UNIVERZITET U NOVOM SADU
FAKULTET TEKNIČKIH NAUKA U NOVOM SADU

Marko Mijatović

APLIKACIJA ZA DIGITALIZACIJU GRADSKOG PREVOZA

ZAVRŠNI RAD
- Osnovne akademske studije -

Novi Sad, 2019.

Obrazac **Q2.HA.11-04** – Izdanje 1

	UNIVERZITET U NOVOM SADU ●**FAKULTET TEHNIČKIH NAUKA** 21000 NOVI SAD, Trg Dositeja Obradovića 6	Datum:
	ZADATAK ZA IZRADU DIPLOMSKOG (BACHELOR) RADA	List: 1/1

(Podatke unosi predmetni nastavnik - mentor)

Vrsta studija:	☐ **Osnovne akademske studije**
Studijski program:	**Primenjeno softversko inženjerstvo**
Rukovodilac studijskog programa:	**prof. dr Dragan Popović**

Student:	**Marko Mijatović**	Broj indeksa:	**PR21/2015**
Oblast:	**Veb programiranje**		
Mentor:	**prof. dr Branko Milosavljević**		

NA OSNOVU PODNETE PRIJAVE, PRILOŽENE DOKUMENTACIJE i ODREDBI STATUTA FAKULTETA IZDAJE SE ZADATAK ZA DIPLOMSKI RAD, SA SLEDEĆIM ELEMENTIMA:

- problem – tema rada;
- način rešavanja problema i način praktične provere rezultata rada, ako je takva provera neophodna;
- literatura

NASLOV DIPLOMSKOG (BACHELOR) RADA:

Aplikacija za digitalizaciju gradskog prevoza

TEKST ZADATKA:

Analizirati sledeću tehnološku platformu za izradu veb aplikacija: serversko razvojno okruženje Node.js, radni okvir za serverske aplikacije Express, relacionu bazu podataka MySQL, ne-relacionu bazu podataka MongoDB, i radni okvir za razvoj veb klijenata Angular. Specificirati aplikaciju za evidenciju stanica, vozila i linija gradskog saobraćaja, upravljanje cenovnikom vožnji, simulaciju kretanja vozila, evidenciju korisnika, kupovinu voznih karata i plaćanje putem PayPal servisa. Implementirati aplikaciju na datoj platformi. Dokumentovati rešenje i analizirati dobijene rezultate.

Rukovodilac studijskog programa:	Mentor rada:

Primerak za: ☐ - Studenta; ☐ - Mentora

SADRŽAJ

1. OPIS REŠAVANOG PROBLEMA

Potrebno je ponovo implementirati back-end stranu postojećeg projekta web aplikacije, iz C# programskog jezika i .NET okruženja u Node.js okvir. Za bazu podataka je korišćena ne-relaciona Mongo baza podataka umesto postojeće relacione SQL baze. Front-end strana aplikacije ostaje ista i sve funkcionalnosti potrebno je da rade kao sa prvim rešenjem back-end strane u C# programskom jeziku.

Realizovana je aplikacija za digitalizaciju gradskog saobraćaja. Aplikacija treba da pojednostavi evidenciju: linija gradskog saobraćaja, karata i putnika, reda vožnje i lokacije vozila.

Sistem je osmišljen tako da ga mogu koristiti neregistrovani korisnici, a korisnici koji se registruju na sistem imaju određenih pogodnosti.

U sistemu postoje četiri vrste korisnika: administrator, kontrolor, putnik i neregistrovani korisnik. Administrator ima odobrene sve operacije u sistemu. Može da dodaje nove linije i stanice i da ih briše. Za dodatu liniju može da izabere stanice na koje će ona stajati. Može da dodaje nova vozila u sistem, menja postojeći cenovnik ili dodaje novi. Kontrolor ima mogućnost da kontrološe karte, pregleda dokumente i podatke korisnika koji su poslati na registraciju. Ima mogućnost da im prihvati ili odbije registraciju. Putnik i neregistrovani korisnik imaju iste mogućnosti. Mogu da kupe kartu, prikažu stanice na koje staje izabrana linija i prate trenutnu lokaciju autobusa. Registrovani korisnik ima mogućnost plaćanja karte preko *PayPal* naloga i dobijanja stalnih obaveštenja na mejl kada je izvršena kupovina karte. Pogodnost registracije je i mogućnost ostvarivanja uštede kupovinom dnevne, mesečne ili godišnje karte.

Korisnik prilikom registracije unosi ime, prezime, datum rođenja, mejl adresu, šifru koju mora da potvrdi, bira jedan od tipova putnika (student, penzioner ili običan). Ako je student ili penzioner mora da pošalje sliku indeksa ili penzionog čeka. Može da pošalje registraciju i bez dokumenata, a dokument je u obavezi da dostavi kasnije da bi registracija bila odobrena. Omogućeno je korišćenje dela sistema dok kontrolor ne odobri registraciju. Moguće je praćenje da li je registracije odobrena. Na mejl korisnika stiže obaveštenje kada mu kontrolor odobri registraciju.

Korsnik može da bira koju će kartu da kupi. Vremenska karta traje sat vremena, dnevna, mesečna i godišnja u toku dana, meseca i godine kupovine. Korisnik može da pošalje upit sistemu i dobije trenutnu cenu za određenu kartu. Ako je korisnik penzioner ili student dobija popust na redovnu cenu karte. Prilikom kupovine karte, u bazi podataka čuvaju se

informacije o karti i korisniku koji je izvršio kupovinu. Kontrolor kasnije može da izvrši kontrolu karte, proverava da li je karta važeća.

Registrovani korisnik ima mogućnost da menja svoje podatke koje je poslao prilikom registracije.

Ovakav sistem je moguće koristiti u različitim situacijama i ima široku primenu. Više tipova korisnika ima mogućnost da koristi sistem i lako je izmeniti i prilagoditi ga za novi tip korisnika. Tehnologije koje su korišćene pružaju lako dodavanje novih mogućnosti u sistem.

2. OPIS KORIŠĆENIH TEHNOLOGIJA I ALATA

U ovom poglavlju će biti opisane tehnologije i alati koji su korišćeni u izradi zadatka.

