Univerzitet u Novom Sadu,
Fakultet tehničkih nauka

SEMINARSKI RAD

Nastavni predmet: Tehnologije i sistemi eUprave

Tema: MUP - Vozila

Nemanja Pokornić

1. Sažetak

Ovaj rad opisuje softverski servis za vozila vezan za Ministarstvo unutrašnjih poslova (MUP) koji omogućava kreiranje vozačkih dozvola, vozača, registraciju vozila, pretragu vozila, dobavljanje kazni. Sistem koriste policajci I obicni korisnici za pregled, upravljanje informacijama o vozilima i vozačima. Backend je razvijen u Go programskom jeziku, frontend u React-u, a sistem je dokerizovan za lakšu implementaciju. Autentifikacija korisnika se obavlja putem single sign-on (SSO) tehnologije, a ceo sistem je organizovan u mikroservisnoj arhitekturi za bolju skalabilnost i fleksibilnost.

2. Ključne Reči

- MUP
- Vozila
- Lična karta
- Vozačka dozvola
- Zabrane vožnje
- Mikroservisi

3. Uvod

Mikroservisna 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 vise drugih sistema. Ovaj seminarski rad ima za cilj da objasni korišćenje mikroservisa unutar MUP-a, osmišljenih za čuvanje i upravljanje podacima o vozilima, vozačima i vozačkim dozvolama. Detaljno ćemo analizirati i prikazati funkcionalnosti ovog mikroservisa, zajedno sa neophodnim 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.

Sistem razvijen za MUP koristi sledeće tehnologije:

- **Go programski jezik**: Backend je razvijen u Go jeziku zbog njegove efikasnosti i podrške za konkurentno programiranje.
- **React**: Frontend je razvijen u React biblioteci.
- **Docker**: Docker je korišćen za kontejnerizaciju aplikacije, omogućavajući jednostavniju implementaciju i skaliranje.

Sistem je podeljen na mikroservise, pri čemu svaki mikroservis upravlja specifičnim funkcionalnostima kao što su upravljanje vozačima, vozačkim dozvolama i vozilima. Servisi međusobno komuniciraju kako bi obezbedili integralnost i konzistentnost podataka. Na primer, prilikom izdavanja vozačke dozvole, sistem proverava da li postoji poternica za korisnikom od suda.

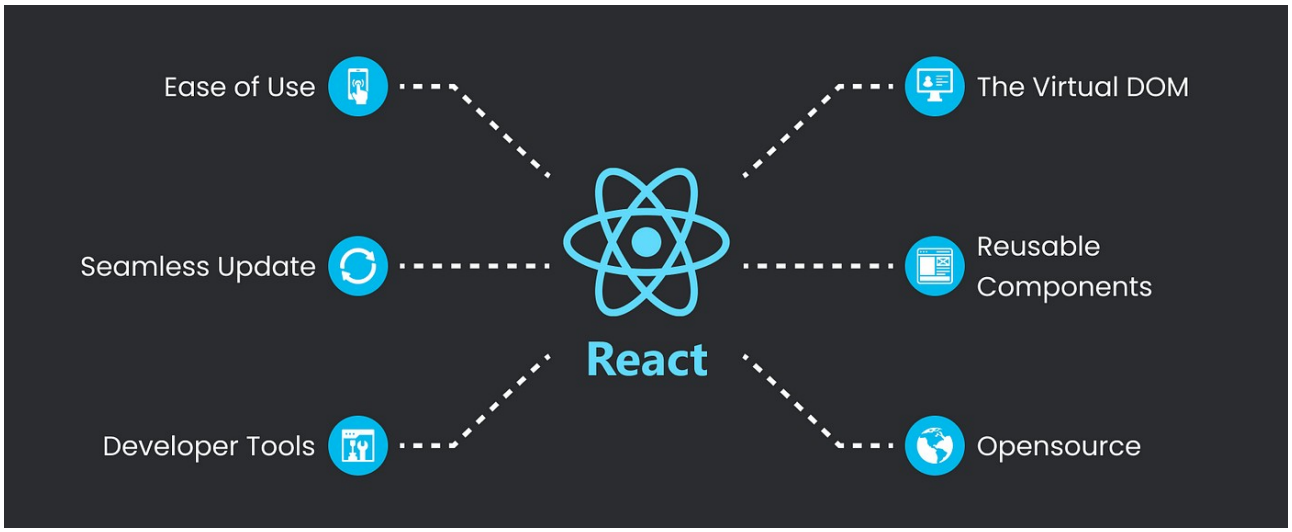
4. Korišćene tehnologije

4.1 Go programski jezik

Go, poznat i kao Golang, je programski jezik razvijen od strane Google-a. Njegova jednostavnost, efikasnost i performanse čine ga idealnim za izgradnju skalabilnih i visoko performantnih aplikacija. Go nudi brzu kompilaciju, jednostavnu sintaksu i ugrađenu podršku za konkurentnost kroz goroutine-e, što omogućava lakše upravljanje paralelnim operacijama. Ove karakteristike čine Go idealnim izborom za razvoj serverskih aplikacija, mrežnih servisa i mikrousluga.

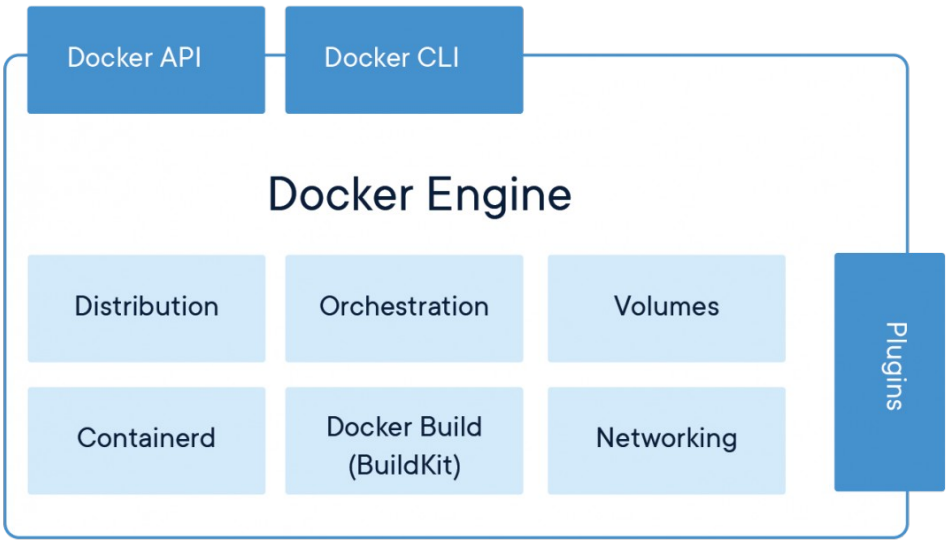
4.2 React

React je JavaScript biblioteka razvijena od strane Facebook-a koja se koristi za izgradnju korisničkih interfejsa. Fokusira se na razvoj komponenti koje omogućavaju ponovnu upotrebu koda i olakšavaju održavanje kompleksnih aplikacija. React koristi virtualni DOM (Document Object Model) koji omogućava brzu i efikasnu manipulaciju stranicom, čime se postiže bolje korisničko iskustvo. Njegova popularnost i široka podrška u zajednici čine ga standardnim izborom za moderni frontend razvoj.



