Univerzitet u Novom Sadu,
Fakultet tehničkih nauka

SEMINARSKI RAD

Nastavni predmet: Tehnologije i sistemi eUprave
Naziv teme: MUP - Vozila

Student,
Anastasija Čukelj SR 2/2021

Jun, 2024

1. Sažetak 3
2. Ključne reči 3
3. Uvod 4
4. Srodna istraživanja 5
5. Korišćene tehnologije 6
5.1 Go programski jezik 6
5.2 Angular 7
5.3 Docker 8
5.4 Single Sign-On (SSO) 9
6. Specifikacija zahteva 10
6.1 Specifikacija funkcionalnih zahteva 11
6.2 Specifikacija nefunkcionalnih zahteva 16
7 Specifikacija dizajna 17
8. Implementacija sistema 19
9. Demonstracija 39
10 Literatura 47

1. Sažetak

U ovom radu opisan je servis za vozila, pri ministarstvu unutrašnjih poslova, koji omogućava kreiranje vozačkih dozvola, vozača, registraciju vozila, pretragu istih i generisanje izveštaja po određenim kriterijumima. Ovo softversko rešenje koriste policajci. Za realizaciju projekta korišćen je Go programski jezik [1] na beckend-u, Angular [2] na frontend-u, dokerizacija [3], single sign-on za autentifikaciju korisnika, kao i mikroservisna arhitektura zbog komunikacije sa drugim servisima.

2. Ključne reči

- mup
- vozila
- mikroservisi
- veb
- pretraga
- izveštaj

3. Uvod

U savremenom dobu, efikasno upravljanje informacijama je ključ rada svake organizacije, a naročito državnih institucija kao što je **Ministarstvo unutrašnjih poslova (MUP)**. Tradicionalni monolitni pristup sve više zamenjuje mikroservisna arhitektura, koja omogućava lakše upravljanje informacijama, kao i skalabilnije, fleksibilnije i otpornije sisteme.

Mikrosrevisna arhitektura se u ovom slučaju pokazala kao odlično rešenje za upravljanje podacima koji teku kroz više servisa, naročito ako imamo sistem gde MUP komunicira sa saobraćajnom policijom, kao u našem slučaju.
Ovaj seminarski rad ima za cilj da objasni korišćenje mikroservisa unutar MUP-a, koji je osmišljen za čuvanje i upravljanje podacima o vozilima, vozačima i vozačkim dozvolama. Detaljno ćemo analizirati i prikazati funkcionalnosti ovog mikroservisa, zajedno sa neophodnom dijagramima i slučajevima korišćenja.

Pre razvoja savremenih tehnologija, ovi podaci su se čuvali kroz papirnu dokumentaciju, i samim tim unosili ručno, što je često rezultiralo sporim protokom informacija, greškama, gubljenjem podataka i rizicima od gubitka ovakvih podataka.

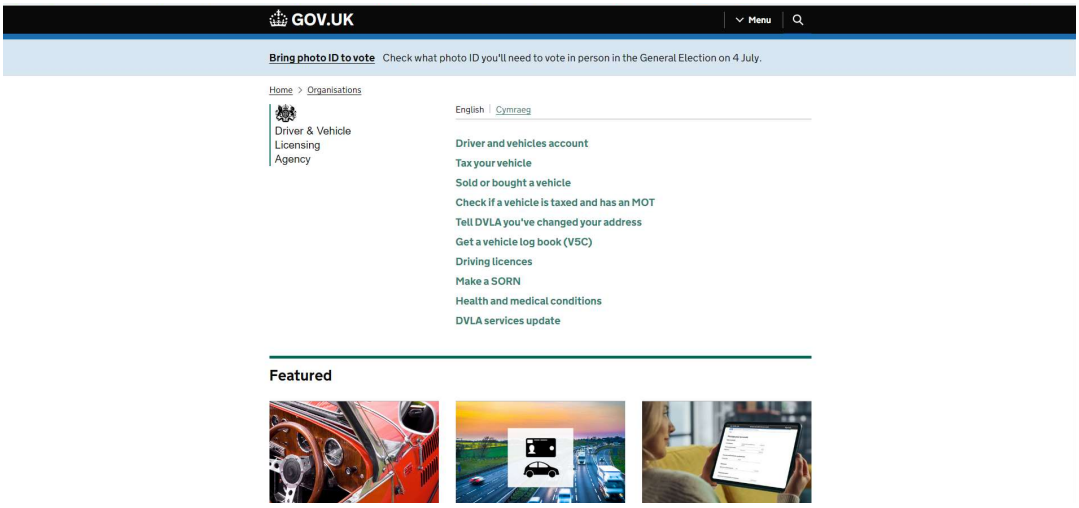
Ostatak ovog rada je je objašnjen u nastavku. U četvrtom poglavlju su opisana srodna istraživanja. Korišćene tehnologije su opisane u petom,a specifikacija zahteva za sistem u šestom. Specifikacija dizajna sistema je opisana u sedmom poglavlju. Osmo opisuje implementaciju rešenja. Demonstracija funkcionalnosti data je u devetom poglavlju, a zaključak rada je dat u pretposlednjem poglavlju. Poslednje, odnosno jedanaesto poglavlje sadrži literaturu korišćenu za izradu ovog rada.

4. Srodna istraživanja

U ovom poglavlju data su istraživanja organizacija koja na sličan način prikupljaju i menjaju podatke, kao i tehnologije koje omogućavaju isto.

Department of Motor Vehicles (DMV) u Sjedinjenim Američkim Državama [4] : DMV je državna institucija u SAD-u odgovorna za registraciju vozila i izdavanje vozačkih dozvola. Takođe građanima pruža usluge kao što su inspekcija vozila, evidencija prekršaja u saobraćaju i izdavanje identifikacionih kartica.
Na veoma sličan način upravlja podacima o vozilima, vozačima i dozvolama. Ovakvi sistemi često uključuju mikroservisnu arhitekturu kako bi se ogromna količina podataka obradila što efikasnije. Takođe, ovaj pristup pruža brzu pretragu i ažuriranje informacija, kao i veću sigurnost.

Driver and Vehicle Licensing Agency (DVLA) u Ujedinjenom Kraljevstvu [5]: DLVA takođe upravlja podacima o vozilima i vozačima unutar Ujedinjenog Kraljevstva. Arhitektura ovog sistema je takođe mikroservisna. Pruža integraciju različitih usluga kao što su online prijave za vozačke dozvole, registracija vozila i pretraga podataka.



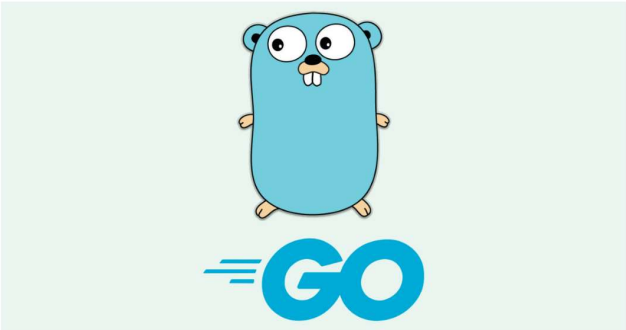
Slika 1 - početna stranica DLVA agencije

5. Korišćene tehnologije

5.1 Go programski jezik

Go [1] je programski jezik otvorenog koda nastao u Guglu 2007. godine. Osmišljen je od strane Roberta Grisimera, Roba Pajka i Ken Tomposn-a. Jezik je kompajliran, sa statičkim tipovima podataka, strukturnim tipovima i sadrži automatsko upravljanje memorijom. Ovaj jezik je prvenstveno bio namenjen sistemskom programiranju.
Početak rada na jeziku je septembar 2007. godine, a zvanično je promovisan 2009. godine. Postoje dve glavne implementacije, gc koja je glavna implementacija koju je razvio Gugl, gcgcgo iz GNU kolekcije kompajlera. Najveći uticaj na ovaj jezik su jezici iz porodice C i Paskal.

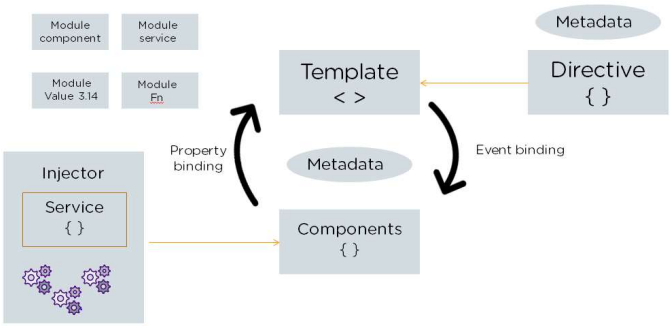
Danas, ovaj jezik je široko prihvaćen u industriji zbog svoje jednostavnosti, efikasnosti i podrške za konkurentno programiranje. Koristi se za web aplikacije, mrežne alate, distribuirane sisteme i mikroservise. Neke od poznatih kompanija koje ga koriste su Dropbox, Netflix, Uber, Docker i mnoge druge. Često je izbor u DevOps okruženjima, zbog dobre podloge za razvoj infrastukture i efikasno upravljanje Cloud-om.