Korišćene tehnologije: C# programski jezik, Angular 7 radni okvir, SQL relaciona baza podataka, Entity Framework, .NET WEB API 2, Repository paretn, JWT (*Json Web Token*) autentifikacija.

Korišćeni alati: Microsoft Visual Studio 2017, Visual Code, Sql Server Managment Studio, Postman.

Serverska strana projekta rađena je u Microsoft Visual Studio 2017 razvojnom okruženju, C# programskom jeziku sa .NET WEB API 2 tehnologijom i Repository paternom.

Klijentska strana projekta rađena je u Visual Studio Code razvojnom okruženju i Angular 7 okviru.

2.1. Microsoft Visual Studio 2017

Microsoft Visual Studio 2017 je integrisano razvojno okruženje koje se koristi za razvijanje, pokretanje i objavljivanje programa. Postoji više verzija ovog okruženja, *Community, Professional, Enterprise, Code*. Sve verzije se mogu koristiti na *Windows* operativnom sistemu dok je samo *Code* verzija podržana za *Linux* i *MacOs*. Visual Studio između ostalih sadrži editor koda, *debugger, solution explorer* i dosta drugih alata. U toku razvoja programa u ovom okruženju omogućen je uvid u greške i njihovo lakše otklanjanje uz *debugger* alat. Moguće je imati i uvid u bazu podataka iz ovog okruženja, vršiti izmene i dodavati nove podatke u bazi uz *solution explorer* alat.

2.2. .NET Framework

.NET Framework predstavlja kontrolisano okruženje za razvoj aplikacija na Windows operativnom sistemu. .NET podržava više programskih jezika. Podržan je i jezik korišćen u izradi back-end starne C#. Programi u .NET Framework-u se izvršavaju u softverskom okruženju CLR (*Common Language Runtime*), koje kontroliše odnos sa operativnim sistemom. Tako je obezbeđeno razvijanje aplikaciju nezavisno od operativnog sistema.

2.3. C# programski jezik

C# je objekto orijentisani programski jezik. C# ima za cilj da pruži moderan i jednostavan razvoj visokih preformansi na .NET platformi, na internet stranicama [2], [3] i knjizi [1] je detaljno opisan razvoj. Namenjen je za razvoj različitih vrsta aplikacija, web , konsolnih, WPF (*Windows Presentation Foundation*) aplikacija. C# je strogo tipiziran jezik. Sastoji se od .cd datoteka, jedna .cs datoteka predstavlja jednu klasu. Klasa je osnovni subjekat nad kojim se radi u objektom programiranju. Prednosti u odnosu na ostale programske jezike je njegovo razvijanje od strane velike kompanije Microsoft. Postoji mnogo biblioteka i pomagala u razvojnom okruženju koje olakšavaju rad sa C# programskim jezikom.

2.4. Entity Framework

Entity Framework je objektno-relacioni okvir koji je Microsoft razvio da bi se olakšao posao pri razvijanju aplikacije koja koristi bazu podataka kao skladište. Okvir apstrahuje veze sa relacionom bazom podataka tako da programer koristi niz objekata. Kod starijih tehnologija bilo je potrebno napisati sql upit i tako doći do podataka iz baze. Entity Framework nam nudi sve to odrađeno, podatke nam vraća iz baze u obliku našeg modela sa kojim radimo. Tako se razdvaja logika aplikacije od logike rada sa bazom. Entitz Framework nudi dve vrste rada sa bazom, *Database-First* i *Code-First. Database-First* je postupak dobijanja modela podataka iz postojeće baze. Prednost ovog načina je tada se baza može projektovati do visokog nivoa detalja i posle se dobija odgovarajući model klasa. Ovaj način koriste projektanti baza podataka. U projektu je korišćen *Code-First* postupak. On omugućava da lako dođemo do baze podata iz koda. Nije potrebno baviti se projektovanjem baze. Dovoljno je osmisliti model koji će biti odgovarajući za potrebe sistema. Iz klasa koje odgovaraju modelu Entity Framework generiše bazu podataka.

2.5. Repository patern

U okviru serverske strane korišćen je Repository patern. Patern pruža apstrakciju podataka tako da aplikacija radi sa jednostavnom apstrakcijom podataka koja je približno slična onoj iz baze podataka. Stvara se novi sloj između baze podataka i aplikacije. Tako nas ograničava da direktno radimo sa podacima u aplikaciji. Izabran je za rad jer olakšava dodavanje, brisanje i

ažuriranje podataka u kolekciji bez potrebe za nekim dodatnim bavljenjem sa bazom podataka.

2.6. Visual Studio Code i Angular 7

Visula Studio Code je moćno razvojno okruženje za uređivanje izvornog koda. Podržava JavaScript, TypeScript, Node.js i ima bogat paket proširenja za druge programske jezike. Najviše ga koriste web programeri. U izradi aplikacije korišćen je za klijentsku stranu a kasnije i za serversku kada je prepravljana u Node.js.

Angular 7 je okvir za razvijanje web aplikacija u HTML-u i TypeScript-u. Zasnovan na TypeScript-u. Angular aplikacija je definisana skupom NgModula, više detalja dato je na internet stranici [7]. Aplikacija uvek ima korenski modul od kojeg se pokreće. Na korenski modul je moguće dalje dodavanje i proizvoljno organizovanje modula. Prednost u odnosu na starije tehnologije je organizacija projekta u komponente. Svaka komponenta ima svoj .ts i .html fajl. U .ts falju nalazi se logika i podaci koji se prikazuju u .html fajlu. U toku rada aplikacije svaka izmena stanja u delu komponente se automatski osvežava i prikazuje se novo stanje. Nije potrebno osvežiti celu stranicu ako se sastoji iz više komponenti, osvežava se samo komponenta u kojoj se desila izmena. Neke od tehnologija sličnih Angularu, koriste za automatsko osvežavanje virtualno DOM (*Document Object Model*) stablo. Angular nema virtualno DOM stablo već koristi svoj mehanizam za otkrivanje promena. Svaka Angular komponenta ima detektor promena koji se kreira u vreme pokretanja aplikacije. Detektor radi jednostavan posao. Za svako svojstvo proverava vrednosti pre i posle i ako su različiti postavlja flegove izmene na tačno, koji dalje vode do osvežavanja komponente novim vrednostima.

