Programsko inženjerstvo

Ak. god. 2020./2021.

Parkiraj Me

Dokumentacija, Rev. 1.0

Grupa: NDB

Voditelj: Daniel Ranogajec

Datum predaje: *14. 1. 2021.*

Nastavnik: Nikolina Frid

Sadržaj

1. Dnevnik promjena dokumentacije

Rev.	Opis promjene/dodatka	Autori	Datum
0.1	Napravljen predložak	Jovanović	13.10.2020
0.2	Opis projektnog zadatka	Jovanović Marušić Fabijanić	16.10.2020
0.3	Specifikacije programske potpore, funkcionalni zahtjevi, obrasci uporabe i dijagrami obarazaca uporabe	Jovanović Erceg Fabijianić Vuica	17.10.2020
0.3.1	Uvod u sekvencijske dijagrame	Jovanović	19.10.2020
0.3.2	Sekvencijski dijagrami i njihovi opisi	Erceg	23.10.2020
0.4	Baza podataka	Briševac Marušić Ranogajec	24.10.2020
0.4.1	Dijagram razreda	Briševac Marušić	24.10.2020
0.5	Ostali zahtjevi	Jovanović	31.10.2020
0.6	Prepravljanje opisa projektnog zadatka	Fabijanić	9.11.2020.
0.7	Prepravljanje specifikacija programske potpore	Vuica	10.11.2020.
0.8	Revizija pred predaju	Marušič Jovanović Fabijanić	13.11.2020.
1.0	Verzija samo s bitnim dijelovima za 1. ciklus		13.11.2020
1.1	Ispitivanje programskog rješenja	Jovanović Marušić	14.12.2020
1.2	Korištene tehnologije i alati	Jovanović Ranogajec	14.12.2020

Rev.	Opis promjene/dodatka	Autori	Datum
1.3	Revizija nakon predaje	Jovanović Marušić Vuica Fabijanić Briševac	17.12.2020
1.4	Dodan dijagram aktivnosti Dodan dijagram stanja	Fabijanić	11.01.2021
1.5	Dijagram razmještaja	Erceg	12.01.2021
1.6	Dijagram komponenti	Briševac	13.01.2021
1.7	Zaključak i budući rad	Jovanović	13.01.2021
1.8	Upute za puštanje u pogon	Ranogajec	14.01.2021
1.9	Revizija pred predaju	Fabijanić Jovanović Marušić	14.01.2021.
2.0	Verzija samo s bitnim dijelovima za 2. ciklus		14.1.2021

2. Opis projektnog zadatka

Cilj ovog projekta je razviti programsku podršku za stvaranje web aplikacije „Parkiraj Me“ koja korisnicima omogućuje praćenje stanja parkirališnih mjesta u gradu te njihovu rezervaciju. Aplikacija će objedinjavati parkirališne površine diljem Zagreba svih tvrtki zainteresiranih za ponudu svojih površina putem aplikacije. Ovakvim rješenjem bi se uvelike olakšalo nalaženje parkirališnih mjesta u gusto naseljenim urbanim sredinama među koje se svrstaje i grad Zagreb.

Prilikom pokretanja aplikacije prikazuje se karta s ucrtanim parkirališnim površinama koje se nalaze u blizini. Pomoću pametnih kamera se omogućuje praćenje zauzeća pojedinih parkirališnih mjesta.

Sustav je povezan sa Google Maps uslugom, tj. pomoću toga će se korisnicima prikazivati karta i lokacije parkinga na njoj. Vezano za tu uslugu implementirana je i opcija prijedloga parkirališnih površina na temelju trenutne lokacije korisnika. Kod davanja prijedloga u situaciji kada je na geografski najbližoj lokaciji slobodan mali broj mjesta, prednost je dana sljedećoj najbližoj lokaciji s većim brojem slobodnih mjesta, jer se može dogoditi da kad vozač stigne do parkinga više nema slobodnih mjesta.

Neregistriranom korisniku odabirom parkirališne površine prikazuju se informacije o statusu parkirališne površine:

- broj slobodnih mjesta
- udaljenost od parking lokacije

Svaki punoljetni neregistrirani korisnik ima opciju registracije. Moguće se registrirati kao privatni korisnik, tj. vozač ili kao tvrtka koja nudi svoja parking mjesta na svojim parkirališnim površinama.
Pri postupku registracije korisnici koji žele napraviti račun kao vozač unose slijedeće podatke:

- korisničko ime
- OIB
- Ime i prezime
- E-mail
- Registarska oznaka vozila
- Broj kreditne kartice
- Lozinku koju žele koristiti

Korisnici koji žele napraviti račun kao vlasnici parking površina unose slijedeće podatke:

- OIB tvrtke
- Ime tvrtke
- Adresu sjedišta tvrtke
- E-mail
- Lozinku koju žele koristiti

Što se tiče samih rezervacija postoje tri opcije:

- jednokratna za vremenski period kraći od 24 sata koja se mora obaviti barem 6h unaprijed
  - naplata u trenutku rezervacije
- ponavljajuća rezervacija koja mora trajati najmanje 1h i ponavljati se barem jednom tjedno tijekom mjesec dana
  - naplata svakih 30 dana
- trajna rezervacija mjesta (0-24h svaki dan na neodređeni period)

Naplata će se vršiti po principu da korisnici plaćaju direktnom uplatom aplikaciji što je ostvareno pomoću vanjskog servisa naplate. Dalje se te uplate prosljeđuju vlasnicima parking lokacija.

Vozač ima slijedeće mogućnosti prilikom korištenja aplikacije:

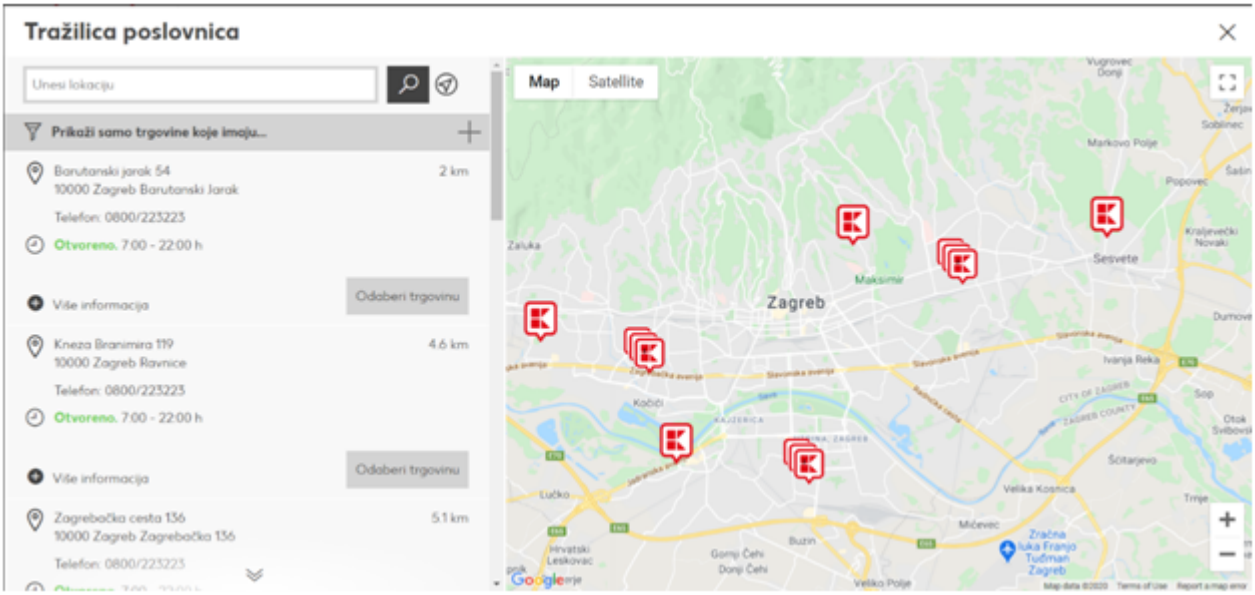
- rezervacija parking mjesta
- pregled i izmjena:
  - osobnih podataka
  - registriranih vozila
  - aktivnih rezervacija

Tvrtkama se omogućuju slijedeće mogućnosti prilikom korištenja aplikacije:

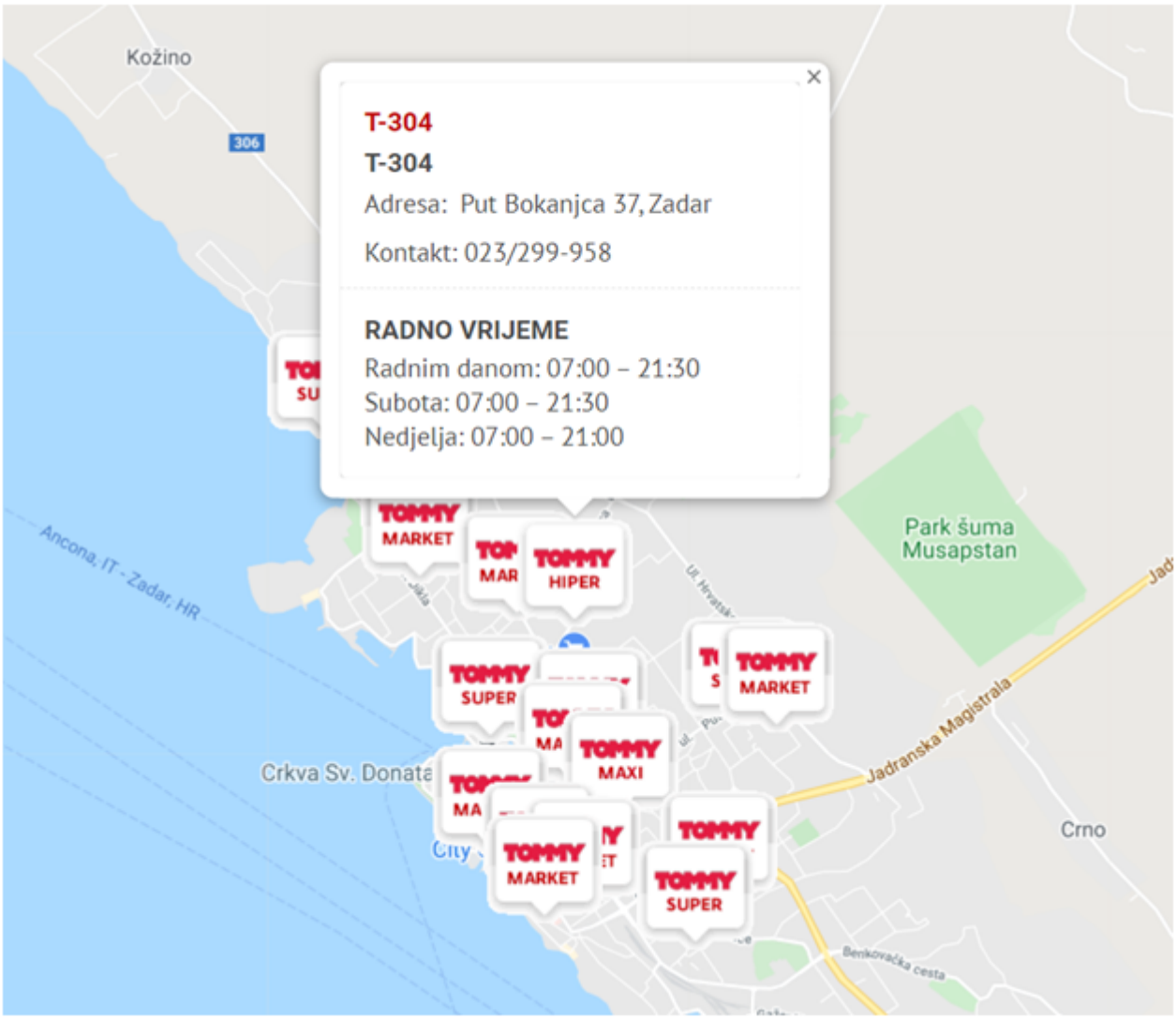
- pregled i izmjena:
  - osobnih podataka o računu
  - prijavljenih parking površina
  - postavljenih cijena rezervacija

Administrator je zadnji tip korisnika. Administrator sustava ima najveće ovlasti te može upravljati i računima klijenata i računima tvrtki. Ima mogućnost i brisanja tih računa. Prilikom brisanja računa klijenata, ako ima neiskorištenih rezervacija uplata bi se refundirala za preostalo (neiskorišteno) vrijeme. Prilikom brisanja računa tvrtki obrisale bi se i sve njihove parkirališne površine, a sve preostale neiskorištene rezervacije bi se otkazale i refundirale.

Slična implementacija lokacija na karti i prikazivanja nekih informacija o njima prikazana je na donjim slikama.



Slika 2.1: Odabir poslovnica tvrtke Kaufland, `https://www.kaufland.hr/usluge/poslovnica.html`



Slika 2.2: Odabir poslovnica tvrtke Tommy, `https://tommy.hr/hr/prodajna-mjesta`

3. Specifikacija programske potpore

3.1 Funkcionalni zahtjevi

Dionici:

1. Tvrtke
2. Korisnici
   (a) Vozač
3. Administrator
4. Razvojni tim

Aktori i njihovi funkcionalni zahtjevi:

1. Tvrtka (inicijator) može:
   (a) registrirati se u aplikaciji unoseći OIB, ime, adresu sjedišta, adresu e-pošte
   (b) prijaviti, pregledati, urediti i obrisati parkirališnu površinu
2. Neregistrirani korisnik (inicijator) može:
   (a) pregledati parkirališne površine
   (b) stvoriti korisnički račun unoseći OIB, ime, prezime, adresu e-ppošte, broj registracije svojeg automobila i broj kreditne kartice
   (c) dobiti ovlasti registriranog korisnika
3. Vozač (inicijator) može:
   (a) pregledati parkirališne površine
   (b) pregledati i promijeniti osobne podatke
   (c) pregledati, dodati, obrisati registarsku oznaku automobila u aplikaciju
   (d) dobiti prijedlog najbližih parkirališnih površina za rezervaciju
   (e) rezervirati parkirališno mjesto
   (f) platiti rezervaciju

(g) pregledati, urediti, obrisati rezervacije

4. Administrator (inicijator) može:

   (a) vidjeti popis odabranih korisnika
   (b) urediti i obrisati odabrane korisnike
   (c) vidjeti popis odabranih tvrtki
   (d) urediti i obrisati odabrane tvrtke

5. Baza podataka (inicijator) može:

   (a) pohranjuje sve podatke o korisnicima
   (b) pohranjuje sve podatke o tvrtkama

3.1.1 Obrasci uporabe

UC1 - Pregled parkirališnih površina

- **Glavni sudionik:** Vozač, neregistrirani korisnik
- **Cilj:** Pregledati parkirališne površine
- **Sudionici:** Baza podataka, Google maps
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
  1. Karta je prikazana prilikom otvaranja aplikacije
  2. Korisnik na karti odabire parkirališnu površinu
  3. Prikazuju se informacije o odabranoj površini

UC2 - Registracija

- **Glavni sudionik:** Neregistrirani korisnik, Tvrtka
- **Cilj:** Stvoriti korisnički račun
- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju za registraciju (vozač ili tvrtka)
  2. Korisnik unosi potrebne korisničke podatke
- **Opis mogućih odstupanja:**

  2.a Odabir već zauzetog korisničkog imena i/ili e-pošte, unos korisničkog podataka u nedozvoljenom formatu ili pružanje neispravne e-pošte
  1. Sustav obavještava korisnika o neuspjelom upisu i vraća ga na stranicu za registraciju
  2. Korisnik mijenja potrebne podatke te završava unos ili odustaje od registracije

UC3 - Prijava u sustav

- **Glavni sudionik:** Vozač
- **Cilj:** Dobiti pristup korisničkom sučelju
- **Sudionici:** Baza podataka
- **Preduvjet:** Registracija
- **Opis osnovnog tijeka:**
  1. Unos korisničkog imena i lozinke
  2. Potvrda ispravnosti unesenih podataka

3. Pristup korisničkim funkcijama

- **Opis mogućih odstupanja:**

2.a Neispravno uneseni podatci

1. Sustav obavještava korisnika o neuspjeloj prijavi

UC4 - Pregled osobnih podataka

- **Glavni sudionik:** Vozač, Tvrtka
- **Cilj:** Pregledati osobne podatke
- **Sudionici:** Baza podataka
- **Preduvjet:** Korisnik je prijavljen
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju „osobni podatci“
  2. Aplikacija prikazuje osobne podatke korisnika

UC5 - Promjena osobnih podataka

- **Glavni sudionik:** Vozač, tvrtka
- **Cilj:** Promijeniti osobne podatke
- **Sudionici:** Baza podataka
- **Preduvjet:** Korisnik je prijavljen
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju „osobni podatci“
  2. Korisnik mijenja svoje osobne podatke
  3. Korisnik sprema promjene
  4. Baza podataka se ažurira
- **Opis mogućih odstupanja:**

