Mobilna aplikacija za evidenciju rada djelatnika unutar tvrtke

Suk, Sven

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIHTEHNOLOGIJA

Sveučilišni studij

IZRADA MOBILNE APLIKACIJE ZA EVIDENCIJU RADA DIJELATNIKA UNUTAR TVRTKE

Diplomski rad

Sven Suk

Osijek, 2022.

SADRŽAJ

1. UVOD *1*

1.1. Zadatak diplomskog rada **1**

2. USPOREDBA S VEĆ POSTOJEĆIM RJEŠENJIMA *2*

2.1. Day Off – Absence **2**

2.2. Leave Dates **3**

2.3. Jira Vacation Manager **3**

3. RAZVOJNO OKRUŽENJE *4*

3.1. Programski jezik TypeScript **4**

3.2. React Native **5**

4. PROGRAMSKO RJEŠENJE I PRIMJENA APLIKACIJE *6*

4.1. Kreiranje i podešavanje projekta **6**

4.2. Temeljne i prilagodljive komponente **7**

4.3. Struktura koda **8**

4.4. Spremnik stanja **9**

4.5. Navigacija **10**

4.6. Autentikacija **11**

4.6. Kreiranje i pregled zaposlenika **13**

4.7. Godišnji odmori **17**

4.8. Nagrade **23**

5. ZAKLJUČAK *27*

LITERATURA *28*

SAŽETAK *29*

ABSTRACT *30*

ŽIVOTOPIS *31*

PRILOZI *32*

1. UVOD

Vođenje i praćenje djelatnika unutar tvrtke može zahtijevati ogromnu količinu vremena jer u taj proces ulaze stavke kao što su pregled zaposlenih i kreiranje rasporeda godišnjih odmora. Povećanjem broja zaposlenih dolazi do povećanja obveza koje tvrtka mora ispuniti u određenom vremenskom razdoblju za uspješno poslovanje. Iz toga razloga tvrtke svakodnevno rade na razvijanju rješenja za bolju unutrašnju organizaciju koja bi im pomogla u suzbijanju nepotrebnih vremenskih i novčanih troškova.

Izrada mobilne aplikacije potaknuta je navedenim problemom. Mobilna aplikacija se temelji na ideji da omogući nadređenima efikasan način upravljanja djelatnicima te njihovim zahtjevima za godišnje odmore. Osim toga, uvodi se sustav dodjeljivanja bodova koji se mogu zamijeniti za nagrade. On se uvodi sa svrhom unaprjeđenja rada dijelatnika. Za ostvarenje navedenog, aplikacija ima dvije uloge, administrator i zaposlenik. Administratoru se dodjeljuju prava upravljanja, dok zaposlenik ima samo pravo dodijele bodova i kreiranja zahtjeva za godišnji odmor. Aplikacija se izrađuje korištenjem razvojnog okvira React Native te je dostupna za Android i iOS uređaje. Korišteni programski jezik je TypeScript. Stoga je potrebno imati osnovna predznanja o radu skriptnih jezika kako bi se mogao razumjeti kod.

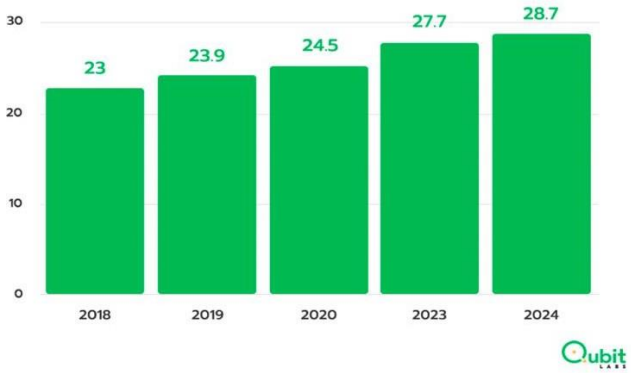
Diplomski rad je strukturiran na način da nakon uvodnog dijela, u drugom poglavlju, obavlja usporedbu s već postojećim sličnim rješenjima, a to obuhvaća navođenje prednosti i mana. U trećem poglavlju opisuju se tehnologije i korišteno razvojno okruženje, a u četvrtom se provodi detaljna analiza programskog koda. Završno poglavlje daje osvrt na cjelokupan rad.

1.1. Zadatak diplomskog rada

Zadatak ovog diplomskog rada je izraditi mobilnu aplikaciju za praćenje godišnjih odmora te nagrađivanja djelatnika unutar tvrtke. Aplikacija treba imati dvije uloge, zaposlenik i administrator. Zaposlenik šalje zahtjev za godišnji odmor koji generira notifikaciju administratoru. Administrator odgovara na zahtjeve. Osim zahtjeva, administrator ima uvid u sve zaposlenike te ih može dodavati, brisati i uređivati. Također, postoji mogućnost kreiranja nagrada koje zaposlenici mogu kupovati kada skupe dovoljan broj bodova. Podaci s kojima aplikacija rukuje dohvaćaju se s udaljenog servera koji je kreiran od strane tvrtke za koju se izrađuje mobilna aplikacija.

2. USPOREDBA S VEĆ POSTOJEĆIM RJEŠENJIMA

Zbog rasprostranjenosti interneta i tehnologije sve je više programera, a time i gotovih rješenja za brojne ideje. Slika 2.1. prikazuje povećanje broja programera koje iznosi gotovo milijun na godišnjoj razini. Zbog navedenog izuzetno je teško osmisliti nešto novo. No, razmišljanje izvan okvira, odnosno razmišljanje na drugačiji način i nekonvencionalno [1] omogućuje pronalazak rješenja koja nisu implementirana.



Slika 2.1. *Broj programera u svijetu [2]*

Kako je ranije navedeno, cilj ovog diplomskog rada je izrada mobilne aplikacije za praćenje godišnjih odmora i nagrađivanje djelatnika unutar tvrtke. Na Trgovini Play i App Storu postoji mnogo sličnih rješenja, a najpoznatija su: Day Off – Absence, Leave Dates i Jira Vacation Manager. Sve navedene aplikacije kreirane su s jednom svrhom, a to je poboljšanje unutrašnje organizacije.

U narednim poglavljima izvršava se detaljna usporedba navedenih aplikacija s aplikacijom koja se izrađuje ovim diplomskim radom.

2.1. Day Off – Absence

Day Off – Absence je interaktivna mobilna aplikacija za upravljanje dopustima zaposlenika, odsutnostima, bolovanjima i godišnjim odmorima u vašoj kompaniji [3]. Dostupna je za iOS i Android uređaje. Kao i aplikacija koja se izrađuje ovim diplomskim radom, sadrži dvije uloge. Administrator poziva zaposlenike u svoj tim te zatim zaposlenici mogu kreirati zahtjeve za godišnje odmore. Aplikacija ima dva načina korištenja, a to su besplatni i plaćeni. U besplatnom načinu moguće je samo pozvati zaposlenike te im odgovarati na zahtjeve, no nije moguće pregledati zahtjeve na kalendaru. U plaćenoj verziji aplikacija pruža brojne mogućnosti, no fokusirana je samo na odobravanje godišnjih odmora te nema mogućnost nagrađivanja djelatnika unutar tvrtke.

2.2. Leave Dates

Leave Dates je aplikacija tvrtke Norton Five kojoj je cilj omogućiti jednostavnije planiranje stvari unutar kompanije. Pomaže kompanijama da izbjegnu situaciju da su ključni članovi tima slobodni istog dana [4]. Za razliku od aplikacije kreirane ovim diplomskim radom, ne sadrži dvije uloge. Aplikacija se temelji na ideji da zaposlenici unesu svoje godišnje odmore i neradne dane. Pomoću tih informacija, generira se dijagram koji služi u svrhu preraspodijele posla. Može se koristiti na mobitelu, tabletu i laptopu. Također, kao i prethodna aplikacija ima plaćenu i besplatnu verziju.

2.3. Jira Vacation Manager

Jira je alat koji omogućuje agilno upravljanje projektima, razvijena je od tvrtke Atlassian [5]. Postoji mnogo proširenja Jire, a jedno od njih je Vacation Manager. Kao i aplikacija koja se izrađuje ovim diplomskim radom, sadrži dvije uloge. Korisnik odabire jednu od dostupnih vrsta izostanaka te Vacation Manager pokazuje koliko slobodnih dana može uzeti. Nakon slanja zahtjeva, administrator odgovara na zahtjev. No, kao i navedene aplikacije nema mogućnost nagrađivanja djelatnika unutar tvrtke i sadrži dvije verzije, plaćenu i besplatnu.

3. RAZVOJNO OKRUŽENJE

Ovo poglavlje pruža detaljan opis razvojnog okruženja te korištenog programskog jezika za izradu mobilne aplikacije.

