APLIKACIJA ZA OGLAŠAVANJE AUTOMOBILA

Vrdoljak, Karlo

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

KARLO VRDOLJAK

Z A V R Š N I R A D

APLIKACIJA ZA OGLAŠAVANJE AUTOMOBILA

Split, rujan 2020.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

Predmet: Programiranje na internetu

Z A V R Š N I R A D

Kandidat: Karlo Vrdoljak

Naslov rada: Aplikacija za oglašavanje automobila

Mentor: Marina Rodić, predavač

Split, rujan 2020.

Sadržaj

Sažetak

Primarni cilj aplikacije je omogućiti korisnicima oglašavanje svojih automobila i jednostavniju prodaju, odnosno kupovinu istih. Tehnologije koje su korištene za izradu poslužiteljskog dijela aplikacije (engl. *backend*) su razvojni okviri (engl. *framework*) *NestJS* i *TypeORM*. Klijentska strana je izrađena u radnom okviru *Angular*. Za oblikovanje i prikaz korisničkog sučelja (engl. *user interface*) korištene su biblioteke (engl. *library*) *tailwindcss*, *PrimeNG* i *ngx-gallery*.

Ključne riječi: kupovina, prodaja, automobili, *Angular*, *NestJS*

Summary

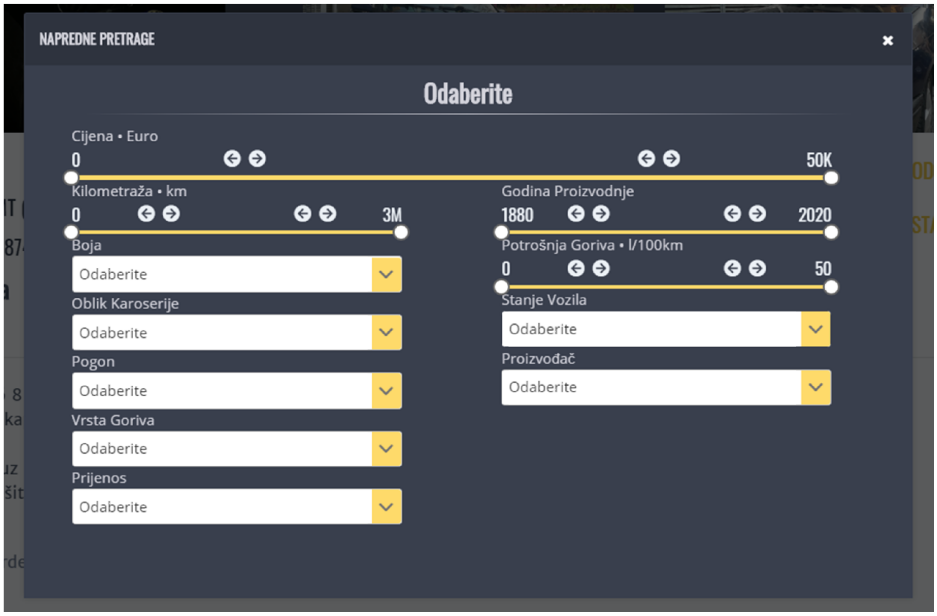
Car advertising application

The main goal of this bachelor thesis is to enable users to advertise their cars, and make it easier to sell or buy them. The server side technologies include development frameworks such as NestJS and TypeORM, and client side is built using Angular framework. The libraries tailwindcss, PrimeNG and ngx-gallery were also used to design and display the user interface.

Keywords: purchase, sell, cars, Angular, NestJS

1. Uvod

Ideja ovog završnog rada je izraditi aplikaciju za kupnju i prodaju osobnih automobila, gdje će korisnici brzo i jednostavno doći do željenog automobila, kao i napraviti vlastiti oglas za svoj automobil. Važna stavka aplikacije je učinkovita pretraga, gdje je moguće precizno tražiti oglase automobila, koristeći neke ili sve od ponuđenih pojmova za pretragu (*Slika 1*).



Slika 1: *Napredno pretraživanje oglasa*

Korisnici mogu dodati pronađene oglase u svoje omiljene, odnosno spremiti ih za kasniji pregled kako bi lakše pratili određeni oglas. Omogućeno je i komentiranje oglasa kako bi kupci mogli dobiti odgovore na moguće nejasnoće ukoliko ih ima. Vlasnici oglasa imaju mogućnost generiranja kupoprodajnog ugovora, gdje imaju slobodan izbor načina ispunjavanja istog. Mogu učitati podatke o kupcu ukoliko on ima kreiran račun, učitati svoje podatke i podatke vezane za automobil.