2.a Korisnik promijeni svoje osobne podatke, ali je nešto neispravno uneseno

1. Sustav obavještava korisnika da je nemoguće izmjeniti podatke prilikom odabira „Spremi promjene“

UC6 - Registracija auta

- **Glavni sudionik:** Vozač
- **Cilj:** Registrirati automobil
- **Sudionici:** Baza podataka
- **Preduvjet:** Vozač je prijavljen
- **Opis osnovnog tijeka:**

1. Vozač odabire opciju „dodaj auto“
2. Vozač unosi potrebne podatke
3. Vozač prima obavijest o uspješnoj registraciji

- **Opis mogućih odstupanja:**

2.a Unesena neispravna/već registrirana registracija

1. Sustav obavijesti vozača da nije moguće registrirati auto, te ga vraća na stranicu za registraciju automobila
2. Vozač mijenja podatke ili odustaje

UC7 - Pregled registriranih automobila

- **Glavni sudionik:** Vozač
- **Cilj:** Pregledati registrirane automobile
- **Sudionici:** Baza podataka
- **Preduvjet:** Vozač je prijavljen
- **Opis osnovnog tijeka:**
  1. Vozač odabire opciju „pregledaj aute“
  2. Aplikacija prikazuje automobile koje je vozač prijašnje unio ako postoje

UC8 - Brisanje registriranih automobila

- **Glavni sudionik:** Vozač
- **Cilj:** Obrisati registrirane automobile
- **Sudionici:** Baza podataka
- **Preduvjet:** Vozač je prijavljen
- **Opis osnovnog tijeka:**
  1. Vozač odabere opciju „pregledaj aute“
  2. Vozač odabire auto koje želi obrisati i stisne „obriši“
  3. Baza podataka se ažurira

UC9 - Rezervacija parkirališnog mjesta

- **Glavni sudionik:** Vozač
- **Cilj:** Rezervirati parking mjesto
- **Sudionici:** Baza podataka, Google maps
- **Preduvjet:** Vozač je prijavljen
- **Opis osnovnog tijeka:**
  1. Vozač odabere opciju „Napravi rezervaciju“
  2. Sustav pronalazi vozaču najbližu parkirališnu površinu

3. Vozač odabire ponuđenu parkirališnu površinu ili sam na karti pronalazi drugu
4. Vozač odabire vozilo za koje želi napraviti rezervaciju
5. Vozač ispuni potrebne informacije za rezervaciju
6. Vozač odabire opciju „Potvrdi rezervaciju“
7. Sustav naplati rezervaciju

- **Opis mogućih odstupanja:**

3.a Na parkingu nema slobodnog mjesta

1. Sustav obavještava vozača da nema slobodnog mjesta i daje korisniku opciju mijenjanja datuma rezervacije

UC10 - Pregled aktivnih rezervacija

- **Glavni sudionik:** Vozač
- **Cilj:** Pregledati aktivne rezervacije
- **Sudionici:** Baza podataka
- **Preduvjet:** Vozač je prijavljen
- **Opis osnovnog tijeka:**
  1. Vozač odabire opciju „Pregledaj rezervacije“
  2. Sustav prikaže vozaču aktivne rezervacije ukoliko takve postoje

UC11 - Brisanje aktivnih rezervacija

- **Glavni sudionik:** Vozač
- **Cilj:** Obrisati rezervaciju
- **Sudionici:** Baza podataka
- **Preduvjet:** Vozač je prijavljen i napravio je barem jednu rezervaciju
- **Opis osnovnog tijeka:**
  1. Vozač odabire opciju „Pregledaj rezervacije“
  2. Sustav prikaže listu rezervacija vozača
  3. Vozač odabire opciju „Obriši rezervaciju“
  4. Rezervacija se briše iz baze podataka
  5. Vozaču se vraćaju novci ako ima neiskorištenih rezervacija
- **Opis mogućih odstupanja:**

4.a Greška prilikom refundiranja novca

1. Sustav obavještava vozača da je došlo do greške prilikom refundiranja novca na njegov račun, te se odustaje od brisanja rezervacije.

UC12 - Prijava parkirališne površine

- **Glavni sudionik:** Tvrtka
- **Cilj:** Prijaviti parking objekt
- **Sudionici:** Baza podataka
- **Preduvjet:** Tvrtka je prijavljena
- **Opis osnovnog tijeka:**
  1. Tvrtka odabire opciju „Dodaj parkirališnu površinu“
  2. Unos podataka o parkirališnoj površini
  3. Tvrtka unese sve podatke i odabire „Spremi“
  4. Ažurira se baza podataka
- **Opis mogućih odstupanja:**

  2.a Unesen neispravan/već prijavljen parking
  1. Sustav obavijesti korisnika da nije moguće prijaviti parking te ga vraća na stranicu za prijavu parkirališne površine
  2. Korisnik mijenja podatka ili odustaje

UC13 - Pregled prijavljenih parkirališnih površina

- **Glavni sudionik:** Tvrtka
- **Cilj:** Pregled prijavljenih parkirališnih površina
- **Sudionici:** Baza podataka
- **Preduvjet:** Tvrtka je prijavljena i postoji prijavljena parkirališna površina
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju „Upravljaj parkirališnim površinama“
  2. Otvara se lista prijavljenih parkirališnih površina

UC14 - Brisanje parkirališnih površina

- **Glavni sudionik:** Tvrtka
- **Cilj:** Obrisati parkirališnu površinu
- **Sudionici:** Baza podataka
- **Preduvjet:** - Tvrtka je prijavljena i ima barem jedan prijavljeni parkirališnu površinu
- **Opis osnovnog tijeka:**
  1. Korisnik odabire opciju „Upravljaj parkirališnim površinama“
  2. Sustav prikaže listu prijavljenih parkirališnih površina
  3. Korisnik briše željenu parkirališnu površinu

4. Ažurira se baza podataka

UC15 - Pregled vozača

- **Glavni sudionik:** Administrator
- **Cilj:** Vidjeti popis vozača
- **Sudionici:** Baza podataka
- **Preduvjet:** - Administrator prijavljen
- **Opis osnovnog tijeka:**
  1. Administrator odabire opciju „Popis vozača“
  2. Pojavi se lista vozača

UC16 - Uređivanje vozača

- **Glavni sudionik:** Administrator
- **Cilj:** Urediti vozače
- **Sudionici:** Baza podataka
- **Preduvjet:** Administrator prijavljen i postoji barem jedan korisnik
- **Opis osnovnog tijeka:**
  1. Administrator pronalazi željenog vozača i odabire opciju „Uredi“
  2. Sustav prikazuje podatke o odabranom vozaču
  3. Administrator upravlja njegovim podacima
  4. Ažurira se baza podataka

UC17 - Brisanje vozača

- **Glavni sudionik:** Admin
- **Cilj:** sudionik: Obrisati odabranog vozača
- **Sudionici:** Baza podataka
- **Preduvjet:** Admin prijavljen i postoji barem jedan korisnik
- **Opis osnovnog tijeka:**
  1. Administrator pronalazi željenog vozača i odabire opciju „Obriši“
  2. Vozač se obriše
  3. Ažurira se baza podataka

UC18 - Pregled tvrtki

- **Glavni sudionik:** Admin
- **Cilj:** Vidjeti popis odabranih tvrtki
- **Sudionici:** Baza podataka

- **Preduvjet:** Admin prijavljen
- **Opis osnovnog tijeka:**
    1. Administrator odabire opciju „Popis tvrtki“
    2. Pojavi se lista s odabranim tvrtkama

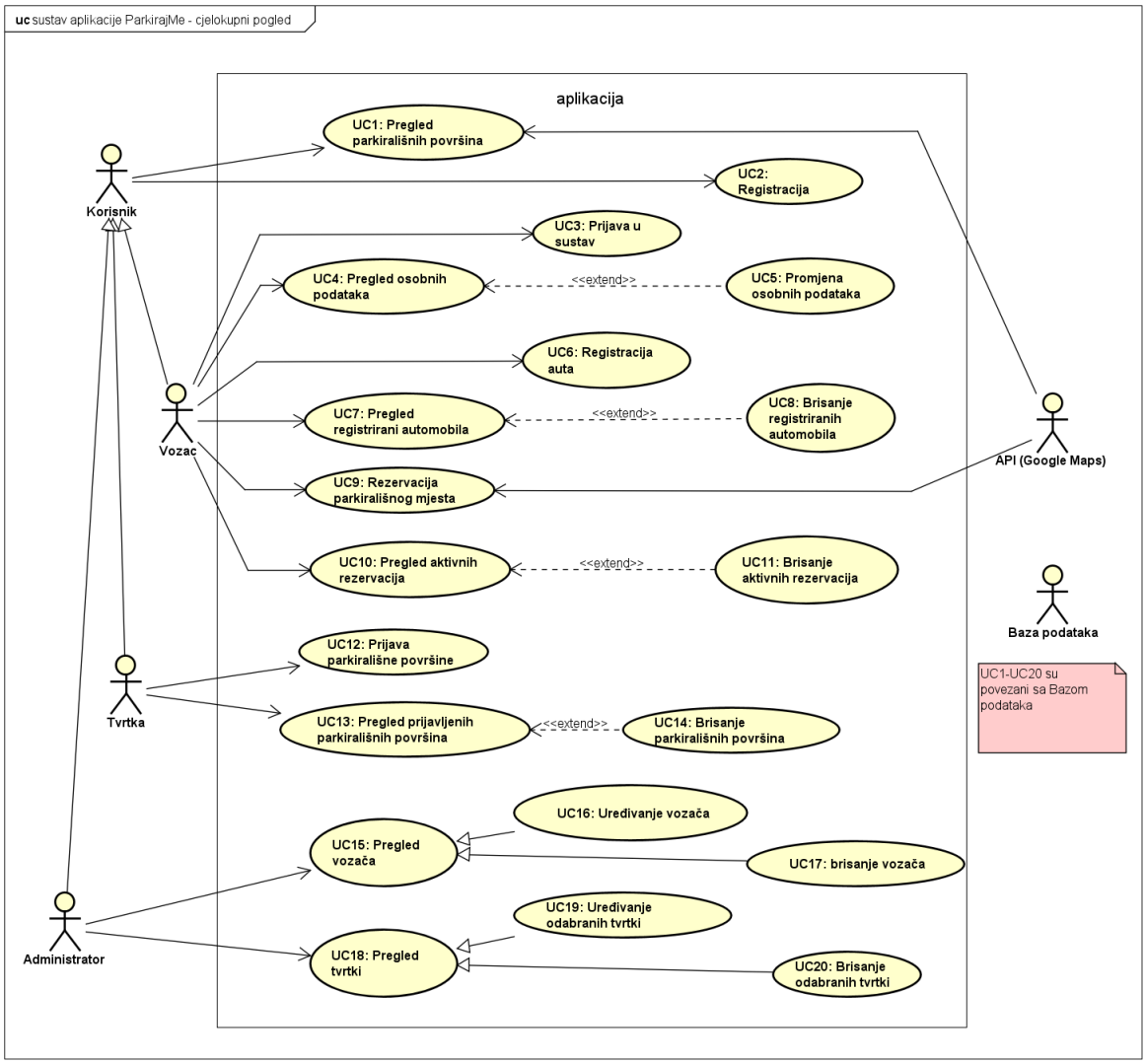
UC19 - Uređivanje odabranih tvrtki

- **Glavni sudionik:** Administrator
- **Cilj:** Urediti odabrane tvrtke
- **Sudionici:** Baza podataka
- **Preduvjet:** Administrator prijavljen i postoji barem jedna tvrtka
- **Opis osnovnog tijeka:**
    1. Administrator pronalazi željenu tvrtku i odabire opciju „Uredi“
    2. Sustav prikazuje podatke o odabranoj tvrtki
    3. Administrator upravlja njezinim podacima
    4. Ažurira se baza podataka

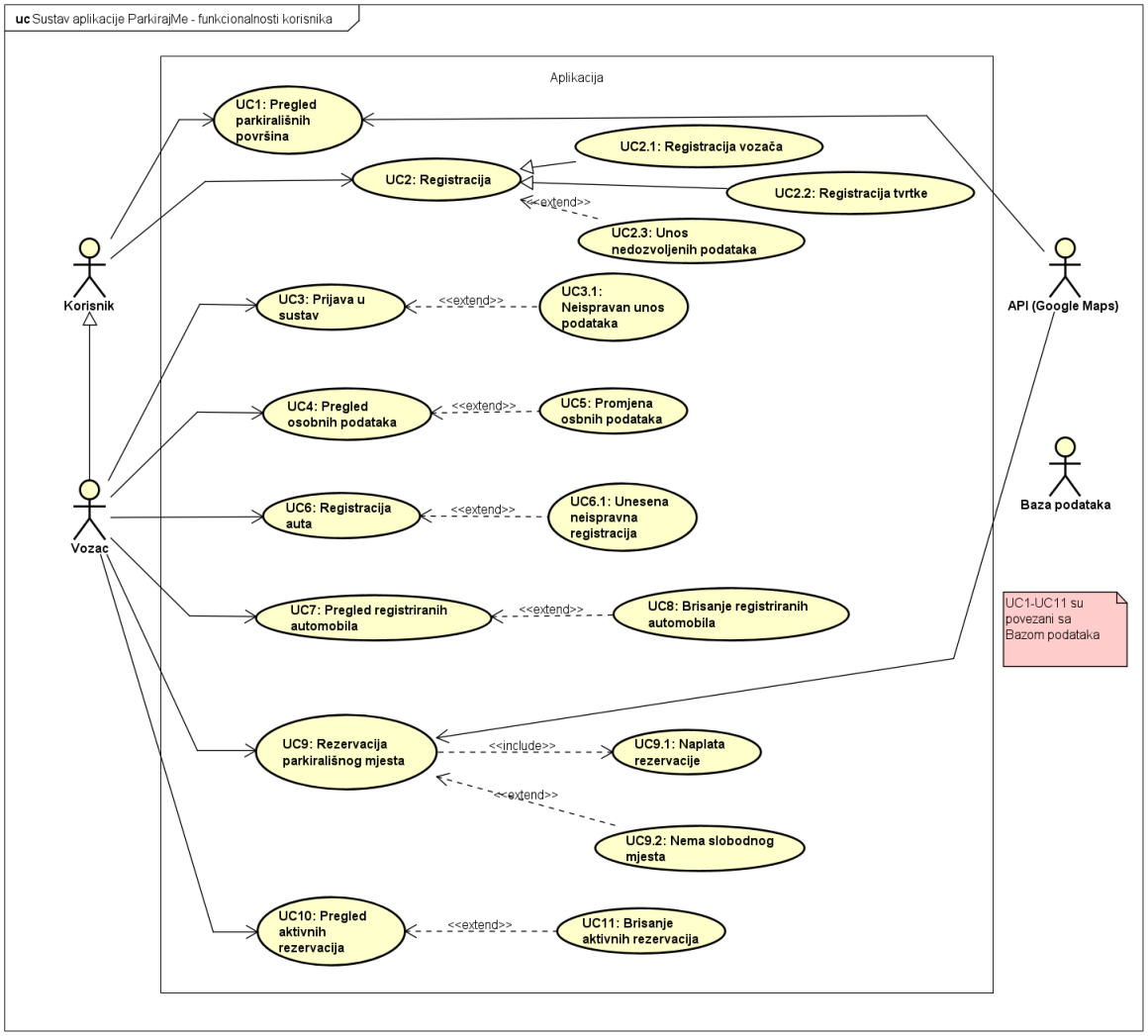
UC20 - Brisanje odabranih tvrtki

- **Glavni sudionik:** Administrator
- **Cilj:** Obrisati odabrane tvrtke
- **Sudionici:** Baza podataka
- **Preduvjet:** - Administrator je prijavljen i postoji barem jedan korisnik
- **Opis osnovnog tijeka:**
    1. Administrator odabire željenu tvrtku i odabire opciju „Obriši“
    2. Tvrtka se obriše
    3. Ažurira se baza podataka