3.1. Programski jezik TypeScript

TypeScript je programski jezik objavljen 2012. godine od strane Microsoft-a. Kreiran je s jednom svrhom, a to je da popravi nedostatke JavaScript-a. Predstavlja nadskup JavaScript-a, a definira se kao strogo tipirani, objektno orijentirani i kompajlirani jezik [6]. Navedena definicija znači da TypeScript omogućava programeru da deklarira varijable i druge strukture podataka da budu određenog tipa te provjeru valjanosti njihovih vrijednosti. Moguće ga je koristiti u bilo kojem JavaScript okviru, alatu ili biblioteci, a upravo to predstavlja glavni izazov prilikom izgradnje programskih jezika. Osim primitivnih tipova podataka, podržava podatke kao što su: Array, Enums i void. Na slici 3.1. prikazan je primjer TypeScript koda za formatiranje podataka unutar aplikacije. Funkcija za formatiranje vraća objekt tipa EmployeeCreditsFormatted koji ima proizvoljan broj svojstava tipa Credits.

```
You, 2 days ago | 1 author (You)
export interface Credits {
    index: number;
    fullName: string;
    availableCredits: number;
}
You, 2 days ago | 1 author (You)
export interface EmployeeCreditsFormatted {
    [id: string]: Credits;
}
            const formatEmployeeCredits: (credits: EmployeeCredits[]) => EmployeeCreditsFormatted
export const formatEmployeeCredits = (credits: EmployeeCredits[]) => {
    return credits.reduce(
        (prev, curr, index) => ({
            ...prev,
            [curr.id]: {
                index: index + 1,
                fullName: `${curr.first_name} ${curr.last_name}`,
                availableCredits: curr.remaining,
            },
        }),
        {} as EmployeeCreditsFormatted
    );
};
```

Slika 3.1. *Primjer TypeScript koda*

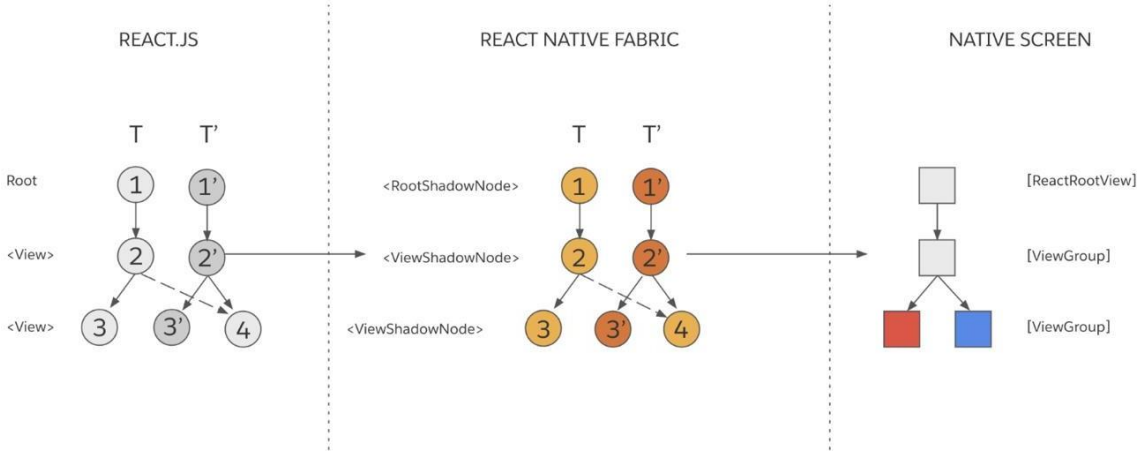
3.2. React Native

React Native je razvojni okvir za izgradnju aplikacija na više platformi kao što su iOS, Android, Android TV omogućavajući programerima korištenje iste baze koda. Temelji se na React-u, a slavu je stekao u izradi mobilnih aplikacija. Radi na način da React elemente sa slike 3.2. rekurzivno smanjuje na osnovne dijelove te zatim kreira Shadow Tree od tih komponenti koji je nepromjenjiv (engl. immutable).

```
function MyComponent() {
  return (
    <View>
      <View
        style={{ backgroundColor: 'red', height: 20, width: 20 }}
      />
      <View
        style={{ backgroundColor: 'blue', height: 20, width: 20 }}
      />
    </View>
  );
}
```

Slika 3.2. *React elementi*

Zatim se većina izračuna izgleda elemenata izvršava u C++, a nakon toga se izvršava zadnja faza koja pretvara Shadow Tree u piksele na ekranu [7]. Svaka promjena stanja unutar aplikacije kreira novi Shadow Tree te se zatim izvršava usporedba s prethodnim i ažuriraju se samo potrebni dijelovi (slika 3.3.).



Slika 3.3. *Rukovanje promjenama u React Native [8]*

4. PROGRAMSKO RJEŠENJE I PRIMJENA APLIKACIJE

4.1. Kreiranje i podešavanje projekta

Za uspješno kreiranje React Native projekta prolazi se kroz službenu dokumentaciju te se podešavaju Xcode i Android Studio kako bi podržali platforme iOS i Android. Nakon uspješnog podešavanja razvojnog okruženja, pomoću naredbe sa slike 4.1. generira se React Native projekt sa svim potrebnim mapama i datotekama te se omogućava korištenje programskog jezika TypeScript.

```
→  ~ npx react-native init Veikei --template react-native-template-typescript
```

Slika 4.1. *Naredba za kreiranje React Native projekta*

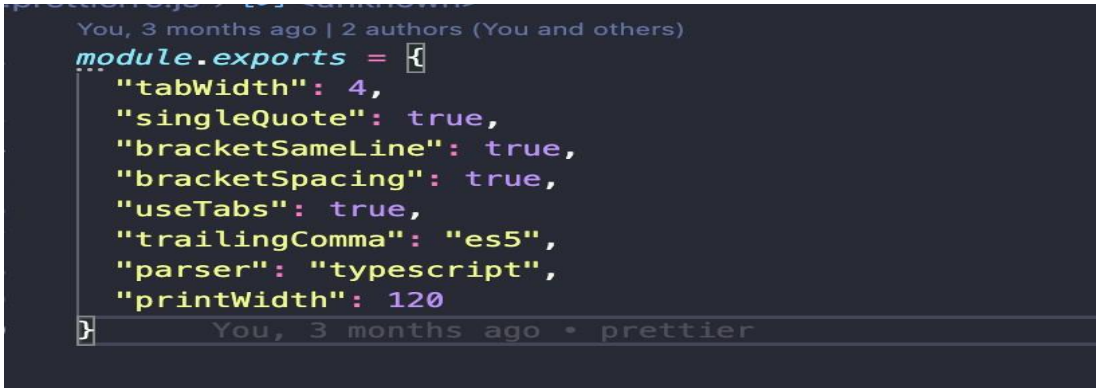
Kreirani projekt pokrećemo na iOS simulatoru korištenjem naredbe yarn ios te na Android emulatoru korištenjem naredbe yarn android. Prije samog početka pisanja koda uvode se pravila koja olakšavaju razvoj programskog rješenja, a za to su zaduženi Prettier i ESLint. Alat ESLint omogućava analizu koda kojeg pišemo u svrhu prepoznavanja problema. Za njegov ispravan rad kreira se konfiguracijska datoteka koja sadržava upute za validaciju koda. Upute sa slike 4.2. predstavljaju najbolje prakse prilikom izrade React i React Native aplikacija utemeljene kroz dugi niz godina.

```
module.exports = {
        root: true,
        extends: '@react-native-community',
        rules: {
                'react/jsx-filename-extension': [2, { extensions: ['.js', '.jsx', '.ts', '.tsx'] }],
                'import/extensions': 0,
                'import/no-extraneous-dependencies': 0,
                'no-return-await': 'off',
                'no-empty-function': 0,
                radix: 'off',
                'lines-between-class-members': ['error', 'always', { exceptAfterSingleLine: true }],
                'jsx-a11y/label-has-associated-control': 'off',
                'import/prefer-default-export': 'off',
                'no-shadow': 'off',
                'no-unneeded-ternary': 'off',
                'react/destructuring-assignment': 'off',
                'import/no-mutable-exports': 'off',
                camelcase: 'off',
                'no-return-assign': 'off',
                'react/jsx-props-no-spreading': 'off',
                'react/require-default-props': 'off',
                'no-param-reassign': 'off',
                'jsx-a11y/click-events-have-key-events': 'off',
                'jsx-a11y/no-noninteractive-element-interactions': 'off',
                curly: [2, 'multi'],
        },
        settings: {
                'import/resolver': {
                        node: {
                                extensions: ['.js', '.jsx', '.ts', '.tsx'],
                        },
                },
        },
};
```

Slika 4.2. *Konfiguracija ESLint-a*

Dok ESLint služi za validaciju napisanog koda, Prettier s druge strane služi za validaciju dosljednosti stila pisanja. On osigurava da sva poravnanja, duljine redaka i sl. budu konzistentni.

Za njegov ispravan rad, također se kreira konfiguracijska datoteka (slika 4.3.).



Slika 4.3. *Konfiguracija Prettier-a*

4.2. Temeljne i prilagodljive komponente

React Native pruža niz ugrađenih temeljnih komponenti spremnih za upotrebu, a neke od njih su: View, Text, Image, TextInput i StyleSheet [9]. Svaka temeljna komponenta prilikom pokretanja aplikacije preslikava se u njenu izvornu verziju ovisno o platformi. Na taj način se omogućuje da React Native aplikacije rade i pružaju osjećaj kao sve druge aplikacije.

