React Native mobilna aplikacija za planiranje dnevnih zadataka korisnika

Morosavljević, Matija

VELEUČILIŠTE U BJELOVARU
PRIJEDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

REACT NATIVE MOBILNA APLIKACIJA ZA PLANIRANJE DNEVNIH ZADATAKA KORISNIKA

Završni rad br. 18/RAČ/2022

Matija Morosavljević

Bjelovar, travanj 2023.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Matija Morosavljević** JMBAG: **0314020790**

Naslov rada (tema): **React Native mobilna aplikacija za planiranje dnevnih zadataka korisnika**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Informacijski sustavi**

Mentor: **Tomislav Adamović, mag. ing. el.** zvanje: **viši predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Krunoslav Husak, dipl. ing. rač., predsjednik**
2. **Tomislav Adamović, mag. ing. el., mentor**
3. **Krešimir Markota, mag. ing. comp., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 18/RAČ/2022

U sklopu završnog rada potrebno je:

1. Izraditi dizajn aplikacije za planiranje dnevnih aktivnosti korisnika koristeći React Native UI komponente
2. Primijeniti dinamičke komponente za bolje korisničko iskustvo u radu s aplikacijom za planiranje dnevnih zadataka korisnika
3. Primijeniti Firebase sustav autentikacije aplikacije za planiranje dnevnih zadataka korisnika
4. Izraditi logičke strukture za spremanje podataka u Firestore bazu podataka
5. Upravljati funkcionalnostima aplikacije za planiranje dnevnih zadataka korisnika s pomoću Context API-a
6. Izraditi aplikaciju za planiranje dnevnih zadataka korisnika za objavljivanje na Google Play Store

Datum: 09.12.2022. godine

Mentor: **Tomislav Adamović, mag. ing. el.**

REPUBLIKA HRVATSKA VELEUČILIŠTE U BJELOVARU BJELOVAR

Sadržaj

1. Uvod

U današnje vrijeme kada ljudi imaju pristup moru informacija na koje nailaze beskonačnim listanjem društvenih mreža vrlo često dolazi do pregaranja. Mobitel gotovo svi imaju uz sebe i unatoč tome što upravo uz njega ljudi najčešće dolaze do pregaranja, on ipak može poslužiti kao alat – koristeći neke od aplikacija za planiranje. U ovom radu opisan je proces izrade aplikacije pomoću koje korisnik može isplanirati dan, postaviti podsjetnike za važne događaje i sl. U drugom poglavlju prikazan je razvoj mobilnih aplikacija i četiri tipa mobilnih aplikacija. U trećem su poglavlju opisane korištene tehnologije u projektu i razlozi njihova odabira. U četvrtom je poglavlju prikazana aplikacija – bitni dijelovi kao što su postavljanje projekta, neke od korištenih funkcija i slike zaslona gotove aplikacije.

2. RAZVOJ I TIPOVI MOBILNIH APLIKACIJA

Mobilna je aplikacija tip aplikacijske programske podrške (engl. *software*) dizajnirane za pokretanje i izvršavanje na mobilnom uređaju [1]. Razvoj mobilnih aplikacija je proces izrade aplikacijske programske podrške koja će se izvršavati na mobilnom uređaju odnosno operativnom sustavu mobilnog uređaja [2]. Dominantni operativni sustavi mobilnih uređaja su iOS tvrtke Apple i Android tvrtke Google. S obzirom da postoji više operativnih sustava na kojima se mobilne aplikacije izvršavaju, postoji i više načina razvoja odnosno tipova mobilnih aplikacija. U daljnjem su tekstu prikazani tipovi mobilnih aplikacija.

2.1 Nativne mobilne aplikacije

Nativne mobilne aplikacije su aplikacije koje su napisane u nativnom programskom jeziku same platforme i izvršavaju se izravno na platformi. Aplikacije napisane za iOS za razvoj koriste programski jezik Swift, a Android aplikacije koriste programski jezik Kotlin. Najveće prednosti nativnog razvoja su brzina izvođenja aplikacije i pristup aplikacijskom programskom sučelju (engl. *Application Programming Interface*) mobilnog uređaja kao što su kamera, obavijesti (engl. *notifications*), kalendar i sl. Nativni razvoj podrazumijeva razvoj aplikacije samo za jednu određenu platformu, za što je potrebno više vremena i timova (odnosno više resursa) u odnosu na razvoj iste te aplikacije za više platformi.

2.2 Višeplatformske mobilne aplikacije

Višeplatformske mobilne aplikacije se izvršavaju izravno na platformi kao i nativne mobilne aplikacije, ali nisu napisane u programskom jeziku same platforme, već su s nekog drugog prevedene na programski jezik koji odgovara određenoj platformi. Za razvoj višeplatformskih mobilnih aplikacija postoji nekoliko programskih okvira (engl. *framework)* od kojih su najpoznatiji React Native koji koristi programski jezik JavaScript i Flutter koji koristi programski jezik Dart. Prijevod tih aplikacija u nativni kod vrši se putem mosta (engl. *bridge*) što može rezultirati sporijom aplikacijom u odnosu na nativnu aplikaciju. Ovim pristupom se također kao i u nativnom razvoju aplikacije može pristupiti značajkama mobilnog uređaja putem aplikacijskog programskog sučelja. Najveća prednost ovakvog pristupa je ta što je iz jedne baze koda (engl. *codebase*) moguće razviti aplikaciju za više platformi što znatno smanjuje resurse potrebne za razvoj aplikacije.

2.3 Hibridne mobilne aplikacije

Hibridne mobilne aplikacije su napisane pomoću *web* tehnologija (HTML, CSS, JavaScript). One se ne izvršavaju u *web* pregledniku već se izvršavaju unutar nativne komponente mobilnog uređaja WebView. Ta komponenta radi kao *web* spremnik (engl. *container)* unutar kojeg se izvršava aplikacija. Ovim pristupom razvoja, aplikacije i dalje imaju pristup značajkama mobilnog uređaja kojeg omogućava programski okvir Apache Cordova koji putem mosta komunicira s programskim sučeljem mobilnog uređaja. Unatoč tome, ovakve aplikacije nemaju pristup svim značajkama mobilnog uređaja kao što je to slučaj u nativnim aplikacijama. Pristup ovakvim aplikacijama moguć je izvan mreže, a funkcionira na način da podatke koji su izmijenjeni izvan mreže sprema lokalno te ih sinkronizira čim se aplikacija ponovno poveže na internet. Brzina izvođenja ovakvih aplikacija lošija je u odnosu na prva dva prikazana tipa.

2.4 Progresivne mobilne aplikacije

Progresivne mobilne aplikacije su *web* stranice prilagođene za mobilne uređaje. Takve mobilne aplikacije nije potrebno preuzeti niti instalirati već ih je dovoljno pokrenuti unutar *web* preglednika. Unatoč tome što su to *web* aplikacije, takve aplikacije koriste mnoge mogućnosti koje *web* preglednik pruža, a to su izvanmrežni rad, pozadinski procesi (npr. za obavijesti ako je aplikacija ugašena) i umetanje poveznice (engl. *link*) aplikacije na početni zaslon mobilnog uređaja. Na taj se način korisniku pruža iskustvo kao da se radi o mobilnoj aplikaciji, a ne *web* aplikaciji. Najveći nedostatci ovakvih aplikacija su ti što mogućnosti same aplikacije uvelike ovise o značajkama koje korišteni *web* preglednik podržava i to što je brzina uvelike manja u odnosu na prethodno prikazana tri tipa aplikacije.

3. KORIŠTENE TEHNOLOGIJE

3.1 Baza podataka

3.1.1 Firebase

Firebase je platforma za razvoj aplikacija osnovana od strane Google-a 2011. godine. Jedna od glavnih usluga što Firebase pruža su *hosting* usluge za više tipova aplikacija (npr. Android, iOS, Web, C++ itd.). Uz *hosting* usluge, Firebase pruža okruženje za testiranje aplikacije, monitor performansi itd. Firebase korisnicima pruža nadzornu ploču na kojoj se vrši analitika aplikacije, ponašanje korisnika kroz aplikaciju i sl. Baze podataka koje Firebase pruža su Realtime Database i Cloud Firestore. Budući da je u radu za bazu podataka korišten Cloud Firestore, o Realtime Database bazi podataka neće biti govora. Više o Realtime Database i drugim uslugama Firebase-a moguće je pronaći na službenoj dokumentaciji [3].

3.1.2 Cloud Firestore

Cloud Firestore je fleksibilna i skalabilna NoSQL baza podataka u oblaku koja služi za manipuliranje podatcima na klijent i server strani (engl. *Client and Server Side development).* Za razliku od klasične SQL baze podataka, Cloud Firestore koristi dokumentno orijentiranu NoSQL bazu podataka (engl. *document-oriented database).* Takva baza podataka za pohranu podataka ne koristi tablice nego dokumente koji su organizirani u kolekcije (engl. *collections).* Svaki takav dokument sadrži set ključ – vrijednost parova (engl. *key – value pairs).* Cloud Firestore pohranjuje i sinkronizira podatke u stvarnom vremenu pomoću slušatelja u stvarnom vremenu (engl. *realtime listeners),* a podatci se dohvaćaju pomoću metoda koje su dio Firebase softverskog razvojnog paketa (engl. *Software Development Kit).* S obzirom da se u radu baza podataka koristi za mobilnu aplikaciju vrlo je bitna podrška izvanmrežnog rada što je upravo i jedan od razloga zašto je za bazu podataka odabran Cloud Firestore.

3.2 Mobilna aplikacija

3.2.1 React Native

React Native je softverski okvir temeljen na komponentama (engl. *component based framework)* kojeg je osnovala tvrtka Meta Platforms 2015. godine. Za razvoj aplikacije

koristi JavaScript koji se pomoću JavaScript mosta prevede u nativnu aplikaciju ovisno o platformi. Najčešće se koristi za izradu Android i iOS aplikacija iako se pomoću React Native mogu razvijati i aplikacije za druge platforme poput androidTV, macOS, tvOS. React Native je višeplatformski razvojni alat što znači da je s jednom bazom koda moguće razviti aplikaciju za više platformi. To se postiže tako da se napisani JavaScript kod prevodi s gore spomenutim mostom koji React Native komponente pretvara u komponente odgovarajuće platforme. React Native za razvoj koristi komponente koje se uvoze ovisno o potrebni od kojih je temeljna komponenta View. Radi lakšeg razumijevanja View se može usporediti s *div*-om na *webu*. Za pristup značajkama mobilnog uređaja brine se React Native API ili React Native zajednica, jedna od najaktivnijih zajednica otvorenog koda (engl. *open source communities*) na Github-u.