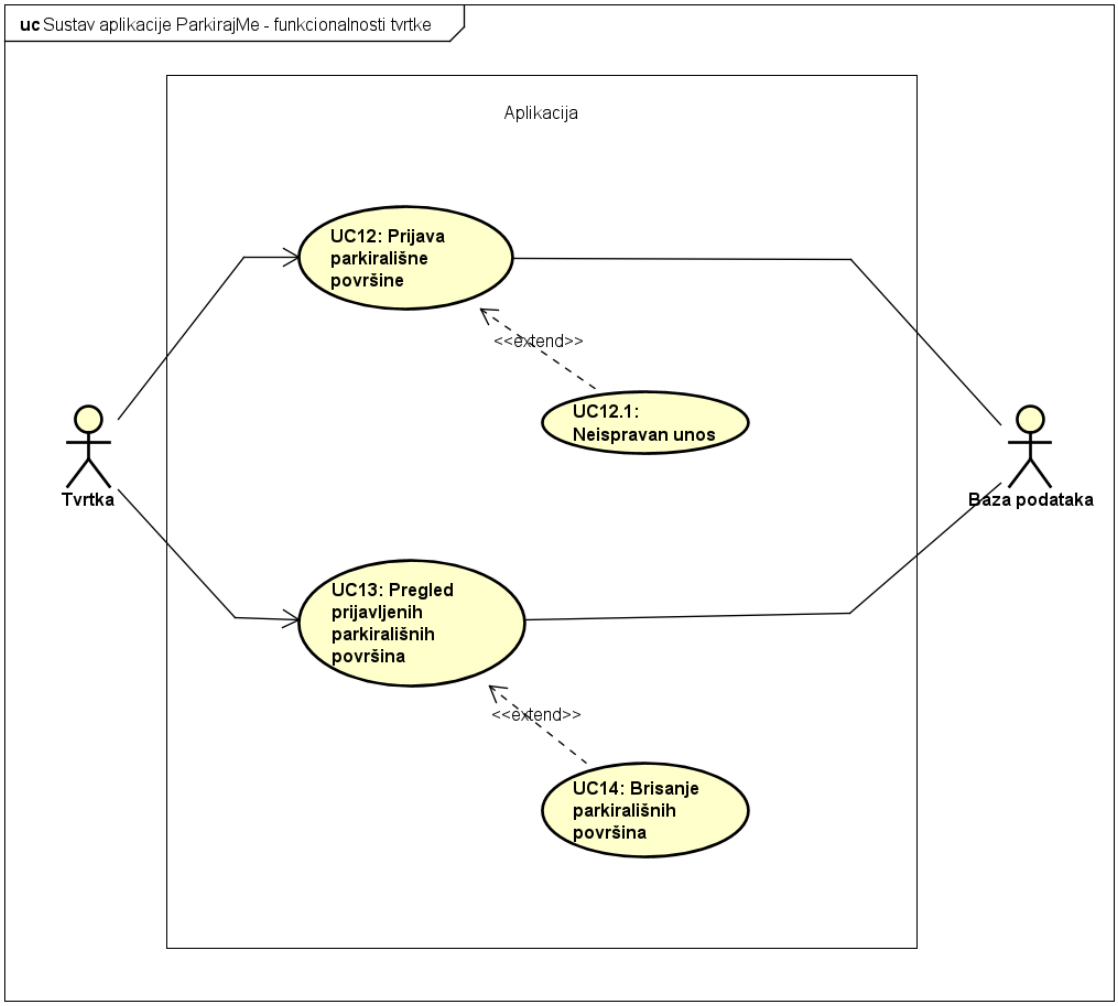
Dijagrami obrazaca uporabe



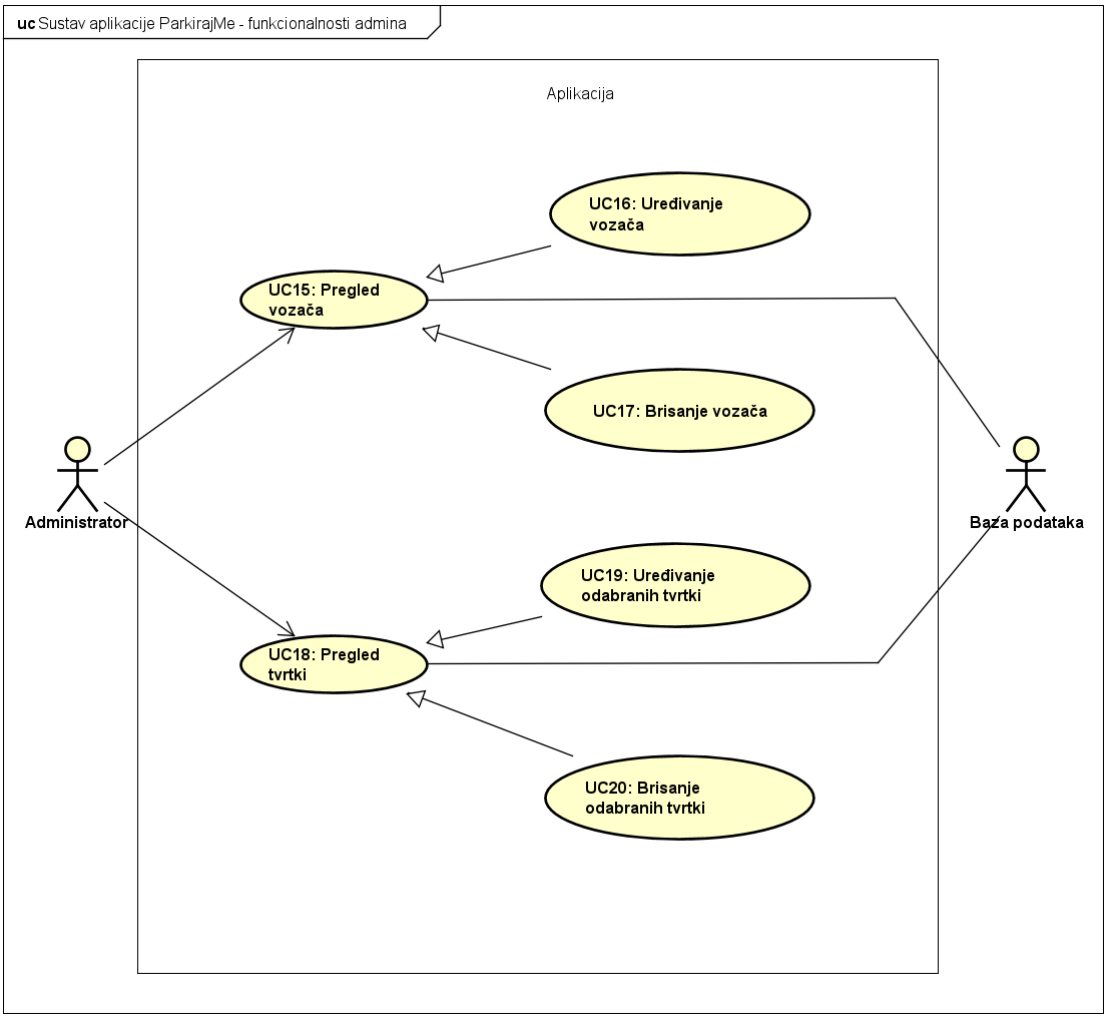
Slika 3.1: Cjelokupni pogled



Slika 3.2: Funkcionalnosti korisnika



Slika 3.3: Funkcionalnosti tvrtke

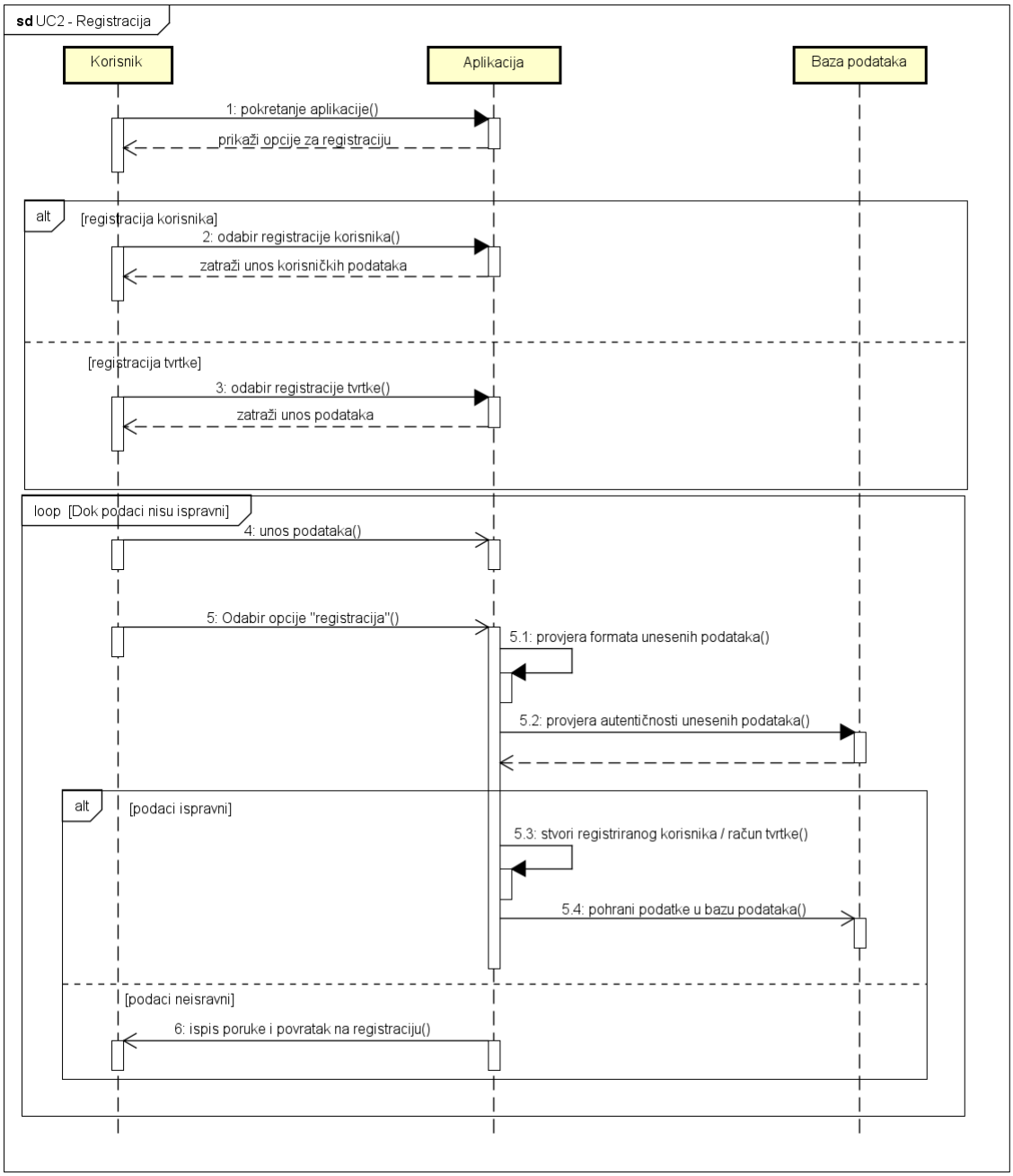


Slika 3.4: Funkcionalnosti admina

3.1.2 Sekvencijski dijagrami

Obrazac uporabe UC2 - Registracija

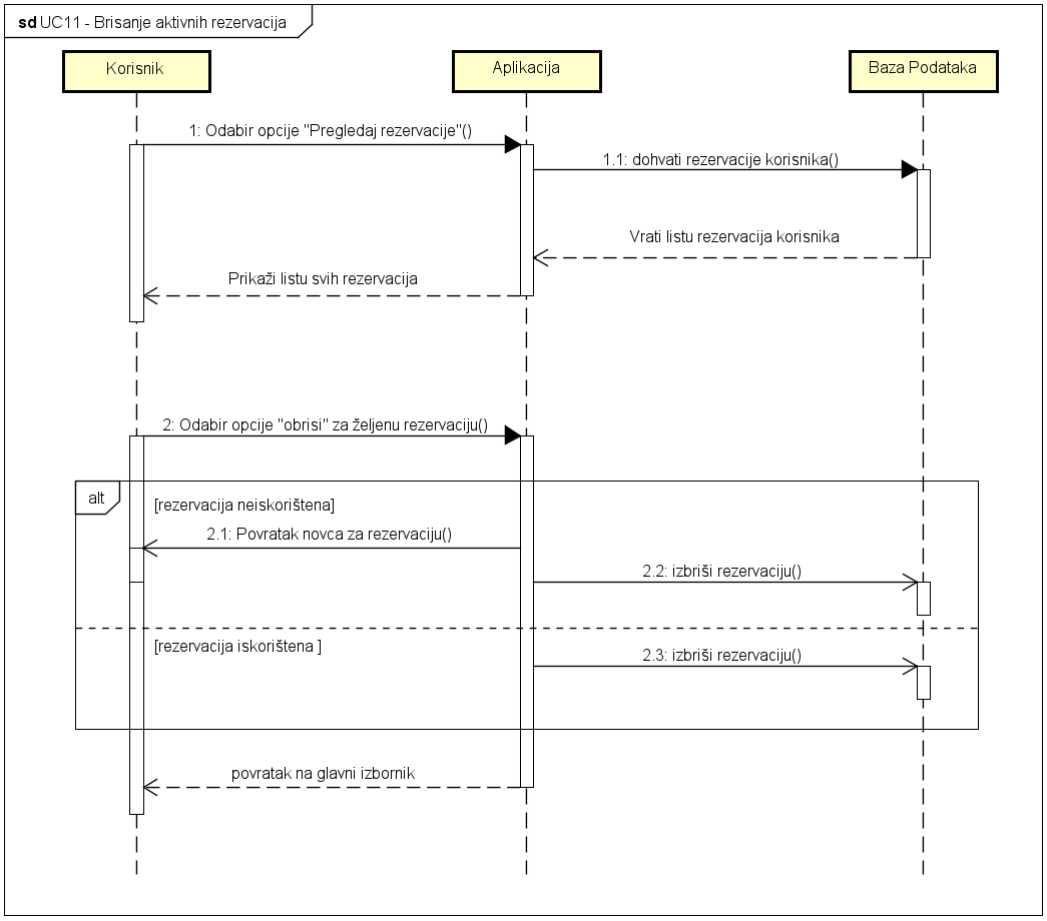
Korisnik bira jednu od dvije opcije registracije: ”Registriraj se kao korisnik” ili ”Registriraj se kao tvrtka”. Nakon odabira željene registracije aplikacija ga traži unos potrebnih podataka. Aplikacija provjerava jesu li sva polja popunjena i je li format unesenih podataka ispravan (broj znamenki OIB-a,format adrese e-pošte...). Nakon toga aplikacija pristupa bazi podataka i provjerava jedinstvenost unesenih podataka. Ako su podaci ispravni, ovisno o vrsti registracije, stvara se novi vozač ili korisnički račun tvrtke te se podaci pohranjuju u bazu podataka. Ako su uneseni podaci neispravni, korisnik prima obavijest i aplikacija ga vraća na glavni izbornik.



Slika 3.5: Sekvencijski dijagram - Registracija

Obrazac uporabe UC11 - Brisanje aktivnih rezervacija

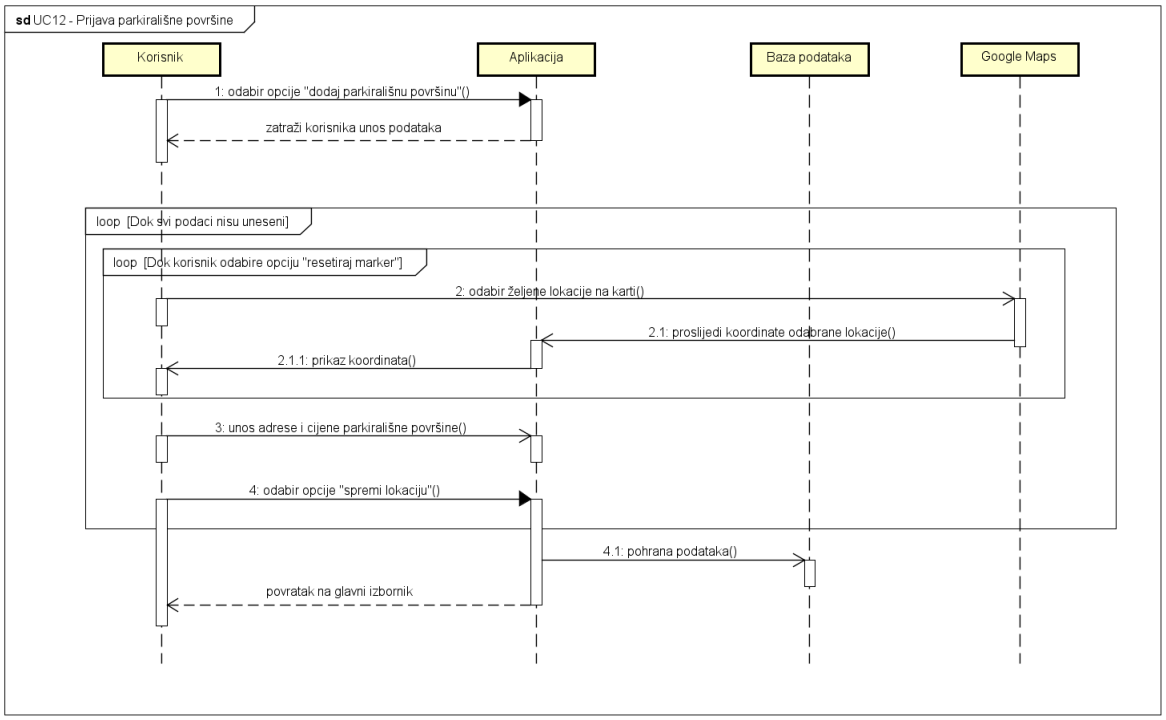
Vozač odabire opciju ”pregledaj rezervacije” nakon čega poslužitelj pristupa bazi podataka i dohvaća listu svih vozačevih rezervacija. Vozač odabire jednu rezervaciju koju želi obrisati i klikom na gumb ”obriši” rezervacije se briše. Ako rezervacija još nije iskorištena, aplikacija uplaćuje cijenu rezervacije na vozačev račun i briše ju iz baze podatatka. Ako je rezervacija već iskorištena, aplikacija ju uklanja iz baze podataka. Nakon brisanja rezervacije, korisnik se vraća na glavni izbornik.



