Razvoj mobilne aplikacije za evidenciju servisnih usluga

Brozan, Stefano

VELEUČILIŠTE U RIJECI

Stefano Brozan

Razvoj mobilne aplikacije za evidenciju servisnih naloga

(završni rad)

Rijeka, 2021.

VELEUČILIŠTE U RIJECI

Odjel telematika

Stručni studij Telematika

Razvoj mobilne aplikacije za evidenciju servisnih naloga

(završni rad)

MENTOR

dr. sc. Ida Panev, viši predavač

STUDENT

Stefano Brozan

MBS: 2427000006/17

Rijeka, rujan 2021.

VELEUČILIŠTE U RIJECI
Stručni studij Telematike
Rijeka, 23. 04. 2021.

ZADATAK

za završni rad

Pristupnik Stefano Brozan, MBS: 2427000006/17.

Studentu preddiplomskog stručnog studija Telematika izdaje se zadatak za završni rad – tema završnog rada pod nazivom:

RAZVOJ MOBILNE APLIKACIJE ZA EVIDENCIJU SERVISNIH NALOGA

Sadržaj zadatka:

Opisati tehnologije i alate koji su korišteni u razvoju mobilne aplikacije: Javascript, React, React – native, Redux.js, JSON, Node.js, Visual Studio Code, Firebase platforma. Analizirati informacijski sustav i izraditi dijagram dekompozicije i model procesa (primjenom DTP-a). Opisati važnije implementirane algoritme. Detaljno opisati korištenje gotove aplikacije.

Preporuka:

Rad obraditi sukladno odredbama Pravilnika o završnom radu Veleučilišta u Rijeci.

Zadano: 23. 04. 2021. **Predati do: 15. 09. 2021.**

Mentor:
dr. sc. Ida Panev, v. pred.

Voditelj studija:
Damir Malnar, pred.

Zadatak primio dana: 23.4.2021.

Stefano Brozan

Dostavlja se:
- mentoru
- pristupniku

IZJAVA

Izjavljujem da sam završni rad pod naslovom_____

Razvoj mobilne aplikacije za evidenciju servisnih naloga izradio samostalno pod

nadzorom i uz stručnu pomoć mentora Ide Panev.

Ime i prezime

Stefano Brozan

(potpis studenta)

Sažetak

Tema ovog završnog rada je razvoj mobilne aplikacije pod nazivom: *Evidencija servisnih naloga*. Namijenjena je za tvrtku E-computing iz Pazina koja se bavi prodajom i servisom informatičke opreme. U prvom dijelu opisuju se korištene tehnologije (Javascript, React, React – native, Redux.js, JSON, Visual Studio Code, Node.js, Firebase), dok se u drugom dijelu analizira informacijski sustav, dijagram dekompozicije i model procesa (primjenom DTP-a). Zatim su opisani važniji implementirani algoritmi te samo korištenje gotove aplikacije. Aplikacija pohranjuje podatke o kupcu, servisnom nalogu te sliku naloga (ako je potrebno), sve u svrhu lakšeg evidentiranja zaprimljenog predmeta.

Ključne riječi: React - native, Redux.js, Firebase, Dijagram dekompozicije, Dijagram toka podataka.

SADRŽAJ

1. Uvod

Tema ovog završnog rada je izrada mobilne aplikacije pod nazivom: "*Evidencija servisnih naloga*". Ova jednostavna aplikacija omogućuje zaposlenicima tvrtke evidenciju svih servisnih naloga. U prvom dijelu rada opisuju se korištene tehnologije prilikom izrade same aplikacije.

Ulaskom u aplikaciju od zaposlenika se traži prijava putem njihovog korisničkog imena i lozinke. To se izvršava Firebase-ovom autentifikacijom. Nakon uspješne prijave, dohvaćaju se svi postojeći podaci iz baze podataka (Firebase). Tada se zaposleniku pruža mogućnost izrade novog servisnog naloga, ispis naloga na lokalnom mrežnom printeru, pregled postojećih naloga te uređivanje i završavanje naloga. Nalozi mogu biti u obradi ili izvršeni. Kad zaposlenik popravi neki predmet (npr. printer, laptop, mobitel), na servisni nalog upisuje utrošeni materijal te cijenu usluge. Utrošeni materijal se naknadno fakturira kroz druge programe, dok se cijena usluge također određuje prema vanjskim kriterijima unutar tvrtke. U izgradnji ove mobilne aplikacije korištene su tehnologije: Javascript, React, React - native, Redux.js, JSON, Node.js, Firebase te razvojno okruženje Visual Studio Code.

2. Tehnologije i alati korišteni u razvoju mobilne aplikacije

Sustav aplikacije "*Evidencija servisnih naloga*" podijeljen je na frontend[1] i backend[2]. Kao frontend se pretežito koristi Javascriptov framework[3] React – native, dok se za backend koristi Google-ov Firebase. Sve navedene te ostale primjenjene tehnologije i alati korišteni u izradi aplikacije opisani su u nastavku.

Javascript je najpopularniji međuplatformski programski jezik. To je interpretiran jezik namijenjen razvoju interaktivnih HTML[4] stranica. Većina današnjih web stranica koristi Javascript jer je najlakše primjenjiv u ostalim frameworcima. Zbog takvih frameworka olakšana je sama izrada bilo web aplikacije ili mobilne aplikacije. Ovaj jezik omogućuje stvaranje dinamičkog sadržaja, kontrolu multimedije, animacija i sličnih interaktivnih efekata [1]. U aplikaciji *Evidencija servisnih naloga* Javascript je kao programski jezik korišten tijekom cijele aplikacije, najviše kroz njegove frameworke i biblioteke.

React je biblioteka Javascripta za izgradnju korisničkog sučelja [2]. Izrađena je od strane Facebooka. Glavna značajka Reacta je korištenje komponenti. One omogućuju višekratnu uporabu, odnosno kada je jednom definirana, može se ponovno upotrijebiti. Druga važna stavka kod Reacta su "*Hooks*" koji dopuštaju korištenje stanja aplikacije bez pisanja klasa [3]. Važni su za upravljanje stanjima aplikacije. Još jedna važna karakteristika Reacta je u izradi aplikacije - pri promjeni podataka, stranica ili zaslon se ne treba ponovno učitavati. Ova biblioteka koristi JSX[5] sintaksu koja pretvara HTML u Reactove komponente. U aplikaciji su korišteni Reactovi *Hooks*: *useState*, *useEffect*, *useContext.*

[1] Frontend – sve ono s čime korisnik u programu stupa u interakciju
[2] Backend – sve što korisnik ne vidi, a bitno je za funkcioniranje programa
[3] Framework – skupina koda koja pruža višekratnu mogućnost primjene u raznim aplikacijama, sastoji se od raznovrsne biblioteka koda
[4] HTML – HyperText Markup Language
[5] JSX – Javascript XML (Extensible Markup Language)

React - native je Javascriptov framework namijenjen za izradu mobilnih aplikacija [3]. Temelji se na Reactu te također koristi JSX sintaksu. Velika prednost React native–a naspram drugih frameworka je da se ista aplikacija može pokretati na android i IOS uređajima. Neke najvažnije React-Native komponente korištene u aplikaciji *Evidencija servisnih naloga* su: <View>, <Text>, <Image>, <ScrollView>, <FlatList> itd.

Redux je biblioteka za upravljanje i ažuriranje stanja aplikacije [4]. Poznat pod engleskim nazivom *State Management*. Koristi se kad postoji veći broj stanja aplikacije koje su potrebne na više mjesta u aplikaciji. Npr. aplikacija *Evidencija servisnih naloga* na početku korištenja dohvaća podatke o zaposlenicima. Ulaskom u aplikaciju dohvaćaju se podaci o servisnim nalozima. Kreiranjem novog naloga ponovno se dohvaćaju podaci o trenutnim nalozima i dodaje novi nalog na stog aplikacije. Odjavom zaposlenika opet se moraju dohvaćati podaci o zaposlenicima itd. Redux je potrebno implementirati za svaki objekt koji se koristi. Sastoji se od više komponenti: Vrste (eng. *Types*), Reduktori (eng. *Reducers*), Akcije (eng. *Actions*), Korijenski reduktor (eng. *Root Reducer*) i Skladište (eng. *Store*). Redux je u aplikaciji *Evidencija servisnih naloga* prvenstveno korišten kako bi se smanjio broj upita prema bazi podataka – Firebase-u. Bez Reduxa bi se trebao slati upit prema bazi svaki put kada se dogodi promjena nad podacima. Najvažnije funkcije Reduxa korištene u aplikaciji *Evidencija servisnih naloga*, s kojim se povezuje React i Redux su: *mapStateToProps* i *mapDispatchToProps.*

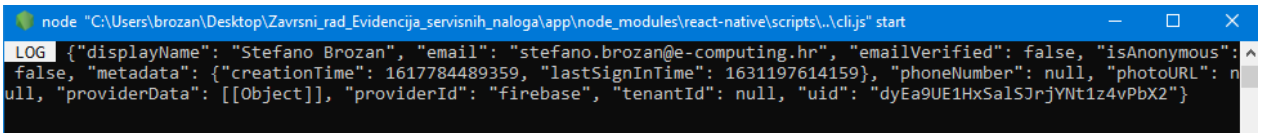
JSON (eng. *JavaScript Object Notation*) je format za pohranu i prijenos podataka koji je čitljiv i razumljiv čovjeku i stroju [5]. JSON se često koristi kada se podaci šalju s poslužitelja na web stranicu jer ne koristi oznake (eng. *tags*) kao XML, čime se postiže da je ovaj format razumljiviji za čitanje i jednostavniji za pisanje. JSON se u aplikaciji koristi u važnoj klasi "*package.json*" u kojoj se nalaze sve postavke za rad aplikacije. Na taj način JSON pruža bolji pregled postavki aplikacije.