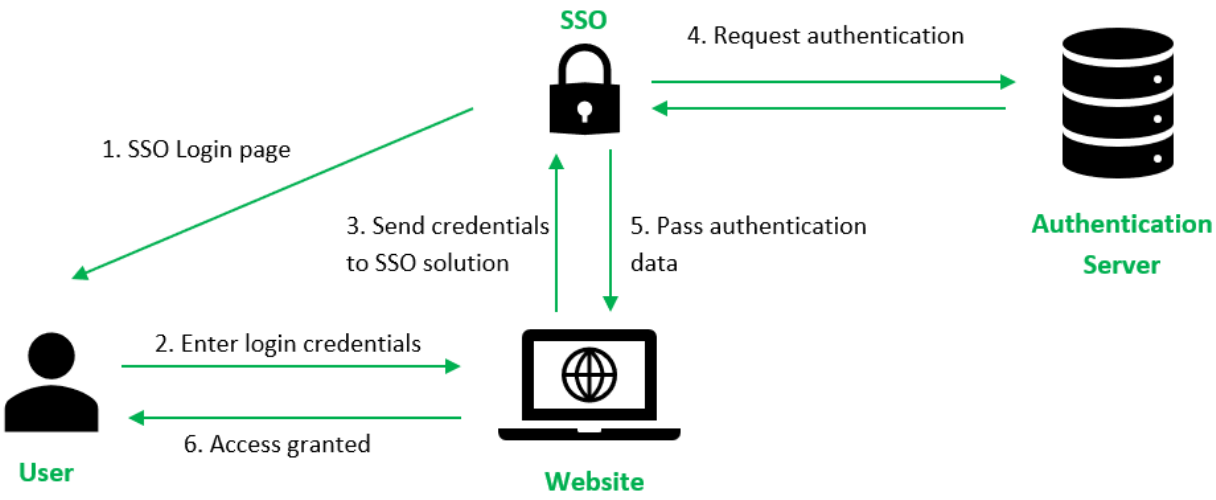
4.3 Docker

Docker je platforma koja omogućava kreiranje, isporuku i pokretanje aplikacija u izolovanim kontejnerima. Kontejneri su lagane, prenosive jedinice koje sadrže sve što je potrebno za izvršavanje aplikacije – od koda i biblioteka do postavki okruženja. Korišćenje Dockera omogućava konzistentnost okruženja kroz razvoj, testiranje i produkciju, čime se smanjuje mogućnost problema uzrokovanih različitim okruženjima. Docker takođe olakšava skaliranje aplikacija i upravljanje resursima.



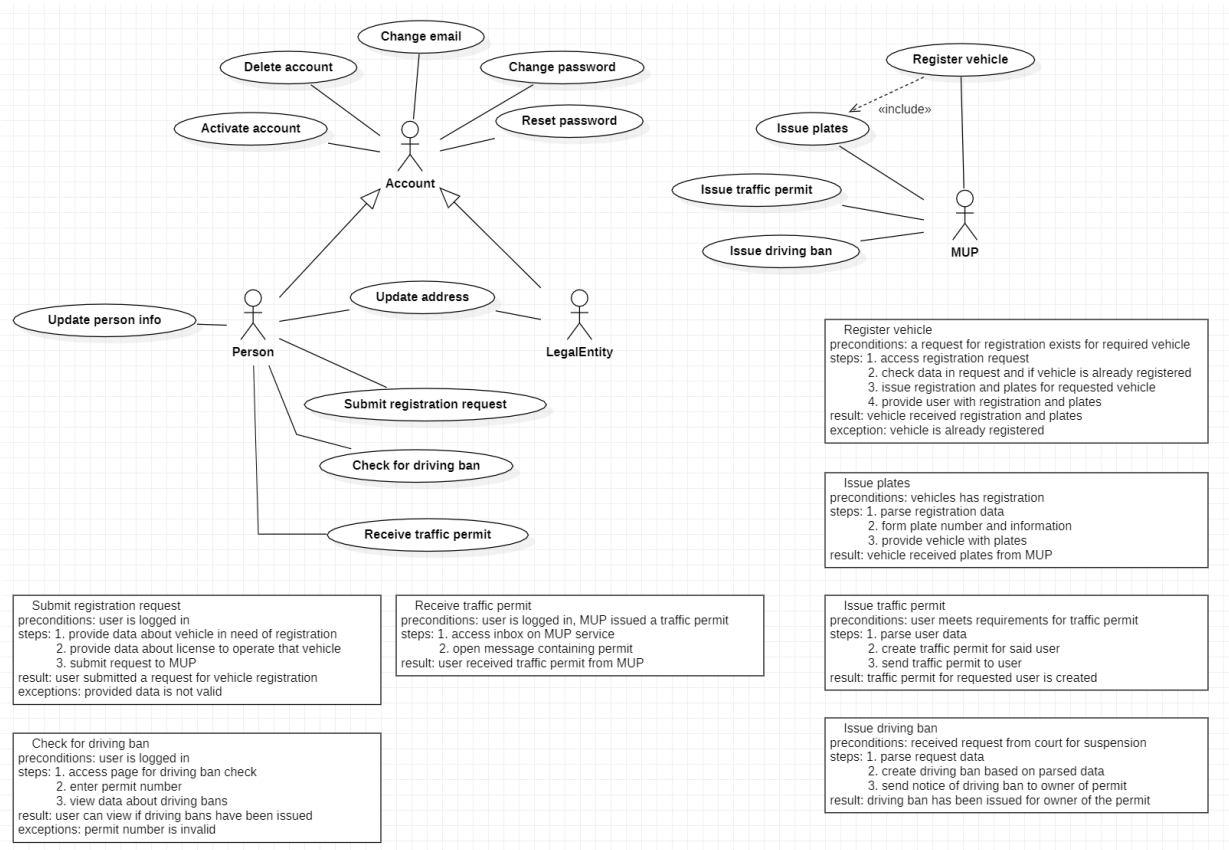
4.4 Single Sign-On (SSO)

Single Sign-On (SSO) je metoda autentifikacije koja omogućava korisnicima da pristupe višestrukim aplikacijama sa jednim skupom kredencijala (korisničkim imenom i lozinkom). Ova tehnologija poboljšava korisničko iskustvo smanjenjem potrebe za ponovnim unosom podataka za prijavu prilikom prelaska između aplikacija. SSO takođe povećava sigurnost jer omogućava centralizovano upravljanje pristupom i autentifikacijom. Implementacija SSO može koristiti različite protokole kao što su OAuth, SAML ili OpenID Connect.



5. 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.



5.1 Funkcionalni zahtevi:

5.1.1 Registracija vozila

- Preduslovi: postoji zahtev za registraciju potrebnog vozila.
- Koraci:
    1. Pristupiti zahtevu za registraciju.
    2. Proveriti podatke u zahtevu i da li je vozilo već registrovano.
    3. Izdati registraciju i tablice za traženo vozilo.
    4. Korisniku pružiti registraciju i tablice.
- Rezultat: vozilo je registrovano i dobilo je tablice.
- Izuzeci: vozilo je već registrovano.

5.1.2 Izdavanje tablica

- Preduslovi: vozilo ima registraciju.
- Koraci:
    1. Parsirati podatke o registraciji.
    2. Formirati broj tablica i informacije.
    3. Pružiti korisniku tablice.
- Rezultat: vozilo je dobilo tablice od MUP-a.

5.1.3 Izdavanje saobraćajne dozvole

- Preduslovi: korisnik ispunjava uslove za saobraćajnu dozvolu.
- Koraci:
    1. Parsirati korisničke podatke.
    2. Kreirati saobraćajnu dozvolu za navedenog korisnika.
    3. Poslati dozvolu korisniku.
- Rezultat: saobraćajna dozvola za traženi korisnički nalog je kreirana.

5.1.14 Izdavanje zabrane vožnje

- Preduslovi: primljen je zahtev od suda za suspenziju.
- Koraci:
    1. Parsirati podatke o zahtevima.
    2. Kreirati zabranu vožnje na osnovu parsiranih podataka.
    3. Poslati obaveštenje o zabrani vlasniku dozvole.
- Rezultat: zabrana vožnje je izdata za vlasnika navedene dozvole.

5.1.5 Podnošenje zahteva za registraciju

- Preduslovi: korisnik je prijavljen.
- Koraci:
    1. Pružiti podatke o vozilu koje treba registrovati.
    2. Pružiti podatke o licenci za upravljanje vozilom.
    3. Podneti zahtev MUP-u.
- Rezultat: korisnik je podneo zahtev za registraciju vozila.
- Izuzeci: pruženi podaci nisu validni.

5.1.6 Primanje saobraćajne dozvole

- Preduslovi: korisnik je prijavljen, MUP je izdao saobraćajnu dozvolu.
- Koraci:
    1. Pristupiti inbox-u na MUP servisu.
    2. Otvoriti poruku koja sadrži dozvolu.
- Rezultat: korisnik je primio saobraćajnu dozvolu od MUP-a.

5.1.7 Provera zabrane vožnje

- Preduslovi: korisnik je prijavljen.
- Koraci:
    1. Pristupiti stranici za proveru zabrane vožnje.
    2. Uneti broj dozvole.
    3. Pogledati podatke o zabranama vožnje.
- Rezultat: korisnik može videti da li su izdate zabrane vožnje.
- Izuzeci: broj dozvole nije validan.

5.2 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.

5.2.1. Dokerizacija

Aplikacija je mikroservisne arhitekture, gde svaki mikroservis ima svoj kontejner. Za ovo se koristi Docker Compose alat. Docker omogućava izolaciju servisa, olakšava razvoj, testiranje i implementaciju, te osigurava da aplikacija može raditi u različitim okruženjima bez dodatnih konfiguracija.