U današnje vrijeme, u izradi mobilnih aplikacija izuzetno je bitno kreirati prilagodljive komponente koje se koriste za višestruku upotrebu. Takve komponente ubrzavaju razvoj mobilnih aplikacija i poboljšavaju strukturu projekta. Kreiraju se u slučajevima kada je potrebno isti dio koda koristiti na više mjesta, a najlakše je prepoznati takve situacije ako za aplikaciju postoji dizajn. Prilikom izrade ove aplikacije kreirane su brojne prilagodljive komponente, a neke od njih su gumb, komponenta za unos teksta, tekst koji se skalira u ovisnosti o ekranu i sl.

```
type TextSize = 'small' | 'medium' | 'large' | 'tiny' | 'extraLarge';
You, 1 second ago | 2 authors (You and others)
export interface LocalTextProps {
    text: string;
    textSize?: TextSize;
    font?: TextFontStyle;
    textStyle?: TextStyle | TextStyle[];
}

function LocalText({ text, textStyle, textSize = 'medium', font = 'dxRegular' }: LocalTextProps) {
    const textSizeStyle = React.useMemo(() => getTextSizeStyle(textSize), [textSize]);
    const fontStyle = React.useMemo(() => getTextFontStyle(font), [font]);

    return <Text style={[styles.textColor, fontStyle, textSizeStyle, textStyle]}>{text}</Text>;
}
```

Slika 4.4. *Prilagodljiva komponenta za tekst*

Na slici 4.4. prikazana je komponenta koja se koristi u cijeloj aplikaciji za prikaz teksta. Može primiti nekoliko svojstava kao što su veličina, font i tekst koji se prikazuje. Također, ovisno o

veličini teksta koja je predana izvršava se skaliranje teksta te se na taj način osigurava da se tekst poveća ili smanji ovisno o širini i visini zaslona.

4.3. Struktura koda

Prilikom pisanja React Native aplikacija postoji jednostavan obrazac koji ubrzava razvoj i poboljšava čitljivost koda. Temelj obrasca je kreiranje kontejnera i prezentacijskih komponenti. Uloga kontejnera je da rukuju logikom, pružaju podatke i ponašanja prezentacijskim komponentama te odrađuju pozive servisa. Za razliku od njih, prezentacijske komponente kreiraju izgled i prikazuju podatke koji su im proslijeđeni. Ovakav pristup omogućuje bolju mogućnost ponove upotrebe jer dozvoljava korištenje iste prezentacijske komponente s potpuno različitim izvorima stanja (slika 4.5.).

```tsx
import React from 'react';

import PrezentacijskaKomponenta from './PrezentacijskaKomponenta';

const Kontejner = () => {
    const [counter, setCounter] = React.useState(0);

    const incrementCounter = () => {
        setCounter((curr) => curr + 1);
    };

    return <PrezentacijskaKomponenta value={counter} onPress={incrementCounter} />;
};

export default Kontejner;
```

```tsx
import React from 'react';
import { Text, View } from 'react-native';
import { TouchableOpacity } from 'react-native-gesture-handler';

interface Props {
    value: number;
    onPress: () => void;
}

const PrezentacijskaKomponenta = ({ value, onPress }: Props) => {
    return (
        <View>
            <TouchableOpacity onPress={onPress}>
                <Text>Gumb</Text>
            </TouchableOpacity>
            <Text>{value}</Text>
        </View>
    );
};

export default PrezentacijskaKomponenta;
```

Slika 4.5. *Kontejner i prezentacijska komponenta*

Strukturiranje servisa koji obavlja dohvaćanje podataka sa servera predstavlja veliki izazov jer svaka faza prikazuje drugačiju poruku. Iz toga razloga, unutar ove aplikacije kreirana su tri sloja kroz koja prolazi svaki zahtjev, a to su: sloj učitavanja, pogreške i poziva (slika 4.6.).

```tsx
export interface Service extends AuthService, VacationService, RewardsService {}

const services = serviceNames.reduce(
    (prev, curr) => ({
        ...prev,
        // @ts-ignore
        [curr]: new JSONRPCClient(buildUpURL(curr), '', AxiosInstance),
    }),
    {} as ServiceInstances
);

const client = new ClientMiddleware(services);
const error = new ErrorMiddleware(client);
const service = new LoadingMiddleware(error);

export default service;
```

Slika 4.6. *Strukturiranje servisa*

Svaki zahtjev koji se obavlja korištenjem servisa započinje sa slojem učitavanja koji jednostavno upali indikator učitavanja i proslijedi zahtjev sloju pogreške. Zatim, sloj pogreške prosljeđuje zahtjev do sloja poziva koji pravi zahtjev na server. Nakon što dobije odgovor sa servera, vraća ga sloju pogreške koji rukuje pogreškama i prikazuje određene poruke. Neovisno o uspješnosti zahtjeva, sloj pogreške, vraća odgovor natrag do sloja učitavanja koji gasi indikator. Ovakvom strukturom postiže se čitljivost koda jer se izdvaja rukovanje fazama zahtjeva izvan kontejnera.

4.4. Spremnik stanja

Svaka kompleksnija aplikacija zahtjeva neku vrstu spremnika stanja koji sadržava podatke korištene unutar različitih kontejnera. Postoje mnoga rješenja za ostvarenje navedenog, no unutar ove aplikacije odlučeno je koristiti Redux zbog poznavanja njegovog rada i dobre dokumentacije. On rješava problem upravljanja stanja aplikacije s jednim globalnim objektom pod nazivom Store [10]. Obilježen je trima načelima, a to su: postoji jedan globalni objekt, stanje je dostupno samo za čitanje i promjene stanja je moguće napraviti samo pomoću čistih funkcija. Na slici 4.7. spremnik se inicijalizira na korijenskom nivou aplikacije, no prilikom promjene nekog stanja unutar globalnog objekta ne uzrokuje ponovo izvršavanje svih komponenti nego samo onih koje koriste promijenjeno stanje. Unutar aplikacije, uloga Redux-a očituje se u spremanju podataka o korisniku nakon njegove uspješne prijave jer ti podaci moraju biti dostupni na svim zaslonima. Takvo globalno stanje omogućuje da podatke dohvaćene sa servera koji se ne mijenjaju spremimo i na jednostavan način pročitamo kada su potrebni.

```
const App = () => {
    const { url } = useLinking();

    return (
        <SafeAreaProvider>
            <Provider store={store}>
                <SafeAreaView style={styles.container}>
                    <ErrorWrapper>
                        <AuthentificationWrapper>
                            <Router key={url} url={url} />
                        </AuthentificationWrapper>
                    </ErrorWrapper>
                </SafeAreaView>
                <ToastMessage />
            </Provider>
        </SafeAreaProvider>
    );
};
```

Slika 4.7. *Korijenski nivo aplikacije*

4.5. Navigacija

Jedan od najvećih izazova React Native aplikacija je pružanje ponašanja kao sve ostale aplikacije. Kako bi to bilo izvedivo, React Native tim kreirao je JavaScript biblioteku koja je zadužena za rukovanje navigacijom na potpuno izvoran način. Postoje brojne vrste navigacije, no u ovoj aplikaciji korištena je navigacija stoga koja funkcionira na način da se svaki novi zaslon postavlja na vrh stoga. Prema zadanim postavkama, navigator stoga je konfiguriran tako da ima izvorni izgled i dojam. Na iOS-u novi zasloni klize s desne strane, dok na Androidu koriste zadanu animaciju OS-a.

```
const Stack = createNativeStackNavigator();
const Router = ({ url }: Props) => {
    const userDetails = useAppSelector((state) => state.user.user);
    return (
        <NavigationContainer>
            <Stack.Navigator
                screenOptions={{
                    headerShown: false,
                    gestureEnabled: true,
                    animation: Platform.OS === 'android' ? 'none' : 'default',
                }}>
                {!userDetails && <Stack.Screen name={Routes.Auth}>{() => <AuthRouter url={url} />}</Stack.Screen>}
                {userDetails && (
                    <Stack.Screen name={Routes.RoleRouter}>
                        {() => (
                            <View style={styles.container}>
                                <Header userDetails={userDetails} />
                                <RoleRouter role={userDetails.role} />
                            </View>
                        )}
                    </Stack.Screen>
                )}
            </Stack.Navigator>
        </NavigationContainer>
    );
};
```

Slika 4.8. *Korijenska razina navigacije*

Na slici 4.8. prikazana je korijenska razina navigacije jer NavigationContainer predstavlja početak navigacijskog stabla. On je odgovoran za povezivanje navigatora najviše razine s okruženjem aplikacije [11]. Navigator najviše razine kreira se pozivom funkcije createNativeStackNavigator koja zatim omogućuje postavljanje navigatora i zaslona s određenim svojstvima. Unutar glavnog navigatora, iz Redux-a se čitaju korisnikovi podaci te se ovisno o njima prikazuje određeni navigator. U slučaju ne postojanja podataka prikazuje se navigator za prijavu, a u suprotnome ovisno o ulozi prikazuje se administratorski ili zaposlenikov navigator.