Visual studio code je razvojno okruženje za pisanje programskog koda. Izrađen je od strane Microsofta te se koristi na Windows, Linux i macOS platformi [6]. Kao uređivač koda, Visual studio sadrži mnoge značajke za ispravljanje pogrešaka, razna dovršavanja pisanja koda te ima

ugrađeni Git[6] koji omogućava spremanje koda na neki web poslužitelj kao što je npr. GitHub[7]. Visual studio code kao programsko razvojno okruženje korišteno je za razvoj cijele aplikacije *Evidencija servisnih naloga*.

Node.js je platforma otvorenog koda za izradu brzih serverskih aplikacija koristeći Javascript. Koristi se za vrijeme trajanja, izvođenja same aplikacije. Omogućuje pokretanje složenih programa u realnom vremenu izvan preglednika (eng. *Runtime environment*) [7]. Često se pokreće u sustavu s Visual Studio Code–om gdje se podaci koji se zatraže unutar *console.log()* funkcije ispisuju u Node.js terminalu. Node.js terminal se pri izradi aplikacije *Evidencija servisnih naloga* koristi za razna prikazivanja podataka, odnosno što se događa u pozadini aplikacije. Također se koristi za pokretanje testiranja aplikacije naredbom unutar Visual studio code-a: „*npx react-native run-android*“. Uz tu postoje i mnoge druge naredbe koje su važne za izradu aplikacije. U nastavku je prikaz podataka prijavljenog zaposlenika aplikacije *Evidencija servisnih naloga* unutar Node.js terminala (slika 1).

Slika 1: Prikaz podataka prijavljenog zaposlenika



Izvor: autor

Firebase je Google - ov backend framework koji služi za implementaciju logike sustava. Unutar Firebase–a postoje usluge kao što su: autentifikacija (eng. *authentication*), pohrana (eng. *storage*), baza podataka (eng. *Firestore Database*) i mnoge druge. Navedene usluge su besplatne do određene granice, a cijene su prihvatljive i kad se granica prekorači.

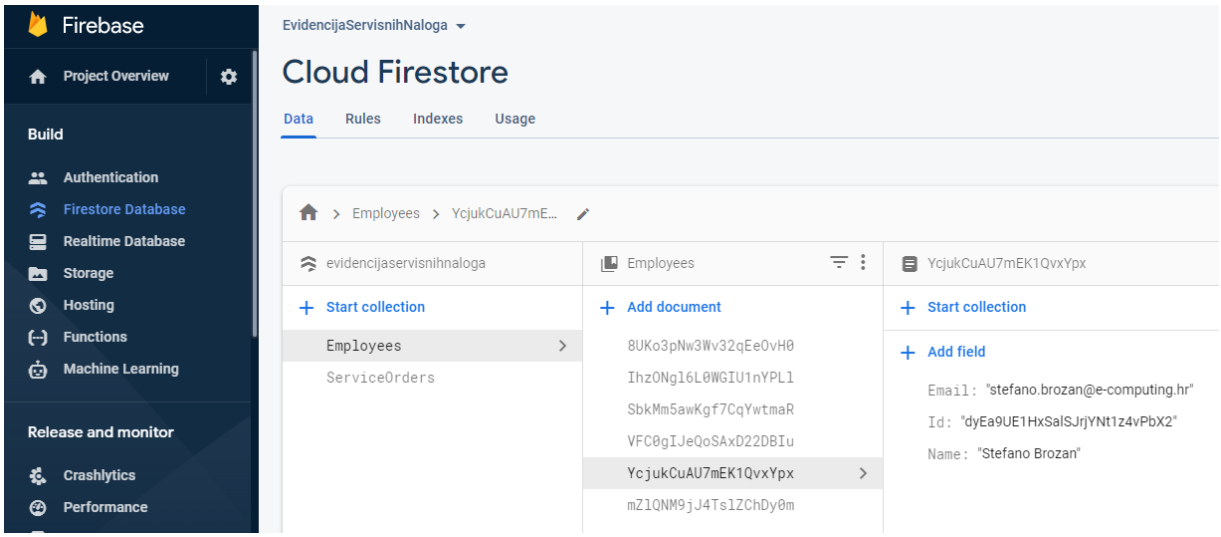
Firestore Database funkcionira na temelju NoSQL dokumentno orijentirane baze podataka. U takvom sustavu nema tablica i redova. Umjesto toga podaci se spremaju u dokumente koji su

[6] Git – distribuirani sustav za upravljanje izvornim kodom.
[7] Github - pružatelj internetskog hostinga za razvoj softvera i kontrolu verzija pomoću Git – a.

organizirani u zbirke dokumenata. Svaki dokument sadrži set univerzalnog ključa [8]. Firestore je optimiziran za pohranu velikih zbirki malih dokumenata.

U Firebase-u se izradi projekt koji se registrira u sustavu te se funkcijama povezuje na aplikaciju iz Visual Studio Code–a. Firebase je isplativ i stabilan za vođenje manjih programa. U nastavku je slika prikaza Firebase-ova Firestora, odnosno zbirke dokumenata korištene u aplikaciji *Evidencija servisnih naloga (slika 2)*. To su dvije zbirke od kojih jedna čuva sve podatke o zaposlenicima (*Employees*), a druga sve podatke o servisnim nalozima (*ServiceOrders*).

Slika 2: Prikaz kolekcija dokumenata aplikacije "Evidencija servisnih naloga"



Izvor: autor

3. Analiza informacijskog sustava

Sustav mobilne aplikacije za evidenciju servisnih naloga sprema podatke o zaposlenicima i servisnim nalozima u Firestore. U nastavku će biti objašnjeni dijagrami dekompozicije i dijagrami toka podataka.

3.1. Dijagram dekompozicije

"Dijagram dekompozicije je dijagram koji neki složeni sustav prikazuje u obliku hijerarhijske strukture podsustava" [9]. U aplikaciji *Evidencija servisnih naloga* postoje 3 glavna procesa: Autentifikacija zaposlenika, Evidencija naloga i Printanje naloga. Oni se dalje granaju na svoje podprocese.

Proces Autentifikacija zaposlenika je početni i krajnji proces u aplikaciji. Najvažniji je proces u aplikaciji zbog toga što se svaki zaposlenik prije korištenja aplikacije mora prijaviti kako bi mogao koristiti aplikaciju, a nakon korištenja se mora odjaviti. Podprocesi procesa Autentifikacija su: 1.1. Upis zaposlenika, 1.2. Prijava zaposlenika, 1.3 Odjava zaposlenika. Upis zaposlenika omogućuje unos novog zaposlenika, brisanje postojećeg zaposlenika ili promjenu lozinke postojećeg zaposlenika kroz Firebase platformu. Prijava zaposlenika omogućuje prijavu postojećeg zaposlenika u aplikaciju odabirom na jedno od ponuđenih imena na popisu početnog zaslona. Zaposlenik unosom točne lozinke unutar skočnog prozora može nastaviti koristiti aplikaciju i doći do ostala 2 procesa, dok mu se u suprotnom javlja greška "netočna lozinka" te ostaje na trenutnom zaslonu. Odjava zaposlenika se izvršava na kraju korištenja aplikacije. Zaposlenik pritiskom na tipku navigacije – "tri crtice" (eng. *Anchor button*) odabire ikonu "Odjava zaposlenika" čime ga aplikacija pita ako se zaista želi odjaviti. Potvrdom, zaposlenik se vraća na početni zaslon prijave zaposlenika.