5.2.2. Otpornost na otkaze pojedinačnih servisa

Ukoliko jedan servis iz bilo kog razloga prestane sa radom, ostali će nastaviti da funkcionišu. Ovo obezbeđuje upravo Docker i njegovi kontejneri. Time se smanjuje rizik od potpunog prekida rada sistema i omogućava se lakša identifikacija i rešavanje problema.

5.2.3. Performanse i skalabilnost

Aplikacija mora biti sposobna da efikasno obrađuje veliki broj zahteva u kratkom vremenskom periodu. Implementacija mikroservisa omogućava horizontalno skaliranje, gde se novi resursi mogu dodati po potrebi kako bi se zadovoljile veće potrebe za obradom podataka.

5.2.4. Sigurnost

Podaci unutar sistema moraju biti zaštićeni od neovlašćenog pristupa i manipulacije. Single Sign-On (SSO) tehnologija omogućava sigurno autentifikaciju korisnika i osigurava da samo ovlašćeni

korisnici imaju pristup određenim funkcionalnostima. Dodatno, komunikacija između mikroservisa je šifrovana kako bi se zaštitili preneti podaci.

5.2.5. Održavanje i ažuriranje sistema

Sistem mora omogućiti lako održavanje i ažuriranje, bez potrebe za potpunim prekidom rada. Korišćenje kontejnera omogućava da se pojedinačni mikroservisi ažuriraju ili zamene bez uticaja na celokupan sistem. Ovo omogućava kontinuiranu integraciju i isporuku (CI/CD) i brže prilagođavanje promenama.

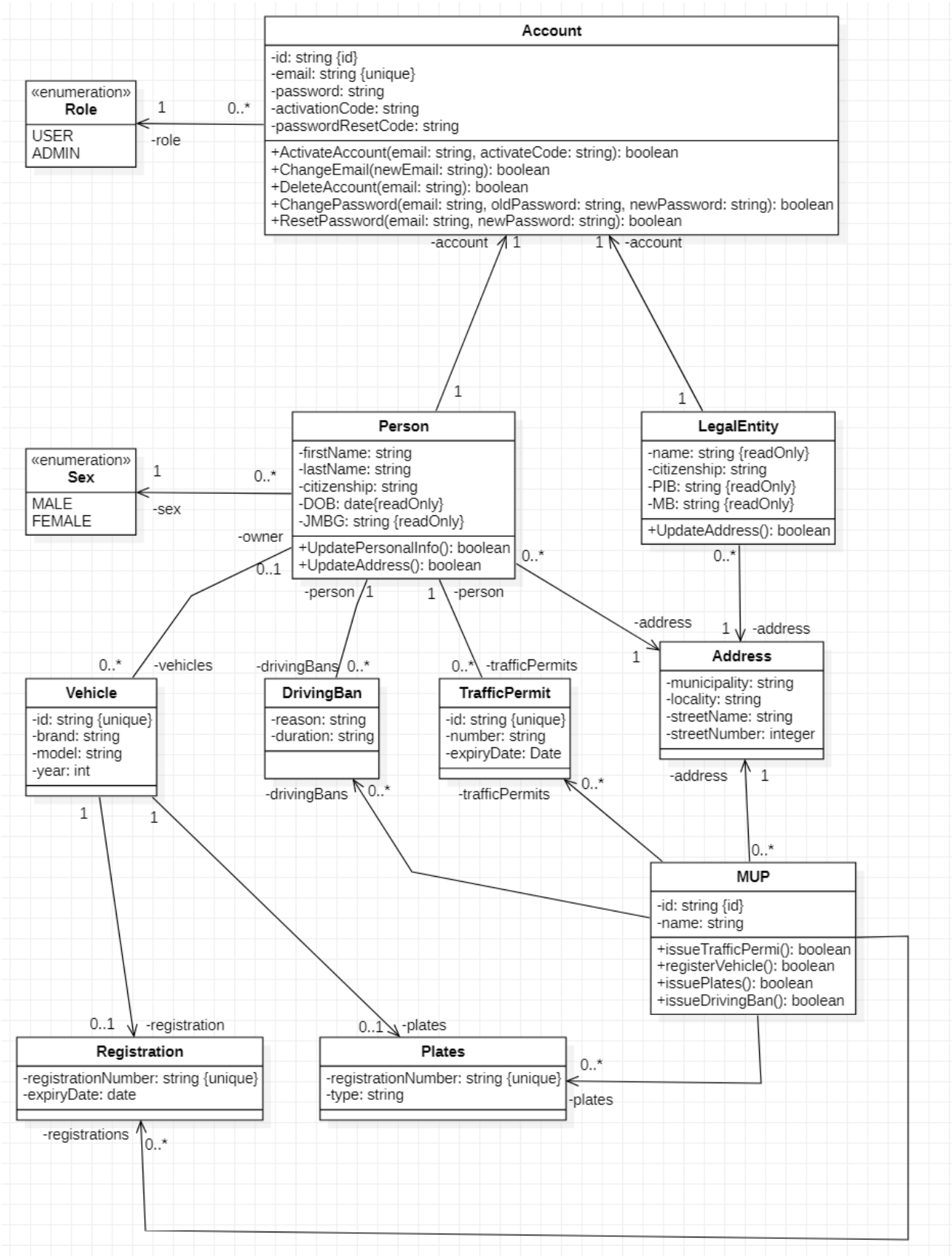
6. Specifikacija dizajna

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. **Korisnički interfejs (Frontend)** - React aplikacija, preko koje policajac, a po potrebi i saobraćajni policajac pristupaju sistemu kroz web pretraživač.
2. **Servisni sloj (Backend)** - Poslovna logika implementirana kroz Go programski jezik, koja u sebi sadrži sledeće komponente: servisni sloj, model podataka, handler-e za rukovanje HTTP zahtevima, rutiranje i vraćanje odgovarajućih grešaka.
3. **Baza podataka** - MongoDB noSQL baza podataka gde se čuvaju podaci o vozačima, vozačkim dozvolama i vozilima (registrovanim i neregistrovanim).

Dijagram klasa (Class diagram):



Na dijagramu se nalaze sledeći entiteti:

Vozilo

- **Atributi**:
    - vlasnik (JMBG) - string
    - broj registarskih tablica - string
    - model vozila - string
    - datum registracije - date
    - kategorija - string
- **Relacije**:
    - Jedan vozač može posedovati više vozila (0..*)
    - Jedno vozilo pripada tačno jednom vozaču (1)
- **Funkcionalnosti**:
    - Kreiranje novog vozila (registrovanje)
    - Dobavljanje svih vozila
    - Dobavljanje registrovanih vozila
    - Prebrojavanje vozila po kategoriji
    - Dobavljanje vozila po ID-ju
    - Dobavljanje vozila po kategoriji
    - Dobavljanje vozila po godini

Vozačka dozvola

- **Atributi**:
    - vlasnik (JMBG) - string
    - broj vozačke dozvole (UUID) - string
    - mesto izdavanja - string
    - lista kategorija - string
- **Funkcionalnosti**:
    - Kreiranje vozačke dozvole
    - Dobavljanje svih dozvola
    - Dobavljanje jedne dozvole po ID-ju
    - Dobavljanje jedne dozvole po JMBG-u vozača

