Sustav za pomoć u radu servisa za vozila

Veselin, Ivan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTRORTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

SUSTAV ZA POMOĆ U RADU SERVISA ZA VOZILA

Diplomski rad

Ivan Veselin

Osijek, 2019.

Sadržaj

1. UVOD

Cilj ovog diplomskog rada je istražiti način rada servisa za vozila, predložiti moguća unaprjeđenja radnog procesa korištenjem računalnih sustava, implementirati računalne sustave u obliku mobilne aplikacije i web poslužitelja te na kraju ponovno analizirati rad servisa koji koristi implementirane sustave. Mobilna aplikacija izrađena je za Android platformu, a web poslužitelj pomoću Node.js platforme. Mobilna aplikacija namijenjena je krajnjim korisnicima servisa za vozila. Kroz nju korisnici mogu obaviti registraciju i prijavu u sustav, urediti podatke o svom automobilu te obaviti prijavu na jedan od slobodnih termina servisa za vozila. Svrha web poslužitelja je pružanje upravljačkog sučelja vlasniku i popisa zaduženja radnicima servisa. Vlasniku je omogućeno kreiranje rasporeda slobodnih termina, pregled stanja trenutnog tjedna, pregled ne dodijeljenih zadataka i registriranje novih radnika. Nakon te registracije, radnicima je omogućena prijava i pregled zadanih te označavanje odrađenih radnih zadataka.

U drugom poglavlju opisan je trenutni način rada autoservisa i problemi koji se javljaju u svakodnevnom poslovanju te je opisano predloženo rješenje. U trećem poglavlju opisana je izvedba rješenja s detaljnim opisima funkcionalnosti uz priložene dijelove programskog koda. Zatim je opisan način testiranja sustava i rezultati toga testiranja.

1.1. Zadatak diplomskog rada

Kratko opisati način rada servisa za vozila i predložiti moguće poboljšanje u obliku sustava u dva dijela. Poslužiteljski dio omogućuje vlasniku pregled rezerviranih termina, poslova i zadavanje zadataka djelatnicima. Vlasniku automobila omogućiti traženje slobodnog termina za servis vozila s ponuđenim slobodnim terminima putem Android aplikacije. Izrađeni sustav testirati i navesti rezultate testiranja i rada sustava.

2. ANALIZA RADA SERVISA ZA VOZILA

U ovom poglavlju opisan je trenutni način rada malog servisa za vozila, poglavito u vidu komunikacije s krajnjim korisnicima i komunikacije između radnika unutar servisa. Također je objašnjen način optimizacije rada korištenjem računalnih sustava iz dva dijela: Android aplikacije i web poslužitelja.

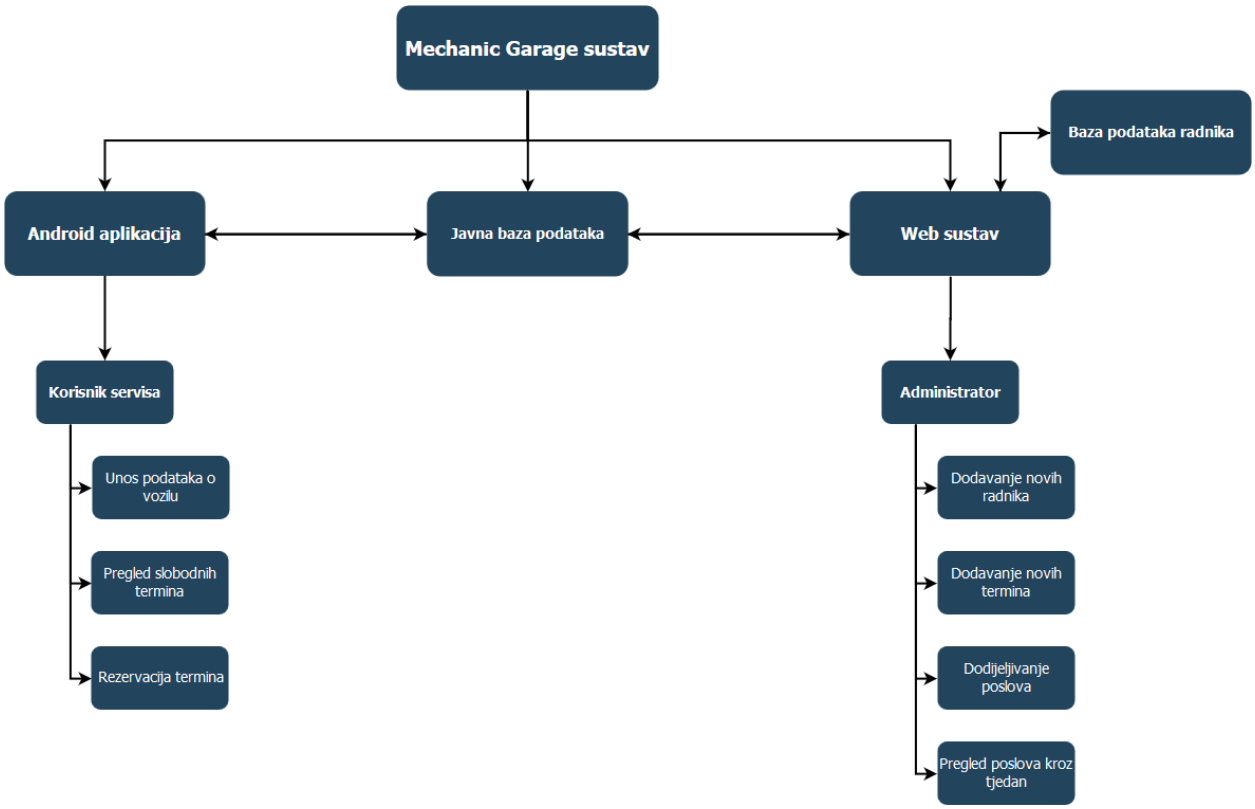
2.1. Trenutni način rada servisa

Opis trenutnog načina rada dobiven je promatranjem jednog radnog dana i razgovorom s vlasnikom jednog malog autoservisa. U servisu rade dvije osobe: vlasnik i zaposlenik mehaničar. Vlasnik, uz poslove vezane za djelatnost, obavlja i administracijske poslove potrebne za vođenje poslovanja. Također, komunicira s dobavljačima i prodajnim predstavnicima, s krajnjim mušterijama, dogovora cijene, termine i ostalo. Uz to, često je i ometan s telefonskim pozivima u obavljanju svoje primarne djelatnosti – popravljanja i servisiranja automobila. To dovodi do smanjenja efikasnosti i uvodi mogućnost grešaka u radu zbog dekoncentracije. Taj zaključak potvrdio je i vlasnik tijekom razgovora, gdje je izrazio frustraciju sa stalnim prekidanjima tijekom rada i problemima s kojima se susreće zbog toga što je obvezan imati stalnu interakciju s krajnjim korisnicima servisa. Ta interakcija većinom se tiče trivijalnih stvari kao što su dogovor obostrano zadovoljavajućeg termina, dogovor oko posla koji treba odraditi i sl. Čest izvor nesporazuma je nedostatak znanja krajnjih korisnika koji često ne znaju objasniti koje usluge su im potrebne. Još jedan problem koji se nazire iz razgovora s vlasnikom je i čimbenik ljudske zaboravnosti gdje bi vlasnik zaboravio naručiti potrebne dijelove za određeni posao ili bi naručio više vozila u isto vrijeme što je dovodilo do neugodnosti i, na kraju, do nezadovoljstva korisnika.

2.2. Opis predloženog poboljšanja

Problemi s kojima se susreću vlasnici malih servisa su kompleksni i nisu svi rješivi uvođenjem računalnih tehnologija. Ovaj rad fokusiran je na samo dio cjelokupnog problema: komunikaciju između vanjskih korisnika i servisa te na raspored poslova u servisu u vidu rasporeda poslova djelatnicima. Rješenje se sastoji od dva dijela. Prvi dio, Android aplikaciju, koriste krajnji korisnici za pregled slobodnih termina, naručivanje na željeni termin, opis potrebnog posla i sl. Drugi dio

je web sustav koji služi kao administratorsko sučelje za vlasnike servisa. Vlasnici servisa kroz sučelje pregledavaju naručene poslove, dodaju nove radnike u sustav, dodjeljuju poslove pojedinim radnicima, objavljuju raspored dostupnih termina za korisnike i sl. Funkcionalni dijagram sustava prikazan je na slici 2.1.



Sl. 2.1. Funkcionalni dijagram sustava

U korisničku Android aplikaciju korisnici se mogu sami registrirati tako što unesu svoju email adresu i postave svoju lozinku. Nakon registracije korisnici mogu unijeti podatke o svom automobilu, pregledati termine i rezervirati željeni termin s opisom posla koji treba obaviti.

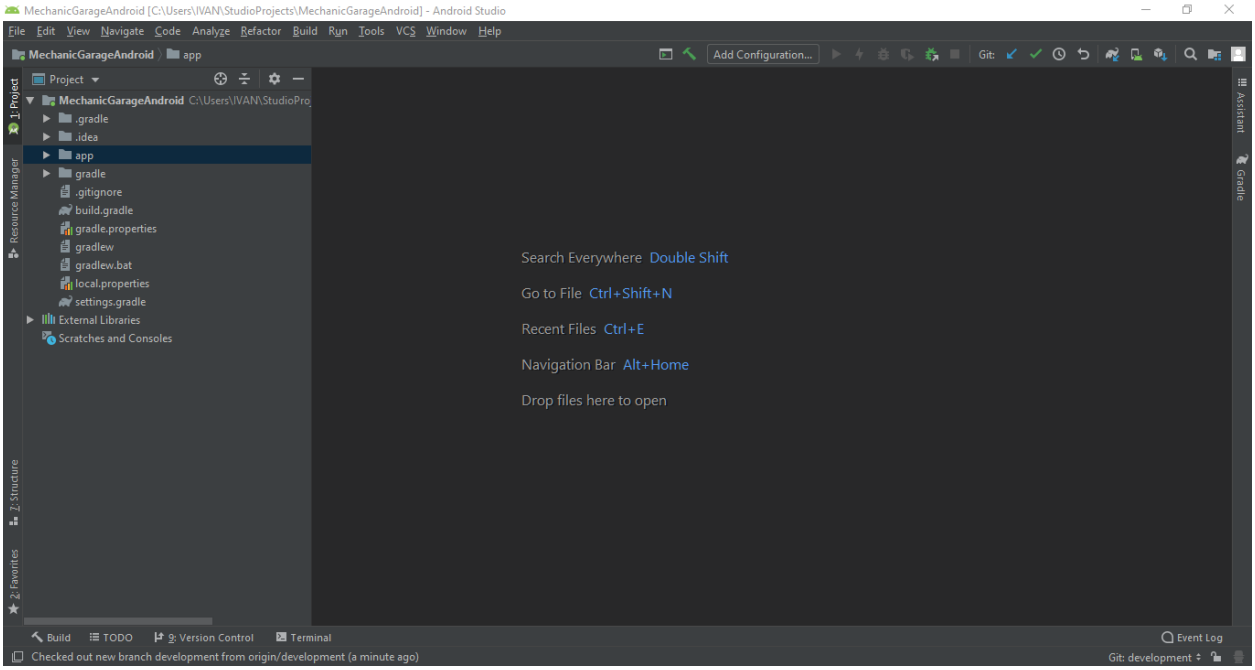
U web sustav mogu pristupiti samo prethodno registrirani korisnici. Korisnici koji imaju administratorske privilegije mogu registrirati nove korisnike/radnike koji onda s tim podatcima mogu pristupiti svojem računu. Ovisno o tome je li korisnik web poslužitelja administrator ili ne, mijenja se prikaz, odnosno dostupnost podataka. Administratori imaju pristup registraciji novih korisnika, pregledu svih popunjenih termina, kreiranju novih slobodnih termina i trenutnom stanju svih poslova u tijeku. Obični korisnici tj. radnici u servisu imaju pristup samo poslovima koji su dodijeljeni njima.