Programski kod 3.1: Primjer osnovne React Native aplikacije

```
import React from "react";
import { View, Text } from "react-native";

export default function App() {
    return (
        <View>
            <Text>Hello world!</Text>
        </View>
    );
}
```

3.2.2 Android Studio

Android Studio je službeno integrirano razvojno okruženje za Android. Dizajniran je za razvoj Android aplikacija, a izgrađen je na IntelliJ IDEA softveru firme JetBrains. Uz mnoge druge opcije o kojima se može pronaći nešto više na službenoj dokumentaciji [4], Android Studio nudi mogućnost pokretanja emulatora Android uređaja. To znači da je uz pomoć Android Studio-a moguće testirati aplikaciju na Android operativnom sustavu bez potrebe za fizičkim uređajem s instaliranim Android operativnim sustavom, što je i glavni razlog upotrebe Android Studio-a u ovom projektu.

3.2.3 Expo

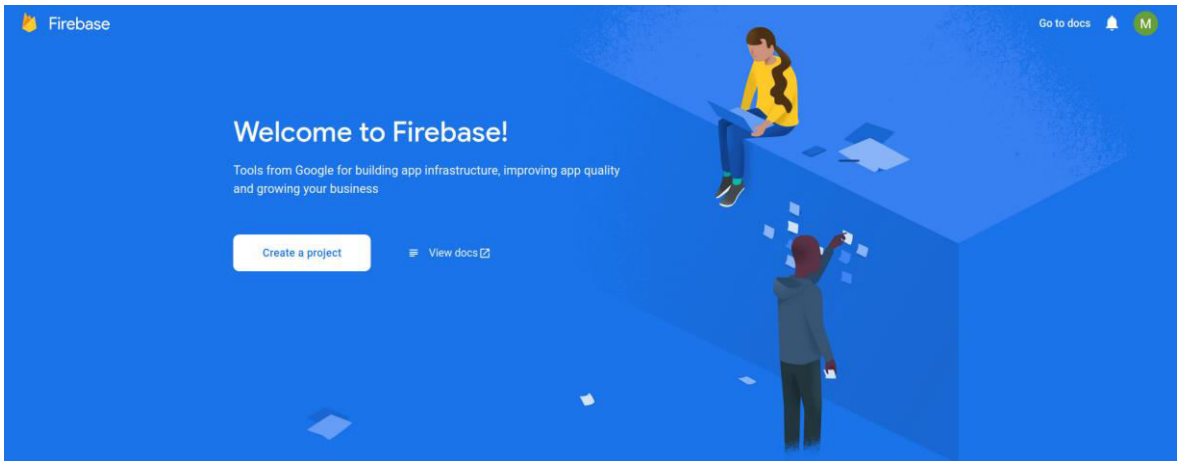
Expo je softverski okvir otvorenog koda koji kao nadogradnja na React Native služi kao alat za lakše skaliranje, testiranje i objavljivanje aplikacije. Expo nudi mogućnost izgradnje (engl. *build*) aplikacije i korištenje brojnih paketa i API-a za pristup značajkama

mobilnog uređaja bez potrebe Xcodea (za iOS) i Android Studio-a (za Android) kao što je to slučaj u golom (engl. *bare*) React Native projektu. Uz Expo je također moguće testirati aplikaciju na fizičkom uređaju bez Xcode-a ili Android Studio-a ili u *web* pregledniku u Expo Snack *web* aplikaciji u kojoj se nalazi emulator koji simulira ponašanje aplikacije u iOS, Android ili *web* okruženju. Expo pri pokretanju projekta pokreće LAN server i generira QR kod kojeg se može skenirati nakon čega se aplikacija pokreće na fizičkom uređaju. QR kod se skenira s kamera aplikacijom u slučaju da se radi o iOS uređaju ili s Expo aplikacijom u slučaju da se radi o Android uređaju. U oba slučaja na uređaju Expo aplikacija mora biti prethodno instalirana.

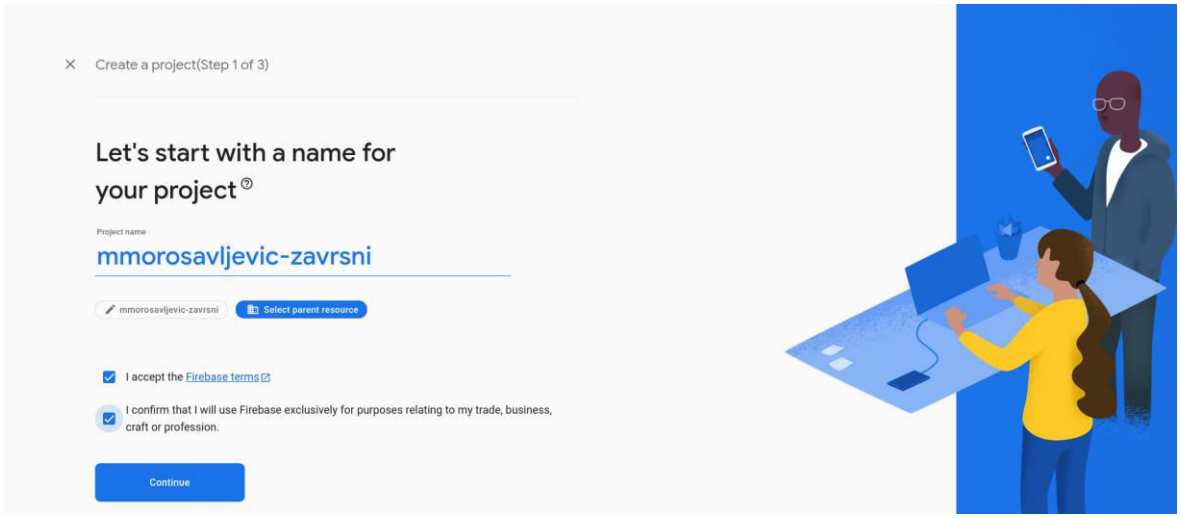
4. PRIKAZ APLIKACIJE

4.1 Baza podataka

Baza podataka korištena za ovaj projekt je Cloud Firestore. Kao što je rečeno ranije u radu, Cloud Firestore nije klasična *SQL* baza podataka, stoga će se korištenje i manipuliranje podacima u dokumentima Cloud Firestore baze podataka vršiti putem metoda odnosno funkcija koje omogućava Firebase. Za izradu Cloud Firestore baze podataka prvo je potrebno inicijalizirati Firebase projekt. Projekt se inicijalizira na početnoj stranici Firebase-a klikom na „Kreiraj projekt“ nakon čega se odabire naziv projekta. To je vidljivo na slikama 4.1 i 4.2.