Oglasi mogu imati status *aktivan*, *neaktivan* i *izbrisan.* Početni status svakog oglasa je *aktivan*, što znači da je oglas vidljiv i dostupan svima. Oglasi sa stanjem *neaktivan* su vidljivi isključivo vlasniku oglasa. Na posljetku status *izbrisan* označava kako oglas nije vidljiv na oglasniku te će zauvijek biti izbrisan nakon 30 dana.

U sljedećem poglavlju su opisane korištene tehnologije za izradu aplikacije, potom je navedena specifikacija aplikacije gdje su navedene sve važne funkcionalnosti

uz detaljnu razradu istih uz pripadajuće primjere. Na posljetku, u zaključku se iznose prednosti i mane korištenih tehnologija uz primjer mogućih poboljšanja same aplikacije.

2. Korištene tehnologije

Aplikacija se može podijeliti na dvije logičke cijeline, a to su poslužiteljski dio i klijentski dio. Temelj poslužiteljskog dijela je napisan u programskom jeziku *TypeScript* koji je razvijen od strane *Microsoft Corporation*. Temelj ovog programskog jezika je *JavaScript*, programski jezik više razine (engl. *high-level*) za kojeg se smatra da je jezgra svih tehnologija kojima je radno okruženje na WWW (engl. *World Wide Web*).

Osim toga, u izradi poslužiteljskog dijela korišten je razvojni okvir *NestJS* čija je uloga odgovarati na HTTP (engl. *Hypertext Transfer Protocol*) upite, obrada poslanih podataka, komunikacija s bazom podataka preko biblioteke *TypeORM* i autentikacija koristeći JWT (engl. *JSON Web Tokens*), putem biblioteke *Passport*.

Za klijentski dio zadužen je *Angular*, razvojni okvir koji krajnjem korisniku omogućuje prikaz i doživljaj same aplikacije. *Angular* pruža mnoštvo alata kojima je konačni produkt brza i prilagodljiva aplikacija na svim uređajima. Za ovu aplikaciju, korištenjem dodatnih bibiloteka kao što su *ngx-gallery*, *SwiperJS*, *ngx-scrollreveal*, postiže se moderno korisničko sučelje koje omogućuje ugodnije korištenje same aplikacije.

U nastavku poglavlja opisane su tehnologije i funkcionalnosti koje su ključne za rad aplikacije uz pripadajuće primjere te zaključak gdje su navedene prednosti i mane.

2.1. TypeScript

TypeScript je razvijen iz programskog jezika *JavaScript*. Glavne funkcionalnosti koje pruža *TypeScript* su tipovi podataka (engl. *data type*), proizvoljno ulančavanje (engl. *optional chaining*), operator sjedinjenja ništavila (engl. *nullish coalescing operator*). Funkcionalnosti su prikazane u sljedećem primjeru kôda (*ispis 1*).

```
interface Type { foo: number; bar: string; }

class Feature { A: string; B: any; C: Type; }

let feature = {
  A: null,
  B: { X: 0, Y: '' },
  C: null
```

```
} as Feature;
console.log(feature.C?.bar); // undefined
console.log(feature.C ?? 'none'); // 'none'
console.log(feature.C?.bar, feature.C && feature.C.bar);
// undefined null
```

Ispis 1: Prikaz sintakse i načina rada operatora i tipova

TypeScript je uglavnom korišten u objektno orjentiranom pristupu, gdje bi svaki objekt trebao imati svoj definirani tip podatka, ukoliko nije jedan od primitivnih kao što su *number* ili *string*. Programeru to omogućuje poznavanje modela podataka, koji se koriste u aplikaciji i olakšava rad. Korištenje tipova nije nužan uvjet i može se koristiti objekt bez tipa deklarirajući ga kao *any*.

Optional chaining je vrlo korisna funkcionalnost, koja skrati kôd prilikom uvjetovanja svojstva (engl. *property*) objekta. Važno je naglasiti da ukoliko nije definirano svojstvo koje se provjerava, rezultat će biti *undefined*, dok rezultat provjere bez ove funkcionalnosti je *null* ili *undefined*.