Unutar aplikacije, do pojedinih zaslona dolazi se pomoću navigation objekta. On sadrži metodu navigate koja prima ime zaslona kojem želimo pristupiti (slika 4.9.).

```
const navigation = useNavigation<NavigationProp<RouterParamList, T>>();

const goToLogin = () => {
    navigation.navigate(Routes.Login);
};
```

Slika 4.9. *Navigacija unutar aplikacije*

4.6. Autentikacija

Za ostvarivanje komunikacije između administratora i zaposlenika potrebno je imati udaljenu bazu podataka koja sadrži informacije o korisnicima. Nakon otvaranja aplikacije, korisniku se prikazuje zaslon za prijavu koji od njega zahtjeva email i lozinku (slika 4.10.).

Slika 4.10. *Zaslon za prijavu*

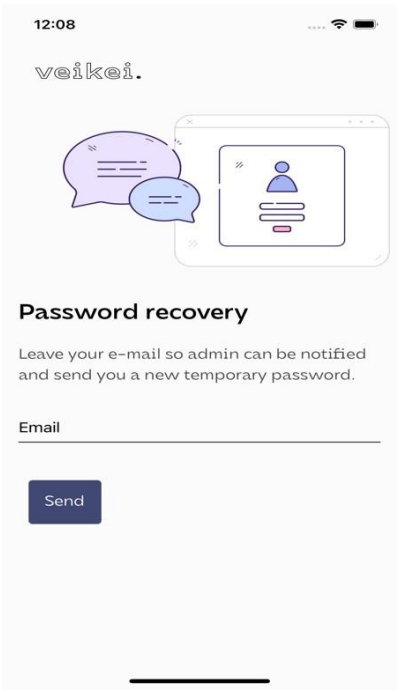
Nakon što korisnik ispuni formu i pritisne gumb za prijavu, podaci se šalju serveru koji obavlja validaciju (slika 4.11.). Ako je zahtjev uspješan, klijentska strana prima token za pristup i osvježavanje. Pristupni token sadrži vjerodajnice za sesiju prijave i identificira korisnika, koristi se za slanje API zahtjeva u ime korisnika [12]. Kako bi identificirao korisnika, postavlja se u zaglavlje svakog API zahtjeva. Nakon postavljanja tokena, izvršava se čitanje i spremanje korisnikovih podataka u Redux te se ovisno o njima prikazuje administratorov ili zaposlenikov navigator.

```
const login = React.useCallback(
    async (loginData: LoginFields) => {
        const { email, password } = loginData;
        await service.login({ email, password });
        const userDetails = await service.readMyDetails();
        dispatch(setUser(userDetails));
    },
    [dispatch]
);
```

Slika 4.11. *Slanje zahtjeva za prijavu i čitanje korisnikovih detalja*

Svaki pristupni token ima svoj životni vijek te ga je potrebno osvježiti kada prestane biti valjan, a to je glavni razlog postojanja tokena za osvježavanje. Za uspješno osvježavanje pristupnog tokena implementiran je presretač zahtjeva (engl. Interceptor). On u slučaju neovlaštenog pristupa pokušava osvježiti token i ponoviti zadnji zahtjev na server. U slučaju uspješnog osvježavanja dobiva se novi pristupni token koji se postavlja u zaglavlje te token za osvježavanje koji se sprema u asinkroni spremnik. Ovakav pristup omogućuje automatsku prijavu korisnika u slučaju postojanja tokena.

Osim prijave, korisnik može promijeniti svoju lozinku, a za to mu je potrebna samo email adresa (slika 4.12.).



Slika 4.12. *Zaslon za obnavljanje lozinke*

Nakon unosa i slanja zahtjeva za promjenu lozinke korisnik zaprima email. Na njegov pritisak

otvara se aplikacija sa zaslonom za unos nove lozinke. Navedeno je ostvareno pomoću dubokog povezivanja (engl. Deep linking). Za uspješan rad dubokog povezivanja kreirane su sheme za Android i iOS koje omogućavaju otvaranje aplikacije pritiskom na url. Osim shema, kreiran je "Custom Hook" koji vraća url ako postoji (slika 4.13.). Hook se sastoji od dva dijela, prvi je pozivanje funkcije getInitialUrl koja se izvršava samo prilikom pokretanja, a drugi je osluškivanje postoji li url dok je aplikacija otvorena. Hook-ovi općenito dozvoljavaju korištenje stanja i drugih značajki React-a bez pisanja klasa, a Custom Hook predstavlja funkciju čije ime započinje s use i poziva druge Hook-ove unutar sebe [13].

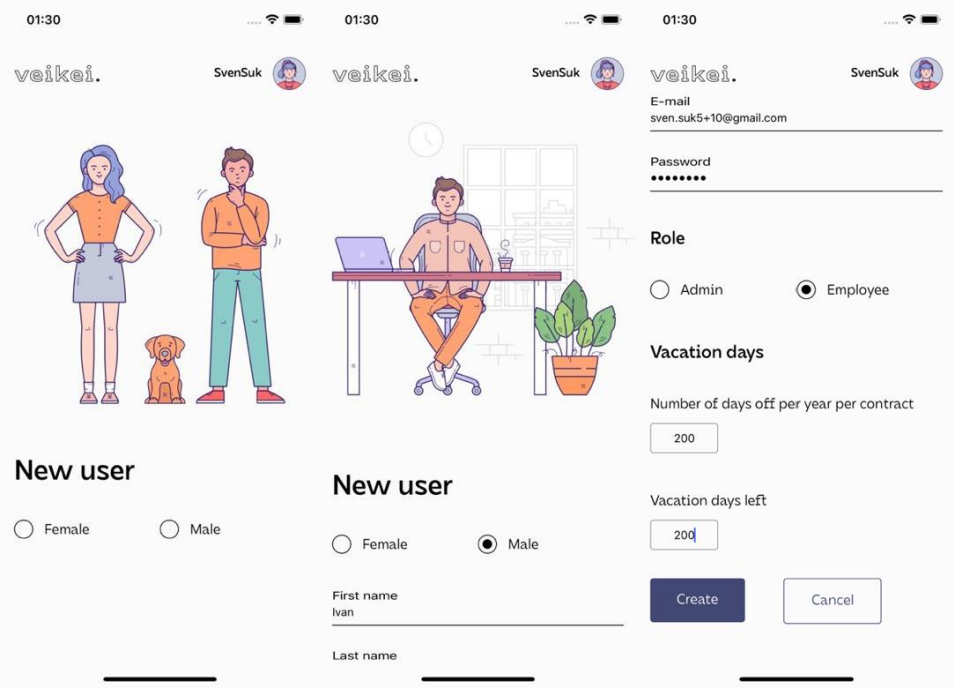
```
import React from 'react';
import { Linking } from 'react-native';

export function useLinking() {
    const [url, setUrl] = React.useState<string>('');
    const [error, setError] = React.useState<unknown>();
    async function getInitialUrl() {
        try {
            const linkingUrl = await Linking.getInitialURL();
            if (linkingUrl) {
                setUrl(linkingUrl);
            }
        } catch (ex) {
            setError(ex);
        }
    }
    React.useEffect(() => {
        function handleOpenUrl(ev: { url: string }) {
            setUrl(ev.url);
        }
        getInitialUrl();
        Linking.addEventListener('url', handleOpenUrl);
        return () => Linking.removeAllListeners('url');
    }, []);
    return { url, error };
}
```

Slika 4.13. *Custom Hook za osluškivanje url*

Promjena lozinke funkcionira na način da se osim nove lozinke šalje jedinstveni identifikator kojeg sadrži url, a on služi serveru za validaciju zahtjeva. Jedina razlika između zaslona sa slike 4.12. i zaslona za potvrdu promjene lozinke je u tekstu koji se prikazuje.

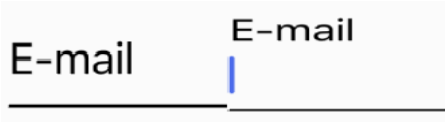
4.6. Kreiranje i pregled zaposlenika

Kako bi bilo moguće koristiti aplikaciju kreira se administratorski račun. Njegovo kreiranje se izvršava direktnim unosom u bazu podataka te ga stoga nije moguće obrisati putem aplikacije. Jedna od njegovih brojnih zadaća je upravljanje zaposlenicima, a upravo to se opisuje ovim poglavljem. Nakon ulaska u aplikaciju, administratorova zadaća je unos zaposlenika.



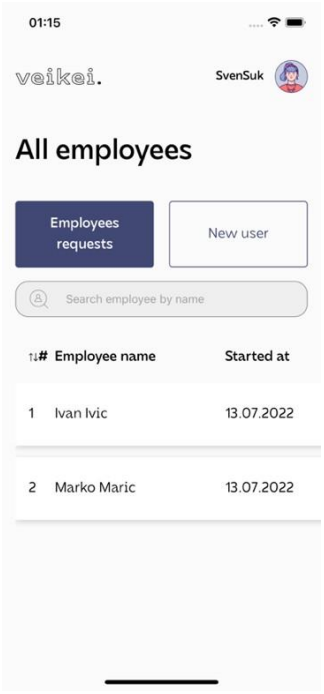
Slika 4.14. *Unos zaposlenika*

Na slici 4.14. prikazan je proces kreiranja zaposlenika. Prvo se odabire spol pomoću dva radijska gumba. Ovisno o spolu prikazuje se fotografija i otvara se forma koju je potrebno popuniti. Za forme se koristi eksterna biblioteka pod nazivom React Hook Form. Ona smanjuje količinu koda kojeg je potrebno napisati te uklanja nepotrebna ponovna renderiranja [14]. Također, pruža validaciju unesenog teksta po pravilima koje programer specificira. Takav pristup smanjuje pogreške jer ne dozvoljava slanje neispravnih podataka serveru te daje korisniku do znanja gdje je pogriješio. Forma se sastoji od animiranih polja za unos teksta, a animacije se izvršavaju u slučaju fokusiranja polja. Kada se polje fokusira, oznaka polja se pomiče po y osi i skalira za određenu vrijednost (slika 4.15.).