2.7. Sql Server Management Studio

Sql Server Management Studio je softver sa kojim se upravlja bazama podataka. Poseduje grupu grafičkih alata sa brojnim mogućnostima uređivanja, konfigurisanja, pristupa i upravljanja bazama podataka.

Sql je relaciona baza podataka. Korišćena je za skladištenje informacija iz sistema. U bazi postoje tabele sa informacijama o: korisnicima, linijama, stanicama, vozilima, redu vožnje, kartama i cenovnicima. Između određenih tabela postoje relacije.

3. OPIS REŠENJA PROBLEMA

Pre kreiranja aplikacije osmišljen je relacioni model koji će rešiti problem skladištenja podataka u aplikaciji. Ukratko o tabelama koje se nalaze u bazi. Tabela sa korisnicima sadrži podatke koje je korisnik poslao prilikom registracije i dva dodatna polja. Polje koje ukazuje da li je korisniku odobrena registracija i drugo polje koje označava koju ulogu korisnik ima u sistemu. Tabela cenovnik ima datum važenja cenovnika i da li je trenutni cenovnik aktuelan. Polje aktuelnosti je potrebno jer je moguće napraviti novi cenovnik pre isteka važenja starog. Tada nije moguće utvrditi koji je trenutno važeći cenovnik samo uz pomoć datuma važenja cenovnika. U tom slučaju bi se preklapali datumi važenja novog i starog cenovnika. Karte imaju svoju tabelu sa poljima da li je karta čekirana, do kada važi, podatak o korisniku koji je kupio kartu, cenom karte i brojem transakcije, ako je karta plaćena preko PayPal-a. Tabela cena karte je projektovana jer u sistemu postoji više tipova karti pa se u ovoj tabeli čuva cena za svaki tip karte. Linije imaju tabelu sa oznakom linije kojim se korisnik vozi. Red vožnje sadrži podatak kada su polasci, za koji dan u nedelji i za koju liniju. Stanica ima svoj naziv, adresu i koordinate gde se ona nalazi na mapi. Vozilo sadrži koordinate trenutne lokacije, tip vozila i oznaku na kojoj liniji saobraća.

U sistemu postoje Binding Modeli koji su korišćeni za prijem podataka na servis i dalji rad sa njima. U tim modelima postoje polja koja su naznačena kao obavezna, kao što je korisničko ime kod prijema podataka korisnika za registraciju. Polje šifre je takođe obavezno ali ima i minimalnan broj karaktera koja je postavljena na 6 (Listing 3.1.). Tip podatka je postavljen na Password što znači da šifra koja stiže mora da ima jedno veliko slovo, broj i jedan znak kao karakter. Ako korisnik pošalje šifru koja ne odgovara podešenim uslovima dobiće poruku o grešci pri registraciji. Drugo polje za šifru služi za njenu potvrdu (Listing 3.2.). Potrebno je uneti dva puta šifru, servis će ih uporediti i ako se ne poklapaju poslaće poruku o grešci korisniku.

```
[Required]
[StringLength(100, ErrorMessage – "The {0} must be at
least {2} characters long.", MinimumLength = 6)]
[DataType(DataType.Password)]
[Display(Name = "Password")]
public string Password { get; set; }
```

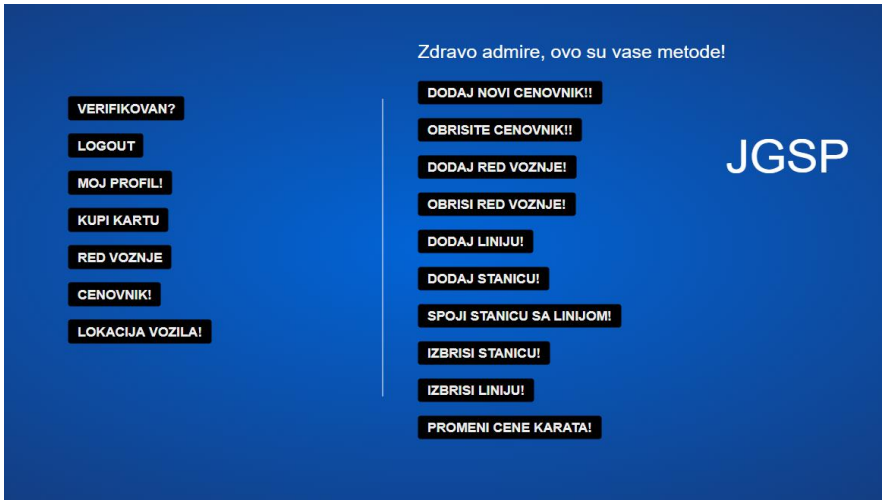
Listing 3.1. Obavezno polje *Password*

```
[DataType(DataType.Password)]
[Display(Name = "ConfirmPassword")]
[Compare("Password", ErrorMessage = "The password and
confirmarion password do not match.")]
Public string ConfirmPassword { get; set; }
```

Listing 3.2. Potvrda polja *Password*

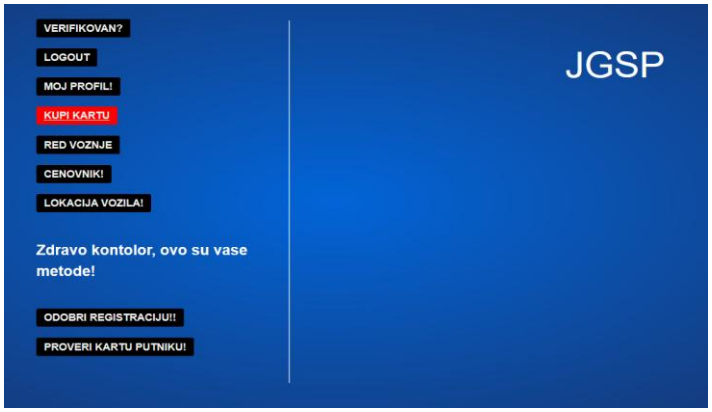
Sistem počinje Home page stranicom koja nudi vise različitih prikaza u zavisnosti od korisnika. Na početnoj stranici moguće je izabrati registraciju ili logovanje na sistem ako je prethodno korisnik registrovan.