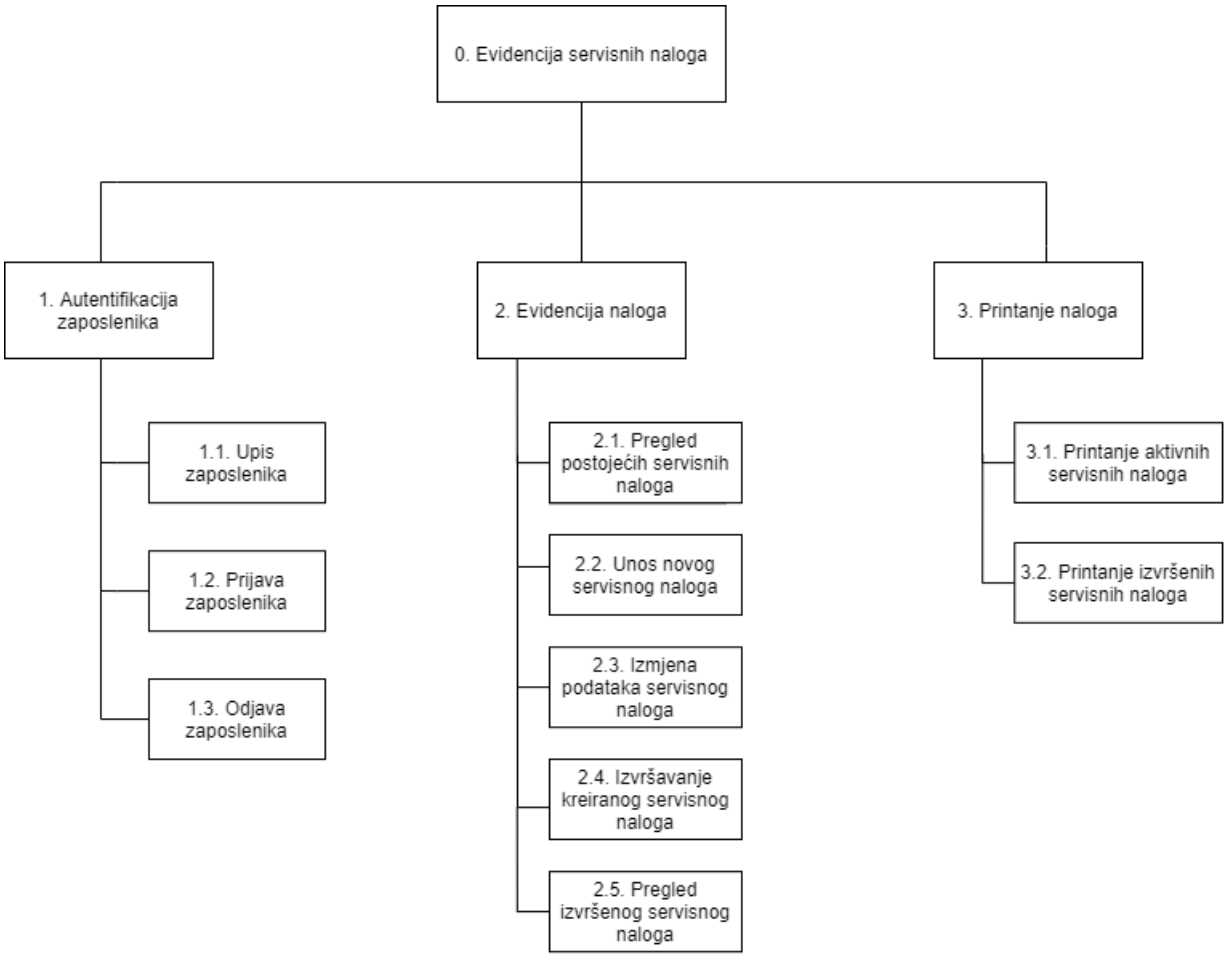
Proces Evidencija naloga je proces kojim se dohvaćaju svi postojeći servisni nalozi koji se dijele na dvije skupine: trenutni i izvršeni servisni nalozi. Trenutni nalozi su oni koji se trenutno izvršavaju u servisu, dok su izvršeni servisni nalozi oni koji su već izvršeni, naplaćeni, ali su bitni za cjelokupnu evidenciju servisnih naloga. Podprocesi procesa Evidencija naloga su: 2.1. Pregled postojećih servisnih naloga, 2.2. Unos novog servisnog naloga, 2.3. Izmjena podataka servisnog naloga, 2.4. Izvršavanje kreiranog servisnog naloga, 2.5. Pregled izvršenog servisnog naloga. Podproces Pregled postojećih servisnih naloga služi za dohvat svih postojećih (trenutnih) servisnih naloga iz Firebase-a u aplikaciju. Unos novog servisnog naloga služi za dodavanje servisnog naloga preko aplikacijskog sučelja koji se sprema u Firebase. Izmjena podataka servisnog naloga i izvršavanje kreiranog servisnog naloga su podprocesi koji se izvršavaju pritiskom na isti gumb, simbol olovke. Dizajnirano je tako da bude čim jednostavnije za korištenje. Izmjenom podataka servisnog naloga moguće je mijenjati sve podatke koji su

uneseni u nalog, a naknadno se zbog nekog razloga žele promijeniti. Izvršavanje kreiranog servisnog naloga se koristi kada je servis odrađen. Zaposlenik tada zove kupca na broj telefona koji je upisan na servisni nalog. Zaposlenik tada unosi podatke o izvršenju naloga te se klikom na "Završi nalog" nalog premješta iz trenutnih u izvršene servisne naloge. Pregled izvršenog servisnog naloga služi kako bi zaposlenik pritiskom na navigacijski gumb odabrao izvršene naloge gdje dalje ima mogućnosti pregleda i ispisa izvršenih naloga.

Proces Printanje naloga je važan za ispis trenutnih i izvršenih naloga na papir. Aplikacija komunicira s lokalnim mrežnim printerom. Na uređaju na kojem se pokreće aplikacija mora biti instalirana aplikacija za pokretanje servisa mrežnog printera kako bi ispis bio moguć. Proces Printanje naloga se dijeli na 2 podprocesa: 3.1. Printanje aktivnih servisnih naloga i 3.2. Printanje izvršenih servisnih naloga. Printanje aktivnih servisnih naloga se izvršava pri kreiranju novog servisnog naloga, kada kupac donese neki artikl na servis. Printanje izvršenih servisnih naloga se izvršava prilikom završavanja servisnog naloga da se kupcu ispisuje koje su usluge izvršene te koliko je novaca potrebno platiti. To je opcionalna mogućnost, radi uštede papira, čuvanja šuma i svega ostalog. U realnosti se taj podproces može i zanemariti.

U nastavku slijedi slika dijagrama dekompozicije (slika 3).

Slika 3: Dijagram dekompozicije

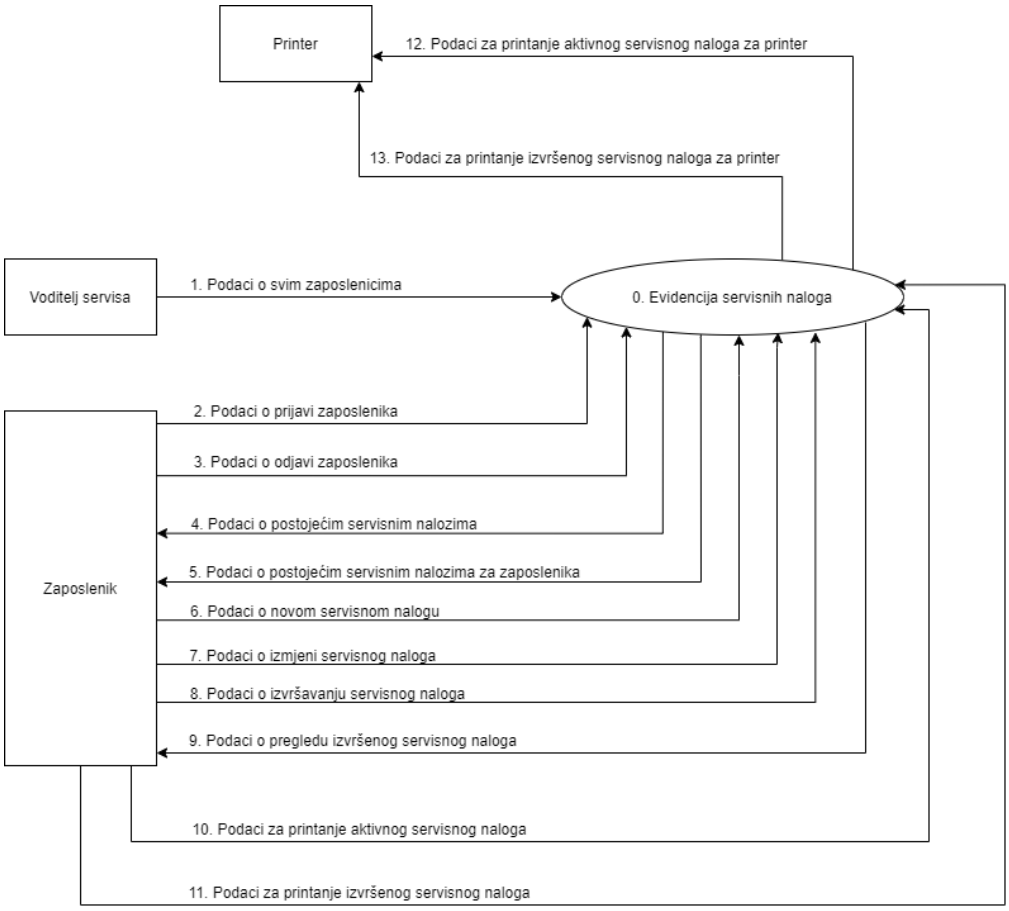


Izvor: autor

3.2. Dijagram konteksta

Dijagram nulte razine dekompozicije (dijagram konteksta, DTP 0. razine) je dijagram toka podataka najviše razine dekompozicije. Prikazuje čime se sustav bavi te okolinu u kojoj djeluje [10]. U aplikaciji *Evidencija servisnih naloga* postoje 3 vanjska sustava: voditelj servisa, zaposlenik i printer. Voditelj servisa određuje podatke o zaposlenicima koje zapisuje u sustav (Firestore Database). Nakon što zaposlenik pokrene aplikaciju, iz liste svih zaposlenika odabire svoje ime te upisuje svoju lozinku s kojom ulazi u sustav aplikacije. Ulaskom u aplikaciju prikazuju mu se svi postojeći servisni nalozi. Zaposlenik može unositi nove, pregledavati i uređivati postojeće naloge te završavati postojeće naloge. Također može printati naloge koji su u obradi te izvršene naloge. U nastavku slijedi slika dijagrama nulte razine (slika 4).