3. REALIZACIJA RAČUNALNIH SUSTAVA

U ovom poglavlju pobliže su opisani korišteni alati i tehnologije te je dan detaljan opis implementacije pojedinih komponenti sustava

3.1. Mobilna aplikacija

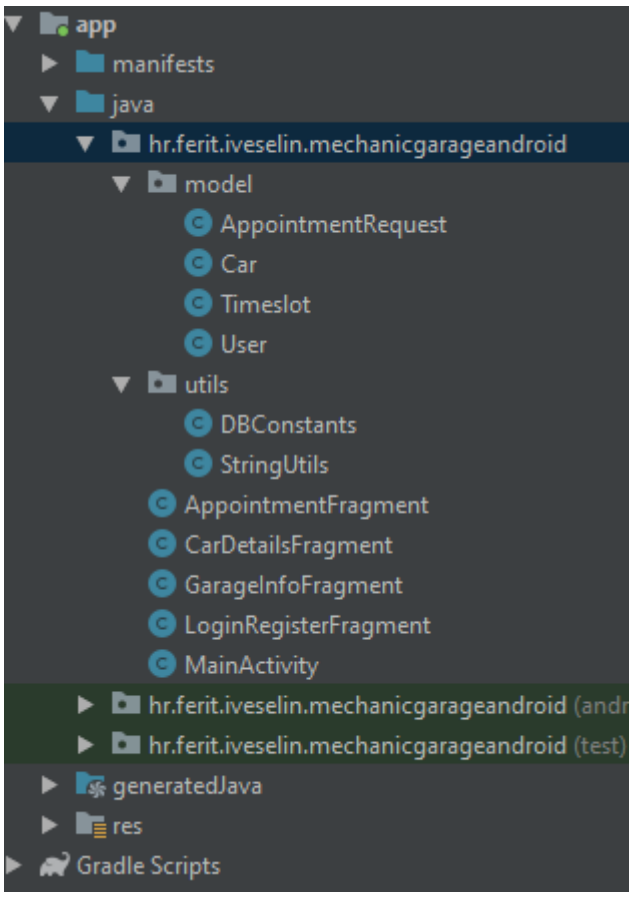
Mobilna aplikacija izrađena je za Android okruženje koristeći službeno razvojno okruženje (IDE – eng. *Integrated Development Environment*) cijele Android platforme – *Android Studio* . *Android Studio* nastao je kao proširenje popularnog alata *IntelliJ IDEA* tvrtke *JetBrains* koji se koristi primarno za *Java* okruženje. Korišteni programski jezik je *Java. Java* je objektno orijentirani programski jezik opće namjene koji se temelji na klasama i koristi na raznim uređajima jer se isti kod može izvršavati na svim uređajima čiji operacijski sustav podržava JVM (eng. *Java VirtualMachine*).



Sl. 3.1. Sučelje Android Studio okruženja

Odabrana arhitektura Android aplikacije je arhitektura s jednom aktivnosti (eng. Single Activity Architecture). Osnovni koncept iza ove arhitekture je imati samo jednu aktivnost u

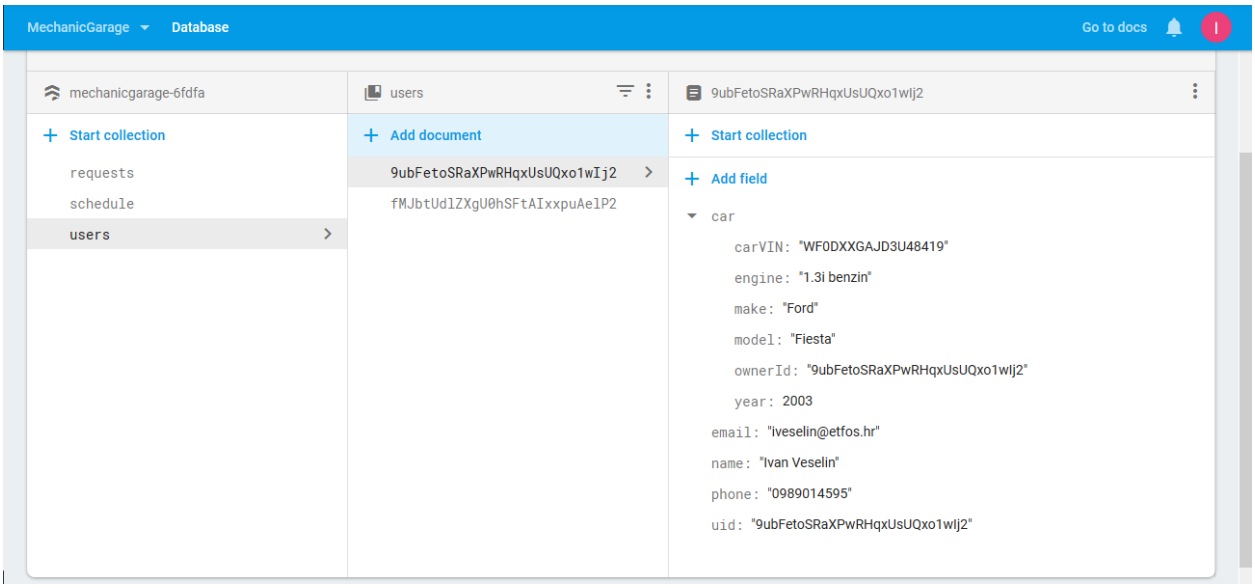
aplikaciji – (eng. *Activity*) i samo izmjenjivati sadržaj prikazan na ekranu korištenjem fragmenata. Arhitektura je odabrana jer je prikladna za jednostavna sučelja bez puno sadržaja zbog toga što otvaranje nove aktivnosti troši puno više resursa i dulje traje od jednostavne zamjene fragmenata unutar prozora aktivnosti. Zbog jednostavnosti aplikacije nisu korištene posebne arhitekture koda kao što su MVP(*Model-View-Presenter*) ili MVVM(*Model-View-ViewModel*), ali su datoteke podijeljene u logičke cjeline vrlo slično gore navedenim arhitekturama. U mapi *model* mogu se naći svi podatkovni modeli – u pravilu sve novo definirane klase u projektu. U *utils* mapi nalaze se pomoćne datoteke kao što su nekakve konstante ili pomoćnici za rad s tekstom u programskom kodu – string tip podataka i sl. Unutar korijenske mape nalaze se datoteke s kodom povezanim s Android sustavom – aktivnost i fragmenti.



Sl. 3.2. logički raspored programskog koda

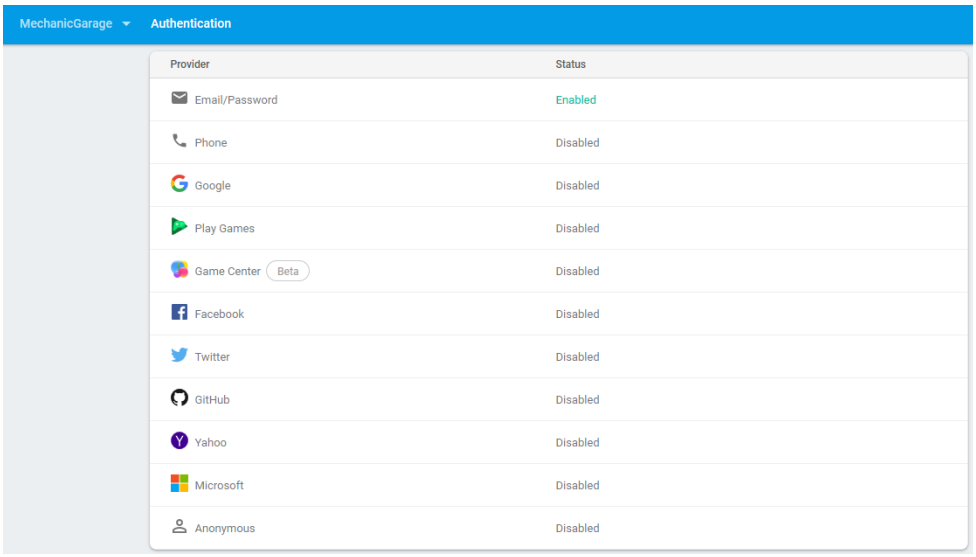
Kao javna baza podataka zajednička za mobilnu aplikaciju i web poslužitelj koristi se *Cloud Firestore*. Radi se NoSql bazi podataka u oblaku koja se bazira na principu kolekcija-dokument(Sl. 3.3.). Održava ju Google i nasljednik je popularne *Firebase* baze podataka. Podatci se spremaju u

obliku kolekcija koje sadrže dokumente koji su povezani logički, ali ne moraju nužno imati istu formu kao što je slučaj sa SQL bazama. Dokumenti mogu sadržavati osnovne tipove podatak kao što su tekst, brojevi i slično, ali mogu sadržavati i svoje podkolekcije što daje veliku slobodu u oblikovanju modela podataka koji je potreban.



Sl. 3.3. Primjer podataka u Cloud Firestore bazi

Još se koristi i *Firebase Authentication* [2] – SDK (eng. *Software Development Kit*) iza kojeg stoji Google i koji omogućuje jednostavnu registraciju i autentifikaciju u mobilnoj aplikaciji. SDK podržava različite načine autentifikacije, od najjednostavnije korištenjem email računa i lozinke do složenijih korištenjem Google ili Facebook računa. Svi podržani načini autentifikacije vidljivi su na slici 3.4.



Sl.3.4. Podržani načini autentifikacije

U izrađenoj Android aplikaciji koristi se najjednostavnija metoda – email adresa i lozinka koja je odabrana zbog svoje jednostavnosti i za korisnika i implementaciju.