Slika 2 - dabar koij je simbol Go programskog jezika

5.2 Angular

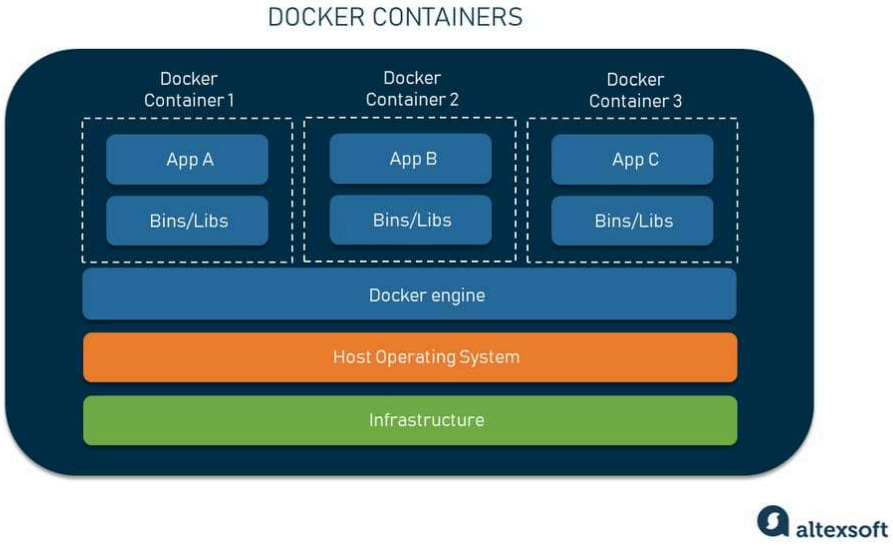
Angular [2] je framework otvorenog koda napravljen od strane Google-a. Prvobitno je nastao kao AngularJS 2010. godine, a 2016. godine je doživeo dosta promena i preimenovan je u Angular. Ovom promenom je zadovoljio moderne standarde za razvoj web aplikacija. Koristi TypeScript kao osnovnih jezik, tako da pruža snažnu tipizaciju. Pruža ugrađene funkcije za upravljanje stanjem, rutiranje, forme, komunikaciju sa bekendom. što ga čini jednim od najboljih rešenja za razvoj web aplikacija.
Koristi se u mnogim industrijama za kreiranje robusnih korisničkih interfejsa kao što su Google, Microsoft, IBM, Upwork i tako dalje.
Osnovni gradivni blok svake Angular aplikacija je komponenta, koja se sastoji od HTML stranica, CSS stilova i poslovne logike unutar TypeScript-a. Komponente se mogu ugnježđavati, što omoguće kreiranje kompleksnih korisničkih interfejsa.



5.3 Docker

Docker [3] je platforma za brzo kreiranje, testiranje i pokretanje aplikacija. Osnovne stvari kojima docker rukuje su kontejneri (slika 3), odnosno jedinice koje sadrže sve što je softveru potrebno za rad, kao što su biblioteke, runtime okruženje i sistemski alati.

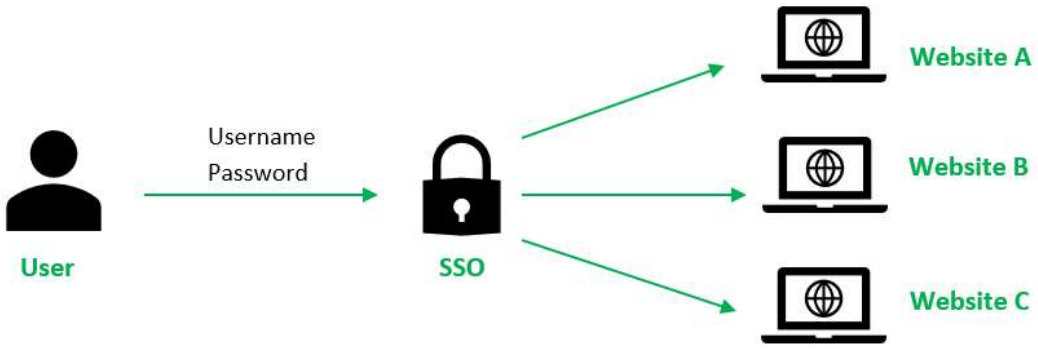
Docker se pojavio prvi put 2013. godine i razvila ga je istoimena kompnaija. Postao je ključni alat u softverskom inženjerstvu zbog toga što pojednostavljuje testiranje, razvoj i implementaciju putem kontejnerizacije. Naročito se koristi u mikroservnisim arhitekturama, kakva je i u implementaciji ovog seminarskog rada.



Slika 3 - arhitektura Docker-a

5.4 Single Sign-On (SSO)

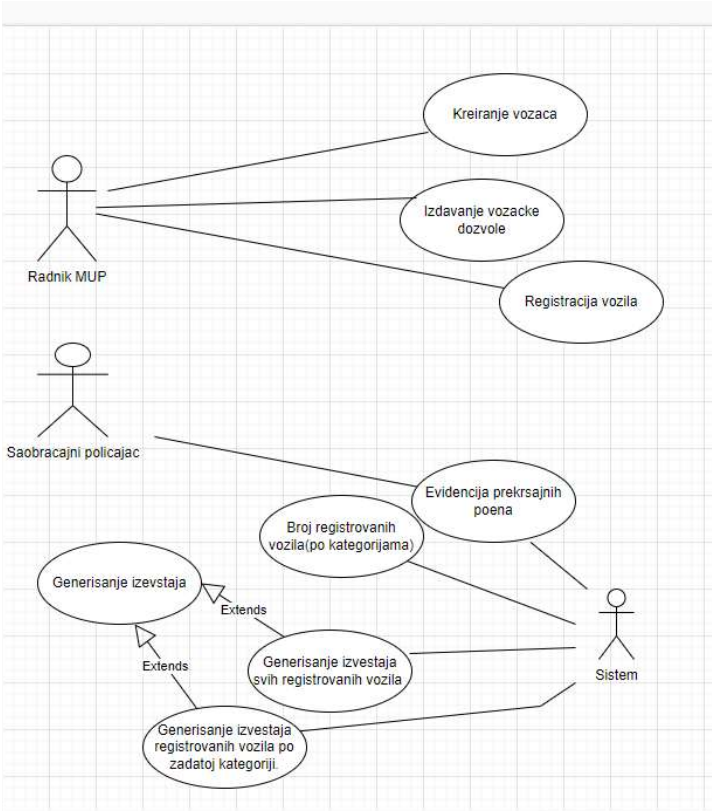
Single sign-on je metoda za autentifikaciju korisnika i sesija, koja omogućava korisniku da koristi određene kredencijale, na primer korisničko ime i lozinku, kako bi mogao da pristupi skupu aplikacija. Ovim se smanjuje potrebu za korišćenjem više lozinki. Šematski prikaz nalazi se na slici 4.
U implementaciji ovog projekta, policajac pored servisa za vozila unutar MUP-a, ima pristup i statističkim podacima, odnosno u isto vreme vidi ono što i običan građanin, što je primer SSO-a.



Slika 4 - SSO

6. Specifikacija zahteva

U ovom poglavlju objašnjeni su funkcionalni i nefunkcionalni zahtevi koje ima servis za vozila unutar MUP-a. Funkcionalni zahtevi prikazani su UML use case dijagramom (slika 5).



Slika 5

6.1 Specifikacija funkcionalnih zahteva

Slučajevi korišćenja

Tabela 1 prikazuje opis slučaja korišćenja *Kreiranje vozača.*

Naziv	Kreiranje vozača
Učesnici	Radnik MUP-a
Preduslovi	Policajac je pristupio sistemu (sistem je funkcionalan)
Koraci	1. Policajac je pristupio sistemu 2. Policajac unosi podatke i kreira vozača
Rezultat	Novi vozač je registrovan u sistem
Izuzeci	1.Došlo je do greške sa serverske strane 2.JMBG nije validan. 3.Ime i prezime nisu validni. 4.Datum rođenja je u budućnosti.

Tabela 1 - Opis slučaja korišćenja *Kreiranja vozača*

Tabela 2 prikazuje opis slučaja korišćenja *Izdavanje vozačke dozvole.*

Naziv	Izdavanje vozačke dozvole
Preduslovi	1.Vozač postoji u sistemu. 2.Sistem je funkcionalan i policajac ima pristup sistemu
Učesnici	Policajac
Rezultat	Nova vozačka dozvola je kreirana
Koraci	1. Radnik MUP-a pristupa sistemu. 2. Kreira se vozačka dozvola.
Izuzeci	1.Vozačka dozvola pod unetim jedinstvenim matičnim brojem već postoji. 2.Serverska greška 3. Vozač je imao prekršaj za vožnju pod dejstvom alkohola. 4. Nevalidan JMBG

Tabela 2 - opis slučaja korišćenja *Izdavanje vozačke dozvole*

Tabela 3 prikazuje opis slučaja korišćenja *Registracija vozila.*

Naziv	Registracija vozila
Preduslovi	1.Radnik MUP-a ima pristup servisu. 2.Vozač postoji u sistemu.
Učesnici	Policajac
Koraci	1.Radnik MUP-a pristupa servisu. 2. Kreira se novo vozilo.
Rezultat	Novo vozilo je registrovano.
Izuzeci	1.JMBG koji je unet ne postoji ili nije validan. 2. Forma registarskih tablica nije validna.

Tabela 3 - opis slučaja korišćenja *Registracija vozila*

Tabela 4 prikazuje opis slučaja korišćenja *Evidencija prekršajnih poena.*

Naziv	Evidencija prekršajnih poena
Preduslovi	1.Sistem je funkcionalan 2. Prekršaj je evidentiran unutar sistema saobraćajne policije. 3.Saobraćajni policajac ima pristup ruti za izmenu prekršajnih poena unutar MUP sistema
Učesnici	Saobraćajni policajac, Sistem
Koraci	1.Prekršaj je kreiran u servisu saobraćajne policije. 2. MUP servis pronalazi vozača po JMBG-u 3. Saobraćajni policajac ažurira prekršajne poene vozača, neposredno nakon što unese prekršaj u svoj sistem
Rezultat	Prekršajni poeni su uneti.
Izuzeci	1.JMBG iz servisa saobraćajne policije ne postoji u MUP-u

Tabela 4 - opis slučaja korišćenja *Evidencija prekršajnih poena*

Tabela 5 prikazuje opis slučaja korišćenja *Generisanje izveštaja svih registoravnih vozila.*

Naziv	Generisanje izveštaja svih registrovanih vozila.
Preduslovi	1.Sistem je funkcionalan i sadrži sva registrovana vozila
Učesnici	Sistem, policajac
Koraci	1.Radnik MUP-a pristupa sistemu. 2. Otvara stranicu sa svim registrovanim vozilima. 3.Klik na dugme *Generiši izveštaj* 4. Izveštaj se čuva u fajl sistem
Rezultat	Izveštaj je sačuvan na računaru
Izuzeci	1.Greška prilikom generisanja PDF-a. 2. Stranica za dobavljanje svih registrovanih vozila ne radi. 3.Sistem ne radi

Tabela 5 - opis slučaja koriščenja *Generisanje izveštaja svih registrovanih vozila*

Tabela 6 prikazuje opis slučaja korišćenja *Generisanje izveštaja o registrovanim vozilima po određenoj kategoriji.*