Slika 4: Dijagram nulte razine dekompizicije (dijagram konteksta)

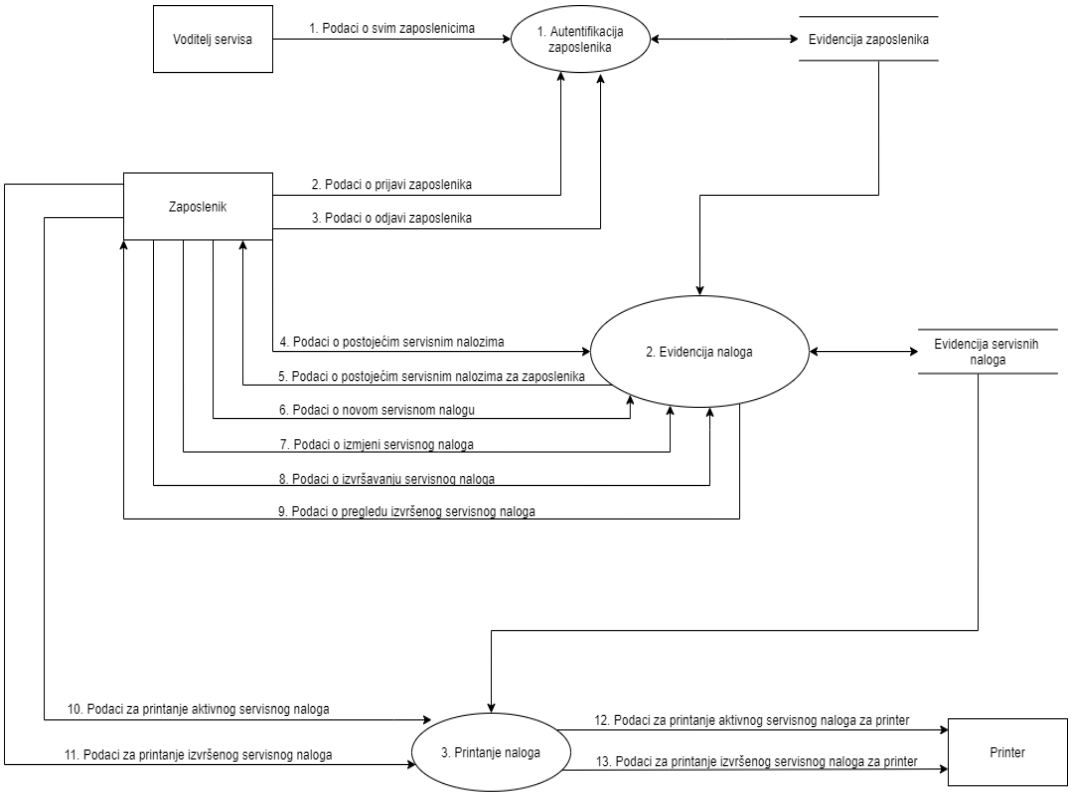


Izvor: autor

3.3. Dijagram toka podataka prve razine dekompozicije

Dijagram prve razine dekompozicije detaljnije opisuje protok podataka kroz sustav. Dijelovi dijagrama su: procesi, tokovi podataka, vanjski sustavi i spremišta podataka. U aplikaciji *Evidencija servisnih naloga* postoje 2 spremišta podataka: spremište Evidencija zaposlenika i spremište *Evidencija servisnih naloga*. Spremišta imaju obostrane strelice toka zbog toga što procesi u njih mogu upisivati podatke ili ih iz njih čitati. U nastavku je slika dijagrama prve razine dekompozicije (slika 5).

Slika 5: Dijagram prve razine dekompozicije



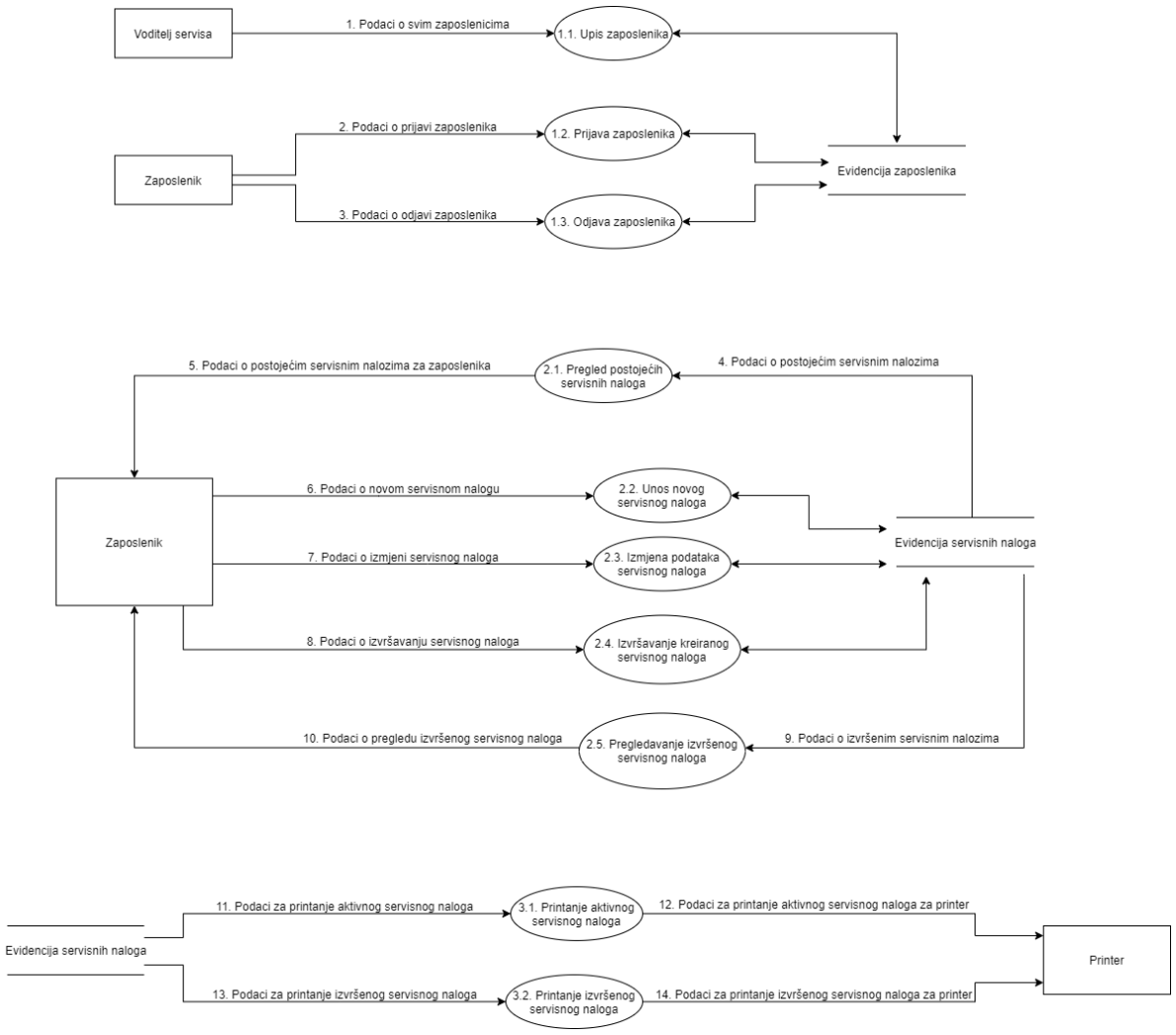
Izvor: autor

Opis dijagrama konteksta je sljedeći: “Voditelj servisa” u sustav koji se nalazi na Firebase-u dostavlja “Podatke o svim zaposlenicima”. Proces “1. Autentifikacija zaposlenika” upisuje te podatke u spremište “Evidencija zaposlenika”. “Zaposlenik” pri početku korištenja aplikacije u sustav unosi “Podatke o prijavi zaposlenika”, dok na kraju korištenja aplikacije unosi “Podatke o odjavi zaposlenika” koji se spremaju u spremište “Evidencija zaposlenika”. Ovi se podaci iz spremišta prosljeđuju procesu “2. Evidencija naloga”. “Zaposlenik” nadalje šalje upit o “4. Podacima o postojećim servisnim nalozima”. Ako ih ima, spremište “Evidencija servisnih naloga” prosljeđuje podatke procesu “2. Evidencija naloga” koji šalje “5. Podatke o postojećim servisnim nalozima za zaposlenika” prema “Zaposleniku”. “Zaposlenik” dostavlja “6. Podatke o novom servisnom nalogu”, “7. Podatke o izmjeni servisnog naloga” te “8. Podatke o izvršavanju servisnog naloga” procesu: “2. Evidencija naloga” koji ih sprema u spremište “Evidencija servisnih naloga”. Ako postoje izvršeni servisni nalozi, tada spremište “Evidencija servisnih naloga” preko procesa “2. Evidencija naloga” dostavlja “9. Podatke o pregledu izvršenog servisnog naloga” prema “Zaposleniku”. Spremište “Evidencija servisnih naloga” dostavlja sve navedene tokove podatka iz procesa “2. Evidencija naloga” prema procesu “3. Printanje naloga”. “Zaposlenik” šalje “10. Podatke za printanje aktivnog servisnog naloga” i “11. Podatke za printanje izvršenog servisnog naloga” procesu: “3. Printanje naloga”. Taj proces ih dostavlja “Printeru” preko “12. Podataka za printanje aktivnog servisnog naloga za printer” te “13. Podataka za printanje izvršenog servisnog naloga za printer”.

3.4. Dijagram toka podataka druge razine dekompozicije

Dijagram druge razine prikazuje kako tokovi podataka kolaju između podprocesa. Dijagrami su razdijeljeni po procesima i njihovim podprocesima. U nastavku je slika dijagrama druge razine dekompozicije za procese 1,2 i 3 (slika 6).

