
name: tutorial.name,
correctAnswers: 0,
takenQuestions: 0,
lessonId: 0,
exerciseId: 0,
dateStarted: new Date(),
dateFinished: null
```

Kod 3.12 – Struktura spremanja korisničkih uspjeha na kursevima

Ovaj primjer prikazuje početne podatke o kursu, dakle, kada korisnik prvi put pokrene kurs. Ovdje vidimo da se sprema i jedinstveni identifikator kursa kako bismo ga mogli identificirati. Također za svaki kurs se sprema i koliko pitanja, odnosno, koliko vježbi je korisnik prošao, na kojoj lekciji te na kojoj vježbi je stao. Dodano je i da se sprema datum kada je korisnik pokrenuo kurs, te kada ga je završio. To je dodano zato što je polaganje kursa ograničeno na vremenski razmak od 30 dana. Dakle, kada korisnik završi određeni kurs, može ga polagati ponovno nakon 30 dana.

Dakle, trebalo je pomoću podataka iz „AsyncStorage“-a realizirati vraćanje na kurs te njegovo prvo otvaranje. Sljedeći kod predstavlja implementaciju navedene funkcionalnosti:

```
if (!tutorialIds.includes(tutorial.uid)) {
      tutorials.push({
      uid: tutorial.uid,
      name: tutorial.name,
      correctAnswers: 0,
      takenQuestions: 0,
      lessonId: 0,
      exerciseId: 0,
      dateStarted: new Date(),
      dateFinished: null
})
      await AsyncStorage.setItem('@tutorials', JSON.stringify
(tutorials))
      this.startTutorial(tutorial)
            }
else {
      const tutorialStats = tutorials.find(tut => tut.uid ===
 tutorial.uid)
      this.continueTutorial(tutorial, tutorialStats)
}
```

Kod 3.13 – Učitavanje zadataka

Ovaj dio koda se izvršava u „componentDidMount“, ugrađenoj *React* metodi koja je već ranije spomenuta, te se izvodi nakon što se otvori *screen* na kojem se nalaze vježbe. Dakle, dohvate se podaci o kursu iz „AsyncStorage“-a. Zatim se provjerava da li je korisnik već započeo kurs. Ako nije, dodaju se početni (engl. *default*) podaci za kurs u „AsyncStorage“ i pokreće se funkcija „startTutorial“, koja po prvi put pokreće kurs i sprema prvu vježbu iz prve lekcije u *state* od komponente. Kako se *state* mijenja, ponovno se pokreće cijela komponenta te se kod pojavljuje u „TextInput“-u, odnosno, interaktivnom sučelju. Ako je korisnik već polagao kurs, pokreće se funkcija „continueTutorial“, koja podatke o trenutnoj vježbi, koji su dohvaćeni iz „AsyncStorage“-a, sprema u *state* te sprema trenutnu vježbu u *state*. I u tom slučaju, naravno, *state* se promijenio te se vježba učitava u interaktivno sučelje. Nakon toga korisnik nastavlja sa kursom. Funkcionalnosti kod polaganja kursa biti će detaljnije objašnjene u sljedećem poglavlju.

Nakon svega navedenog, postavljaju se neka vrlo logična pitanja. Kako realizirati, odnosno prikazati sve vježbe jednu za drugom? Koliko *screenova* treba da bi se to realiziralo? Rješenje ovog problema leži u `react-navigation` paketu, koji je u aplikaciji korišten za navigaciju između komponenti. Za sve vježbe koje se prikazuju koristi se samo jedna komponenta, jedan *screen.* Naime, `react-navigation` paket sadrži funkciju „replace“ u *navigation* objektu koja zamijenjuje postojeći *screen* s nekim drugim. No, moguće je i zamijeniti trenutni *screen* s istim. Dakle, u ovom slučaju se *screen* „Exercise“ zamijeni ponovno sa „Exercise“ *screenom.* Da bi se otvorila sljedeća vježba, potrebno je ponovno poslati statistiku kursa u „props“ od „Exercise“ *screena.* Ovaj dio je implementiran sljedećim kodom:

```
nextExercise = async () => {
        const tutorialsStorage = await AsyncStorage.getItem('
@tutorials')
        const tutorialsStats = JSON.parse(tutorialsStorage)

        const currentTutorial = tutorialsStats.find((tutorial
) => tutorial.uid === this.state.tutorialId);
        const currentTutorialIndex = tutorialsStats.indexOf(c
urrentTutorial);

        const tutorialData = this.state.tutorialData
        if (tutorialsStats[currentTutorialIndex].lessonId ==
tutorialData.lessons.length) {
            await this.tutorialFinishedHandler(tutorialsStats
)
        }
        else {
            this.props.navigation.replace('Exercise', {
                tutorial: this.state.tutorialData,
                tutorialsStats: tutorialsStats
            })
        }
    }
```

Kod 3.14 – Implementacija učitavanja sljedeće vježbe

Dakle, pritiskom na dugme „Next Exercise“ pokreće se funkcija „nextExercise“ iz koda. Ona dohvaća trenutno stanje polaganja kursa (u kojem se nalazi identifikator *id* sljedeće vježbe te lekcije, poglavlja, u kojem se vježba nalazi). Nakon toga se provjerava da li ta vježba postoji u kursu, odnosno, da li je korisnik završio kurs. Ako jest aplikacija će korisnika preusmjeriti na *screen* sa kursevima, a korisnik može vidjeti svoj uspjeh na kursu unutar svojih korisničkih postavki, što ćemo vidjeti u sljedećem poglavlju.