Naziv	Generisanje izveštaja registrovanih vozila po kategoriji.
Preduslovi	1.Sistem je funkcionalan i sadrži sva registrovana vozila
Učesnici	Sistem, policajac
Koraci	1.Radnik MUP-a pristupa sistemu. 2. Otvara stranicu sa svim registrovanim vozilima. 3.Biranje kategorije po kojoj će se izveštaj generisati 4.Klik na dugme *Generiši izveštaj* 5. Izveštaj se čuva u fajl sistem
Rezultat	Izveštaj je sačuvan na računaru
Izuzeci	1.Greška prilikom generisanja PDF-a. 2. Stranica za dobavljanje svih registrovanih vozila ne radi. 3.Sistem ne radi

Tabela 6 - opis slučaja korišćenja *Generisanje izveštaja o registrovanim vozilima po određenoj kategoriji*

Tabela 7 prikazuje opis slučaja korišćenja *Broj registrovanih vozila po kategorijama*

Naziv	Broj registrovanih vozila po kategorijama
Preduslovi	1.Sistem je funkcionalan i sadrži sva registrovana vozila
Učesnici	Sistem
Koraci	1.Radnik MUP-a pristupa sistemu. 2. Otvara stranicu sa svim registrovanim vozilima. 3.Na stranici sa registrovanim vozilima prikazan je broj svih vozila po kategorijama.
Rezultat	Prikaz broja vozila po svakoj kategoriji na stranici
Izuzeci	1. Stranica za dobavljanje svih registrovanih vozila ne radi. 2.Sistem ne radi

Tabela 7 - opis slučaja korišćenja *Broj registrovanih vozila po kategorijama*

6.2 Specifikacija nefunkcionalnih zahteva

Nefunkcionalni zahtevi u aplikaciji se odnose na kriterijume koji se koriste za određivanje performansi sistema, ali nisu direktno vezani za implementaciju sistema. Ovi parametri su veoma važni za zadovoljstvo korisnika. Najčešći nefunkcionalni zahtevi su efikasnost, brzina, pouzdanost, skalabilnost i tako dalje. U nastavku su opisani neki nefunkcionalni zahtevi vezani za datu aplikaciju eUprave.

1.Dokerizacija

Aplikacija je mikroservisne arhitekture, gde svaki mikroservis ima svoj kontejner. Za ovo se koristi Docker Compose alat.

2. Otpornost na otkaze pojedinačnih servisa

Ukoliko jedan servis iz bilo kog razloga prestane sa radom, ostali će nastaviti da funkcionišu. Ovo obezbeđuje upravo doker i njegovi kontejneri (Listing 1).

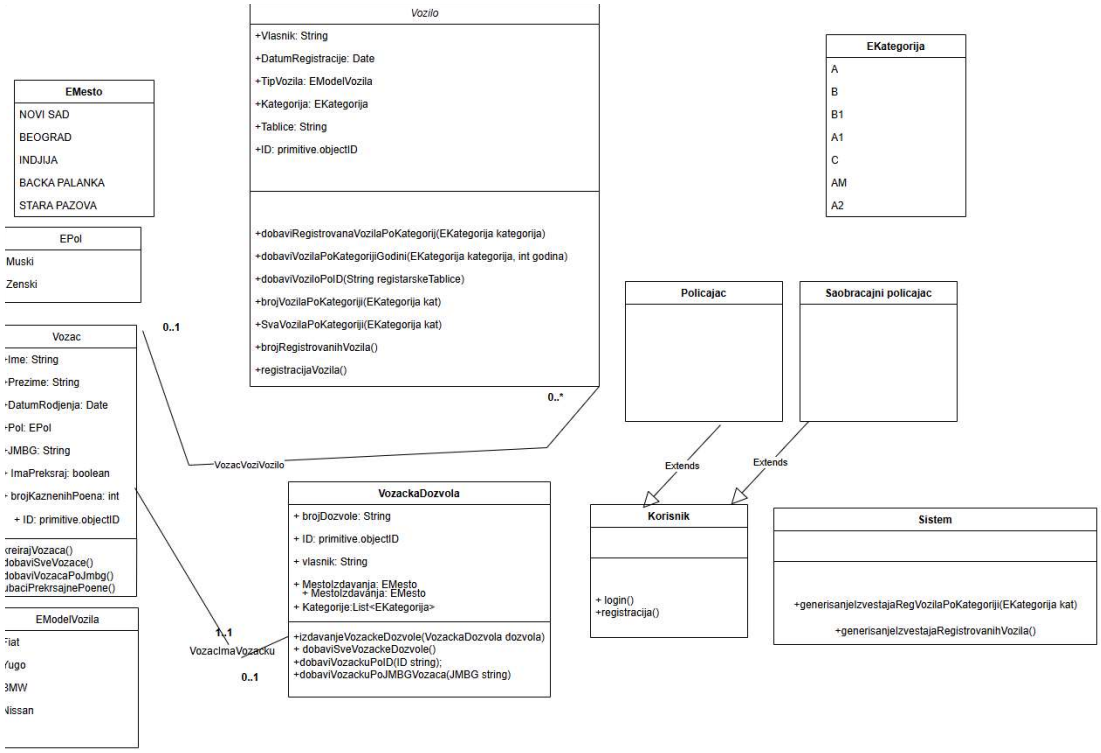
```
resp, err := s.performAuthorizationRequestWithContext( method: "GET", ctx, token, url)
if err != nil {
    if ctx.Err() == context.DeadlineExceeded {
        errorMsg := map[string]string{"error": "Authorization service is not available."}
        errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
        return
    }
    errorMsg := map[string]string{"error": "Error performing authorization request."}
    errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
    return

}
defer resp.Body.Close()
```

Listing 1 - deo koda koji rukuje parcijalnim otkazima sistema

7 Specifikacija dizajna

Aplikacija je implementirana po mikroservisnoj arhitekturi - svaki podsistem eUprave je zaseban servis koji ima komunikaciju sa barem jednim drugim servisom. Pored MUP-a, koji je fokus ovog seminarskog rada, u aplikaciji imamo i saobraćajnu policiju, prekršajni sud, zavod za statistiku i autorizacioni servis. Zavod za statistiku takođe dobavlja sve neophodne podatke od MUP-a. MUP komunicira sa saobraćajnom policijom, jednom pri izdavanju vozačke dozvole gde proverava da li određeni korisnik ima istoriju prekršaja sa vožnjom pod dejstvom alkohola, a drugi put pri upisu prekršajnih poena koji dolaze iz saobraćajne policije.
Komponente sistema unutar MUP-a:
1. Korisničiki interfejs (Frontend) - Angular [2] aplikacija, preko koje policajac,a po potrebi i saobraćajni policajac pristupaju sistemu kroz web pretraživač.
2. Servisni sloj (Backend) - poslovna logika implementirana kroz Go [1] programski jezik, a koja u sebi sadrži sledeće komponente: servisni sloj, model podataka, handler-e za rukovanje http zahtevima, rutiranje i vraćanje odgovarajućih grešaka.
3.Baza podataka - Mongo [6] noSQL baza podataka gde se čuvaju podaci o vozačima, vozačkim dozvolama i vozilima (registrovanim i neregistrovanim).

Model podataka prikazan je na sledečem dijagramu:



Slika 6 - klasni dijagram

Na dijagramu se nalaze sledeći entiteti:
Vozilo - sadrži string vlasnika (JMBG), broj registarskih tablica, model vozila, datum registracije i kategoriju. Jedan vozač može posedovati više vozila, a jedno vozila pripada tačno jednom vozaču, što se iz priloženog dijagrama vidi. Možemo kreirati novo vozilo (registrovati ga), dobaviti sva vozila, dobaviti sva registrovana vozila, prebrojati vozila po kategoriji, dobaviti vozilo po ID-ju, dobaviti ih po kategoriji i godini, i dobaviti sva registorvana vozila po kategoriji.
Vozačka dozvola - u sebi ima string vlasnika (JMBG), broj vozačke dozvole koji je UUID, mesto izdavanja i listu kategorija. Možemo kreirati vozačku dozvolu, dobaviti sve dozvole, dobaviti jednu po ID-ju, kao i po JMBG-u vozača.
Vozač - sadrži JMBG, ime, prezime, datum rođenja, boolean polje koje označava ima li prekršaj, broj kaznenih poena i pol. Možemo kreirati vozača, dobaviti sve vozače, dobaviti sve vozače po ID-ju i uneti prekršajne poene (pri unosi prekršajnih poena, komunicira se sa saobraćajnom policijom).
Pol, kategorija, mesto i **model** vozila predstavljeni su enumeracijama.
Policajac koji koristi ovaj servis je korisnik koji je definisan u servisu za autorizaciju, a u ovom servisu uloga koja je neophodna jeste policajac.
Saobraćajni policajac je uloga iz servisa saobraćajne policije, koja ima pristup jednom API-ju iz ovog servisa. Zato je naveden u klasnom dijagramu.
Policajac i saobraćajni policajac nasleđuju klasu korisnika.
Sistem predstavlja MUP sistem koji u pozadini radi određene operacije kao što je generisanje izveštaja.

8. Implementacija sistema

U ovom poglavlju biće opisane sve funkcionalnosti sistema koje su navedene unutar use case i klasnog dijagrama.
Komunikacija teče od handler-a do servisa koji izvršavaju poslovnu logiku. Handler klase služe za rukovanje HTTP zahtevima. Svaki servis ima svoju implementaciju, i svaka komponenta (vozači, vozačke dozvole i vozila) ima svoju klase za rutiranje, gde se definiše koji API gađa koji url, sa određenom metodom.