Vozač

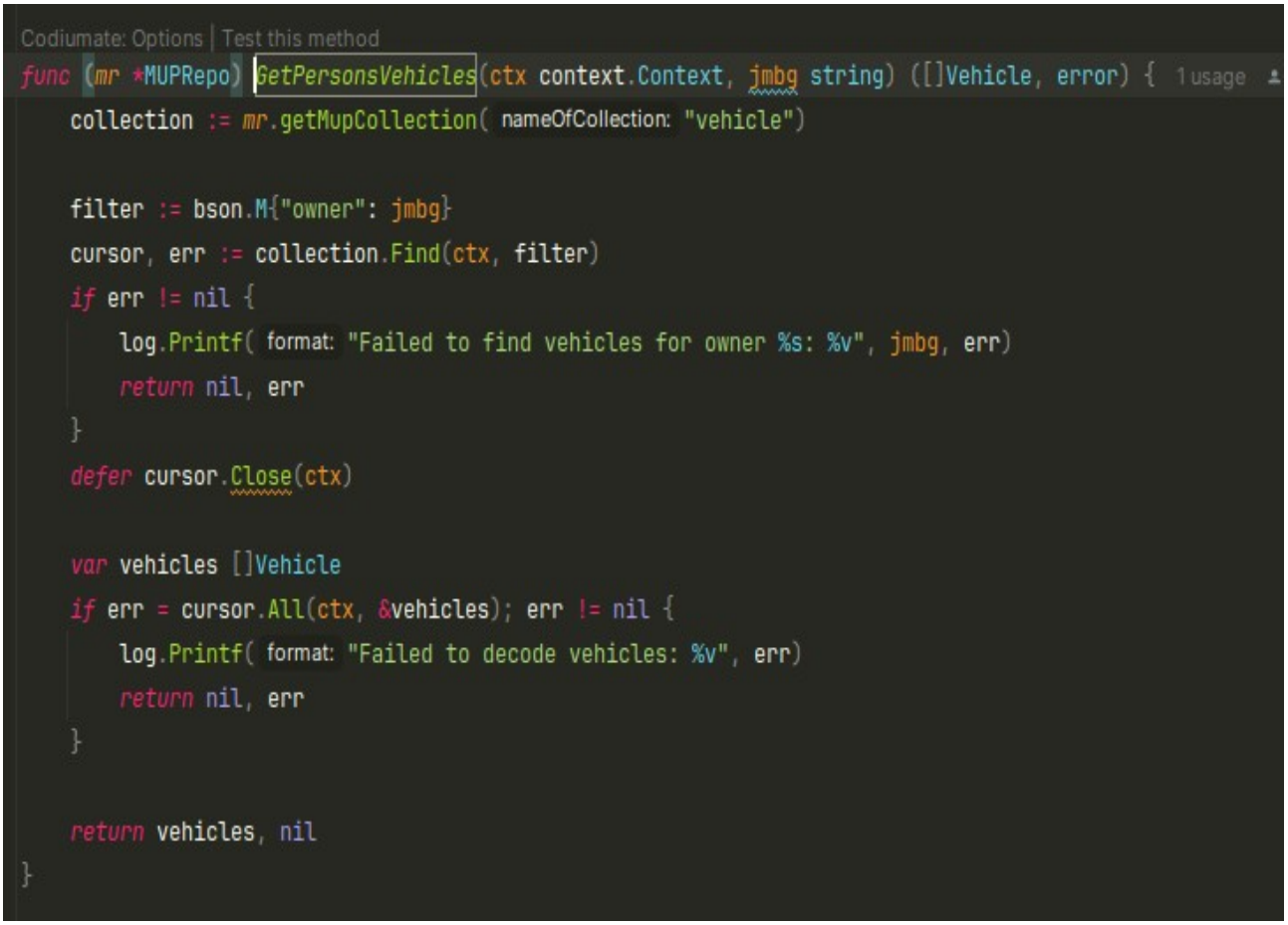
- **Atributi**:
    - JMBG - string
    - ime - string
    - prezime - string
    - datum rođenja - date
    - ima prekršaj - boolean
    - broj kaznenih poena - int
    - pol - string
- **Funkcionalnosti**:
    - Kreiranje vozača

- Dobavljanje svih vozača
- Dobavljanje vozača po ID-ju
- Unošenje prekršajnih poena

Saobraćajni policajac

- **Atributi**:
  - Uloga iz servisa saobraćajne policije
  - Pristup određenim API-jima iz ovog servisa

7. Implementacija sistema



Dobavljanje korisnikovih automobila.
Funkcija GetPersonsVehicles prvo poziva servis za autorizaciju, koji proverava da li je token poslat unutar HTTP header-a validan. Ako jeste, JMBG korisnika se ekstrahuje iz tokena. Ako ovo prođe uspešno, nastavlja se sa dobavljanjem vozila povezanih sa osobom.

Handler funkcija (GetPersonsVehicles u MupHandler strukturi):

1. Ekstrakcija tokena:

- tokenStr := mh.extractTokenFromHeader(r) - Ekstrahuje token iz HTTP zaglavlja zahteva.

2. Dobavljanje JMBG-a iz tokena:
    - jmbg, err := mh.getJMBGFromToken(tokenStr) - Ekstrahuje JMBG iz tokena.
    - Ako dođe do greške pri ekstrakciji JMBG-a, vraća se HTTP status 400 (Bad Request) i odgovarajuća poruka o grešci.
3. Dobavljanje vozila za datu osobu:
    - vehicles, err := mh.service.GetPersonsVehicles(r.Context(), jmbg) - Poziva se servisna metoda za dobavljanje vozila na osnovu JMBG-a.
    - Ako dođe do greške pri dobavljanju vozila, vraća se HTTP status 500 (Internal Server Error) i odgovarajuća poruka o grešci.
4. Odgovor sa podacima o vozilima:
    - Podešava se zaglavlje odgovora da je sadržaj JSON (ContentType, ApplicationJson).
    - Vraća se HTTP status 200 (OK) i podaci o vozilima u JSON formatu.
    - Ako dođe do greške pri enkodiranju vozila u JSON, vraća se HTTP status 500 (Internal Server Error) i odgovarajuća poruka o grešci.

Repo funkcija (GetPersonsVehicles u MUPRepo strukturi):

1. Dobavljanje kolekcije vozila:
    - collection := mr.getMupCollection("vehicle") - Dobija kolekciju vozila iz baze podataka.
2. Definisanje filtera:
    - filter := bson.M{"owner": jmbg} - Kreira filter za pretragu vozila na osnovu vlasnika (JMBG).
3. Pretraga kolekcije:
    - cursor, err := collection.Find(ctx, filter) - Pretražuje kolekciju vozila koristeći definisani filter.
    - Ako dođe do greške pri pretrazi, vraća se greška i nil vrednost za listu vozila.
4. Dekodiranje rezultata pretrage:
    - var vehicles []Vehicle - Definiše promenljivu za skladištenje dekodiranih rezultata.
    - err = cursor.All(ctx, &vehicles) - Dekodira rezultate pretrage u listu vozila.
    - Ako dođe do greške pri dekodiranju, vraća se greška i nil vrednost za listu vozila.
5. Vraćanje rezultata:
    - Vraća se lista vozila i nil vrednost za grešku ako je operacija uspešna.

```go
func (mh *MupHandler) CheckForPersonsDrivingBans(rw http.ResponseWriter, r *http.Request) {  1 usage  bunjo
    ctx := r.Context()

    tokenStr := mh.extractTokenFromHeader(r)

    jmbg, err := mh.getJMBGFromToken(tokenStr)
    if err != nil {
        fmt.Printf( format: "Error while reading JMBG from token: %v", err)
        http.Error(rw, FailedToReadUsernameFromToken, http.StatusBadRequest)
        return
    }

    drivingBans, err := mh.service.CheckForPersonsDrivingBans(ctx, jmbg)
    if err != nil {
        http.Error(rw,  error: "Failed to retrieve persons driving bans", http.StatusInternalServerError)
        return
    }

    rw.Header().Set(ContentType, ApplicationJson)
    rw.WriteHeader(http.StatusOK)
    if err := json.NewEncoder(rw).Encode(drivingBans); err != nil {
        http.Error(rw, FailedToEncodeDrivingBans, http.StatusInternalServerError)
    }
    fmt.Println( a...: "Successfully fetched driving bans")
}
```

```go
Codiumate: Options | Test this method
func (mr *MUPRepo) CheckForPersonsDrivingBans(ctx context.Context, userID string) (DrivingBans, error) {
    collection := mr.getMupCollection( nameOfCollection: "drivingBan")

    filter := bson.D{{ Key: "person",  Value: userID}}

    var drivingBans DrivingBans

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

    for cursor.Next(ctx) {
        var drivingBan DrivingBan
        if err := cursor.Decode(&drivingBan); err != nil { return nil, err }
        drivingBans = append(drivingBans, drivingBan)
    }

    if err := cursor.Err(); err != nil { return nil, err }

    return drivingBans, nil
}
```

Dobavljanje korisnikovih kazni.

Opis funkcije za proveru zabrane vožnje povezane sa osobom

Funkcija CheckForPersonsDrivingBans prvo poziva servis za autorizaciju, koji proverava da li je token poslat unutar HTTP header-a validan. Ako jeste, JMBG korisnika se ekstrahuje iz tokena. Ako ovo prođe uspešno, nastavlja se sa proverom zabrane vožnje povezanih sa osobom.