Elementi vidljivi na ekranu i njihov raspored određuje se pomoću elemenata XML datoteka koje se spremaju u mapu *layout* koja se nalazi u mapi *res*. Kako bi mogli iz *Java* koda mijenjati, čitati ili detektirati akcije korisnika, potrebno je elemente iz XML datoteke povezati s odgovarajućom Android komponentom. Kako bi se taj proces olakšao, koristi se *Butterknife library* za *bindanje* elemenata. Na taj način smanjuje se broj ponavljajućih linija (eng. *boilerplate code*) što kod čini čitljivijim i razumljivijim.

Za navigaciju kroz aplikaciju koristi se *NavigationDrawer* [3], izbornik koji izlazi s lijeve strane uređaja pritiskom na tzv. *habmurger* ikonu ili povlačenjem s lijevog ruba ekrana. U izborniku se nalaze gumbi za sva sučelja dostupna korisniku. Ukoliko korisnik nije prijavljen, može pristupiti samo početnom zaslonu i zaslonu za prijavu/registraciju. Takva funkcionalnost se ostvaruje korištenjem *onAuthStateChanged listenera* koji se okida svaki puta kada se stanje autentifikacije promijeni. U ovisnosti o novom stanju, mijenjaju se i tekst i dostupnost gumba u izborniku(programski kod 3.1.).

```
@Override
public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
    FirebaseUser user = firebaseAuth.getCurrentUser();

    View headerView = navView.getHeaderView(0);
    TextView headerText = headerView.findViewById(R.id.nav_header_text);
```

```
    Menu menu = navView.getMenu();
    MenuItem login = menu.findItem(R.id.nav_register_login);
    MenuItem appointment = menu.findItem(R.id.nav_set_appointment);
    MenuItem details = menu.findItem(R.id.nav_car_details);

    if (user != null) {
        headerText.setText(user.getDisplayName());
        login.setTitle(R.string.logout item text);
        appointment.setEnabled(true);
        details.setEnabled(true);
    } else {
        headerText.setVisibility(View.GONE);
        login.setTitle(R.string.login_item_text);
        appointment.setEnabled(false);
        details.setEnabled(false);
    }
}
```

Programski kod 3.1. Odaziv na promjenu stanja autentifikacije

Pritiskom na dostupnu stavku u izborniku, okida se *onNavigationItemSelected listener* gdje se kliknuta stavka postavlja kao aktivna i, ovisno o kojoj stavci se radi, prikazuje pripadajući *fragment* koristeći *setFragment* metodu.

```
@Override
public boolean onNavigationItemSelected(@NonNull MenuItem item) {

    item.setChecked(true);
    drawerLayout.closeDrawers();
    Fragment fragment = null;
    switch (item.getItemId()) {
        case R.id.nav_register_login:
            if (firebaseAuth.getCurrentUser() == null) {
                fragment =
LoginRegisterFragment.newInstance(LoginRegisterFragment.LOGIN_FRAGMENT);
            } else {
                firebaseAuth.signOut();
            }
            break;
        case R.id.nav_info:
            fragment = GarageInfoFragment.newInstance();
            break;
        case R.id.nav_car_details:
            fragment = CarDetailsFragment.newInstance();
            break;
        case R.id.nav_set_appointment:
            fragment = AppointmentFragment.newInstance();
            break;
    }

    if (fragment == null) {
        fragment = GarageInfoFragment.newInstance();
    }

    setFragment(fragment);
    return false;
```

```
}


private void setFragment(Fragment fragment) {
    FragmentManager fragmentManager = getSupportFragmentManager();
    FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();

    fragmentTransaction.replace(R.id.fragment_container,
fragment).addToBackStack(null);
    fragmentTransaction.commit();
}
```

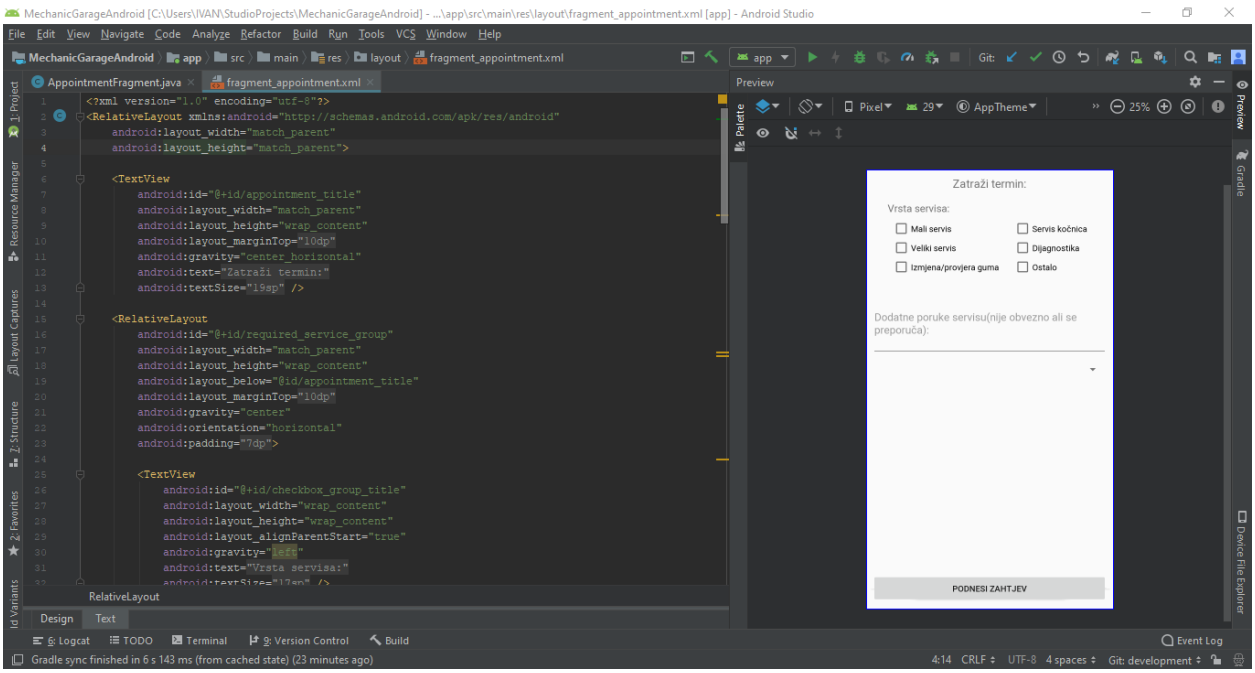
Programski kod 3.2. Odziv na odabir u izborniku

Osim spremanja osnovnih podataka o korisniku, omogućeno je i spremanje podataka o automobilu koji se koriste od strane radnika servisa kako bi mogli naručiti pripadajuće dijelove. Podatci o automobilu spremaju se kao objekti klase *Car* unutar dokumenta u kolekciji *users*. Točan dokument u koji se spremaju podatci pronalazi se po *id-u* prijavljenog korisnika. Ukoliko korisnik već ima spremljene podatke o automobilu, ponuđeno mu je ažurirati iste. Prilikom pritiska na gumb za spremanje, upisani podatci se validiraju i spremaju u *Firestore* bazu.

```
@OnClick(R.id.submit_car)
void onCarSubmitClicked() {
    if (!inputsValidated()) {
        return;
    }
    Car carToSave = new Car(user.getUid(), carMake, carModel, carYear,
carEngine, carVIN);

    setUserData(carToSave);
    Log.d(TAG, "onCarSubmitClicked: saving a car: " + carToSave.toString());
}
```

Programski kod 3.3. Metoda pozvana nakon pritiska na spremanje

Prijava korisnika na dostupne termine obrađuje se unutar *AppointmentFragmenta*, gdje je korisniku ponuđena forma s kućicama za višestruko označavanje, poljem za unos teksta tipa *EditText* i padajućim izbornikom tipa *Spinner*. U padajućem izborniku nalaze se termini koje je administrator mehaničarske radionice postavio kao dostupne. Na slici 3.5 vidljiv je dio XML koda datoteke *fragment_appointment.xml* koja predstavlja sučelje *AppointmentFragmenta.*



Sl.3.5. *fragment_appointment.xml* otvoren u Android Studio okruženju

Prilikom klika na gumb za podnašanje zahtjev, aktivira se *onSubmitRequestClicked listener,* metoda u *AppointmentFragment* klasi koja provjerava valjanost unesenih podataka. Ukoliko su podatci ispravni poziva se metoda *addUserRequest* koja prima prima objekt klase *AppointmentRequest* i sprema ga u kolekciju *requests* sa auto generiranim *id*-em. Taj *id* se zatim spremi u odabrani termin iz kolekcije *schedule*.

```
@OnClick(R.id.submit_service_request)
void onSubmitRequestClicked() {
    if (markedCheckboxes.isEmpty()) {
        Toast.makeText(getActivity().getApplicationContext(),
R.string.no_service_checked_error, Toast.LENGTH_SHORT).show();
        return;
    }

    String extraNote = extraNoteInput.getText().toString().trim();
    List<String> serviceTypes = new ArrayList<>(markedCheckboxes);
    AppointmentRequest request = new AppointmentRequest(serviceTypes,
extraNote, user.getUid(), timeSlot.getTime());
    Log.d(TAG, "onSubmitRequestClicked: saving request: " +
request.toString());

    addUserRequest(request);
}

private void addUserRequest(final AppointmentRequest request) {
    firestoreDB.collection(DBConstants.DB_REQUESTS_COLLECTION).add(request)
            .addOnSuccessListener(new OnSuccessListener<DocumentReference>()
{
```