Implementacija kreiranja vozačke dozvole

U nastavku je data handler funkcija koja kreira vozačku dozvolu

```
func (s *DriverLicenceHandler) CreateDriverLicence(c *gin.Context) {
   rw := c.Writer
   h := c.Request

   token := h.Header.Get("Authorization")
   url := "http://auth-service:8085/api/users/currentUser"
   timeout := 5 * time.Second
   ctx, cancel := context.WithTimeout(context.Background(), timeout)
   defer cancel()

   resp, err := s.performAuthorizationRequestWithContext("GET", ctx, token,
url)
   if err != nil {
      if ctx.Err() == context.DeadlineExceeded {
         errorMsg := map[string]string{"error": "Authorization service is
not available."}
         errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
         return
      }
      errorMsg := map[string]string{"error": "Error performing
authorization request."}
      errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
      return
   }
   defer resp.Body.Close()

   statusCode := resp.StatusCode
   if statusCode != 200 {
      errorMsg := map[string]string{"error": "Unauthorized."}
      errorMessage.ReturnJSONError(rw, errorMsg, http.StatusUnauthorized)
      return
   }

   decoder := json.NewDecoder(resp.Body)
```

```
   var responseUser struct {
      LoggedInUser struct {
         username string        `json:"username"`
         email    string        `json:"email"`
         UserRole data.UserRole `json:"userRole"`
      } `json:"user"`
   }

   if err := decoder.Decode(&responseUser); err != nil {
      errorMsg := map[string]string{"error": "User object was not valid."}
      errorMessage.ReturnJSONError(rw, errorMsg, http.StatusUnauthorized)
      return
   }

   if responseUser.LoggedInUser.UserRole != data.Policeman {
      errorMsg := map[string]string{"error": "Unauthorized. You are not a
policeman."}
      errorMessage.ReturnJSONError(rw, errorMsg, http.StatusUnauthorized)
      return
   }

   driverLicence, exists := c.Get("driverLicence")
   if !exists {
      errorMsg := map[string]string{"error": "Driver licence object was not
valid"}
      errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
      return
   }

   driverLicenceInsert, ok := driverLicence.(domain.DriverLicenceCreate)
   vehicleDriverId := driverLicenceInsert.VehicleDriver
   vehicleDriver, _ :=
s.driverService.GetVehicleDriverByID(vehicleDriverId, ctx)

   existingLicence, err :=
s.service.GetDriverLicenceByDriver(vehicleDriverId, ctx)

   if existingLicence != nil {
      errorMsg := map[string]string{"error": "Licence for this driver
already exists."}
      errorMessage.ReturnJSONError(rw, errorMsg, http.StatusConflict)
      return
   }

   if vehicleDriver == nil {
      errorMsg := map[string]string{"error": "There's no driver with that
ID in database."}
      errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
      return
   }

   driverLicenceInsert, ok = driverLicence.(domain.DriverLicenceCreate)
   if !ok {
```

```
      errorMsg := map[string]string{"error": "Invalid type for driver
licence."}
      errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
      return
   }

   url = "http://police-service:8084/api/delict/get/delictType/Voznja pod
uticajem alkohola"
   delictResp, err := s.performAuthorizationRequestWithContext("GET", ctx,
token, url)
   if err != nil {
      if ctx.Err() == context.DeadlineExceeded {
         errorMsg := map[string]string{"error": "Authorization service is
not available."}
         errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
         return
      }
      fmt.Println(err)
      errorMsg := map[string]string{"error": "Failed to check delicts."}
      errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
      return
   }
   defer delictResp.Body.Close()

   statusCode = delictResp.StatusCode
   if statusCode != 200 {
      errorMsg := map[string]string{"error": "Wrong delict type data."}
      errorMessage.ReturnJSONError(rw, errorMsg, http.StatusUnauthorized)
      return
   }

   decoderDelict := json.NewDecoder(delictResp.Body)

   if delictResp.StatusCode == http.StatusOK {
      var delicts []map[string]interface{}
      if err := decoderDelict.Decode(&delicts); err != nil {
         errorMsg := map[string]string{"error": "Failed to decode
delicts."}
         errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError)
         return
      }

      for _, delict := range delicts {
         fmt.Println(delicts)
         fmt.Println("Delicts")

         driverID, ok := delict["driver_identification_number"].(string)
         if !ok {
            continue
         }
         if driverID == driverLicenceInsert.VehicleDriver {
```

```
            errorMsg := map[string]string{"Greska": "Vozac ima prekrsaj
vezan za voznju pod dejstvom alkohola. Nije moguce izdati vozacku
dozvolu!"}
            errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusBadRequest)
            return
         }
      }
   }

   driverLicenceInsertDB, _, err :=
s.service.InsertDriverLicence(&driverLicenceInsert)
   if err != nil {
      errorMsg := map[string]string{"error": "Database problem."}
      errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
      return
   }

   rw.WriteHeader(http.StatusCreated)
   jsonResponse, err1 := json.Marshal(driverLicenceInsertDB)
   if err1 != nil {
      errorMessage.ReturnJSONError(rw, fmt.Sprintf("Error marshaling JSON:
%s", err1), http.StatusInternalServerError)
      return
   }
   rw.Write(jsonResponse)
}
```

Listing 2 - handler funkcija za kreiranje vozačke dozvole

Ova funkcija prvo poziva servis za autorizaciju, koji proverava da li je token poslat unutar header-a validan. Ako jeste, JSON tog korisnika se vraća, iz njega se čitaju podaci i proverava rola korisnika. Ukoliko ovo prođe uspešno, nastavljamo na kreiranje vozačke dozvole. Vozačka dozvola se može kreirati samo ukoliko vozač postoji u sistemu, i ukoliko nema prekršaj sa vožnjom pod dejstvom alkohola.

Dobavljanje svih vozačkih dozvola

Autorizacija u ovom funkciji i svakoj sledećoj funkciji je potpuno ista kao i u prethodnom listingu. Shodno tome, u sledećem listingu taj deo je izostavljen radi jednostavnosti.
Kada dobavljamo sve vozačke dozvole, handler poziva servisni sloj koji komunicira direktno sa bazom.

```
vehicles, err := s.service.GetAllDriverLicences()
if err != nil {
   errorMsg := map[string]string{"error": "Failed to retrieve licences from
the database."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError)
   return
}

jsonResponse, err := json.Marshal(vehicles)
if err != nil {
   errorMsg := map[string]string{"error": "Error marshaling JSON."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError)
   return
}

rw.Header().Set("Content-Type", "application/json")
rw.WriteHeader(http.StatusOK)
rw.Write(jsonResponse)
```

Listing 3 - dobavljanje svih vozačkih dozvola

Dobavljanje vozačke dozvole po ID-ju

```
driverLicenceID := c.Param("id")
vehicle, err := s.service.GetDriverLicenceById(driverLicenceID, ctx)
if err != nil {
   errorMsg := map[string]string{"error": "Failed to retrieve driver
licence from the database. No such licence."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError)
   return
}
jsonResponse, err := json.Marshal(vehicle)
if err != nil {
   errorMsg := map[string]string{"error": "Error marshaling JSON."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError)   return }
rw.Header().Set("Content-Type", "application/json")
rw.WriteHeader(http.StatusOK)
rw.Write(jsonResponse)
```

Listing 4 - dobavljanje vozačke dozvole po ID-ju

Dobavljanje vozačke dozvole po JMBG-u vozača

```
driverID := c.Param("id")
   vehicle, err := s.service.GetDriverLicenceByDriver(driverID, ctx)
   if err != nil {
      errorMsg := map[string]string{"error": "Failed to retrieve driver
licence from the database. No such licence."}
      errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError)
      return
   }

   jsonResponse, err := json.Marshal(vehicle)
   if err != nil {
      errorMsg := map[string]string{"error": "Error marshaling JSON."}
      errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError)
      return
   }
   rw.Header().Set("Content-Type", "application/json")
   rw.WriteHeader(http.StatusOK)
   rw.Write(jsonResponse)
}
```

Listing 5 - dobavljanje vozačke dozvole po JMBG-u vozača

Prilikom ovog dobavljanja, id vozača dobijamo iz url-a, koji se kasnije prosleđuje servisnom sloju. **Ctx, odnosno kontekst označava je kontekst zahteva koji može sadržati informacije o sesiji, deadline-ovima i slično. Koristi se u svakoj handler funkciji!**

Dobavljanje registrovanih vozila po kategoriji

```
category := c.Param("category")
if err != nil {
   errorMsg := map[string]string{"error": "Invalid year parameter"}
   errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
   return
}

vehicles, err :=
s.service.GetAllRegisteredVehiclesByCategory(domain.Category(category))
if err != nil {
   errorMsg := map[string]string{"error": "Failed to retrieve registered
vehicles from the database."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError)
   return
}

jsonResponse, err := json.Marshal(vehicles)
if err != nil {
   errorMsg := map[string]string{"error": "Error marshaling JSON."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError)
   return
}
rw.Header().Set("Content-Type", "application/json")
rw.WriteHeader(http.StatusOK)
rw.Write(jsonResponse)
```

Listing 5 - dobavljanje registrovanih vozila po kategoriji

Ovde takođe, iz URL-a preuzimamo kategoriju vozila, a nakon toga servis pokušava dobaviti sve registrovana vozila iz baze podataka. Ukoliko dođe do greške, dobijamo JSON sa statusom 500 i odgovarajućom porukom o grešci. Ako uspešno dobavimo vozila, dobijamo 200 OK i vidimo odgovor.

Dobavljanje vozila po kategoriji i godini

```
category := c.Param("category")
yearStr := c.Param("year")
year, err := strconv.Atoi(yearStr)
if err != nil { errorMsg := map[string]string{"error": "Invalid year
parameter"}
   errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest return
}
vehicles, err :=
s.service.GetAllVehiclesByCategoryAndYear(domain.Category(category), year)
if err != nil {
   errorMsg := map[string]string{"error": "Failed to retrieve registered
vehicles from the database."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError)  return
}
jsonResponse, err := json.Marshal(vehicles)
if err != nil {
   errorMsg := map[string]string{"error": "Error marshaling JSON."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError) return
}
rw.Header().Set("Content-Type", "application/json")
rw.WriteHeader(http.StatusOK)
rw.Write(jsonResponse)
```

Listing 6 - dobavljanje vozila po kategoriji i godini

Logika ovog HTTP handlera, je potpuno kao i sve prethodne navedene. Jedino što se razlikuje se url parametri koje šaljemo, a u ovom slučaju to su godina i kategorija.