Admin dodatne funkcionalnosti može da nađe sa desne strane početnog ekrana dok su sa leve strane standardne operacije za običnog korisnika kojima i admin ima pristup (Slika 3.1.).



Slika 3.1. Početna strana za admina

Početna strana za kontrolora ima dodatne dve operacije, odobrenje registracije i provera karte, u odnosu na običnog korisnika (Slika 3.2.).



Slika 3.2. Početna strana za kontrolora

Za neregistrovanog korisnika prvi korak je registracija a potom i logovanje korisnika (Slika 3.3. i 3.4.). Prilikom logovanja sa servera se šalje jwt tag. Jwt je standard koji je opisao način za sigurno prenošenje informacija između dve strane. Informacijama koje se prenose može se u potpunosti verovati jer su digitalno potpisane. Digitalni potpis kod jwt standarda je moguće napraviti pomoću tajnog ili kombinacije javnog i tajnog ključa RSA (*Rivest-Shamir-Adleman*) algoritam. Jwt je iskorišćen da bi se u sistemu vršila autorizacija i proverilo koju korisnik ima ulogu. Na taj način je osigurano da korisnici koji nemaju dozvolu za određene operacije u sistemu ne mogu da pristupe tom delu aplikacije.

Slika 3.3. Login

Prilikom registracije potrebno je popuniti sva polja osim dodavanja slike koje je opciono.

Slika 3.4. Registracija

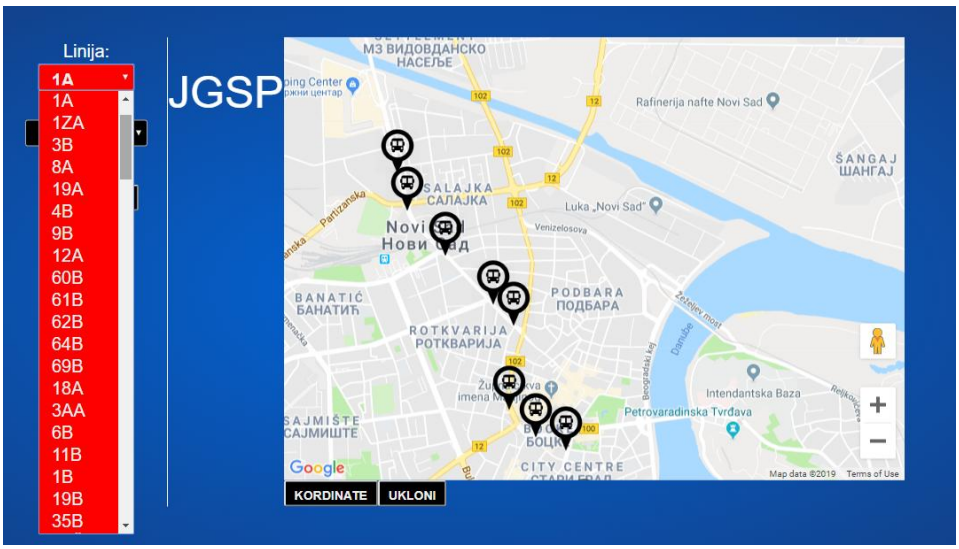
Na klijentskoj strani implementiran je Interceptor mehanizam (Listing 3.3.). Mehanizam presreće svaki zahtev sa servera i proverava jwt tag i postavlja ga u zaglavlje svakog klijenstkog zahteva koji se upućuje na servis.

```
@Injectable()
Export class TokenInterceptor implements
HttpInterceptor{
      Intercept(req: HttpRequest<any>, next:
      HttpHandler): Observable<HttpEvent<any>>{
            Let jwt = localStorage.jwt;

            if(jwt){
                  req = req.clone({
                        setHeaders: {
                              “Authorization”:”Berare” +
                              jwt;
                        }
                  });
            }
            Return next.handle(req);
      }
}
```

Listing 3.3. Interceptor mehanizam

Prikaz reda vožnje i rasporeda stanica za odabranu liniju dozvoljen je svim korisnicima. U komponenti za prikaz reda vožnje i rasporeda stanica, pri učitavanju prikazuje se mapa, ponuđene linije koje su dostupne u sistemu i dan za koji će se prikazati red vožnje. Angular 7 ima opciju da se prilikom inicijalizacije komponente podesi dinamički prikaz njenih delova. Tako je u ovom slučaju podešen prikaz dostupnih linija. Pri inicijalizaciji komponente klijentska strana traži od servera sve linije. Kada dobije spisak linija prikazuje ih i daje korisniku da izabere jednu liniju koja ga interesuje. Red vožnje će se prikazati u odnosu na izabranu liniju i dan u nedelji. Za prikaz stanica na mapi bitno je izabrati liniju (Slika 3.5.).



Slika 3.5. Prikaz stanica na mapi

Za mapu korišćena je *AgmMap* komponenta koja prikazuje Google mape. Da bi se mapa prikazala u okviru komponente potrebno je podesiti visinu i širinu prikaza mape kao u Listingu 3.4.

```
@Component({
      selector: 'app-map',
      templateUrl: './map.component.html',
      styleUrls: ['./map.component.'],
      styles: [ 'agm-map {height: 500px; width: 700px}']
})
```

Listing 3.4. Podešavanje visine i širine mape

Uz podešavanje veličine mape, moguće je podesiti deo mape koji će se prikazati na početku, početno uveličanje. Na mapi se mogu prikazati markeri i iscrtati linije. Stanice se prikazuju kao markeri. Svaki marker je dobio x i y koordinatu stanice i ikonicu. Kada se traži prikaz stanica, na server se šalje informacija o liniji. Server u bazi pronalazi sve stanice na koje staje linija koju je korisnik izabrao i vraća spisak linija (Listing 3.5.). Na mapi je moguće ukloniti prikazane stanice i izabrati novu liniju čije će se stanice prikazati.

```
// GET: api/Stanicas/5
[AllowAnonymous]
[ResponseType(typeof(string))]
[Route("GetStanica/{linijaBroj}")]

public IHttpActionResult GetStanica(string linijaBroj)
{

      List<Linija> sveLinije =
      Db.Linija.GetAll().ToList();

      Linija izabranaLinija = null;

      foreach(var l in sveLinije)
      {
            if(l.RedniBroj == linijaBroj)
            {
                  izabranaLinija = l;
                  break;
            }
      }


      if (izabranaLinija == null)
      {
            return NotFound();
      }


      List<Kordinate> listaKordinata = new
      List<Kordinate>();
```

```
        foreach(var stanica in izabranaLinija.Stanice)
        {
              Kordinate k = new Kordinate() { x =
              stanica.X, y = stanica.Y, name =
              stanica.Naziv };

              listaKordinata.Add(k);
        }

        return Ok(listaKordinata);
}
```