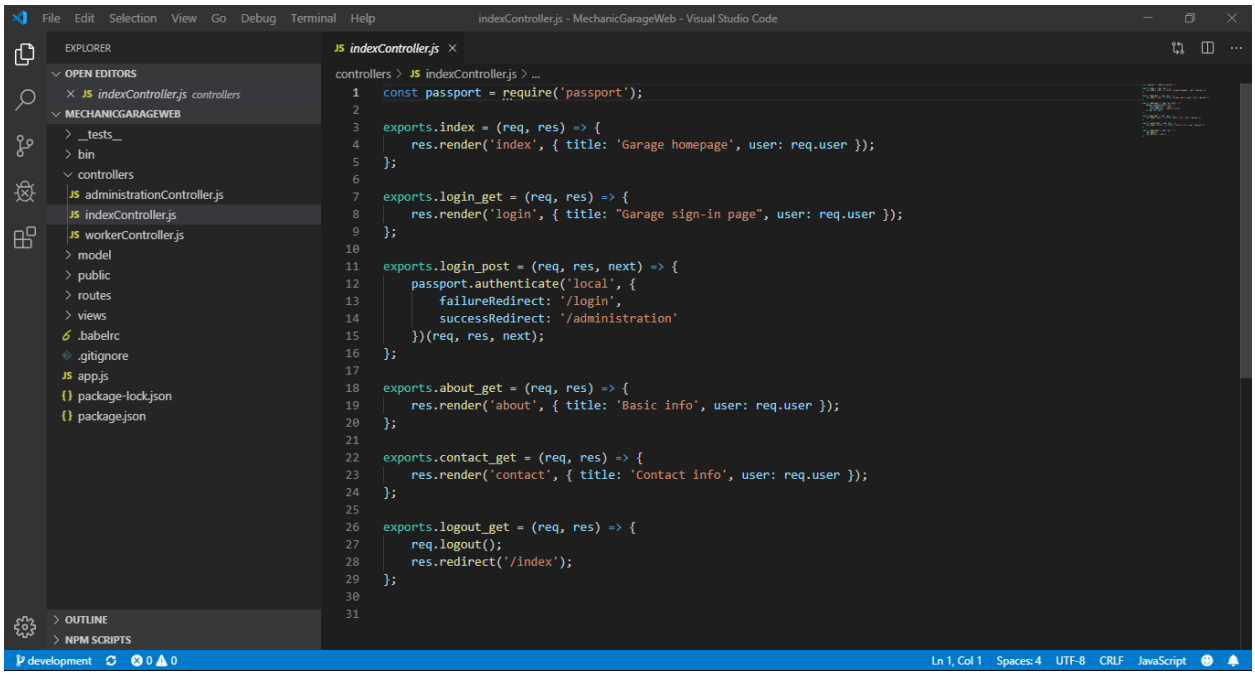
```
                @Override
                public void onSuccess(DocumentReference documentReference) {
                    Log.d(TAG, "onSuccess: request saved: " +
documentReference.getId());
                    timeSlot.setReqId(documentReference.getId());
                    timeSlot.setTaken(true);
                    requestSaved();
                }
            })
            .addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Log.d(TAG, "onFailure: failed" + e.getMessage());
                }
            });

}
```

Programski kod 3.4. Metode koje se pozivaju prilikom odabira termina

3.2. Web poslužitelj

Aplikacija web poslužitelja pisana je u *Javascript* (skraćeno *JS*) programskom jeziku i izvodi se u *Node.js*[4] okruženju. *Javascript* je skriptni jezik prvotno namijenjen izvođenju u internet preglednicima na klijentskom računalu. Objavom *Node.js* okruženja omogućeno je izvršavanje *JS* koda na računalima i izvan preglednika. Time su otvorena vrata da se *JS* koristi i kao programski jezik poslužiteljskih aplikacija. Programsko rješenje oslanja se na *Express*[5], web *framework* pisan za *Node.js*. Za razvojno okruženje odabran je *Visual Studio Code*, besplatni alat otvorenog koda koji je nastao pod pokrićem *Microsoft-a* i dalje razvijen od strane raznih volontera. Pruža podršku za razne programske jezike i prilagodbu mogućnosti instaliranjem raznih proširenja iz on-line baze.

Poslužitelj je razvijen primjenom MVC (eng. *Model-View-Controller*) arhitekture. *Model* predstavljaju podatkovni modeli koji se koriste unutar aplikacije, *view* su dijelovi koda koji služe za prikaz sadržaja na ekranu i nemaju nikakvu logiku vezanu za sam rad sustava. *Controlleri* obavljaju svu poslovnu logiku sustava. MVC sa sobom donosi podjelu koda u funkcionalne cjeline tj. prati *separation of concerns* princip kao što je vidljivo na slici 3.6.



Sl. 3.6. Sučelje *Visual Studio Code* okruženja i prikaz MVC arhitekture

Kao primarna baza podataka poslužitelja odabrana je *MongoDB,* NoSql baza u kojoj se pohrana bazira na dokumentima sa strukturom podataka sličnom JSON-u(eng. *JavaScript Object Notation*). Umjesto instalacije lokalne verzije *MongoDB*, koristi se online *Database-as-a-Service mLab* koji je besplatan za korištenje do granice od 500 MB podataka. Na taj način osiguravamo dosljednost podataka u slučaju pada sustava i dr.

Umjesto direktnog rada s bazom, koristi se *mongoose*[6] alat koji omogućuje stvaranje modela podataka i korištenje tih modela za spremanje/dohvaćanje podataka. *Mongoose* omogućava i dodavanje *plugin*-ova koji obavljaju dodatne radnje s podatcima. U ovom slučaju dodan je *passport-local-mongoose*, plugin koji služi za autentifikaciju tako što povezuje *Passport.js,* popularni *middleware* za *Node.js,* i *User* model, obavlja enkripciju lozinki i dr.

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var passportLocalMongoose = require('passport-local-mongoose');

var UserSchema = new Schema({
    username: String,
    password: String,
    role: {
        type: String,
        enum: ['owner','worker'],
        default:'worker'
    },
    tasks: [{
        type:String
    }]
});

UserSchema.plugin(passportLocalMongoose);

module.exports = mongoose.model('User', UserSchema);
```

Programski kod 3.5. *User* model podataka s dodanim *plugin*-om za autentifikaciju

Upotrebom *Passport.js middlewarea*, odvojen je pristup različitim dijelovima aplikacije za različite korisnike. Neprijavljeni korisnici mogu pristupiti početnoj stranici s osnovnim podatcima o servisu, stranici s kontakt informacijama i stranici za prijavu. Nakon uspješne prijave, ovisno o ulozi koju korisnik ima (admin ili radnik), korisnik može pristupiti odgovarajućim stranicama.

Poslužitelj ima i pristup podatcima u *Firestore* bazi preko *Firebase Admin SDK* [7] paketa koji nam omogućuje čitanje i pisanje podataka u *Firestore* bazu s web poslužitelja. Pristup se koristi kako bi se objavio raspored dostupnih termina u garaži, pročitale narudžbe iz rezerviranih termina i ostalo.

Umjesto ručnog pisanja HTML-a upotrebljava se *template engine* ugrađen u *Express.js*. Odabrani jezik je *pug* koji može primati podatke i ovisno o njima kompajlirati se u odgovarajući HTML. *Pug* podržava i umetanje sadržaja jedne *pug* datoteke u drugu što je iskorišteno za izdvajanje sadržaja koji se ponavlja na svim stranicama kao što su zaglavlje i podnožje stranice, bočni prostor i sl.

```
extends ../layout

block content
    .container-fluid.text-center
        .row.content
            include ../includes/left_sidebar

            .col-sm-8.text-left
                h4 Zadani posao:
                br
                each file, i in data
                    if file.done
                        a.alert.alert-
success(href="/administration/requests/"+file.object_id) #{i+1}.
#{moment(file.time*1000).format('DD.MM.YYYY  HH:mm')}
                    else if !file.worker
                        a.alert.alert-
danger(href="/administration/requests/"+file.object_id) #{i+1}.
#{moment(file.time*1000).format('DD.MM.YYYY  HH:mm')}
                    else
                        a.alert.alert-
warning(href="/administration/requests/"+file.object_id)  #{i+1}.
#{moment(file.time*1000).format('DD.MM.YYYY  HH:mm')}
                    hr
                else
                    h3 No data

            include ../includes/right_sidebar
```

Programski kod 3.6. Primjer *pug* koda za početnu stranicu radnika

U programskom kodu 3.6. pokazane su prednosti korištenja *puga* u vidu umetanja drugih *pug* datoteka, petlje koja prolazi kroz primljene podatke i za svaki podatak dodaje HTML element na ekran, *if* grananja za prikaz prikladnih poruka i dr.

Za pozivanje određenih funkcija u kontrolerima(eng. *routing*) koristimo se ugrađenim *routerom Express.js frameworka*. *Router* ima metode koje odgovaraju HTTP zahtjevu na pripadajuću adresu, npr. za obradu GET zahtjeva na početnoj stranici poziva se *get()* metoda *routera* s pripadajućim parametrima. Primjer je prikazan dolje.

```
var express = require('express');
var router = express.Router();
var index_controller = require('../controllers/indexController');

//homepage route(no auth needed)
router.get(['/', '/index'], index_controller.index);

//login routes
router.route('/login')
    .get(index_controller.login_get)
    .post(index_controller.login_post);

router.get('/about', index_controller.about_get);

router.get('/contact', index_controller.contact_get);

//logout request handling
router.get('/logout', index_controller.logout_get);

module.exports = router;
```

Programski kod 3.7. Korištenje ugrađenog *routera*

```
exports.index = (req, res) => {
    res.render('index', { title: 'Garage homepage', user: req.user });
};
```

Programski kod 3.8. Kontroler funkcija za obradu GET zahtjeva na index ruti

Kako se sva logika nalazi u kontrolerima, *router* zahtjeve preusmjerava na odgovarajuće funkcije kontrolera koji na zahtjev odgovara. Funkcije koje odgovaraju na zahtjeve moraju primiti *req i res* objekt. *Req* objekt (eng. *Request*) u sebi zadrže podatke poslane u zahtjevu, a *res*(eng. *Response*) objekt se koristi za odgovor na zahtjev. Primjer je vidljiv u programskom kodu 3.8. gdje

funkcija *index* odgovara na zahtjev tako što u odgovor renderira *pug* datoteku imena *index* s prikladnim naslovom i *user* objektom(ako je prisutan). Na tom principu bazira se cijela aplikacija web poslužitelja, samo se pozivaju prikladne funkcije za određene web adrese.

Za razliku od Android aplikacije, gdje se krajnji korisnici mehaničarske radionice mogu sami registrirati, na poslužitelj nove korisnike(radnike) dodaje vlasnik/administrator sustava. To je omogućeno tako što je funkcija koja obrađuje zahtjev za stranicom za registraciju stavljena u *administrationController* koji je zaštićen *middleware* funkcijom koja provjerava autentifikaciju i autorizaciju[8].

```
exports.auth_check = (req, res, next) => {
    if (req.isAuthenticated() && req.user.role == "owner") {
        return next();
    } else {
        res.redirect('../worker');
    }
};
```

Programski kod 3.9. *middleware za provjeru autentifikacije i autorizacije*

U gore priloženom programskom kodu provjeru autentifikacije izvodi ugrađena funkcija u *req* objektu *isAuthenticated*, dok provjeru autorizacije izvodi provjera *role* korisnika. Ova *middleware* funkcija izvodi se prije svih ruta koje pripadaju *administrationControlleru* i na taj način osigurava da je svaki zahtjev dozvoljen i pravovaljan za obradu.

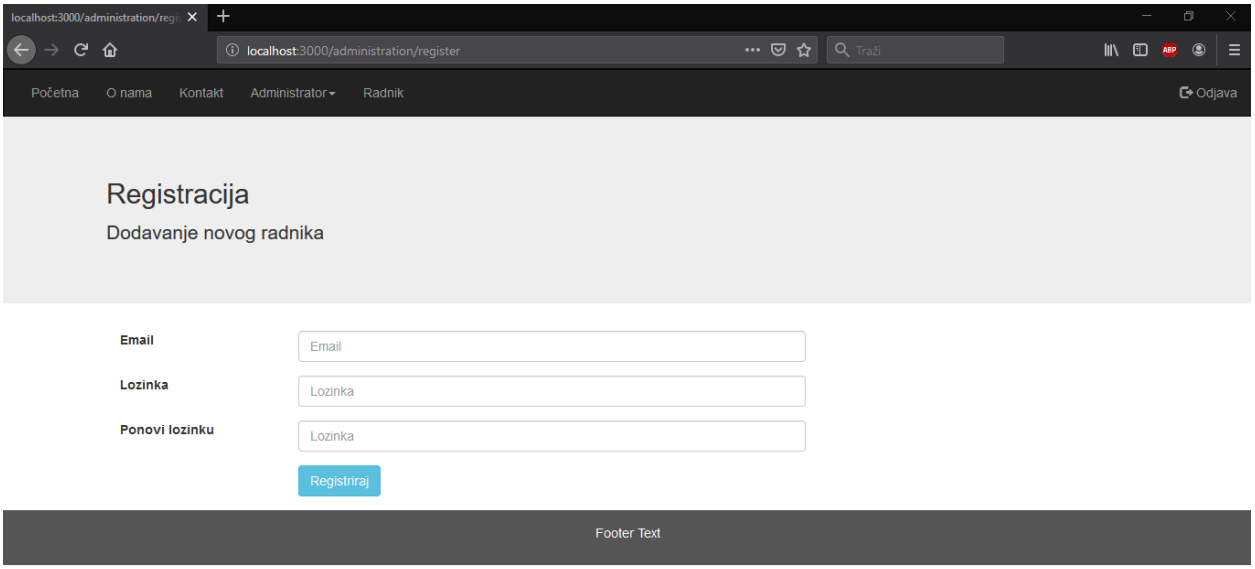
Postupak kojim sustav obrađuje zahtjev za registracijom novog korisnika je slijedeći:

1. Web stranica nakon klika na „*Registriraj novog radnika*“ šalje *GET* zahtjev na relativnu adresu *administration/register*
2. Zahtjev dolazi na server gdje se zahtjev prebacuje na *adminRouter* što aktivira *auth_check middleware* (programski kod 3.9.) koji provjerava autentifikaciju i autorizaciju. U slučaju greške zahtjev se prebacuje na *workerRouter*, a u slučaju prolaska zahtjev prolazi na rutu */register* gdje se aktivira *get* funkcija *routera* koja zahtjev predaje *register_worker_get* funkciji *administrationControllera.*

```
//register routes
router.route('/register')
  .get(administration_controller.register_worker_get)
  .post(administration_controller.register_worker_post);
```

Programski kod 3.10. dio *admin_routera* koji obrađuje zahtjeve na *administration/register* rutu

3. Zahtjev se obrađuje u kontroleru tako što se renderira *register.pug* datoteka u odgovor (eng. *response*).



Slika 3.7. Renderirana stranica *register.pug* predloška

4. Nakon što korisnik unese tražene podatke i klikne na „*Registriraj*“ forma šalje *POST* zahtjev na adresu *administration/register* koji ponovno prolazi kroz *router,* ali ovaj puta se poziva *register_worker_post* funkcija kontrolera
5. Prvi korak u *register_woker_post* funkciji je validacija primljenih podataka i ukoliko su valjani, sprema se novi korisnik koristeći funkciju *register* iz *User* modela te se novi korisnik sada može prijaviti s dodijeljenim podatcima. Ukoliko podatci nisu valjani, funkcija vraća ponovno renderirani *register.pug,* ali s dodanim podatcima o pronađenim greškama.

```
//additional worker registration request process
exports.register_worker_post = async (req, res, next) => {
    // Validation
    req.checkBody('username', 'Email is required').notEmpty();
    req.checkBody('username', 'Email is not valid').isEmail();
    req.checkBody('password', 'Password is required').notEmpty();
```

```
    req.checkBody('password2', 'Passwords do not
match').equals(req.body.password);

    const errors = req.validationErrors();
    if (errors) {
        res.render('register', {
            errors: errors
        });

    } else {
        const email = req.body.username;
        const password = req.body.password;
        try {
            await User.register(new User({ username: email }), password);
            res.redirect('/administration');
        } catch (err) {
            return next(err);
        }
    }
};
```

Programski kod 3.11. funkcija za registraciju novih korisnika

Kako bi korisnici u Android aplikaciji mogli odabrati termin na koji se žele prijaviti, ti termini trebaju biti određeni od strane vlasnika radionice. Na taj način je osigurano da vlasnik radionice može rasporediti druge poslove unutar fiksnog radnog vremena. Web poslužitelj raspored objavljuje u *Firestore* bazu pod kolekcijom *schedule* kojoj onda pristupa Android aplikacija i po njima pravi odabire u padajućem izborniku. Podatci se spremaju u obliku *UNIX* vremenskih oznaka(eng. *timestamp*) jer je takav zapis jednoznačan na svim sustavima. Kada korisnik napravi željeni raspored i odabere spremanje, ugrađena forma šalje *POST* zahtjev na *administration/schedule* adresu. Zahtjev se preko *routera* šalje na *schedule_post* funkciju *administrationControllera.* Ta funkcija podatke validira tako što prvo provjeri jesu li svi potrebni podatci primljeni, a zatim provjeri i u *Firestore* bazi je li raspored za odabrani tjedan već popunjen. Ukoliko sve prođe uredu sustav sprema sve odabrane termine u *Firestore* bazu koristeći *batch* zapis. *Batch* zapis se koristi jer je potrebno napraviti poseban dokument u kolekciji za svaki odabrani termin.

```
let docReference;
const batchWrite = db.batch();
data.forEach(timestamp => {
      docReference = scheduleReference.doc();
```

```
      batchWrite.set(docReference, { time: parseInt(timestamp, 10), week: week,
taken: false, });
});
try {
      await batchWrite.commit();
      res.redirect('/administration');
} catch (error) {
      return next(error);
}
```

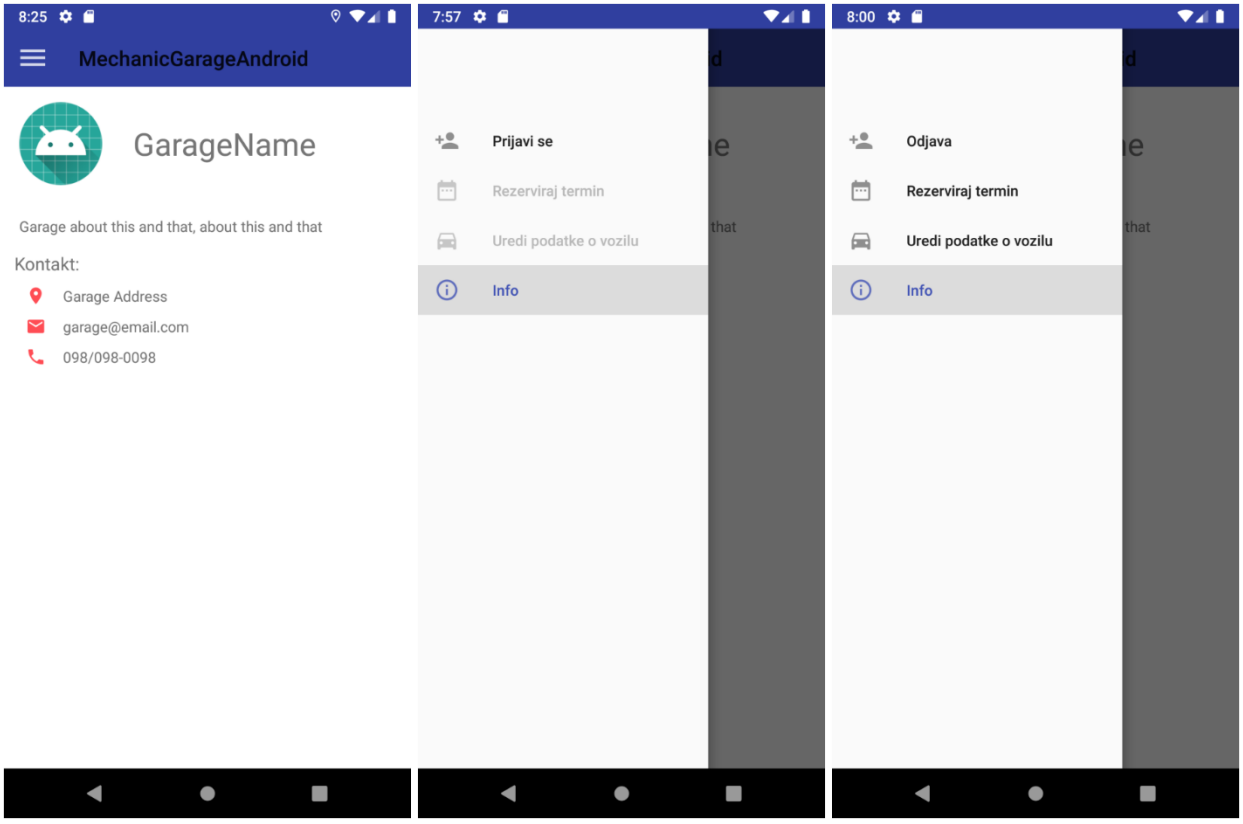
Programski kod 3.12. *batch* zapis u *schedule* kolekciju

4. REZULTATI IMPLEMENTACIJE SUSTAVA

Rezultati implementacije dobiveni su simulacijom uporabe cjelokupnog sustava od strane svih kategorija korisnika – vlasnika/administratora mehaničarskog servisa, radnika u servisu i krajnjeg korisnika usluga servisa. Kako je Android aplikacija namijenjena samo krajnjim korisnicima, neće se razmatrati njeno korištenje od strane vlasnika i radnika servisa.

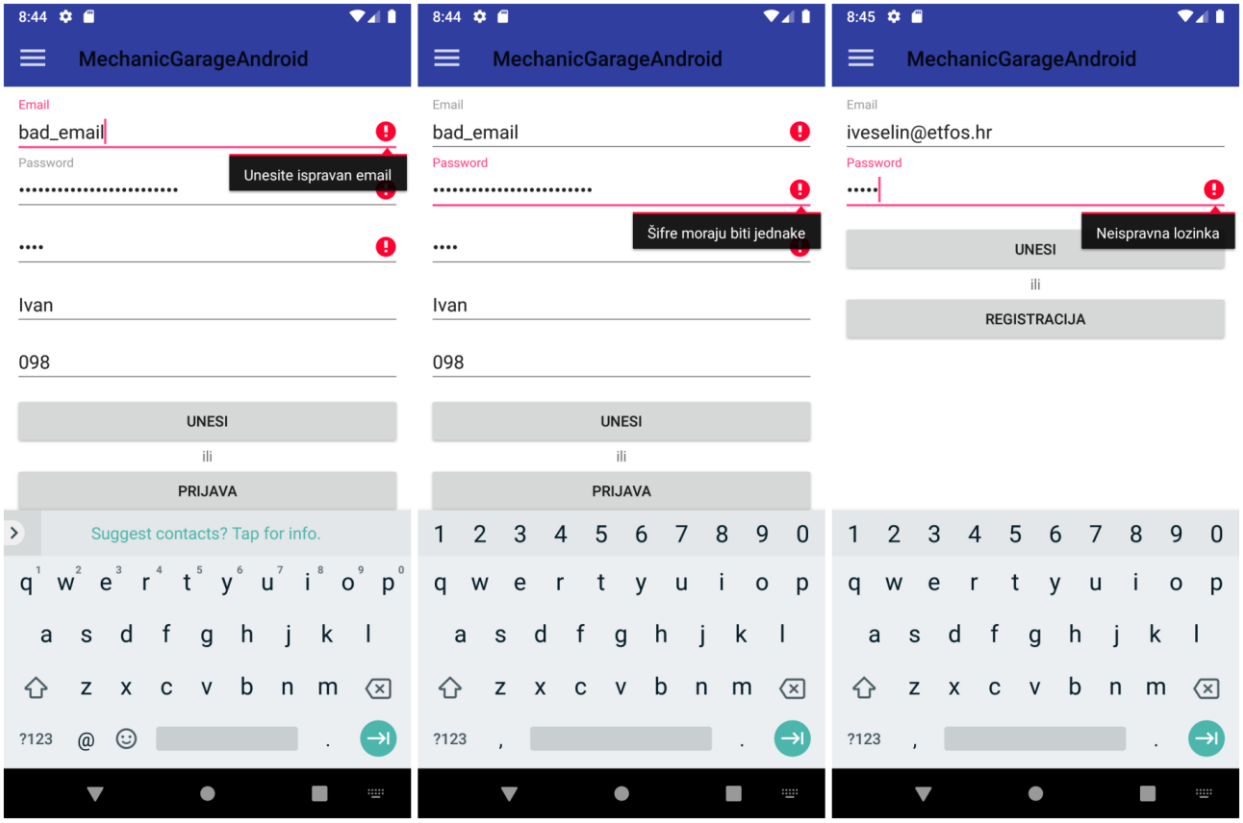
4.1. Testiranje Android aplikacije

Testiranje Android aplikacije obavljeno je na više uređaja što je preporučena praksa zbog velikog broja različitih konfiguracija hardvera, veličine ekrana i dr. na uređajima u Android platformi. Slike zaslona aplikacije priložene dolje uhvaćene su na emuliranom uređaju *Pixel 3* s verzijom sustava *Android 9* kodnog imena *Pie*. Na prvom izresku zaslona prikazan je početni ekran koji vide svi korisnici nakon otvaranja aplikacije. Drugi i treći zaslon prikazuje izgled bočnog izbornika (*Navigation Drawer*) u ovisnosti je li korisnik prijavljen ili ne.



Sl. 4.1. Početni zaslon i bočni izbornik u ovisnosti o prijavi korisnika

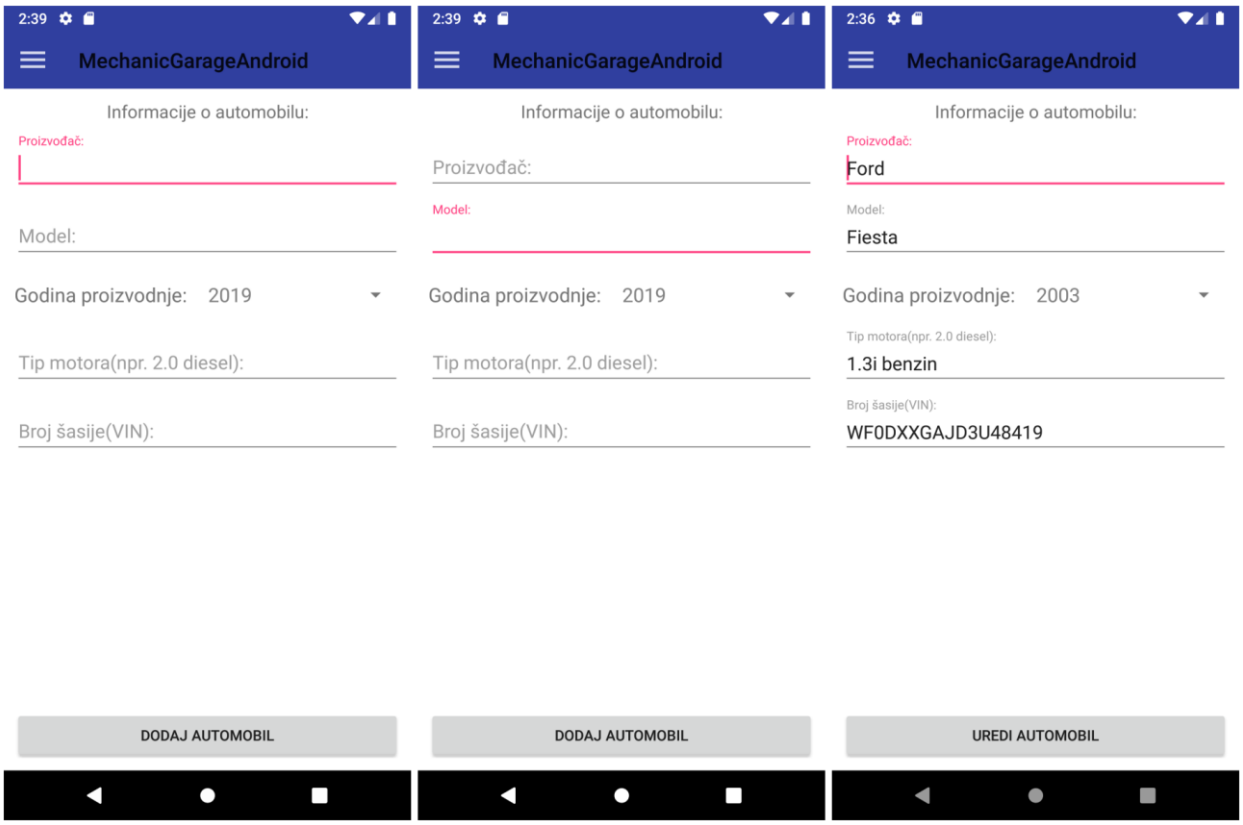
Prilikom prijave ili registracije provjerava se ispravnost unesenih podataka. Ukoliko uneseni podatak nije točan, prikazuje se prikladna greška na polju koje ima pogrešnu informaciju. Time je osigurana točnost informacija u sustavu i izbjegavaju se slučajne pogreške u poljima sa sakrivenim unosom(npr. postavljanje lozinke). Pri registraciji provjerava se valjanost email adrese i istovjetnost unesenih lozinki dok se prilikom prijave provjerava ispravnost lozinke.



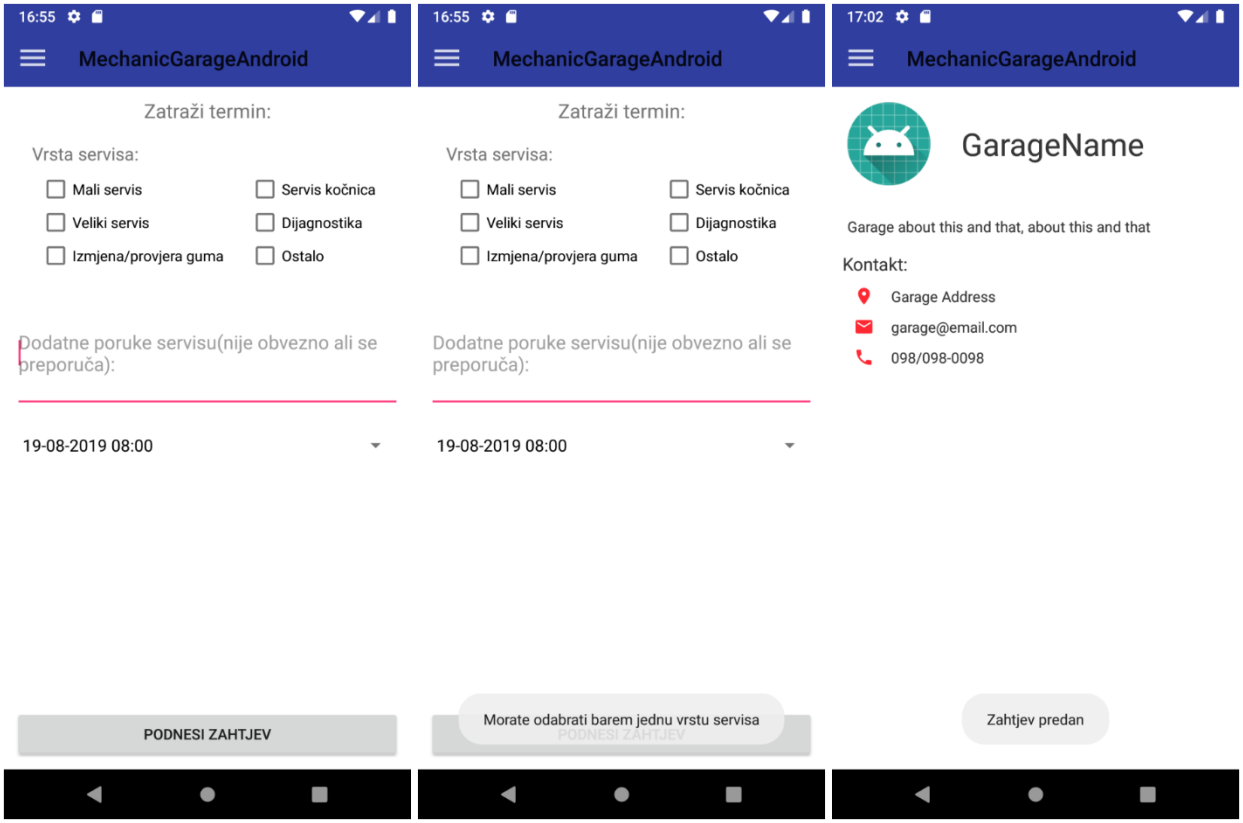
Sl. 4.2. Greške validacije prilikom registracije/prijave

Nakon registracije i prijave korisnik ima mogućnost upisivanja podataka o svom automobilu ili odabira i rezervacije termina koji mu odgovara. Prilikom upisa podataka o automobilu aplikacija provjeri ima li prijavljeni korisnik već upisan automobil, ukoliko ima polja se ispune postojećim podatcima pa ih korisnik može promijeniti, a ukoliko nema polja su ispunjena s naznakama o kojim se podatcima radi. (Slika 4.3.)

Prilikom rezervacije termina, aplikacija daje korisniku mogućnost odabira jednog ili više vrsta servisa koji su mu potrebni, mogućnost unosa dodatnih poruka servisu (neobavezno) i odabira termina iz liste ponuđenih. Ukoliko prilikom rezervacije nije odabran ni jedan ponuđeni servis, aplikacije ispisuje prikladnu poruku na ekran. (Slika 4.4.)



Sl. 4.3. Unos podataka o automobilu u mobilnoj aplikaciji

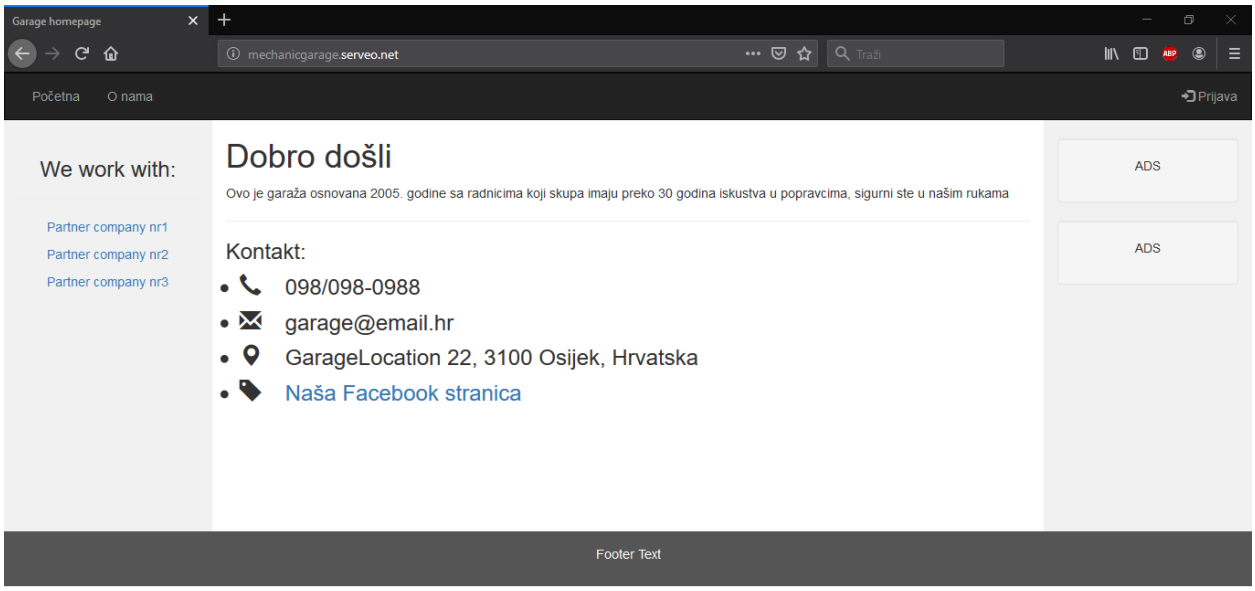


Sl. 4.4. Rezervacija termina za servis

4.2. Testiranje web poslužitelja

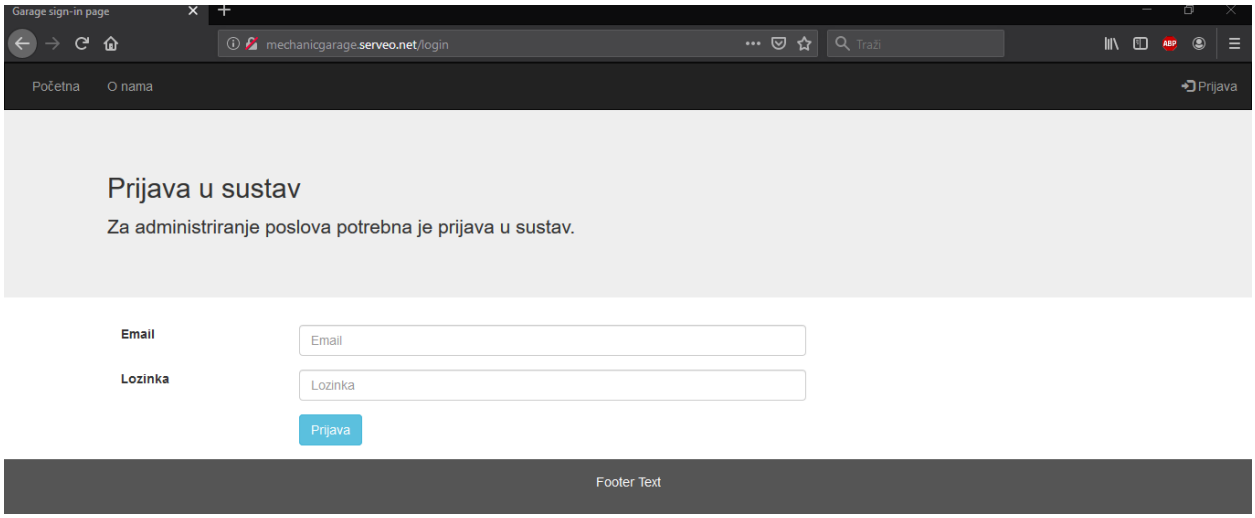
Testiranje web poslužitelja obavljeno je u više web preglednika što je uobičajena praksa zbog različitih načina rada pojedinih preglednika. Za priložene izreske zaslona korišten je *Mozilla Firefox 68.0.2.*

Početni zaslon isti je za sve korisnike neovisno jesu li prijavljeni ili ne i prikazuje ekran dobrodošlice s osnovnim kontakt informacijama. Javno je još dostupan zaslon s podatcima o garaži. Za pristup ostalim računima potrebno je biti prijavljen i ukoliko im se proba pristupiti bez prijave automatski se otvara zaslon za prijavu.

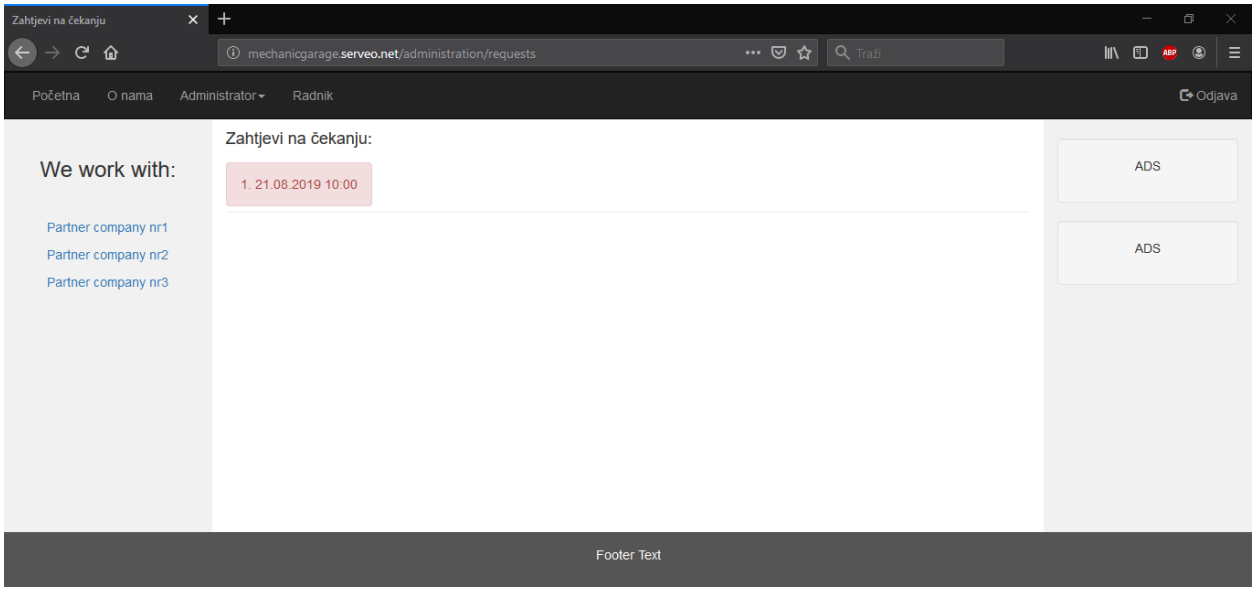


Sl. 4.5. Početni stranica web poslužitelja

Prilikom predaje aplikacije web poslužitelja vlasniku/administratoru isti dobiva administratorski račun kojim dalje upravlja sa sustavom. Na slici 4.6. prikazan je zaslon za prijavu gdje korisnik unosi svoje podatke i ovisno o svojoj ulozi vidi prikladan popis zadataka. Administrator vidi popis zadataka koji nisu nikome dodijeljeni za rad jer je to najveći prioritet, a radnik vidi popis zadataka dodijeljenih njemu. Na slici 4.7. vidi se zaslon s popisom zadataka koji vidi administrator nakon prijave. Administrator također ima i uvid u zadatke koji se odvijaju u trenutnom radnom tjednu. Odrađeni zadatci su označeni zeleno, dodijeljeni zadatci narančasto, a crveni su ovotjedni zadatci koji nisu nikome dodijeljeni. (Sl. 4.8.)

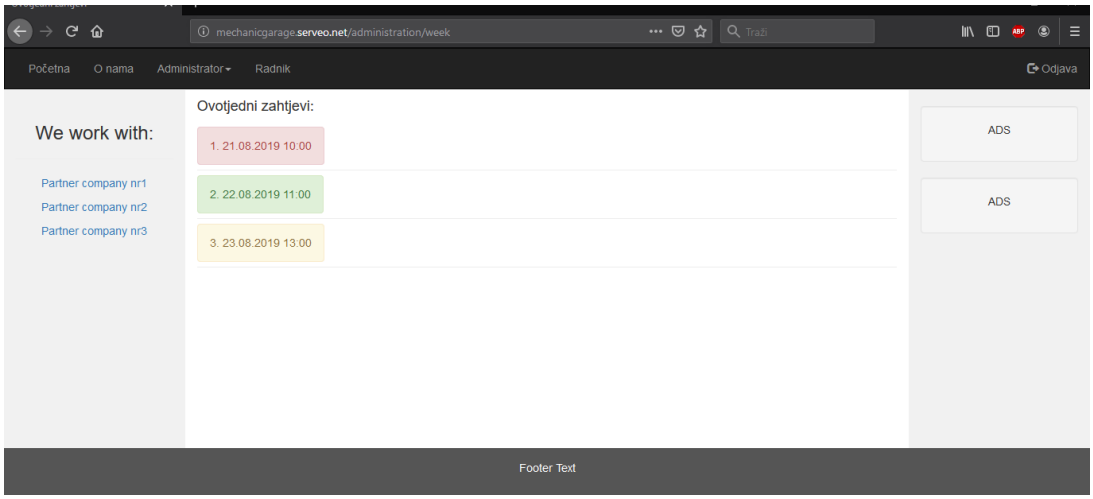


Sl. 4.6. Stranica za prijavu

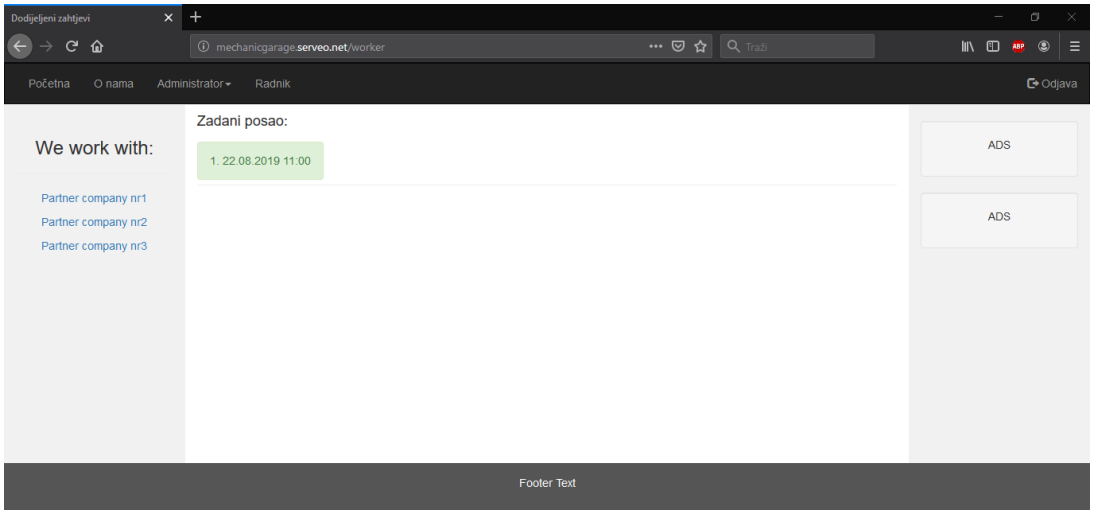


Sl. 4.7. Početni ekran administratora

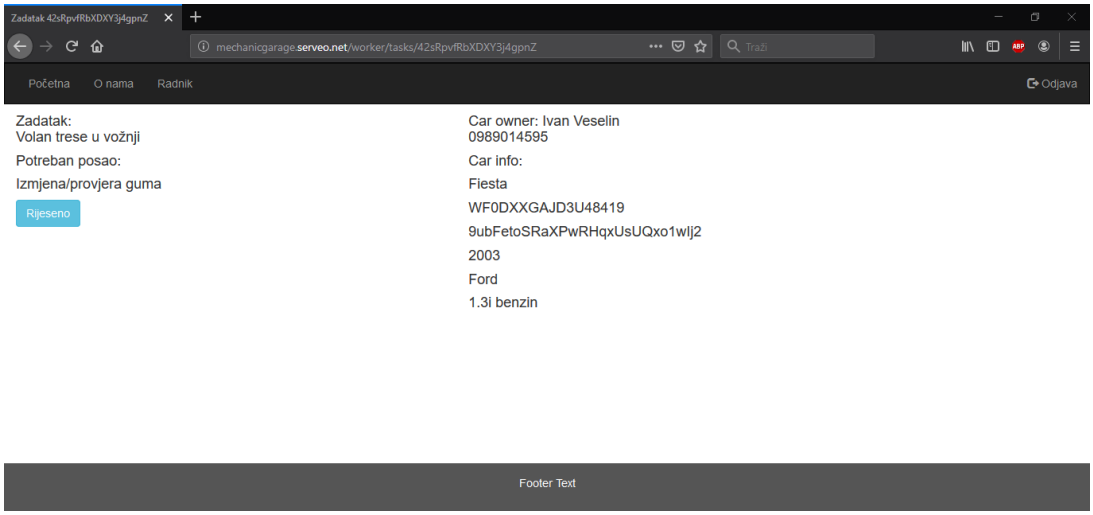
Radnici nakon prijave vide sve zadatke koji su dodijeljeni njima za odrađivanje(Sl. 4.9.). Odabirom pojedinog zadatka otvara se stranica s detaljima kao što su odabrani servisi, podatci o automobilu i dr.(Sl.4.10.). Na kraju imaju ponuđen gumb kojim označavaju zadatak kao odrađen koji im se onda na popisu pojavi u zelenoj boji koja predstavlja odrađeni zadatak.



Sl. 4.8. Stranica s ovotjednim zadatcima



Sl. 4.9. Radnikov popis dodijeljenih zadataka



Sl. 4.10. Detaljni prikaz zadatka

5. ZAKLJUČAK

Sustav za pomoć u radu servisa za vozila sastoji se od dva dijela, mobilne aplikacije i web poslužitelja. Mobilna aplikacija razvijena je za mobilne uređaje s Android sustavom minimalne verzije 5.0(*Lollipop*). Aplikacija web poslužitelja pisana je u *JavaScript* jeziku korištenjem *Express frameworka* za izvođenje u *Node.Js* okruženju.

Android aplikacija namijenjena je krajnjim korisnicima servisa te služi za pregled slobodnih i odabir odgovarajućeg termina. Korisnici aplikacije imaju i mogućnost unosa podataka o svom automobilu koji služe radnicima u servisu za naručivanje dijelova i sl. Termine koji su ponuđeni u aplikaciji odabire administrator/vlasnik servisa vozila korištenjem web poslužitelja. U aplikaciju se korisnici sami registriraju i prijavljuju pomoću svoje email adrese i odabrane lozinke.

Web poslužitelj ima i javno dostupne web stranice kojima mogu pristupiti svi kao što su stranica s osnovnim podatcima ili stranica s kontakt informacijama, ali za pristup web stranicama koje se tiču poslovanja servisa za vozila zahtjeva prijavu. Vlasnik servisa za vozila prilikom prvog postavljana sustava dobije svoj administratorski korisnički račun. Korisnik s administratorskim računom ima pristup pregledu zauzetih termina, može rasporediti prijavljene zahtjeve za određenim poslovima svojim radnicima, ima pregled trenutnog tjedna, može objavljivati raspored slobodnih termina za nadolazeće tjedne te jedini može registrirati nove korisnike u sustav. Korisnici registrirati od strane administratora dobivaju status radnika. Uz sve administratorske ovlasti, također ima i pristup kakav imaju i korisnici sa statusom radnika. Radnici se u sustav prijavljuju s podatcima koje im dodijeli njihov administrator/poslodavac. Nakon prijave imaju pristup pregledu posla koji je zadan njemu i označavanje odrađenih zadataka kao gotove.

Sustav je osmišljen i razvijen kako bi rasteretio vlasnike malih servisa za vozila koji često svoju djelatnost obavljaju sami ili imaju mali broj zaposlenika. Posao ne mogu obavljati bez kontakta s ljudima, što krajnjim korisnicima servisa, što prodajnim predstavnicima, dostavljačima, zaposlenicima i dr. Ta interakcija im oduzima veliki dio radnog vremena, pa im ostaje manje vremena za obavljanje svoje primarne djelatnosti koja im donosi prihode. Ovaj sustav im olakšava dio komunikacije s krajnjim korisnicima na obostrano zadovoljstvo – vlasnici imaju više vremena za obavljanje svoje djelatnosti, a korisnici mogu odabrati termin koji njima najbolje odgovara.

LITERATURA

[1] *Cloud Firestore*, https://firebase.google.com/docs/firestore/ (20.8.2019.)

[2] *Firebase Authentication*, https://firebase.google.com/docs/auth/ (20.8.2019.)

[3] *Navigation Drawer*, https://developer.android.com/guide/navigation/navigation-ui#add_a_navigation_drawer (7.8.2019.)

[4] *Node.Js,* https://nodejs.org/en/ (20.8.2019.)

[5] *Express*, https://expressjs.com/ (11.8.2019.)

[6] *moongose*, https://mongoosejs.com/docs/guide.html (20.8.2019.)

[7] *Firebase Admin SDK,* https://firebase.google.com/docs/admin/setup (20.8.2019.)

[8] *Authentication vs Authorization*, https://medium.com/datadriveninvestor/authentication-vs-authorization-716fea914d55 (11.8.2019.)

SAŽETAK

Sustav za pomoć u radu servisa za vozila namijenjen je manjim, privatno posjedovanim servisima automobila čiji vlasnici žele olakšati postupak dogovaranja termina i delegiranja posla zaposlenicima korištenjem informatičkih tehnologija. Sustav pruža jednostavan i razumljiv prikaz svim grupama korisnika – krajnjim korisnicima servisa, vlasniku radionice i njegovim zaposlenicima. Android aplikacija namijenjena je krajnjim korisnicima i omogućuje jednostavan i brz odabir termina koji je prihvatljiv za obje strane. Aplikacija web poslužitelja primarno je namijenjena vlasniku i radnicima gdje im je na jednom mjestu dostupno sve što im treba za rad na pojedinom zadatku.

Ključne riječi: Android, Firestore, Node.js, servis automobila

ABSTRACT

Title: Vehicle Service Assistance System

Vehicle Service Assistance System is intented for small, privately owned car shops whose owners want to simplifiy the process of making appointments and delegating work to their employees by using information technology. System provides simple and understandable overview to all groups of users – workshop end-users, workshop owner and his employees. Android application is intended to be used by end-users and it enables them to make appointments fast and simple. Web service is primarily intended to be used by the owner and his employees as a place where all needed information for working on a particular task is present.

Key words: Android, car workshop, Firestore, Node.js

ŽIVOTOPIS

Ivan Veselin rođen je 15. siječnja 1995. godine u Požegi. Od rođenja živi u malom mjestu Radovanci nadomak Velike. U Velikoj je završio osnovnu školu Ivana Gorana Kovačića. Nakon toga upisuje Tehničku školu u Požegi čijim završetkom 2013. godine dobiva kvalifikaciju tehničara za računalstvo. Iste godine upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija koji završava u redovnom roku i 2016. stječe titulu sveučilišnog prvostupnika inženjera računarstva. Iste godine na istom fakultetu nastavlja obrazovanje upisom diplomskog studija računarstva – smjer Programsko inženjerstvo. Krajem 2018. godine zapošljava se u vukovarskoj tvrtki *Code Consulting* čiji je zaposlenik i sada.

_____

Ivan Veselin

PRILOG

1. Git repozitorij Android aplikacije
   https://github.com/iveselin/MechanicGarageAndroid
2. Git repozitorij web poslužitelja
   https://github.com/iveselin/MechanicGarageWeb