Slika 6: Dijagrami druge razine dekompozicije za proces 1,2 i 3



Izvor: autor

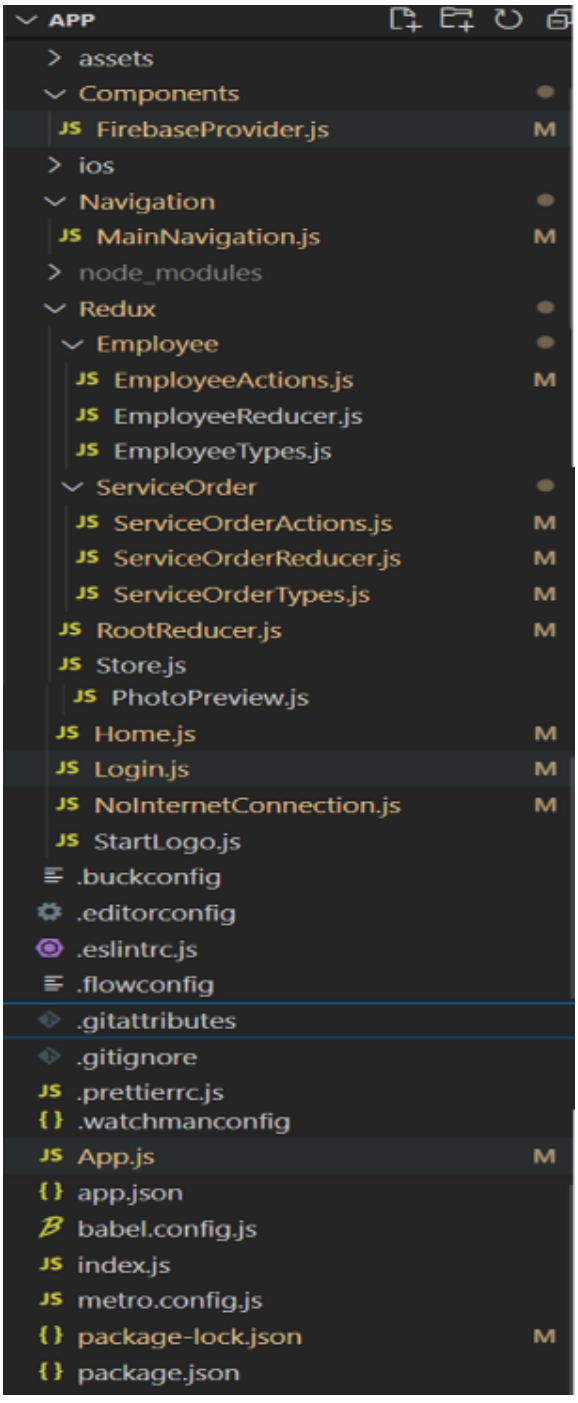
4. Opis važnijih implementiranih algoritama i struktura aplikacije

Aplikacija *Evidencija servisnih naloga* izrađena je pomoću programskog alata Visual Studio Code-a i Firebase platforme čija je svrha pružanje backend usluga. U nastavku će biti opisani neki od važnijih implementiranih algoritama korišteni u aplikaciji te struktura aplikacije.

4.1. Struktura mobilne aplikacije *Evidencija servisnih naloga*

Aplikacija je strukturirana mnogim komponentama i funkcijama koje moraju međusobno komunicirati kako bi sustav mogao pravilno funkcionirati. Prije pisanja koda potrebno je uvesti razne biblioteke koje sadrže predefinirane funkcije koje olakšavaju rad. Aplikacija se pokreće preko komponente App.js u kojoj se definiraju sve komponente koje će imati određeni zadatak tijekom izvođenja aplikacije. Kako bi se zaposleniku omogućilo kretanje po aplikaciji, potrebno je kreirati komponentu MainNavigation.js koja sadrži sve prozore aplikacije. Kako bi se u aplikaciji znalo koji je zaposlenik trenutno prijavljen, potrebno je obgrliti kostur aplikacije s FirebaseProvider.js komponentom unutar koje je implementirana logika autentifikacije zaposlenika. Unutar Redux mape nalaze se implementirane logike za pojedini model (zaposlenik i servisni nalog), skladište koje omogućuje spremanje te korijenski reduktor koji spaja ranije spomenute modele i njihova stanja. Redux služi isključivo za komunikaciju s backendom (Firebase-om). Screens je mapa koja sadrži komponente koje predstavljaju prozore aplikacije te isto tako definiraju zadatke dinamičkih dijelova kao što su: kreiranje, pregledavanje, izmjena, završavanje servisnih naloga. U nastavku je slika glavne strukture aplikacije (slika 7).

Slika 7: Struktura aplikacije unutar Visual Studio Code-a



Izvor: autor

4.2. Funkcija App.js

U funkciji App.js prikazano je na koji način aplikacija komunicira sa svim svojim dijelovima. Drugim riječima, to je korijenska komponenta aplikacije. MainNavigation se nalazi u sredini, a obgrljen je FirebaseProviderom koji komunicira s Firebase–om te Providerom koji komunicira s Reduxom. Na kraju se nalazi PaperProvider koji daje definirani dizajn aplikacije. U nastavku je slika prikaza App.js funkcije (slika 8).

Slika 8: Prikaz App.js funckije

Izvor: autor

4.3. Funkcija MainNavigation.js

U funkciji MainNavigation.js se izvršava glavna navigacija u aplikaciji. Ispituje se ako je zaposlenik ulogiran. Ukoliko je, onda se prikazuju zasloni aplikacije (eng. *Stack Screens*) koji se nalaze unutar glavnog navigacijskog zaslona (eng. *Stack Navigator*), u suprotnom mu se prikazuje prozor Prijave (eng. *Login*). Navigacija radi na prinicpu stoga (eng. *Last in First out*) što znači da ukoliko zaposlenik pređe na drugu stranicu ima mogućnost vraćanja na prethodnu stranicu bez izlaska iz aplikacije. Isto tako prikazuje se početni logo (eng. *Splash Screen*) koji traje određen broj sekundi nakon čega se pokreće aplikacija. Ukoliko nema internetske veze prikazuje se slika koja daje upozorenje da nema internetske veze, a nakon što je veza uspostavljena rad aplikacije se nastavlja. Svi ti zasloni se nalaze unutar *NavigationContainer*–a što je komponenta koja čuva sve zaslone u jednom. U nastavku je slika prikaza strukture zaslona aplikacije (slika 9).

Slika 9: Prikaz strukture zaslona aplikacije

```
53    return (
54      <NavigationContainer theme={{colors: {background: 'white'}}}>
55        <Stack.Navigator initialRouteName="Home" mode="modal">
56          {user ? (
57            <Stack.Screen
58              name="Home"
59              component={Home}
60              options={({navigation}) => ({
61                headerLeft: () => (
62                  <Menu
63                    visible={menuVisible}
64                    statusBarHeight={48}
65                    style={{marginLeft: 8}}
66                    contentStyle={{
67                      height: '110%',
68                      width: Dimensions.get('window').width - 150,
69                    }}
```

Izvor: autor

4.4. Funkcija FirebaseProvider.js

Funkcija FirebaseProvider.js je slušač događaja za logiranog zaposlenika. Kad god se promjeni stanje zaposlenika (ako se zaposlenik prijavi ili odjavi), to novo stanje se sprema u varijablu „user“. Unutar funkcije FirebaseProvider.js nalazi se Reactov konstruktor Context.Provider koji služi za spremanje vrijednosti varijable koja će se spremati tijekom korištenja aplikacije. U ovom slučaju konstruktor obgrljuje zaposlenika, što znači da se aplikacija ne može pokrenuti sve dok nije unesena točna lozinka tijekom prijave. Funkcija FirebaseProvider je vidljiva na slici 10.

Slika 10: Prikaz funkcije FirebaseProvider

Components > JS FirebaseProvider.js > ...

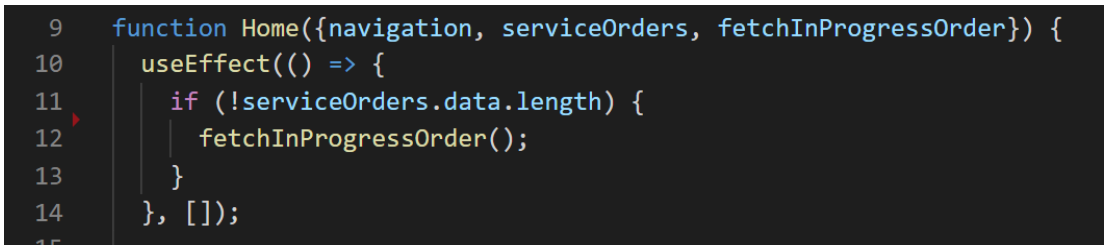
```
1  import React, {createContext, useState} from 'react';
2
3  export const Context = createContext();
4
5  export const FirebaseProvider = ({children}) => {
6    const [user, setUser] = useState(null);
7
8    return (
9      <Context.Provider
10       value={{
11         user,
12         setUser,
13       }}>
14       {children}
15     </Context.Provider>
16   );
17 };
```

Izvor: autor

4.5. Funkcija Home.js

U funkciji Home.js dohvaćaju se svi postojeći servisni nalozi u aplikaciji. Koristi se Reactova UseEffect funkcija. Ona nalaže nekoj komponenti da treba nešto izvesti nakon što se funkcija „renderira“. Ovdje se nalazi i funkcija za ispis servisnih naloga, u koju se za prikaz izgleda papira koristi HTML. Uređaj se spaja na lokalni printer te ispisuje nalog. Funkcija Home.js vidljiva je na slici 11.