Sa ovim je potpuno opisana implementacija interaktivnog sučelja te polaganje kurseva. U sljedećem poglavlju, gdje se opisuju funkcionalnosti aplikacije, vidjet ćemo i funkcionalnosti polaganje kurseva.

3.3. Responsivnost

U korisničkim zahtjevima smo vidjeli uvod u teoriju responsivnosti te je rečeno što ona predstavlja u ovoj aplikaciji te u drugim aplikacijama ovakvoga tipa. U ovoj aplikaciji responsivnost je rješena na vrlo jednostavan način. Naime, *React Native* sadrži API (engl. *Application Programmable Interface*) zvan *Dimensions,* koji sadrži funkciju koja u svakom trenutku može dohvatiti visinu i širinu *screena,* te osluškivati okretanje *screena* iz portretnog oblika (engl. *portrait mode*) u pejzažni (engl. *landscape mode*). Na osnovu tih podataka vrlo je lako rasporediti sadržaj aplikacije (*screena*) na način na koji god želimo. Sljedeći dio koda prikazuje uređivanje sadržaja aplikacije pomoću navedenog API-ja:

```
button: {
        width: Dimensions.get('window').width * 0.3,
        height: 50,
        backgroundColor: '#4a148c',
        justifyContent: 'center',
        marginTop: 30
    },
```

Kod 3.15 – *Dimensions* API kao uređivač sadržaja

Ovdje je uređivano dugme, te je njegova širina postavljena na 30% širine *screena.* Metoda „get“ *Dimensions* API-ja vraća objekt koji sadrži visinu i širinu *screena.* Ovdje je širina koja je dohvaćena pomnožena sa 0.3, što je 30% od ukupne širine.

Ovdje je definiran objekt „button“ koji sadrži svojstva navedena u kodu. No, kako dodijeliti ova svojstva dugmetu koje želimo urediti? U *React Native-u* ne postoji CSS (engl. *Cascading Style Sheet*), kao kod *web* stranica u pregledniku. Ovdje se uređivanje vrši na potpuno drugačiji način.

React Native sadrži API zvan *StyleSheet,* koji stvara nešto slično kao što je CSS *StyleSheet. StyleSheet* sadrži metodu „create“ koja stvara objekt sa stilovima koji su uneseni u objekt. Nakon toga svakome elementu u aplikaciji pridodajemo elemente tog objekta koje želimo i na taj način ih uređujemo.

```
const styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'flex-start',
        alignItems: 'center'
    },
```

Kod 3.16 – Uređivanje sadržaja aplikacije u *React Native-u*

Dakle, pomoću „create“ metode je stvoren objekt „styles“ u kojem se nalaze svi stilovi potrebni za sadržaj unutar komponente (u svakoj komponenti aplikacije se posebno mora kreirati objekt). Moguće je napraviti jedan zajednički objekt koji se može koristiti u cijeloj aplikaciji (kao u CSS-u), no to bi bilo dosta kofuzno jer bi bilo dosta podudaranja u nazivima).

Nakon što su stilovi stvoreni, potrebno je pridodati vrijednosti „styles“ objekta pojedinim komponentama, kako bi stilovi bili primjenjeni. Svaka komponenta u *React Native-u* ima svojstvo *style.* Potrebno je svojstvu *style* pridodati odgovarajuću vrijednost iz objekta „styles“ (vrijednost koja predstavlja stil tog sadržaja):

```
<TouchableOpacity
      disabled={!this.state.answerSubmited}
      style={styles.button}
      onPress={this.showAnswer}>
      <Text style={styles.buttonText}>SHOW ANSWER</Text>
</TouchableOpacity>
```

Kod 3.17 – Stvaranje i uređivanje gumbova

U kodu sa prethodne stranice se vidi kako je pomoću „styles“ objekta uređeno dugme „SHOW ANSWER“, za prikazivanje odgovora na vježbu. U *style* od komponente „TouchableOpacity“ (*React Native* alternativa za dugme, ima ih još puno) je dodan stil za uređivanje širine, visine i pozadine dugmeta, dok je posebno definiran stil za sami tekst unutar dugmeta te je dodan u „Text“ komponentu.