Slika 3.6: Sekvencijski dijagram - Brisanje aktivnih rezervacija

Obrazac uporabe UC12 - Prijava parkirališne površine

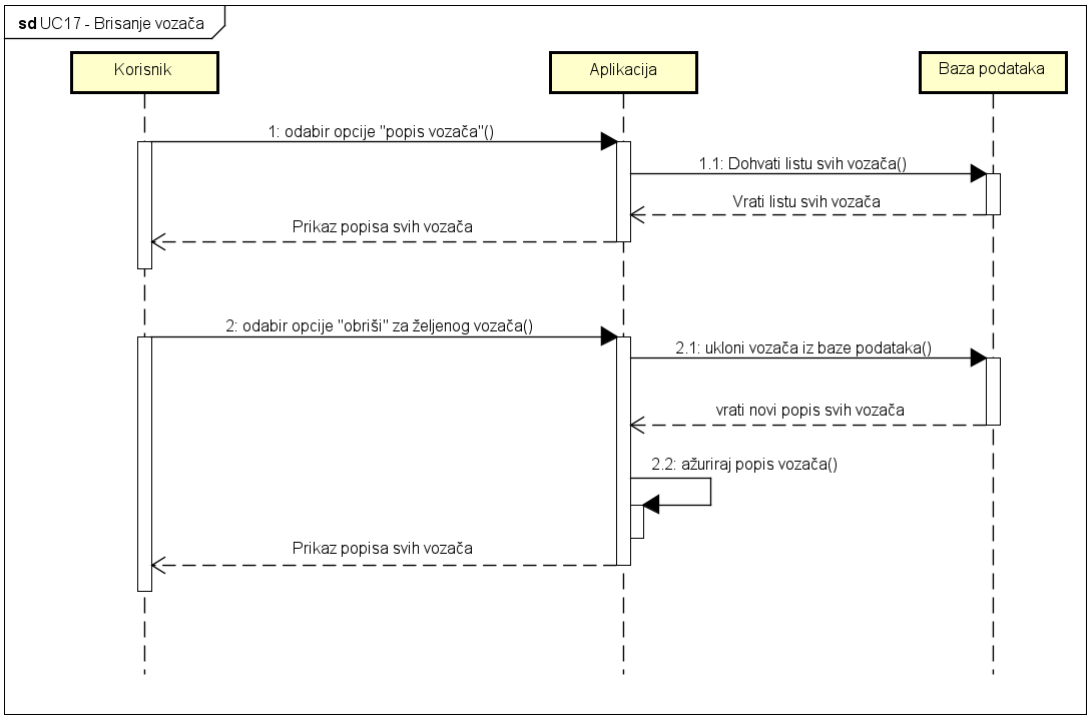
Korisnik koji je prijavljen sa korisničkim računom tvrtke može prijaviti novu parkirališnu površinu. Odabirom opcije ”Dodaj parkirališnu površinu” otvara se izbornik koji od korisnika traži unos podataka. Klikom miša na karti, korisnik postavlja marker na željenu lokaciju i time označava mjesto parkirališne površine. Google Karte prosljeđuju koordinate lokacije aplikaciji te aplikacija prikazuje koordinate korisniku. Ako korisnik nije zadovoljan odabranim koordinatama, klikom na opciju ”restartaj marker” može ponovno označiti lokaciju na karti. Nakon što korisnik unese ostale tražene podatke i odabere ”spremi lokaciju”, podaci se spremaju u bazu podataka.



Slika 3.7: Sekvencijski dijagram - prijava parkirališne površine

Obrazac uporabe UC17 - Brisanje vozača

Administrator može pregledati i upravljati svim vozačima. Odabirom opcije ”Popis vozača” prikazuje se lista svih prijavljenih vozača. Klikom na gumb ”Obriši” pored željenog vozača, aplikacija pristupa bazi podataka i uklanja ga. Baza podataka šalje novi popis prijavljenih vozača, aplikacija se ažurira i prikazuje novu listu svih vozača.



Slika 3.8: Sekvencijski dijagram - brisanje vozača

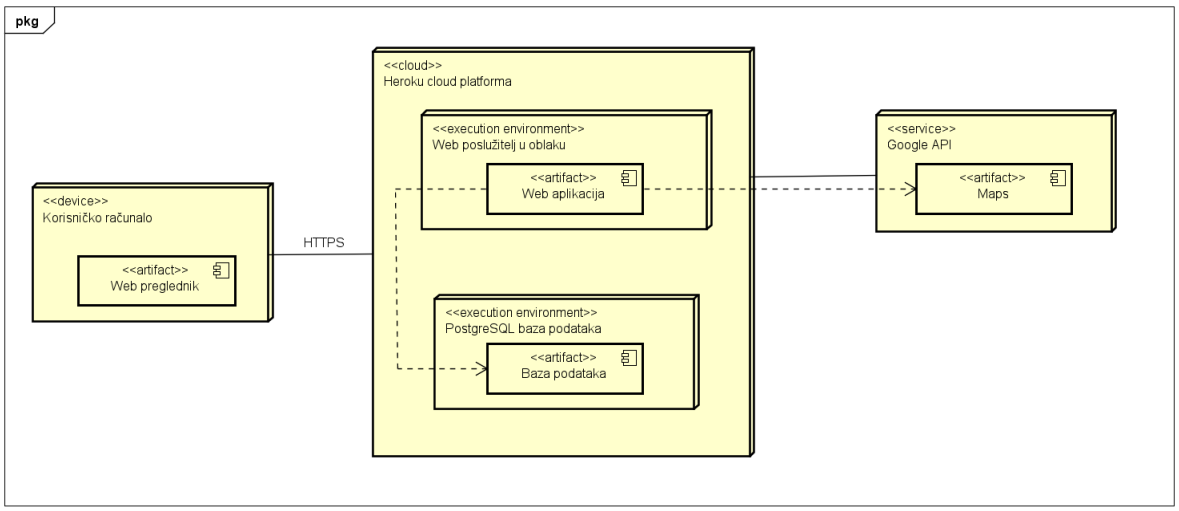
3.2 Ostali zahtjevi

- Sustav treba omogućiti rad više korisnika u stvarnom vremenu
- Sustav mora podržavati hrvatsku abecedu koji se mogu pojavljivati u korisničkim imenima i nazivima objekata na karti grada Zagreba
- Komunikacija sa bazom podataka (slanje upita i primanje odgovora od baze) ne smije trajati dulje od nekoliko sekundi
- Sustav treba biti implementiran kao web aplikacija
- Korištenje sustava na način na koji nije zamišljeno ne smije narušiti funkcionalnost i rad
- Sustav treba biti intuitivan za korištenje
- Nadogradnja sustava provodi se na temelju zadnje verzije bez mijenjanja ključnih dijelova sustava
- Sustav kao valutu treba podržati HRK
- Cijene rezervacija prikazuju točan iznos do drugog decimalnog mjesta
- Baza podataka treba biti otporna na napade i greške
- Pristup sustavu treba biti omogućen iz javne mreže pomoću HTTPS-a
- Sustav korisniku mora omogućiti registraciju neograničenog broja automobila
- Lokacija parkirališnih površina ima odstupanje od maksimalno 30 metara
- Sustav je dostupan 24 sata dnevno, 365 dana u godini (osim u trenucima obnavljanja ili nadogradnje sustava)

4. Arhitektura i dizajn sustava

Arhitektura je podjeljena na tri sustava:

- *Web Aplikacija*
- *Web Poslužitelj*
- *Baze podataka*



Slika 4.1: Dijagram arhitekture sustava

Korisnik aplikaciji pristupa preko web preglednika. *Web preglednik* je program kojim korisnik pristupa web sadržaju. Jedna od njegovih glavnih zadaća je prevođenje koda u konačni produkt sa svim sadržajima namijenjen korisniku.

Pošto je aplikacija dinamička, preglednik komunicira s web aplikacijom preko *web poslužitelja*, odnosno servera, čija je glavna zadaća prosljeđivanje korisničkih zahtjeva aplikaciji. Ta komunikacija odvija se preko HTTPS protokola kako bi se omogućio siguran prijenos korisnikovih podataka i informacija.

Web aplikacija obrađuje te zahtjeve, pristupa bazi podatka i ostalim vanjskim servisima po potrebi te šalje odgovor na korisinkov zahtjev u obliku HTML dokumenta razmuljiv web pregledniku. Ako se zahtjev nije uspio izvršiti, aplikacija će korinsiku dojaviti pogrešku.

Baza podataka služi za pohranu, izmjenu i dohvat podataka koji će se obrađivati u aplikaciji. Za njenu implementaciju koristiti ćemo Postgresql.

Za izradu web aplikacije odlučili smo koristiti Node.js, te ćemo većinu funkcionalnosti aplikacije ostvariti uz pomoć Express frameworka te Pug HTML template enginea. Za spajanje aplikacije s bazom podataka koristiti ćemo Connect-pg-Simple.

Aplikaciju ćemo razvijati u razvojnom okruženju Microsoft Visual Studio. Arhitektura sustava temeljit će se na Model-View-Controller (MVC) arhitekturi.

4.1 Baza podataka

Za naš sustav koristiti ćemo relacijsku bazu podataka, ostvarenu u aplikaciji Postgresql. Baza je prilagođena brzom pohranjivanju i dohvaćanju podataka, te su entiteti napravljeni tako da odgovaraju modelima u dijagramu razreda.

Baza podataka sastoji se od sljedećih entiteta:

- *Korisnik*
- *Tvrtka*
- *Parking*
- *Vozilo*
- *Rezervacija*
- *Sjednica*

4.1.1 Opis tablica

Korisnik je entitet koji čuva informacije o korisniku aplikacije *ParkirajMe*. Sadrži atribute: Korisničko ime i OIB korisnika, broj kreditne kartice, e-mail korisnika, te njegovo ime i prezime. Ovaj entitet u vezi je *One-to-Many* s entitetom vozilo preko korisničkog imena, zato jer svaki korisnik može posjedovati više vozila, a jedno vozilo također može imati više korisnika koji ga parkiraju.

korisnik		
Korisničko ime	VARCHAR	Jedinstveno korisničko ime korisnika
Lozinka	VARCHAR	Hash jedinstvene lozinke korisnika
OIB	CHAR(11)	Osobni identifikacijski broj korisnika
Broj kred. kartice	VARCHAR	Broj kreditne kartice korisnika
Email	VARCHAR	adresa elektroničke pošte korisnika
Ime	VARCHAR	Ime korisnika
Prezime	VARCHAR	Prezime korisnika
Razina ovlasti	INT	Razina ovlasti korisnika

Tvrtka je entitet koji predstavlja korisnički račun tvrtke koji je u njeno ime otvorio ovlašteni zaposlenik tvrtke. Sadrži atribute: korisničko ime, lozinka, OIB

tvrtke, Email , ime te adresu sjedišta. Povezan je vezom *One-To-Many* s entitetom parking, zato jer je jedna tvrtka može imati više parkinga.

tvrtka		
Korisničko ime	VARCHAR	Jedinstveno korisničko ime ovlaštenog zaposlenika tvrtke
Lozinka	VARCHAR	Hash jedinstvene lozinke ovlaštenog zaposlenika tvrtke
OIB tvrtke	CHAR(11)	Osobni identifikacijski broj tvrtke
Email	VARCHAR	adresa elektroničke pošte tvrtke
Ime	VARCHAR	Ime tvrtke
adresaTvrtka	VARCHAR	Adresa sjedišta tvrtke

Vozilo je entitet koji pohranjuje koja sve vozila je u aplikaciju unio neki korisnik, te su to jedina dva atributa ovog entiteta. U vezi je *Many-to-One* s entitetom korisnik preko atributa korisničko ime, te u vezi *One-to-Many* s entitetom rezervacija preko atributa registracija.

vozilo		
Korisničko ime	VARCHAR	Jedinstveno korisničko ime vlasnika vozila
Registracija	VARCHAR	Registracija automobila

Rezervacija je entitet koji sadržava podatke o rezervaciji jednog parkirnog mjesta. Njezini atributi su: ID rezervacije, ID grupe rezervacija kojoj ova rezervacija pripada, Korisničko ime onog tko je rezervirao, registracija vozila, ID parkinga, datum i vrijeme početka i kraja rezervacije. Povezan je vezom *Many-to-One* sa entitetom Vozilo preko atributa Korisničko ime i rezervacija (Ključ entiteta vozilo), te s entitetom Parking vezom *Many-to-One* preko atributa Parking ID.

rezervacija		
IDRezervacija	SERIAL	Jedinstveni ID rezervacije
ID grupe	INT	Jedinstvena oznaka grupe rezervacija
Korisničko ime	VARCHAR	Jedinstveno korisničko ime vlasnika vozila

Registracija vozila	VARCHAR	Registracija automobila koji će zauzeti mjesto
IDParking	INT	Jedinstvena oznaka parkinga
Pocetak rezervacije	TIMESTAMP	Početak rezervacije
Kraj rezervacije	TIMESTAMP	Završetak rezervacije

Parking je entitet koji sadržava podatke o jednom parkingu. Njegovi atributi su: ID parkinga, OIB tvrtke koja ga je objavila, ukupan kapacitet, broj slobodnih mjesta, cijena parkirnog mjesta, adresa parkinga, te geografska širina i dužina njegove lokacije. Povezan je sa entitetom tvrtka vezom *Many-to-One* pošto jedna tvrtka može imati više parkinga, te s entitetom rezervacije vezom *One-to-Many* zato jer na jednom parkingu može biti više rezervacija. Također je povezan s entitetom Lokacija vezom *One-to-One* preko atributa ID lokacija.

parking		
ID parkinga	SERIAL	Jedinstvena oznaka parkinga
Korisničko ime tvrtke	VARCHAR	korisničko ime tvrtke koja je vlasnik parkinga
Kapacitet	INT	Ukupan kapacitet parkinga
Broj slob. mjesta	INT	Broj slobodnih mjesta na parkingu
Cijena po satu	NUMERIC	Cijena rezervacije parkinga po satu
ID lokacija	INT	Jedinstvena oznaka lokacije parkinga

Lokacija je entitet koji sadržava podatke o jednoj lokaciji. Njegovi atributi su ID lokacije, njena adresa, te geografska širina i dužina. Povezan je vezom *One-to-One* s entitetom Parking zato jer se na jednoj lokaciji može nalaziti samo jedan parking.

