DIZAJN I IZRADA WEB APLIKACIJE ZA AUTOBUSNE KARTE KORIŠTENJEM RAZVOJNOG OKRUŽENJA DJANGO I ANGULAR

Topić, Andrea

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split / Sveučilište u Splitu**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:228:890987

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Repository of University Department of Professional Studies

zir.nsk.hr

UNIVERSITY OF SPLIT

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska Tehnologija

ANDREA TOPIĆ

Z A V R Š N I R A D

DIZAJN I IZRADA WEB APLIKACIJE ZA AUTOBUSNE KARTE KORIŠTENJEM RAZVOJNOG OKRUŽENJA DJANGO I ANGULAR

Split, rujan 2021.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska Tehnologija

Predmet: Informacijski sustavi

Z A V R Š N I R A D

Kandidat: Andrea Topić

Naslov rada: Dizajn i izrada web aplikacije za autobusne karte korištenjem razvojnog okruženja Django i Angular

Mentor: dr. sc. Igor Nazor, profesor v.š.

Split, rujan 2021.

SADRŽAJ

Sažetak 1
1. Uvod 2
2. Korištenje tehnologija 3
2.1. Postavke i opis okruženja Django REST 3
2.2. Postavke i opis okruženja Angular 12 9
3. Izrada funkcionalnosti autentifikacije i autorizacije 12
3.1. Autentifikacija 12
3.2. Autorizacija 15
4. Mogućnosti korištenja aplikacije 16
4.1. Prikaz za sve korisnike 16
4.2. Prikaz za registrirane korisnike 18
4.3. Prikaz administratorskog sučelja 20
5. Zaključak 24
Literatura 25
Popis slika 26

Sažetak

Ideja za izradom web aplikacije za preuzimanje autobusnih karata nastala je kako bi omogućila korisnicima autobusa brzu i jednostavnu pristupačnost kartama i preplatama, te informacijama vezanih za prometovanje autobusnih linija, a zaposlenicima lakše upravljanje podacima. Aplikacija se sastoji od upravljačke ploče (engl. *admin panel*), korisničke ploče (engl. *user panel*) i glavnog prikaza za sve korisnike. Za implementaciju funkcionalnosti aplikacije (engl. *backend*) korišten je razvojni okvir Python Django REST Framework (tzv. DRF) koji za pohranu podataka koristi ugrađenu vlastitu bazu podataka SQLite, a za dizajn i izgled (engl. *frontend*) korišten je razvojni okvir Angular verzije 12. Za testiranje točaka aplikacijskog sučelja koje su potrebne za razmjenu podataka između dva poslužitelja (engl. *server*) korištena je aplikacija Postman.

Ključne riječi: upravljačka ploča, backend, frontend, Django, Angular

Summary

Bus ticket application design and development using Django and Angular

Idea for developing web application was based on providing fast and simple availability of bus tickets and subscriptions, and information about bus lines for the users of the bus as a public transport, as for the employee's point of view they get to manage data more efficiently. The application consists of three parts: administrators panel, users panel (sc. dashboard) and main view for all users. The application functionality (backend) is implemented using Django REST framework which offers SQLite database for data management, and design was created using Angular framework version 12. Application program interface (API) endpoints that are used for data sharing between two servers were evaluated using Postman application.

Keywords: admin panel, backend, frontend, Django, Angular

1. Uvod

Glavni motiv za izradu aplikacije je modernizacija javnog prijevoza i omogućavanje korisnicima da preuzmu kartu i preplatu u bilo kojem trenutku bez čekanja u redu uz sigurnost i zaštitu osobnih podataka. Svi korisnici mogu pregledavati polaske autobusnih linija i preuzimati karte, a za dodatne mogućnosti potrebno se je registrirati.

Poslužiteljski dio aplikacije razvijen je korištenjem razvojnog okruženja Django REST koji sadrži pakete za jednostavniji razvoj API točaka. Korisnički dio aplikacije je izrađen korištenjem razvojnog okvira Angular 12 kojim se postiže popularan izgled, a s obzirom na to da je aplikacija jednostrana (engl. *Single-Page*) smanjeno je vrijeme učitavanja u odnosu na višestrane web aplikacije. Posluživanje web aplikacije omogućeno je s različitih priključnih točaka lokalnog poslužitelja. Komunikacija među njima je bila glavni problem koji je riješen korištenjem CORS zaglavlja. Problem sigurnosti prijenosa podataka riješen je pomoću JSON web token (JWT) tokena, što će biti u trećem poglavlju detaljnije objašnjeno.

U drugom poglavlju je opisan način primjene razvojnih okruženja Django i Angular za izradu web aplikacije, postavke okruženja i način prijenosa podataka između poslužitelja i korisnika. U trećem poglavlju analizirana je autentifikacija, autorizacija i sigurnost kod prijenosa podataka. Četvrto poglavlje opisuje korisničko iskustvo (engl. u*ser experience* - *UX*), što je način na koji korisnik koristi aplikaciju. Korištenjem aplikacije korisnik pristupa funkcionalnostima koje su mu na raspolaganju ovisno o ulozi koja mu je dodijeljena.

2. Korištenje tehnologija

2.1. Postavke i opis okruženja Django REST

Za izradu web aplikacije za pristupne točke programskog sučelja (engl. a*pplication programming interface access points* – API) koristi se programski jezik Python.

API pristupne točke omogućavaju dijeljenje resursa iz baze podataka sa drugim udaljenim poslužiteljima. U ovom radu pristupne točke služe za primanje zahtjeva i slanje valjanih podataka unutar tijela http odgovora (engl. *http response*). Aplikacija je kreirana unutar virtualnog okruženja instaliranog pomoću Python paketa `pip`, a nakon aktivacije unutar virtualnog okruženja instalira se Django i REST (engl. *Representational State Transfer*).

REST skup alata je dio okruženja Django, a dolazi sa raznim bibliotekama koje se uključuju unutar skripti što omogućava jednostavniju izradu API točaka. Funkcionalnosti REST-a se omogućuju instaliranjem paketa `rest_framework` u backend aplikaciju. Potrebno je pozicionirati se unutar mape u kojemu je virtualno okruženje te izvršiti nekoliko Python naredbi u komandnoj liniji koje će kreirati mapu i skripte potrebne za izradu i pokretanje web aplikacije. Naredbe za izradu nalaze se na slici 2.1.1, a izgled mape nakon izvršavanja istih nalazi se na slici 2.1.2.

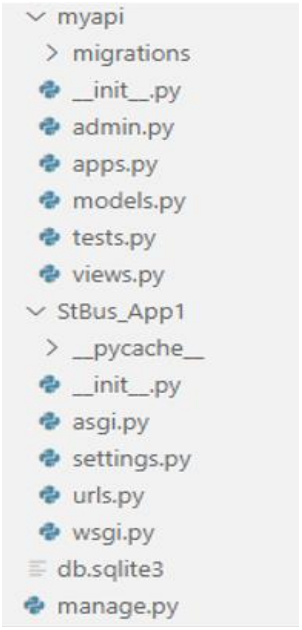
```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\Docs\Projekt> env1/Scripts/activate
(env1) PS D:\Docs\Projekt> django-admin startproject StBus_App
(env1) PS D:\Docs\Projekt> cd StBus_App
(env1) PS D:\Docs\Projekt\StBus_App> python manage.py startapp myapi
(env1) PS D:\Docs\Projekt\StBus_App>
```

Slika 2.1.1: Naredbe za izradu Django projekta



Slika 2.1.2: Kreirani Django projekt

Unutar kreirane mape nalaze se postavke i skripte za razvoj aplikacije i prazna baza podataka SQLite. Baza podataka SQLite puni se podacima primjenom migracija nakon što se u skripti `models.py` definiraju klase. Klase predstavljaju tablice u bazi, a njihove varijable retke u tablici. Python skripta `manage.py` sadrži funkcije za pokretanje aplikacije u pregledniku i migriranje podataka u bazu. Migriranje podataka potrebno je izvršiti nakon definiranja tablica. Instalacijom aplikacije korištenjem okruženja Django nude se već ugrađene tablice za korisnike, grupe i dozvole, pa je bilo potrebno migrirati podatke nakon što su sve tablice zadane, a za potrebe ove aplikacije korišten je ugrađeni model za korisnike.