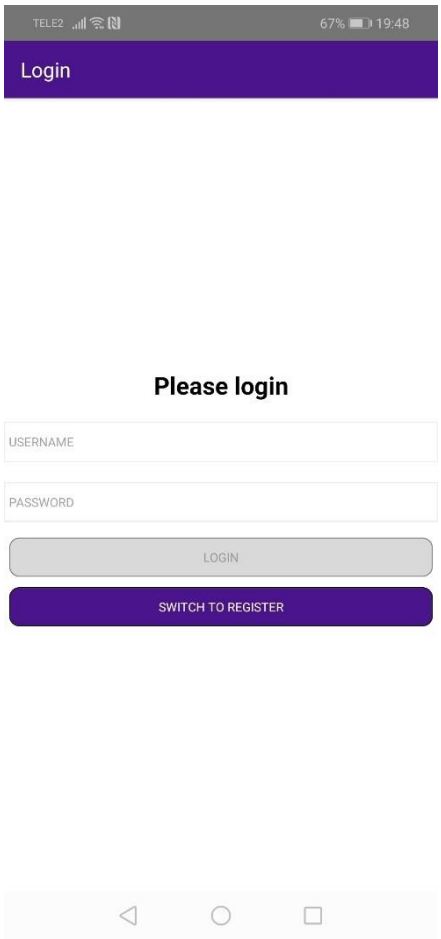
Na potpuno isti način su ređivane i ostale komponente u cijeloj aplikaciji, te je na taj način realiziran sam izgled aplikacije i njena responsivnost. Napokon smo opisali implementacije svih glavnih funkcionalnosti unutar aplikacije. Kako smo vidjeli, unutar tih implementacija je mnogo toga korišteno što ne pripada samom *React Native-u.* To su tzv. paketi trećega reda koji su već otprije poznati. Dakle, to nam daje neku opću sliku o *React Native-u,* koji je još u razvoju, te se na dnevnoj bazi radi na novim paketima trećeg reda, koji će sve više raširiti spektar primjenjivosti same tehnologije.

Kako je implementacija aplikacije opisana, još je potrebno vidjeti što sve aplikacije može, te vidjeti njene funkcionalnosti na djelu, vizualno. To ćemo vidjeti u sljedećem poglavlju.

4. Funcionalnosti aplikacije

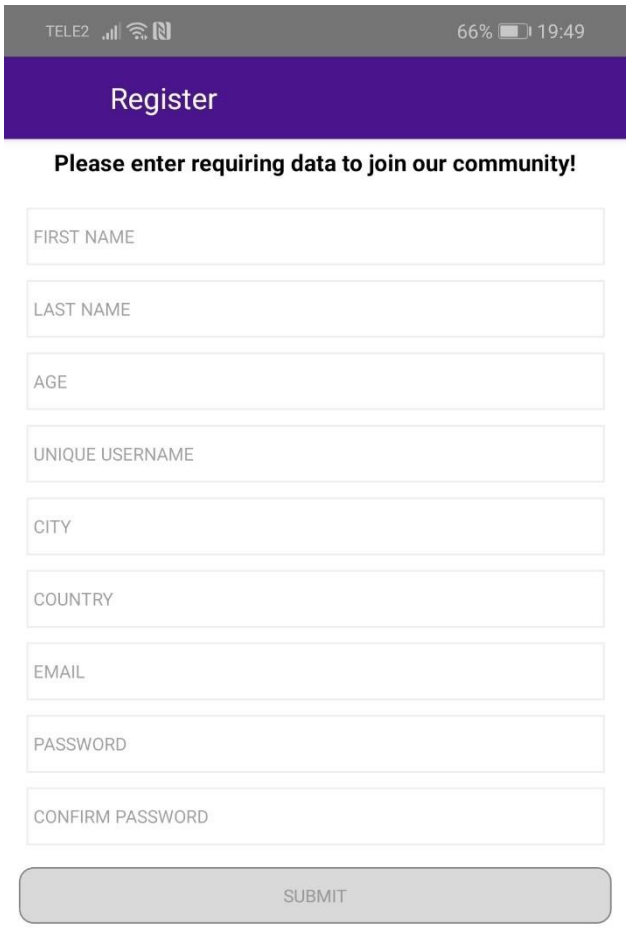
U ovom poglavlju ćemo vizualno proći kroz sve funkcionalnosti aplikacije, od prijave, preko liste kurseva i detalja o kursevima, preko korisničkih postavki i uspjeha, do izbornika i odjave korisnika. Dakle, detaljno ćemo proći i opisati svaki gumb i svaku funkcionalnost koju ova aplikacija omogućuje. Stoga, krenimo prvo otpočetka, od autentikacije.