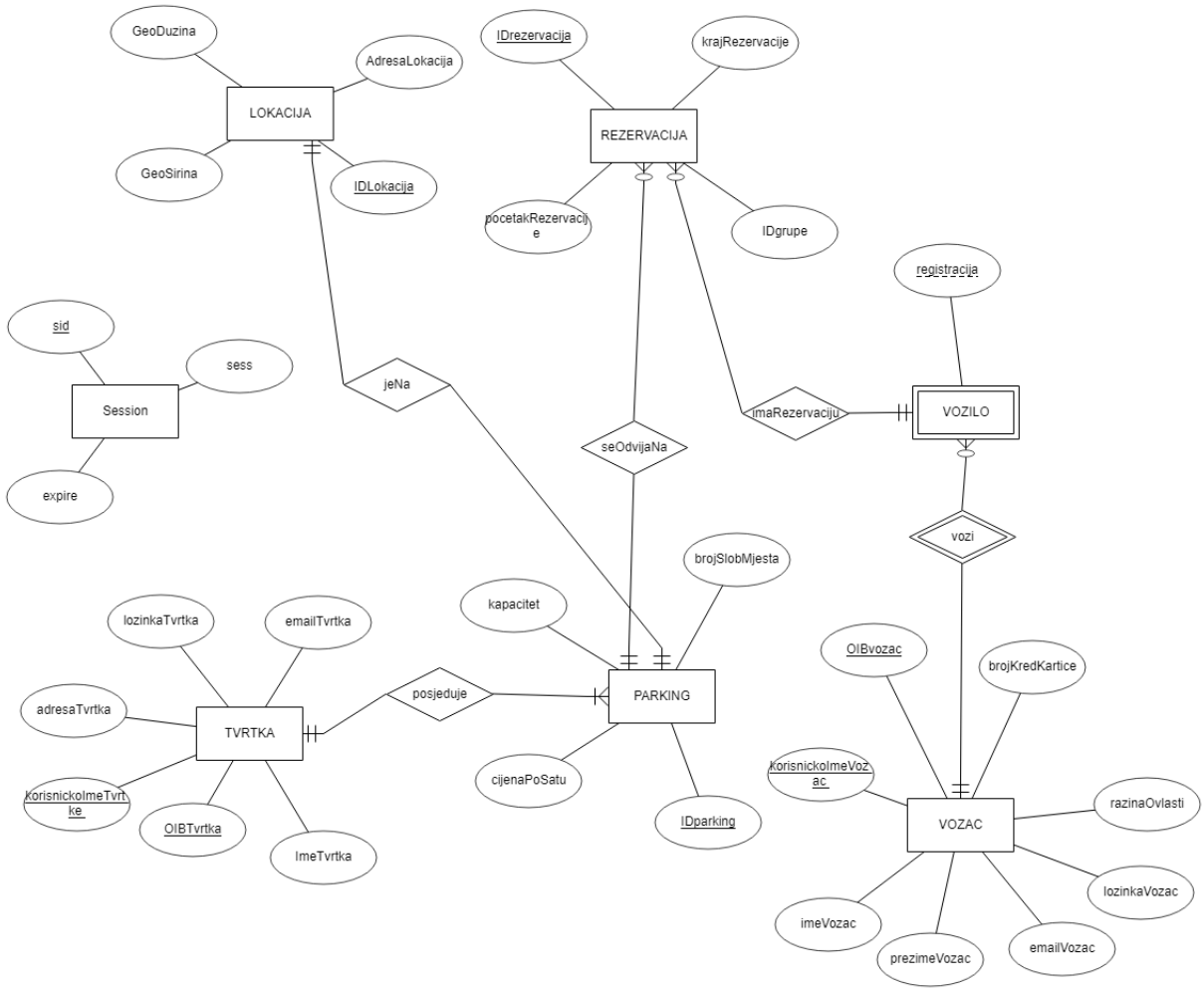
lokacija		
ID lokacije	SERIAL	Jedinstvena oznaka lokacije
Adresa lokacije	VARCHAR	Adresa lokacije
Geo. širina	NUMERIC	Geograska širina lokacije
Geo. dužina	NUMERIC	Geograska dužina lokacije

Sjednica je entitet koji sadrži informacije o sjednicama koje se stvaraju na stranici ParkirajMe. Sadrži atribute ID sjednice, informacije o sjednici ti vrijeme

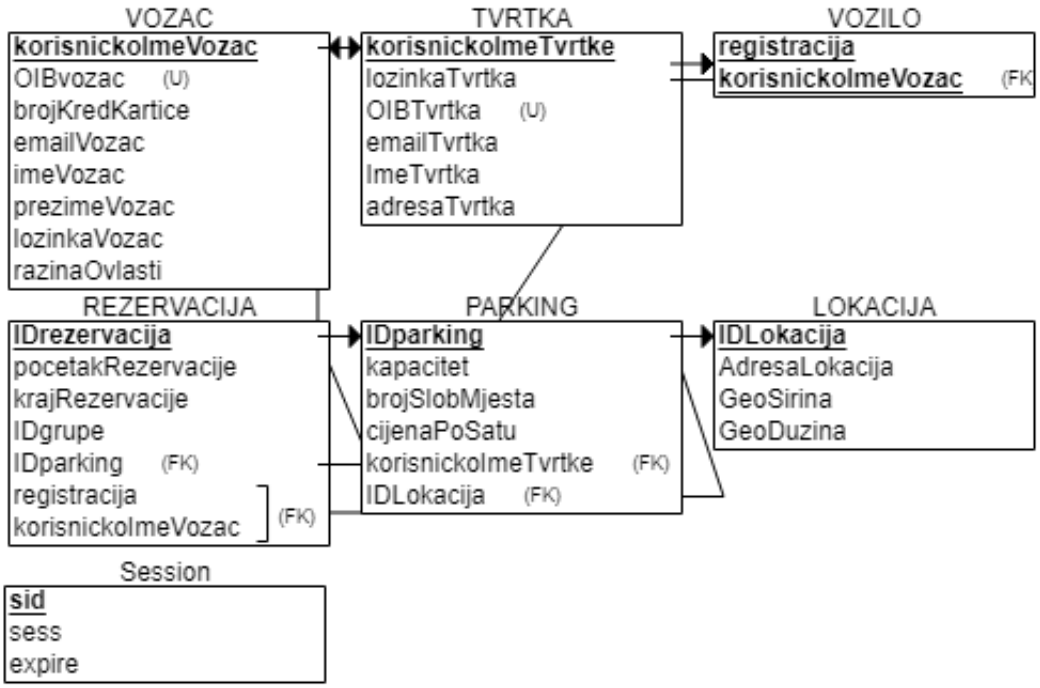
trajanja sjednice.

sjednica		
ID sjednice	VARCAHR	Jedinstvena oznaka sjednice
Info	JSON	Informacije o sjednici
Trajanje	TIMESTAMP	Trajanje sjednice

4.1.2 Dijagram baze podataka



Slika 4.2: ER model



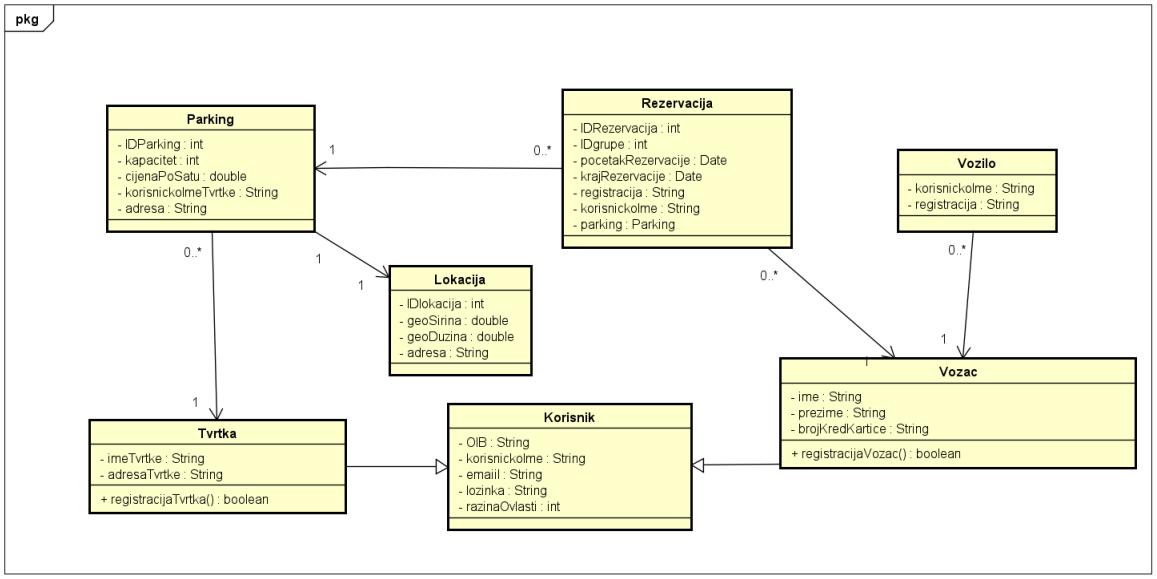
Slika 4.3: relacijska shema

4.2 Dijagram razreda

Dijagrami razreda MVC arhitketure sustava prikazani su na slikama 4.4, 4.5 i 4.6. Razredi na slici 4.3 predstavljaju klase modela, koji su uglavnom direktno povezani s entitetima u bazi podataka.
Razredi na slici 4.4 su pomoćne klase koje pomažu upravljanju podatcima, te najčešće služe za komunikaciju s bazom podataka.
Na slici 4.5 se vide razredi koji implementiraju funkcionalnosti *routera*[1]. Oni prikazuju pug[2] datoteke koje određuju sadržaj i izgled web stranice.
Razred **Korisnik** predstavlja bilo kojeg korisnika aplikacije, bio to vozač, tvrtka, administrator ili ne-registrirani korisnik. **Vozač** predstavlja korisnika aplikacije koji želi rezervirati parirališno mjesto, a **Tvrtka** korisnika koji želi iznajmljivati svoju parkirališnu površinu. **Administrator** je korisnik s najvećim ovlastima na aplikaciji. Klasa **Rezervacija** u sebi pohranjuje informacije o registracijama, kojih postoji 3 različite vrste (jednokratne, ponavljajuće i trajne) koje se označuju atributom **idgrupe** koja označava pripadajuću vrstu rezervacije:

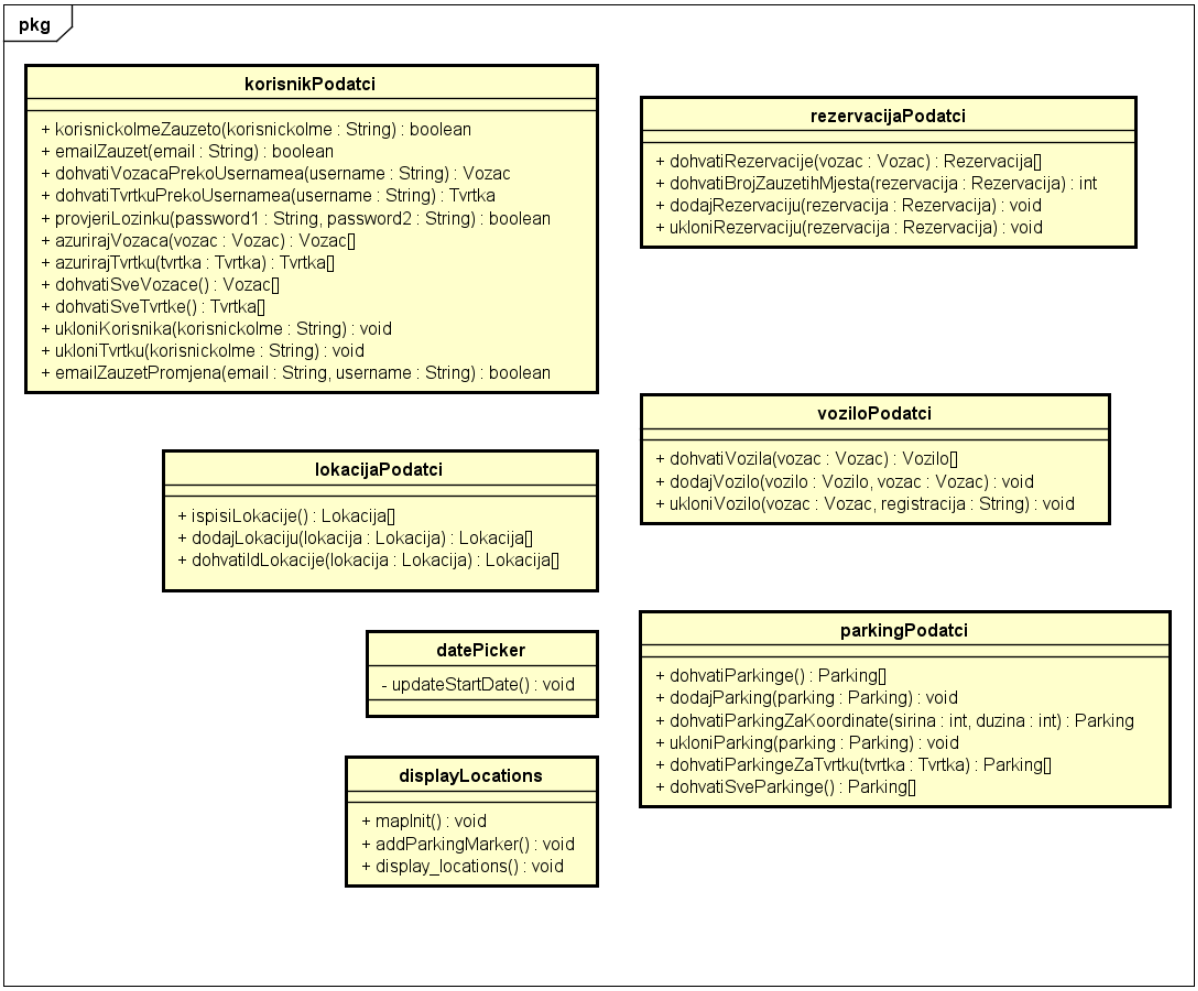
- jednokratna = 0
- ponavljajuća = 1
- trajna = 2

[1] https://expressjs.com/en/guide/routing.html
[2] https://pugjs.org/api/getting-started.html



Slika 4.4: Dijagram razreda za modele

Pomoćne klase su klase koje u sebi sadrže pomoćne funkcije koje potpomažu ostvarivati funkcionalnosti drugih klasa, te većina služi za direktno slanje upita bazi podataka. Klasa **datePicker** služi za prikaz izbornika datuma, dok **displayLocations** služi za prikaz karte preko *Google Maps APIa*, te prikaz lokacija na karti.

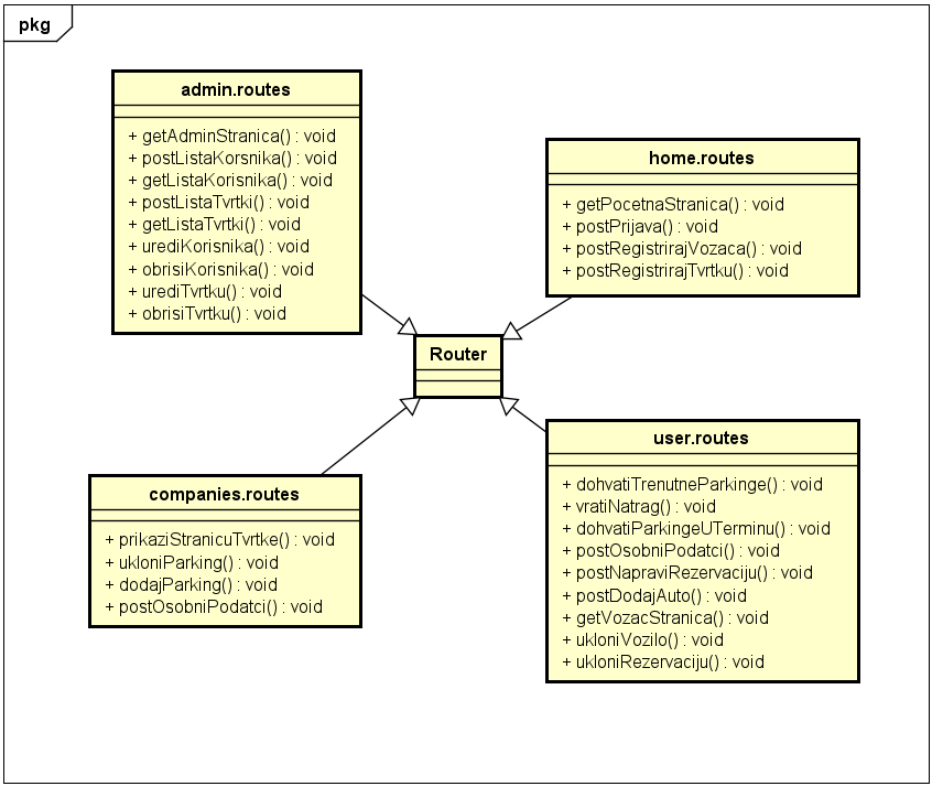


Slika 4.5: Dijagram razreda za pomoćne razrede

Routeri su klase koji služe za povezivanje sa *frontendom*, odnosno ostvaruju prikazivanje pug datoteka. Postoje routeri za **home-page**, stranicu za vozače (**user.routes**), stranicu za tvrtke (**company.routes**) te administratora (**admin.routes**).

Napomena: sve funkcije kao parametre primaju *request* i *response* objekt, te referencu *next* na idući *middleware*[3], no parametri nisu prikazani na dijagramu radi preglednosti.