Handler funkcija (CheckForPersonsDrivingBans u MupHandler strukturi):

1. Dobavljanje JMBG-a iz tokena:
    - jmbg, err := mh.getJMBGFromToken(tokenStr) - Ekstrahuje JMBG iz tokena.
    - Ako dođe do greške pri ekstrakciji JMBG-a, vraća se HTTP status 400 (Bad Request) i odgovarajuća poruka o grešci.
2. Dobavljanje zabrana vožnje za datu osobu:
    - drivingBans, err := mh.service.CheckForPersonsDrivingBans(ctx, jmbg) - Poziva se servisna metoda za dobavljanje zabrana vožnje na osnovu JMBG-a.
    - Ako dođe do greške pri dobavljanju zabrana vožnje, vraća se HTTP status 500 (Internal Server Error) i odgovarajuća poruka o grešci.
3. Odgovor sa podacima o zabranama vožnje:
    - Podešava se zaglavlje odgovora da je sadržaj JSON (ContentType, ApplicationJson).
    - Vraća se HTTP status 200 (OK) i podaci o zabranama vožnje u JSON formatu.
    - Ako dođe do greške pri enkodiranju zabrana vožnje u JSON, vraća se HTTP status 500 (Internal Server Error) i odgovarajuća poruka o grešci.

Repo funkcija (CheckForPersonsDrivingBans u MUPRepo strukturi):

1. Dobavljanje kolekcije zabrana vožnje:
    - collection := mr.getMupCollection("drivingBan") - Dobija kolekciju zabrana vožnje iz baze podataka.
2. Definisanje filtera:
    - filter := bson.D{{Key: "person", Value: userID}} - Kreira filter za pretragu zabrana vožnje na osnovu vlasnika (JMBG).
3. Pretraga kolekcije:
    - cursor, err := collection.Find(ctx, filter) - Pretražuje kolekciju zabrana vožnje koristeći definisani filter.
    - Ako dođe do greške pri pretrazi, vraća se greška i nil vrednost za listu zabrana vožnje.
4. Dekodiranje rezultata pretrage:
    - var drivingBans []DrivingBan - Definiše promenljivu za skladištenje dekodiranih rezultata.
    - for cursor.Next(ctx) { - Iterira kroz rezultate pretrage.
        - err := cursor.Decode(&drivingBan) - Dekodira rezultate pretrage u listu zabrana vožnje.
        - Ako dođe do greške pri dekodiranju, vraća se greška i nil vrednost za listu zabrana vožnje.
        - drivingBans = append(drivingBans, drivingBan) - Dodaje dekodiranu zabranu vožnje u listu.
5. Vraćanje rezultata:

- Vraća se lista zabrana vožnje i nil vrednost za grešku ako je operacija uspešna.

Opis funkcije za dobavljanje registracija povezanih sa osobom

```go
func (mh *MupHandler) GetPersonsRegistrations(rw http.ResponseWriter, r *http.Request) {  1 usage  bunjo
    ctx := r.Context()

    tokenStr := mh.extractTokenFromHeader(r)
    jmbg, err := mh.getJMBGFromToken(tokenStr)
    if err != nil {
        http.Error(rw,  error: "Failed to read JMBG from token", http.StatusBadRequest)
        return
    }

    registrations, err := mh.service.GetPersonsRegistrations(ctx, jmbg)
    if err != nil {
        http.Error(rw,  error: "Failed to retrieve user registrations", http.StatusInternalServerError)
        return
    }

    rw.Header().Set( key: "Content-Type",  value: "application/json")
    rw.WriteHeader(http.StatusOK)
    if err := json.NewEncoder(rw).Encode(registrations); err != nil {
        http.Error(rw,  error: "Failed to encode registrations", http.StatusInternalServerError)
    }
}
```

```go
func (mr *MUPRepo) GetPersonsRegistrations(ctx context.Context, jmbg string) (Registrations, error) {
    collection := mr.getMupCollection( nameOfCollection: "registration")

    filter := bson.D{{ Key: "owner",  Value: jmbg}}

    var registrations Registrations

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

    for cursor.Next(ctx) {
        var registration Registration
        if err := cursor.Decode(&registration); err != nil { return nil, err }
        registrations = append(registrations, registration)
    }

    if err := cursor.Err(); err != nil { return nil, err }

    return registrations, nil
}
```

Dobavaljanje svih korisnikovih registrovanih auta.

Funkcija GetPersonsRegistrations prvo poziva servis za autorizaciju, koji proverava da li je token poslat unutar HTTP header-a validan. Ako jeste, JMBG korisnika se ekstrahuje iz tokena. Ako ovo prođe uspešno, nastavlja se sa dobavljanjem registracija povezanih sa osobom.

Handler funkcija (GetPersonsRegistrations u MupHandler strukturi):

1. Dobavljanje JMBG-a iz tokena:
    - jmbg, err := mh.getJMBGFromToken(tokenStr) - Ekstrahuje JMBG iz tokena.
    - Ako dođe do greške pri ekstrakciji JMBG-a, vraća se HTTP status 400 (Bad Request) i odgovarajuća poruka o grešci.
2. Dobavljanje registracija za datu osobu:
    - registrations, err := mh.service.GetPersonsRegistrations(ctx, jmbg) - Poziva se servisna metoda za dobavljanje registracija na osnovu JMBG-a.
    - Ako dođe do greške pri dobavljanju registracija, vraća se HTTP status 500 (Internal Server Error) i odgovarajuća poruka o grešci.
3. Odgovor sa podacima o registracijama:
    - Podešava se zaglavlje odgovora da je sadržaj JSON (ContentType, ApplicationJson).
    - Vraća se HTTP status 200 (OK) i podaci o registracijama u JSON formatu.
    - Ako dođe do greške pri enkodiranju registracija u JSON, vraća se HTTP status 500 (Internal Server Error) i odgovarajuća poruka o grešci.

Repo funkcija (GetPersonsRegistrations u MUPRepo strukturi):

1. Dobavljanje kolekcije registracija:
    - collection := mr.getMupCollection("registration") - Dobija kolekciju registracija iz baze podataka.
2. Definisanje filtera:
    - filter := bson.D{{Key: "owner", Value: jmbg}} - Kreira filter za pretragu registracija na osnovu vlasnika (JMBG).
3. Pretraga kolekcije:
    - cursor, err := collection.Find(ctx, filter) - Pretražuje kolekciju registracija koristeći definisani filter.
    - Ako dođe do greške pri pretrazi, vraća se greška i nil vrednost za listu registracija.
4. Dekodiranje rezultata pretrage:
    - var registrations []Registration - Definiše promenljivu za skladištenje dekodiranih rezultata.
    - for cursor.Next(ctx) { - Iterira kroz rezultate pretrage.
        - err := cursor.Decode(&registration) - Dekodira rezultate pretrage u listu registracija.
        - Ako dođe do greške pri dekodiranju, vraća se greška i nil vrednost za listu registracija.

- registrations = append(registrations, registration) - Dodaje dekodiranu registraciju u listu.

5. Vraćanje rezultata:
    - Vraća se lista registracija i nil vrednost za grešku ako je operacija uspešna.

```go
func (mr *MUPRepo) SubmitRegistrationRequest(ctx context.Context, registration *Registration) error {
    collection := mr.getMupCollection( nameOfCollection: "registration")

    _, err := collection.InsertOne(ctx, registration)
    if err != nil {
        log.Printf( format: "Failed to create vehicle: %v", err)
        return err
    }

    err = mr.SaveRegistrationIntoMup(ctx, registration)
    if err != nil {
        log.Printf( format: "Failed to save registration into mup: %v", err)
        return err
    }

    err = mr.SaveRegistrationIntoVehicle(ctx, registration)
    if err != nil {
        log.Printf( format: "Failed to save registration into vehicle: %v", err)
        return err
    }

    return nil
}
func (ms *MupService) SubmitRegistrationRequest(ctx context.Context, registration *data.Registration) error {
    registration.Approved = false
    registration.IssuedDate = time.Now()
    registration.ExpirationDate = registration.IssuedDate
    registration.RegistrationNumber = utils.GenerateRegistration()
    registration.Plates = ""

    err := ms.repo.SubmitRegistrationRequest(ctx, registration)
    if err != nil { return err }

    err = ms.repo.SaveRegistrationIntoVehicle(ctx, registration)
    if err != nil { return err }

    return nil
}
```

Opis funkcije za podnošenje zahteva za registraciju
Funkcija SubmitRegistrationRequest omogućava podnošenje zahteva za registraciju vozila. Proces uključuje kreiranje i čuvanje registracije u bazi podataka.