U prethodnom poglavlju, na samom početku je rečeno da cijela aplikacija kreće od registracije i prijave korisnika. Da bi se korisnik mogao prijaviti potrebno je da se registrira. Dakle, prvi *screen* koji aplikacija otvori neprijavljenom korisniku je *screen* za prijavu koji je prikazan na sljedećoj slici.



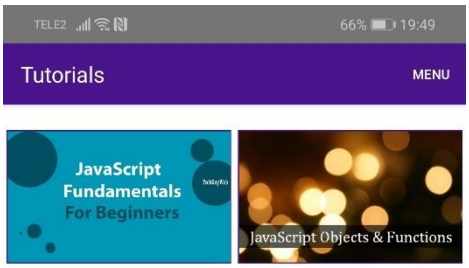
Slika 4.1 Logiranje korisnika

Dakle, ovdje se vidi i jasno je naznačeno da korisnik treba unijeti svoje jedinstveno korisničko ime i odgovarajuću lozinku, kako bi nastavio koristiti aplikaciju. Naravno, kada korisnik prvi put koristi aplikaciju, nema napravljen korisnički račun i potrebno ga je napraviti. Na *screenu* za prijavu vidimo na dnu gumb za prebacivanje na *screen* za registraciju, kako bi se mogao registrirati. *Screen* za registriranje izgleda ovako:



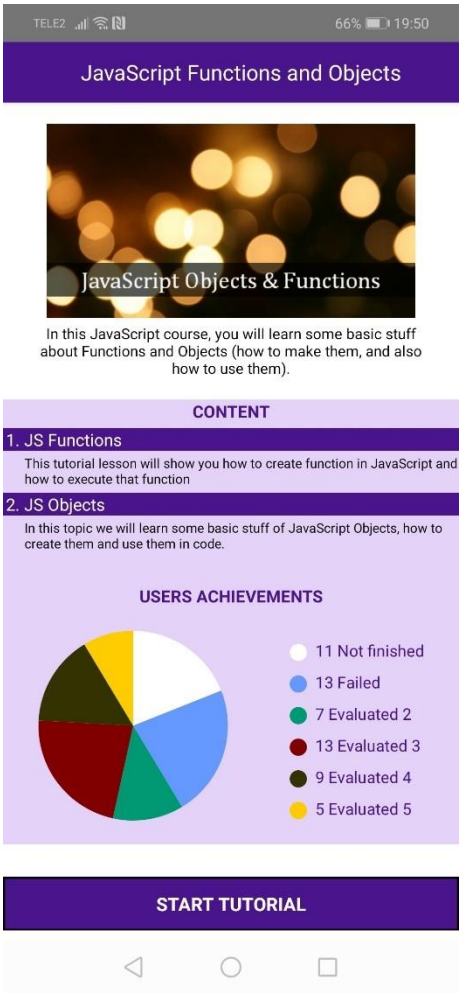
Slika 4.2 Registriranje korisnika

Ovdje vidimo da se od korisnika traži puno veći unos podataka od unosa kod prijave. Ovdje se traži ime, prezime, dob, email, itd. To su naime podaci koji će se spremati u *Firebase-ovu* bazu podataka, te će se ti podaci prikazivati u korisničkim postavkama, što ćemo vidjeti malo poslje. Dakle, već smo spomenuli da kada se korisnik registrira, aplikacija ga vraća na stranicu za prijavu, gdje se onda može prijaviti jer je napravio svoj korisnički račun. Nakon što se korisnik prijavi, njegov jedinstveni identifikator *id* se sprema te korisnik može početi koristiti aplikaciju. Kada je prijavljen, korisnik je preusmjeren na glavni *screen* sa listom svih kurseva koje aplikacija ima na raspolaganju. Ta stranica izgleda ovako:



Slika 4.3 Prikaz svih kurseva aplikacije

Dakle, vrlo jednostavan prikaz koji sadrži slike sa nazivom kurseva na koje se može pritisnuti. Pritiskom na sliku kursa otvara se *screen* na kojem se nalazi puni opis kursa te statistika, kao na slici.