Dobavljanje vozila po ID-ju (registarskim tablicama)

```
vehicleID := c.Param("id")
vehicle, err := s.service.GetVehicleByID(vehicleID, ctx)
if err != nil {
   errorMsg := map[string]string{"error": "Failed to retrieve vehicle from
the database.No such vehicle."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError) return
}
jsonResponse, err := json.Marshal(vehicle)
if err != nil {
   errorMsg := map[string]string{"error": "Error marshaling JSON."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError) return
}
rw.Header().Set("Content-Type", "application/json")
rw.WriteHeader(http.StatusOK)
rw.Write(jsonResponse)
```

Listing 7 - dobavljanje vozila po ID-ju

Broj vozila po kategoriji

```
category := c.Param("category")
count, err :=
s.service.GetNumberOfRegisteredVehiclesByCategory(domain.Category(category)
)
if err != nil {
   errorMsg := map[string]string{"error": "Failed to retrieve number of
registered vehicles by category."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError)  return
}
response := map[string]int64{"count": count}
jsonResponse, err := json.Marshal(response)
if err != nil {
   errorMsg := map[string]string{"error": "Error marshaling JSON."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError) return
}
rw.Header().Set("Content-Type", "application/json")
rw.WriteHeader(http.StatusOK)
rw.Write(jsonResponse)
```

Listing 7 - dobavljanje broja vozila po kategoriji

Jedina razlika u logici ove funkcije jeste što servis koristi MongoDB [6] funkciji koja broji dokumente unutar baze.

```
filter := bson.M{
   "category":          category,
   "registration_date": bson.M{"$gte": cutoffDate},
}

count, err := s.collection.CountDocuments(s.ctx, filter)
if err != nil {
   return 0, err
}

return count, nil
```

Listing 8 - brojanje entiteta unutar Mongo baze

Dobavljanje vozila po ID-u je identično kao i dobavljanje dozvole koje je već opisano, shodno tome nećemo prikazivati listing.

Kreiranje i dobavljanje svih vozila

Na ovim primerima prikazaćemo kako servis radi direktno sa bazom.

```
func (s *VehicleServiceImpl) InsertVehicle(vehicle *domain.VehicleCreate)
(*domain.Vehicle, string, error) {
   var vehicleToInsert domain.Vehicle
   vehicleToInsert.ID = primitive.NewObjectID()
   vehicleToInsert.RegistrationPlate = vehicle.RegistrationPlate
   vehicleToInsert.VehicleModel = vehicle.VehicleModel
   vehicleToInsert.VehicleOwner = vehicle.VehicleOwner
   vehicleToInsert.RegistrationDate = time.Now()
   vehicleToInsert.Category = vehicle.Category
   result, err := s.collection.InsertOne(context.Background(), vehicle)
   if err != nil {
      return nil, "", err
   }
   insertedID, ok := result.InsertedID.(primitive.ObjectID)
   if !ok {
      return nil, "", errors.New("failed to get inserted ID")
   }
   insertedID = result.InsertedID.(primitive.ObjectID)
   return &vehicleToInsert, insertedID.Hex(), nil
}
```

Listing 9 - kreiranje jednog vozila u Mongu

```
func (s *VehicleServiceImpl) GetAllVehicles() ([]*domain.Vehicle, error) {
   var vehicles []*domain.Vehicle

   filter := bson.D{}

   cursor, err := s.collection.Find(s.ctx, filter)
   if err != nil {
      return nil, err
   }
   defer cursor.Close(s.ctx)

   for cursor.Next(s.ctx) {
      var vehicle domain.Vehicle
      if err := cursor.Decode(&vehicle); err != nil {
         return nil, err
      }
      vehicles = append(vehicles, &vehicle)
   }
   if err := cursor.Err(); err != nil {
      return nil, err
   }

   return vehicles, nil
}
```

Listing 10 - dobavljanje svih vozila unutar servisa

Dobavljanje svih registrovanih vozila

```
func (s *VehicleServiceImpl) GetAllRegisteredVehicles() ([]*domain.Vehicle,
                                error) {
                        var vehicles []*domain.Vehicle
                cutoffDate := time.Now().AddDate(-1, 0, 0)
                            filter := bson.M{
                        "registration_date": bson.M{
                              "$gte": cutoffDate,
                                   },   }
              cursor, err := s.collection.Find(s.ctx, filter)
                              if err != nil {
                               return nil, err
                                    }
                        defer cursor.Close(s.ctx)
                         for cursor.Next(s.ctx) {
                         var vehicle domain.Vehicle
                if err := cursor.Decode(&vehicle); err != nil {
                              return nil, err     }
                    vehicles = append(vehicles, &vehicle)}
                   if err := cursor.Err(); err != nil {
                              return nil, err   }
                            return vehicles, nil}
```

Listing 10 - funkcija unutar servisa koja dobavlja sva registrovana vozila

Ova funkcija filtrira dokumente po polju registration_date, odnosno upit traži dokumente čiji je datum registraciju veći ili jednak trenutnom datumu pomerenom unazad za godinu dana. Registracija auta važi godinu dana, pa je logika zbog toga implementirana na ovaj način.

Kreiranje vozača

```
            vehicleDriver, exists := c.Get("vehicleDriver")
                                if !exists {
   errorMsg := map[string]string{"Error": " vehicleDriver object was not
                                valid."}
     errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
                                return
                                 }

  vehicleDriverInsert, ok := vehicleDriver.(domain.VehicleDriverCreate)

    identificationNumber := vehicleDriverInsert.IdentificationNumber

                    existingVehicleDriver, err :=
       s.service.GetVehicleDriverByID(identificationNumber, ctx)

                   if existingVehicleDriver != nil {
     errorMsg := map[string]string{"error": "Vozac sa ovim JMBG-em vec
                              postoji."}
      errorMessage.ReturnJSONError(rw, errorMsg, http.StatusConflict)
                                return
                                 }

                                if !ok {
     errorMsg := map[string]string{"error": "Invalid type for vehicle
                              driver."}
     errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
                                return
                                 }

                  vehicleDriverInsertDB, _, err :=
          s.service.InsertVehicleDriver(&vehicleDriverInsert)
                              if err != nil {
       errorMsg := map[string]string{"error": "Database problem."}
     errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
                                return
                                 }

                 rw.WriteHeader(http.StatusCreated)
       jsonResponse, err1 := json.Marshal(vehicleDriverInsertDB)
                             if err1 != nil {
   errorMessage.ReturnJSONError(rw, fmt.Sprintf("Error marshaling JSON:
              %s", err1), http.StatusInternalServerError)
                                return
                                 }
                         rw.Write(jsonResponse)
```

Listing 11 - funkcija za kreirajne vozača

Unutar konteksta zahteva proverava se da li postoji objekat tipa vehicleDriver. Ako ne postoji, vraćamo grešku. Nakon toga se proverava tip objekta vehicleDriver. Ako nije ispravan, takođe vraćamo grešku. Ukoliko postoji vozač sa istim JMBG-om u bazi, ne

možemo ga kreirati. JSON odgovor je 201, odnosno CREATED. Takođe rukujemo greškom u slučaju neuspešnog unosa u bazi ili čitanja JSON objekta.

Dobavljanje svih vozača

```
                    vehicleDriverID := c.Param("id")

vehicleDriver, err := s.service.GetVehicleDriverByID(vehicleDriverID, ctx)
                            if err != nil {
     errorMsg := map[string]string{"error": "Failed to retrieve vehicle
                driver from the database.No such driver."}
                  errorMessage.ReturnJSONError(rw, errorMsg,
                      http.StatusInternalServerError)
                                  return
                                   }

            jsonResponse, err := json.Marshal(vehicleDriver)
                            if err != nil {
      errorMsg := map[string]string{"error": "Error marshaling JSON."}
                  errorMessage.ReturnJSONError(rw, errorMsg,
                      http.StatusInternalServerError)
                                  return
                                   }

           rw.Header().Set("Content-Type", "application/json")
                       rw.WriteHeader(http.StatusOK)
                           rw.Write(jsonResponse)
```

Listing 12 - dobavljanje svih vozača

Dobavljanje vozača po JMBG-u

```
                    vehicleDriverID := c.Param("id")
vehicleDriver, err := s.service.GetVehicleDriverByID(vehicleDriverID, ctx)
                            if err != nil {
     errorMsg := map[string]string{"error": "Failed to retrieve vehicle
                driver from the database.No such driver."}
                  errorMessage.ReturnJSONError(rw, errorMsg,
                      http.StatusInternalServerError)
                                  return
                                   }
            jsonResponse, err := json.Marshal(vehicleDriver)
                            if err != nil {
      errorMsg := map[string]string{"error": "Error marshaling JSON."}
                  errorMessage.ReturnJSONError(rw, errorMsg,
                      http.StatusInternalServerError)
                                  return
                                   }
           rw.Header().Set("Content-Type", "application/json")
                       rw.WriteHeader(http.StatusOK)
                           rw.Write(jsonResponse)
```

Listing 13 - dobavljanje vozača po JMBG-u

Unos prekršajnih poena

```
if err := decoder.Decode(&responseUser); err != nil {
   errorMsg := map[string]string{"error": "User object was not valid."}
   errorMessage.ReturnJSONError(rw, errorMsg, http.StatusUnauthorized)
   return
}
if responseUser.LoggedInUser.UserRole != data.TrafficPoliceman {
   errorMsg := map[string]string{"Unauthorized": " You are not policeman."}
   errorMessage.ReturnJSONError(rw, errorMsg, http.StatusUnauthorized)
   return
}
var input struct {
   Points int64 `json:"points"`
}
if err := c.ShouldBindJSON(&input); err != nil {
   errorMsg := map[string]string{"error": "Invalid input."}
   errorMessage.ReturnJSONError(rw, errorMsg, http.StatusBadRequest)
   return
}
vehicleDriverID := c.Param("id")
err = s.service.UpdatePenaltyPoints(vehicleDriverID, input.Points, ctx)
if err != nil {
   errorMsg := map[string]string{"error": "Failed to update penalty
points."}
   errorMessage.ReturnJSONError(rw, errorMsg,
http.StatusInternalServerError)  return
}
rw.WriteHeader(http.StatusOK)
rw.Write([]byte(`{"message": "Penalty points updated successfully."}`))
```

Listing 14 - unos prekršajnih poena

Ovaj kod prvo dekodira responseUser objekat i proverava da li je validan. Ako nije, vraća grešku 401, odnosno Unauthorized. Ovu operaciju može da izvrši saobraćajni policajac koji ima pristup MUP servisu. Zahtev ove funkcije sadrži JSON sa brojem kaznenih poena. Vozač se dobavlja po ID-ju iz parametara zahteva, i njegovi poeni se ažuriraju. Prikazuju se određene poruke o greškama, kao i u svakoj funkciji.