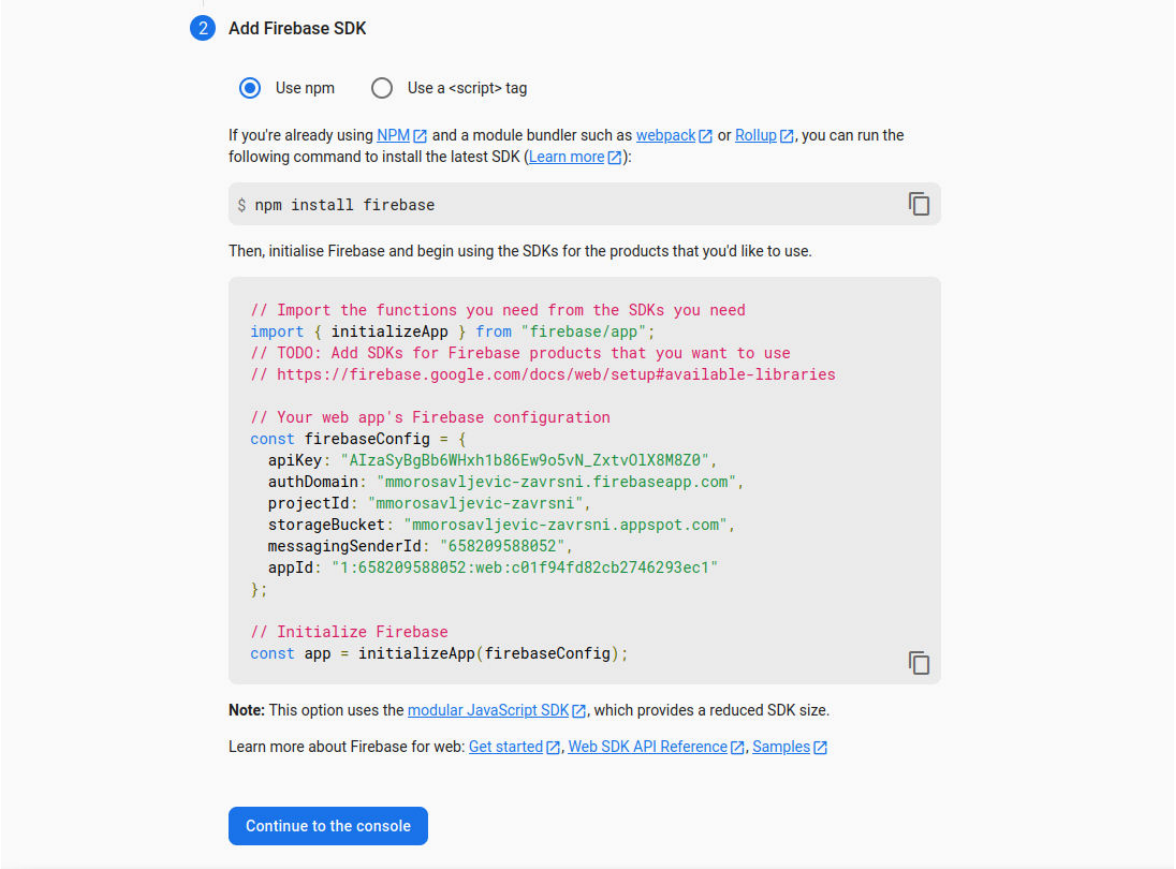
Slika 4.1: Početni ekran Firebase-a



Slika 4.2: Ekran prvog koraka inicijalizacije Firebase projekta

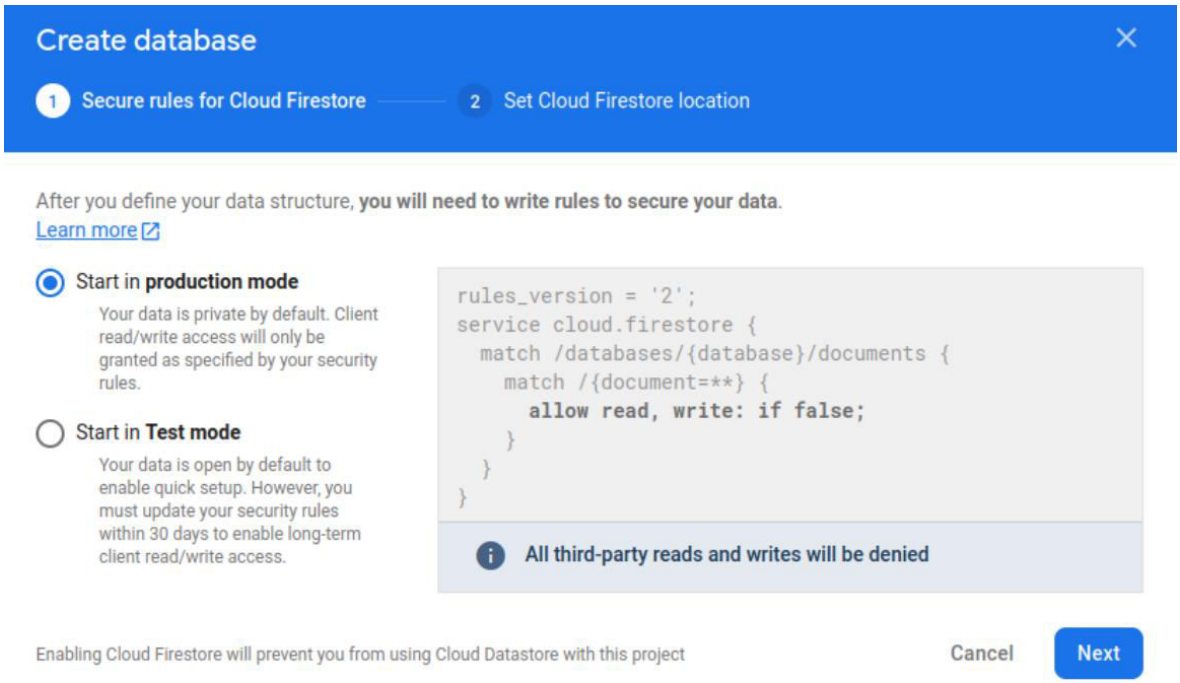
Nakon što je *Firebase* projekt inicijaliziran potrebno je odabrati za koju se platformu aplikacija razvija kako bi se dobio odgovarajući kod za inicijalizaciju Firebase-a

unutar React Native aplikacije. Za odabir platforme Firebase nudi iOS, Android, Web, Unity i Flutter. S obzirom da je aplikacija u ovom radu napisana u React Native-u koji koristi programski jezik JavaScript, za platformu se odabire „*Web* aplikacija“. Nakon odabira platforme dobije se kod s koracima potrebnim za inicijalizaciju *Firebase-a* kao što je vidljivo na slici 4.3. Kod za inicijalizaciju Firebase-a se stavlja u firebase-config.js datoteku u korijenskom direktoriju projekta (engl. *root directory)* te ga uvezemo (engl. *import)* u datoteci koja se prva pokreće prilikom pokretanja projekta, a to je App.js. Razlog zbog kojeg je potrebno uvesti Firebase konfiguraciju u datoteku koja se prva izvršava pri pokretanju aplikacije je taj što Firebase SDK mora biti inicijaliziran prije nego što ga druge komponente u aplikaciji počnu koristiti.

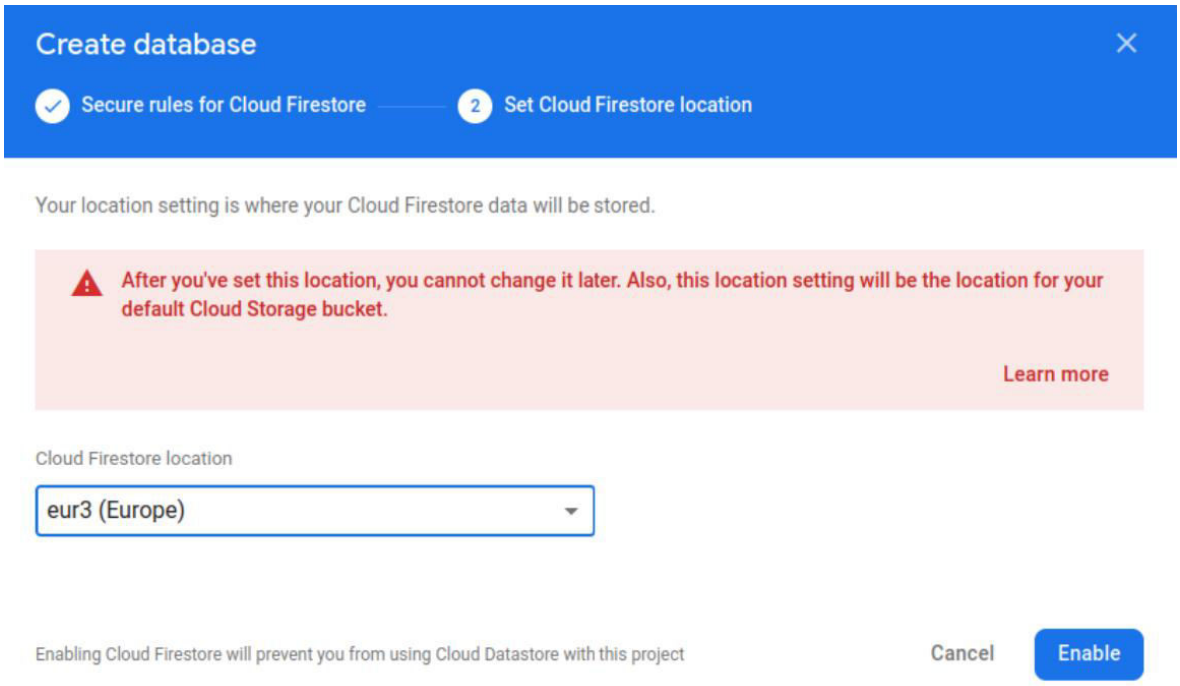


Slika 4.3: Koraci za inicijalizaciju Firebase projekta u JavaScriptu

Nakon odabira platforme, na nadzornoj ploči projekta potrebno je stvoriti Cloud Firestore bazu podataka. Pri stvaranju baze podataka odabire se mjesto servera gdje će se podatci pohranjivati i sigurnosna pravila za manipuliranje podatcima kao što je prikazano na slikama 4.4 i 4.5.