Nullish coalescing služi kako bi se svojstvima kojima je vrijednost *null* ili *undefined*, pridodjelila druga inicijalna vrijednost. Osim toga, nudi dodatno smanjenje količine koda i brz način provjere vrijednosti nekog svojstva.

2.2. NestJS

Temeljna inspiracija arhitekture razvojnog okvira *NestJS* je pronađena u *Angularu*. Glavne stavke su brza konfiguracija, organiziranost samog projekta i mogućnost stvaranja skalabilne poslužiteljske aplikacije.

Zadatak svake poslužiteljske aplikacije je odgovaranje na HTTP zahtjeve. *NestJS* tu stavku odrađuje na sljedeći način (*ispis 2*).

```
@Controller('todos')
class TodosController {
    @Get()
    getAllTodos() {
        return this.todosRepository.getAllTodos();
    }
    @Post()
    createToDo(@Body() newToDoData: NewToDoData) {
        return this.todosRepository.saveToDo(newToDoData);
    }
}
```

Ispis 2: *Način prihvaćanja HTTP upita i odgovor na iste*

Kao što se vidi iz primjera, rukovanje HTTP upitima je vrlo jednostavno i svodi se na kreiranje klase (engl. *Class*) u kojoj se nalaze ključne riječi nakon znaka @. Ključna riječ

@Controller govori kako će se klasa navedena u nastavku, koristiti kao kontroler. Odnosno vezat će se za određenu putanju, kako bi se dobio očekivani HTTP odgovor. Nadalje *@Get* i *@Post* su samo neke od podržanih ključnih riječi iz popisa HTTP metoda koje definiraju tip zahtjeva.

Obrada korisničkih zahtjeva uglavnom podrazumijeva rad nad podacima koji se nalaze u bazi podataka. *NestJS* koristi vanjsku biblioteku *TypeORM,* koja omogućuje jednostavan i učinkovit način rukovanja nad entitetima (engl. *Entity*) (*ispis 3*).

```
@Entity()
class Dog {
    @PrimaryGeneratedColumn()
    id: number;

    @Column()
    name: string;

    @Column()
    ownerName: string;
}
```

Ispis 3: *Način definiranja entiteta*

Iako strukturno identično, koriste se druge ključne riječi, specifične za tablicu u bazi podataka. U primjeru je lako uočiti ključne riječi:

- *@Entity*, opisuje definiranu klasu kao tablicu u bazi podataka
- *@PrimaryGeneratedColumn,* označava kako će polje u tablici s nazivom *id* biti tip *integer*, koji će automatski poprimati novu vrijednost uvećanu za jedan prilikom svakog novog unosa u tablicu i na posljetku
- *@Column*, definira svojstvo kao stupac u bazi podataka.

Stavka koja je najvažnija svakoj aplikaciji je sigurnost. *NestJS* pruža svoju implementaciju vanjske biblioteke *Passport* koja nudi mogućnost zaštite putem JWT (engl. *JSON web token*) tokena, koja je danas sve više u upotrebi. Postavljanje rada biblioteke *Passport* je opisano u službenoj dokumentaciji okvira *NestJS.*

```
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor() {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: jwtConstants.secret,
    });
  }

  async validate(payload: any) {
    return {
      userId: payload.sub, username: payload.username,
      roles: payload.roles
    };
  }
}
```

Ispis 4: *Implementacija JWT token zaštite*

Implementacija *JwtStrategy* klase koja nasljeđuje klasu *PassportStrategy* omogućava aplikaciji da putem konstruktora navedene klase konfigurira JWT zaštitu. Pozivajući konstruktor nadklase koristeći funkciju *super* se prosljeđuju parametri koji služe biblioteci *Passport* za konfiguraciju JWT zaštite.

Svojstvo *jwtFromRequest* pruža metodu preko koje će razvojni okvir izdvojiti JWT token iz HTTP zahtjeva gdje gore navedena vrijednost nalaže da će *Passport* koristiti standardnu izvedbu, a to je čitanje iz zaglavlja preko svojstva *authorization.* Vrijednost svojstva *authorization* mora biti tekstualan s početnom riječi *Bearer*.

Svojstvo *ignoreExpiration* je oznaka koja može imati vrijednost *true* ili *false* te preko nje *Passport* donosi odluku o provjeri isteka valjanosti JWT tokena.