Slika 4.15. *Animacija polja za unos teksta*

Nakon ispunjavanja forme, administrator šalje zahtjev serveru za kreiranje novog korisnika. Prolazi se kroz međuslojeve te u slučaju uspješnog zahtjeva, otvara se zaslon gdje administrator vidi sve zaposlenike (slika 4.16.).



Slika 4.16. *Pregled svih zaposlenih*

Ukoliko zaposlenici ne postoje u Redux-u dohvaćaju se sa servera. Zatim se izvršava formatiranje podataka te prikaz u tabličnom obliku. Tablica predstavlja jednu od prilagodljivih komponenata kreiranih za potrebe aplikacije. Ona prima brojna svojstva, a najbitnija su informacije o stupcima i formatirani podaci. Informacije o stupcima daju precizne specifikacije tablici o pojedinom stupcu, a povezuju se s formatiranim podacima po ključu. Na slici 4.17. prikazano je sučelje Column koji može imati proizvoljan broj svojstava tipa CellParams.

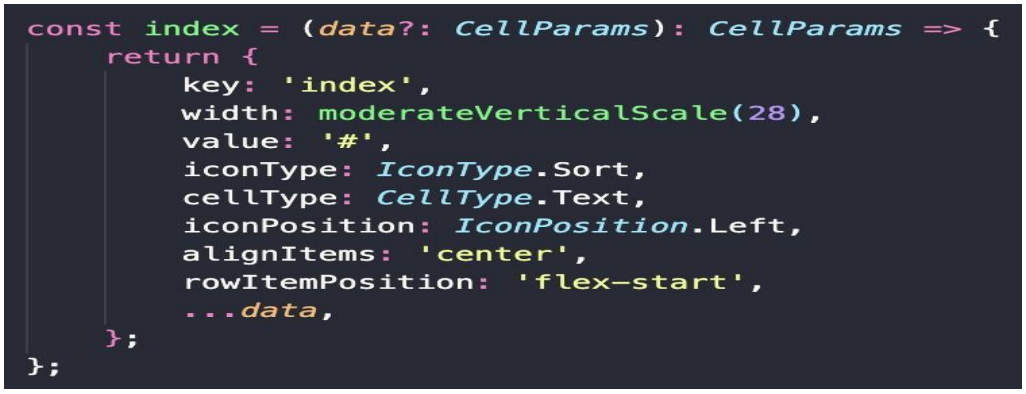
```
You, 2 months ago | 1 author (You)
export interface CellParams {
    key?: string;
    width?: number;
    value?: string;
    isDayjs?: boolean;
    cellType?: CellType;
    iconType?: IconType;
    alignItems?: 'stretch' | 'flex-end' | 'flex-start' | 'center';
    iconPosition?: IconPosition;
    rowItemPosition?: 'stretch' | 'flex-end' | 'flex-start' | 'center';
}

You, 4 months ago | 1 author (You)
export interface Column {
    [key: string]: CellParams;
}
```

Slika 4.17. *Sučelja za stupce tablice*

Nakon definiranja sučelja, za svaku tablicu opisuju se njeni stupci na način da se kreira konstanta tipa Column. Njeni ključevi se moraju podudarati s formatiranim podacima koji se šalju tablici (slika 4.18.).

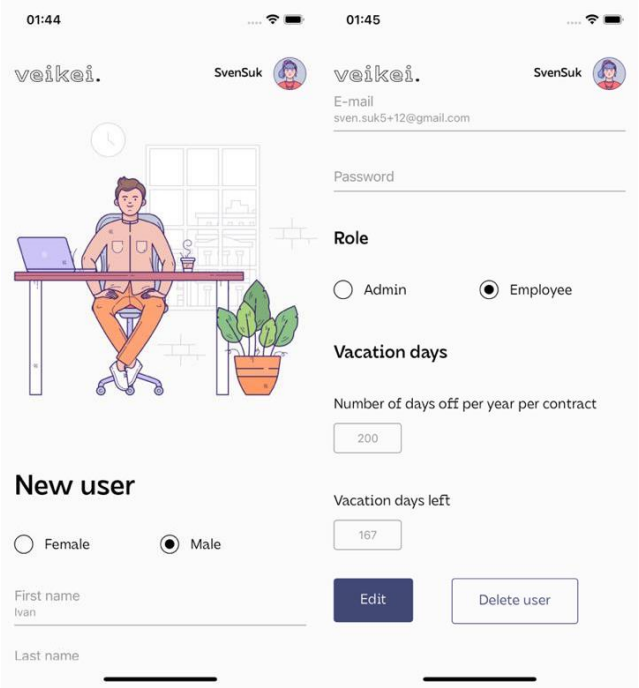
Slika 4.18. *Informacije o stupcima i formatiranje podataka*

Na slici 4.18. za generiranje informacija o stupcima pozivaju se funkcije koje su kreirane kako bi se spriječilo dupliciranje koda, a one jednostavno vraćaju objekt tipa CellParams (slika 4.19.).



Slika 4.19. *Funkcija za kreiranje informacija o stupcu index*

Nakon što tablica zaprimi specifikaciju za stupce i formatirane podatke izvršava se njihovo povezivanje na temelju ključeva. Podaci za koje ne postoji specifikacija ne prikazuju se u tablici. Također, tablica koristi virtualizirani popis čija je uloga poboljšanje performansi. Popis radi na principu da uvijek kreira elemente koji su vidljivi na ekranu te na taj način smanjuje potrošnju memorije. Svaki redak je klikabilan i pritiskom na neki otvara se zaslon gdje su vidljivi godišnji odmori, ali i gumb za uređivanje korisnika. Daljnjom navigacijom dolazi se do zaslona sa slike 4.20.



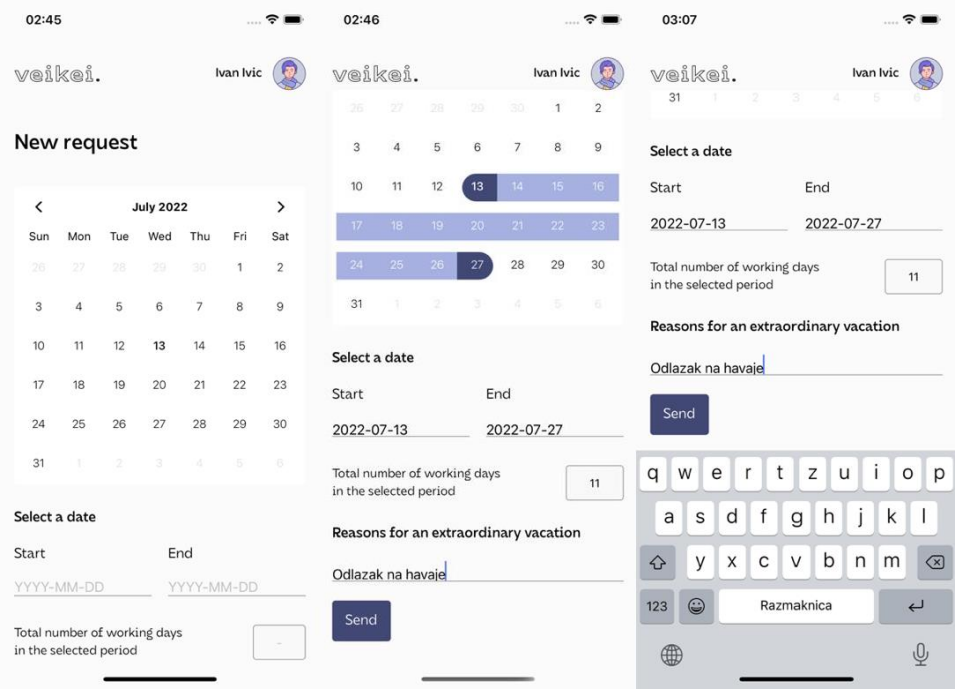
Slika 4.20. *Uređivanje i brisanje korisnika*

Otvaranjem zaslona sa slike 4.20. ne izvršava se zahtjev na server, nego se podaci čitaju iz Redux-a. Za prikaz korisnikovih detalja pobrinula se React Hook Forma kojoj se predaju početne vrijednosti. Osim pregleda, moguće je urediti ili obrisati zaposlenika pritiskom na gumbe. Izvršavanjem bilo koje akcije šalje se zahtjev na server te u slučaju uspjeha ažuriraju se podaci spremljeni u Redux-u.