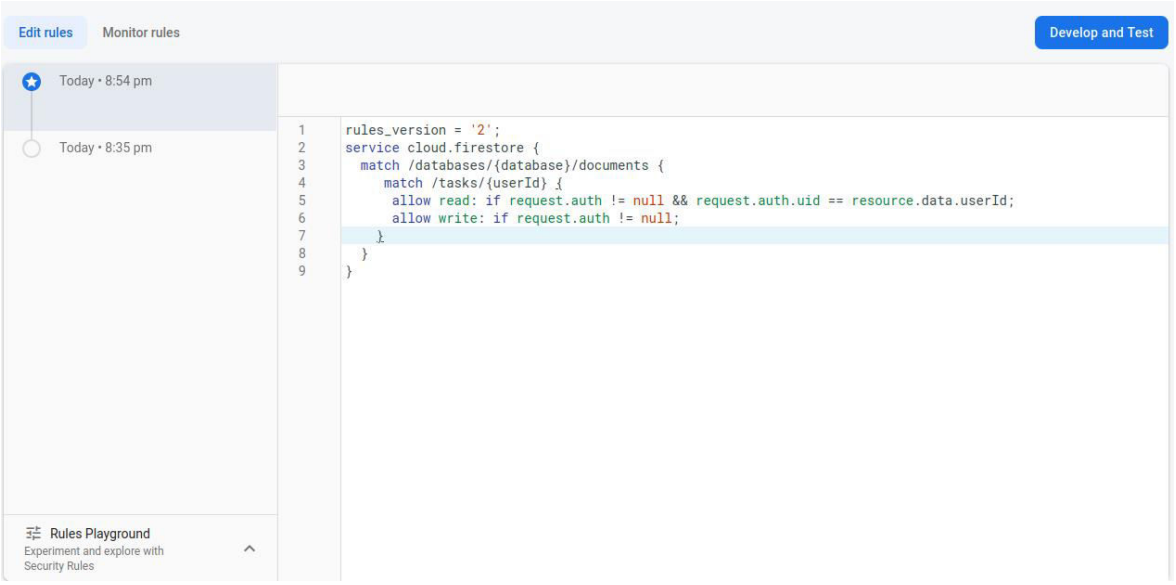


Slika 4.4: Ekran prvog koraka kreiranja Cloud Firestore baze podataka



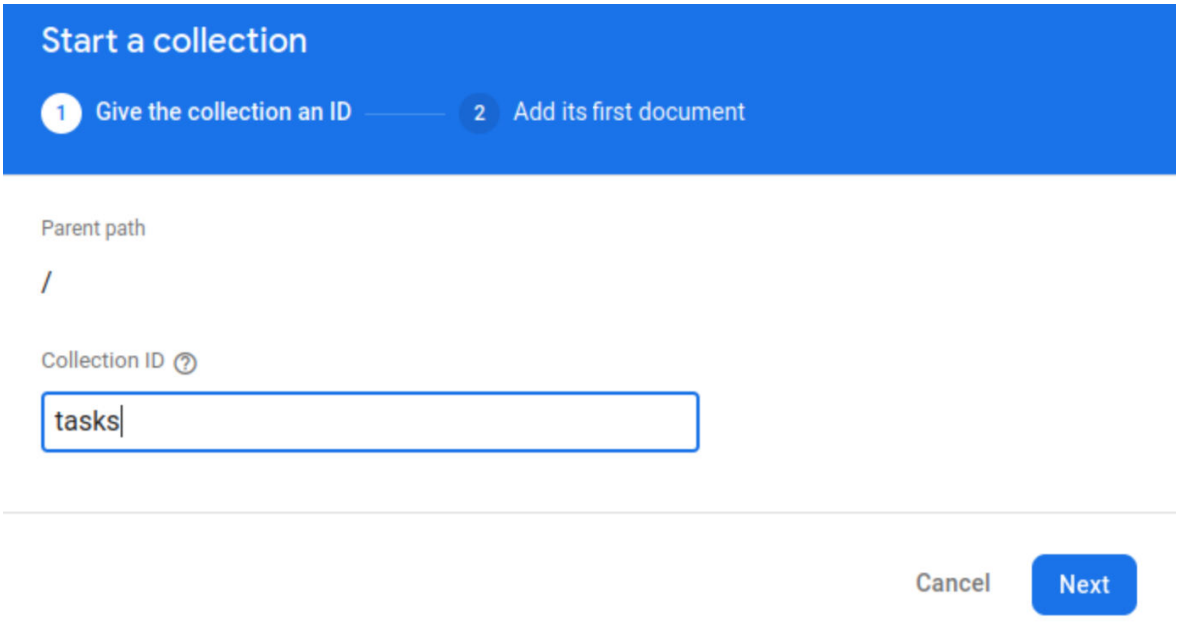
Slika 4.5: Ekran drugog koraka kreiranja Cloud Firestore baze podataka

Za potrebe ovog projekta odabire se server u Europi jer je najbliži i postavljaju se sigurnosna pravila kako bi samo autenticirani korisnici mogli pisati u bazu podataka te kako bi korisnici mogli čitati samo dokumente koje su i sami napravili. Takvo pravilo napisano je i prikazano na slici 4.6.



Slika 4.6: Prikaz postavljenih sigurnosnih pravila u Firebase-u

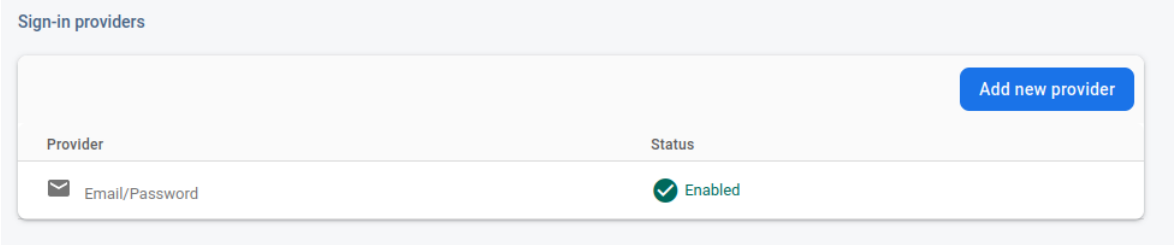
S obzirom da je Cloud Firestore NoSQL baza podataka, nije potrebno raditi nikakvu tablicu u koju bi se mogli pohranjivati samo prethodno definirani modeli podataka. Dovoljno je napraviti jednu kolekciju (engl. *collections)* u koju se mogu pohraniti dokumenti u kojem polja dokumenta ne moraju odgovarati nekom prethodno definiranom modelu podataka već mogu imati različit broj polja, a ta polja ne moraju biti isti tip podatka. Izrada kolekcije „zadatci“ vidljiva je na slici 4.7.



Slika 4.7: Prikaz prvog koraka za kreiranje kolekcije u Firebase-u

Pisanje i čitanje dokumenata vrši se pomoću funkcija koje omogućava Firebase. Kasnije će se u radu prikazati njihova upotreba.

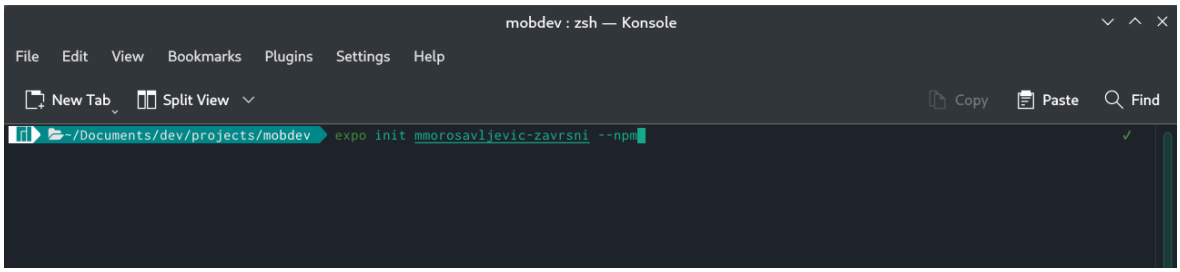
Za autentikaciju korisnika nije potrebno raditi novu kolekciju za korisnike zato što Firebase nudi gotove servise za autentikaciju korisnika. Potrebno je navigirati na „Authentication“ karticu u Firebase konzoli, nakon čega je potrebno dodati novi pružatelj usluga. Firebase za autentikaciju nudi prijavu putem Google-a, Facebook-a, Apple-a, GitHub-a i mnogih drugih, međutim za ovaj je projekt odabrana autentikacija putem e-adrese i lozinke. Dodavanje novog pružatelja usluga vidljivo je na slici 4.8.



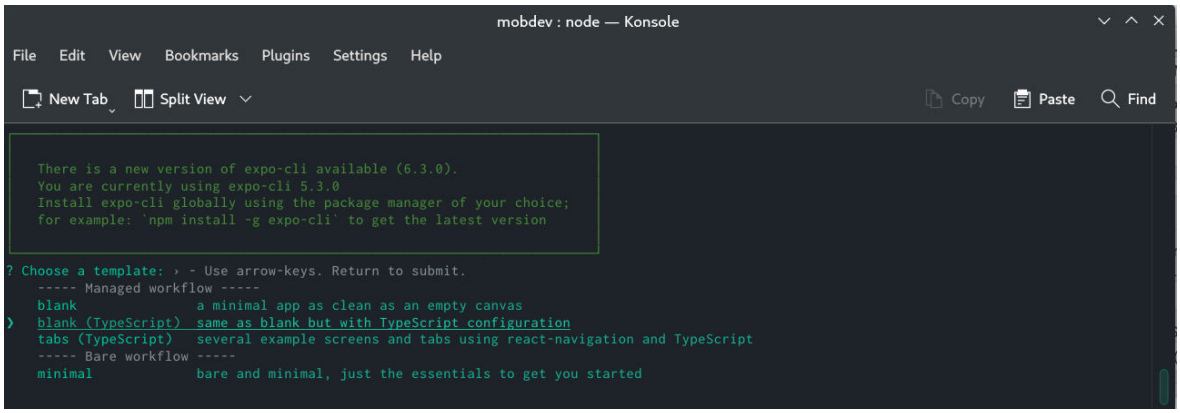
Slika 4.8: Prikaz omogućenih pružatelja usluga u Firebase-u

4.2 Inicijalizacija projekta

Prije nego što se projekt inicijalizira bitno je naglasiti da je na sustavu nužno imati instalirane alate Node i Expo. Nakon što su svi potrebni alati za inicijalizaciju projekta instalirani, potrebno je uključiti naredbeni redak te navigirati u mapu u koju se želi inicijalizirati aplikacija. U željenoj mapi izvršava se naredba prikazana na slici 4.9 nakon čega se odabire predložak aplikacije od kojih su ponuđeni vidljivi na slici 4.10.

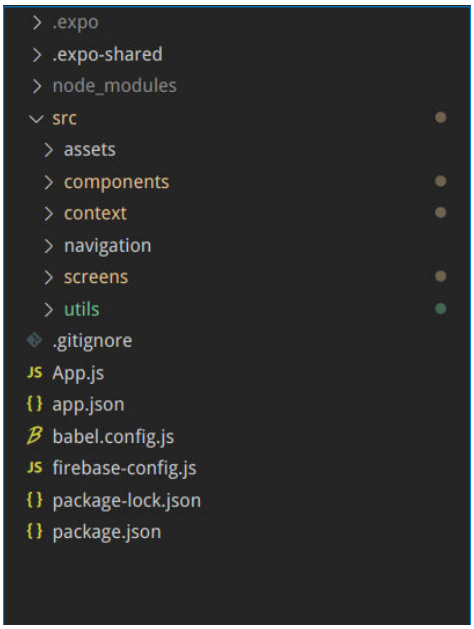


Slika 4.9: Prikaz naredbe za inicijalizaciju Expo projekta u naredbenom retku



Slika 4.10: Prikaz ponuđenih predložaka za inicijalizaciju Expo projekta

Za ovaj projekt odabire se predložak aplikacije upravljane alatom Expo – upravljani tijek rada (engl. *managed workflow)*. Nakon što je projekt stvoren, potrebno je navigirati u mapu projekta te unutar mape otvoriti Microsoft Visual Studio Code. Na slici 4.11 prikazana je struktura projekta. Unutar mape „*assets*“ nalaze se statične datoteke koje su potrebne za vrijeme izvršavanje aplikacije. Primjer datoteka koje se nalaze unutar mape „*assets*“ su fontovi, slike i datoteke potrebne za prevođenje aplikacije na više jezika – „i18n“ datoteke. U mapi „*components*“ nalaze se komponente koje se koriste na jednom ili više mjesta u aplikaciji. Mogu biti ponovno korištene ili se tamo nalaze iz razloga da se aplikacija podijeli na što više komponenti kako bi kod bio čitljiviji i organiziraniji. U mapi „*context*“ nalaze se Context komponente koje se stvaraju da bi varijable ili funkcije unutar njih bile dostupne kroz sve ili više slojeva aplikacije. Primjer funkcija koje moraju biti dostupne u cijeloj aplikaciji su servisi koji omogućavaju komuniciranje s bazom podataka. U mapi „*navigation*“ nalazi se kod koji upravlja navigiranjem između ekrana aplikacije. U React Native-u postoji više načina za upravljanje navigacije između ekrana u aplikaciji od kojih će neki od njih biti kasnije objašnjeni u radu. U mapi „*screens*“ nalaze se datoteke u kojima se dizajnira i piše logika za zasebne ekrane. Sve ostale funkcije, konfiguracije, sučelja (engl. *interface*) i sl., stavljaju se u organizirane podmape u mapi „*utils*“.