Na posljetku, *secretOrKey* je simetrični ključ tekstualnog oblika koji služi za potpisivanje stvorenog JWT tokena.

Klasa *PassportStragey* omogućuje i metodu *validate* koja od biblioteke kao parametar dobije dekodirani objekt vrijednosti iz tokena. Omogućuje provedbu autorizacije i autentičnosti pojedinog korisnika koji stvara upit na sadržaj aplikacije.

2.3. Angular

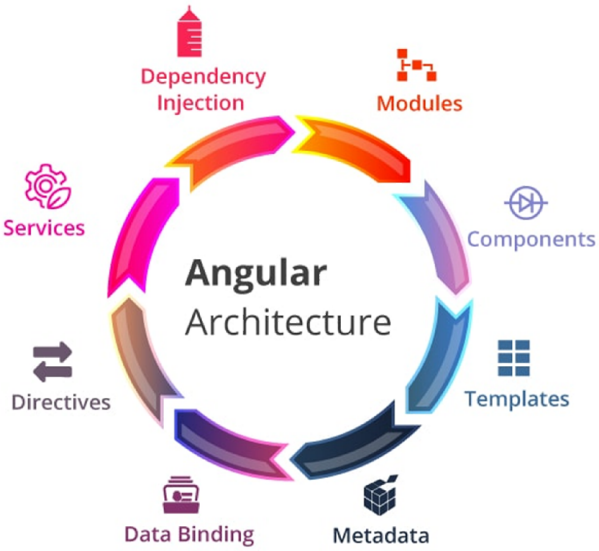
Angular je razvojno okruženje na klijentskoj strani pisan u *TypeScriptu* i razvijan od strane *Google LLC*. Danas je među najpopularnijim razvojnim okruženjima za izradu jednostraničnih *web* aplikacija (engl. *single-page application*). Glavne stavke koje definiraju *Angular*:

- **Modul**, definiran kao samostalna cjelina koja sadrži mnoštvo funkcionalnosti

koje pružaju komponente, servise i direktive.

- **Komponenta**, cjelina unutar modula koja pruža jednoznačnu funcionalnost, koja se može višestruko iskoristiti unutar istog konteksta aplikacije.
- **Servisi** (engl. *Service*), višestruko iskoristljive funkcionalnosti, koje su neovisne o pogledu (engl. *View*)
- **Direktive** (engl. *Directive*), posebni atributi koji nadopunjuju sintaksu za HTML datoteke.
- **Predložak**, HTML datoteke koje definiraju kako će pogled izgledati na ekranima korisnika.
- **Vezanje podataka**, (engl. *Data Binding*) podrazumijeva sinkronizaciju između podataka i pogleda.

Navedene stavke opisuju glavne pojmove arhitekture svake *Angular* aplikacije, no potrebno je pročitati službenu dokumentaciju koja je detaljna i jasno opisuje kako razvojno okruženje funkcionira. Za bolje razumijevanje kako radi *Angular* priložena je skica u kojoj je prikazana arhitektura *Angular* aplikacije (*slika 2*).



Slika 2: *Arhitektura Angular aplikacije*

Arhitektura *Angulara* na *slici 2* prikazuje faze kreiranja pogleda:

- **Injekcija ovisnosti** (engl. *Dependency injection*), u ovoj fazi *Angular* učitava sve potrebne ovisnosti (engl. *dependencies*) koje su objekti potrebni nekoj klasi kako bi ona mogla obaljati svoju funkciju.
- **Moduli**, po završetku injekcije ovisnosti *Angular* učitava sve potrebne module aplikacije.

- **Predlošci**, stvara se HTML sadržaj iz predložaka pojedinog modula.
- **Meta podaci** (engl. *Metadata*), nakon stvaranja HTML sadržaja učitavaju se meta podaci modula.
- **Vezanje podataka**, *Angular* sinkronizira podatke iz modula s podacima pogleda.
- **Direktive**, stvara HTML sadržaj iz posebnih atributa, ovom fazom *Angular* završava učitavanje sadržaja za prikaz.
- **Servisi**, na posljetku, učitaju se svi potrebni servisi koji služe modulu aplikacije za obavljanje svoje funkcije.