Nakon kreiranja zaposlenika, omogućen im je ulazak u aplikaciju. Što se tiče navedenih mogućnosti, zaposlenici imaju samo uvid u svoje detalje. Izgled je gotovo identičan onome sa slike 4.20. Jedina razlika je što gumbi za uređivanje i brisanje ne postoje.

4.7. Godišnji odmori

Svaki zaposlenik unutar aplikacije može kreirati zahtjev za godišnji odmor. Prilikom kreiranja zahtjeva ispunjava formu vidljivu na slici 4.21.



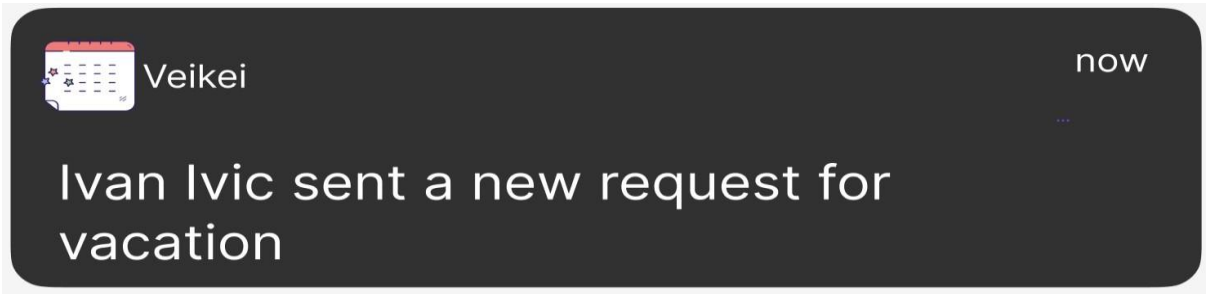
Slika 4.21. *Forma za godišnji odmor*

Kako ne bi morao upisivati početak i kraj godišnjeg odmora, zaposleniku je omogućena interakcija s kalendarom. Prilikom odabira početnog i završnog dana, kalendar generira raspon te također izračunava koliko je radnih dana obuhvaćeno. Za generiranje označenih dana poziva se funkcija getMarkedDates koja popunjava polje u ovisnosti o početku i kraju godišnjeg odmora. Tijekom generiranja dana izvršava se provjera za neradne dane. Nakon toga, poziva se funkcija formatMarkedDates koja formatira podatke u oblik prihvatljiv kalendaru (slika 4.22.). Format koji kalendar očekuje je objekt čiji su ključevi datumi s određenim svojstvima. Kalendaru je potrebno naglasiti koji je dan početni, a koji završni.

```
export const formatMarkedDates = (dates: string[]) => {
    return dates.reduce((acc, curr, index) => {
        if (index === 0) {
            return { [curr]: { startingDay: true, color: colors.darkSlateBlue, textColor: colors.white } };
        } else if (index === dates.length - 1) {
            return { ...acc, [curr]: { endingDay: true, color: colors.darkSlateBlue, textColor: colors.white } };
        }
        return { ...acc, [curr]: { color: colors.dodgerLightBlue, textColor: colors.white } };
    }, {});
};

const isWeekend = (day: number): boolean => {
    return day === 6 || day === 0;
};

export const getBetweenDates = (start: Dayjs, end: Dayjs) => {
    let workingDays: number = 0;
    const daysBetween = [];
    for (let i = 0; start.add(i, 'day') <= end; i++) {
        if (!isWeekend(start.add(i, 'day').day())) {
            workingDays++;
        }
        daysBetween.push(start.add(i, 'day').format(CALENDAR_DATE_FORMAT));
    }
    return { daysBetween, workingDays };
};

export const getMarkedDates = (fromDate?: Dayjs, toDate?: Dayjs): MarkedDatesDetails => {
    let markedDates: MarkedDates = {};
    if (fromDate && toDate) {
        const betweenDates = getBetweenDates(fromDate, toDate);
        markedDates = formatMarkedDates(betweenDates.daysBetween);
        return { markedDates, workingDays: betweenDates.workingDays };
    } else if (fromDate) {
        markedDates[fromDate.format(CALENDAR_DATE_FORMAT)] = {
            startingDay: true,
            color: colors.darkSlateBlue,
            textColor: colors.white,
        };
    }
    return { markedDates, workingDays: 1 };
};
```

Slika 4.22. *Formatiranje podataka u oblik prihvatljiv kalendaru*

Osim kalendara, koristi se forma koja je ponovo implementirana pomoću React Hook Forme u svrhu sprječavanja nepotrebnih renderiranja. Ispunjena forma šalje se serveru na pritisak tipke Send, no prije samoga slanja provjerava se je li zaposlenik ima dovoljan broj dana. U slučaju uspješnog slanja, server šalje push notifikaciju administratoru (slika 4.23.).



Slika 4.23. *Izgled push notifikacije*

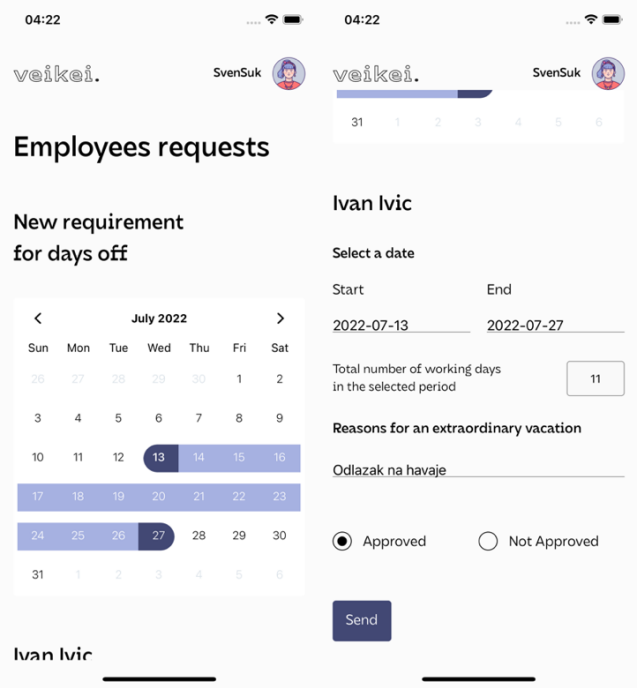
Push notifikacije unutar ove aplikacije funkcioniraju na način da se prilikom prijave, serveru šalje token za notifikacije kojeg on povezuje s korisnikom. Na taj način server u svakom trenutku zna kojem uređaju mora poslati notifikaciju. U slučaju odjave server se obavještava o promjenama te briše token za notifikacije s određenog korisničkog računa.

Za rukovanje notifikacijama koriste se biblioteke Notifee i FCM. One pružaju razvojnim programerima da brzo izgrade notifikacije s jednostavnim API sučeljem [15] .

```
const onMessageReceived = async (message, isForeground) => {
    const { title, body, channelID } = message.data;
    if (message.data?.vacationID && message.data.channelID === 'VACATION' && isForeground) {
        await notifee.displayNotification({
            body,
            title,
            data: message.data,
            android: {
                channelId: channelID,
                sound: 'default',
                smallIcon: 'ic_notification',
                pressAction: {
                    id: 'default',
                    mainComponent: 'veikei',
                },
            },
        });
    }
};

messaging().setBackgroundMessageHandler(async (remoteMessage) => {
    Platform.OS === 'android' && onMessageReceived(remoteMessage, false);
    dangerouslyNavigate();
});
notifee.onBackgroundEvent(async ({ type }) => {
    if (type === EventType.PRESS) {
        dangerouslyNavigate();
    }
});
notifee.onForegroundEvent(async ({ type }) => {
    if (type === EventType.PRESS) {
        dangerouslyNavigate();
    }
});
```

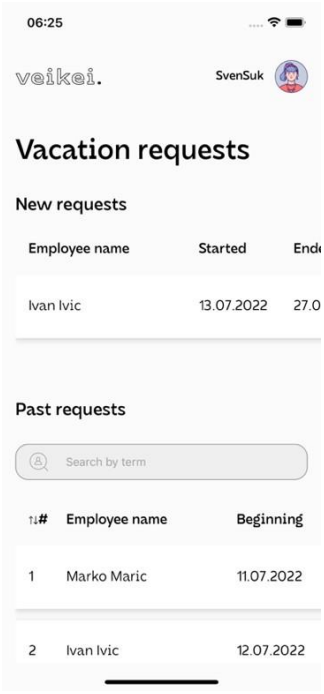
Slika 4.24. *Rukovanje notifikacijama*

Na slici 4.24. prikazane su dvije funkcije za osluškivanje pozadinskih događaja, a razlog tomu su Android uređaji kod kojih se ne poziva funkcija onBackgroundEvent kada je aplikacija ugašena. Sve postavljene funkcije u glavnini obavljaju identičnu stvar, a to je ako je korisnik administrator otvara se aplikacija sa zaslonom za pregled godišnjeg odmora (slika 4.25.).