Generisanje izveštaja o vozilima po kategoriji

```
func (s *VehicleHandler) GenerateAndServeVehiclesByCategoryReportPDF(c
*gin.Context) {
   categoryParam := c.Param("category")
   if categoryParam == "" {
      c.JSON(http.StatusBadRequest, gin.H{"error": "Category parameter is
required"})  return
   }
   category := domain.Category(categoryParam)
   vehicles, err := s.service.GetAllRegisteredVehiclesByCategory(category)
   fmt.Println(vehicles)
   fmt.Println("ALL REGISTERED VEHICLES BY CATEGORY")
   if err != nil {
      c.JSON(http.StatusInternalServerError, gin.H{"error": "Error
retrieving registered vehicles for category"})
      log.Println("Error retrieving registered vehicles for category:",
err)   return
   }
   pdf := gofpdf.New("P", "mm", "A4", "")
   pdf.AddPage()
   pdf.SetFont("Arial", "B", 16)
   pdf.SetFillColor(240, 240, 240)
   pdf.Rect(10, 10, 190, 12, "F")
   pdf.SetTextColor(0, 0, 0)
   pdf.CellFormat(0, 12, "Izvestaj o registrovanim vozilima za kategoriju
"+string(category), "", 0, "C", true, 0, "")
   pdf.Ln(15)
   pdf.SetFont("Arial", "B", 14)
   pdf.SetFillColor(255, 255, 255)
   pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
   pdf.SetTextColor(0, 0, 0)
   pdf.CellFormat(0, 8, "Detalji registrovanih vozila", "", 0, "C", false,
0, "")
   pdf.Ln(10)
   pdf.SetFont("Arial", "", 12)
   for _, vehicle := range vehicles {
      pdf.SetFillColor(240, 240, 240)
      pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
      pdf.SetTextColor(0, 0, 0)
      pdf.CellFormat(0, 8, fmt.Sprintf("ID: %s", vehicle.ID.Hex()), "", 0,
"", false, 0, "")
      pdf.Ln(10)
      pdf.SetFillColor(240, 240, 240)
      pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
      pdf.SetTextColor(0, 0, 0)
      pdf.CellFormat(0, 8, fmt.Sprintf("Registraciona tablica: %s",
vehicle.RegistrationPlate), "", 0, "", false, 0, "")
      pdf.Ln(10)
      pdf.SetFillColor(240, 240, 240)
      pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
      pdf.SetTextColor(0, 0, 0)
```

```
      pdf.CellFormat(0, 8, fmt.Sprintf("Model vozila: %s",
vehicle.VehicleModel), "", 0, "", false, 0, "")
      pdf.Ln(10)
      pdf.SetFillColor(240, 240, 240)
      pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
      pdf.SetTextColor(0, 0, 0)
      pdf.CellFormat(0, 8, fmt.Sprintf("Vlasnik vozila: %s",
vehicle.VehicleOwner), "", 0, "", false, 0, "")
      pdf.Ln(10)
      pdf.SetFillColor(240, 240, 240)
      pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
      pdf.SetTextColor(0, 0, 0)
      pdf.CellFormat(0, 8, fmt.Sprintf("Datum registracije: %s",
vehicle.RegistrationDate.Format("02.01.2006")), "", 0, "", false, 0, "")
      pdf.Ln(10)
      pdf.SetFillColor(240, 240, 240)
      pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
      pdf.SetTextColor(0, 0, 0)
      pdf.CellFormat(0, 8, fmt.Sprintf("Kategorija: %s", vehicle.Category),
"", 0, "", false, 0, "")
      pdf.Ln(20) // Add extra space here between each vehicle's details
   }
   pdf.SetFooterFunc(func() {
      // Footer
      pdf.SetY(-15)
      pdf.SetFont("Arial", "I", 10)
      pdf.CellFormat(0, 10, "Generisano od strane eUprave", "", 0, "C",
false, 0, "")
   })
   // Serve PDF as downloadable file
   c.Writer.Header().Set("Content-Disposition", "attachment;
filename=registered_vehicles_report_"+string(category)+".pdf")
   c.Writer.Header().Set("Content-Type", "application/pdf")
   err = pdf.Output(c.Writer)
   if err != nil {
      c.JSON(http.StatusInternalServerError, gin.H{"error": "Error
generating PDF"})
      log.Println("Error generating PDF:", err)
      return
   }
   log.Println("Generated PDF served successfully")
}
```

Listing 15 - generisanje izveštaja o registrovanim vozilima po kategoriji

Ovaj kod predstavlja handler funkciju za generisanje i serviranje PDF izveštaja o registrovanim vozilima po kategoriji koristeći Gin web framework i biblioteku `gofpdf` za kreiranje PDF-a.

Funkcija dobija parametar kategorija iz URL-a. Nakon svih neophodnih validacija, kategorija se prosleđuje metodi za dobavljanje svih registrovanih vozila po kategoriji. Ako su vozila pronađena, funkcija kreira novi PDF dokument i dodaje stranicu. Podešavaju se fontovi, boje i zaglavlja. Zatim se iterira kroz listu vozila koju smo dobili od funkcije za dobavljanje svih registrovanih vozil po kategoriji i upisuje podatke u PDF. Na kraju postavljamo se footer i konfiguriše HTTP header za preuzimanje PDF-a kao datoteke. Ukoliko dođe do greške prilikom generisanja, dobijamo grešku 500. Ako je PDF uspešno generisan, šalje se kao odgovor na HTTP zahtev.

Generisanje izveštaja o registrovanim vozilima

```
func (s *VehicleHandler) GenerateAndServeVehiclesReportPDF(c *gin.Context)
                                    {
                                // Generate PDF
            vehicles, err := s.service.GetAllRegisteredVehicles()
                                if err != nil {
          c.JSON(http.StatusInternalServerError, gin.H{"error": "Error
                      retrieving registered vehicles"})
            log.Println("Error retrieving registered vehicles:", err)
                                    return
                                     }

                    pdf := gofpdf.New("P", "mm", "A4", "")
                                 pdf.AddPage()

                         pdf.SetFont("Arial", "B", 16)

                        pdf.SetFillColor(240, 240, 240)
                         pdf.Rect(10, 10, 190, 12, "F")
                           pdf.SetTextColor(0, 0, 0)
    pdf.CellFormat(0, 12, "Izvestaj o registrovanim vozilima", "", 0, "C",
                                true, 0, "")
                                  pdf.Ln(15)

                         pdf.SetFont("Arial", "B", 14)

                        pdf.SetFillColor(255, 255, 255)
                    pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
                           pdf.SetTextColor(0, 0, 0)
   pdf.CellFormat(0, 8, "Detalji registrovanih vozila", "", 0, "C", false,
                                    0, "")
                                  pdf.Ln(10)

                         pdf.SetFont("Arial", "", 12)

                      for _, vehicle := range vehicles {
                         pdf.SetFillColor(240, 240, 240)
                    pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
                           pdf.SetTextColor(0, 0, 0)
```

```
pdf.CellFormat(0, 8, fmt.Sprintf("ID: %s", vehicle.ID.Hex()), "", 0,
                        "", false, 0, "")
                              pdf.Ln(10)

                      pdf.SetFillColor(240, 240, 240)
                  pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
                         pdf.SetTextColor(0, 0, 0)
        pdf.CellFormat(0, 8, fmt.Sprintf("Registraciona tablica: %s",
          vehicle.RegistrationPlate), "", 0, "", false, 0, "")
                              pdf.Ln(10)

                      pdf.SetFillColor(240, 240, 240)
                  pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
                         pdf.SetTextColor(0, 0, 0)
            pdf.CellFormat(0, 8, fmt.Sprintf("Model vozila: %s",
           vehicle.VehicleModel), "", 0, "", false, 0, "")
                              pdf.Ln(10)

                      pdf.SetFillColor(240, 240, 240)
                  pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
                         pdf.SetTextColor(0, 0, 0)
           pdf.CellFormat(0, 8, fmt.Sprintf("Vlasnik vozila: %s",
           vehicle.VehicleOwner), "", 0, "", false, 0, "")
                              pdf.Ln(10)

                      pdf.SetFillColor(240, 240, 240)
                  pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
                         pdf.SetTextColor(0, 0, 0)
          pdf.CellFormat(0, 8, fmt.Sprintf("Datum registracije: %s",
vehicle.RegistrationDate.Format("02.01.2006")), "", 0, "", false, 0, "")
                              pdf.Ln(10)

                      pdf.SetFillColor(240, 240, 240)
                  pdf.Rect(10, pdf.GetY()+2, 190, 8, "F")
                         pdf.SetTextColor(0, 0, 0)
     pdf.CellFormat(0, 8, fmt.Sprintf("Kategorija: %s", vehicle.Category),
                       "", 0, "", false, 0, "")
       pdf.Ln(20) // Add extra space here between each vehicle's details
                                    }

                       pdf.SetFooterFunc(func() {
                                 // Footer
                              pdf.SetY(-15)
                       pdf.SetFont("Arial", "I", 10)
       pdf.CellFormat(0, 10, "Generisano od strane eUprave", "", 0, "C",
                            false, 0, "")
                                   })

                    // Serve PDF as downloadable file
         c.Writer.Header().Set("Content-Disposition", "attachment;
                filename=registered_vehicles_report.pdf")
         c.Writer.Header().Set("Content-Type", "application/pdf")
```

```
err = pdf.Output(c.Writer)
if err != nil {
c.JSON(http.StatusInternalServerError, gin.H{"error": "Error generating PDF"})
log.Println("Error generating PDF:", err)
return
}

log.Println("Generated PDF served successfully")
}
```

Listing 16 - generisanje izveštaja o registrovanim vozilima

Rutiranje

```
type VehicleRouteHandler struct {
handler       handlers.VehicleHandler
service       services.VehicleService
driverService services.VehicleDriverService
}

func NewVehicleRouteHandler(handler handlers.VehicleHandler, service services.VehicleService, driverService services.VehicleDriverService) VehicleRouteHandler {
return VehicleRouteHandler{handler, service, driverService}
}

func (vr *VehicleRouteHandler) VehicleRoute(rg *gin.RouterGroup) {
router := rg.Group("/vehicle")
router.POST("/createVehicle", MiddlewareVehicleDeserialization, vr.handler.CreateVehicle)
router.GET("/all", vr.handler.GetAllVehicles)
router.GET("/all/registered", vr.handler.GetAllRegisteredVehicles)
router.GET("/get/category/:category/year/:year", vr.handler.GetAllVehiclesByCategoryAndYear)
router.GET("/get/:id", vr.handler.GetVehicleByID)
router.GET("/registeredVehicles/pdf", vr.handler.GenerateAndServeVehiclesReportPDF)
router.GET("/registeredVehicles/category/:category/pdf", vr.handler.GenerateAndServeVehiclesByCategoryReportPDF)
router.GET("/count/category/:category", vr.handler.GetNumberOfRegisteredVehiclesByCategory)
router.GET("/all/registered/:category", vr.handler.GetAllRegisteredVehiclesByCategory)

}

func MiddlewareVehicleDeserialization(c *gin.Context) {
var vehicle domain.VehicleCreate
```