Angular koristi ključne riječi za razlikovanje vrste funkcionalnosti koje pruža definirana klasa. Implementacija je prikazana u sljedećem primjeru (*ispis 5*).

```
@Component({ ... })
export class AppComponent { ... }

@Injectable({ ... })
export class AppService { ... }

@Directive({ ... })
export class AppDirective { ... }

@NgModule({
  declarations: [ AppComponent ],
  providers: [ AppService, AppDirective ],
})
export class AppModule { }
```

Ispis 5: *Prikaz registriranja klasa s različitim funkcionalnostima u aplikaciji*

Glavne ključne riječi, koje definiraju način korištenja klasa *Angular* aplikacije su:

- ***@Component***, osnovni element svake *Angular* aplikacije za izradu korisničkog sučelja se zove komponenta. Svaka komponenta sadrži svoj birač (engl. *selector*), predložak i pripadnu *css* datoteku.

- ***@Injectable***, anotacija koja označava pojedinu klasu kao servis, koja se može koristiti u bilo kojoj komponenti aplikacije

- ***@Directive***, klasa pod ovom anotacijom se koristi isključivo u predlošku komponente kao nadopuna HTML i DOM strukture.

- ***@NgModule***, označava klasu koja koristi komponente s meta podacima koji kompajleru opisuju kako će se modul koristiti u aplikaciji. Odnosno, kako će se modul moći uključiti u životni ciklus aplikacije uz sve popratne funkcionalnosti. Klase pod dekoratorom *@Injectable ili @Directive* se

uključuju pod *providers* u modulu, dok *@Component* klase se uključuju pod *declarations*.

2.4. Dodatne biblioteke korisničkog sučelja

Svaka aplikacija je stvorena u svrhu rješavanja nekog poslovnog problema ili pružanja određenog skupa funkcionalnosti ili usluga. Korisnici će odabrati aplikaciju koja ne samo najbolje rješava njihove potrebe nego i zbog pozitivnog dojma kojeg im je aplikacija ostavila. Takav dojam može pridobiti korisnike koji kasnije shvate da im je baš takva aplikacija nužna.

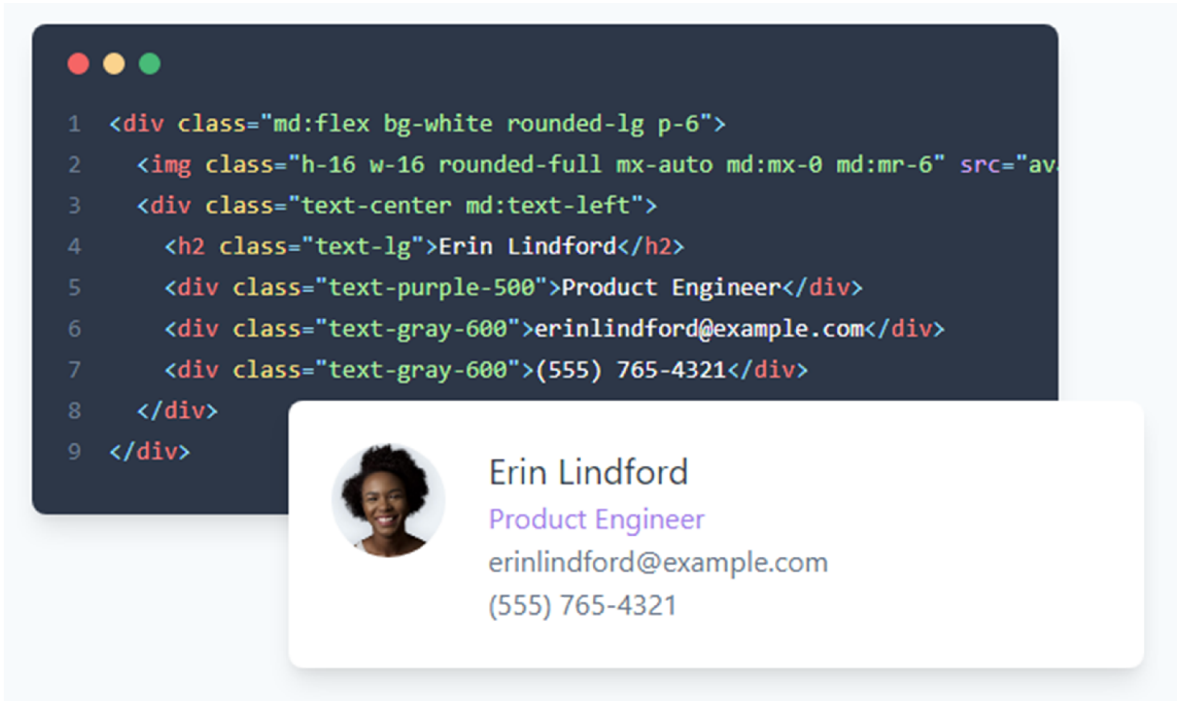
Kako bi se aplikacija mogla razlikovati od drugih i ostaviti pozitivni dojam korisnicima, stvoreni su razni razvojni okviri ili biblioteke korisničkog sučelja. Njihova svrha je nadopuniti dizajn i omogućiti moderan pristup podacima te kvalitetniji doživljaj aplikacije.