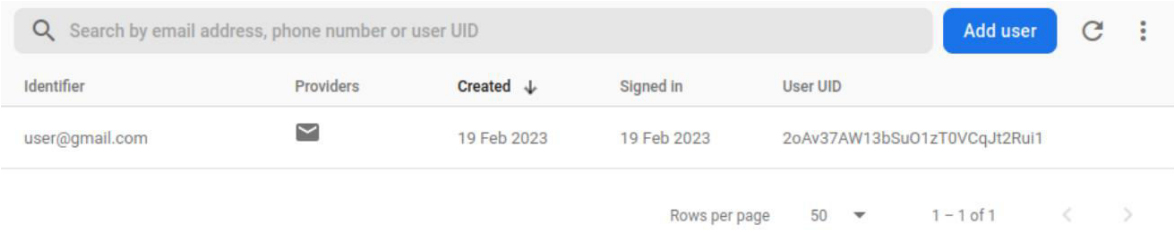
Slika 4.11: Prikaz strukture projekta

4.3 Context

Context je u React-u i React Native-u rješenje za upravljanje stanjima varijabli globalno u aplikaciji [3]. Pomoću Contexta moguće je bilo gdje u aplikaciji pratiti stanje određene varijable zbog čega nije potrebno prosljeđivati neku varijablu kroz puno komponenti kako bi se zapamtilo njeno stanje.

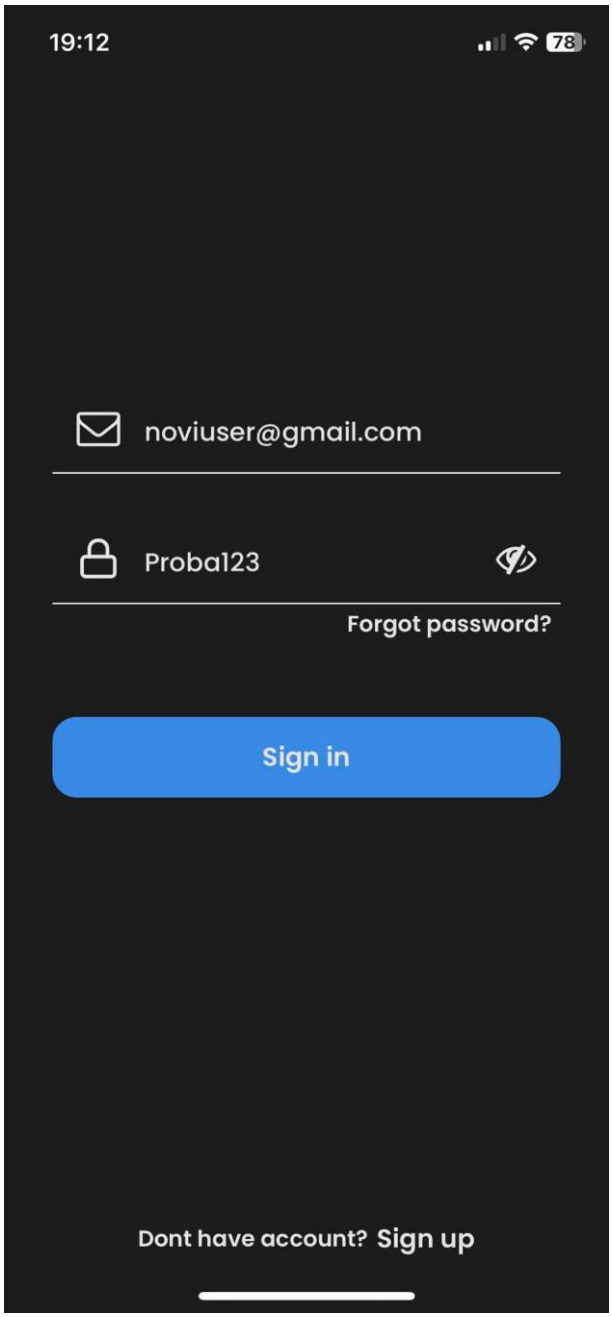
4.3.1 Autentikacija

Autentikacija i autorizacija često su vrlo bitne komponente aplikacije. U ovom projektu potrebno je autenticirati korisnike kako bi svatko imao pristup samo podatcima koje oni sami unesu. Autentikacija u ovom projektu je napravljena pomoću Firebase-a odnosno pomoću ugrađenih SDK funkcija. Kako bi se omogućila autentikacija pomoću e-adrese i lozinke kako je to napravljeno u ovom projektu, potrebno je otvoriti *„Authentication“* odjeljak na nadzornoj ploči projekta. Na novootvorenom ekranu potrebno je dodati novog davatelja usluga za autentikaciju. U slučaju ovog projekta dodaje se pružatelj usluga za prijavu pomoću e-adrese i lozinke. Nakon što je dodan pružatelj usluga pojavljuje se tablica u kojoj se nalaze svi registrirani korisnici. Na slici 4.12. nalazi se prikaz tablice s jednim korisnikom koji je stvoren u svrhe testiranja.

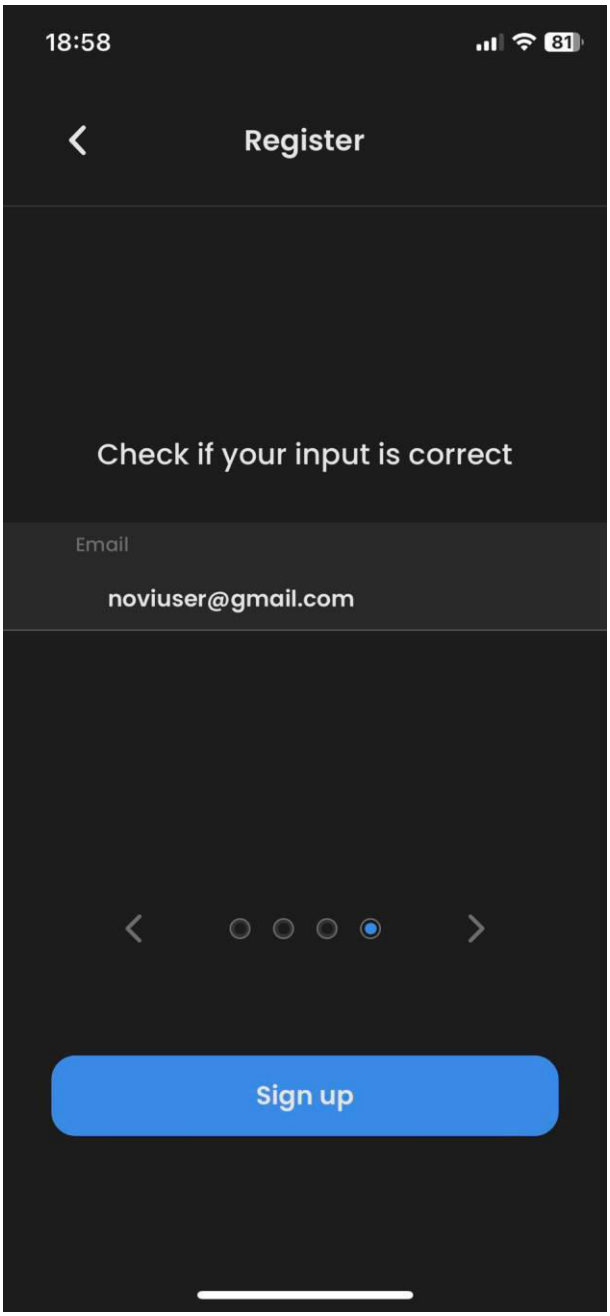


Slika 4.12: Prikaz registriranih korisnika u Firebase-u

Za omogućavanje registracije i prijave korisnika unutar aplikacije potrebno je napraviti korisničko sučelje za registraciju i prijavu. Na slici 4.13 i 4.14 prikazani su ekrani za prijavu i registraciju korisnika. Ekrani za prijavu i registraciju sadrže nekoliko tekstualnih polja za unos koja su dio React Native ugrađenih komponenti. Prije nego što se pošalju podatci funkcijama koje pruža Firebase SDK, potrebno je provjeriti unos korisnika. Najčešće provjere unosa korisnika su provjera odgovara li unesena e-adresa dozvoljenom formatu, ima li lozinka više od određenog broja znakova i sl. Nakon što je unos provjeren i valjan, za registraciju i prijavu šalje se objekt koji sadrži e-adresu i lozinku. Ako je unos ispravan, asinkrona funkcija vraća korisnika i njegove podatke (UUID, e-adresa, lozinka) ili poruku greške u slučaju da korisnik ne postoji pri pokušaju prijave u aplikaciju, u slučaju da korisnik s istom e-adresom već postoji, pri netočno unesenim podatcima pri prijavi ili slično. Pri prijavi korisnik se sprema u Context i na taj je način dostupan kroz cijeli projekt. Funkcija koja je dio Firebase SDK kojoj se prosljeđuje objekt s e-adresa i lozinkom vidljiva je na isječku koda 4.1. Sve ostale funkcije za registraciju drugim pružateljima usluga moguće je pronaći na službenoj dokumentaciji Firebase-a [5].