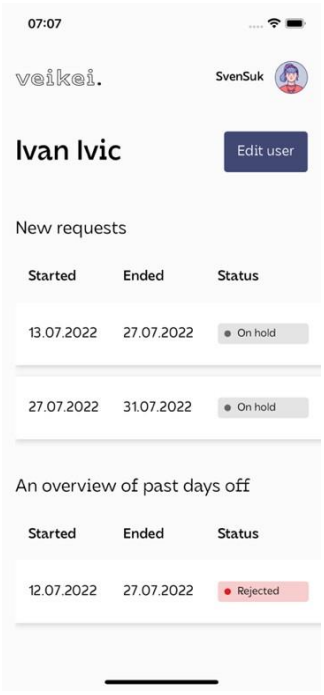
Slika 4.25. *Pregled i odgovaranje na zahtjev*

Za odgovaranje na zahtjev koristi se identična prezentacijska komponenta kao i kod slanja zahtjeva, ali s različitim stanjima. Polja za unos teksta i kalendar su isključeni za uređivanje, a jedino što administrator može napraviti je odgovoriti na zahtjev pomoću radijskih gumbova. No, nije nužno odmah odgovoriti na zahtjev jer postoji zaslon gdje se mogu pregledati svi zahtjevi. Slika 4.26. prikazuje nove i stare zahtjeve koji se dohvaćaju sa servera i prikazuju u tabličnom obliku. Tablice imaju fiksnu visinu od trideset posto i mogu se listati horizontalno i vertikalno. Svaki redak je klikabilan te vodi na detalje o zahtjevu, a u slučaju da na zahtjev nije poslan odgovor otvara se zaslon sa slike 4.25.



Slika 4.26. *Pregled svih novih i starih zahtjeva*

Osim pregleda svih zahtjeva, pruža se i pregled zahtjeva po zaposleniku. Pritiskom na nekog zaposlenika iz liste svih zaposlenika otvara se zaslon sa slike 4.27.



Slika 4.27. *Pregled novih i starih zahtjeva specifičnog zaposlenika*

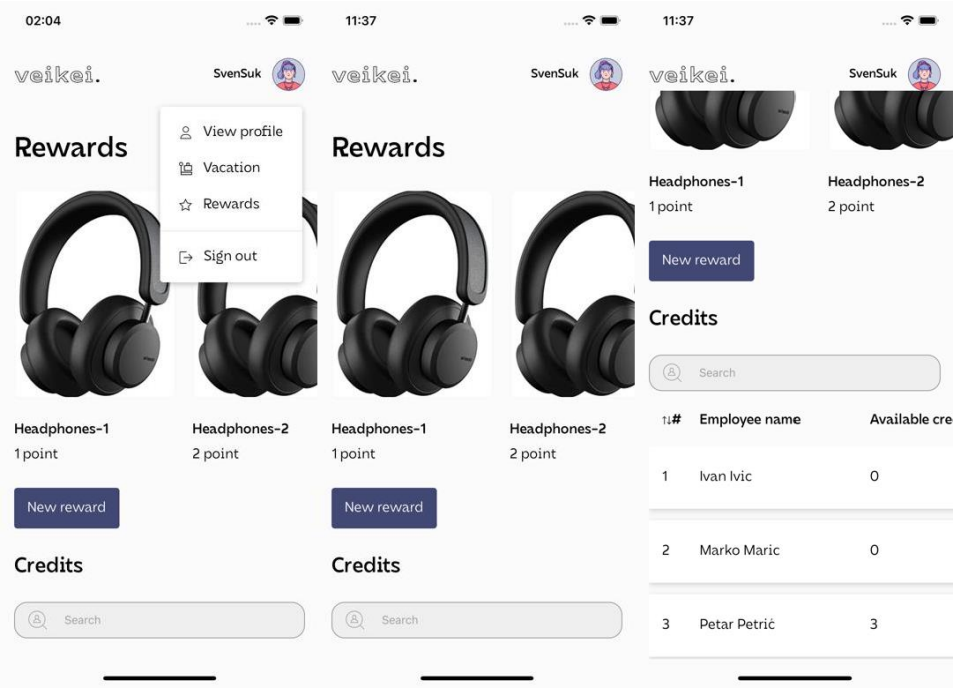
Zaposlenik također ima uvid u svoje zahtjeve te može pratiti status pojedinog. Izgled zaslona na kojemu zaposlenik može pratiti zahtjeve je gotovo identičan onome sa slike 4.26. Jedina razlika je što se u zaglavlju ne nalazi ime korisnika i gumb za uređivanje. Pritiskom na neki zahtjev, zaposlenika se vodi na detalje zahtjeva gdje mu je omogućen pregled i brisanje zahtjeva. Kada

administrator odgovori na zahtjev, zaposlenik zaprima notifikaciju te pritiskom na nju odvodi ga se na zaslon gdje može vidjeti odgovor.

4.8. Nagrade

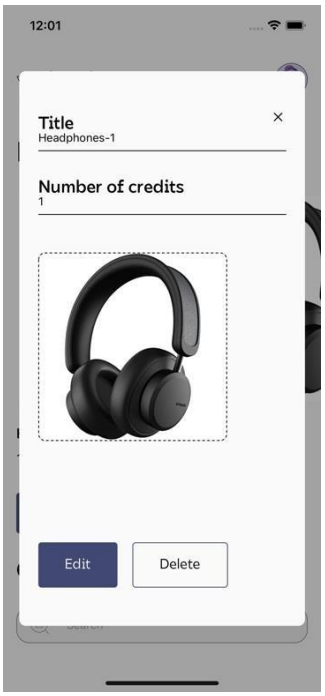
Kako bi se potaknuo timski rad unutar kompanije kreira se sustav međusobnog nagrađivanja. Na mjesečnoj bazi zaposlenici dobivaju tri zvjezdice koje mogu poslati bilo kojem drugom zaposleniku. Svaka zvjezdica ima svoju težinu i moguće ju je zamijeniti za nagrade koje unutar aplikacije postavlja administrator. U jednom mjesecu nije moguće poslati više od jedne zvjezdice istom zaposleniku. Administratorski račun ne sudjeluje u slanju zvjezdica jer on u suštini predstavlja kompaniju, a ne zaposlenika s većim ovlastima.

Do pregleda trenutnih nagrada dolazi se otvaranjem navigacije u zaglavlju te pritiskom na gumb Rewards.



Slika 4.28. *Administratorov zaslon nagrade*

Na slici 4.28. prikazane su nagrade koje se dohvaćaju sa servera i podaci zaposlenika koji se čitaju iz Redux-a. Nagrade su implementirane pomoću virtualizirane liste te je svaki stupac učinjen klikabilnim. Pritiskom na nagradu otvara se modal koji predstavlja osnovni način prikaza sadržaja iznad zaslona [16] (slika 4.29.).



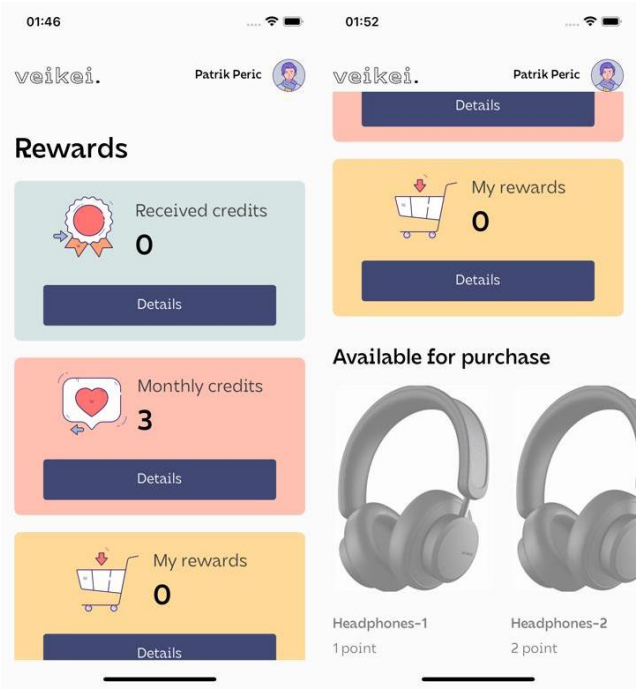
Slika 4.29. *Prikaz nagrade*

Nakon otvaranja modala, nagradu je moguće urediti i obrisati. Pritiskom na gumb, kreiraj novu nagradu, otvara se identičan modal kojemu je forma prazna. Za učitavanje slika koristi se eksterna biblioteka koja daje pristup slikama skladištenim na uređaju. Također, vraća base64 format, a upravo takav format zahtjeva server (slika 4.30.).

```
const handleUploadImage = React.useCallback(async () => {
    const response = await launchImageLibrary({ mediaType: 'photo', quality: 1, includeBase64: true });
    if (response.assets) {
        setImage(response);
    }
}, []);
```