U ovoj aplikaciji, očekivano je da će korisnici prenositi svoje fotografije kako bi mogli lakše prodati ili kupiti određeni automobil, stoga su odabrane biblioteke *ngx-gallery* i *SwiperJS* koje stvaraju ugodniji dojam pregleda i unosa fotografija (*slika 3* i *slika 5*).

Slika 3: *Prikaz funkcionalnosti koju pruža ngx-gallery biblioteka*

Slika 5: *Prikaz funkcionalnosti SwiperJS biblioteke*

Nadalje korištena je biblioteka *tailwindcss*, koja je skup *CSS* (engl. *Cascading Style Sheets*) klasa. Na ovaj način programeru je olakšano pisanje *CSS* kôda i dana mu je standardizirana paleta boja, kako bi dizajn bio profesionalan (*slika 2.4.3*).



Slika 6: *Prikaz mogućnosti tailwindcss biblioteke*

Zanimljivost biblioteke je u tome što za dizajn koji je prikazan na slici 6 nije napisana niti jedna linija *CSS* kôda.

3. Funkcionalna specifikacija aplikacije

U ovom poglavlju opisan je način rada aplikacije, odnosno zadaci i funkcionalnosti koje obavlja poslužiteljski i klijentski dio aplikacije.

3.1. Poslužitelj

Kao što je navedeno u uvodu ovog rada, poslužiteljski dio aplikacije obavlja *NestJS* razvojni okvir, u kombinaciji s pomoćnim bibliotekama. Svaka aplikacija mora imati svoju početnu strukturu iz koje se dalje razvijaju mnoge funkcionalnosti. Sljedeća poglavlja će redom opisati strukturu poslužitelja i na posljetku zadatke koje obavlja.

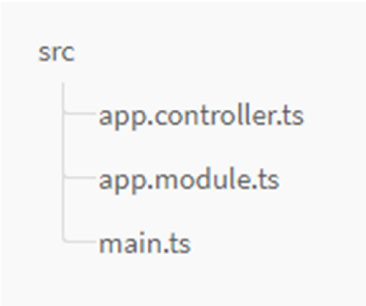
3.1.1. Inicijalizacija

Kako bi opisali funkcionalnosti ove aplikacije potrebno je prikazati način njenog nastajanja. U sljedećem primjeru su prikazani koraci kreiranja novog *NestJS* projekta (*ispis 6*).

```
$ npm i -g @nestjs/cli
$ nest new project-name
```

Ispis 6: *Prikaz inicijalizacije novog NestJS projekta*

Novostvoreni projekt sada ima strukturu kao na slici 7.



Slika 7: *Struktura NestJS projekta*

U direktoriju *src* nalaze se dvije datoteke koje su ključne za rad aplikacije i datoteka *app.controller.ts* koja je primjer korištenja kontrolera. Datoteka main.ts pokreće proces pokretanja aplikacije kao što je prikazano na ispisu 7.