Slika 4.13: Prikaz ekrana za prijavu u aplikaciju



Slika 4.14: Prikaz ekrana za registraciju korisnika

Programski kod 4.1: Funkcija za prijavu u aplikaciju

```
const login = async (formData) => {
    setAuthState({
        ...authState,
        isLoading: true,
    });
    signInWithEmailAndPassword(auth, formData.email,
formData.password)
        .then((userCredential) => {
            const user = userCredential.user;

            setAuthState({
                ...authState,
                isAuthenticated: true,
                isLoading: false,
                user: user,
            });
        })
        .catch((error) => {
            Alert.alert(error.message);

            setAuthState({
                ...authState,
                isLoading: false,
                user: null,
            });
        });
};
```

4.3.2 Internacionalizacija

Kako bi mobilna aplikacija bila uspješna i kvalitetna vrlo je bitno koristiti prakse koje pružaju najbolje korisničko iskustvo. Jedna od tih praksi je internacionalizacija. Pri izradi aplikacije vrlo je bitno prevesti aplikaciju na više jezika kako bi bila dostupna što većem broju korisnika. U ovom projektu korištena je biblioteka „react-i18next“. Nakon instalacije biblioteke, pri samom pokretanju aplikacije potrebno je podesiti opcije koje biblioteka zahtjeva kao što su glavni jezik, jezik koji se prikazuje ukoliko od ponuđenih u aplikaciji nije na listi korisnikovog glavnog jezika u sustavu i sl. Također, pri inicijalizaciji je potrebno proslijediti objekt koji određuje koliko je jezika dostupno u aplikaciji što je vidljivo na isječku kodu 4.2 Svaki od dostupnih jezika ima svoju JSON datoteku u kojoj su prijevodi pohranjeni u obliku ključ – vrijednost što je vidljivo na isječku kodu 4.3 Ključ označava oznaku pomoću koje se dohvaća prijevod, a vrijednost sadrži tekstualnu vrijednost prijevoda. U svakoj JSON datoteci svi ključevi moraju biti isti kako bi u

aplikaciji bili dostupni neovisno na kojem je jeziku aplikacija, a prijevod se generira ovisno o jeziku koji je odabran u aplikaciji. Ključevi su najčešće napisani na engleskom jeziku zbog konzistencije koda. Na isječku koda 4.4 prikazano je korištenje biblioteke.

Programski kod 4.2: Inicijalizacija biblioteke „react-i18next“

```
i18n.use(initReactI18next).init({
fallbackLng: "gb",
resources: i18nResources,
compatibilityJSON: "v3",
});
```

Programski kod 4.3: Primjer JSON datoteka koje sadrže hrvatski i engleski prijevod

```
{
    "common": {
        "sign-in": "Sign in"
    }
}

{
    "common": {
        "sign-in": "Prijava"
    }
}
```

Programski kod 4.4: Primjer korištenja prijevoda u aplikaciji

```
const { t } = useTranslation();

{
    t("common.sign-in", { ns: "app" });
}
```

4.3.3 Tema

Većina modernih aplikacija sadrži opciju za svijetli i tamni način rada. S obzirom da postoje korisnici koji preferiraju različite načine rada, najbolja opcija pri izradi aplikaciji je podržavanje oba načina rada. Osim varijabli boja za tamni i svijetli način rada, tema može sadržavati različite fontove, primarnu boju koju korisnik preferira ili slično. Iz razloga što tema mora biti dostupna na svim ekranima kroz cijelu aplikaciju, tema se sprema u Context. Pomoću useColorScheme funkcije koja je ugrađena u React Native može se vidjeti preferira li korisnik svijetli ili tamni način rada. Na temelju toga inicijalno se

postavlja u način rada kojeg korisnik preferira odnosno koji je trenutno aktivan na sustavu mobilnog uređaja. Kasnije je u aplikaciji moguće promijeniti temu te se odabir sprema u lokalnu memoriju uređaja. Pri svakom pokretanju aplikacije provjerava se ima li korisnik preferiran način rada spremljen u lokalnoj memoriji, ako nema, aplikacija radi u načinu rada koji je trenutno aktivan na korisnikovom sustavu. Primjer koda provjere načina rada odnosno teme vidljiv je na isječku koda 4.5.

Programski kod 4.5: Prikaz spremanja vrijednosti u varijablu ovisno o odabranom načinu rada

```
const colorScheme = useColorScheme();
const [mode, setMode] = useState(colorScheme);

useEffect(() => {
    getStringValue("@colorScheme").then((val) => {
        if (!val) return;

        setMode(val);
    });
}, []);

const theme = {
    colors: {
        background: mode === "dark" ? "#1C1C1C" : "#E6E6E6",
        card: mode === "dark" ? "#282828" : "#E1E1E1",
        text: mode === "dark" ? "#E6E6E6" : "#2C2C2C",
    },
};
```

4.4 Navigacija

Jako bitan i neizbježan koncept pri izradi mobilnih aplikacija je upravljanje odnosno navigacija ekranima (engl. *screen navigation*). U ovom projektu korištena je biblioteka React Navigation. To je ujedno najpopularnija i biblioteka koju tim Meta najviše promovira za korištenje za upravljanje ekranima u React Native aplikaciji. Postoji nekoliko način za upravljanje između različitih ekrana od kojih su u ovoj aplikaciji korišteni Stack navigation i Drawer navigation. Stack navigation je najčešći oblik navigacije u mobilnim aplikacijama, a funkcionira na način da se ekrani stavljaju jedan na drugi. Može se usporediti sa *stog* tipom podataka što znači da ako se otvori nekoliko ekrana, do prvog ekrana se može doći nakon što se zatvore svi ekrani koji su otvoreni prije njega. Svi ekrani ispod glavnog odnosno trenutno aktivnog ekrana su i dalje aktivni samo što nisu vidljivi. U ovoj aplikaciji Stack navigator korišten je za sve ekrane koji se nalaze u „Postavke“

ekranu, za početni ekran i za ekran koji služi za dodavanje novog zadatka. U isječku koda 4.6 vidljiv je kod za navigaciju između početnog ekrana i ekrana za dodavanje novog zadatka.

Programski kod 4.6: Stack navigacija između home i add task screena

```
const HomeStackNavigator = createStackNavigator();

const HomeStack = () => {
    return (
        <HomeStackNavigator.Navigator
initialRouteName="HomeListScreen">
            <HomeStackNavigator.Screen
                name="HomeListScreen"
                component={HomeScreen}
                options={{
                    header: () => null,
                    headerShown: false,
                }}
            />
            <HomeStackNavigator.Screen
                name="AddTask"
                component={AddTaskScreen}
                options={{
                    header: () => null,
                    presentation: "modal",
                    gestureEnabled: true,
                }}
            />
        </HomeStackNavigator.Navigator>
    );
};
```

Drugi način navigacije koji je korišten u aplikaciji naziva se Drawer navigation. Takav način navigacije koristi se kada u aplikaciji postoji više ekrana kojima bi korisnički pristup trebao biti jednostavan. Drawer navigator se nalazi s lijeve strane ekrana. U početnom stanju može biti skriven ili vidljiv, što znači da kada je aktivan prikazuje se iznad svih ekrana zauzimajući pola širine ekrana osim ako je podešeno drugačije. Više o upravljanju ekrana u React Native aplikaciji moguće je pronaći na službenoj dokumentaciji React Navigation-a [6].

4.5 Obavijesti

U mobilnim aplikacijama postoje dvije vrste obavijesti, a to su *Push Notifications* i *Scheduled (Local) Notifications. Push Notifications* su obavijesti koje korisniku dolaze s udaljenog poslužitelja (engl. *remote server),* a to su npr. obavijesti o novoj zaprimljenoj poruci ako se radi o aplikaciji za dopisivanje. *Scheduled Notifications* su obavijesti koje su napravljene lokalno na uređaju i izvršavaju se nakon zadanog intervala. Na primjer, u aplikaciji je moguće podesiti da se nakon što korisnik ugasi aplikaciju, izvrši obavijest odnosno podsjetnik da korisnik ponovno upali aplikaciju nakon nekog vremena. U ovom radu korištene su *Scheduled Notifications* koje korisnik sam stvara u obliku podsjetnika za određene zadatke.

4.6 Izrada

Lokalne obavijesti u ovoj su aplikaciji potrebne kako bi korisnik prilikom izrade zadatka mogao izraditi podsjetnik odnosno obavijest u odabrano vrijeme. Za sve vrste obavijesti potreban je pristanak korisnika. Upit za pristanak za slanje obavijesti korisniku izvršava se pomoću asinkrone funkcije koju je u ovom radu potrebno izvršiti pri samom uključivanju aplikacije. Iz tog razloga funkcija se stavlja u Context koji će se izvršiti prije nego što se pojavi bilo koji ekran aplikacije. Funkcija s kojom se korisnika pita za dopuštenje za slanje obavijesti prikazana je na isječku koda 4.7. [4]

Programski kod 4.7: Funkcija za upit korisnika za dopuštenje za slanje obavijesti

```
const registerForPushNotificationsAsync = async () => {
    let token;
    if (Device.isDevice) {
        const { status: existingStatus } =
            await Notifications.getPermissionsAsync();
        let finalStatus = existingStatus;
        if (existingStatus !== PermissionStatus.GRANTED) {
            const { status } = await
Notifications.requestPermissionsAsync();
            finalStatus = status;
        }
        if (finalStatus !== PermissionStatus.GRANTED) {
            alert("Failed to get push token for push
notification!");
            return;
        }
        token = (await
Notifications.getExpoPushTokenAsync()).data;
    } else {
        alert("Must use physical device for Push
Notifications");
    }
    if (Platform.OS === "android") {

Notifications.setNotificationChannelAsync("default", {
            name: "default",
            importance:
Notifications.AndroidImportance.MAX,
            vibrationPattern: [0, 250, 250, 250],
            sound: true,
            lightColor: "#FF231F7C",
            lockscreenVisibility:
Notifications.AndroidNotificationVisibility.PUBLIC,
            bypassDnd: true,
        });
    }
    return token;
};
```

Nakon upita korisnika za slanje obavijesti potrebno je izraditi funkcije za izradu obavijesti za zadatak, dohvat i brisanje obavijesti za određeni zadatak. Funkcije su prikazane na isječcima koda 4.8, 4.9 i 4.10.