U postavkama unutar kreiranog projekta potrebno je dodati paket naziva `rest_framework`, koji je prethodno instaliran i ime aplikacije, u ovom primjeru `my_api` te pozvati naredbu za pokretanjem poslužitelja kako bi bilo moguće vidjeti aplikaciju u pregledniku.

Django je zasnovan na MVT (engl. *Model-View-Template*) arhitekturi koja je dizajnirana na način da prati uzorak ili putanju podataka iz baze podataka prema korisniku koji je zatražio podatke i obrnuto. Model služi kao posrednik za komunikaciju sa bazom podataka SQLite i pregledima koji unutar funkcija izvršavaju upite nad bazom.

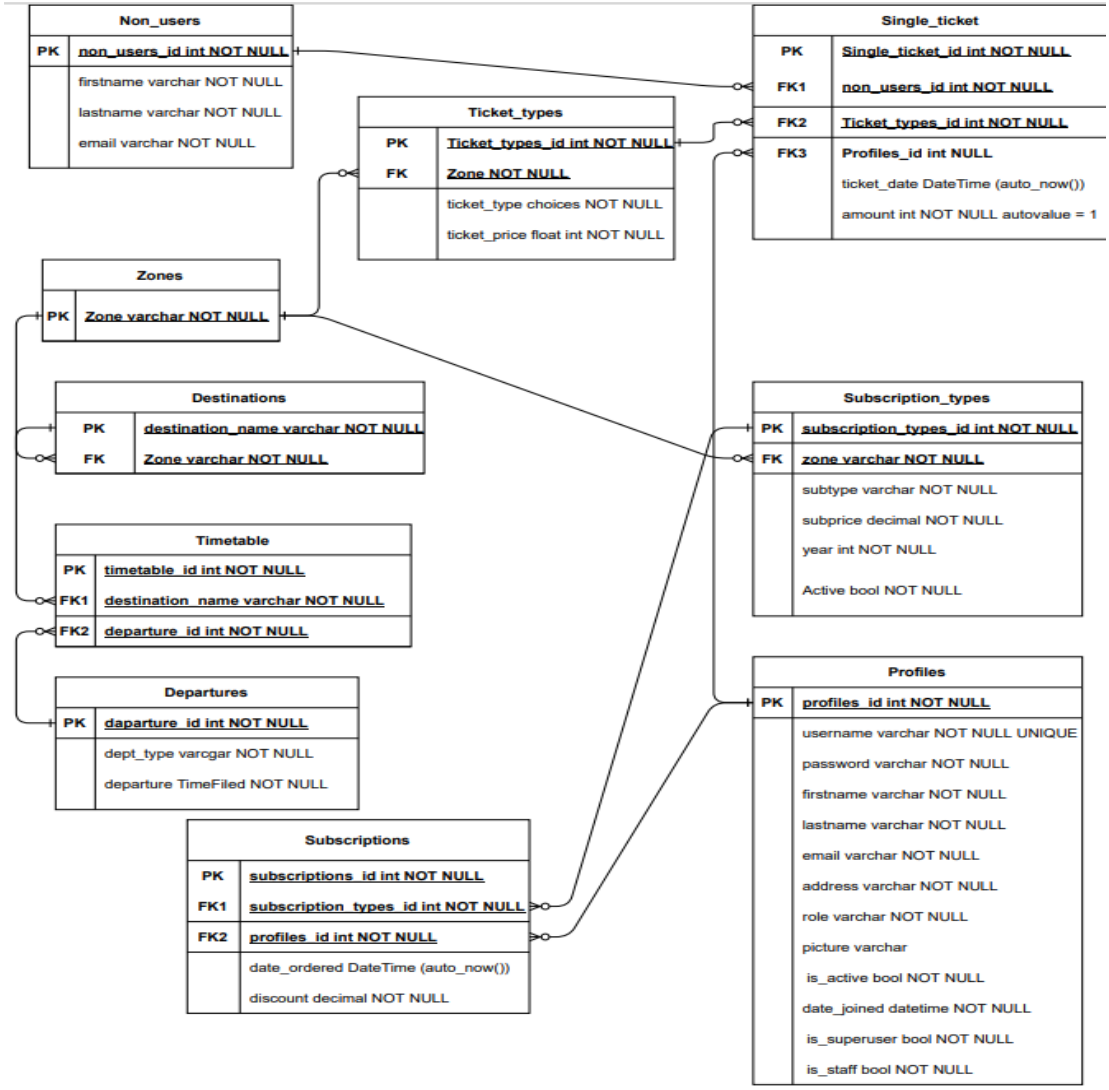
Nakon kreiranja klasa u skripti `models.py`, baza podataka je migrirana i prilagođena potrebama aplikacije.

Za kreiranje baze podataka bilo je potrebno prvo isplanirati zahtjeve aplikacije i odrediti tok podataka. Planiranje baze podataka pomoglo je da se podaci unutar tablica koriste efikasnije i točnije, a tablica modela podataka prikazana je na slici 2.1.3.

Svi redci na slici 2.1.3 pretvoreni su u Django modele, a model *Profiles* je preko postavki zadan kao model nad kojim se vrši autorizacija i autentifikacija. Naredba u postavkama zadana je kao:

```
AUTH_USER_MODEL = 'my_api.Profiles'
```

Modelu *Profiles* dodijeljena su predefinirana polja. Polja koja su potrebna za aplikaciju, a nisu predefinirana od okruženja, definiraju se u modelima. Polja koja su dodana zadavanjem klase u skripti *models.py* nalaze se na slici 2.1.4.



Slika 2.1.3: Model podataka

```python
class Profiles(AbstractUser):
    rola = [
        ('none', 'none'),
        ('student', 'student'),
        ('unemployed','unemployed'),
    ]
    email = models.CharField(max_length=64, null=False, unique=True)
    address = models.CharField(max_length=128, blank=True)
    picture = models.ImageField(upload_to='StBus_app1/media', blank=True)
    role = models.CharField(max_length=10, null=True, choices=rola, blank=True)
    username = models.CharField(max_length=64, null=False, unique=True)
    REQUIRED_FIELDS = ['first_name', 'email', 'password']
    def __str__(self):
        return self.email
```

Slika 2.1.4: Definicija modela *Profiles*

Skripta `urls.py` unutar kreirane `my_api` aplikacije je jedan od važnijih dijelova aplikacije jer se u njoj nalaze putanje koje zadaju poveznicu i vezuju je uz pregled (engl. *View*). Poveznica služi kao pristupna točka, a funkcija iz pregleda vraća podatak ili upisuje podatak u bazu.

Primjer zadavanja pristupnih točki nalazi se na slici 2.1.5. Poveznice su organizirane prema potrebama korisnika na način da je pomoću pregleda definirano koji podaci će biti prikazani i koja metoda je dopuštena. Metoda zahtjeva koja je definirana u servisima okruženja Angular mora odgovarati metodi koja je zadana unutar dekoratora funkcije iz skripte `Views.py`.

```python
from . import views
from django.urls import path, include
from rest_framework import routers
from rest_framework_jwt.views import obtain_jwt_token, refresh_jwt_token, verify_jwt_token

router = routers.DefaultRouter(trailing_slash=False)

urlpatterns = [
    # everyone
    path('destinations/', views.AllDestinations, name='destinations'),
    path('destinations/<int:pk>', views.DestZone, name='destinations_by_zone'), # filter des
    path('timetable/<int:did>', views.Show_Timetable), # gets the timetable of chosen destin
    path('register/', views.RegisterAPI.as_view(), name='register'),
    path('checkuser/<str:username>',views.CheckusernameAPI), # does username exists -for log
    path('checkmail/<str:email>',views.CheckemailAPI), #does email exists - register check
    path('nonusers/', views.nonUser), #for ticket creating
    path('buy_ticket/', views.Buy_Ticket),
    path('ttype/', views.tickettyp), # gets all ticket types or admin creats new ticket type
```

Slika 2.1.5: API krajnje točke

Pregled je skripta koja sadrži funkcije (engl. *Function Based Views*) ili klase (engl. *Class Based Views*) u kojima se pomoću CRUD (engl. *Create Read Update Delete*) metoda izvršavaju operacije nad podacima. CRUD predstavlja kraticu za naredbe kreiraj, pročitaj, ažuriraj i izbriši. Osim što se unutar pregleda pozivaju modeli za dohvaćanje podataka iz baze, podatke je potrebno pretvoriti u odgovarajući tip. U ovoj aplikaciji kreirana je i skripta `serializers.py` unutar koje se nalaze klase koje pretvaraju podatke iz Python modela u `json` i obrnuto.