Repo funkcija (SubmitRegistrationRequest u MUPRepo strukturi):

1. Dobavljanje kolekcije registracija:
    - collection := mr.getMupCollection("registration") - Dobija kolekciju registracija iz baze podataka.
2. Umetanje registracije u kolekciju:
    - _, err := collection.InsertOne(ctx, registration) - Umeće novu registraciju u kolekciju.
    - Ako dođe do greške pri umetanjanju, vraća se greška i zapisuje se poruka o grešci u log.
3. Čuvanje registracije u MUP:
    - err = mr.SaveRegistrationIntoMup(ctx, registration) - Čuva registraciju u MUP sistemu.
    - Ako dođe do greške pri čuvanju, vraća se greška i zapisuje se poruka o grešci u log.
4. Čuvanje registracije u vozilo:
    - err = mr.SaveRegistrationIntoVehicle(ctx, registration) - Čuva registraciju u vozilu.
    - Ako dođe do greške pri čuvanju, vraća se greška i zapisuje se poruka o grešci u log.
5. Vraćanje rezultata:
    - Ako su svi koraci uspešni, vraća se nil vrednost za grešku.

Servisna funkcija (SubmitRegistrationRequest u MupService strukturi):

1. Postavljanje početnih vrednosti za registraciju:
    - registration.Approved = false - Postavlja se vrednost da registracija nije odobrena.
    - registration.IssuedDate = time.Now() - Postavlja se datum izdavanja na trenutni datum i vreme.
    - registration.ExpirationDate = registration.IssuedDate - Postavlja se datum isteka na datum izdavanja.
    - registration.RegistrationNumber = utils.GenerateRegistration() - Generiše se broj registracije.
    - registration.Plates = "" - Postavlja se da tablice budu prazne.
2. Podnošenje zahteva za registraciju:
    - err := ms.repo.SubmitRegistrationRequest(ctx, registration) - Poziva se repo funkcija za podnošenje zahteva za registraciju.
    - Ako dođe do greške, vraća se greška.
3. Čuvanje registracije u vozilu:
    - err = ms.repo.SaveRegistrationIntoVehicle(ctx, registration) - Poziva se repo funkcija za čuvanje registracije u vozilu.
    - Ako dođe do greške, vraća se greška.
4. Vraćanje rezultata:
    - Ako su svi koraci uspešni, vraća se nil vrednost za grešku.

```go
func (mh *MupHandler) ApproveRegistration(rw http.ResponseWriter, r *http.Request) { 1 usage  bunjo
    var registration data.Registration

    if err := json.NewDecoder(r.Body).Decode(&registration); err != nil {
        http.Error(rw, FailedToDecodeRequestBody, http.StatusBadRequest)
        log.Printf( format: "Failed to decode request body: %v", err)
        return
    }

    if err := mh.service.ApproveRegistration(r.Context(), registration); err != nil {
        log.Printf( format: "Failed to approve registration: %v", err)
        http.Error(rw,  error: "Failed to approve registration", http.StatusInternalServerError)
        return
    }

    registration.Approved = true

    rw.Header().Set(ContentType, ApplicationJson)
    rw.WriteHeader(http.StatusOK)
    if err := json.NewEncoder(rw).Encode(registration); err != nil {
        log.Printf( format: "Failed to encode approved registration: %v", err)
        http.Error(rw,  error: "Failed to encode approved registration", http.StatusInternalServerError)
    }
    log.Printf( format: "Successfully updated registration '%s'", registration.RegistrationNumber)
}

func (ms *MupService) ApproveRegistration(ctx context.Context, registration data.Registration) error { 1 usage
    expirationDate := time.Now().AddDate( years: 5,  months: 0,  days: 0)
    registration.Approved = true
    registration.ExpirationDate = expirationDate

    platesNumber := utils.GeneratePlates()

    plates := data.Plates{
        RegistrationNumber: registration.RegistrationNumber,
        PlatesNumber:       platesNumber,
        PlateType:          "vehicle plates",
        VehicleID:          registration.VehicleID,
        Owner:              registration.Owner,
    }

    registration.Plates = platesNumber

    err := ms.repo.ApproveRegistration(ctx, registration)
    if err != nil { return err }

    return ms.repo.IssuePlates(ctx, plates)
}
```

Opis funkcije za odobravanje registracije:

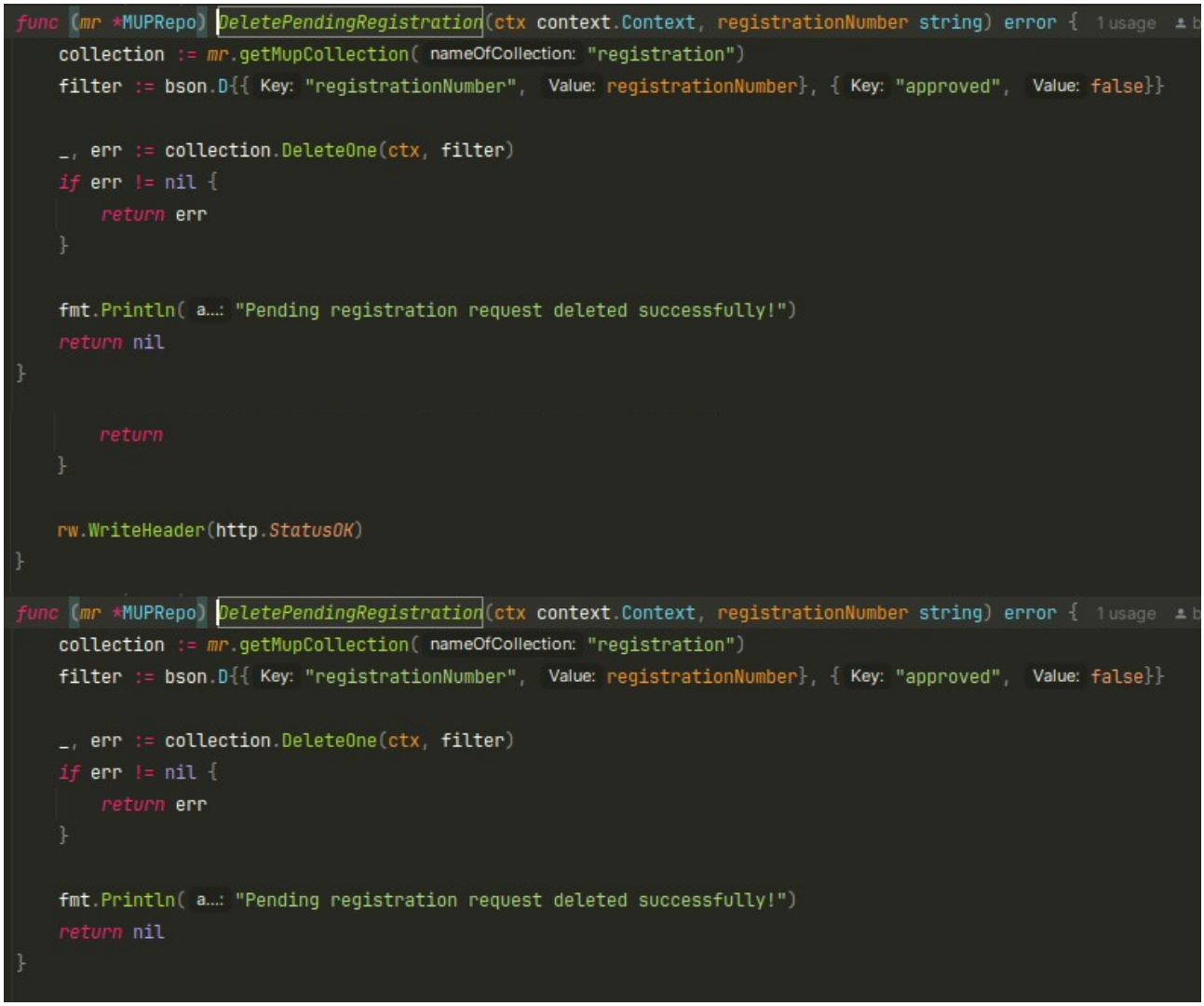
Funkcija ApproveRegistration omogućava odobravanje registracije vozila. Proces uključuje dekodiranje registracionih podataka iz HTTP zahteva, odobravanje registracije, ažuriranje statusa registracije i čuvanje tablica u bazi podataka.