Slika 11:Prikaz Home.js funkcije



Izvor: autor

4.6. Funkcija Create.js

Funkcija Create.js služi za dodavanje novog servisnog naloga. Zaposlenik upisuje podatke o kupcu i servisu te opcionalno dodaje sliku. Korišteni su razni React-native elementi kao što su: Image, ScrollView, TouchableOpacity, View i drugi. Image element služi za dodavanje slike, gdje mora biti navedeno iz kojeg izvora je slika uzeta. ScrollView omogućuje pomicanje prema dolje, odnosno prikaz i ostalih elemenata u aplikaciji. Renderira sve elemente istovremeno. U tom procesu sprema sve podatke u radnu memoriju uređaja na kojem se izvodi aplikacija. Inače dosta sličan element ScrollViewu je FlatList koji djelomično renderira elemente, kako se lista pomiče prema dolje, tako se prikazuju novi elementi. Koristi se kada postoji više sadržaja za pregledavanje. Taj je element korišten u Home.js i DoneOrders.js. TouchableOpacity je element koji potamnjuje neki gumb pritiskom na njega. Pritiskom na gumb „Kreiraj nalog“, on se potamnjuje na trenutak, potom opet postaje kakav je bio prije. View je element koji služi za prikaz sadržaja. U HTML–u bi mu ekvivalent bio . Moguće je dodati razne stilove, poravnanja, margine itd.

5. Redux u aplikaciji

Kao što je već navedeno, Redux je biblioteka za upravljanje stanjima aplikacije – *state management*. Služi kao centralizirano skladište za stanja koja se koriste u aplikaciji, s pravilima koja osiguravaju da se stanje može ažurirati samo na predvidljiv način. Redux je jedna od glavnih biblioteka aplikacije *Evidencija servisnih naloga*. Sve funkcije koje su u aplikaciji, komuniciraju s Firebase-ovom bazom podataka preko Reduxa. Najvažnija karakteristika je da kad se jednom dohvate podaci o nalogu, lokalno stanje (eng. *state*) se sprema u Redux. To se može koristiti cijelo vrijeme. Kad se izradi novi nalog, sve se opet sprema u Redux. Da nema Reduxa, aplikacija bi slala puno više upita prema bazi, dok se ovako to rješava lokalno za vrijeme korištenja aplikacije.

5.1. Vrste Reduxa

Vrste Reduxa (eng. *Redux Types*) su bitne za definiranje svih mogućih procesa za koje će se koristiti u aplikaciji. Moguća stanja su: upit (eng. *Request*), uspjeh (eng. *Success*) i neuspjeh (eng. *Failure*). Na primjer, nekoj funkciji se u početnom stanju šalje upit prema bazi podataka. Ako funkcija uspješno pristupi bazi podataka te izvrši traženu naredbu, tada joj se šalje stanje *uspjeh*, a ako ne uspije, šalje joj se stanje *neuspjeh*. Drugim riječima, kada se zaposlenik prijavljuje u aplikaciju te unese ispravnu lozinku za ulaz, dobiva stanje *uspijeh* i ulazi u aplikaciju. Ako unese pogrešnu lozinku onda ostaje na toj stranici i dobiva stanje *neuspjeh*. U nastavku je slika funkcije EmployeeTypes.js gdje se prikazuju vrste Reduxa (slika 12).

Slika 12: Prikaz funkcije EmployeeTypes.js

Redux > Employee > JS EmployeeTypes.js > ...

```
1  //Prijava zaposlenika
2  export const EMPLOYEE_LOGIN_REQUEST = 'EMPLOYEE_LOGIN_REQUEST';
3  export const EMPLOYEE_LOGIN_SUCCESS = 'EMPLOYEE_LOGIN_SUCCESS';
4  export const EMPLOYEE_LOGIN_FAILURE = 'EMPLOYEE_LOGIN_FAILURE';
5
6  //Odjava zaposlenika
7  export const EMPLOYEE_LOGOUT_REQUEST = 'EMPLOYEE_LOGOUT_REQUEST';
8  export const EMPLOYEE_LOGOUT_SUCCESS = 'EMPLOYEE_LOGOUT_SUCCESS';
9  export const EMPLOYEE_LOGOUT_FAILURE = 'EMPLOYEE_LOGOUT_FAILURE';
10
11 //Dohvat zaposlenika
12 export const FETCH_EMPLOYEES_REQUEST = 'FETCH_EMPLOYEES_REQUEST';
13 export const FETCH_EMPLOYEES_SUCCESS = 'FETCH_EMPLOYEES_SUCCESS';
14 export const FETCH_EMPLOYEES_FAILURE = 'FETCH_EMPLOYEES_FAILURE';
15
```

Izvor: autor

5.2. Akcije Reduxa

Akcije Reduxa (eng. *Redux Actions*) služe za promjenu stanja aplikacije te stvaranje funkcija za komunikaciju s bazom podataka. Najbitnije su „*dispatch*“ funkcije koje služe za prosljeđivanje akcija u Reduktor (eng. *Reducer*). U nastavku je slika dohvaćanja imena zaposlenika u aplikaciji (slika 13).

Slika 13: Funkcija prikaza dohvaćanja zaposlenika

```
85  export const fetchEmployees = () => {
86    let loadingData = true;
87    return dispatch => {
88      dispatch(fetchEmployeesRequest(loadingData));
89      firestore()
90        .collection('Employees')
91        .get()
92        .then(querySnapshot => {
93          loadingData = false;
94          const result = querySnapshot.docs.map(e => e._data);
95          dispatch(fetchEmployeesSuccess(result, loadingData));
96        })
97        .catch(error => {
98          loadingData = false;
99          dispatch(fetchEmployeesFailure(error.message, loadingData));
100       });
101   };
102 };
```

Izvor: autor

5.3. Reduktor Reduxa

Reduktor Reduxa (eng. *Redux Reducer*) je funkcija koja služi za spajanje stanja i akcija. Uzima stanje i akciju kao argumente te vraća novo stanje. Kao parametre prima funkciju i stanje te vraća novo stanje objekta. Početno se stanje ažurira novim stanjem. Iz Vrsta Reduxa uvezuju se funkcije upita, uspjeha i neuspjeha. Na kraju funkcije se treba nalaziti konstruktor „*mapDispatchToProps*“ s kojim se dohvaćaju funkcije iz Reduxa te povezuju funkcije u različitim zaslonima. Treba se nalaziti i „*connect*“ funkcija s kojom se povezuju Redux funkcije i stanja s trenutnom komponentom. *Dispatch* poziva određeni tip akcija (upit, uspjeh, neuspjeh) za određenu funkciju. U nastavku slijedi prikaz Reduktor funkcije za dohvat zaposlenika u kojoj su navedena sva tri moguća stanja koja se mogu dohvatiti (slika 14). Zaposlenik pita sustav da mu dostavi podatke o prijavi zaposlenika. Sustav dostavlja *uspjeh* ili *neuspjeh* prema zaposleniku.

Slika 14: Reduktor funkcija za dohvat zaposlenika

```
19  const reducer = (state = initialState, action) => {
20    switch (action.type) {
21      case FETCH_EMPLOYEES_REQUEST:
22        return {
23          ...state,
24          loadingData: action.payload,
25        };
26      case FETCH_EMPLOYEES_SUCCESS:
27        return {
28          ...state,
29          employees: action.payload,
30          loadingData: action.loadingData,
31        };
32      case FETCH_EMPLOYEES_FAILURE:
33        return {
34          ...state,
35          loadingData: action.loadingData,
36          employees: [],
37          errorMsg: action.payload,
38          error: true,
39        };
```

Izvor: autor

5.4. Korijenski Reduktor i Redux spremište

Korijenski Reduktor (eng. *Root Reducer*) služi za spajanje svih postojećih Reduktora u jedan. Važan je kako bi se kreiralo spremište svih Reduktora u Redux spremište. Korijenski Reduktor spaja sva stanja iz postojećih Reduktora u jedno spremište (eng. *Redux Store*). Redux Spremište uzima sve podatke iz napunjenog Korijenskog Reduktora. Unutar njega nalazi se i „*Thunk*“ što predstavlja standardni pristup pisanja asinkrone logike u Redux-ovim aplikacijama. Obično se koriste za dohvat podataka. U nastavku slijedi prikaz funkcije punjenja korijenskog Reduktora (slika 15) i prikaz funkcije napunjenog Redux skladišta (slika 16).

Slika 15: Prikaz funkcije punjenja korijenskog Reduktora podacima

```
5  const rootReducer = combineReducers({
6    serviceOrdersData: serviceOrderReducer,
7    employeesData: employeeReducer,
8  });
9  export default rootReducer;
```

Izvor: autor

Slika 16: Prikaz funkcije napunjenog Redux skladišta

```
7  const store = createStore(
8    rootReducer,
9    composeWithDevTools(applyMiddleware(thunk))
10 );
11 export default store;
```

Izvor: autor

6. Prikaz uporabe programskog rješenja

Aplikacija *Evidencija servisnih naloga* podijeljena je na 3 glavna procesa: Autentifikacija zaposlenika, Evidencija naloga i printanje naloga. Ulazak u aplikaciju nije moguć bez početne autentifikacije zaposlenika. Ulaskom u sustav aplikacije zaposleniku se prikazuju svi postojeći servisni nalozi. Odabirom određenog naloga zaposlenik može vidjeti sve njegove detalje. Ako nalog ima sliku, zaposlenik može odabrati sliku kako bi mu se ta slika prikazala.

6.1. Autentifikacija zaposlenika

Pokretanjem aplikacije zaposleniku se odmah nakon početnog loga, iz baze podataka prikazuju svi trenutni zaposlenici. Zaposlenik odabire svoje ime. Pri odabiru imena, prikazuje mu se polje za unos lozinke kojom može ući u aplikaciju. U nastavku slijedi prikaz zaslona prijave zaposlenika (slika 17) te prikaz zaslona unosa lozinke zaposlenika preko skočnog prozora (slika 18).