Pojam serijalizacije predstavlja pretvorbu Python u neki drugi format, a deserijalizacija pretvaranje `json` formata, u kojem se šalju upiti prema poslužitelju, u format kojeg Python može pročitati. Predložak (engl.*template*) skripta je HTML formata koja se povezuje sa nekom od funkcija iz `views.py` skripte na način da se definira putanja u `urls.py` skripti.

U ovom radu kao predložak korišteno je okruženje Angular 12 gdje se preko servisa navodi putanja iz `urls.py` skripte da bi se poslao zahtjev na Django poslužitelj.

Da bi komunikacija između klijenta i poslužitelja funkcionirala, Django poslužitelj mora prepoznati i odobriti primljene zahtjeve klijenta koji je na udaljenom poslužitelju. Nakon instalacije `corsheaders` paketa potrebno je unutar `settings.py` postaviti dopuštenje za primanjem zahtjeva i uključiti CORS zaglavlja (engl. *Cross Origin Resource Sharing headers*). CORS zaglavlja koriste se da bi se ograničio pristup resursa sa udaljenih poslužitelja, te se na taj način sprječavaju napadi i krađe podataka, a predefinirane postavke postavljene su tako da su resursi omogućeni samo za zadani poslužitelj na kojemu se nalazi baza. Kako bi se sa Angular poslužitelja ostvario uspješan pristup podacima potrebno je zadati vrijednost CORS zaglavlja.

CORS zaglavlja koriste se i za autentifikaciju na način da udaljeni poslužitelj šalje u zaglavlju zahtjeva (engl. *request*) podatak u obliku ključ - vrijednost gdje je ključ vrsta autentifikacije koju koristimo, a njena vrijednost u ovoj aplikaciji je JSON Web token. Sve dodatne postavke za projekt `StBus_app` prikazane su na slikama 2.1.6 i 2.1.7.

```
ALLOWED_HOSTS = ['*']
CORS_ORIGIN_WHITELIST = [
    'http://localhost:4200',

    ]
CORS_ALLOW_CREDENTIALS = True
CORS_ORIGIN_ALLOW_ALL = True

CORS_ALLOW_HEADERS = [
    'accept',
    'accept-encoding',
    'authorization',
    'content-type',
    'origin',
    'user-agent',
    'X-CSRFToken',
    'csrftoken',
    'x-csrftoken',
    'Access-Control-Allow-Origin',
]

CSRF_TRUSTED_ORIGINS = [
    'http://localhost:4200/',
]
```

2.1.6: Dodatne postavke za CORS zaglavlja

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'corsheaders.middleware.CorsPostCsrfMiddleware',
    'corsheaders.middleware.CorsMiddleware',
]
```

2.1.7: Dodatne postavke za *CORS middleware*

Pomoćna aplikacija koja je korištena tijekom izrade web aplikacije, a u svrhu ispitivanja i analiziranja točnosti slanja i primanja podataka na API točkama, je *Postman*. Testira se na način da se pošalje zahtjev prema Django poslužitelju sa odgovarajućim metodama i poveznicom pa se analizira odgovor. Korištene funkcionalnosti su: slanje različitih tipova podataka prema poslužitelju, postavke odgovarajućih zaglavlja za poslane zahtjeve, detaljni opis grešaka u komunikaciji što olakšava pronalaženje i uklanjanje grešaka u kodu.

2.2. Postavke i opis okruženja Angular 12

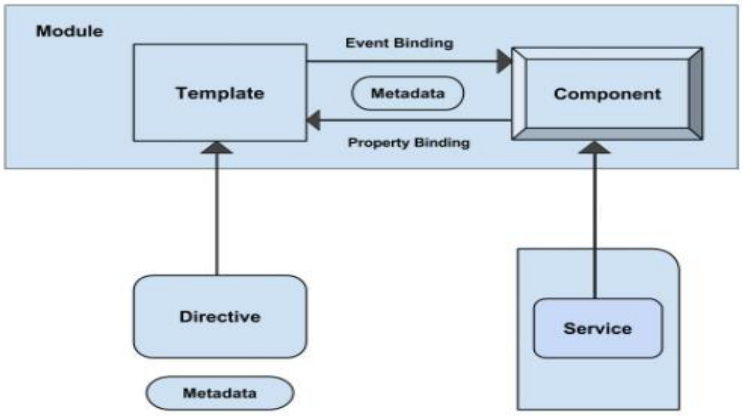
Za dizajn aplikacije za autobusne karte korištena je najnovija aktualna verzija 12 okruženja Angular koja je izašla u proljeće 2021. Prije instalacije Angular 12 potrebno je imati instalirano *npm* i *node.js* na računalu. Nakon izvršavanja naredbe za instalaciju, unutar Django projekta dodana je nova mapa sa potrebnim datotekama i skriptama za razvoj dizajna aplikacije. Dodatne postavke se podešavaju u konfiguracijskoj skripti *angular.json* gdje je potrebno postaviti *json* skriptu koja sadrži poveznicu na Django poslužitelj. Postavke koje su primijenjene prikazane su na slici 2.2.1. Pomoću naredbe:

```
npm run ng serve frontend
```

omogućava se vidljivost aplikacije u pregledniku na priključku 4200.

Slika 2.2.1: Postavke Angular aplikacije

Arhitektura okruženja Angular bazira se na komponentama kao funkcionalnim blokovima podataka, modulima i servisima, a koristi se za dizajn jednostranih web aplikacija. Komponente sadrže skripte u *typescript*, *html* i *css* formatu. Moduli i servisi su *typescript* skripte koje imaju određene funkcionalnosti. Prednost jednostranih web aplikacija je brzina učitavanja i obrade podataka. Komponente, servisi i moduli mogu se kreirati korištenjem Angular CLI terminala ili manualno, a svaka komponenta može koristiti drugu unutar sebe na način da se unese naziv komponente koja se želi koristiti. Slično kao i kod okruženja Django i kod Angulara se prati određeni uzorak toka podataka za prikaz istih. Primjer toka podataka u okruženju Angular prikazan je na slici 2.2.2.



Slika 2.2.2: Arhitektura aplikacije u Angularu

Zahtjev za dohvaćanje podataka definira se unutar metode iz `service.ts` skripte koja šalje upit na udaljeni poslužitelj. Metoda iz komponente nad metodom iz servisa koristi neke od zadanih metoda koje nudi Angular kao što je pretplata (engl. *subscribe*). Da bi vrijednost bila pročitala u prikazu (engl. *template*), podatak iz odgovora unutar pretplate se postavlja u određenu varijablu. Zahtjevi za dohvaćanje, umetanje i brisanje podataka iz baze zahtijevaju postavljanje argumenata kao što su vrijednosti polja forme ili nekih drugih vrijednosti. Potrebno je zadati metode za zahtjeve dohvati, umetni, izbriši (engl. *get, put, delete*) prije slanja zahtjeva na određenu API točku.

Komponente, servisi i moduli glavni su dijelovi za razvoj dizajna u Angular okruženju, a u datoteci `app.modules.ts` nalazi se popis svih korištenih biblioteka, komponenti, servisa i modula potrebnih da bi aplikacija funkcionirala. Svaku skriptu koju je dodana i koja se koristi u nekom dijelu aplikacije potrebno je uključiti u skriptu `app.module.ts`. Pomoću `app-routing` modula omogućen je prikaz komponenti kao rute na jednoj stranici, a definira se putanja tako što se poveznica poveže sa komponentom. Komponente `nav`, `register` i `get-ticket` nalaze se unutar drugih komponenti, a ostale komponente pozivaju se preko definiranih ruta. Za provjeru zahtjeva koje Angular šalje na drugi poslužitelj korišten je presretač (engl. *interceptor*) koji je kreiran da bi presretao zahtjev koji je poslan sa klijentskog dijela aplikacije. Presretač provjerava je li postavljeno odgovarajuće zaglavlje koje će omogućiti da Django poslužitelj odgovori na zahtjev. Kontrola prikaza za logirane korisnike i posjetitelje omogućena je sa `auth-guard.service.ts` skriptom. Osim kontrole prikaza, provjerava trajanje tokena dok je korisnik aktivan i osvježava ga svakih sat vremena.

Pomoću servisa omogućeno je rukovanje zahtjevima koji se šalju na udaljeni poslužitelj, a unutar skripte nalazi se klasa sa metodama koje prosljeđuju ili primaju podatke sa udaljenog poslužitelja uz pomoć `httpClient` servisa. `HttpClient` se deklarira unutar konstruktora. Udaljeni poslužitelj šalje podatke ili grešku ako korisnik nema prava. Kako bi se prilikom slanja zahtjeva zadovoljili uvjeti zaglavlja, unutar svakog servisa gdje je korisniku ograničen pregled, potrebno je poslati kod tokena u zaglavlju. Primjer zaglavlja nalazi se na slici 2.2.3.

```
private  headers = new HttpHeaders({'Content-Type': 'application/json',
                                    'Access-Control-Allow-Origin': '*',
                                    'Authorization':'JWT '+ localStorage.getItem("user")});
```

Slika 2.2.3: Zaglavlje unutar skripte `auth.service.ts`

Skripte čija imena završavaju s `component.html` su predlošci koji dinamički koriste podatke koji se nalaze u skriptama čija imena završavaju s `component.ts`. Varijabla u predlošku se ispisuje unutar dvostrukih vitičastih zagrada: *{{ naziv_varijable }}* . Podaci koji su sadržani unutar niza su većinom rječnici (engl. *dictionary*) koji sadrže parove ključ – vrijednost. Podaci iz rječnika se ispisuju na web stranicu , korištenjem Angular direktive `ngFor`.

Angular okruženje omogućuje razvoj dinamičkih stranica, pa je u ovoj aplikaciji implementirana dinamička asinkrona provjera (engl. *async validation*) upisnih (engl. *input*) polja kreiranjem prilagođenog validatora za provjeru polja kod upisa lozinke. Validator, funkcionalnost koja vrši razne provjere unosa podataka na web stranicu dio je modula `AngularForms`. Validator se okida kod izvršavanja definiranog događaja (engl. *event*). Validator uvijek mora vratiti vrijednost. Primjer validatora za lozinku prikazan je na slici 2.2.4.

```
import { AbstractControl, ValidatorFn } from '@angular/forms';

export default class Validation{
    static  PassMatchValidator(controlName: string, checkControlName: string): ValidatorFn {
        return (controls: AbstractControl) => {
          const control = controls.get(controlName);
          const checkControl = controls.get(checkControlName);

          if (checkControl?.errors && !checkControl.errors.matching) {
            return null;
          }

          if (control?.value !== checkControl?.value) {
            controls.get(checkControlName)?.setErrors({ matching: true });
            return { matching: true };
          } else {
            return null;
          }
        };
      }
}
```

Slika 2.2.4: Validator za provjeru lozinke

3. Izrada funkcionalnosti autentifikacije i autorizacije

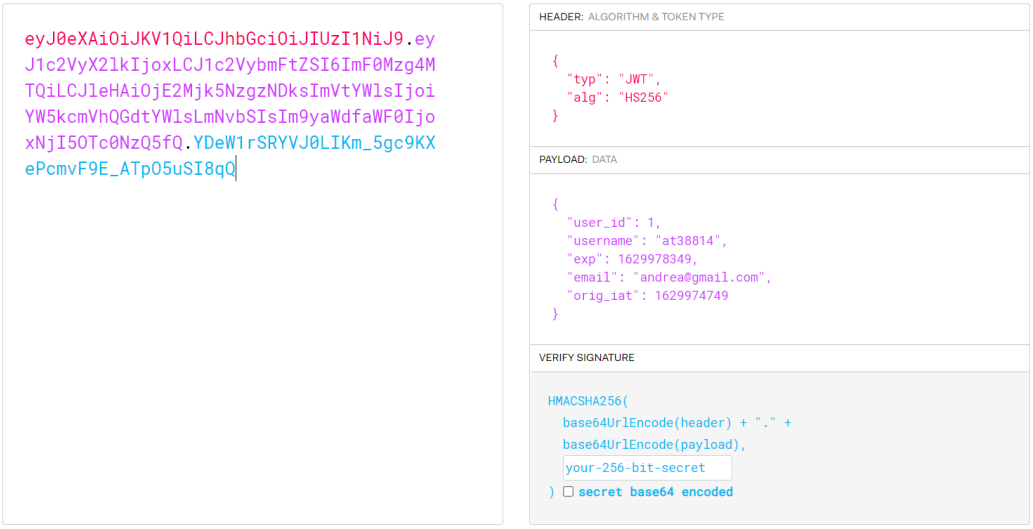
3.1. Autentifikacija

Pojam autentifikacije označava provjeru identiteta korisnika. U aplikaciji za autobusne karte, kod prijave korisnika, potrebno je utvrditi točnost podataka slanjem upita u bazu. Baza odgovara postoji li korisnik sa unesenim korisničkim imenom. U drugom koraku korisnik unosi lozinku, a sustav provjerava njenu ispravnost. Upit sa podacima prosljeđuje se na API točku koja, ukoliko je lozinka odgovarajuća, šalje JWT (engl. *JSON Web Token*) token u tijelu odgovora kojeg frontend sprema u lokalno spremište (engl. *local storage*). JWT token je standardni zapis korisničkih podataka u kriptiranom obliku, podijeljen na tri dijela točkama. JWT token je standardizirana metoda autentifikacije između klijenta i poslužitelja koja sadrži kodirane podatke iz baze. Kako bi poslužitelj verificirao korisnika, klijent mu vraća primljeni JWT token. Ako od poslužitelja stigne potvrda da je korisnik autentificiran, prosljeđuje se na svoju kontrolnu ploču. Oba razvojna okvira podržavaju JWT token autentifikaciju uz određene postavke. Klijent prilikom svakog spajanja na API točku šalje u zaglavlju JWT token.

Django REST framework ima alate za korištenje JWT tokena koje se pozivaju u postavkama. Kako bi poslužitelj koristio JWT token potrebno je dodati JWT paket koji generira JWT tokene svakih sat vremena. Programski kôd kojeg je potrebno umetnuti prikazan je u ispisu 3.1. Većina grešaka koje su nastajale prilikom izrade autentifikacije su zbog pogrešno zadanih postavki. Pomoću zadnje naredbe u ispisu 3.1 omogućeno je korištenje JWT tokena. Prvi redak daje mogućnost da se proizvoljno deklarira vrijednost unutar zaglavlja koje se kod odgovora klijenta čita iz vrijednosti polja `authentication` CORS zaglavlja. Postavlja se početna vrijednost pomoću koje poslužitelj prepoznaje da se radi o JWT tokenu, kodiranom vrijednošću koju dekodira uz pomoć daljnje prikazanih postavki.

Postoji više algoritama za kriptirati podatke (engl. *hashed data*) koji pretvaraju podatke iz baze pomoću sigurnosnog ključa (engl. *security key*) i zadanog algoritma u JWT token, a korišteni je HS256.

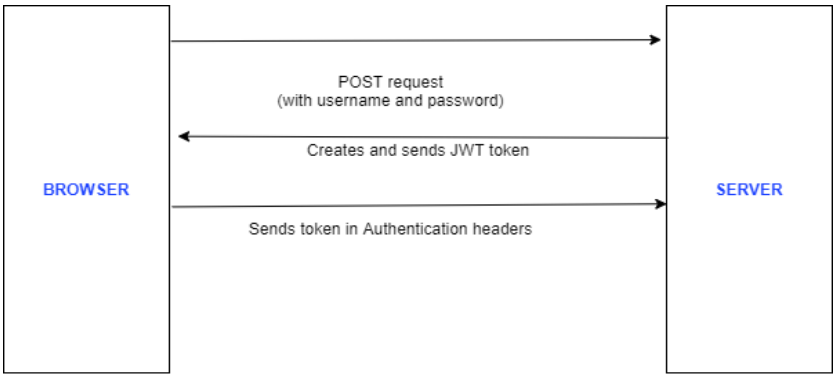
JWT token podijeljen je točkama u dijelove koji predstavljaju kriptirane informacije o korisniku iz baze. Prvo dio je zaglavlje koje sadrži informacije o algoritmu za kriptiranje podataka, drugi dio je sadržaj koji sadrži podatke o korisniku, a treći sadrži verificirani potpis. Primjer kriptiranog i dekriptiranog tokena nalazi se na slici 3.1.1, a dekripcija tokena moguća je na službenoj stranici [7].



Slika 3.1.1: Prikaz JWT tokena u kriptiranom i dekriptiranom obliku

Nakon dekripcije tokena, unutar tijela zadani su podaci o korisniku iz baze podataka koji se prilikom slanja upita sa klijenta na poslužitelja mogu dekodirati. JWT token je popularan i široko korišten za sigurni prijenos podataka od klijenta do poslužitelja kod kojeg nije potrebno korištenje sesija. API pristupne točke za autorizaciju, verifikaciju i osvježenje JWT tokena realizirane su na način da se uvezu (engl. *import*) funkcije iz `rest_framework_jwt` paketa i dodaju u putanje i time je omogućeno spajanje na iste.

Komunikacija između klijenta i poslužitelja za razmjenu JWT tokena prikazana je na slici 3.1.2.



Slika 3.1.2: Razmjena JWT tokena

```
JWT_AUTH = {
     'JWT_AUTH_HEADER_PREFIX': 'JWT',
     'JWT_ENCODE_HANDLER':'rest_framework_jwt.utils.jwt_encode_handler',
     'JWT_DECODE_HANDLER': 'rest_framework_jwt.utils.jwt_decode_handler',
     'JWT_PAYLOAD_HANDLER': 'rest_framework_jwt.utils.jwt_payload_handler',
     'JWT_PAYLOAD_GET_USER_ID_HANDLER':
'rest_framework_jwt.utils.jwt_get_user_id_from_payload_handler',
     'JWT_EXPIRATION_DELTA': datetime.timedelta(hours=1),
     'JWT_RESPONSE_PAYLOAD_HANDLER':
'rest_framework_jwt.utils.jwt_response_payload_handler',
     'JWT_SECRET_KEY': SECRET_KEY,
     'JWT_GET_USER_SECRET_KEY': None,
     'JWT_PUBLIC_KEY': None,
     'JWT_PRIVATE_KEY': None,
     'JWT_ALGORITHM': 'HS256',
     'JWT_VERIFY': True,
     'JWT_VERIFY_EXPIRATION': True,
     'JWT_AUDIENCE': None,
     'JWT_ISSUER': None,
    'JWT_ALLOW_REFRESH': True,
    'JWT_REFRESH_EXPIRATION_DELTA': datetime.timedelta(hours=8),
    'JWT_AUTH_COOKIE': None,
}
    REST_USE_JWT = True
```

Ispis 3.1: Postavke u skripti *settings.py* za korištenje JWT tokena

Angular okruženje nudi modul *jwt-decode* koji je korišten da bi se dekodirao JWT token u svrhu korištenja vremena isteka tokena koji se nalazi u tijelu tokena kao vrijednost polja *exp*. Da bi korisnik mogao neometano koristiti aplikaciju duže od zadanog ograničenja trajanja tokena, u skripti *auth.service.ts* nalazi se funkcija koja pošalje poslužitelju zahtjev za osvježenjem JWT tokena (engl. *refresh JWT token*). Zbog polja u zahtjevu za osvježenjem tokena koja poslužitelj traži od klijenta, na klijentskom poslužitelju je bilo potrebno dekodirati token i poslati validne podatke. Izlaz korisnika iz aplikacije briše lokalno skladište kao i token koji se nalazi u njemu.

3.2. Autorizacija

Autorizacijom su određene ovlasti i ograničene mogućnosti određenih korisnika, a dodjeljivanjem prava korisnicima određuje se tko može koristiti određene resurse iz baze podataka. U bazi aplikacije podaci su ograničeni za sve korisnike, a dio za registrirane korisnike, dok zaposlenici imaju kontrolu nad podacima. Django REST autorizacija može biti korištena na više načina. Funkcijama koje imaju ograničen pristup je potrebno odrediti prava pristupa nad podacima postavljanjem dekoratora iznad iste, za klase se unutar dodaju uvezena prava iz *rest_framework.permissions* biblioteke ili se po potrebi naprave klase u skripti *permissions.py* koje se dodaju u dekoratore, a u aplikaciji su realizirana sva tri načina. Prije korištenja biblioteka za autorizaciju potrebno ih je definirati u postavkama kao i autentifikaciju. Na slici 3.2.1. prikazan je kôd za omogućavanje autorizacije i autentifikacije.

Unutar okruženja Angular autorizacija je korištena da bi se ograničio prikaz za pojedine korisnike koji nemaju određena prava. Pristup stranicama zaposlenika i registriranog korisnika nije moguć ukoliko korisnik nije autentificiran. Omogućavanje prikaza određenog elementa ovisi o ulozi korisnika kao što je element na vrhu korisničke ploče (prikazuje se ukoliko je korisnik zaposlenik).

```
REST_FRAMEWORK = {
    # Use Django's standard `django.contrib.auth` permissions,
    # or allow read-only access for unauthenticated users.
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_jwt.authentication.JSONWebTokenAuthentication',
        'rest_framework.authentication.TokenAuthentication',
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.SessionAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticatedOrReadOnly',
        'rest_framework.permissions.AllowAny',
        'rest_framework.permissions.IsAuthenticated',
        'rest_framework.permissions.IsAdminUser',
    ],
}
```

Slika 3.2.1: Postavke za autentifikaciju i autorizaciju

4. Mogućnosti korištenja aplikacije

Izrada aplikacije prema korisničkim zahtjevima bazira se na podjeli mogućnosti korištenja aplikacije od strane različitih tipova korisnika. Potrebno je bilo omogućiti svim korisnicima brz pristup autobusnim kartama kako bi se smanjio red čekanja i pristupačnost informacijama voznog reda pojedinih autobusnih linija. Odabrani tip karte preuzima se u pdf formatu ili se otvara u pregledniku. Korisnicima koji su se registrirali omogućene su dodatne funkcionalnosti kao korištenje preplata na odabrani period, pregled i kreiranje istih. Prikaz navigacije se mijenja prilikom ulaska na profil. Unutar korisničkog sučelja korisnik može vidjeti prethodno kreirane karte i preplate, kreirati nove te mijenjati svoje podatke. Na ovaj način je omogućeno svakom korisniku javnog prijevoza da na svom uređaju ima potvrdu odabrane karte ili pretplate te ju je moguće priložiti zaposleniku na pregled u bilo kojem trenutku.

Aplikacija zaposlenicima pruža najviše mogućnosti jer je potrebno imati sustav kojim je jednostavno upravljati i održavati podatke ažurnima. Brisanje podataka iz baze podataka bi uzrokovalo probleme u vođenju evidencije prethodnih događaja i načina korištenja podataka, s toga je brisanje implementirano dodatnim poljem kojeg je zaposlenik u mogućnosti isključiti ako ne želi koristiti odabrani podatak.

Kako bi se prikazalo korištenje metode brisanja podataka iz baze omogućeno je brisanje vremena za kreiranje rasporeda iz razloga što su vrijednosti zamjenjive, a promjena ne utječe na povijest korištenja podataka.

4.1. Prikaz za sve korisnike

Naslovna stranica osim navigacije uključuje informacije o aplikaciji koje su skrivene dok se ne klikne na dio o kojem se želi znati više, a sadrži i informacije o tipovima karata. Nalaze se i dvije forme koje mogu popunjavati svi korisnici, a podaci se šalju na poslužitelj pomoću `post` metode. Ovaj dizajn omogućuje da se pažnja korisnika skrene na forme tako da se u fokus dovodi jedna od glavnih funkcija aplikacije, a to je izrada i preuzimanje autobusnih karata. Forme su komponente koje su pozvane unutar predloška `home` komponente.

Svim korisnicima je omogućeno preuzimanje autobusnih karata nakon što pošalju podatke na API pristupnu točku. Preuzimanje karate realizirano je sa dvije forme tako da korisnik prvo upisuje podatke o sebi kako bi popunio tablicu u bazi, a ako podaci već postoje poslužitelj vraća u odgovoru iste podatke čiji se identifikacijski broj prosljeđuje u skriveno polje unosa (engl. *hidden input*) sljedeće forme.

Forma za odabir tipa karte, količinu i vrijeme početka korištenja karte prosljeđuje se u bazu podataka, a korisniku je omogućeno klikom na gumb preuzeti odabranu kartu sa unesenim podacima. Klikom na gumb za preuzimanje okida se događaj `onclick` koji tablicu prebacuje u pdf format i nudi mogućnost preuzimanja ili pregleda u pregledniku.

Na stranici za pregled rasporeda autobusnih linija svima je omogućeno da filtriraju podatke prema zoni tj. području na kojem se kreću određene linije. Unutar prikazane tablice destinacija, klikom na ikonu kod pojedine destinacije, prikazuje se tablica rasporeda odabrane destinacije. Prikaz rasporeda realiziran je tako da se proslijedi identifikacijski broj destinacije u zahtjevu za dohvaćanje koji se šalje poslužitelju.

Za provjeru postojanja korisničkog imena ili emaila korišteni su događaji u Angular okruženju čije funkcije nakon unosa podataka u polje za unos šalju upit prema poslužitelju sa unesenim vrijednostima. Dobiveni odgovor je tipa `boolean` i ukoliko je negativan prikazuju se ogovarajuće poruke na web stranici. Unutar registracijske forme potrebno je popuniti sva polja ispravno kako bi bilo moguće nastaviti, a pri uspješnoj registraciji podaci su proslijeđeni u bazu i prikazuje se poruka. Za kreiranje formi i uspješno prikupljanje unesenih podataka koristi se modul `FormGroup` i `FormBuilder` koji je uvezen unutar skripte od komponente. Formu je potrebno definirati u konstruktoru da bi se podaci bili ispravno pročitani i formatirani. Primjer kôda kojim je definirana registracijska forma pomoću okruženja Angular nalazi se na slici 4.1.1.

```
constructor(public formBuilder: FormBuilder,
            private AuthService: AuthService) {
              this.registerForm = this.formBuilder.group({
                username: ['', Validators.required ],
                first_name: ['', Validators.required],
                last_name: ['', Validators.required ],
                email: ['',  [Validators.required, Validators.pattern(this.emailPattern)] ],
                password: ['', [Validators.required,  Validators.pattern(this.passwordPattern)]],
                password2:['', [ Validators.required]],
                role:[''],
              },  {
                validators:[ Validation.PassMatchValidator('password', 'password2')]
              });

              }
```

Slika 4.1.1: Deklariranje forme za registraciju u *register.component.py* skripti

4.2. Prikaz za registrirane korisnike

Korisniku su nakon registracije dodijeljena prava na korištenje korisničke ploče u kojoj osim što ima mogućnosti da kreira i preuzima karte, može na isti način koristiti preplate i uređivati vlastite podatke. Ulaskom na svoj profil, ukoliko je korisnik zaposlenik, prikazuje se poseban gumb koji vodi na upravljačko sučelje. Korisničko sučelje realizirano je kao komponenta koja je zadana rutom, a izgled za korisnika koji je zaposlenik je prikazan na slici 4.2.1.

Slika 4.2.1: Korisničko sučelje

Za slanje *post* zahtjeva kreirane su forme u *user-panel.components.ts* skripti koje uz pomoć servisa prosljeđuju podatke u bazu podataka. Korištene su tri forme: za mijenjanje informacija o korisniku, za dodavanje karte i za dodavanje nove pretplate. Podaci kao što su informacije o tipu karata i pretplati, te podaci o korisniku, preuzimaju se prilikom učitavanja komponente *user-panel* pozivanjem metoda koje koristeći servise dobivaju podatke za prikaz, a pozivaju se unutar metode *ngOnInit()*.

Servisi koriste *Observable* promatrače, blokove *rxjs* biblioteke koji se koriste isključivo u Angularu kao niz. Da bi podaci bili raspoređeni potrebno je primijeniti *.subscribe()* nad metodom iz servisa, a ta metoda se deklarira kao *Observable*. Pomoću promatrača podaci se raspoređuju u klasu *User* ili *Ticket*. Potrebno je kreirati klasu unutar *typescript* skripte u kojoj su deklarirane varijable i koja služi da bi se podaci pravilno rasporedili i pročitali nakon što su primljeni sa poslužitelja.

Metodom *get* sa servisa se dohvaćaju podaci preko komponente i promatrača iz servisa koje se prikazuju unutar predloška. Podaci unutar predloška se čitaju uz pomoć *ngFor* direktive.

Klasa je deklarirana slično kao i model u *my_api* dijelu aplikacije, primjer klase koja služi za prikaz korisnika i tipova karte prikazano je na slikama 4.2.2 i 4.2.3.

```
export class User {
    id!: number;
    email?: string;
    username?: string;
    password?: string;
    first_name?: string;
    last_name?: string;
    address?: string;
    role?: string;
    token!: string;
    picture?: string;
}
```

Slika 4.2.2. Klasa User

```
export class Ticket {
    id!: number;
    ticket_date!: Date;
    validfrom!: Date;
    amount!: number;
    non_users_id!: number;
    Profiles_id!: number;
    ticket_types_id!: number
}
```

Slika 4.2.3. Klasa Ticket

Podaci o trenutnom korisniku unutar korisničkog sučelja se prikazuju pozivanjem metode unutar *ngOnInit* koja je prikazana u primjeru kôda na slici 4.2.4. U primjeru je prikazana metoda *getmyData* koja na poziv promatrača *observable* iz servisa dobiva podatke, raspakirava i prosljeđuje varijable iz odgovora na sljedeći zahtjev. Sljedeći zahtjev dohvaća karte i pretplate korisnika te ih postavlja u deklariranu varijablu. Da bi podaci bili primljeni potrebno je pretplatiti se na metodu iz uvezenog servisa *dashboard.service.ts* koja šalje zahtjev „dohvati“ prema Django poslužitelju.

Za prikaz svih podataka iz baze, potrebnih za pregled korisničkog sučelja, prosljeđuje se identifikacijski ključ korisnika iz odgovora u upit poslužitelju koji filtrira tablicu svih autobusnih karata i vraća u odgovoru samo one od zadanog korisnika.

Uređivanje korisničkih podataka postignuto je slanjem nove postavljene vrijednosti `put` metodom unutar forme, kojoj se prosljeđuje i `id` korisnika. Korisnik odabire ulogu (engl. *role*) kako bi zaposlenik mogao postavljati određeni popust na iznose karata i preplata.

Forma za uređivanje podataka ujedno služi i kao trenutni prikaz podataka, a trenutna vrijednost je prikazana koristeći `ngModel` direktive polja za unos.

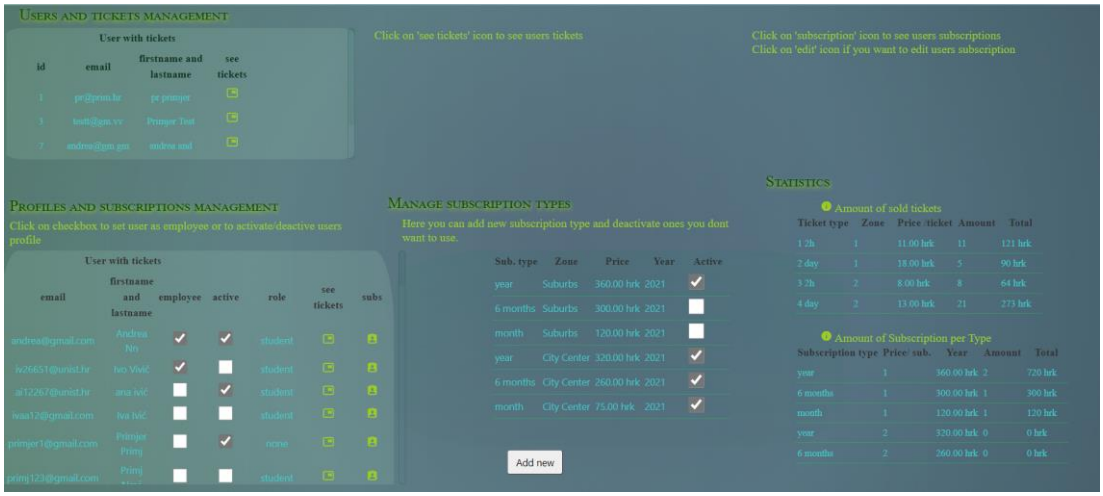
```
getmyData(){
  this.auth.myprofiledata().subscribe(
    response =>{
      this.user=response;
      this.dashboard.getSubs(this.user[0]['id']).subscribe(
        res=>{this.mySubs = res;}
      );
      this.dashboard.myTickets(this.user[0]['id']).subscribe(
        res=>{this.myTicks = res;}
      );
    }
  );
}
```

Slika 4.2.4: Funkcija za dohvaćanje podataka za prikaz na korisničkom sučelju

4.3. Prikaz administratorskog sučelja

Zaposlenici uz sve mogućnosti korisnika imaju i dodatne. Korisnici koji imaju aktivirana prava zaposlenika imaju u navigaciji dodatnu putanju na prikaz i upravljanje korisnicima, kartama i pretplatama. Gumb koji je prikazan na njihovom profilu vodi na upravljačko sučelje za upravljanje destinacijama, tipovima karata, rasporedima i vremenima polazaka.

Kod upravljanja korisnicima dizajn je logički organiziran u blokove podataka koji su raspoređeni u redovima. Tablice sa popisom korisnika nalaze se lijevo, a statistika preuzetih i korištenih preplata i karata u donjem desnom kutu. Prikaz za upravljanje korisnika nalazi se na slici 4.3.1.



Slika 4.3.1: Upravljanje korisnicima

Kod prikaza statistike okruženje Angular omogućava dodavanje matematičkih i logičkih izraza unutar vitičastih zagrada u predlošcima. Podaci u tablici se računaju dinamički, preuzimanjem karte ili pretplate podaci se mijenjaju s obzirom na dodanu vrijednost. Polje `total` služi kao niz koji se puni podacima iz odgovora. Sa dijelom računanja statistike rukuje poslužitelj koji računa količinu karata koja se nalazi u bazi, dok iznose za preuzete karte računaju metode u Angular okruženju. Primjer kôda prikazan je na slici 4.3.2.

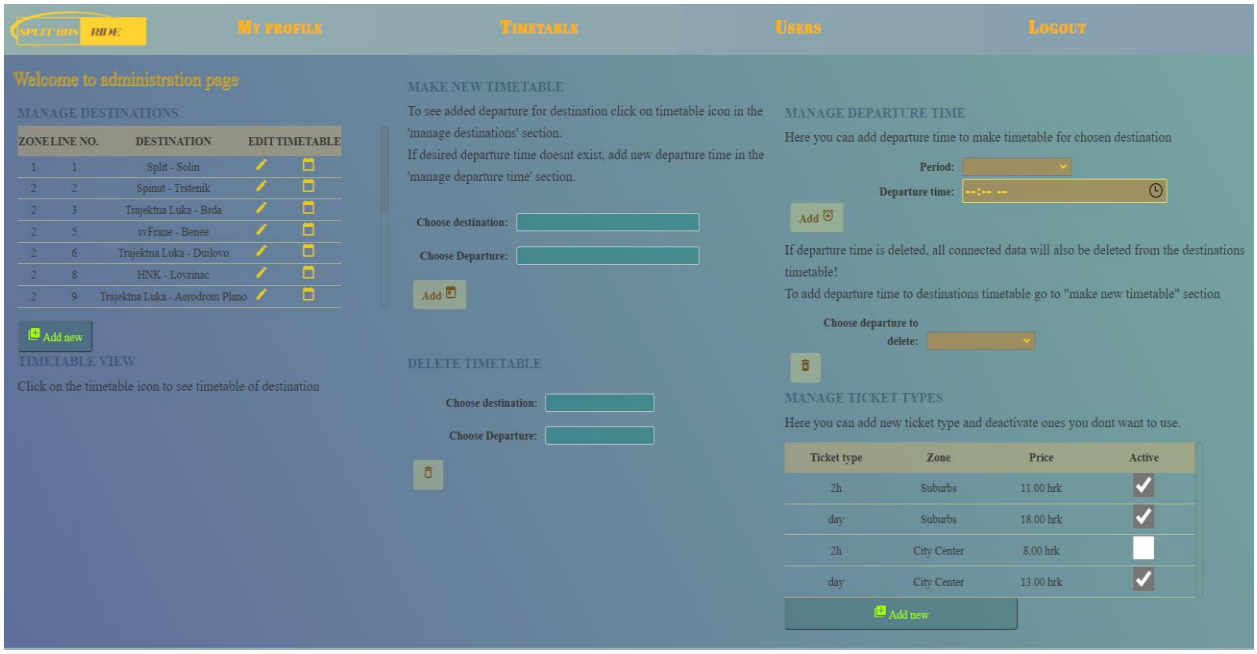
```
<tbody *ngFor='let type of ticket_types'>
  <tr style="border-bottom: 1px solid rgba(1, 250, 84, 0.3);">
      <td>{{ type.id }} {{ type.tickettype }}</td>
      <td> {{ type.zone }}</td>
      <td>{{ type.ticket_price }} hrk</td>
      <td *ngIf='type.id'> {{ total[type.id -1] }} </td>
      <td *ngIf='type.id'> {{ type.ticket_price * total[type.id -1] }} hrk</td>
  </tr>
</tbody>
```

Slika 4.3.2: Dio tablice za statistiku sa dinamičkim podacima

Klikom na ikone unutar tablica korisnika omogućuje se prikaz dodatnih informacija o njihovim kartama ili pretplatama. Ikona unutar tablice sa pretplatama otvara modalni prozor i omogućuje zaposleniku da unese popust za korisnikovu odabranu kartu ili preplatu. U kartici `Users` omogućeno je dodavanje novih tipova karte i deaktiviranje prethodnih. Dodavanje novog tipa karte vidljivo je odmah nakon prosljeđivanja podataka iz forme modalnog prozora na poslužitelj. Brisanje je u ovom slučaju implementirano deaktiviranjem određenog tipa kojeg korisnik neće moći vidjeti niti odabrati jer bi brisanje utjecalo na tablice za statistiku.

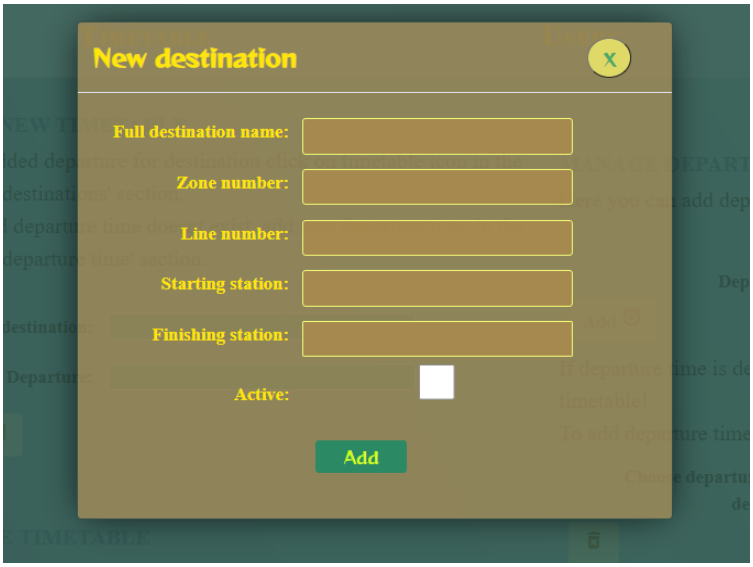
Za postavljanje drugih korisnika kao zaposlenika potrebno je označiti *chekbox* kvadrat, a ako korisnik želi deaktivirati svoj račun, zaposlenik klikom u redu odabranog korisnika u polju *active* deaktivira kvadrat i time se onemogućava korisniku korištenje profila. Implementacija je omogućena pomoću događaja *onclick* koji šalje vrijednost kvadrata *checkbox* na poslužitelj.

Upravljačko sučelje za upravljanje destinacijama i rasporedima je brz način da se podaci mijenjaju i budu odmah vidljivi krajnjim korisnicima. Prikaz sučelja na *admin* putanji logički je podijeljeno po stupcima tako da prvi stupac predstavlja upravljanje destinacijama, drugi stupac raspored a treći vrijeme polaska linija koje je potrebno za formiranje rasporeda. Primjer upravljačkog sučelja prikazan je na slici 4.3.3.



Slika 4.3.3: Upravljanje destinacijama i rasporedom

Kreiranje rasporeda ovisi o destinaciji i vremenu polaska, što znači da podaci o destinaciji i polasku moraju već bit prisutni u bazi, a ukoliko nisu, tada zaposlenik može dodati informaciju koja mu nedostaje. Gumbi koji otvaraju modalne prozore na ovoj stranici se razlikuju po boji od ostalih koji služe za slanje podataka iz forme na poslužitelj. Modalni prozori u ovom prikazu služe za dodavanje nove, uređivanje postojeće destinacije i za dodavanje novog tipa karte, a pozivaju se na event klikom na gumb *Add New*. Prikaz modalnog prozora za dodavanje destinacije nalazi se na slici 4.3.4.



Slika 4.3.4: Kreiranje destinacije

Autorizacija je implementirana i koristeći okruženje Angular ograničavajući prikaz određenih dijelova aplikacije uz pomoć *auth-guard.service.ts* skripte. *Auth-Guard* skripta, kao što joj samo ime govori, čuva podatke od korisnika koji nemaju profil ili im je token istekao pa ga je potrebno osvježiti. Unutar skripte kreira se metoda koja se dodaje u *app-routing.module* skriptu kod putanje kojoj je ograničen pregled.

5. Zaključak

Aplikacija je isplanirana na način da korisniku olakša korištenje usluga javnog prijevoza. Danas u svijetu gdje se živi ubrzanim životnim stilom potrebno je što prije doći do informacija, a s obzirom na postojeću autobusnu infrastrukturu, korisniku javnog prijevoza je otežan pristup podacima i informacijama vezanih za autobusne linije i karte. Aplikacija kao rješenje za javni prijevoz bi smanjila redove čekanja za pretplate i karte, te time omogućila ugodno korištenje usluga javnog prijevoza. Dizajn aplikacije teži ka jednostavnošću korištenja, pristupačnošću i brzinom pružanja usluge. Također dizajn nije monoton nego je za svaku putanju različit prikaz iako je stranica jednostrana. Ciljevi dizajna su minimalnost zbog efikasnijeg i bržeg korištenja aplikacije.

Razvojem aplikacije u okruženju Django REST pomoglo je da se API pristupne točke izrade efikasno zbog raznih biblioteka koje nudi, a razvojem dizajna u Angularu postignuto je učitavanje stranica i njihovih podataka bez čekanja.

Upravo količina dostupnih alata za oba okruženja može napraviti problem u odabiru odgovarajućih, jer svaki alat zahtjeva drugačije postupanje te je potrebno razmisliti koji su alati i dodaci potrebni za razvoj aplikacije. Kod okruženja Django problem je nastao kod sigurnosti jer nudi tri vrste autentifikacije za uvoz: pomoću tokena, sesijsku i klasičnu. Svaka vrsta autentifikacije ima svojih prednosti i nedostataka, a kao rješenje odabrana je autentifikacija pomoću JWT tokena iz razloga što se široko koristi. Sigurnost podataka unutar aplikacije je bitna, pa s obzirom da se i dalje vode rasprave koja je vrsta autentifikacije sigurnija, odlučeno je prema raširenosti korištenja u web aplikacijama.

Mogućnosti koje nudi Angular su velike: od brzog učitavanja, upravljanja podacima primljenih sa poslužitelja, do kreiranja komponenti, servisa ili metoda koje se mogu dijeliti među drugim komponentama. Okruženje Angular ima veliku popularnost, brz razvoj novih poboljšanih verzija, i široku primjenu, što je bio jedan od glavnih razloga odabira okruženja za dizajn. Strukturno oba okruženja za izradu web aplikacija koriste uzorak što ih čini lakšima za savladavanje. Uzorak određuje tok podataka koji je potreban da bi podatak bio proslijeđen od baze do krajnjeg korisnika. S druge strane to može stvoriti problem u početku pogotovo ako je prisutna velika količina kôda a prate se dva uzorka. S obzirom na korisničke zahtjeve korištena okruženja zadovoljila su potrebe aplikacije, a ideja tehnologija je korištenjem moćnih i popularnih alata za web razvoj stvoriti korisnu aplikaciju.

Literatura

[1] Real Python: Django REST Framework Introduction, Django Rest Framework – An Introduction – Real Python (posjećeno 22.8.2021.)

[2] Django REST framework dokumentacija, Home - Django REST framework (django-rest-framework.org) (posjećeno 24.8.2021.)

[3] Techiediaries Team: CORS in Django, CORS in Django REST Framework | Techiediaries (posjećeno 25.8.2021.)

[4] Choradia, S.C.: Django Rest API with JWT Tokens, Django Rest API with JSON web token(JWT) authentication. | by Sambhav Choradia | Analytics Vidhya | Medium (posjećeno 25.8.2021.)

[6] Angular 12 dokumentacija, Home – Angular (angular.io) (posjećeno 25.8.2021.)

[5] Techiediaries Team: Angular 12 REST CRUD APIs & HTTP GET Requests with HttpClient | Techdiaries (posjećeno 26.8.2021.)

[7] JWT token dokumentacija, JSON Web Tokens (jwt.io) (posjećeno 26.8.2021.)

Popis slika

Slika 2.1.1. Naredbe za izradu Django projekta

Slika 2.1.2. kreirani Django projekt

Slika 2.1.3. Model podataka

Slika 2.1.4. Definicija modela *Profiles*

Slika 2.1.5. API krajnje točke

Slika 2.1.6. Dodatne postavke za CORS zaglavlja

Slika 2.1.7. Dodatne postavke za *CORS middleware*

Slika 2.2.1. Postavke Angular aplikacije

Slika 2.2.2. Arhitektura aplikacije u Angularu

Slika 2.2.3. Zaglavlje unutar skripte `auth.service.ts`

Slika 2.2.4. Validator za provjeru lozinke

Slika 3.1.1. Prikaz JWT tokena u kriptiranom i dekriptiranom obliku

Slika 3.1.2. Razmjena JWT tokena

Slika 3.2.1. Postavke za autentifikaciju i autorizaciju

Slika 4.1.1. Deklariranje forme za registraciju u `register.component.py` skripti

Slika 4.2.1. Korisničko sučelje

Slika 4.2.2. Klasa *User*

Slika 4.2.3. Klasa *Ticket*

Slika 4.2.4. Funkcija za dohvaćanje podataka za prikaz na korisničkom sučelju

Slika 4.3.1. Upravljanje korisnicima

Slika 4.3.2. Dio tablice za statistiku sa dinamičkim podacima

Slika 4.3.3. Upravljanje destinacijama i rasporedom

Slika 4.3.4. Kreiranje destinacije