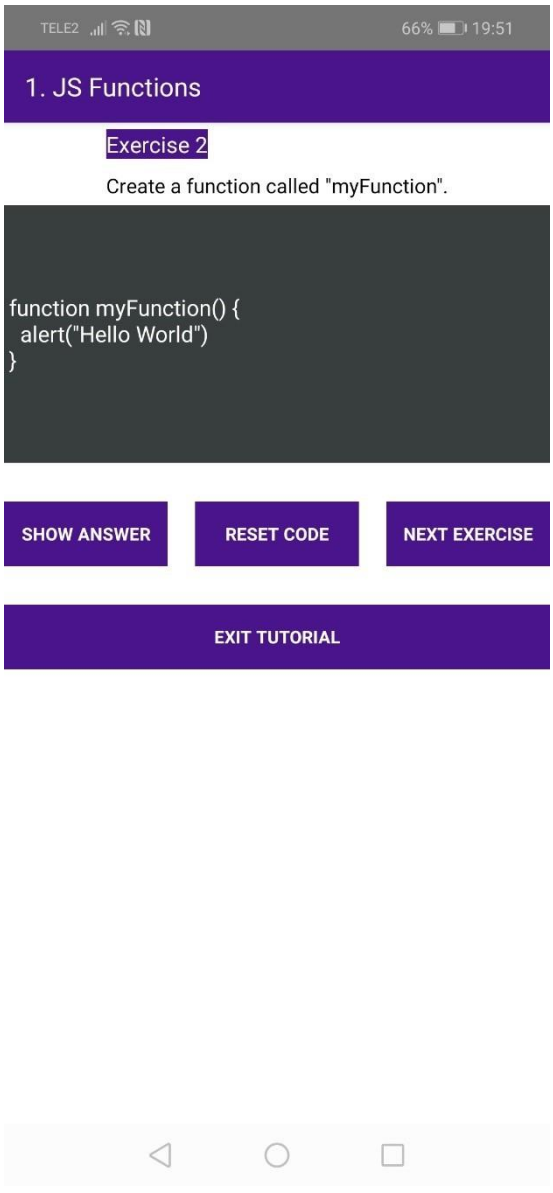


Slika 4.4 Detaljan prikaz kursa

U zaglavlju *screena* vidimo puni naziv kursa, te ispod njega već viđenu naslovnu sliku, malo umanjenu. Ispod slike se nalazi opis kursa, koji ukratko govori što će korisnik naučiti polagajući ovaj kurs. Ispod naslova „CONTENT“ izdvojena su sva poglavlja kursa (s opisom poglavlja koji se nalazi odmah ispod).

Prije gumba za pokretanje kursa imamo graf koji sadrži sve informacije o polaganju kursa, uključujući i statistiku polaganja svih korisnika koji su pokrenuli kurs. Na grafu je prikazano koliko korisnika nije završilo kurs, koliko njih je palo, te koliko ih je dobilo pozitivne ocjene i koje su to ocjene. Naravno, zbroj svih brojeva sa grafa daje ukupan broj svih korisnika koji su koristili kurs.

Na kraju, pritiskom na gumb za pokretanje kursa, korisnik započinje učenje. Otvara se novi *screen* na kojemu se nalaze vježbe i lekcije, te već spomenuto interaktivno sučelje za pisanje koda. Napokon, na sljedećoj slici možemo vidjeti i kako to sučelje izgleda.



Slika 4.5 Interaktivno sučelje i polaganje kursa

Dakle, ovdje korisnik provjerava svoje znanje u pisanju koda određenog kursa. Na vrhu, u zaglavlju *screena* vidimo redni broj i naziv trenutnog poglavlja. Ispod zaglavlja se nalazi broj vježbe te tekst koji opisuje što treba napraviti u zadatku.

Ispod vježbe se nalazi interaktivno sučelje u koje korisnik može upisivati što god želi. No postoji samo jedna verzija koda koja je ujedno i točan odgovor.

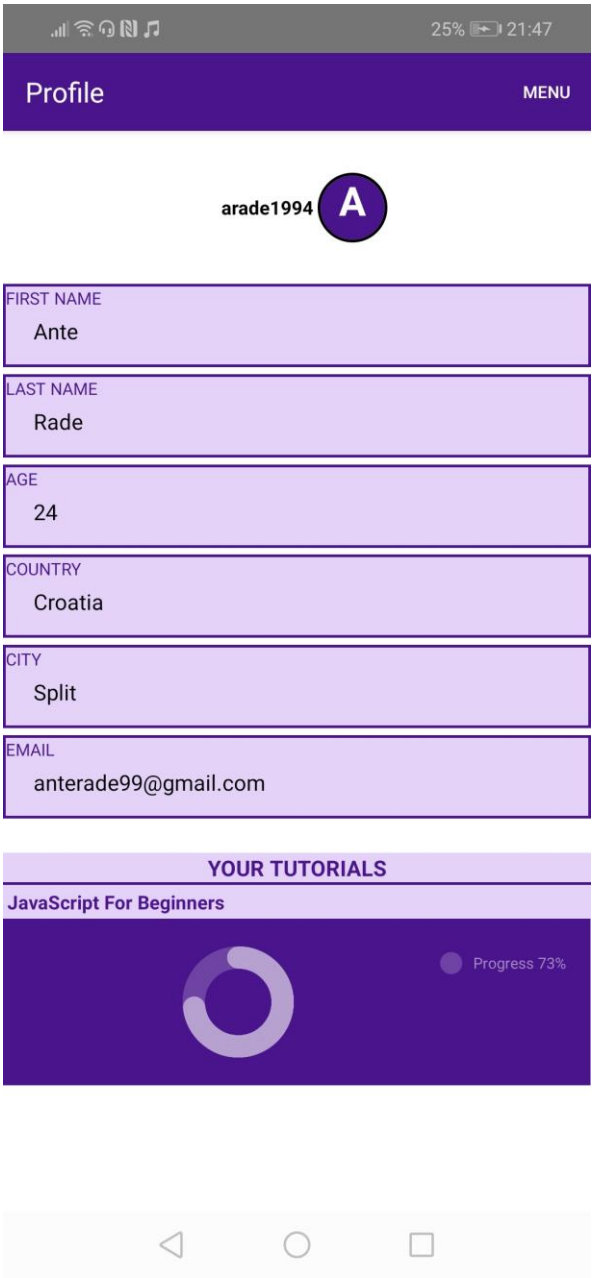
Kod je napisan tako da je većina koda već napisana u interaktivnom sučelju. Korisnik treba nadodati dio koda koji nedostaje kako bi odgovorio na zadatak. Ovakav način ispitivanja je jako dobar jer pruža korisniku naputak za točan odgovor.