Handler funkcija (ApproveRegistration u MupHandler strukturi):

1. Dekodiranje registracionih podataka:
    - err := json.NewDecoder(r.Body).Decode(&registration) - Dekodira podatke o registraciji iz tela HTTP zahteva.
    - Ako dođe do greške pri dekodiranju, vraća se HTTP status 400 (Bad Request) i odgovarajuća poruka o grešci.
2. Odobravanje registracije:
    - err := mh.service.ApproveRegistration(r.Context(), registration) - Poziva se servisna metoda za odobravanje registracije.
    - Ako dođe do greške pri odobravanju registracije, vraća se HTTP status 500 (Internal Server Error) i odgovarajuća poruka o grešci.
3. Ažuriranje statusa registracije:
    - registration.Approved = true - Ažurira se status registracije na odobreno.
4. Odgovor sa ažuriranim podacima o registraciji:
    - Podešava se zaglavlje odgovora da je sadržaj JSON (ContentType, ApplicationJson).
    - Vraća se HTTP status 200 (OK) i ažurirani podaci o registraciji u JSON formatu.
    - Ako dođe do greške pri enkodiranju podataka o registraciji u JSON, vraća se HTTP status 500 (Internal Server Error) i odgovarajuća poruka o grešci.

Servisna funkcija (ApproveRegistration u MupService strukturi):

1. Postavljanje početnih vrednosti za registraciju:
    - expirationDate = time.Now().AddDate(5, 0, 0) - Postavlja se datum isteka registracije na 5 godina od trenutnog datuma.
    - registration.ExpirationDate = expirationDate - Ažurira se datum isteka registracije.
    - registration.Approved = true - Postavlja se status registracije na odobreno.
2. Generisanje broja tablica:
    - platesNumber := utils.GeneratePlates() - Generiše se broj tablica za vozilo.
3. Kreiranje objekta tablica:
    - plates := data.Plates{...} - Kreira se objekat tablica sa svim potrebnim podacima.
4. Čuvanje ažurirane registracije:
    - err := ms.repo.ApproveRegistration(ctx, registration) - Poziva se repo funkcija za čuvanje ažurirane registracije.
    - Ako dođe do greške, vraća se greška.
5. Čuvanje tablica:
    - return ms.repo.IssuePlates(ctx, plates) - Poziva se repo funkcija za čuvanje tablica u bazi podataka.



```
func (mr *MUPRepo) DeletePendingRegistration(ctx context.Context, registrationNumber string) error {
    collection := mr.getMupCollection( nameOfCollection: "registration")
    filter := bson.D{{ Key: "registrationNumber",  Value: registrationNumber}, { Key: "approved",  Value: false}}

    _, err := collection.DeleteOne(ctx, filter)
    if err != nil {
        return err
    }

    fmt.Println( a...: "Pending registration request deleted successfully!")
    return nil
}

        return
    }

    rw.WriteHeader(http.StatusOK)
}
func (mr *MUPRepo) DeletePendingRegistration(ctx context.Context, registrationNumber string) error {
    collection := mr.getMupCollection( nameOfCollection: "registration")
    filter := bson.D{{ Key: "registrationNumber",  Value: registrationNumber}, { Key: "approved",  Value: false}}

    _, err := collection.DeleteOne(ctx, filter)
    if err != nil {
        return err
    }

    fmt.Println( a...: "Pending registration request deleted successfully!")
    return nil
}
```

Opis funkcije za brisanje neodobrene registracije

Funkcija DeletePendingRegistration omogućava brisanje neodobrene registracije iz sistema. Proces uključuje validaciju broja registracije, brisanje registracije iz baze podataka i vraćanje odgovarajućeg HTTP odgovora.

Handler funkcija (DeletePendingRegistration u MupHandler strukturi):

1. Ekstrakcija broja registracije iz URL-a:
    - vars := mux.Vars(r) - Ekstrahuje varijable iz URL-a.
    - request := vars["request"] - Dobija broj registracije iz varijabli URL-a.
    - Ako broj registracije nije prosleđen, vraća se HTTP status 400 (Bad Request) i odgovarajuća poruka o grešci.
2. Pozivanje servisne metode za brisanje registracije:
    - err := mh.service.DeletePendingRegistration(ctx, request) - Poziva se servisna metoda za brisanje neodobrene registracije.

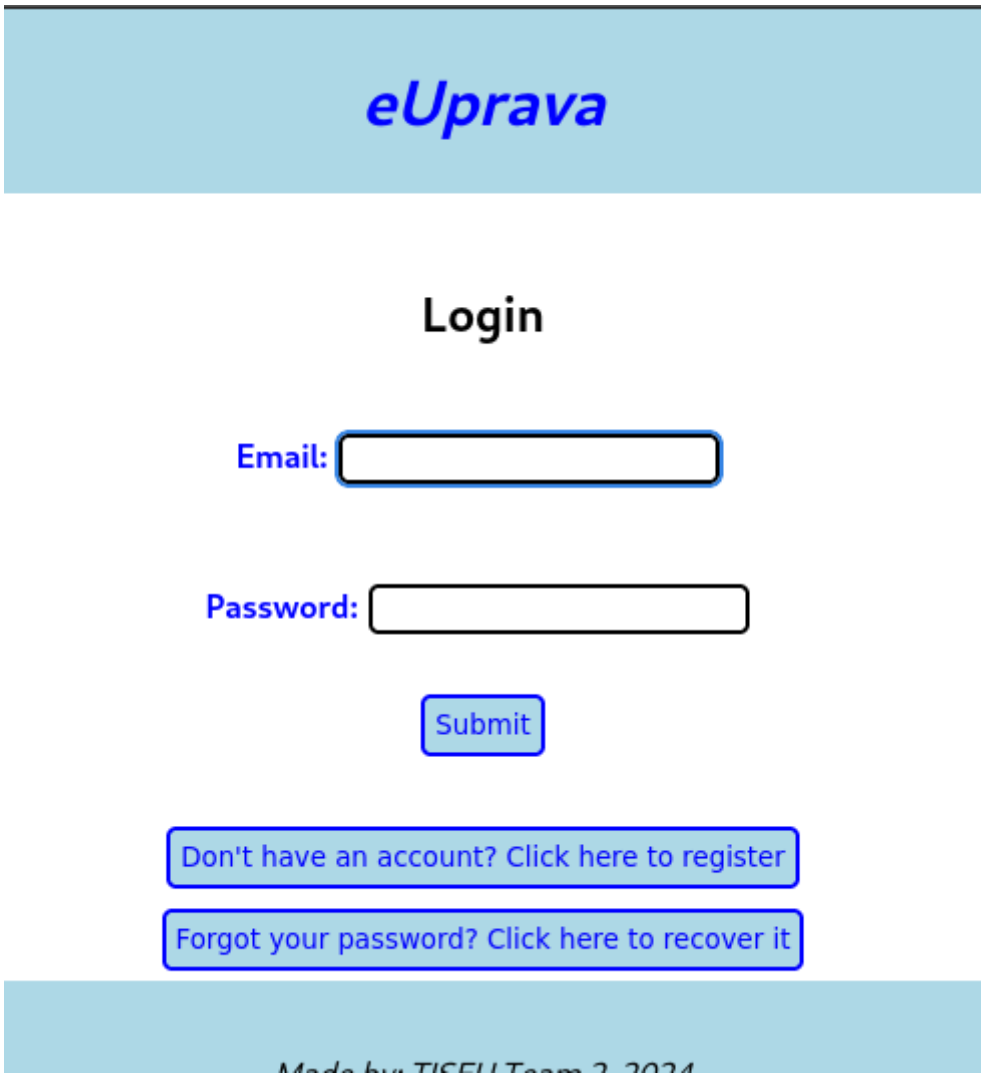
    - Ako dođe do greške pri brisanju, vraća se HTTP status 500 (Internal Server Error) i odgovarajuća poruka o grešci.
3. Vraćanje uspešnog odgovora:
    - rw.WriteHeader(http.StatusOK) - Vraća se HTTP status 200 (OK) ako je operacija uspešna.