Slika 17: Prikaz zaslona prijave zaposlenika

Izvor: autor

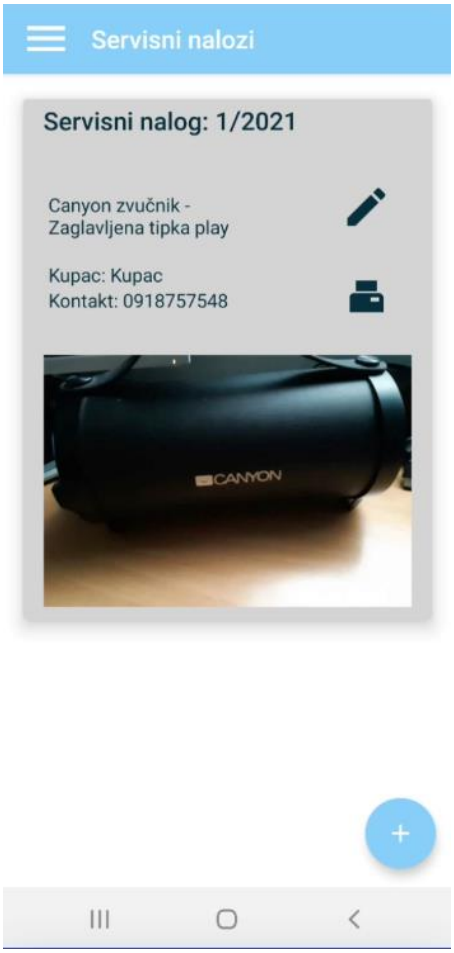
Slika 18: Prikaz zaslona unosa lozinke zaposlenika

Izvor: autor

6.2. Prikaz i detalji postojećih servisnih naloga

Uspješnom autorizacijom zaposleniku se prikazuju svi postojeći servisni nalozi. Zaposlenik ima mogućnost pregleda naloga, pregleda slike unutar postojećeg naloga, dodavanja novog naloga, uređivanja i završavanja postojećeg naloga te ispis naloga. Pritiskom na 3 crtice, odnosno već ranije spomenute tipke navigacije - *Anchor button*, zaposleniku se pružaju još dvije mogućnosti: prikaz izvršenih naloga te odjava zaposlenika. Na slici 19 vidljiv je prikaz postojećih servisnih naloga. Slika 20 prikazuje prikaz detalja servisnog naloga, dok slika 21 prikazuje sliku unutar servisnog naloga, pritiskom na nju unutar zaslona Detalji servisnog naloga.

Slika 19: Prikaz postojećih servisnih naloga



Izvor: autor

Slika 20: Prikaz detalja servisnog naloga

Izvor: autor

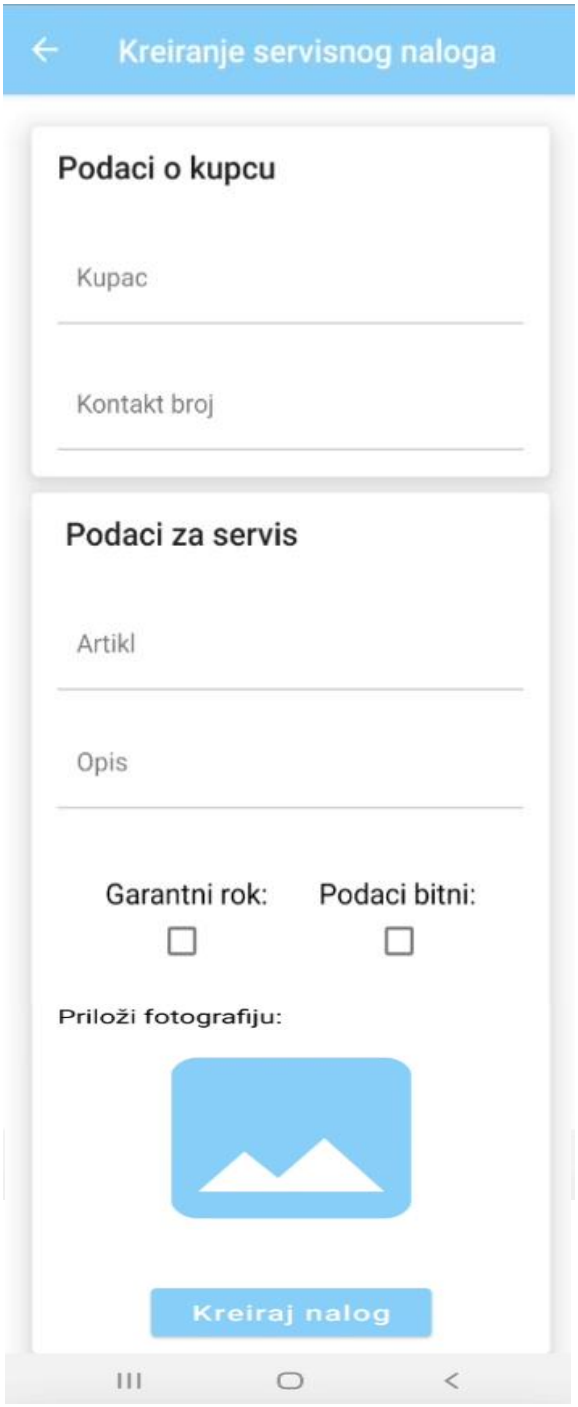
Slika 21: Prikaz slike unutar servisnog naloga

Izvor: autor

6.3. Kreiranje novog servisnog naloga

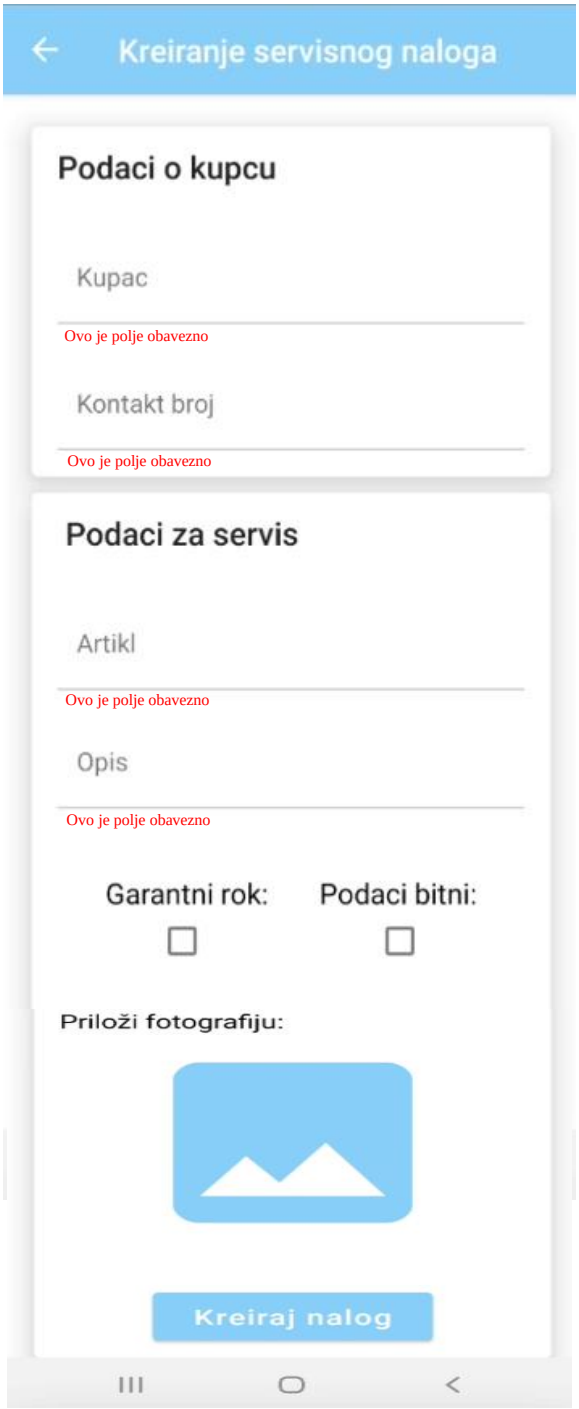
Pritiskom na okruglu plavu ikonu sa simbolom „plus“ na početnom zaslonu, odnosno zaslonu s kreiranim servisnim nalozima, zaposlenik može kreirati novi servisni nalog. Pritom upisuje podatke o kupcu i podatke za servis što je vidljivo na slici 22. Svi su podaci obvezni za unos osim označavanja garantnog roka, važnosti podataka i dodavanja slike koji su opcionalni. Ukoliko zaposlenik ne unese neki od obveznih podataka, prikazuje mu se poruka u kojoj piše kako je to polje obvezno popuniti za nastavak. Slika 23 prikazuje navedeni prikaz upozorenja ukoliko se klikne *Kreiraj nalog* bez unosa.

Slika 22: Kreiranje novog servisnog naloga



Izvor: autor

Slika 23: Prikaz upozorenja ukoliko se klikne Kreiraj nalog bez unosa



Izvor: autor

6.4. Ispis kreiranog naloga te prethodni prikaz poruke

Ako zaposlenik unese sve tražene podatke o nalogu te fotografira artikl, klikom na ikonu „Kreiraj nalog“, nalog se sprema i prikazuje se poruka „Želite li isprintati kreirani nalog?“. Ova je poruka vidljiva na slici 24. Ako zaposlenik klikne „Printaj“, nalog se pretvara u tekstualni oblik te se prosljeđuje lokalnom printeru na ispis. Na ovom zaslonu zaposlenik odabire pisač s kojeg će ispisati nalog. Opcionalno može unijeti koliki broj kopija servisnog naloga želi ispisati, veličinu i vrstu papira na koji će ispisati servisni nalog. Nastavno na sliku 24, klikom na „Odustani“, zaposlenika se vraća na početnu stranicu s postojećim servisnim nalozima. Kada kupac donese neki predmet na servis, ispiše mu se kopija servisnog naloga koju potpiše, kao znak da se predmet nalazi u tvrtki na servisu. Kupac donosi servisni nalog na uvid kad mu zaposlenik javi da je servis obavljen. Slika 25 prikazuje nalog za printanje (ispis), na lokalnom mrežnom pisaču.