Listing 3.5. Akcija na serveru koja vraća listu stanica

Prilikom popunjavanja baze podataka, Entity Framework je sam popunio virtualne kolekciju stanica u klasi Linija (Listing 3.6.). Linija i stanica su u relaciji više na prema više. Entity Framework je prepoznao relaciju i automatski je popunjavao kolekcije u pomenutim klasama. Ovo u mnogome olakšava dalji rad sa čitanjem podataka iz baze. Nije potrebno pisati glomazne upite i raditi spajanje tabela da bi se iščitali podaci.

```
public class Linija
{
        public int Id { get; set; }
        public string RedniBroj { get; set; }
        public virtual ICollection<Stanica> Stanice { get;
        set; }

        public virtual ICollection<RedVoznje> RedoviVoznje
        { get; set; }

        public virtual ICollection<Vozilo> Vozila { get;
        set; }

        public Linija() { }
}
```

Listing 3.6. Klasa *Linija*

Praćenje lokacije autobusa rešeno je pomoću *web socketa*. *Web socketi* su dizajnirani za prenos dokumenta, više detalja na internet stranici [6]. U pitanju je dvosmerna komunikacija, samo u jednom smeru u jednom trenutku. Slanje podataka sa servera klijentu moguće je bez prethodnog

slanja zahteva klijenta ka serveru. Primer su razna obaveštenja na društvenim mrežama. Klijent nije inicirao komunikaciju sa serverom, ali je primio poruku kada se desilo obaveštenje koje mu je namenjeno. *Web socket* se razlikuje od HTTP (*Hypertext Transfer Protocol*) protokola ali je postignuta kompaktibilnost tako sto *Web socket* koristi HTTP zaglavlje. Kod *Web socketa* komunikacija se vrši putem TCP (*Transmission Control Protocol*) protokola.

U projektu napravljena je akcija na kontroleru u serveru, koja prima parametar oznaku linije. Server traži iz baze sve stanice za liniju koja mu je stigla. Kada ih pronađe, šalje ih klasi *LokacijaVozilaHub*. Klasa je nasledila klasu *Hub*, dokumentacija klase [5], koja nam omogućava da definišemo metode na serveru koje će se pozivati sa klijenta. U određenom vremenskom intervalu pozvana metoda (Listing 3.7.) se izvršava i server šalje klijentu trenutne koordinate autobusa.

```
private void OnTimedEvent(object source,
ElapsedEventArgs e)
{
      StringBuilder busData = new StringBuilder("");

      if (stanice != null)
      {

            if (stanicaBrojac >= stanice.Count)
            {
                  stanicaBrojac = 0;
            }
            double[] niz = { stanice[stanicaBrojac].X,
stanice[stanicaBrojac].Y };
            Clients.All.getBusData(niz);
            stanicaBrojac++;
      }
}
```

Listing 3.7. Metoda u klasi *LokacijaVozilaHub*

Metoda getBusData je definisana na serveru i nju poziva klijentska strana. Parametar joj je niz koji sadrži x i y kordinate. Na klijentskoj strani, pozvana je getBusData meotda, i postavljen joj je niz brojeva kao parametar koji očekuje od servera (Listing 3.8.).

```
public Location(): Observable<number[]> {

 return observable.create((observer) =>{

   this.proxy.on('getBusData', (data: number[]) => {
         console.log('received notification: ' + data);
         observer.next(data);
   });
 });
}

public StopTimer() {
   this.proxy.invoke("StopLocationServerUpdates");
}

public StartTimer() {
   this.proxy.invoke("StartLocationServerUpdates");
}
```

Listing 3.8. Metode na klijentu koje pozivaju serverske metoda

Metode StartTimer i SopTimer pozivaju metode na serveru, StartLocationServerUpdates i StopLocationServerUpdates koje pokreću i zaustavljaju tajmer za Hub.

Za plaćanje preko PayPal-a napravljena je posebna komponenta. Komponenta se prikazuje, na stranici za kupovinu karte, tek kada je obavljena kupovina karte. Cela komponenta se prikazje kao PayPal (Slika 3.6.) dugme koje dalje otvara prozor za plaćanje.

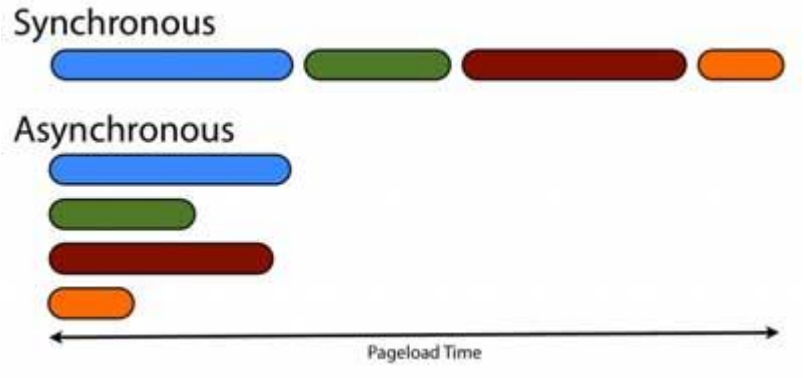
Slika 3.6. *PayPal* komponenta

U PayPal komponentu stiže cena karte koja je kupljena i za koju je potrebno izvršiti plaćanje. U prozoru za PayPal plaćanje koji se prikazuje, prvo je potrebno izvršiti prijavu na probni nalog. Sa tog naloga se vrši plaćanje i moguće je videti stanje novca nakon plaćanja. Kada se obavi plaćanje, PayPal komponenta vraća broj transakcije. Serveru se šalje taj broj koji se dalje čuva uz odgovarajuću kartu u bazi podataka.

4. NODE.JS - SERVERSKA STRANA APLIKACIJE

Ideja prebacivanja serverske strane aplikacije iz C# programskog jezika u Node.js zasnovana je na stvaranju kompletnog sistema aplikacije sa istim programskim jeziku u osnovi. Node.js se koristi za razvoj veb aplikacija visokih preformansi. U osnovi je JavaScrip okruženje u kojem se izvršavaju programi serverske strane u realnom vremenu. JavaScript podržava *first-class* funkcije i *closure*, što je Node.js preuzeo i donosi mu velike prednosti. *First-class* je princip rada sa funkcijama kao sa ostalim promenljivim. Funkcije je moguće proslediti drugim funkcijama kao argument, funcija može vratiti drugu funkciju kao povratnu vrednost i mogućnost dodeljivanja funkcije nekoj promenljivoj. *Closure* je koncept koji dozvoljava čuvanje referenci na lokalne promenljive i njihovo dalje korišćenje nakon završenog izvršavanja bloka u kojem su promenljive deklarisane. To se radi pomoću callback funkcija. Nakon izvršavanja callback funkcije moguće je koristiti njene promenljive u funkciji koja ju je pozvala.

Node.js ima jednu nit koja obrađuje sve pristigle zahteve. On koristi asinhroni i neblokirajući režim rada. Kod sinhronog izvršavanja, operacija može da krene izvršavanje samo ako je prethodna operacija završila svoj rad. Kod asinhronog izvršavanja kakvo je kod Node.js, operacije ne čekaju jedna drugu na izvršavanje nego mogu istovremeno da se izvršavaju. Vreme izvršavanja svake operacije posebno ostaje isto, ali ukupno vreme izvršavanja svih operacija se drastično smanjuje (Slika 4.1.).

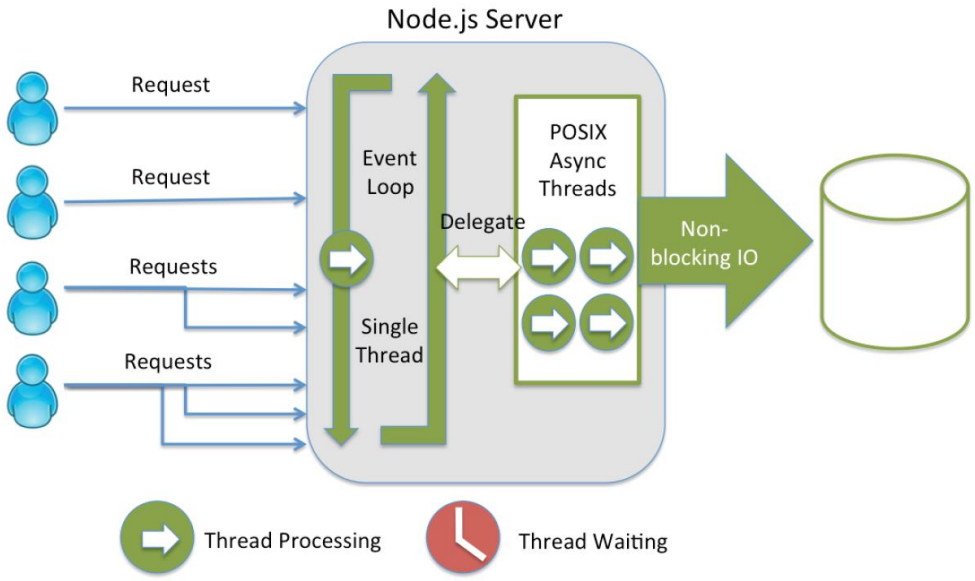


Slika 4.1. Dužina izvršavanja sinhronih i asinhronih operacija

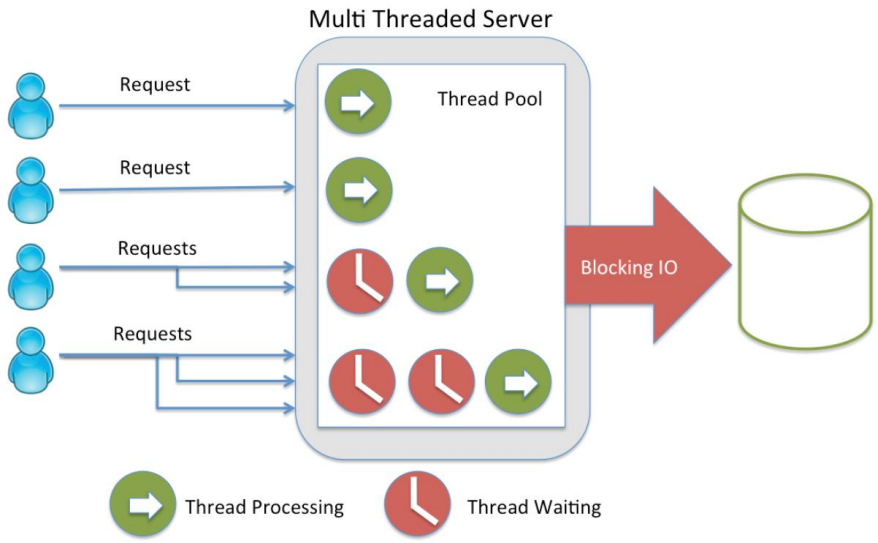
Takođe Node.js je projektovan da radi bez blokirajućih operacija. To znači da ni jedna operacija neće zaustaviti izvršavanje programa da bi se sačekalo na njen rezultat. Ove osobine Node.js-a dozvoljavaju mu visok nivo skalabilnosti čak i ako on u osnovi koristi jednu nit. Postoje Api koji na lak način omogućavaju rad sa više procesa i njihovu međusobnu komunikaciju. Kod desktop aplikacija se sinhrone i blokirajuće operacije izvršavaju dosta brzo, ali posto se Node.js koristi za rad sa veb aplikacijama onda je izuzetno bitno brzo vratiti odgovor klijentu. Veb u potpunosti zavisi od podataka koje dobija kao odgovor od servera. Na serverskoj strani se podaci obično čitaju iz baze podataka, lokalnih fajlova ili se dinamički generišu. Kada bi svaka funkcija bila blokirajuća i sinhrona, klijent bi predugo čekao na odgovor.