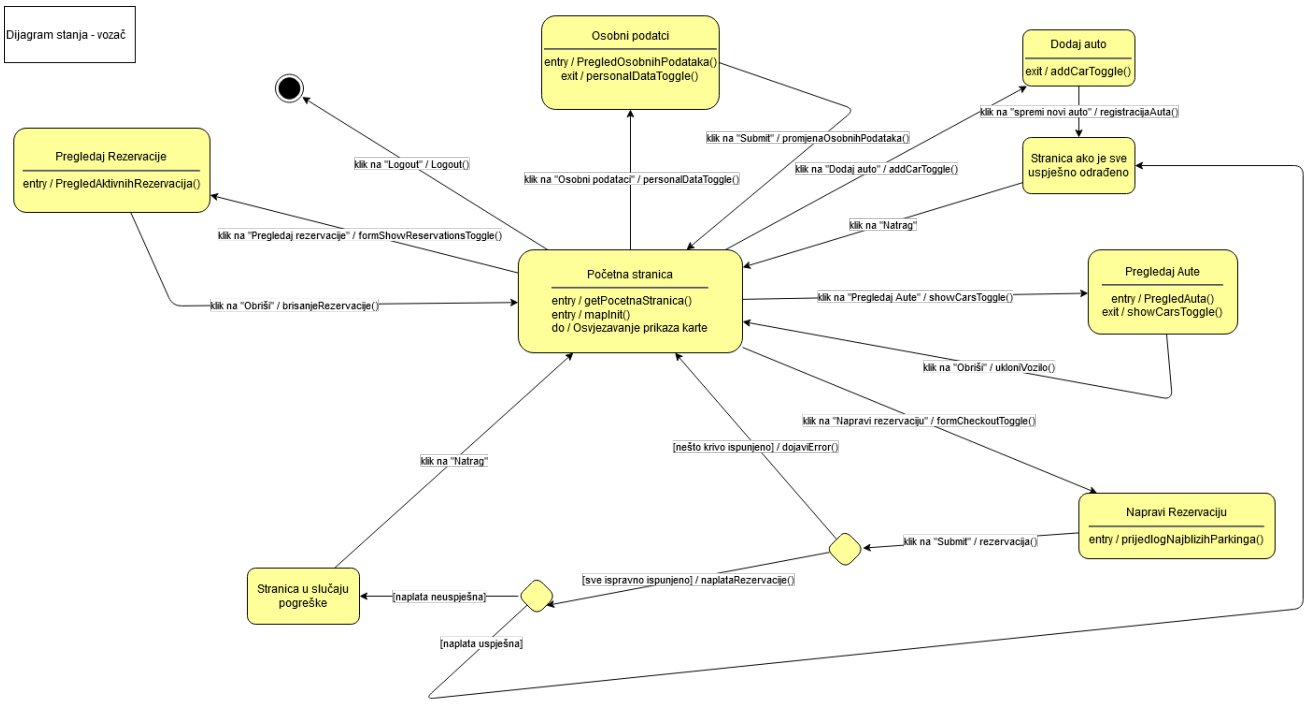
[3] *middleware* funckije su one funkcije koje imaju pristup *req i res* objektima i idućoj *middleware* funkciji



Slika 4.6: Dijagram razreda za routere

4.3 Dijagram stanja

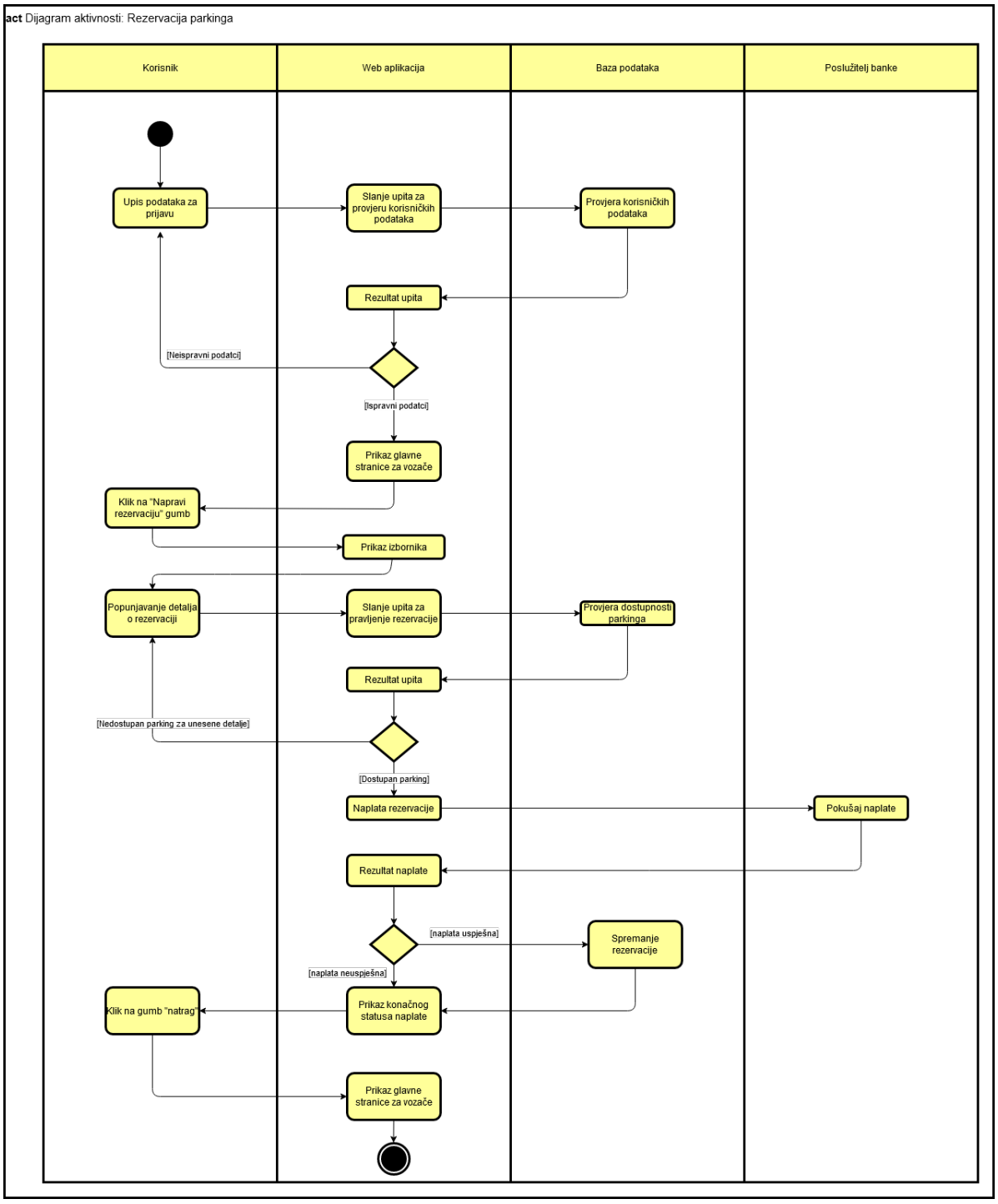
Ovaj dijagram stanja prikazuje funkcionalost razreda **Vozač**. Na web-stranici za vozače (ranije spomenut **user.routes**) prikazuje se karta pomoću integracije sa *Google Maps APIem*, te više opcija za izabrati koje su naznačene u pravokutnicima. Klik na svaku od njih je otvara, odnosno zatvara te se onda nakon otvaranja prikazuju detalji te opcije. Najkompleksnija opcija je ona za pravljenje rezervacije parking mjesta, jer se u sklopu nje komunicira i sa bazom podataka radi provjere detalja rezervacije i sa poslužiteljem banke gdje se izvršava proces naplate. Greška u bilo kojem dijelu dovršavanja rezervacije otkazuje proces zapisivanja iste u bazu podtaka i njenu naplatu, te usmjerava korisnika nazad na glavnu stranicu gdje može pokušati opet uspješno napraviti rezervaciju.



Slika 4.7: Dijagrama stanja - razred vozač

4.4 Dijagram aktivnosti

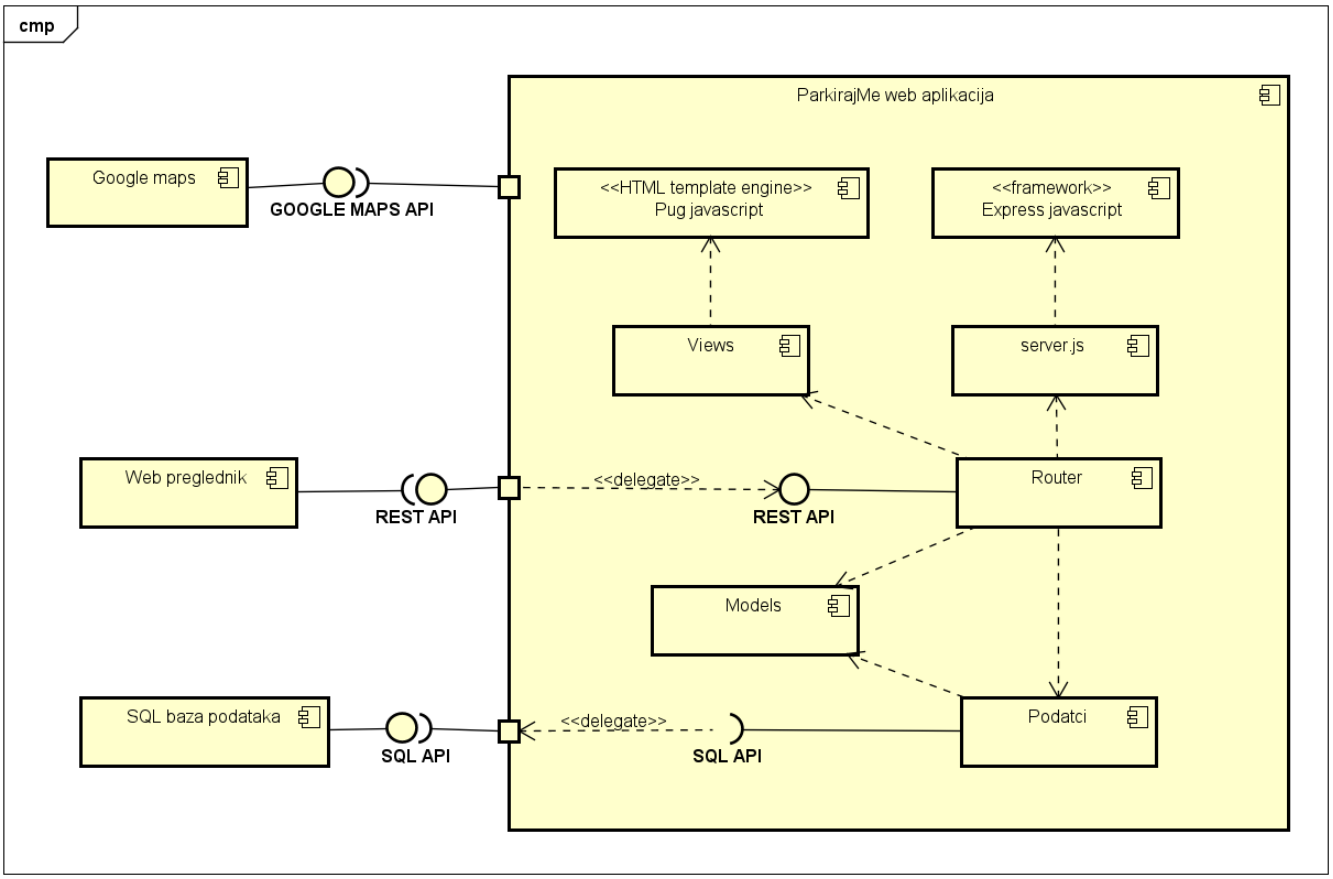
Dio sustava prikazan ovim dijagramom aktivnosti je postupak rezervacije parkinga. Aktivnost započinje (uspješnom) prijavom u sustav. Nakon toga odabire se opcija pravljenja rezervacije gdje se odabire parking i popunjavaju traženi podatci. Važno je napomenuti da se automatski odabire najbliža parking lokacija, koja se onda može ručno izmjeniti po želji korisnika. Nakon ispunjavanja detalja rezervacija se može dovršiti i poslati na naplatu, te ako je sve u redu ona se pamti u bazi podataka i korisnika se obavještava o uspješnoj rezervaciji. Podaci za naplatu se šalju banci na provjeru i izvršavanje naloga pa je neophodno ostvariti sigurnu komunikaciju s tim poslužiteljem radi sigurnosti korisničkih podataka.



Slika 4.8: Dijagrama aktivnosti - rezervacija parkinga

4.5 Dijagram komponenti

Na slici je prikazan dijagram komponenti aplikacije. Dijagram prikazuje organizaciju i međuodnos implementacijskih komponenata napravljenih u okviru MVC arhitekture te njihov odnos s okolinom, tj. vanjskim komponentama aplikacije: bazom podataka, *Google Mapsom* i web preglednikom. Web preglednik komunicira s *routerima* preko REST API sučelja. Veza između SQL baze podataka i web aplikacije, odnosno njene komponente *podatci* koja dohvađa i obrađuje podatke iz baze podataka, ostvarena je pomoću SQL API sučelja dok je veza s vanjskim servisom *Google Maps* ostavren pomoću GOOGLE MAPS API sučelja.



Slika 4.9: Dijagram komponenti

5. Implementacija i korisničko sučelje

5.1 Korištene tehnologije i alati

Komunikacija u timu realizirana je korištenjem aplikacija MS Teams[1], WhatsApp[2] i Discord[3]. Za izradu UML dijagrama korišteni su alati Astah Professional[4] i Draw.io[5]. Kao sustav za upravljanje izvornim kodom korišten je Git[6], a udaljenom repozitoriju projekta se može pristupiti na web platformi GitLab[7].

Kao razvojno okruženje korišten je Microsoft Visual Studio Code[8], te Eclipse[9] i IntelliJ[10] za dijelove koda pisane u Javi. Visual Studio Code je besplatan editor izvornog koda razvijen od tvrtke Microsoft za Windows, Linux i macOS. Njegove značajke uključuju podršku za debuggiranje, isticanje sintakse, inteligentno završavanje napisanog koda, refakotoriranje koda i ugrađen Git. IntelliJ i Eclipse su IDEvi razvijeni za programski jezik Java.

Aplikacija je napisana koristeći Express[11] radni okvir i jezik Javascript za izradu backenda te Node.js[12] i pug[13] za izdradu frontenda. Node.js je kao asikroni i događajima-pokrenut JavaScript runtime dizajniran za izradu skalabilnih web aplikacija. Pug je HTML-ov predprocesor.

Za prikaz karte i lokacije parkinga korišten je Google Maps api[14]. Za potrebe baze podataka korišten je PostgreSQL[15]. Sama aplikacija i baza podataka je

[1] https://www.microsoft.com/en-us/microsoft-365/microsoft-teams/group-chat-software
[2] https://www.whatsapp.com/?lang=en
[3] https://discord.com/
[4] astah.net/products/astah-professional/
[5] https://app.diagrams.net/
[6] https://git-scm.com/
[7] https://gitlab.com/
[8] https://code.visualstudio.com/
[9] https://www.eclipse.org/
[10] https://www.jetbrains.com/idea/
[11] https://expressjs.com/
[12] https://nodejs.org/en/
[13] https://pugjs.org/api/getting-started.html
[14] https://developers.google.com/maps/documentation/javascript/overview
[15] https://www.postgresql.org/

objavljena na cloud platformi Heroku[16].

Za potrebe sistemskog testiranja aplikacije korišten je Selenium[17] i jezik Python (koristeno razvojno okruzenje Pycharm[18]). Selenium je prijenosni radni okvir za testiranje web aplikacija. Pruža alat za pravljenje funkcionalnih testova bez potrebe da se uči skriptni jezik za testiranje.
Za testiranje Node.js funkcija korišten je Mocha[19]. Kao napredni uređivač teksta korišten je Sublime[20].

[16]https://www.heroku.com/
[17]https://www.selenium.dev/
[18]https://www.jetbrains.com/pycharm/
[19]https://mochajs.org/
[20]https://www.sublimetext.com/

5.2 Ispitivanje programskog rješenja

Ispitivanje (testiranje) se provelo u dva oblika. Prvi je kroz sistemske, a drugi kroz *unit* testove. Sistemskim testovima testiran je dio sustava (aplikacije), a *unit* testovima manji segment (funkcija) u kodu.

Sistemskim testovima provjerena je funkcionalnost registracije korisnika, registracija tvrtke, prijava korisnika, prijava tvrtke, dodavanje automobila (korisnička funkcionalnost) te promjena imena (korisnička funkcionalnost).

5.2.1 Ispitivanje komponenti

Testiranje je obavljeno u Node.js-u pomoću Mocha-e.

Testiranjem komponenti željelo se provjeriti ispravnost funkcija napisanih na klijentskoj strani. Ovim testovima provjeravalo se dohvaćanje korisničkih podataka (ime i prezime), dohvaćanje lokacije (preko ID-a i koordinata), dohvaćanje rezervacija i dohvaćanje vozila.