Ispod sučelja se nalaze tri gumba, za pohranjivanje odgovora, poništavanje koda koji služi korisniku da vrati kod u početno stanje ukoliko je negdje pogriješio te gumb za sljedeću vježbu. Naime, kada korisnik jednom pohrani svoj odgovor (kod), više ga ne može mijenjati, niti resetirati kod. Nakon pohranjivanja, korisniku se ispisuje poruka da li je njegov odovor točan ili pogrešan, te se umjesto gumba za pohranjivanje odgovora pojavljuje gumb za prikaz točnog odgovora. Na taj način korisnik može vidjeti točan odgovor i uvidjeti svoje greške, ukoliko je pogriješio, te bolje naučiti. Nakon toga, jedino što korisnik može napraviti je otići na sljedeću vježbu. No, međutim kako svaki *smartphone* ima ugrađene gumbe za prisilni izlazak iz aplikacije, tako može izaći i iz ove. No, to je već pojašnjeno, da ako korisnik izađe iz aplikacije, aplikacija ga vraća direktno na sljedeću vježbu, dakle, ne može varati u aplikaciji kako bi jednu vježbu radio dva puta. Kada korisnik pritisne gumb za sljedeću vježbu, otvara mu se naravno sljedeća vježba i korisnik nastavlja na potpuno isti način, sve dok ne dođe do zadnje vježbe. Kada nakon zadnje vježbe pritisne dugme za sljedeću vježbu, ispisat će se poruka da je kurs završen te da više nema vježbi. Isto tako korisnik će na kraju dobiti i povratne informacije o svome uspjehu na kursu te koju je ocjenu dobio. Nakon poruke korisnik je vraćen na *screen* sa kursevima te može naravno, ako želi nastaviti s nekim drugim kursom.

Naime, kada korisnik završi jedan kurs, podaci na njegovoj profilnoj stranici će biti ažurirani kao i podaci o detaljima kursa (misli se na grafove na *screenu* sa detaljima kursa). Dakle, ponovno će se poslati zahtjevi na *Firebase* te će se podaci ažurirati prvo u bazi, a onda u aplikaciji.

Ono što je još preostalo od funkcionalnosti aplikacije je *screen* sa korisničkim postavkama, točnije sa podacima o korisniku te njegovim uspjesima, te izbornik i *screen* za odjavu korisnika. Krenimo od *screena* sa profilom korisnika.

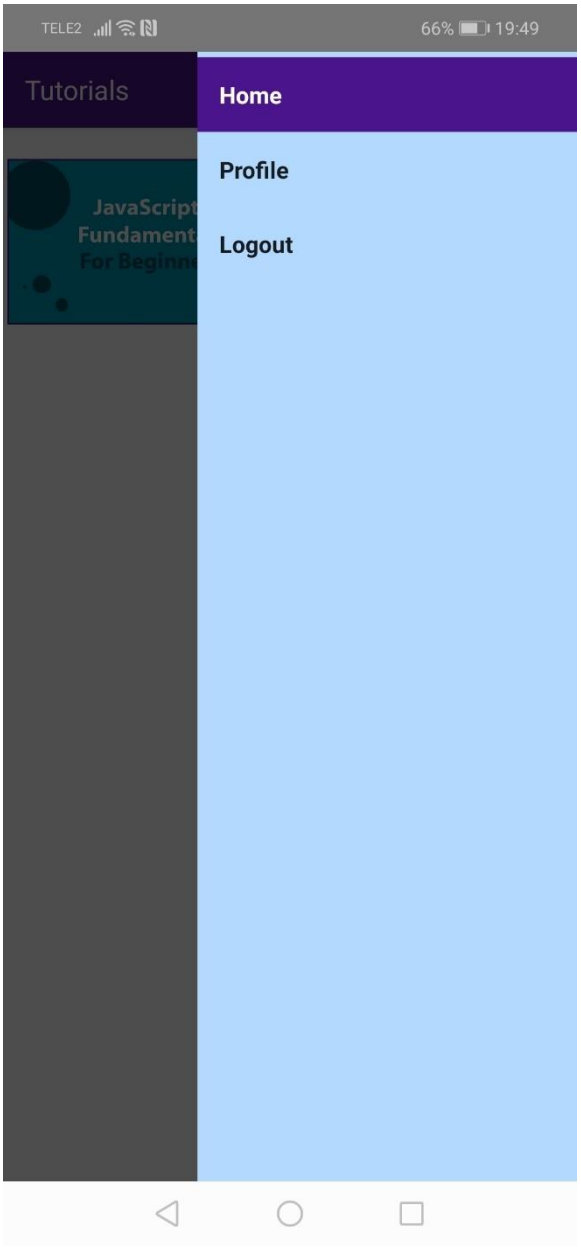
Na sljedećoj stranici nalazi se slika koja prikazuje izgled *screena* sa detaljima o korisniku.



Slika 4.6 Prikaz korisničkih podataka

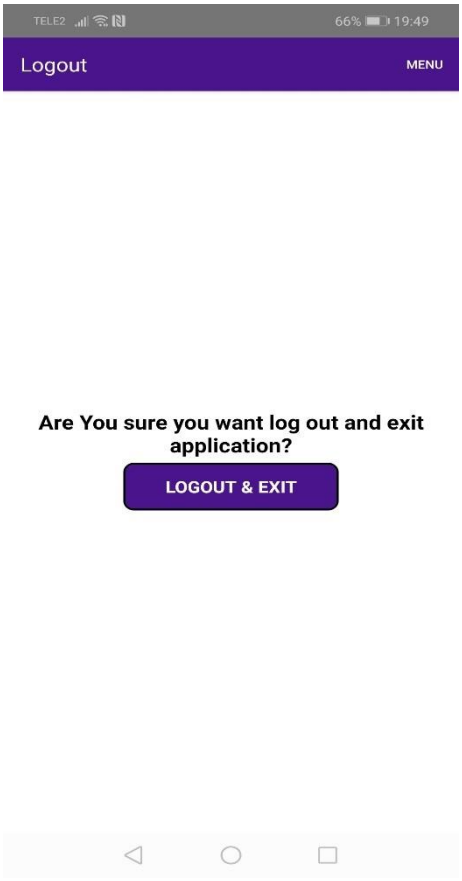
Ovdje se vide svi podaci o korisniku koji su uneseni tijekom registracije. Ispod se nalazi i statistika kurseva koje korisnik polaže, ili je položio. Na grafu se jasno vidi koliki dio kursa je korisnik prešao, te kod završenih kurseva se vidi i ocjena, odnosno uspjeh koji je korisnik ostvario.

Još prije nego što završimo funkcionalnosti aplikacije sa odjavom korisnika, pokazat ćemo kako izgleda izbornik. Sliku na kojoj se nalazi izbornik možete vidjeti na sljedećoj stranici.



Slika 4.7 Izbornik

U izborniku se nalaze navigacija na glavnu stranicu sa kursevima, profil korisnika te na *screen* za odjavu korisnika. Prethodne dvije navigacijeske rute su prikazane, no ako korisnik želi odjavu te pritisne na dugme za odjavu, otvara se *screen* na kojem se korisnika još jedan put pita da li je siguran da se želi odjaviti. Na sljedećoj slici vidimo izgled *screena* za odjavu.



Slika 4.8 Odjava korisnika

Kada se korisnik odjavi, aplikacija se zatvara, te ako korisnik ponovno otvori aplikaciju, mora se naravno, ponovno prijaviti.

Zaključak

Cilj ovoga rada je bio opisati trenutno najmoderniji način usvajanja znanja, putem tehnologije i Interneta, te barem malo pokazati čitatelju na koji način se takvo učenje provodi. Na početku rada smo vidjeli neke statistike, o današnjoj industriji *web* učenja, te njenoj ekonomskoj moći. To doista puno toga govori o samoj popularnosti ovakvoga tipa učenja. No, govori li nam to i o boljem učinku *online* učenja u odnosu na klasično učenje? Možda i da, jer smo vidjeli i statistike koje ukazuju i na zadovoljstvo studenata i profesora s *online* učenjem. *Online* učenje je skoro već preuzelo vodstvo kada su u pitanju velike korporacije. Korporacijama se, kao što smo vidjeli u statistikama, mnogo više isplate *online* kursevi kojima će usavršiti svoje radnike. Isto tako, i mnoge škole te visoke školske institucije koriste sve više *online* učenje. Moje iskreno mišljenje jest da je *online* učenje puno naprednije od klasičnog učenja i usto je i mnogo zabavnije i interaktivnije, barem kada je u pitanju učenje programiranja i programskih jezika. Kao što je već rečeno u uvodu, danas postoji mnoštvo *online* platformi koje svaka na svoj način čini učenje zanimljivim. Stoga sam mišljenja da će u vrlo skoroj budućnosti *online* učenje u potpunosti preuzeti vodstvo te će ljudi puno više napredovati u svojim vještinama i puno brže usvajati znanja iz područja znanosti koja ih zanimaju.

Literatura

[1] TECHRADAR, *Best online learning platforms of 2019*, 5.7.2019.
https://www.techradar.com/news/best-online-learning-platform

[2] ANDROID AUTHORITY, *Best Android learning apps*, Joe Hindy 7.7.2019.
https://www.androidauthority.com/best-android-learning-apps-566227/

[3] THE INTERACTION DESIGN FOUNDATON, *Adaptive vs. Responsive Design*, Mads Soegaard, 2019.
https://www.interaction-design.org/literature/article/adaptive-vs-responsive-design

[4] FIREBASE OFFICIAL SITE
https://firebase.google.com/

[5] TECHNOPEDIA, *Native Mobile Apps*
https://www.techopedia.com/definition/27568/native-mobile-app

[6] TECHJURY, *21-Astonishing E-learning statistics for 2019*, Bobby Chernev, 2.5.2019.
https://techjury.net/stats-about/elearning/

Sažetak

Tema ovoga rada su mobilne aplikacije u vrednovanju postignuća učenika, te je u radu opisano kakav je to tip aplikacija, te što takva aplikacija sadrži. Neke od najpoznatijih aplikacija su i navedene te opisane po čemu su tako posebne i što ih čini najboljima. Navedeni su i osnovni korisnički zahtjevi koje ovakva aplikacija mora sadržavati, te je na primjeru jedne aplikacije objašnjena i implementacija tih zahtjeva. Aplikacija koja je izrađena i prikazana je najosnovnija aplikacija ovoga tipa. Na početku rada su navedene i neke statistike koje pokazuju važnost i učinkovitost aplikacija ovoga tipa, u današnjem modernom svijetu.

Summary

The topic of this paper is mobile application in student evaluation, and the paper describes what type of application is and hat such application contains. Some of the most famous applications are also listed and described what makes them so special and what makes them the best. The basic user requirements are explained on the example of one application. An application that is made and displayed is the most basic application of this type. At the beginning of the paper, some statistics are presented that show importance and effectiveness of applications of this type in today's modern world.

Skraćenice

UI	*User Interface*	korisničko sučelje
LMS	*Learning Management System*	sustav za upravljanje učenjem
iOS	*iphone Operational System*	
BaaS	*Backend as a Service*	
HTTP	*Hyper Text Transfer Protocol*	
JSON	*JavaScript Object Notation*	
SDK	*Software Development Kit*	pribor za razvijanje softvera
Npm	*Node Package Manager*	
URL	*Uniform Resource Locator*	usklađeni lokator sadržaja
API	*Application Programmable Interface*	aplikacijsko programsko sučelje
CSS	*Cascading Style Sheets*	
HTML	*HyperText Markup Language*	

Privitak

Instalacija programske podrške

Programska podrška napravljena u radu se instalira preko USB kabela. Potrebno je uređaj (mobitel) u postavkama sustava postaviti u razvojni mod. Zatim jednostavnom naredbom iz naredbenog retka sa vašeg prijenosnog računala ili PC-a, iz direktorija u kojem se nalazi kod, instalira se aplikacija na vašem uređaju (razvojna verzija, prava verzija nije dostupna na Google Play aplikaciji). Naredba za instaliranje aplikacije je *react-native run-android* za Android uređaje dok je naredba za iPhone uređaje *react-native run-ios*. Da bi potrebna naredba funkcionirala, potrebno je instalirati pojedine softvere i programe, kao što su Android Studio, Python, Java SDK, itd. Potpuna dokumentacija o instaliravanju tih softvera se nalazi na službenoj dokumentaciji React Native-a, na sljedećem linku:

https://facebook.github.io/react-native/docs/getting-started

Korištenje aplikacije je vrlo jednostavno. Uostalom, u radu je detaljno opisana cijela funkcionalnost aplikacije, pa možete se koristiti četvrtim poglavljem rada kako biste uspješno koristili sve funkcionalnosti aplikacije, i nadam se, nešto i naučili.