Slika 24: Prikaz poruke: "Želite li isprintati kreirani nalog?"

Izvor: autor

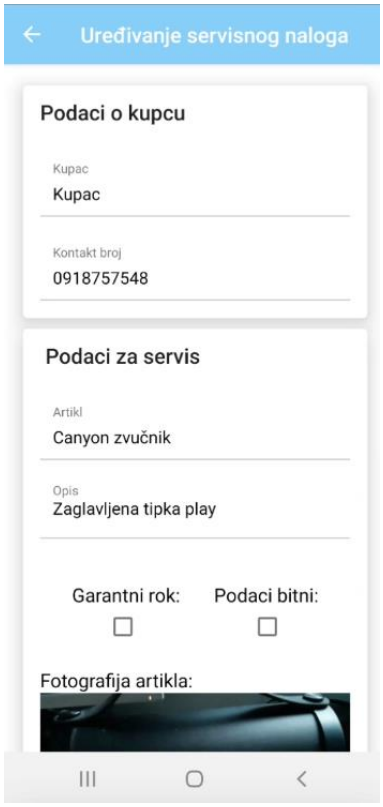
Slika 25: Prikaz naloga za printanje

Izvor: autor

6.5. Uređivanje i završavanje servisnog naloga

Pritiskom na ikonu „olovka“, zaposleniku se omogućuje uređivanje i završavanje servisnog naloga. Uređivanje služi kako bi zaposlenik unio promjene na postojećem nalogu. Npr. ukoliko postoji neka pravopisna pogreška, ukoliko je zaboravljeno dodati još neki predmet predan na servis ili ako je kupac zatražio dodatne radove. Nakon što je servis uspješno obavljen, zaposlenik upisuje izvršene usluge. Upisuje eventualni utrošeni materijal te cijenu odrađene usluge. Klikom na ikonu „Spremi izmjene“, spremaju se sve novonastale izmjene u postojeći nalog. Klikom na ikonu „Završi nalog“, aplikacija zaposlenika pita je li siguran da želi završiti nalog. Ukoliko potvrdi, nalog se premješta iz kreiranih, aktivnih naloga u izvršene naloge. U nastavku slijede 4 prethodno objašnjene slike : Prikaz zaslona za uređivanje i završavanje naloga (slika 26), Prikaz dijela zaslona za završavanje naloga (slika 27), Prikaz poruke: „Želite li završiti nalog?“ (slika 28) i Prikaz izvršenih servisnih naloga (slika 29).

Slika 26: Prikaz zaslona za uređivanje i završvavanje naloga

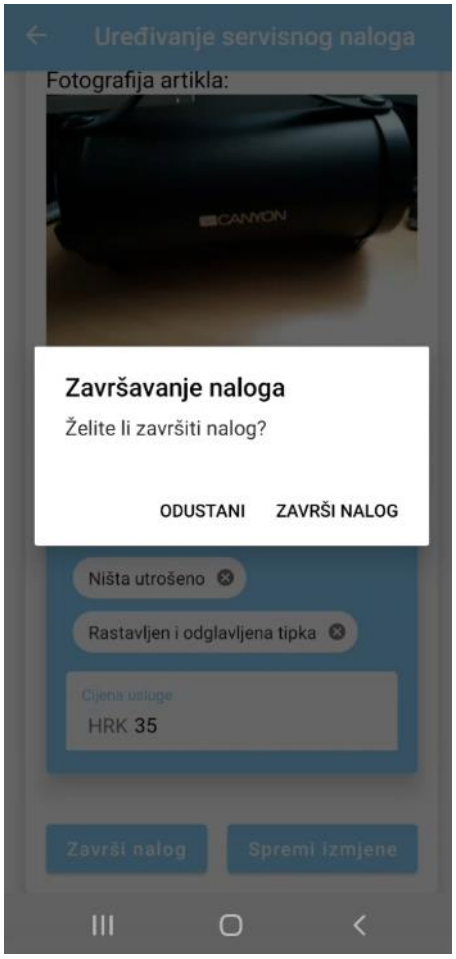


Izvor: autor

Slika 27: Prikaz dijela zaslona za završavanje naloga

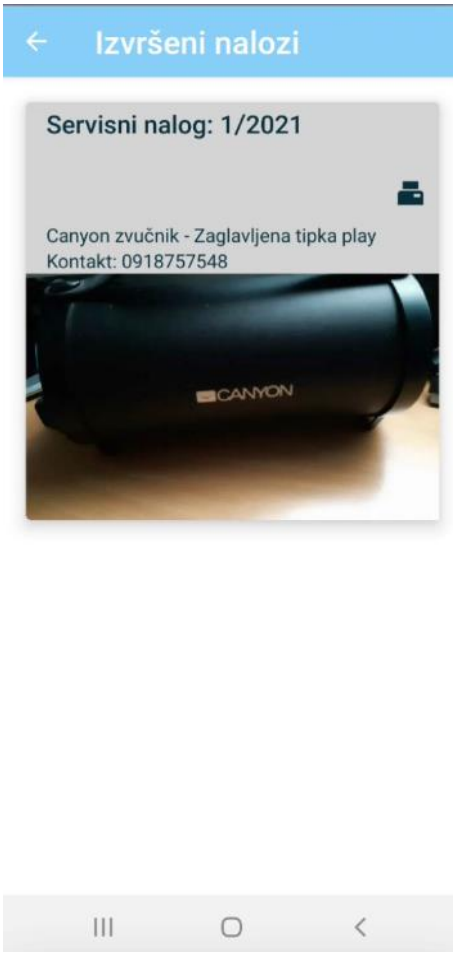
Izvor: autor

Slika 29: Prikaz poruke: "Želite li završiti nalog?"



Izvor: autor

Slika 28: Prikaz izvršenih servisnih naloga



Izvor: autor

6.6. Ispis izvršenih naloga i odjava zaposlenika

Pri završetku obavljanja servisa zaposlenik javlja kupcu da mu je predmet popravljen. Kada kupac dođe preuzeti predmet, zaposlenik ispisuje izvršeni servisni nalog na kojemu piše koje su usluge izvršene te koliko novaca je potrebno platiti. Iako, ovo je opcionalna mogućnost zbog toga što često bude nepotrebno kupcu pismeno dokazivati što je napravljeno nego mu se to usmeno predoči te izreče koliko je novaca potrebno platiti. Isto tako zbog uštede papira. Zaposlenik se pri završetku korištenja aplikacije odjavljuje klikom na ikonu „Odjava“. Kada se zaposlenik odjavi, ponovno se prikazuje početni zaslon aplikacije na kojem se dohvaćaju podaci o zaposlenicima. U

nastavku slijede slike na kojima je Prikaz izvršenog naloga za printanje (slika 30) te Prikaz poruke za odjavu zaposlenika (slika 31).

Slika 30: Prikaz izvršenog naloga za printanje

Izvor: autor

Slika 31:Prikaz poruke za odjavu zaposlenika

Izvor: autor

7. Zaključak

Sve serviserske tvrtke imaju potrebu za servisnim nalozima. Neke ih rade u programima kao što su MS Excel ili MS Word. Potreba za skladištenje svih servisnih naloga na jednom mjestu sve je veća. Ako kupac donese predmet s nekim neispravnim dijelom, tada je to najbolje zabilježiti slikanjem tog dijela. Kupci bi ponekad najradije okrivili servisere za uništene predmete, a to vrlo često nije istina.

Za izradu aplikacije uloženo je 6 mjeseci rada. Bilo je vrlo izazovno i problematično. Usprkos tome, na kraju se sve uspjelo riješiti. Primjerice, u ranoj fazi izrade aplikacije koristila se funkcija za poziv kamere „*RNCamera*“, koja se implementirala u aplikaciju. S vremenom funkcija više nije bila podržana te se više nije mogla koristiti. Morala se naći alternativa – funkcija „*launchCamera*“ iz biblioteke „*react-native-image-picker*“.

Redux kao upravitelj stanjima aplikacije ima veliku važnost u izradi aplikacija. Uvijek postoji upit za nečim, a zatim slijedi odgovor koji može biti ili uspješan ili neuspješan. On se kao slobodna biblioteka može koristiti u raznim frameworcima programskih jezika (uključujući Javascript). Aplikacija bi se u budućnosti mogla unaprijediti tako da se prava SQL baza podatka tvrtke E-computing implementira u postojeću aplikaciju. U tu bazu podataka bi se spremali podaci o kupcima koje bi aplikacija mogla dohvaćati.

Popis literature:

[1]https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript Pristupano 9.9.2021.

[2] https://www.c-sharpcorner.com/article/what-and-why-reactjs/ Pristupano 9.9.2021.

[3] https://www.netguru.com/glossary/react-native Pristupano 9.9.2021.

[4] https://redux.js.org/tutorials/essentials/part-1-overview-concepts Pristupano 9.9.2021.

[5] https://www.json.org/json-en.html Pristupano 9.9.2021.

[6] https://code.visualstudio.com/docs/editor/whyvscode Pristupano 9.9.2021.

[7] https://www.educative.io/blog/what-is-nodejs Pristupano 9.9.2021.

[8] https://firebase.google.com/docs Pristupano 9.9.2021.

[9] dr. sc. Ida Panev, Modeliranje procesa, prezentacija Pristupano 9.9.2021.

[10] dr. sc. Ida Panev, Modeliranje procesa, prezentacija Pristupano 9.9.2021.

Popis slika: