УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Марина Димитријевић, ПР 34/2015

Дигитализација градског саобраћаја

ПРОЈЕКАТ
- Примењено софтверско инжењерство (ОАС) -

Нови Сад, Фебруар 2020

SADRŽAJ

1. OPIS REŠAVANOG PROBLEMA

Projektni zadatak realizuje aplikaciju za digitalizaciju gradskog saobraćaja. Aplikacija pojednostavljuje : evidenciju linija gradskog prevoza, evidenciju stanica,evidenciju karata i mogućnost plaćanja putem paypal-a, kreiranje,prikaz i izmenu reda vožnje , kreiranje, prikaz i izmenu cenovnika, dinamičko praćenje kretanja vozila odredjene linije, evidenciju korisnika,registraciju i prijavu na sistem, kao i prikaz i izmenu ličnih informacija registrovanim korisnicima. Napredne funkcije sistema su dostupne isključivo registrovanim i prijavljenim korisnima, a neregistrovani korisnici imaju pristup osnovnim funkcijama sistema.

Projektni zadatak je realizovan kao klijent-server aplikacija. Serverski deo aplikacije implementiran je korišćenjem C# programskog jezika (.NET WebApi). Za klijentski deo aplikacije korišćen je Angular radni okvir.

Evidencija linija gradskog saobraćaja predstavlja kreiranje, prikaz i brisanje linija. Svaka linija sadrži : redni broj, tip linije i listu stanica. Administatoru sistema je omogućeno kreiranje i brisanje, a svim ostalim korisnicima je omogućen prikaz linija na mapi sa svim njenim podacima.

Pri **evidenciji stanica**, neophodno je u bazi podataka čuvati stanice koje sadrže naziv, adresu, geografske koordinate i listu linija kojima ta stanica pripada. Stanice može da kreira i briše administrator sistema, dok je prikaz postojećih stanica moguć svim korisnicima.

Kupovina karata i mogućnost plaćanja istih putem paypal-a je funcionalnost koja je dostupna svim korisnicima sistema. Tipovi karata mogu biti: vremenska, dnevna, mesečna i godišnja.

Red vožnje sadrži : dan u nedelji, polaske i liniju na koju se odnosi. Mogućnost modifikacije, tj. kreiranje,izmena i brisanje reda vožnje je odobrena administratoru sistema. Korisnicima sistema je omogućen prikaz reda vožnje po izabranim parametrima.

Prikaz **cenovnika** je osnovna funkcija sistema i mogu da je koriste svi korisnici,registrovani i neregistrovani. Kreiranje i izmena cenovnika su dostupni administratoru, dok je brisanje cenovnika funkcija koju sistem ne poseduje. Cenovnik ima odredjen datum važenja, i poseduje cene za svaku od navedenih tipova karata.

Dinamičko praćenje kretanja vozila na mapi podrazumeva praćenje kretanja vozila u realnom vremenu pomoću web socket-a.

Aplikacija omogucava **evidenciju 4 vrste korisnika** : neregistrovani korisnik, registrovani korisnik tj. putnik , kontrolor i administrator. Prijava korisnika na sistem podrazumeva unos ispravnog korisničkog imena i lozinke, a registracija unos sledećih podataka : ime, prezime, email adresa, korisničko ime, lozinka i potvrda lozinke, datum rodjenja, adresa , tip korisnika (djak, penzioner, regularni putnik), kao i slika indeksa ili penzionog čeka ukoliko je kao tip korisnika izabran djak ili penzioner.

2. OPIS KORIŠĆENIH TEHNOLOGIJA I ALATA

Osnovne tehnologije korišćene za implementaciju projekta su .NET WebApi i Angular. Aplikacija se zasniva na klijent-server arhitekturi. Za serversku stranu(*back-end*) je korišćen ASP.NET Web API, dok je klijentska strana(*front-end*) impementirana putem Angular radnog okvira.

ASP.NET Web API je proširiv okvir za izgradnju usluga zasnovanih na HTTP-u kojima se može pristupiti u različitim aplikacijama na različitim platformama kao što su web, Windows, itd. Radi slično kao ASP.NET MVC web aplikacija osim što šalje podatke kao odgovor umesto html prikaza. Karakteristike ASP.NET Web Api-ja :

1. Idealna platforma za izgradnju RESTful usluga
2. Mapira HTTP glagole u imena metoda
3. Podržava različite formate podataka odgovora. Ugradjena podrška za JSON,BSON,KSML format
4. Uključuje novi HttpClient za komunikaciju sa Web Api serverom [1]

AngularJS je strukturni okvir za dinamičke web aplikacije. Omogućuje upotrebu HTML-a kao jezika šablona i omogućava proširenje HTML sintakse kako bi se jasno i sažeto izrazile komponente aplikacije. AngularJS-ovo povezivanje podataka i umetanje zavisnosti eliminišu veći deo koda koji bi inače morao da bude napisan. Sve to se dešava u pretraživaču , što ga čini idealnim partnerom za bilo koju tehnologiju servera.[2] Angular pruža mogućnost izrade *Single-Page-Applications*[SPAs]. Prednost korišćenja *SPAs* je pre svega njihova brzina odgovora koja je ekvivalentna brzini kod desktop aplikacija, pored toga, mogu da rade i u offline režimu i imaju sposobnost samostalnog osvežavanja. Značajno je to što je količina podataka koja se razmenjuje svedena na minimum.[3]

Razvojno okruženje koje je korišćeno na serverskoj strani je Visual Studio 2017 i programski jezik C#. AngularJs verzija 7 je tehnologija pomoću koje je implementirana klijentska,front-end, strana. Razvojno okruženje klijentske strane je Visual Studio Code, a programski kod je pisan u *typescript-u,* HTML-u i CSS-u. Za izgled web stranica je korišćena biblioteka *Bootstrap.*

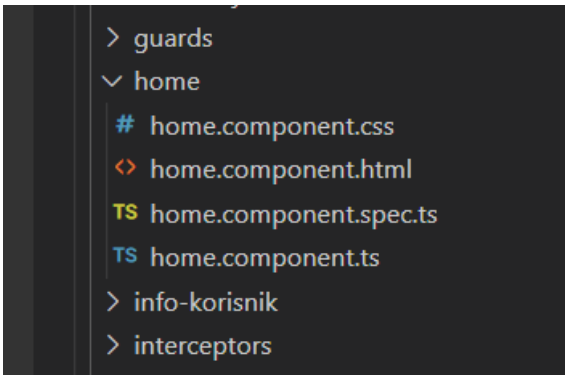
Entity Framework komponenta u okviru .NET platforme obezbedjuje ORM (*object relational mapping*) pristup u programiranju. Jednostavnije rečeno, Entity Framework developerima omogućuje da se podacima bave u formi objekata i odgovarajućih karakteristika, umesto da direktno barataju tabelama i kolonama baza podataka. Osnovni razlog velike popularnosti Entity Framework-a leži u njegovoj sposobnosti da veliki deo koda generiše automatski i na taj način programerima štedi dragoceno vreme i trud. Entity Framework omogućuje nekoliko načina da se uradi ORP mapiranje kroz različiti *development workflow*: *database (schema) first, code first* i *model first*. [4] U ovom projektu je korišćen pristup *Code first* koji znači da se prvo kreiraju klase, pa na osnovu njih baza podataka.

Microsoft SQL Server je relaciona baza podataka. On može da se koristi u različite svrhe kao što su poslovna inteligencija ili za skladištenje podataka. Ipak, SQL Server se najćešće koristi kao krajnja komponenta za smeštanje podataka, a tradicionalne *Client-server* aplikacije i *Web-based* aplikacije često koriste SQL Server za smeštanje aplikativnih podataka.[5] Za potrebe projektnog zadatka je korišćena baš *Microsoft SQL Server* baza.

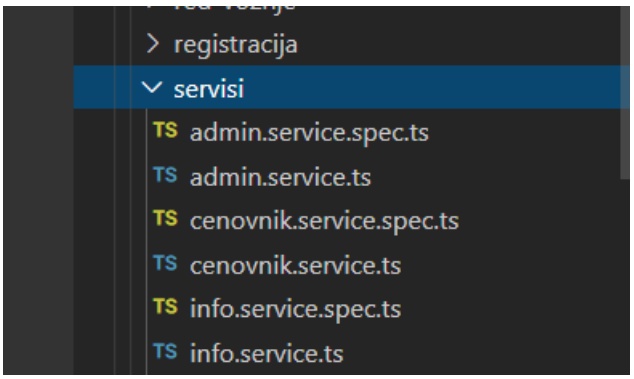
3. OPIS REŠENJA PROBLEMA

Prvi korak u izradi rešenja problema jesu analiza zahteva specificiranih projektnim zadatkom i analize datog inicijalnog projekta. Inicijalni projekat se odnosi na serversku stranu, napravljen korišćenjem ASP.NET Web API-ja u okviru razvojnog okruženja Visual Studio 2017. Kako je ranije navedeno, metoda koja je korišćena u projektu,a odnosi se na Entity Framework, je *Code First* . Po specifikaciji projektnog zadatka, a u skladu sa principom *Code first*, prvo se kreiraju klase koje predstavljaju objekte potrebne za ispravno funkcionisanje aplikacije. Nakon toga, u Package Manager Console-i se prave migracije, pomoću kojih se kreira ili ažurira baza podataka.

Naredni korak u izradi rešenja problema je kreiranje Angular aplikacije, koja predstavlja *front end* projekta. Pri kreiranju aplikacije , kreira se folder sa istim nazivom,skidaju sve neophodne Angular biblioteke, instaliraju i konfigurišu TypeScript, Karma i Protractor. Komponente predstavljaju osnovne gradivne blokove Angular aplikacije.



Slika 1. Struktura komponente



Slika 2. Struktura servisa

Na slici 1 je prikazana struktura konkretne komponente Home. Svaka komponenta u okviru Angular projekta sadrži 4 fajla. Home.component.css sadrži CSS stil za HomeComponent komponentu, home.component.html HTML template povezan sa HomeComponent , home.component.spec.ts sadrži Unit testovi i home.component.ts poslovna logiku HomeComponent komponente.

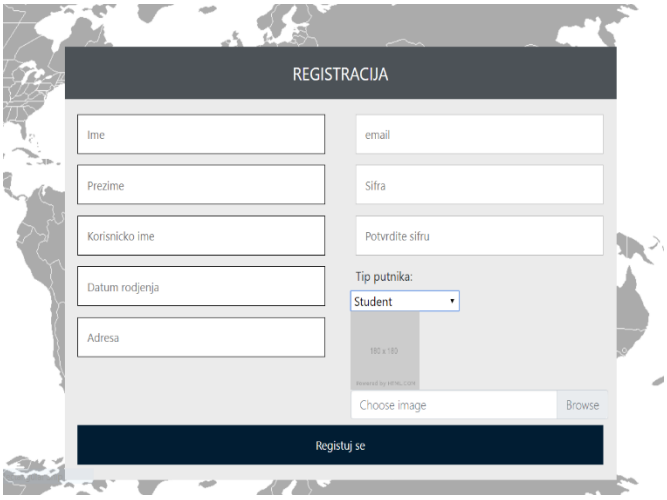
Servisi u Angularu predstavljaju klase koje su dostupne različitim komponentama . Osnovna uloga servisa je komunikacija sa serverskim delom aplikacije. Servisima se takodje omogućava komunikacija izmedju komponenti. Na slici 2 je prikazana struktura konkretnih servisa projektnog zadatka. Za razliku od komponenti, svaki servis se sastoji iz dva fajla. Admin.service.spec.ts sadrži Unit testove, a admin.service.ts poslovnu logiku.

U daljem opisu rešenja problema, akcent će biti stavljen na klijentu aplikaciju.

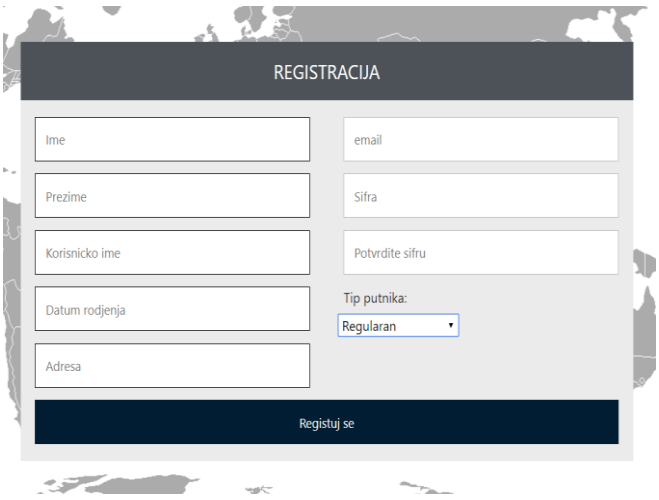
Slika 3. Početna strana aplikacije

Slika 3 predstavlja izgled početne strane koju vidi svaki korisnik sistema, koji se još uvek nije prijavio na sistem.

1. Ikonica Javnog Gradskog Saobraćajnog Preduzeća koja korisnika klikom na nju vraća na početnu stranu, gde god se korisnik u tom trenutku nalazio na aplikaciji.
2. Link koji vodi na stranicu za prijavu korisnika na sistem pomoću korisničkog imena i lozinke
3. Link za registraciju korisnika na sistem
4. Prikaz reda vožnje korisnicima, po odabranim parametrima
5. Prikaz odabrane linije na mapi,sa svim podacima i stanicama
6. Prikaz cenovnika, tj. prikaz cene karte za odabrani tip karte
7. Prikaz lokacije vozila za odabranu liniju



Slika 4. Registracija student/penzioner

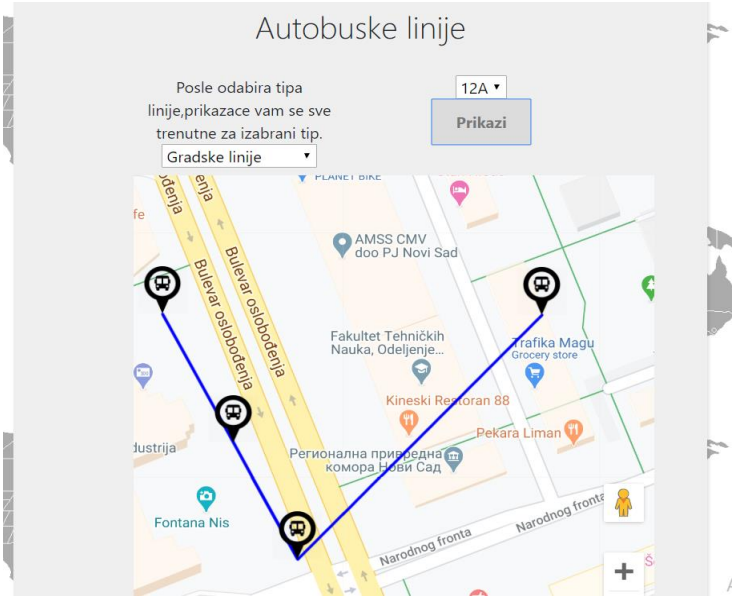


Slika 5. Registracija regularan putnik

Na slikama 4 i 5 prikazane su forme za registraciju korisnika,tj. putnika. Ukoliko korisnik odabere da bude regularan putnik, ne postoji polje za unos dokumenta i to znači da gubi popust za kupovinu bilo koje karte, koju dobijaju studenti i penzioneri(slika 5). Na slici 4. je prikazana forma za registraciju ukoliko korisnik odabere da je penzioner ili student. Posle unosa svih validnih vrednosti, i dostavljenog penzionog čeka, ukoliko je odabran tip putnika penzioner, ili unosa indeksa, ukoliko je odabran tip putnika student, korisnik se uspešno registrovao na sistem. Nakon toga, njegov status je u obradi, i on ne može da kupuje karte, sve dok kontroler ne izvrši verifikaciju njegovog profila. Forma za registraciju je namenjena isključivo za registrovanje korisnika koji će postati putnik, ne postoji forma pomoću koje se administratori ili kontroleri mogu registrovati na sistem.

Slika 6. Prikaz reda vožnje

Na slici 6 se nalazi prikaz reda vožnje za odabrane parametre. Ovako izgleda forma za neregistrovanog korisnika, putnika i kontrolera, u daljem tekstu će biti prikazana i objašnjena i forma koju vide administratori sistema. Tip linije može biti gradski i prigradski. Nakon odabira željenog tipa linije, u polju za odabir konkretne linije se nalaze sve linije za taj tip. U polju za odabir dana se nalaze radni dan, subota i nedelja. Dugme za prikaz reda vožnje je onemogućeno, sve dok nisu popunjena sva polja na osnovu kojih se prikazuje red vožnje.



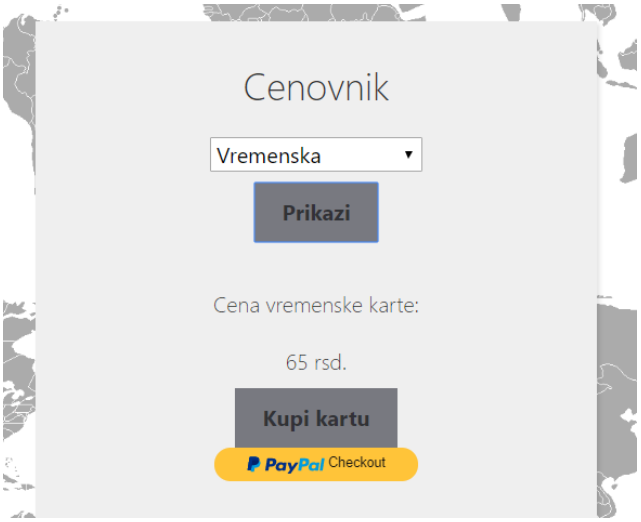
Sledeća funkcionalnost sistema, koju vide svi korisnici sistema, jeste prikaz odabrane linije na mapi. Na slici 7 se nalazi prikaz linije 12A, sa svim njenim stanicama. Kada se kursor miša nadje na odredjenoj stanici, prikazuje se polje sa adresom i nazivom te stanice.

Slika 7. Prikaz linija

```
  <!-- Kreiramo mapu -->
  <agm-map [latitude]="45.242268" [longitude]="19.842954" [zoom]="18" >
<div   *ngIf="pritisnuto" >
    <agm-marker *ngFor="let st of lineStation.Stations" [latitude]="st.XCoordinate" [longitude]="st.YCoordinate"
    [title]="st.Address" [iconUrl]="polyline.icon">
      <!-- Svakom markeru dodajemo info-window (Nije obavezno!)-->
      <agm-info-window [latitude]="st.XCoordinate" [longitude]="st.YCoordinate" >
        <!-- U info-window stavljamo html -->
        <span style="font-size: 16px;font-family:'Times New Roman'">{{st.Name}}</span>
        <br/>
        <span style="font-family:'Times New Roman'; color:gray;">{{st.Address}}</span>
      </agm-info-window>
    </agm-marker>
    <agm-polyline [strokeColor]="polyline.color">
        <agm-polyline-point *ngFor="let point of polyline.path" [latitude]="point.latitude" [longitude]="point.longitude">
        </agm-polyline-point>
    </agm-polyline>
</div>
  </agm-map>
```

Slika 8. Kod za prikaz mape

Deo koda koji je zadužen za prikaz mape na stranici, se nalazi na slici 8. Svaka komponenta koja ima prikaz mape, poseduje ovakav kod, ne identičan ali u velikoj meri sličan. <agm-map> je tag kojim se kreira mapa, latitude i longitude su koordinate mape. <agm-marker> služi u ovom slučaju za prikaz ikonice stanice na mapi, on takodje ima koordinate koje se nalaze u poljima latitude i longitude. <agm-info-window> je za prikaz detalja stanice kada se kursor miša nadje na njoj, a <agm-polilyne> za spajanje dve stanice pravolinijskom linijom. Pored ovog koda, neophodno je u imports niz unutar app.module.ts dodati apiKey koji omogućava korišćenje mapa.

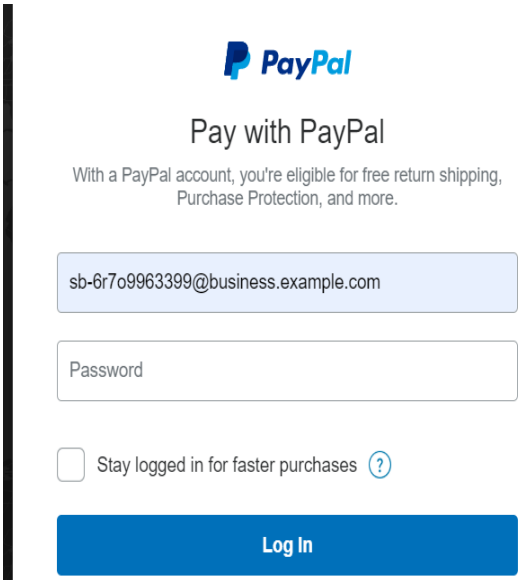


Slika 9. Kupovina karte

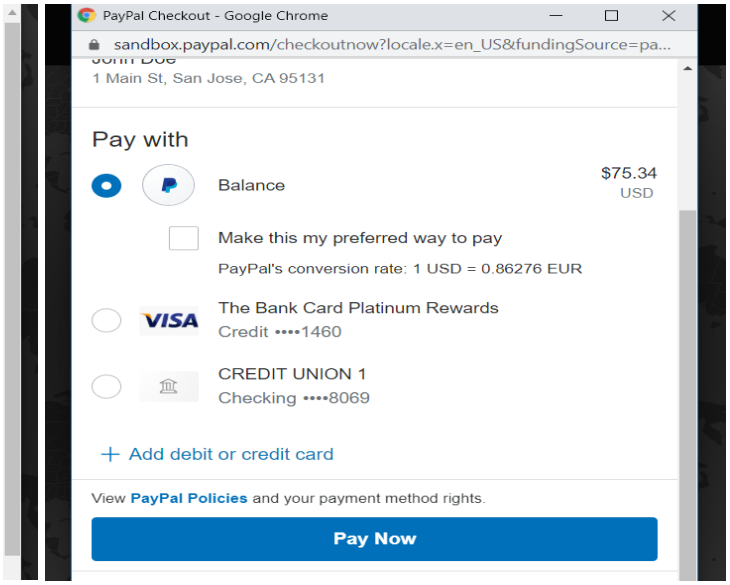
Funkcionalnost sistema koja je omogućena isključivo neregistrovanim korisnicima i putnicima, je kupovina karte. Postoje 4 vrste karte: vremenska, dnevna, mesečna i godišnja. Neregistrovani korisnik ima prikaz svih karti, ali može da kupi samo vremensku kartu, dok putnik može da kupi sve vrste karata. Postoje dva načina za kupovinu karte, a to su putem Paypal-a i redovna kupovina karte koja ne uključuje plaćanje putem Paypal-a već simuliranu kupovinu novcem.

Slika 10. Definisanje konkretnog Paypal naloga

Za potrebe testiranja Paypal-a u okviru projektnog zadatka, u komponenti Cenovnik su definisani kredencijali koji upućuju na već kreiran nalog na PayPal sandboxu(sandbox nalog je development nalog gde je novac virtuelan,tj. simulira se kupovina). Kod koji definiše sandbox nalog je prikazan na slici 10. Kada korisnik pritisne dugme za kupovinu karte preko Paypal-a(slika 9) otvara se prozor koji je prikazan na slici 11.



Slika 11. Prijava na Paypal



Slika 12. Plaćanje karte Paypal

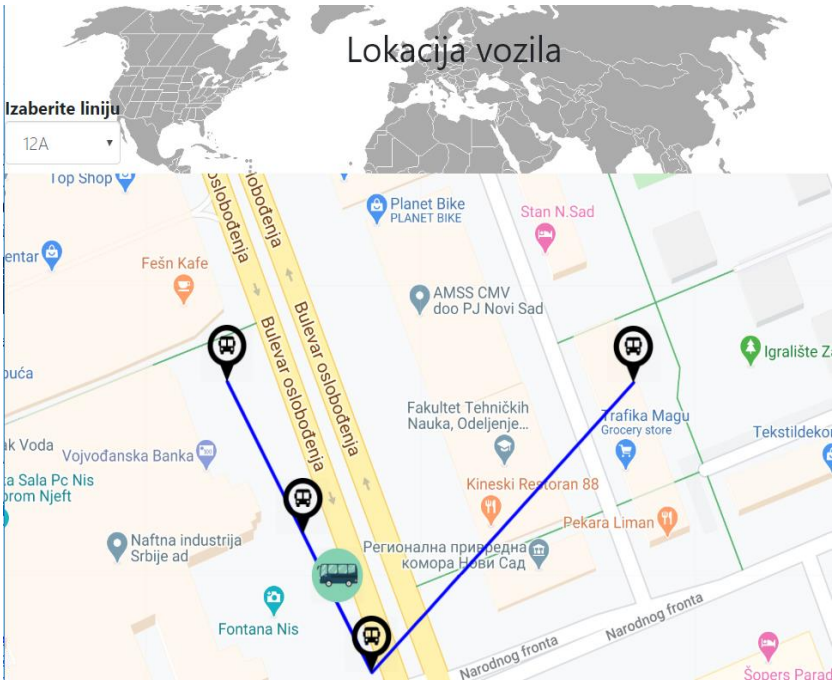
Nakon uspešne prijave na Paypal gde se unosi mejl i šifra koja je definisana kredencijalima u komponenti Cenovnik,koja je prikazana slikom 11. , otvara se prozor za plaćanje karte(slika 12). Kada je kupovina karte uspešna, u bazi podataka na serverskoj strani se svi podaci o karti i id paypala čuvaju zajedno. Primer jedne kupljene karte preko Paypal-a za registrovanog korisnika se nalazi na slici 13.

Polje u bazi „Verifikovana“ je status verifikacije karte, koju vrši kontroler. Konkretno ova karta, par trenutaka nakon kupovine karte nije bila verifikovana od strane kontrolera, te je polje vrednosti False. Kada karta nije kupljena preko Paypal-a, polje u bazi PayPalId ima vrednost null.

	Id	CenaKarteId	Applicatio...	VremeKup...	Verifikovana	PayPalId
▷	1	1	2	2/13/2020 2...	False	4

Slika 13. Tabela Karte u bazi podataka

Dinamičko praćenje kretanja vozila odredjene linije na mapi, je poslednja kartica u nizu na početnoj strani koji korisnik vidi (slika 3) koja izvršava željenu funkcionalnost. Za ovaj deo koda je zadužena komponenta Lokacija. Korisnik može da izabere jednu od postojećih linija i nakon toga prikazuje se marker koji se kreće po toj liniji, od stanice do stanice (tirkizni marker). Kada autobus dodje do kraja linije, on se vraća suprotnom putanjom. Na slici 14 je prikazano kretanje za liniju 12A.



Slika 14. Kretanje vozila jedne linije

Za ovaj deo aplikacije, korišćen je WebSocket protokol. WebSocket je računarski komunikacioni protokol, koji pruža full-dupleks komunikacione kanale preko jedne TCP veze. On omogućava interakciju izmedju web pregledača i web servera,olakšavajući prenos podataka u realnom vremenu.[6] Klijent i server uspostavljaju komunikaciju, pokreće se tajmer koji na svake 2 sekunde šalje niz od dve koordinate, kada klijent primi te koordinate, na njima se iscrtava marker koji predstavlja lokaciju autobusa. Na slici 15 je prikazan deo koda na serverskoj strani koji ovo omogućava.

```
[HubName("notificationsBus")]
public class LokacijaHub : Hub
{
    private static IHubContext hubContext = GlobalHost.ConnectionManager.GetHubContext<LokacijaHub>();
    polja
    public LokacijaHub()
    {
    }
    public void TimeServerUpdates()
    {
        if (timer.Interval != 2000)
        {
            timer.Interval = 2000;
            timer.Elapsed += OnTimedEvent;
        }
        timer.Enabled = true;
    }
    private void OnTimedEvent(object source, ElapsedEventArgs e)
    {
        (source as Timer).Enabled = false;
        if (stations != null)
        {
            if (cnt >= stations.Count)
            {
                unazad = -1;
                cnt = stations.Count-1;
            }
            else if(cnt>=0 && unazad == 0)
            {
                if (broj == 0)
                {
                    double[] niz = { stations[cnt].GeografskeKoordinataX, stations[cnt].GeografskeKoordinataY };
                    hubContext.Clients.All.setRealTime(niz);
                    cnt++;
                    broj++;
```

Slika 15. Serverski deo koda za razmenu koordinata

Linija koda na slici 15 „hubContext.Clients.All.setRealTime(niz)“ je linija u kojoj se na klijentsku metodu „setRealTime“ šalje niz koordinata,prva u nizu je latitude,a druga longitude. Sledeća slika, slika 16, prikazuje deo koda na klijentskoj strani kojim se uspostavlja veza klijenta i servera, primaju podaci sa servera kada se pozove metoda „setRealTime“ i prosledjuje ih dalje kako bi se nacrtao marker na primljenim koordinatama.

```
export class LokacijaService {
  private proxy: any;
  private proxyName: string = 'notificationsBus';
  private connection: any;
  public connectionExists: boolean;
  public notificationReceived: EventEmitter<string>;
  constructor() {
    this.notificationReceived = new EventEmitter<string>();
    this.connectionExists = false;
    // create a hub connection
    this.connection = $.hubConnection("http://localhost:52295/");
    this.connection.qs = {"token" : "Bearer "+ localStorage.jwt};
    // create new proxy with the given name
    this.proxy = this.connection.createHubProxy(this.proxyName);
  }
  public startConnection(): Observable<boolean> {
    return Observable.create((observer) => {
        this.connection.start()
        .done((data: any) => {
            console.log('Now connected ' + data.transport.name + ', connection ID= ' + data.id)
            this.connectionExists = true;
            observer.next(true);
            observer.complete(); })
        .fail((error: any) => {
            console.log('Could not connect ' + error);
            this.connectionExists = false;
            observer.next(false);
            observer.complete(); });   });
  }
  public registerForTimerEvents() : Observable<number[]> {
    return Observable.create((observer) => {
        this.proxy.on('setRealTime', (data: number[]) => {
          console.log("data iz servisa",data);
            observer.next(data);
```

Slika 16. Servis na klijentu za lokaciju vozila

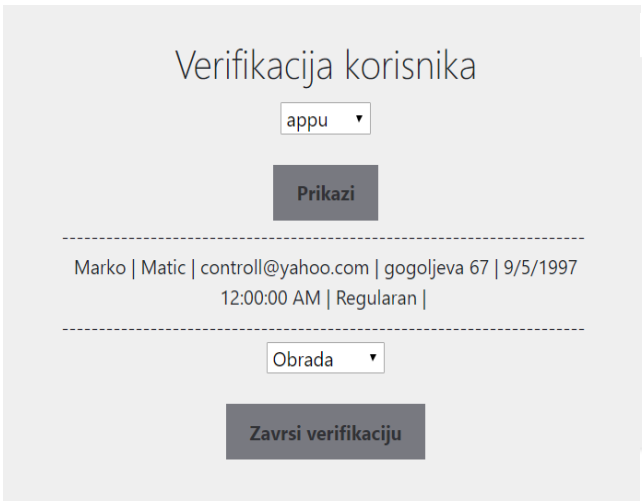
```
public onTimeEvent(pos: number[]){
  this.ngZone.run(() => {
    this.latitude = pos[0];
    this.longitude=pos[1];
    this.time = pos;
     if(this.isChanged){
       this.latitude = pos[0];
        this.longitude = pos[1];
        console.log("pos: ", this.latitude, this.longitude);
        //this.isChanged = false;
      }else{
        this.latitude = 0;
        this.longitude = 0;
      }
  });
}
```

Slika 17. Metoda u komponenti Lokacija

Podatke koje servis primi od servera, on šalje na komponentu Lokacija, gde se unutar „onTimeEvent“, latitudi i longitudi dodeljuju primljeni podaci, tačnije niz od dve koordinate kako je u prethodnom tekstu navedeno. Slika 17 prikazuje „onTimeEvent“ metodu u Lokacija komponenti.

Slika 18. Meni bar za ulogovanog kontrolera

Ulogovani kontroler sistema, vidi meni bar kao sto je prikazan na slici 18. Kontrolor ima mogućnost da verifikuje korisnike i da izvrši validaciju karata,i to su funkcije koje nemaju ni administatori ni putnici.



Slika 19. Verifikacija korisnika

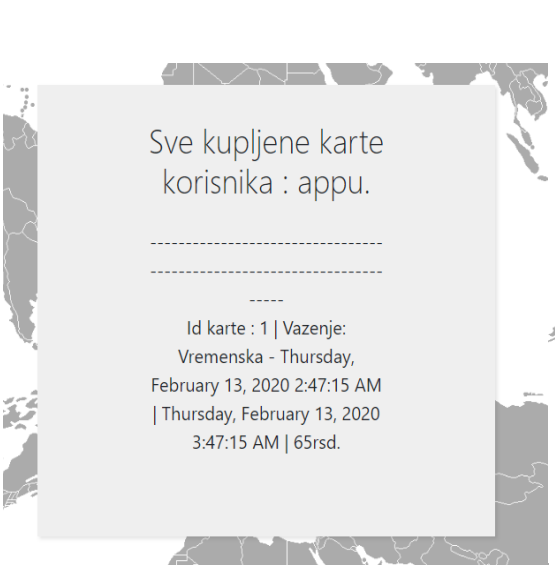
Slika 20. Validacija karata

Slika 19 prikazuje formu za verifikaciju korisnika. Za izabranog korisnika, klikom na dugme „Prikazi“ se ispisuju odredjeni podaci o njemu. Postoje 3 statusa profila korisnika, a to su obrada, verifikovan i odbijen. Kontroler ima mogućnost da u svakom trenutku promeni status profila bilo kod korisnika. Klikom na dugme „Zavrsi verifikaciju“, podaci se prosledjuju serveru i upisuju u bazu.

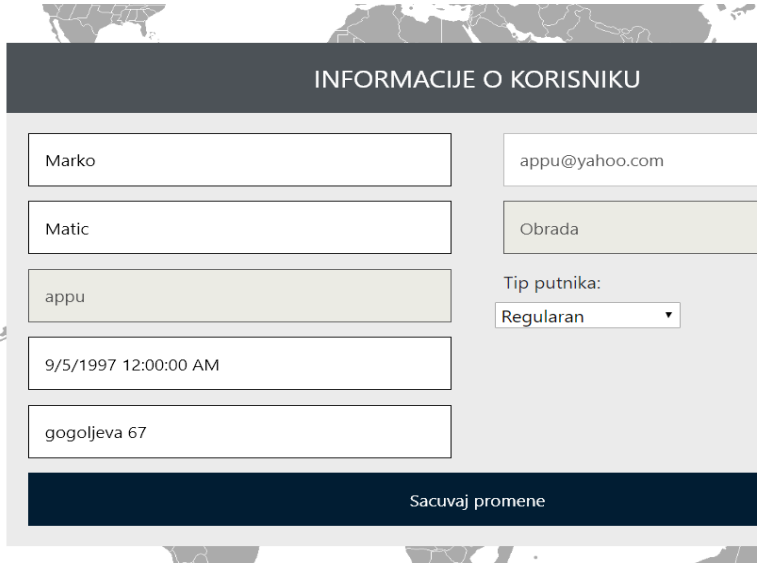
Naredna slika, slika 20, je primer forme za validaciju karata. Izlistaju se Id-evi svih kupljenih karata od svih korisnika i kontroler bira konkretni id. Verifikacija karti vrši na serverskoj strani i potom upisuje u bazu. Vremenska karta važi sat vremena od trenutka kupovine, dnevna do kraja dana kada je kupljena, mesečna do kraja tekućeg meseca i godišnja do kraja tekuće godine.

Slika 21. Meni bar koji vidi ulogovani putnik

Meni bar koji vidi ulogovani putnik(slika 21). Napredne funkcije sistema koje on može da koristi jesu da vidi sve karte koje je kupio i da vidi i ako želi promeni informacije profila. Na slikama 22 i 23 prikazane su te funkcije za konkretnog putnika, u ovom slučaju putnika „appu“.



Slika 22. Kupljene karte putnika „appu“



Slika 23. Informacije o putniku „appu“

Administrator sistema ima mogućnost uredjivanja linija, stanica, redova vožnje i cenovnika. Svako uredjivanje osim cenovnika podrazumeva CRUD(Create,Read,Update,Delete) operacije. Pored brisanja i dodavanja linija i stanica, takodje je moguće i pridružiti stanicu odredjenoj liniji.

U nastavku teksta biće prikazane neke od naprednih funkcija sistema koje koristi administator.

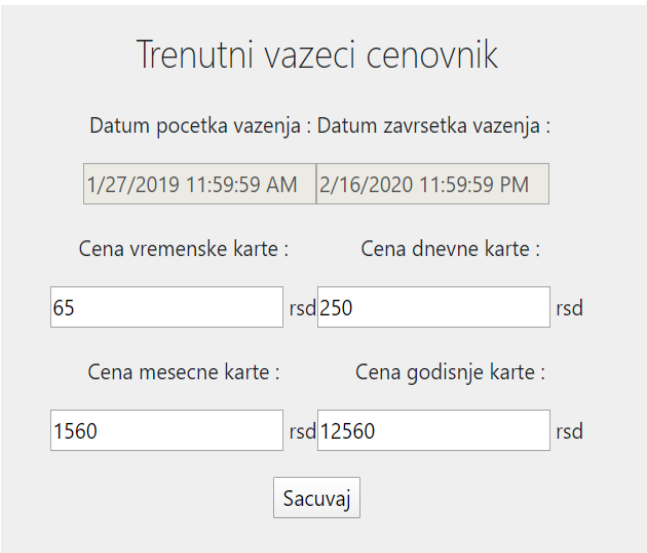
Slika 24. Kreiranje nove stanice

Slika 23 prikazuje kreiranje nove stanice, koja je dodata u konkretnu liniju 7A. Pri kreiranju stanice, popunjavaju se polja za naziv, adresu i kojoj liniji ce pripadati novokreirana stanica. Pored tih polja, na mapi se bira lokacija stanice, koja je prikazana crno-belom ikonicom autobusa. Kreiranje linije je sličan proces kao i kreiranje stanice, sem sto se ne prikazuje mapa za biranje koordinata.

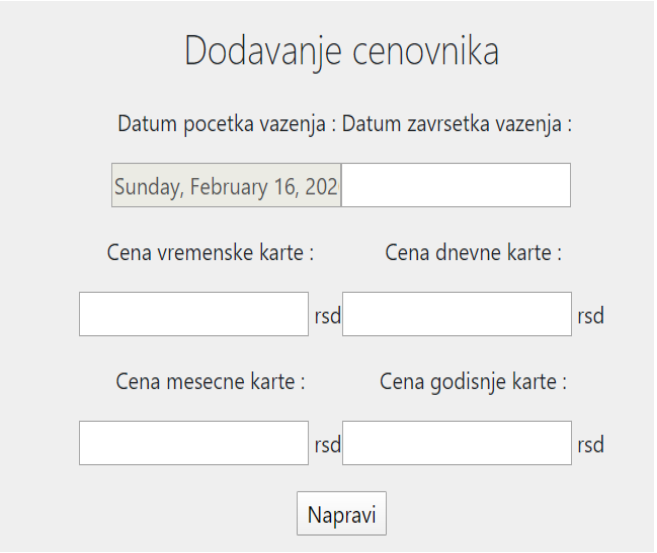
```
[Authorize(Roles = "Admin")]
[System.Web.Http.HttpGet]
[Route("DeleteLine/{lineId}")]
public IHttpActionResult DeleteLine(string lineId)
{
    lock (lockObj)
    {
        if (lineRepo.GetAll().Count() == 0)
        {
            return BadRequest("Line doesn't exist...");
        }
        foreach (var linija in lineRepo.GetAll())
        {
            if (linija.RedBroj.Equals(lineId))
            {
                linija.Aktivna = false;
                db.Entry(linija).State = EntityState.Modified;
                db.SaveChanges();
                break;
            }
        }
        return Ok();
    }
}
```

Slika 25. Kod za brisanje linije

Brisanje linije ili stanice se vrši tako što se izabere naziv željene linije ili stanice, ona se briše logički, što znači da će ona ostati u bazi podataka, ali će polje Aktivna postati False. Kod na serverskoj strani za brisanje linije se nalazi na slici 25.

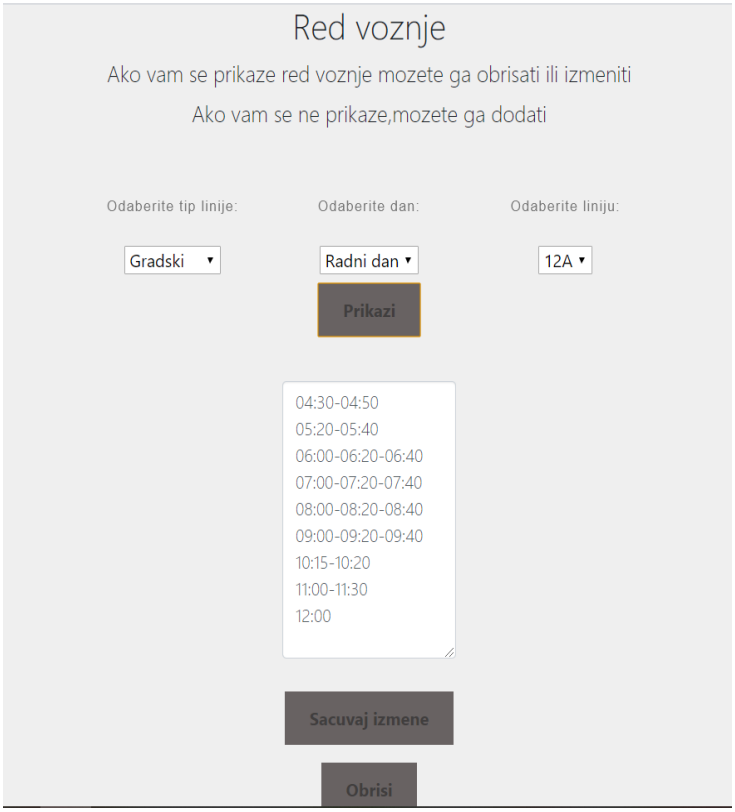


Slika 26. Izmena trenutnog cenovnika



Slika 27. Kreiranje cenovnika

Administratorske funkcije vezane za cenovnik, jesu izmena trenutnog i kreiranje novog cenovnika(slike 26 i 27). Ne postoji mogućnost brisanja cenovnika, jer bi to rezultovalo nepostojanjem cena za karte. Prilikom kreiranja novog cenovnika, početni datum važenja jeste krajnji datum važenja prethodnog cenovnika. Izmena trenutnog cenovnika se odnosi samo na menjanje cena karata.



Slika 28. Prikaz reda vožnje za administratora

Na slici 28 se nalazi prikaz reda vožnje za administratore sistema. CRUD operacije vezane za red vožnje su prikazane na ovoj formi. Kao što na formi piše, ukoliko se za odabrane parametre ne prikaže red vožnje, on se može dodati. Ukoliko se prikaže, mogu se vršiti izmena i brisanje postojećeg reda vožnje za željene parametre.

Parametri koji odredjuju red vožnje su tip linije, dan u nedelji i konkretna linija koja je izabranog tipa linije.

Na serverskoj strani aplikacije, iskorišćen je repozitorijum patern. Koristi se u aplikativnom sloju za pristup podacima. Repozitorijumske klase se registruju unutar interfejsa „IUnitOfWork“, za upravljanje podacima nad bazom. Klasa UnitOfWork održava listu repozitorijumskih klasa u interfejsu IunitOfWork koji definiše operacije nad UnitOfWork klasi. Metoda SaveChanges() skladišti sve izmene u bazu. [7] Na slici 29 se nalazi kod koji demonstrira IUnitOfWork klasu na serveru.

```
public interface IUnitOfWork : IDisposable
{
    ILinijaRepository linijaRepository { get; set; }
    IStanicaRepository stanicaRepository { get; set; }
    IAutobusRepository autobusRepository { get; set; }
    ITipKarteRepository tipKarteRepository { get; set; }
    ITipDanaRepository tipDanaRepository { get; set; }
    IRedVoznjeRepository redVoznjeRepository { get; set; }
    IKartaRepository kartaRepository { get; set; }
    ICenovnikRepository cenovnikRepository { get; set; }
    ICenaKarteRepository cenaKarteRepository { get; set; }
    ITipPutnikaRepository tipPutnikaRepository { get; set; }
    IKorisnikRepository korisnikRepository { get; set; }
    ITipLinijeRepository tipLinijeRepository { get; set; }
    ILinijeStaniceRepository linijeStaniceRepository { get; set; }
    IStatusRepository statusRepository { get; set; }
    IPayPalRepository payPalRepository { get; set; }

    int Complete();
}
```

Slika 29. IUnitOfWork klasa

```
import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Injectable } from '@angular/core';

@Injectable()
export class TokenInterceptor implements HttpInterceptor {
    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

        const jwt = localStorage.getItem('jwt');
        console.log(req);
        console.log('usao inter');
        if (jwt) {
            req = req.clone({
                setHeaders: {
                    'Authorization': 'Bearer ' + jwt
                }
            });
        }

        return next.handle(req);
    }
}
```

Slika 30. Jwt tag

Na klijentkoj strani je implementiran Interceptor mehanizam(slika 30) . Interceptor proverava „jwt“ tag i postavlja ga u zaglavlje svakog klijentkog zahteva. Klijentski zahtev se potom upućuje na servis.

Autentifikacija je proces odredjivanja identiteta nekog subjekta. U praksi subjekt daje odredjene podatke po kojima druga strana može utvrditi da je subjekt upravo taj kojim se predstavlja.

Autorizacija je funkcija navodjenja prava pristupa resursima koji se odnose na bezbednost informacija i računarsku bezbednost uopšte, naročito kontrolu pristupa.

Na serverskoj strani, autorizacija je uradjena proverom *Role* kojoj korisnik pripada, iznad svake metode koja dozvoljava pravo pristupa samo specificiranoj grupi korisnika. Na slici 31 je prikazana autorizacija za metodu za izmenu cenovnika, što je dozvoljeno isključivo administratoru sistema(Role="Admin") i autorizaciju za metodu GetCena, koja je dozvoljena isključivo putniku,koji ima ulogu u sistemu „AppUser".

```
[Authorize(Roles = "Admin")]
[System.Web.Http.HttpPost]
[Route("IzmenaCenovnika2/{model}")]
public async Task<IHttpActionResult> IzmenaCenovnika2(CenovnikBindingModel model)//cu


// GET: api/CenaKarte/GetCena
[Authorize(Roles ="AppUser")]
[ResponseType(typeof(double))]
[Route("GetCena/{type}/{username}")]
public IHttpActionResult GetCena(int type, string username)//u ondaosu na tip dana vr
```

Slika 31. Autorizacija na serverskoj strani

Na klijentskoj strani autentifikacija je prijava na sistem, gde korisnik unošenjem korisničkog imena i lozinke, dokazuje svoj identitet. Autorizacija se vrši preko „Guards"-a.

```
import { Injectable } from '@angular/core';
import { CanActivate, Router,ActivatedRouteSnapshot,RouterStateSnapshot,CanActivateChild,
} from '@angular/router';

@Injectable({
  providedIn: 'root',
})
// ovaj guard ce se koristiti za sve putanje kojima moze da pristupi samo AppUser
export class AdminGuard implements CanActivate, CanActivateChild {
  constructor(private router: Router) { }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    if (localStorage.role === 'Admin') {
      return true;
    }
    // not logged in so redirect to login page
    else {
      console.error("Can't access, not admin");
      if (localStorage.getItem('username') === null) {
        this.router.navigate(['/login']);
      } else {
        this.router.navigate(['/']);
      }
      return false;
    }
  }
  canActivateChild(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    return this.canActivate(route, state);
  }

}
```

Slika 32. Admin guard

```
const routes: Routes = [
  {path: '',component: HomeComponent},
  {path:'registracija',component:RegistracijaComponent},
  { path: 'login', component: LoginComponent },
  {path: 'linije',component:LinijeComponent},
  {path: 'red-voznje',component: RedVoznjeComponent},
  {path: 'cenovnik',component:CenovnikComponent},
  {path: 'info',component: InfoKorisnikComponent,canActivate:[UserGuard]},
  {path:'dodavanjeCenovnika',component:DodavanjeCenovnikaComponent,canActivate:[AdminGuard]},
  {path:'lokacija',component : LokacijaComponent},
  {path:'izmenaCenovnika',component:IzmenaCenovnikaComponent,canActivate:[AdminGuard]},
  {path:'karteKorisnik',component:KarteComponent,canActivate:[UserGuard]},
  {path: 'dodavanjeLinije',component:DodavanjeLinijeComponent,canActivate:[AdminGuard]},
  {path:'brisanjeLinije',component:BrisanjeLinijeComponent,canActivate:[AdminGuard]},
  {path: 'dodavanjeStanice',component:DodavanjeStaniceComponent,canActivate:[AdminGuard]},
  {path:'kontrolerKorisnici',component:KontrolerKorisniciComponent,canActivate:[ControllerGuard]
  {path:'kontrolerKarte',component:KontrolerKarteComponent,canActivate:[ControllerGuard]},
  {path:'pay',component:PaypalProbaComponent},
];
```

Slika 33. Autorizacija za komponente

Na slici 32 se nalazi kod na klijentskoj strani za autorizaciju. Kako smo i ranije naveli, postoje 4 uloge korisnika sistema, administrator, putnik i kontroler i neregistrovani korisnik koji nema svoju konkretnu ulogu. ControllerGuard i UserGuard se implementiraju skoro identično kao i AdminGuard, oni obaveštavaju korisnika da nema pristup željenoj stranici i sprečavaju neautorizovani pristup.

Slika 33 predstavlja klasu app-routing.module.ts, gde se Guardovi vezuju za konkretne komponente.

4. PREDLOZI ZA DALJA USAVRŠAVANJA

Aplikacija za digitalizaciju gradskog saobraćaja, implementirana ovim projektnim zadatkom uradjena je po uzoru na zvaničan sajt JGSP-a. Odredjene funkcionalnosti se razlikuju, kao što su kupovina karti i dinamičko praćenje vozila. Red vožnje je u potpunosti realizovan kao na sajtu Javnog Gradskog Saobraćajnoj Preduzeća.

Ideja i zamisao aplikacije je složena i uvek postoje delovi koji zahtevaju nadogradnju.

Neke od ideja su sledeće:

1. Prilikom kreiranja linije, da se odmah kreira i inverzna
2. Prikaz više linija na mapi
3. Realnije kretanje vozila
4. Spajanje stanica preko ulica
5. Performanse sistema

Kada se kreira nova linija, bilo bi poželjno napraviti i da se automatski kreira inverzna linija, kao u realnom sistemu.

Aplikacija podržava prikaz samo jedne linije na mapi, naprednija funkcija sistema bi bila da je omogućen prikaz više linija na mapi.

Server isporučuje koordinate klijentu za lokaciju vozila, na svake dve sekunde. Realnije kretanje vozila bi zahtevalo mnogo napredniji algoritam. Proširenje ove implementacije bi bilo dodavanje GPS prijemnika u svaki autobus i praćenje prijemnika u realnom vremenu.

Trenutna implementacija prikaza linije izmedju stanica je pravolinijska, ispravniji prikaz bi značio spajanje preko ulica, pravim putem.

Trenutna baza podataka nema previše podataka, niti tabela. Akcenat je bio na funkcionalnosti, a ne na performansama sistema. Može se predpostaviti da u slučaju velikog broja podataka, sistem bi bio znatno sporiji.

5. LITERATURA

[1] *https://www.tutorialsteacher.com/webapi/what-is-web-api*
[2] *https://docs.angularjs.org/guide/introduction*
[3] *https://startit.rs/single-page-apps-uvod-u-ember-js/*
[4] *https://www.helloworld.rs/blog/ENTITY-FRAMEWORK-za-data-orijentisane-aplikacije/289*
[5] *http://www.link-university.com/lekcija/Podr%C5%A1ka-za-SQL-Server-baze-podataka/5277*
[6] *https://en.wikipedia.org/wiki/WebSocket*
[7] *https://www.codeproject.com/Articles/825646/Generic-Repository-and-UnitofWork-patterns-in-MVC*