4.1 Poređenje preformansi Node.js I ASP.NET

Glavna razlika između Node.js i ASP.NET okruženja je njihov način obrade pristiglih zahteva. Node.js koristi asinhroni način obrade dok ASP.NET daje mogućnost izbora. Na sledeće dve slike 4.1.1 i 4.1.2 prikazane su razlike između sinhronih i asinhronih načina rada sa serverskih strana za veb aplikaciju.



Slika 4.1.1 Asinhron način rada



Slika 4.1.2 Sinhroni način rada

Glavna razlika i velika prednost Node.js je u jednoj niti koja obrađuje pristigle zahteve. Red pristiglih zahteva, na slici nazvan Event Loop, obrađuje sve u jednom tredu. Node.js je dizajniran na taj način da bi smanjio opterećenje hardvera, najviše procesora i uštedeo njegovo vreme. Rukovanje sa više tredova i njihovo prebacivanje iz aktivnog u neaktivno stanje oduzima dosta vremena i procesorske snage koja bi trebala da bude dostupna ostatku aplikacije. Glavna razlika se primeti kada broj zahteva premaši broj slobodnih tredova, kod servera dizajniranog sa *thread pool* načinom obrade zahteva. Tada se stvara red čekanja, a procesor je već dovoljno opterećen rukovanjem sa tredovima. .Net 4.5 je pokušao da reši problem sinhronog rada ubacivanjem async i await operatora. Problemi nastaju jer .Net nije u potpunosti napravljen za asinhroni rad kao Node.js koji je od samog početka dizajniran sa idejom asinhronog funkcionisanja.

Samo vreme obrade zahteva je dosta slično. Oba okruženja pružaju dobre preformanse. U maloj prednosti je Node.js što se konkretnih testova tiče. Bitna stvar je da ni jedno okruženje nema negativnih odgovora po povećanju opterećenja.

4.2. Prelazak iz ASP.NET u NODE.js

Prebacivanje serverske strane u Node.js realizovano je uz ideju da se što manje izmena pravi na klijentskoj strani. Ideja je bila da klijentska strana bez izmena može da radi sa oba servera po potrebi. Organizacija

serverske strane u ASP.NET-u zasniva se na kontrolerima i akcijama u okviru kontrolera.

Express biblioteka pruža funkcije za razvoj veb aplikacije. Lak rad sa rutiranjem i HTTP zahtevima.

Funkcija express() eksportuje express modul i inicijalizuje express objekat (Listing 4.2.1.). On koristi *corse* i *body-parser* funkcije. Cors dozvoljava aplikaciji da njen port bude dostupan za komunikaciju sa drugim aplikacijama. Body-parser presreće svaki zahtev i parametre parsira u json format. U *postsRoute* se nalaze sve akcije, get, set, delete metode koje servis daje na korišćenje. Nastavak *api* dodat je da bi se izjednačile putanje koje se nalaze u ASP.NET serveru. Na kraju je zadat isti port kao na ASP.NET serveru i stavljen je u stanje slušanja zahteva.

```
const express = require('express');
const bodyParser = require('body-parser');
var cors = require('cors');
var mongoose = require("mongoose");

const app = express();

app.use(cors());
app.use(bodyParser.json());

const postsRoute = require('./routes/posts');

app.use('/api', postsRoute);
app.listen(52295, () => console.log('Server started on
port 52295'));
```

Listing 4.2.1 Korišćenje *express*, *body-pasers* i *corse* biblioteka

Korišćena je biblioteka *jsonwebtoken* za rad sa jwt tokenom. Podešavanje je urađeno na način da se autorizacija i provera korisničke uloge na klijentu obavljao kao i sa prethodnim serverskim rešenjem.

```
var jwt = require('jsonwebtoken');

//Login, prijava korisnika
router.post('/Account/login', (req, res) => {
    //console.log(req.body.username);

    User.findOne({ username: req.body.username }, (err,
user) => {
```

```
        if (err) {
            console.log(err);
        } else {


            if (user.password == req.body.password) {
                jwt.sign({ user: user.username, role:
user.role }, 'secretkey', (err, token) => {
                    res.json({
                        token
                    });
                });
            } else {
                res.send(404);
            }
        }
    });
});
```

Listing 4.2.1 Korišćenje jwt tokena i *Login* akcija

U Listingu 4.2.1 iskorišćena je *jsonwebtoken* biblioteka, *jwt.sign* funkcija vraća token kao string. Token je zapakovan u jeson format i poslat je kao odgovor klijentu na akciju logovanja.

U istom Listingu 4.2.1 je pozvana metoda nad Mongo bazom za pronalazak korisnika u bazi, metoda *findOne.* Prosleđeno joj je korisničko ime po kojem će pokušati da nađe korisnika u bazi. Ako ga uspešno pronađe, korisničko ime i njegova uloga se koriste u generisanju tokena, *({ user: user.username, role: user.role }.*

4.3 MongoDB i Node.js

Za konekciju i rad sa Mongo bazom podataka korišćena je *mongoose* biblioteka. Pre konektovanja sa Mongo bazom potrebno je pokrenuti mongo servis. Konekcija sa bazom prikazana je u Listingu 4.3.1. Sve što je potrebno da se importuje moongose biblioteka, definiše putanja do baze i iskoristi u *connect* funkciji.

```
var mongoose = require("mongoose");

const url = 'mongodb://localhost:27017/JGSP';
mongoose.connect(url);
```

Listing 4.3.1. Konekcija sa MongoDB

Tabele, ili kako se u Mongo bazi nazivaju kolekcije, se kreiraju uz pomoć šema. Ako nemamo kolekciju u bazi ona će se kreirati kada dodamo novi dokument, kako su entiteti nazvani u MongoDB, pomoću *create* funkcije. Jedna od šema u sistemu prikazana je u Listingu 4.3.2. Šema predstavlja model koji mi želimo da čuvamo u bazi. Potrebno je zadati polja koja model poseduje i zadati naziv koji će kolekcija imati.

```
var mongoose = require("mongoose");

var userSchema = new mongoose.Schema({
    name: String,
    surname: String,
    username: String,
    password: String,
    confirmPassword: String,
    email: String,
    date: String,
    type: String,
    role: String,
    approved: Boolean
});

module.exports = mongoose.model("User", userSchema);
```

Listing 4.3.2. Šema MongoBD za User-a

5. ZAKLJUČAK

Tokom razvoja aplikacije više se obraćala pažnja na njenu funkcionalnost nego na izgled koji će biti dostupan klijentu. Potrebno je dizajn korisničkog dela aplikacije poboljšati i prilagoditi korisnicima.