Programski kod 4.8: Funkcija za izradu obavijesti za određeni zadatak

```
const createScheduledNotification = async ({
taskId, label, date }) => {
    const trigger = date;
    try {
        await
Notifications.scheduleNotificationAsync({
            content: {
                title: "Zavrsni rad",
                body: label,
                data: {
                    taskId: taskId,
                    contentAvailable: 1,
                },
                sound: true,
            },
            trigger,
        });
    } catch (error) {
        Alert.alert(error);
    }
};
```

Programski kod 4.9: Funkcija za dohvat obavijesti za određen zadatak

```
const getNotificationsForTaskId = async (id) => {
    try {
        const notifications =
            await
Notifications.getAllScheduledNotificationsAsync();
        const filteredNotifications =
notifications.filter(
            (n) => n.content.data.taskId == id
        );
        return filteredNotifications;
    } catch (error) {
        Alert.alert(error);
    }
};
```

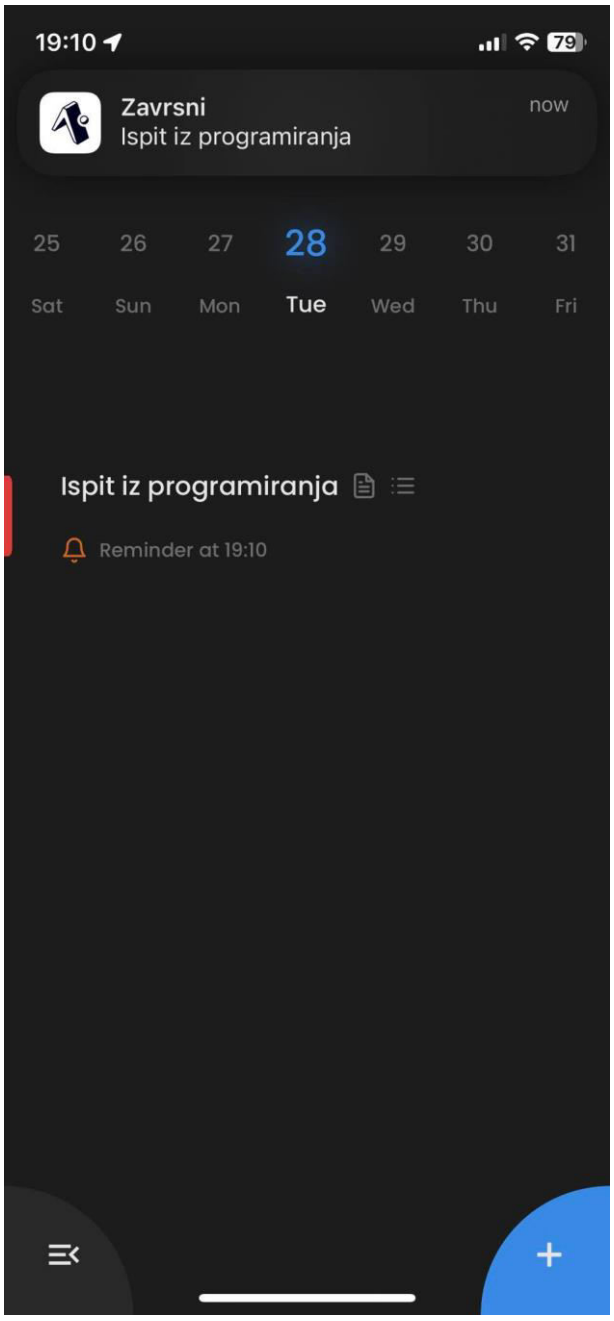
Programski kod 4.10: Funkcija za brisanje obavijesti s određenim id-em

```
const cancelNotification = async (id) => {
    try {
        await
Notifications.cancelScheduledNotificationAsync(id);
    } catch (error) {
        Alert.alert(error);
    }
    return;
};
```

Napisane funkcije koriste se pri izradi zadatka ako je korisnik postavio podsjetnik za zadatak i pri brisanju ili uređivanju određenog zadatka.

4.6.1 *Prikaz*

Na slici 4.15 prikazana je testna obavijest koja je izvršena alatom Expo Push Notification Tool. Taj je alat u procesu izrade aplikacije korišten u svrhe testiranja izgleda i ponašanja obavijesti.



Slika 4.15: Prikaz obavijesti poslane alatom „Expo push notification tool“ na početnom ekranu aplikacije

4.7 Zadaci

4.7.1 Izrada

Izrada zadataka kreće s definiranjem funkcija u Context-u u kojem se nalaze sve funkcije za manipuliranje zadatcima, a to su funkcije za stvaranje, brisanje i uređivanje zadataka. Funkcije pomoću kojih komuniciramo s bazom podataka te spremamo stvorene zadatke u bazu dio su paketa Firebase SDK-a. Na isječcima koda 4.11 i 4.12 prikazane su funkcije za stvaranje i brisanje zadataka. Nakon što su funkcije definirane, one se koriste

odnosno izvršavaju nakon što korisnik u aplikaciji izradi novi zadatak ili nakon što ga obriše na glavnom ekranu aplikacije.

Programski kod 4.11: Funkcija za dodavanje zadatka u bazu

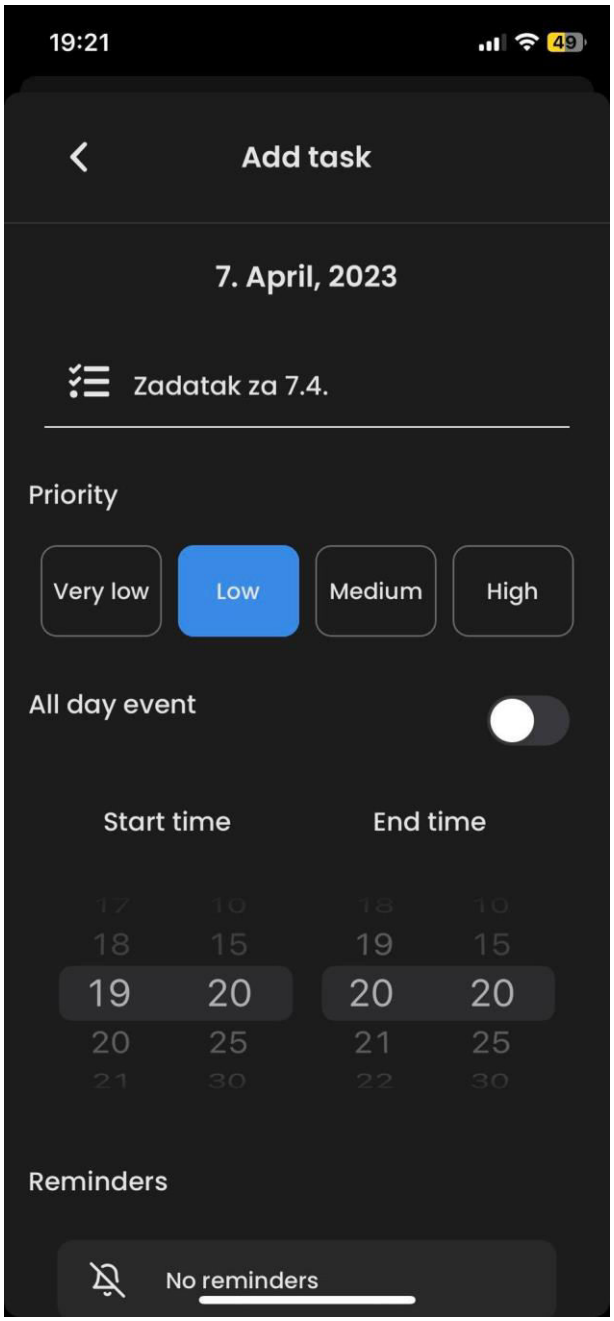
```
const addTask = async (data) => {
    try {
        setTasks([...tasks, data]);


        await addDoc(collection(db,
"tasks"), obj);
    } catch (error) {
        Alert.alert(error);
    }
};
```

Programski kod 4.12: Funkcija za brisanja zadatka iz baze

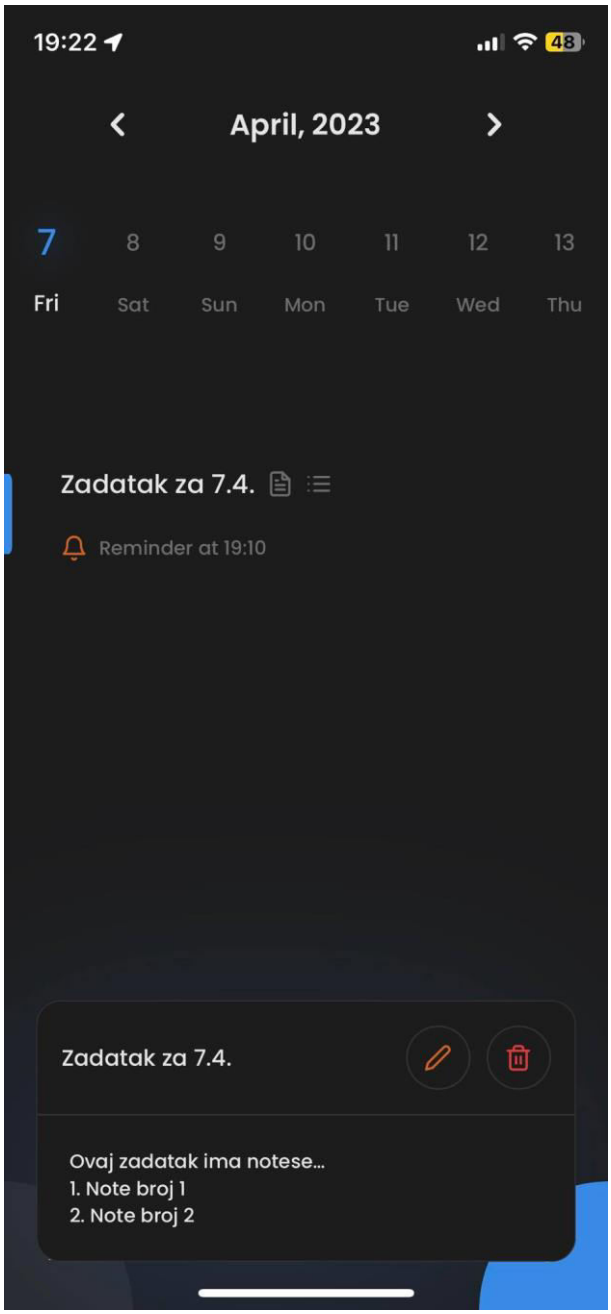
```
const deleteTask = async (id) => {
    try {
        const docRef = doc(db, "tasks",
id);


        await deleteDoc(docRef);
    } catch (error) {
        console.log(error);
    }
};
```

4.7.2 Prikaz

Nakon što su definirane funkcije za manipuliranje podacima, potrebno je napraviti formu koja će korisnicima omogućiti izradu zadatka. Forma je u ovom projektu napravljena pomoću Formik biblioteke. Kako bi korisničko iskustvo bilo što bolje, forma je napravljena na način da su pri samom vrhu polja koja su nužna za izradu zadatka. Forma sadržava polja kao što su naziv zadatka, vrijeme kada zadatak počinje i završava, opcija koja omogućava korisniku dodavanje podsjetnike i sl. Na slici 4.16 prikazan je dio forme pomoću koje korisnik stvara novi zadatak, a na slici 4.17 prikazan je izrađen zadatak na glavnom ekranu aplikacije.



Slika 4.16: Zaslon za izradu zadatka



Slika 4.17: Prikaz odabranog zadatka

4.8 Priprema aplikacije za objavljivanje na Google Play Store

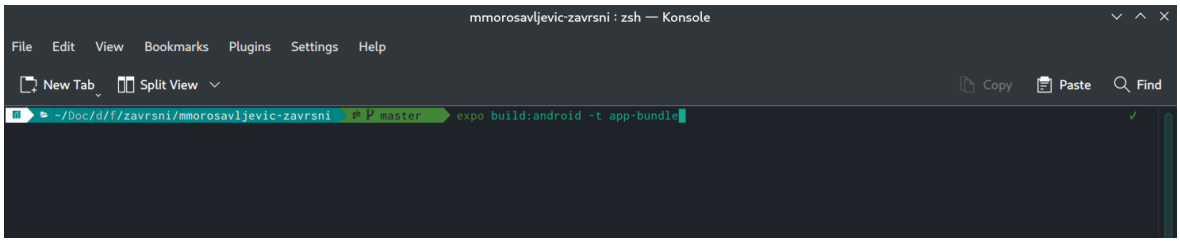
U ovom projektu za izradu aplikacije korišten je React Native, što znači da je aplikacija namijenjena i za iOS i za android. Iako se aplikacija istovremeno razvija za obje platforme, treba uzeti u obzir da te dvije platforme nisu iste i imaju drugačije načine na koje prikazuju određene komponente. Iz tog razloga React Native sadrži ugrađenu komponentu Platform pomoću koje je moguće provjeriti izvršava li se aplikacija na iOS-u ili na Androidu. Ovisno o tome na kojoj se platformi aplikacija izvršava moguće je prikazati drugačije komponente. Prije objave aplikacije na Google Play Store važno je

aplikaciju prilagoditi Android sustavu kako bi korisničko iskustvo bilo što bolje, a to je moguće postići prilagodbom komponenti kao što je vidljivo na programskom kodu 4.13.

Programski kod 4.13: Prikaz ispitivanja aktivnog operativnog sustava

```
if (Platform.OS === "android") {
    // Render a component that can't be scrolled
    <View>
        ...content
    </View>;
} else if (Platform.OS === "ios") {
    // Render a component that can be scrolled
    <ScrollView>
        ...content
    </ScrollView>;
}
```

Nakon što je aplikacija prilagođena, da bi se objavila na Google Play Store potrebno je napraviti Google Play Developer profil koji se može registrirati na Google Play Developer Console stranici. Nadalje, potrebno je izraditi novu aplikaciju u Google Play konzoli u koju se moraju dati podatci kao što su: ime aplikacije koja se objavljuje, opis, snimka zaslona aplikacije (engl. *screenshot*), verzija itd. Kada je aplikacija u Google Play konzoli registrirana potrebno je generirati ključ za potpisivanje (engl. *signing key*) pomoću kojeg će se aplikacija ovjeriti i bez kojeg preuzimanje aplikacije s Google Play Store-a ne bi bilo moguće. Kada su podatci za registraciju ispunjeni potrebno je priložiti APK datoteku koju je moguće izgenerirati izvršavanjem naredbe u naredbenom retku koji je navigiran u korijenskom direktoriju aplikacije prikazane na slici 4.18. Nakon što je priložena APK datoteka, aplikaciju je potrebno svrstati u jednu od ponuđenih kategorija i podesiti značajke kao što su plaćanja unutar aplikacije ako postoje. Konačno, aplikaciju je moguće objaviti tek nakon što su svi prethodno nabrojani koraci riješeni.



Slika 4.18: Expo naredba za generiranje APK datoteke

5. ZAKLJUČAK

Višeplatformski softverski okviri postaju sve popularniji pri izradi aplikacija, bilo da je riječ o *web* aplikacijama, aplikacijama za stolna računala ili mobilnim aplikacijama. Tehnologija se razvija velikom brzinom, stoga je i programerskim timovima bitno da aplikaciju naprave što kvalitetnije u što kraćem vremenskom periodu. Upravo se u ovom radu pisalo o postupku izrade mobilne aplikacije koristeći softverski okvir React Native. Navedeni su slučajevi u kojima je bolje ili isplativije razvijati aplikaciju višeplatformskim softverskim okvirom i koje su prednosti odnosno nedostatci razvijanja aplikacije takvim softverskim okvirom. Podaci su spremljeni u Cloud Firestore bazu podataka. Cloud Firestore je baza podataka u oblaku razvijena od tvrtke Google. Jako bitna stvar za napomenuti je ta da za korištenje servisa Cloud Firestore baze podataka nije potrebno isprva platiti ništa, već se plaća ako se i kako se aplikacija odnosno njezini korisnici povećavaju. Zbog prethodno rečenih razloga, te tehnologije su trenutno jedne od najpopularnijih tehnologija koje su dostupne. Pružaju programerima fleksibilnost, a korisnicima sigurnost i zaštitu podataka uz odlično korisničko iskustvo.

6. LITERATURA

[1] https://www.ibm.com/topics/mobile-application-development

[2] https://aws.amazon.com/mobile/mobile-application-development/

[3] https://reactjs.org/docs/context.html

[4] https://docs.expo.dev/versions/latest/sdk/notifications/

[5] https://firebase.google.com/docs/auth/

[6] https://reactnavigation.org/

7. OZNAKE I KRATICE

androidTV - Android Television (android televizija)

CSS - Cascading Style Sheets (kaskadni listovi stilova)

HTML - Hypertext Markup Language (hipertekst markup jezik)

iOS - iPhone Operating System (iPhone operativni sustav)

itd. – i tako dalje

JDK – Java Development Kit (Java softverski paket)

JSON – JavaScript Oriented Notation (JavaScript objektna notacija)

LAN – Local Area Network (lokalna mreža)

macOS - Macintosh Operating System (operativni sustav Macintosh)

NoSQL – Not Structured Query Language (ne strukturirani jezik upita)

npr. – na primjer

QR Code – Qucik Response Code (kod brzog odgovora)

SDK – Software Development Kit (softverski razvojni paket)

sl. - slično

SQL – Structured Query Language (strukturirani jezik upita)

tvOS – Television Operating System (televizijski operativni sustav)

UI – User Interface (korisničko sučelje)

UUID - Universal Unique Identifier (univerzalni jedinstveni identifikator)

UX – User Experience (korisničko iskustvo)

8. SAŽETAK

Naslov: React Native mobilna aplikacija za planiranje dnevnih zadataka korisnika

Cilj ovog rada bio je predstaviti mobilnu aplikaciju koja će korisnicima olakšati planiranje dnevnih zadataka ili obaveza u obliku podsjetnika. Zbog lake dostupnosti puno informacija, ljudi su sve više skloni zaboravljanju dnevnih zadataka i obaveza koje imaju. Iz tog razloga u ovom radu napravljena je takva mobilna aplikacija pomoću koje se te obaveze mogu zapisati i za koje se mogu napraviti podsjetnici koji bi ljudima potencijalno smanjili zaboravljanje. Za izradu aplikacije korišten je softverski okvir React Native, a podaci su zapisani na platformi Firebase u Cloud Firestore bazi podataka. Također, opisano je što je to razvoj mobilnih aplikacija i koja su četiri osnovna tipa mobilnih aplikacija. U radu se mogu pronaći neke od funkcija od kojih se mobilna aplikacija sastoji i koje su nužne za rad aplikacije. Objašnjeno je zašto su se određene komponente React Native-a koristile u aplikaciji i zašto su bile najbolji odabir za tu namjenu.

Ključne riječi: mobilna aplikacija, produktivnost, React Native, Expo, JavaScript

9. ABSTRACT

Naslov: React Native mobile app for daily tasks planning

The goal of this paper was to present a mobile application that will make it easier for users to plan daily tasks or obligations in the form of reminders. Due to the easy availability of information, people are more prone to forget their daily tasks and obligations. For such reasons, this mobile application was created. The React Native software framework was used to create the application, and the data was stored on the Firebase platform in the Cloud Firestore database. It also describes what mobile application development is and what the four basic types of mobile applications are. In the paper, you can find some of the functions that the mobile application consists of and which are necessary for the operation of the application. Furthermore, it explains why certain React Native components were used in the application and why they were the best choice for that purpose.

Ključne riječi: mobile application, productivity, React Native, Expo, JavaScript

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 29.3.2023.	MATIJA MOROSAVLJEVIĆ	Morosavljević

U skladu s čl. 58, st. 5 Zakona o visokom obrazovanju i znanstvenoj djelatnosti, Veleučilište u Bjelovaru dužno je u roku od 30 dana od dana obrane završnog rada objaviti elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru u nacionalnom repozitoriju.

Suglasnost za pravo pristupa elektroničkoj inačici završnog rada u nacionalnom repozitoriju

MATIJA MOROSAVLJEVIĆ
ime i prezime studenta/ice

Dajem suglasnost da tekst mojeg završnog rada u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu bude pohranjen s pravom pristupa (zaokružiti jedno od ponuđenog):

a) Rad javno dostupan
b) Rad javno dostupan nakon _____ (upisati datum)
c) Rad dostupan svim korisnicima iz sustava znanosti i visokog obrazovanja RH
d) Rad dostupan samo korisnicima matične ustanove (Veleučilište u Bjelovaru)
e) Rad nije dostupan

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 18.4.2023.

Morosavljević
potpis studenta/ice