U nastavku je prikazan kod i rezultati testiranja.

```
const assert = require('chai').assert;
const KorisnikPodatci = require('../tools/KorisnikPodatci');

describe('KorisnikPodatci', async function(){
   it('Trebao bi vratiti test', async function(){
      const vozac = await
         KorisnikPodatci.dohvatiVozacaPrekoUsernamea('testuser');
      assert.equal(vozac.imeVozac, 'test');
   });

   it('Trebao bi vratiti test', async function(){
      const vozac = await
         KorisnikPodatci.dohvatiVozacaPrekoUsernamea('testuser');
      assert.equal(vozac.prezimeVozac, 'test');
   });
});
```

Slika 5.1: Prikaz testa dohvaćanja korisničkih podataka

```
const assert = require('chai').assert;
const LokacijaPodatci = require('../tools/LokacijaPodatci');
const Lokacija = require('../models/Lokacija')

describe('LokacijaPodatci', async function(){
   it('Trebao bi vratiti 36', async function(){
      lokacijaBezId = new Lokacija('trg bana josipa jelacica',
         45.813264153294114, 15.977233094526218);
      const lokacija= await
         LokacijaPodatci.dohvatiIdLokacije(lokacijaBezId);
      assert.equal(lokacija[0].idlokacija, 36);
   });
});
```

Slika 5.2: Prikaz testa dohvaćanja lokacije preko id-a

```
const assert = require('chai').assert;
const ParkingPodatci = require('../tools/ParkingPodatci');

describe('LokacijaPodatci', async function(){
   it('Postoje dva parkinga sa istom lokacijom \
   (moze se dogodit da neka tvrtka slucajno doda),\
   pa se vraca samo jedna', async function(){
      //jelacic koordinate, ima dvije lokacije
      let parking = await
         ParkingPodatci.dohvatiParkingZaKoordinate(
      45.813264153294114, 15.977233094526218);
      assert.equal(parking.idlokacija, 35);
   });
});
```

Slika 5.3: Prikaz testa dohvaćanja lokacije preko koordinata

```
const assert = require('chai').assert;
const RezervacijaPodatci =
    require('../tools/RezervacijaPodatci');
const KorisnikPodatci = require('../tools/KorisnikPodatci');


describe('RezervacijaPodatci', async function(){
   it('Trebao bi vratiti sava, tvrtka, 500, test', async
       function(){
      vozac = await
         KorisnikPodatci.dohvatiVozacaPrekoUsernamea('testuser')
      rezervacije = await
         RezervacijaPodatci.dohvatiRezervacije(vozac);
      assert.equal(rezervacije[0].adresalokacija, 'sava');
      assert.equal(rezervacije[0].korisnickoimetvrtke,
         'tvrtka');
      assert.equal(rezervacije[0].kapacitet, 500);
      assert.equal(rezervacije[0].registracija, 'test');
   });
});
```

Slika 5.4: Prikaz testa dohvaćanja rezervacija

```
const assert = require('chai').assert;
const VoziloPodatci = require('../tools/VoziloPodatci');
const KorisnikPodatci = require('../tools/KorisnikPodatci');
const Vozilo = require('../models/Vozilo')
const chai = require('chai')
const expect = chai.expect
chai.use(require('chai-as-promised'))

describe('VoziloPodatci', async function(){
  it('Trebao bi vratiti test, auto1, auto2', async
      function(){
    vozac = await
        KorisnikPodatci.dohvatiVozacaPrekoUsernamea('testuser')
    const vozila = await VoziloPodatci.dohvatiVozila(vozac);
    assert.equal(vozila[0].registracija, 'test');
    assert.equal(vozila[1].registracija, 'auto1');
    assert.equal(vozila[2].registracija, 'auto2');
  });

  it('Trebao bi vratiti 4, 3, noviauto', async function(){
    vozac = await
        KorisnikPodatci.dohvatiVozacaPrekoUsernamea('testuser')
    vozilo = new Vozilo('noviauto', vozac);
    await VoziloPodatci.dodajVozilo(vozilo, vozac);
    let vozila = await VoziloPodatci.dohvatiVozila(vozac);
    assert.equal(vozila.length, 4);
    await VoziloPodatci.ukloniVozilo(vozac, 'noviauto');
    vozila = await VoziloPodatci.dohvatiVozila(vozac);
    assert.equal(vozila.length, 3);
  });

  it('Trebao bi biti error', async function(){
    vozac = await
        KorisnikPodatci.dohvatiVozacaPrekoUsernamea('testuser')
    vozilo = new Vozilo(' ', vozac);
    await expect(VoziloPodatci.dodajVozilo(vozilo,
        vozac)).to.be.rejectedWith(Error)
  });

});
```

Slika 5.5: Prikaz testa dohvaćanje vozila

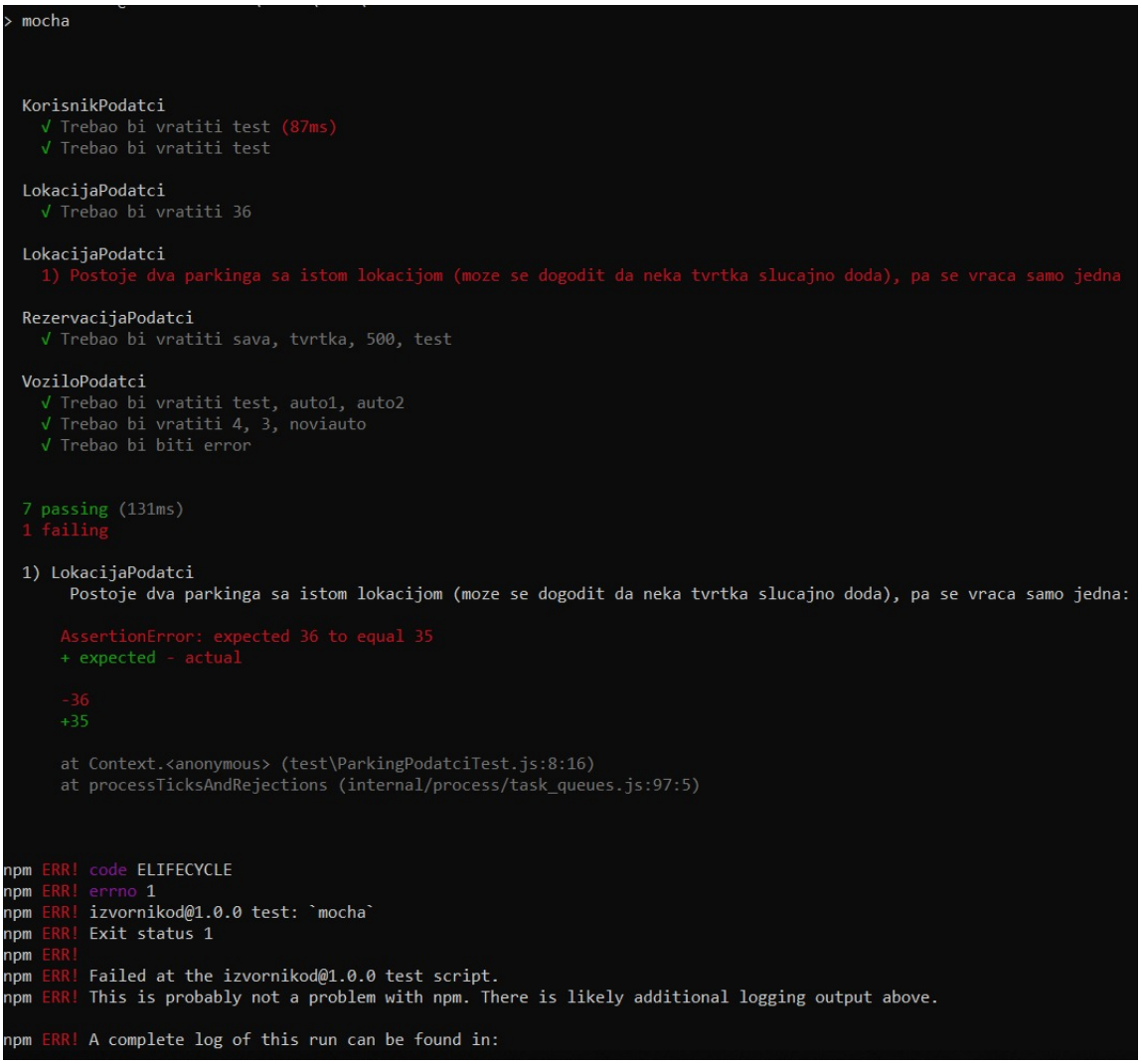
```
C:\Users\roko\ndb>npm run test

> izvornikod@1.0.0 test C:\Users\roko\ndb
> mocha



  KorisnikPodatci
    √ Trebao bi vratiti test (166ms)
    √ Trebao bi vratiti test

  LokacijaPodatci
    √ Trebao bi vratiti 36

  LokacijaPodatci
    √ Postoje dva parkinga sa istom lokacijom (moze se dogodit da neka tvrtka slucajno doda), pa se vraca samo jedna

  RezervacijaPodatci
    √ Trebao bi vratiti sava, tvrtka, 500, test

  VoziloPodatci
    √ Trebao bi vratiti test, auto1, auto2
    √ Trebao bi vratiti 4, 3, noviauto
    √ Trebao bi biti error


  8 passing (219ms)
```

Slika 5.6: Prikaz prolaza svih testova

KorisnikPodatciTest.js | LokacijaPodatciTest.js | ParkingPodatciTest.js | RezervacijaPodatciTest.js | VoziloPodatciTest.js

```
1  const assert = require('chai').assert;
2  const ParkingPodatci = require('../tools/ParkingPodatci');
3
4  describe('LokacijaPodatci', async function(){
5      it('Postoje dva parkinga sa istom lokacijom \
6          (moze se dogodit da neka tvrtka slucajno doda),\
7           pa se vraca samo jedna', async function(){
8          //jelacic koordinate, ima dvije lokacije
9          let parking = await ParkingPodatci.dohvatiParkingZaKoordinate(
10             45.813264153294114, 15.977233094526218);
11         assert.equal(parking.idlokacija, 35);
12     });
13 });
```

Slika 5.7: Prikaz testa u kojem se očekuje drugačija vrijednost od dobivene (označena razlika u odnosu na prijašnji test)



Slika 5.8: Prikaz neprolaženja svih testova

5.2.2 Ispitivanje sustava

Kod za testiranje je pisan u Pythonu, Selenium je korišten za interakciju s elementima u aplikaciji.

Kao pripremu za testiranje registracija napravljene su baze podataka koje sadrže nasumično generirane parametre (ime, prezime, broj kartice i slično). Prilikom registracije dohvaća se jedan set podataka koji se koriste za registraciju te se nakon toga bilježi kako su ti podatci iskorišteni, pri idućoj registraciji dohvaća se novi set podataka. Proces je iterativan. Razlog ovakvog pristupa naspram ustaljenog (procedure koja prije pokretanja testa registracije briše testnog korisnika ili tvrtku kako bi se mogao ponovno koristiti taj isti korisnik ili tvrtka za registraciju) je što se ovako ostvaruje dualna funkcija; testiranje i pisanje

podataka u bazu. Drugo ima za posljedicu lakše ručno testiranje svih funkcionalnosti vezanih uz korisnike i tvrtke pošto su već upisani u bazi te se ne treba stvarati ispitni skup korisnika i tvrki (na primjer: kako bi se testirale administratorske ovlasti nad njima)

Druga priprema koja je napravljena na *home* stranici je pridruživanje svakom relevantnom elementu za testiranje *id*. Svaki *id* je broj, te su vrijednosti posložene tako da se pri testiranju može iterirati po određenom spektru tih brojeva i jednostavno dohvaćati elemente, pritiskati na njih i upisivati u njih željene znakove.

Slika 5.9: Prikaz enumeracije *id*-eva

U nastavku je prikaz koda i rezultata testiranja. Većina testova koristi *driver* koji obavlja test s predanim parametrima koje mu određeni test predaje. Svi testovi se nalaze u klasi *TestApp*.

```
# ... importi

class TestApp(unittest.TestCase):

    PAGE_URL = "http://127.0.0.1:3000/"
    SHOW_LOG = False
    USER_USERNAME = "marin"
    USER_PASSWORD = "123123"

    IS_ALL_OK = True

    driver = webdriver.Chrome(ChromeDriverManager().install())

# ... nastavak koda
```

Slika 5.10: Prikaz varijabli i konstanti

```
# ... unutar TestApp

def login_driver(self, username, password,
    invert_score=False):
   driver = self.driver

   try:
      driver.get(self.PAGE_URL)
      time.sleep(4)

      i_0 = 2
      credentials = [username, password]

      for i_1 in credentials:
         user = driver.find_element_by_id(i_0)
         user.click()
         user.send_keys(i_1)
         i_0 += 1

      user = driver.find_element_by_id(i_0)
      user.click()

      time.sleep(4)
      try:
         tester = driver.find_element_by_id("name_ph")
      except:
         raise self.FailTest("login nije uspjesno napravljen")

      self.assertTrue(not invert_score)

   except self.FailTest as e:
      self.IS_ALL_OK = False
      print(e)

      self.assertTrue(invert_score)

   except:
      self.IS_ALL_OK = False
      import sys
      print(sys.exc_info()[0])
      self.assertTrue(False)
```

Slika 5.11: Prikaz login drivera

```
# ... unutar TestApp
 def test_login_user_false(self):
   username = "marin"
   password = "123d123"

   self.login_driver(username, password, True)
```

Slika 5.12: Prikaz testa koji ne uspjeva se prijaviti u aplikaciji

```
# ... unutar TestApp
def test_user_change_name(self):

   try:
      self.login_driver(self.USER_USERNAME, self.USER_PASSWORD)

      if self.IS_ALL_OK:


         time.sleep(4)

         self.driver.find_element_by_id(0).click()
         time.sleep(1)

         self.driver.find_element_by_id("firstname").click()
         self.driver.find_element_by_id("firstname").clear()
         self.driver.find_element_by_id("firstname").\
            send_keys(get_user_first_name())

         self.driver.find_element_by_id(4).click()

         self.assertTrue(True)

      else:
         raise self.FailTest("login nije uspjesno napravljen")


   except:
      self.assertTrue(False)
```

Slika 5.13: Prikaz testa promjena imena korisnika

```
# ... unutar TestApp

def test_user_add_car(self):

try:
   self.login_driver(self.USER_USERNAME, self.USER_PASSWORD)
   if self.IS_ALL_OK:

      time.sleep(4)

      self.driver.find_element_by_id("addCar").click()
      time.sleep(1)

      self.driver.find_element_by_id("addCarInputField").click()
      self.driver.find_element_by_id("addCarInputField").\
         send_keys(get_user_licence_plate("zg_plate"))

      self.driver.find_element_by_id("addCarButton").click()

      self.assertTrue(True)

   else:
      raise self.FailTest("login nije uspjesno napravljen")

except:
   self.assertTrue(False)
```

Slika 5.14: Prikaz testa dodavanja automobila

```
# ... unutar TestApp
   def test_login_user(self):

   username = "marin"
   password = "123123"

   self.login_driver(username, password)

def test_login_company(self):

   username = "marin doo"
   password = "123123d"

   self.login_driver(username, password)
```

Slika 5.15: Prikaz testova za prijavu (korisnika i tvrtke)

```
# ... unutar TestApp
def signup_user_driver(self, plate_tag):
   driver = self.driver
   credentials = [
      get_user_username(), get_user_oib(),
      get_user_first_name(), get_user_last_name(),
      get_user_email(), get_user_licence_plate(plate_tag),
      get_user_card(), get_user_password()
   ]
   if self.SHOW_LOG:
      print(credentials)
   try:
      driver.get(self.PAGE_URL)
      time.sleep(4)
      i_0 = 5
      user = driver.find_element_by_id(str(i_0))
      user.click()
      i_0 += 1
      password = credentials[-1]
      for i_1 in credentials:
         user = driver.find_element_by_id(str(i_0))
         user.click()
         i_0 += 1
         user.send_keys(i_1)
         if self.SHOW_LOG:
            print(i_1)
      user = driver.find_element_by_id(str(i_0))
      user.click()
      i_0 += 1
      user.send_keys(password)
      user = driver.find_element_by_id(str(i_0))
      user.click()
      time.sleep(4)
      try:
         tester = driver.find_element_by_id("name_ph")
      except:
         raise self.FailTest("login nije uspjesno napravljen")
      self.assertTrue(True)
   except self.FailTest as e:
      print(e)
      self.assertTrue(False)
   except:
      import sys
      print(sys.exc_info()[0])
      self.assertTrue(False)
```

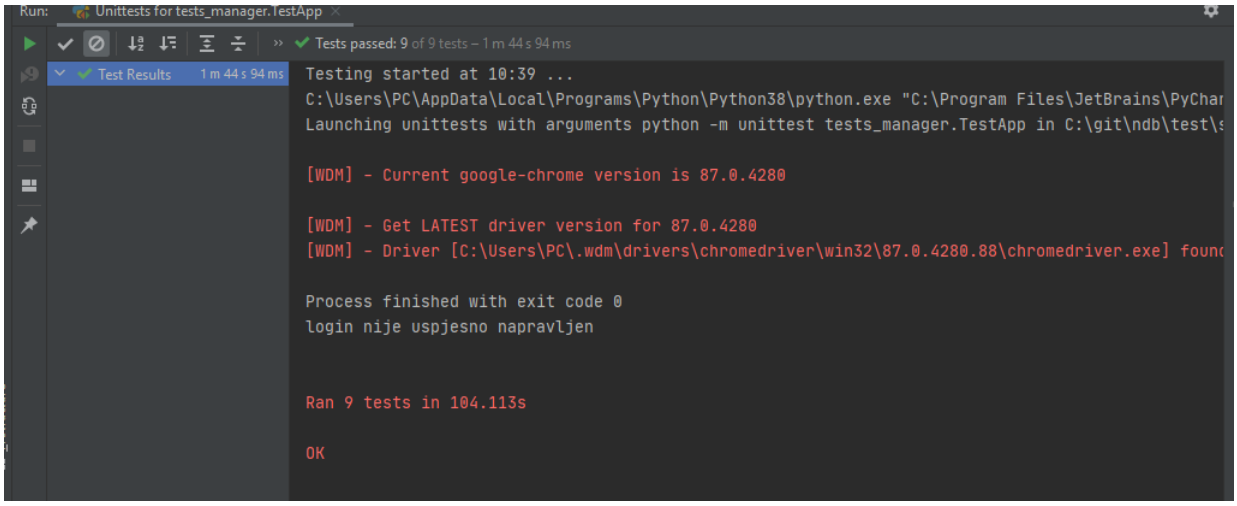
Slika 5.16: Prikaz drivera za registraciju korisnika

```
# ... unutar TestApp
# zg plate
def test_signup_user_1(self):

   self.signup_user_driver("zg_plate")

# cro plates
def test_signup_user_2(self):

   self.signup_user_driver("cro_plates")

# international_plate
def test_signup_user_3(self):

   self.signup_user_driver("international_plate")
```