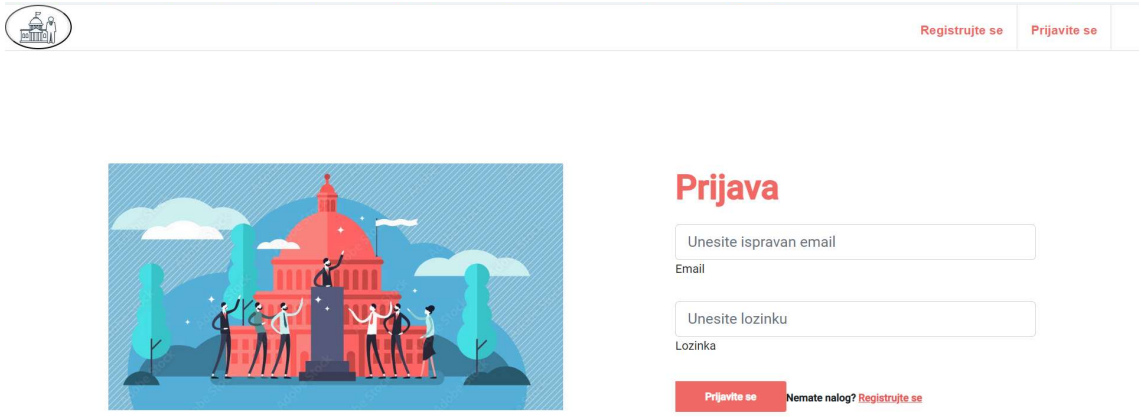
```
        if err := c.ShouldBindJSON(&vehicle); err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": "Unable to decode
                        JSON"})
                          c.Abort()
                           return
                            }

                  c.Set("vehicle", vehicle)
                          c.Next()
                            }
```

Listing 17 - rutiranje

Za rutiranje koristi se Gin web framework, koji uključuje middleware za deserijalizaciju JSON objekata. Middleware funkcija deserijalizuje JSON podatke o vozilu, i postavlja ih u kontekst kako bi bili dostupni handlerima za obradu. Klase koje regulišu rutiranje su dostupne za vozila, vozače i dozvole.

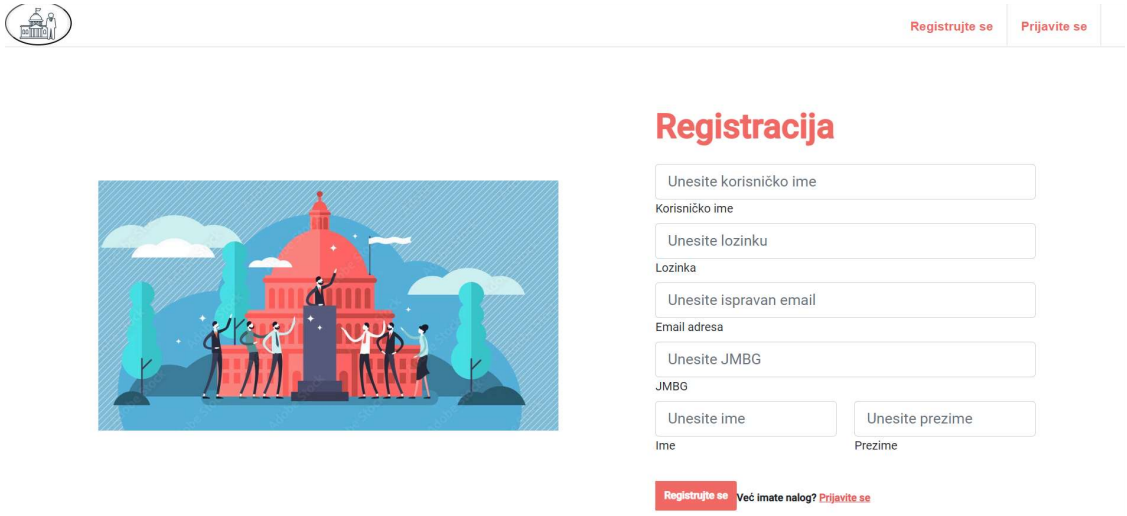
9. Demonstracija

Prijava i registracija



Slika 7 - prijava na sistem

Prilikom pokretanja aplikacije, korisniku se prikazuje forma u kojoj unosi kredencijale, korisničko ime i lozinku. Ukoliko nema nalog, može da se registruje.



Slika 8 - registracija na sistem

Početna stranica

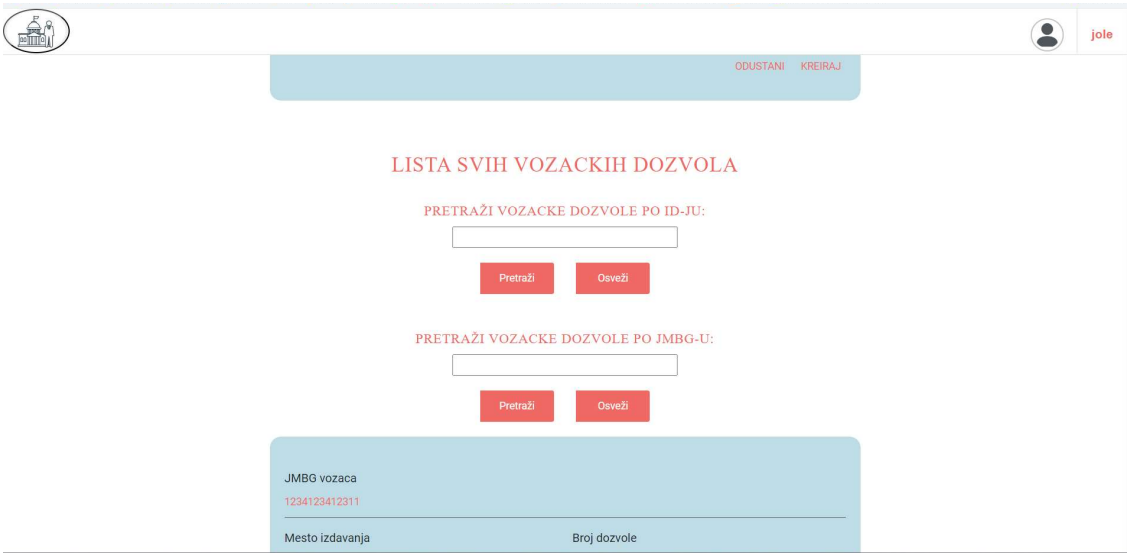
Slika 9 - početna stranica

Na početnoj stranici su linkovi do vozila, vozača i vozačkih dozvola.

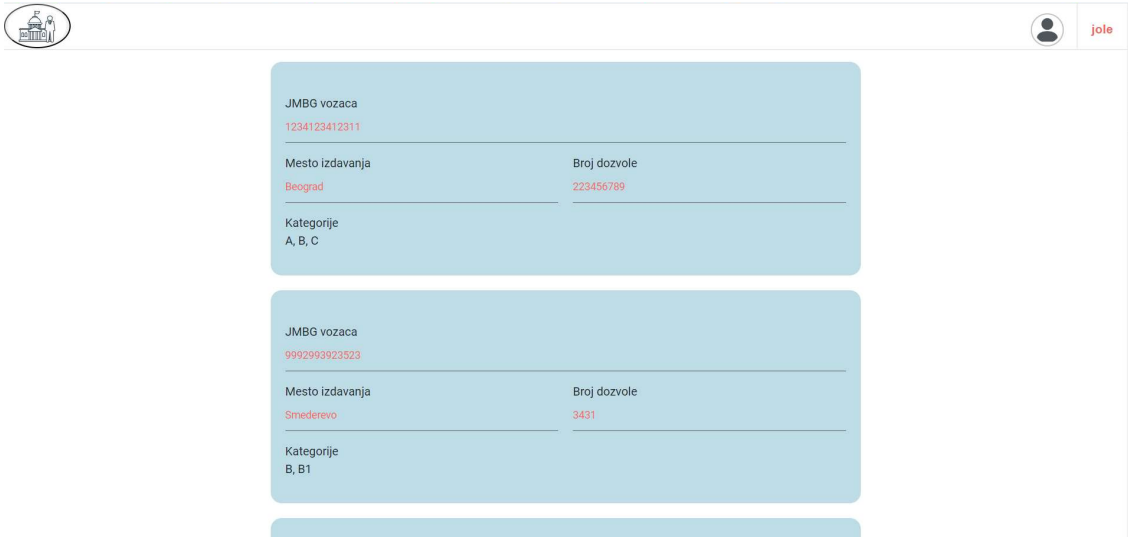
Stranica sa vozačkim dozvolama

Slika 10 - stranica sa vozačkim dozvolama

Na ovoj stranici policajacv ima mogućnost kreiranja nove vozačke dozvole, kao i pregled svih vozačkih dozvola. Takođe, ovde se nalazi forma za pretragu vozačkih dozvola po ID-ju i JMBG-u vozača (Slika 11). Na slici 10 nalazi se forma za kreiranje nove dozvole. Slika 12 prikazuje izlistane vozačke dozvole.