Realno praćenje autobusa je odrađeno kao simulacija. Autobusi se pojavljuju samo na stanicama ne i na putu između njih. Za poboljšanje prikaza i realno korišćenje potrebno je povezati serversku stranu aplikacije sa GPS predajnikom koji bi se nalazio u autobusu i očitavati tačne koordinate autobusa.

PayPal komponenta je naknadno dodata u projekat pa je potrebno dodatno je prilagoditi funkcionalnosti projekta. Kupovina karte u sistemu je izvršena i karta je sačuvana u bazi podataka, tek tada se pojavljuje opcija za plaćanje putem PayPal-a. Potpuno ispravna funkcionalnost bi bila da se u toku kupovine karte prođe kroz plaćanje putem PayPal-a.

Projekat je dorađen, cela serverska strana prebačena je iz C# i .NET tehnologije u Node.js framework. Razlog za prebacivanje je ideja da klijentska i serverska strana budu u istom programskom jeziku (JavaScript).

LITERATURA

[1] C# 6 I .NET Core 1.0, Marko J. Price, Kompijuter Biblioteka, 2015

[2] C#, C# Guide, https://docs.microsoft.com /en-us/dotnet/csharp/

[3] Entity Framework , https://docs.microsoft.com

[5] Hubs, Use hubs in SignalR for ASP.NET CORE https://docs.microsoft.com /en-us/aspnet/core/signalr/hubs?view=aspnetcore-2.2

[6] Web socket , https://en.wikipedia.org /wiki/WebSocket

[7] Angular 7, https://angular.io /guide/ngmodules

BIOGRAFIJA

Marko Mijatović rođen 29.8.1996. godine u Beogradu. Osnovnu školu „Svetozar Marković Toza“ završio je 2011. godine. Gimnaziju „Isidora Sekulić“ u Novom Sadu završio je 2015. godine. Iste godine upisao se na Fakultet tehničkih nauka. Školske 2015/2016. godine upisao se na smer Primenjeno softversko inženjerstvo. Položio je sve ispite predviđene planom i programom.

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, **RBR**:	
Identifikacioni broj, **IBR**:	PR21-2015
Tip dokumentacije, **TD**:	monografska publikacija
Tip zapisa, **TZ**:	tekstualni štampani dokument
Vrsta rada, **VR**:	diplomski rad
Autor, **AU**:	Marko Mijatović
Mentor, **MN**:	prof.dr Branko Milosavljević
Naslov rada, **NR**:	Aplikacija za digitalizaciju gradskog prevoza
Jezik publikacije, **JP**:	Srpski
Jezik izvoda, **JI**:	srpski / engleski
Zemlja publikovanja, **ZP**:	Srbija
Uže geografsko područje, **UGP**:	Vojvodina
Godina, **GO**:	2019
Izdavač, **IZ**:	autorski reprint
Mesto i adresa, **MA**:	Novi Sad, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6
Fizički opis rada, **FO**:	br. Poglavlja 5 / stranica 29 / citata / tabela / slika 9/ grafikona / priloga
Naučna oblast, **NO**:	Informatika
Naučna disciplina, **ND**:	Mrežno bazirani sistemi
Predmetna odrednica / ključne reči, **PO**:	
UDK	
Čuva se, **ČU**:	Biblioteka Fakulteta tehničkih nauka, Trg Dositeja Obradovića 6, Novi Sad
Važna napomena, **VN**:	
Izvod, **IZ**:	U radu je opisano projektovanje i razvijanje aplikacije za digitalizaciju gradskog. Opisan je i proces prebacivanja aplikacije u različite programske tehnologije.
Datum prihvatanja teme, **DP**:	
Datum odbrane, **DO**:	
Članovi komisije, **KO**:	
predsednik	prof. dr Miroslav Zarić, vanr. prof., FTN Novi Sad
član	prof. dr Goran Sladić, vanr. prof., FTN Novi Sad
mentor	Prof. dr Branko Milosavljević, red. prof., FTN Novi Sad
	Potpis mentora

KEY WORDS DOCUMENTATION

Accession number, **ANO**:	
Identification number, **INO**:	PR21-2015
Document type, **DT**:	monographic publication
Type of record, **TR**:	textual material
Contents code, **CC**:	BSc thesis
Author, **AU**:	Marko Mijatović
Mentor, **MN**:	Branko Milosavljević, PhD, full. prof., FTN Novi Sad
Title, **TI**:	Application for digitizing metropolitan public transport
Language of text, **LT**:	serbian
Language of abstract, **LA**:	serbian / english
Country of publication, **CP**:	Serbia
Locality of publication, **LP**:	Vojvodina
Publication year, **PY**:	2019
Publisher, **PB**:	author's reprint
Publication place, **PP**:	Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6
Physical description, **PD**:	5 / 29 / citata / tabela / slika 9/ grafikona / priloga
Scientific field, **SF**:	Computer science
Scientific discipline, **ND**:	Net-centric computing
Subject / Keywords, **S/KW**:	
UDC	
Holding data, **HD**:	Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad
Note, **N**:	
Abstract, **AB**:	The paper describes the design and development of urban transport digitization applications. The process of switching applications in different software technologies is also described.
Accepted by sci. board on, **ASB**:	
Defended on, **DE**:	
Defense board, **DB**:	
president	Miroslav Zarić, PhD, assoc. prof., FTN Novi Sad
member	Goran Sladić, PhD, assoc. prof., FTN Novi Sad
mentor	Branko Milosavljević, PhD, full. prof., FTN Novi Sad
	Mentor's signature