Repo funkcija (DeletePendingRegistration u MUPRepo strukturi):

1. Dobavljanje kolekcije registracija:
    - collection := mr.getMupCollection("registration") - Dobija kolekciju registracija iz baze podataka.
2. Definisanje filtera za pretragu registracije:
    - filter := bson.D{{Key: "registrationNumber", Value: registrationNumber}, {Key: "approved", Value: false}} - Kreira filter za pretragu neodobrene registracije na osnovu broja registracije i statusa odobrenja.
3. Brisanje registracije iz kolekcije:
    - _, err := collection.DeleteOne(ctx, filter) - Briše jednu registraciju iz kolekcije koristeći definisani filter.
    - Ako dođe do greške pri brisanju, vraća se greška.
4. Vraćanje uspešnog rezultata:
    - Ako je operacija uspešna, vraća se nil vrednost za grešku i zapisuje se poruka o uspešnom brisanju u log.

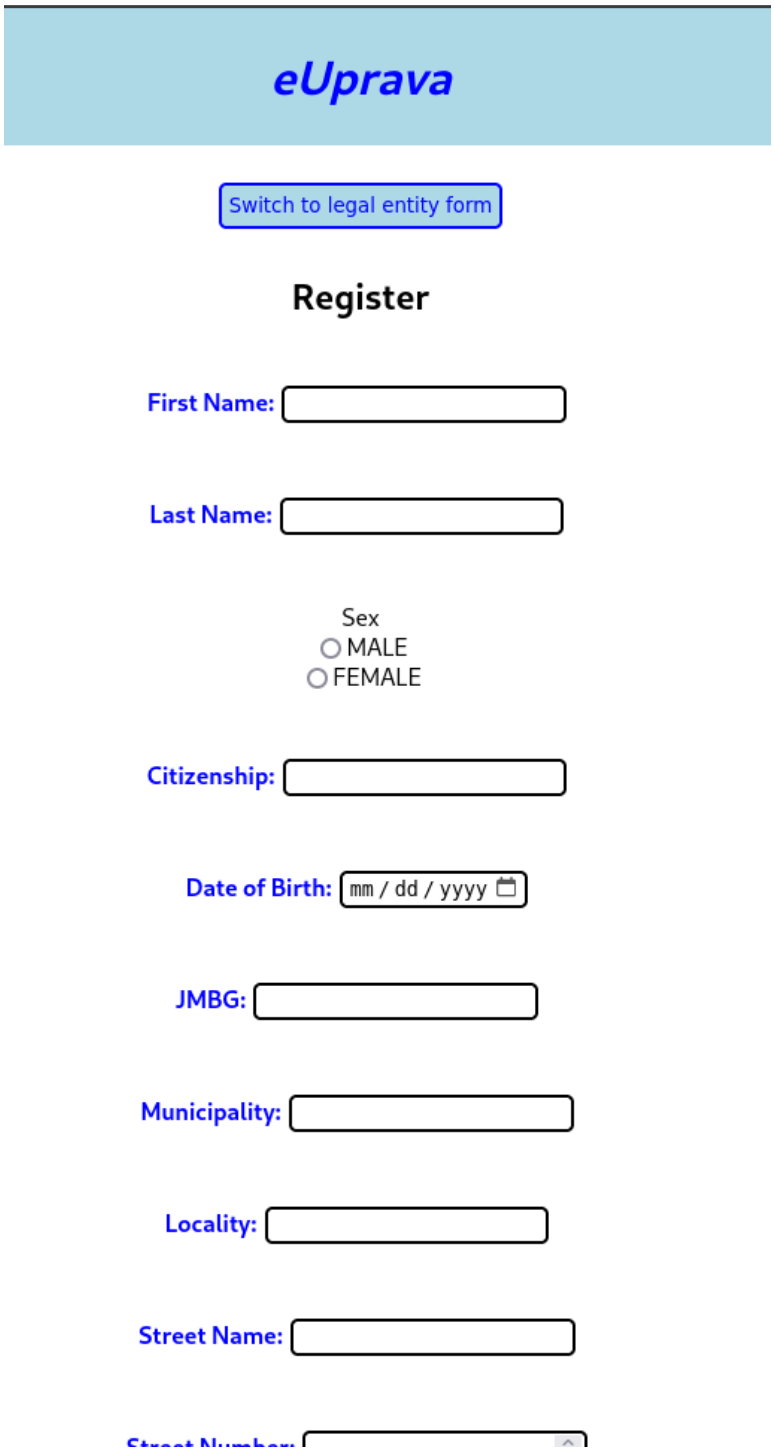
9. Demonstracija

Prijava



Ovo je ekran za prijavu u sistem eUprava. Sadrži polja za unos email adrese i lozinke, označene plavim etiketama. Dugme "Submit" omogućava slanje unetih podataka za prijavu. Ispod forme su dva linka: jedan za registraciju novih korisnika ("Click here to register") i drugi za oporavak zaboravljene lozinke ("Click here to recover it").

Registracija



Ovo je ekran za registraciju u sistem eUprava. Na vrhu ekrana je naziv sistema i dugme koje prebacuje na formu za registraciju pravnih lica. Forma sadrži polja za unos osnovnih informacija o korisniku: ime, prezime, pol (muški/ženski), državljanstvo, datum rođenja, JMBG, opštinu, lokalitet, naziv ulice i broj ulice.

Početna stranica

Ovo je početni ekran sistema eUprava nakon prijave korisnika. Na vrhu ekrana je naziv sistema i pozdravna poruka koja uključuje ime korisnika ("Welcome to eUprava, Mika"). Korisnik može izabrati jedan od četiri dostupna servisa: MUP, Saobraćajna policija (Traffic Police), Sud (Court), i Institut za statistiku (Institute for Statistics). Svaki servis je predstavljen plavim dugmetom. Na dnu ekrana se nalazi dugme "Logout".

Prikaz svih korisnikovih vozila

Ovo je ekran sistema eUprava koji prikazuje informacije o vozilima korisnika. Na vrhu ekrana je naziv sistema. Ispod toga su kartice koje prikazuju podatke o svakom vozilu:

- **Marka vozila (Brand)**: Prikazuje marku vozila, npr. Renault, Škoda.
- **Model vozila (Model)**: Prikazuje model vozila, npr. Talisman, Fabia.
- **Godina proizvodnje (Year)**: Prikazuje godinu proizvodnje vozila.
- **Broj registracije (Registration)**: Prikazuje broj registracije vozila, ili poruku da vozilo nije registrovano.
- **Tablice (Plates)**: Prikazuje registarske tablice vozila, ili poruku da vozilo nije registrovano.

Za svako neregistrovano vozilo se nalazi dugme za registraciju, koje salje zahtev da se odobri.

Prikaz korisnikovih kazni

Ovo je ekran u sistemu eUprava koji prikazuje informaciju o zabrani vožnje za korisnikov nalog. Na ekranu se nalazi poruka "No driving bans for your account" koja znači da korisnik nema zabranu vožnje.

Ovo je ekran u sistemu **eUprava** koji prikazuje listu zahteva za registraciju vozila. Na ekranu je prikazan jedan zahtev sa sledećim informacijama:

- **Broj registracije (Number)**: Jgy8GWCP
- **Datum izdavanja (Issued date)**: 2024-06-30 22:56:51.651
- **Datum isteka (Expiration date)**: 2024-06-30 22:56:51.651

Kartica je stilizovana plavom bojom sa crnim okvirom. Ispod informacija o registraciji nalaze se dva dugmeta: "Approve" za odobravanje zahteva i "Delete" za brisanje zahteva. Na vrhu ekrana nalazi se naslov "List of registration requests" i dugme za zatvaranje u gornjem desnom uglu.

9.Zaključak

U ovom seminarskom radu opisan je projekat **eUprava** koji olakšava rad sa korisnicima, njihovim vozilima, registracijama i zabranama vožnje. Primena modernih web tehnologija i mikroservisne arhitekture u sistemima poput ovog donosi značajne prednosti u efikasnosti, skalabilnosti i održavanju ovih sistema.

Kroz razne funkcionalnosti, kao što su prijava i registracija korisnika, dobavljanje informacija o vozilima, vozačkim dozvolama i zabranama vožnje, te mogućnost odobravanja ili brisanja registracija, videli smo koliko ova tehnologija skraćuje vreme potrebno za dobavljanje informacija i njihov prenos između različitih servisa. Zadaci koji su nekada morali da se obavljaju ručno sada su značajno olakšani i automatizovani.

Mikroservisi pružaju izolovanost i nezavisnost od drugih sistema, što ubrzava razmenu i skladištenje informacija. Kroz ovaj projekat prikazano je kako tehnologija može olakšati obradu podataka, pružiti korisnicima intuitivan interfejs i unaprediti rad zaposlenih lica, čime se postiže veća produktivnost i efikasnost u radu.