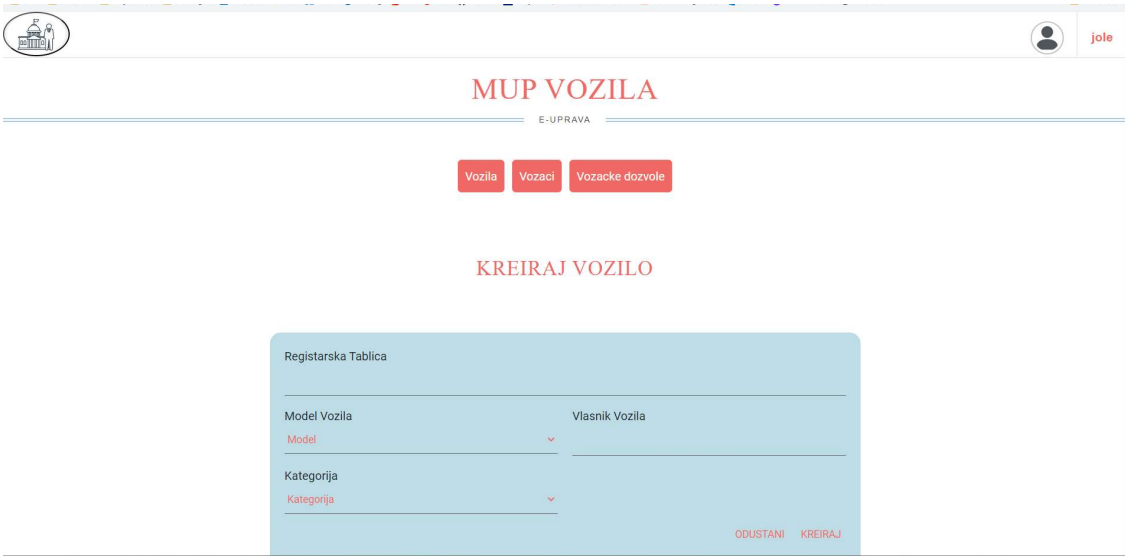
Slika 11 - pretraga vozačkih dozvola



Slika 12 - prikaz svih vozačkih dozvola

Stranica sa svim vozilima

Stranica sa svim voziilima sadrži formu za kreiranje vozila, prikaz svih vozila, pretrage po određenim kriterijumima, kao i link ga stranici sa svim registrovanim vozilima (Slika 13).



Slika 13 - stranica sa svim vozilima

Ukoliko polja ne unesemo u skladu sa validacijama, forma će izbaciti upozorenja. Naravno, ovo važi za sve forme u sistemu, a na jednoj ćemo pokazati izgled validacionih grešaka (Slika 14).

Slika 14 - forma za registrovanje vozila sa validacionim pravilima

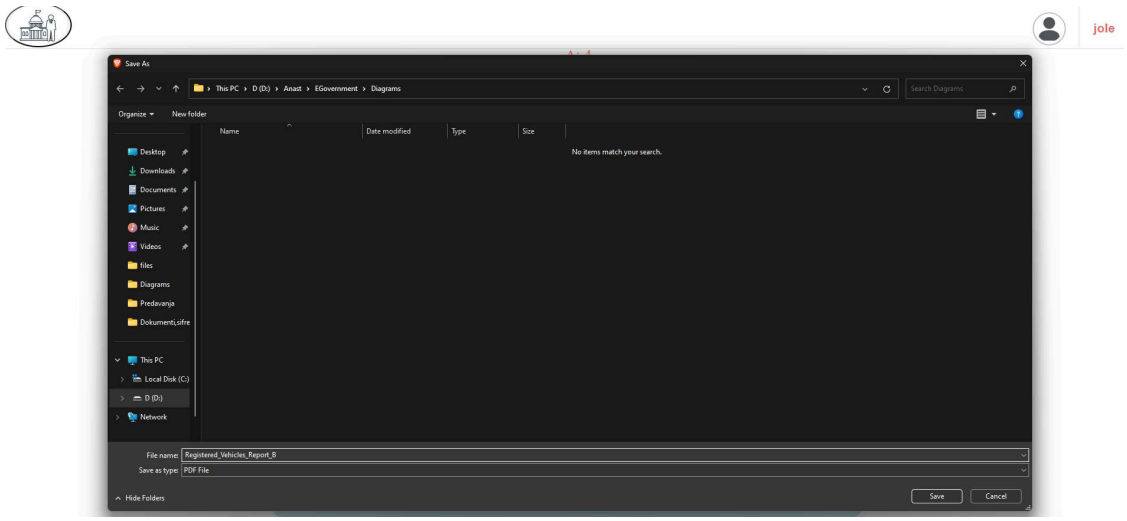
Na istoj stranici imamo formu za pretragu vozila po registarskim tablicama, kategoriji i godini (kategorija i godina su jedna pretraga). Klikom na dugme *Pretraži* lista vozila se menja, a klikom na *Osveži* ponovo se prikazuju sva vozila (Slika 15).

Slika 15 - forme za pretragu vozila

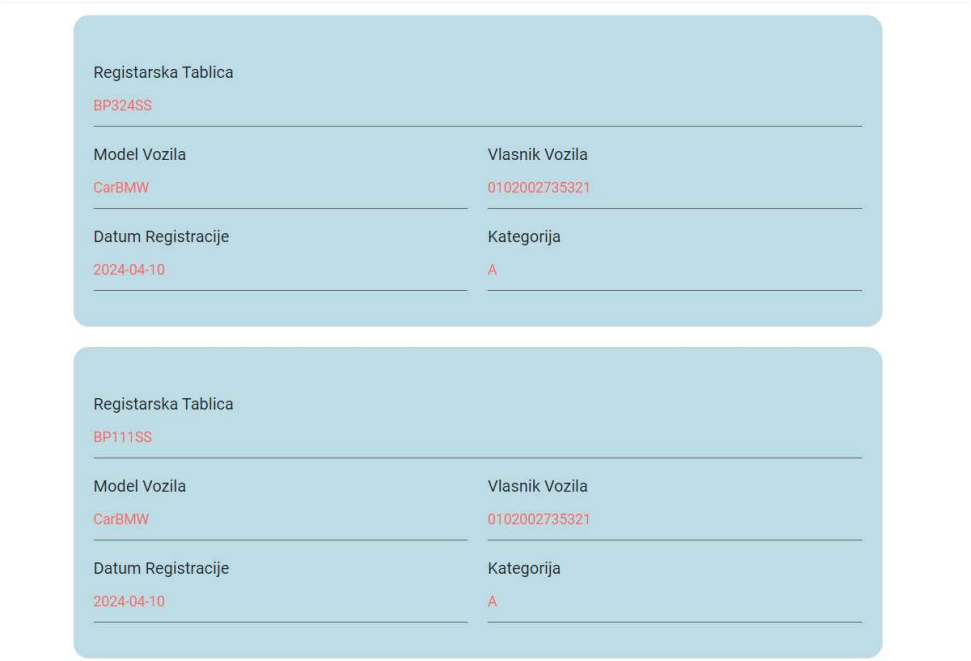
Klikom na link za pregled svih registrovaih vozila, nalazimo se na stranici sa izlistanim registrovanim vozilima (onim čiji datum registracije je bio pre manje od godinu dana). Tu imamo mogućnost generisanja PDF izveštaja sa svim registrovanim vozilima, kao i generisanje izveštaja o registrovanim vozilima po određenoj kategoriji. Takođe, broj svih registrovanih vozila po kategorijama je prikazan (Slika 16).

Slika 16

Klikom na bilo koji preuzmi PDF, imamo opciju da sačuvamo datoteku u naš računar (Slika 17).

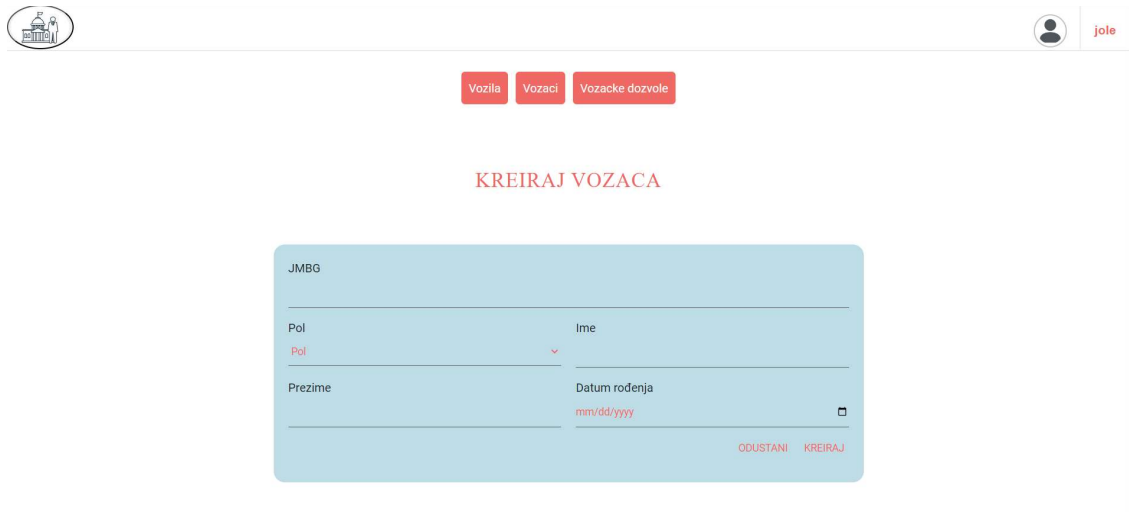


Slika 17



Slika 18 - izlistana registrovana vozila

Stranica sa vozačima



Slika 19 - Stranica sa svim vozačima

Na ovoj stranici se takođe nalazi forma za pretragu vozača po jedinstvenom matičnom broju (Slika 20).

Slika 20

Ova demonstracija prikazala je celokupan interfejs MUP servisa sa vozila.

10. Zaključak

U ovom seminarskom radu opisan je projekat koji olakšava MUP-u rad sa vozačima, njihovim dozvolama i vozilima.
Primena mikroservisne arhitekture u sistemima poput ovog donosi značajne prednosti u efikasnosti, skalabilnosti i održavanju ovih sistema. Kroz razne funkcionalnosti, poput generisanja izveštaja, pretrage raznih podataka o entitetima ovoh sistemima, videli smo koliko ova tehnologija skraćuje vreme potrebno za dobavljanje informacija i njihov prenos između drugih servisa. Stvari koji su nekada morale da se unose i traže ručno, sada su olakšane. Mikroservisi pružaju izolovanost, odnosno nezavisnost od drugih sistema, što ubrzava razmenu i skladištenje informacija.
Kroz ovaj projekat prikazano je kako tehnologija može olakšati obradu podataka, i pružiti korisnicima intuitivan interfejs, te im olakšati rad kao zaposlenim licima.

10 Literatura

[1] Go programski jezik - https://en.wikipedia.org/wiki/Go_(programming_language)
[1] Go programski jezik -https://go.dev/
[2] Angular - https://en.wikipedia.org/wiki/Angular_(web_framework)
[2] Angular - https://v17.angular.io/docs
[3] Docker - https://www.docker.com/
[4] Department of Motor Vehicles - https://www.dmvusa.com/
[5] Driver and Vehicle Licensing Agency - https://www.gov.uk/
[6] Mongo - https://www.mongodb.com/company/what-is-mongodb