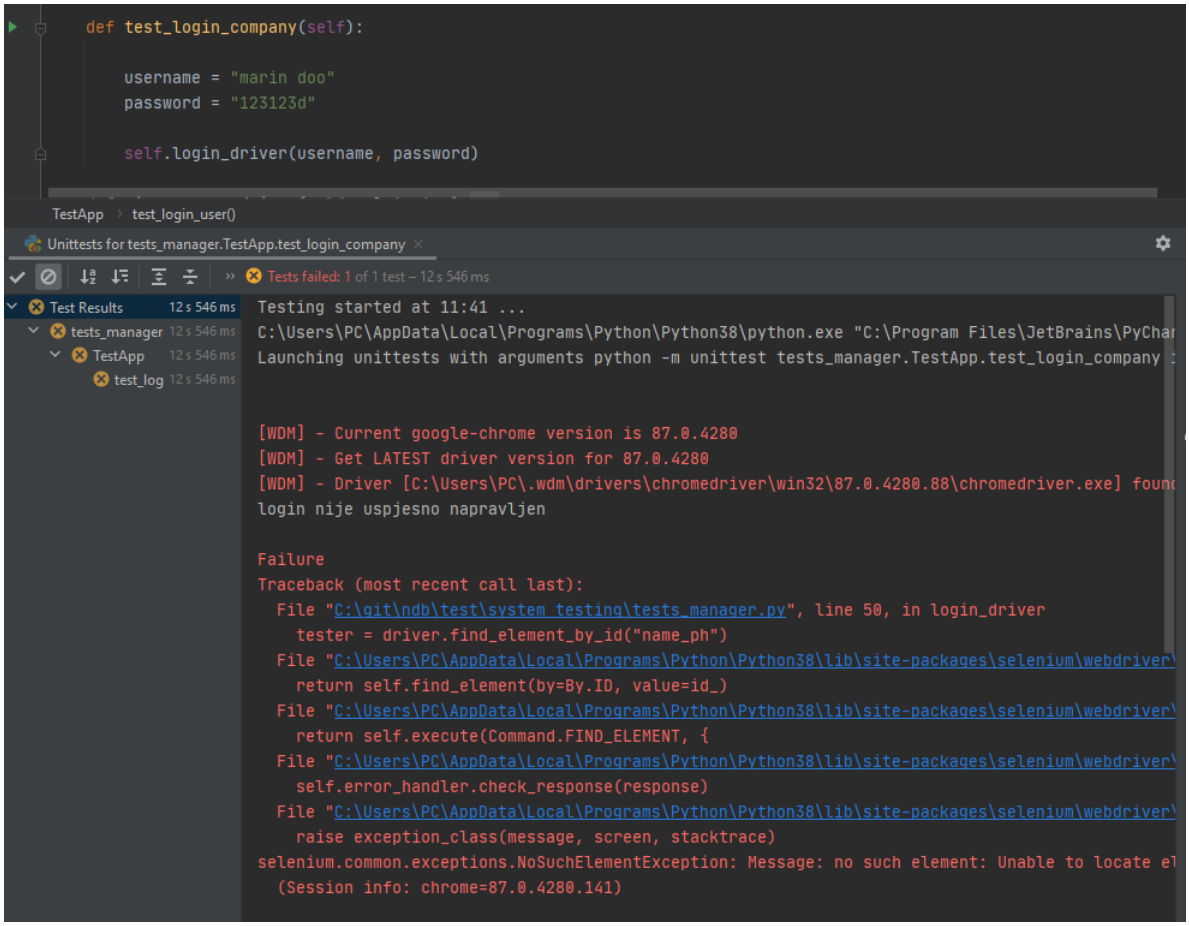
Slika 5.17: Prikaz registracije korisnika s različitim registracijama automobila

```
# ... unutar TestApp
def test_signup_company(self):
    driver = self.driver

    credentials = [
        get_company_oib(), get_company_name(),
        get_company_address(), get_company_email(),
        get_password()
    ]

    try:
        driver.get(self.PAGE_URL)
        time.sleep(4)
        i_0 = 16
        # button: register as company
        user = driver.find_element_by_id(str(i_0))
        user.click()
        i_0 += 1
        password = credentials[-1]
        if self.SHOW_LOG:
            print("password " + str(password))
        for i_1 in credentials:
            user = driver.find_element_by_id(str(i_0))
            user.click()
            i_0 += 1
            user.send_keys(i_1)
            if self.SHOW_LOG:
                print(i_1)
        user = driver.find_element_by_id(str(i_0))
        user.click()
        i_0 += 1
        user.send_keys(password)
        user = driver.find_element_by_id(str(i_0))
        user.click()
        time.sleep(4)
        try:
            tester = driver.find_element_by_id("name_ph")
        except:
            raise self.FailTest("login nije uspjesno napravljen")
        self.assertTrue(True)
    except self.FailTest as e:
        print(e)
        self.assertTrue(False)
    except:
        import sys
        print(sys.exc_info()[0])
        self.assertTrue(False)
```

Slika 5.18: Prikaz testa za registraciju tvrtke



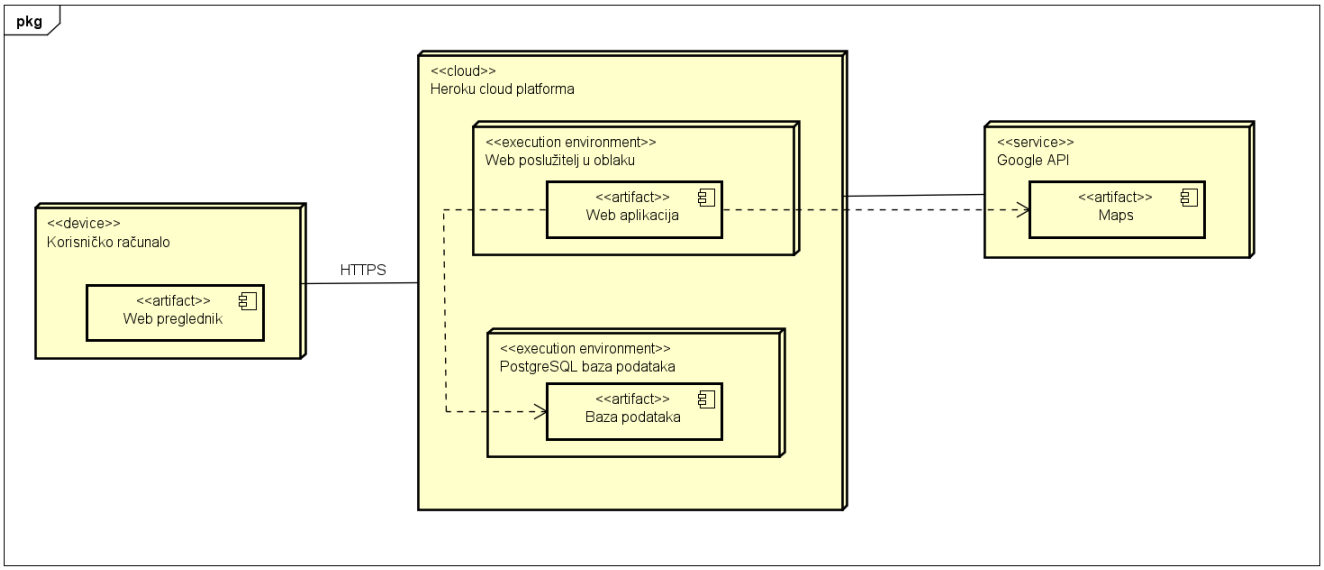
Slika 5.19: Prikaz rezultata nakon provedenog ispitivanja



Slika 5.20: Prikaz primjera rezultata u slučaju da očekujemo rezultat drugačiji od dobivenog (provedeni test je prikazani)

5.3 Dijagram razmještaja

Dijagram razmještaja je strukturni UML dijagram koji opisuje topologiju sustava i prikazuje odnos između sklopovskih i programskih dijelova. Poslužiteljsku stranu predstavlja *cloud* platforma Heroku. Na Heroku se nalaze web poslužitelj i poslužitelj baze podataka, a pristupa kartama omogućen je preko *Google API* servisa. Na klijentskoj strani koristi se web preglednik kojim se pristupa web aplikaciji. Komunikacija između korisnika i oblaka odvija se putem HTTPS veze.



Slika 5.21: Dijagram razmještaja

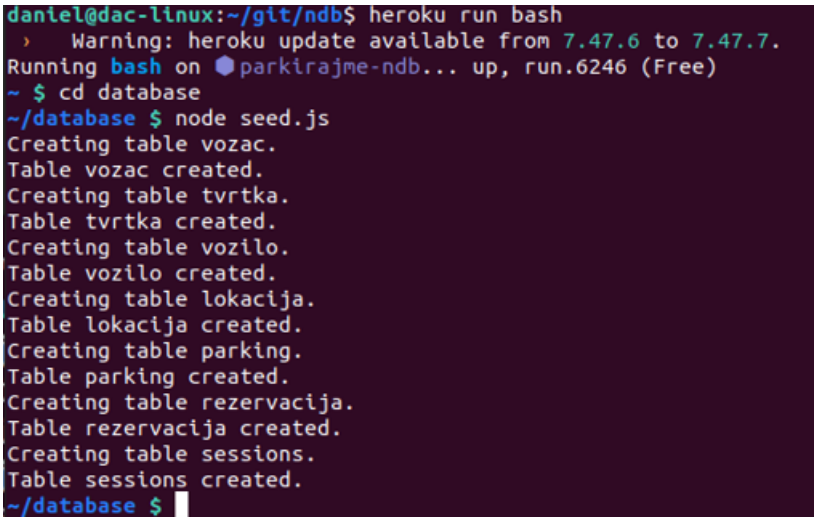
5.4 Upute za puštanje u pogon

Pokretanje Stranice preko Heroku-a:

- *Prvo se treba ulogirati na Heroku*
- *Nakon logina treba se povezati postojeći git repozitorij sa Heroku-om pozivanjem "heroku git:remote -a imestranice"*
- *Nakon povezivanja repozitorij se može pushati sa "git push heroku master" u slučaju da se pusha sa Master brancha ili "git push heroku imebrancha:master"*
- *Nakon što se repozitorij pusha na Heroku, potrebno je napraviti bazu podataka.*
- *Na Resources stranici Heroku-a kao add-on se doda Heroku Postgres te se iz dane baze trebaju uzeti podatci za spajanje*
- *Nakon toga se u konzoli pozove "heroku run bash" kojim se otvori bash te se u bashu pokrene seed.js kako bi se napravila baza podataka*
- *Kada je to napravljeno, web stranica se može pokrenuti*

```
daniel@dac-linux:~/git/ndb$ heroku git:remote -a parkirajme-ndb
set git remote heroku to https://git.heroku.com/parkirajme-ndb.git
daniel@dac-linux:~/git/ndb$ git add .
daniel@dac-linux:~/git/ndb$ git commit -am "heroku"
On branch devdoc2
Your branch is up to date with 'origin/devdoc2'.

nothing to commit, working tree clean
daniel@dac-linux:~/git/ndb$ git push heroku devdoc2:master
Enumerating objects: 10390, done.
Counting objects: 100% (10390/10390), done.
Delta compression using up to 4 threads
Compressing objects: 100% (7053/7053), done.
Writing objects: 100% (10390/10390), 64.93 MiB | 463.00 KiB/s, done.
Total 10390 (delta 4296), reused 6971 (delta 2890)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Building on the Heroku-20 stack
remote: -----> Node.js app detected
remote:
```

Slika 5.22: Upravljanje Heroku-om



Slika 5.23: Kreiranje baze podataka

6. Zaključak i budući rad

Izrada aplikacije je započeta analiziranjem zadanih uvjeta te pretakanjem korisničkih zahtjeva u formalne zapise uvjeta. Prilikom popisivanja zahtjeva dodani su zahtjevi koje smo smatrali da bi bili od koristi korisnicima. Kasnije se pokazalo da to nije bio dobar potez.
Nakon formaliziranja zahtjeva korisnika izrađen je kostur aplikacije te su isprobane jednostavne funkcije.
Ovime je završila prva faza projekta. Dobar potez je bio što smo simultano radili na aplikaciji i usklađivali zahtjeve u dokumentaciji. Loš potez je bio što smo pri izradi kostura krenuli s izradom CSS-a za stranicu što sada smatramo da je trebalo napraviti na kraju. Problem nam je predstavljalo nepoznavanje tehnologije git zbog čega smo puno vremena gubili na raznim konfliktima. Prvi put smo se susreli s Google maps API-jem te je bilo potrebno neko vrijeme da se naviknemo na rad s njim; sada se osjećamo ugodno raditi s njim.

U drugoj fazi projekta smo dovršili aplikaciju i kod. Puno smo se sigurnije osjećali eksperimentirati s tehnologijama pošto smo stekli iskustvo. Ovo je vrlo vjerojatno iz razloga što nismo odlagali izradu za pred rok puštanja u pogon već smo radili kontinuirani pa smo imali vremena i popraviti ako nešto krivo napravimo.

Kako je kod aplikacije postao opsežniji tako smo počeli imati sve više konflikata koje smo rješavali tako što smo prestali u isto vrijeme raditi na istim datotekama. Razlog velikog broja konflikata i gubljenja vremena je što su se često implementirale funkcionalnosti koje su loše dokumentirane i jer nismo slijedili dobre prakse programiranja. Smatrali smo kako naše male promijene u kodu su zanemarive te da ih nije potrebno strukturirati. Iz ovoga smo zaključili kako je bitno od početka dobro strukturirati i dokumentirati kod. Najveća korist koju smo izvukli iz ovog projekta je praksa u zajedničkom radu. Ovo je prvi put da smo morali napraviti zajedno projekt srednje veličine te smatramo to dragocjenim iskustvom. Također smo naučili raditi s novim tehnologijama te podsjetili se nekih stari i produbili svoje znanje o njima.

Zaključili smo kako postoji puno nejednoznačnosti u zahtjevima naručitelja te smo naučili važnost ispravnog i pravovremenog komuniciranja. Valja istaknuti kako je od velike koristi bilo od početka razrješiti što više nedoumica jer kako smo napredovali s izradom aplikacije to je bilo teže prilagoditi kod i dokumentaciju željama naručitelja aplikacije.

Cijeli proces bi bio puno brži i ugodniji da smo već radili kao tim te da je svaki član tima imao specijalno znanje u određenoj tehnologiji, a ne da svaki član radi malo u svakoj tehnologiji. Unatoč neekspertnosti smo se s vremenom podijelili na različite tehnologije te time znatno povećali produktivnost i smanjili konflikte.

Na kraju smo zaključili da je bitnije napraviti što bolje zahtjeve naručitelje nego pokušati dodati nove funkcionalnosti ali izgubiti na kvaliteti koda. Sve funkcionalnosti koje smo mislili implementirati smo implementirali, iznimka su one funkcionalnosti koje smo mislili na početku da ih ima smisla raditi ali se ispostavilo da nisu potrebne (na primjer postojao je use case za brisanje automobila, dodavanje automobila i za uređivanje automobila, posljednji je beskoristan jer se njegova funkcionalnost može riješiti s prva dva).

Zahvalni smo profesorima koji su nas usmjeravali prema pravom putu pri izradi aplikacije i odgovarali na sva pitanja koja smo imali.

Voljeli bi jednoga dana nastaviti raditi na ovom projektu ili sličnom.

NDB

Popis literature

Kontinuirano osvježavanje

1. Programsko inženjerstvo, FER ZEMRIS, http://www.fer.hr/predmet/proinz
2. I. Sommerville, "Software engineering", 8th ed, Addison Wesley, 2007.
3. T.C.Lethbridge, R.Langaniere, "Object-Oriented Software Engineering", 2nd ed. McGraw-Hill, 2005.
4. I. Marsic, Software engineering book", Department of Electrical and Computer Engineering, Rutgers University, http://www.ece.rutgers.edu/~marsic/books/SE
5. The Unified Modeling Language, https://www.uml-diagrams.org/
6. Astah Community, http://astah.net/editions/uml-new

Indeks slika i dijagrama