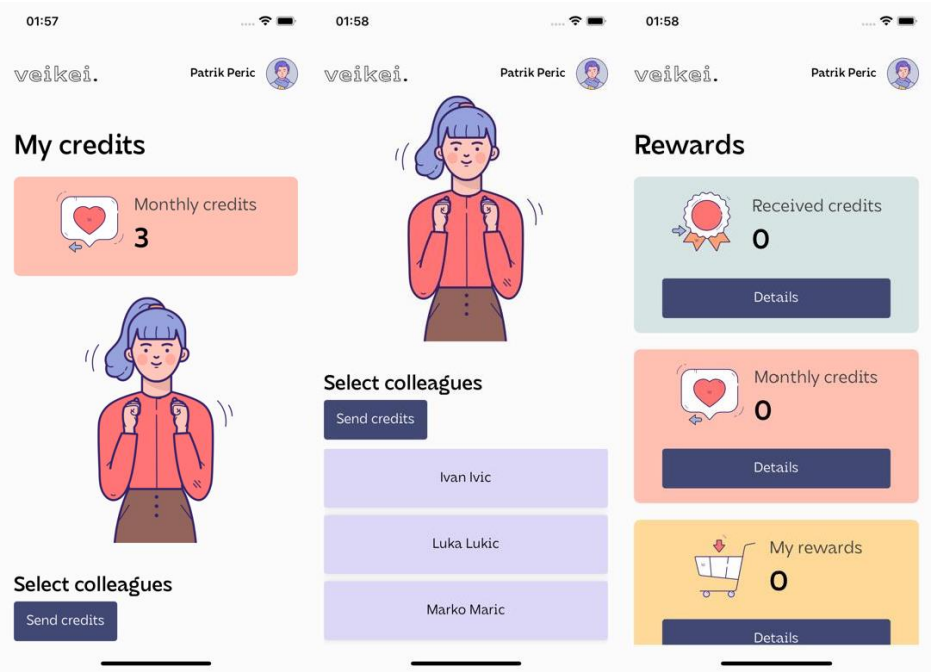
Slika 4.30. *Učitavanje slike*

Popunjena forma šalje se serveru te u slučaju uspješnosti zahtjeva ažurira se stanje s nagradama. Nakon što su nagrade dostupne, zaposlenici mogu zamijeniti svoje bodove za njih. Zaposlenik ulaskom na zaslon s nagradama ima uvid u primljene i raspoložive kredite te kupljene nagrade. Također, na dnu se prikazuje lista s nagradama koje je administrator kreirao (slika 4.31.). Nagrade unutar liste su zaključane sve dok zaposlenik ne skupi dovoljan broj bodova.



Slika 4.31. *Zaposlenikov zaslon nagrade*

Pritiskom na detalje bilo koje kartice otvara se novi zaslon. Unutar zaslona mjesečni krediti, zaposlenik ima uvid kome je poslao kredite te raspoloživost istih. Također, omogućeno mu je slanje kredita koje se izvršava jednostavnim označavanjem imena zaposlenika unutar liste. Poslane kredite nije moguće poništiti ni urediti te je iz toga razloga dozvoljeno pojedinačno slanje. Nakon slanja, stanje kredita se automatski ažurira, a to je ostvareno pomoću Redux-a (slika 4.32.).



Slika 4.32. *Slanje kredita*

Zaposlenici kojima su poslani krediti, zaprimaju email poruku i ulaskom u aplikaciju pod

primljenim kreditima mogu vidjeti tko ih je nagradio za odličan rad. Također, automatski se otključavaju nagrade u ovisnosti o primljenim kreditima. Za implementaciju automatskog ažuriranja više zaslona koristi se Redux. Zaposlenikovi podaci o primljenim i poslanim kreditima te kupljenim nagradama spremaju se u jedan objekt te se ažuriraju prilikom slanja zahtjeva na server. Ažuriranjem podataka izvršava se ponovo renderiranje komponente, a to omogućuje prikaz najnovijih podataka.

5. ZAKLJUČAK

Cilj ovog diplomskog rada je izrada mobilne aplikacije čija je svrha poboljšanje unutrašnje organizacije unutar kompanija. Prilikom izrade aplikacije koristilo se razvojno okruženje React Native i programski jezik TypeScript. Stoga je aplikacija dostupna za platforme iOS i Android.

Aplikacija ima dvije vrste korisnika, a to su administrator i zaposlenik. Kako bi komunikacija između njih bila moguća, mobilna aplikacija komunicira s udaljenom bazom podataka koja je kreirana od strane kompanije za koju se izrađuje mobilno rješenje. Glavna značajka aplikacije je mogućnost praćenja godišnjih odmora. Za ubrzanje procesa odgovaranja na godišnje odmore uvedene su notifikacije koje se kreiraju u slučaju slanja zahtjeva i odgovora. Zaposlenik kreira zahtjev, a administrator odgovara. Osim godišnjih odmora, aplikacija sadrži sustav međusobnog nagrađivanja djelatnika koji je kreiran u svrhu poboljšanja timskog rada. Temelj sustava je međusobno slanje zvjezdica između zaposlenika koje se na kraju mjeseca mogu zamijeniti za nagrade kreirane od strane administratora.

Mobilna aplikacija bi se mogla dodatno poboljšati uvođenjem statistike kod kupnje nagrada. Pomoću nje bi administrator mogao pratiti koje nagrade su najviše kupovane, a koje je potrebno izbaciti. Također, poželjna nadogradnja bi bila integracija s Outlook-om u svrhu prikazivanja zaposlenikovih obaveza na dnevnoj bazi.

LITERATURA

[1] Wikipedija, https://en.wikipedia.org/wiki/Thinking_outside_the_box , 18.07.2022

[2] Qubit Labs, https://qubit-labs.com/how-many-programmers-in-the-world/ , 18.07.2022

[3] Trgovina Play – Day Off Absence, https://play.google.com/store/apps/details?id=com.enozom.dayoffapp&hl=en_US&gl=US, 18.07.2022

[4] Trgovina Play – Leave Dates, https://play.google.com/store/apps/details?id=com.nortonfive.leavedates&hl=hr&gl=US, 18.07.2022

[5] Wikipedija, https://en.wikipedia.org/wiki/Jira_(software), 01.07.2022

[6] TypeScript Handbook, https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html, 01.07.2022

[7] React Native, https://reactnative.dev/architecture/fabric-renderer, 02.07.2022

[8] React Native, https://reactnative.dev/architecture/render-pipeline, 04.07.2022

[9] React Native, https://reactnative.dev/docs/components-and-apis, 04.07.2022

[10] Redux, https://redux.js.org/introduction/getting-started, 07.07.2022

[11] React Native, https://reactnavigation.org/docs/navigation-container/, 08.07.2022

[12] Auth0, https://auth0.com/docs/secure/tokens/access-tokens, 10.07.2022

[13] Custom Hook, https://reactjs.org/docs/hooks-custom.html, 09.07.2022

[14] React Hook Form, https://react-hook-form.com/, 14.07.2022

[15] Notifee, https://notifee.app, 16.07.2022

[16] React Native, https://reactnative.dev/docs/modal, 16.07.2022

SAŽETAK

U ovome se diplomskom radu opisuje izrada mobilne aplikacije za evidenciju rada djelatnika unutar tvrtke. Aplikacija je namijenjena svim kompanijama koje žele poboljšati unutrašnju organizaciju. Ona sadrži dvije uloge, a to su administrator i zaposlenik. Svaka uloga ima drugačiji pogled i mogućnosti unutar aplikacije. Administrator kreira i uređuje zaposlenike, odgovara na njihove zahtjeve za godišnji odmor i uvodi nagrade. S druge strane, zaposlenicima je omogućen pregled i kreiranje zahtjeva za godišnje odmore te slanje zvjezdica drugim zaposlenicima koje se mogu zamijeniti za nagrade. Slanje i odgovaranje na zahtjeve popraćeno je notifikacijama koje služe za ubrzavanje procesa. Prilikom izrade mobilne aplikacije korišten je razvojni okvir React Native i programski jezik TypeScript. Zbog korištenja okvira za razvoj mobilnih aplikacija na više platformi, aplikacija je dostupna za iOS i Android uređaje. Također, korišteni su moderni alati kao ESLint i Prettier koji doprinose konzistentnosti koda.

Ključne riječi: notifikacije, React Native, TypeScript, ESLint, Prettier

ABSTRACT

Title: Mobile application for employee's work records within the company

This thesis describes the development of mobile application for employee work records within the company. The application is intended for all companies that want to improve their internal organization. It contains two roles, administrator and employee. Each role has a different view and capabilities within the application. The administrator creates and edits employees, responds to their vacation requests, and introduces rewards. On the other hand, employees are enabled to view and create vacation requests and send stars to other employees that can be exchanged for rewards. Sending and responding to requests is accompanied by notifications that serve to speed up the process. While creating a mobile application, the React Native development framework and the TypeScript programming language are used. Due to the cross-platform mobile app development framework, the app is available for iOS and Android devices. Also, modern tools such as ESLint and Prettier were used, which contribute to code consistency.

Key words: notifications, React Native, TypeScript, ESLint, Prettier

ŽIVOTOPIS

Sven Suk rođen je u Našicama 22.rujna 1998. godine. U Našicama završava osnovnu školu i prirodoslovno-matematičku gimnaziju s odličnim uspjehom. Nakon završetka gimnazije 2017. godine upisuje se na preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Godine 2020. uspješno završava preddiplomski studij te upisuje diplomski studij programsko inženjerstvo na istoimenom fakultetu. Na diplomskom studiju počinje raditi u struci i trenutno se usavršava u području razvoja mobilnih aplikacija korištenjem React Native razvojnog okvira.

Sven Suk

PRILOZI

Prilog 1: Diplomski rad .docx formata

Prilog 2: Diplomski rad .pdf formata

Prilog 3: Izvorni kod aplikacije