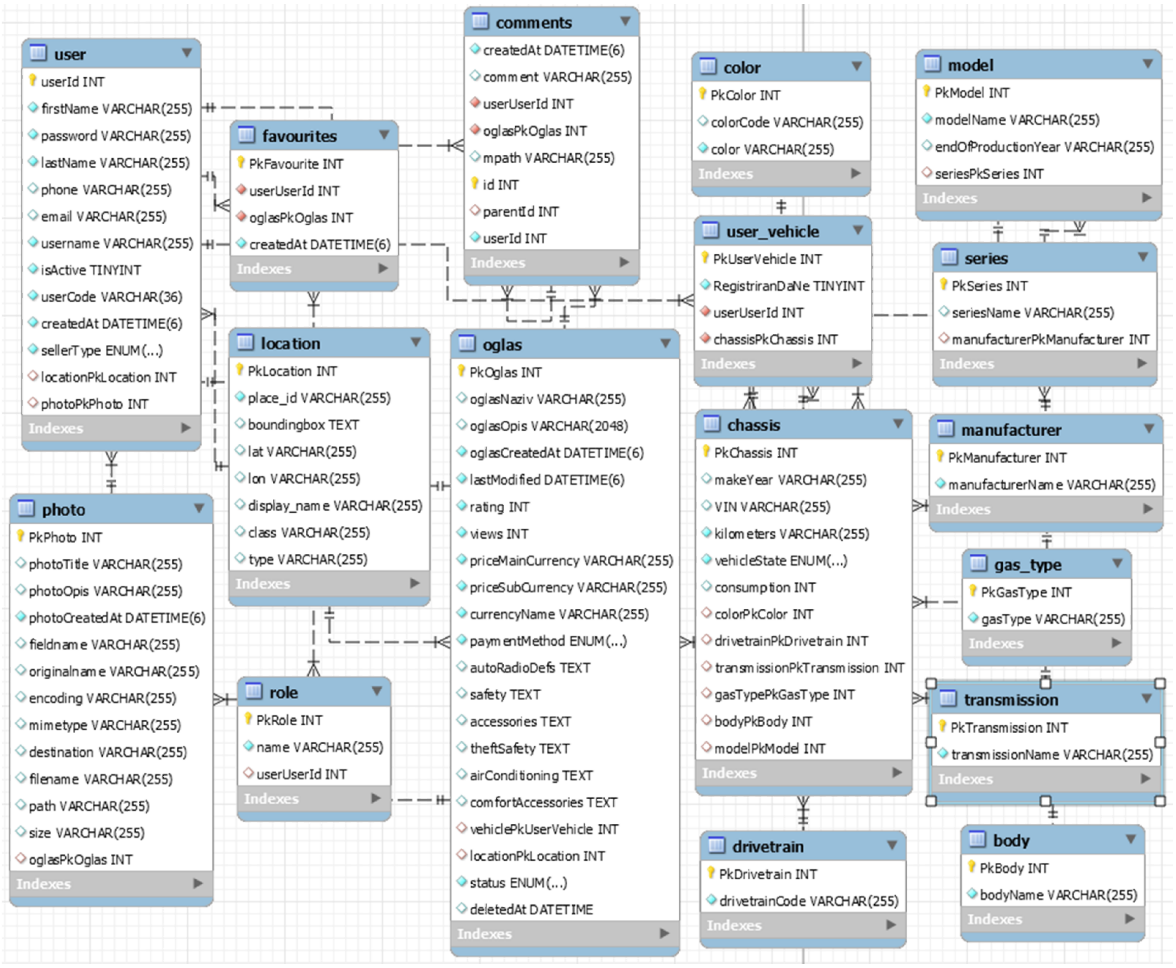
```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  await app.listen(3000);
}
bootstrap();
```

Ispis 7: *Sadržaj main.ts datoteke*

3.1.2. Baza podataka i *TypeORM*

NestJS pruža sve potrebne alate za korištenje baze podataka u aplikaciji, preko vanjske biblioteke *TypeORM*. Naziv biblioteke je nastao iz dvije ključne riječi, a to su Type, što označava da su entiteti objekti strogog tipa i ORM (engl. *Object-relational mapping*) koji je programska tehnika koja pretvara dvije nekompatibilne strukture podataka, kao što su tablice i relacije u relacijskoj bazi podataka, u objekte koje aplikacija može koristiti.

Samim time programer ne mora poznavati arhitekturu i način korištenja relacijske baze podataka, već radi u poznatom okruženju koji je sastavni dio aplikacije. U ovom slučaju koristi se *MySQL*, sustav za upravljanje bazama podataka. Na sljedećem ER (engl. *Entity Relationship*) dijagramu je prikazana struktura baze podataka ove aplikacije (*slika 8*).



Slika 8: *ER dijagram baze podataka*

Aplikacija se služi sa složenim modelom podataka koji se sastoji od 17 tablica, stoga je uloga paketa *TypeORM* vrlo korisna. Dohvaćanje samo jednog automobila iz oglasnika je proces koji obuhvaća nekoliko tablica. Takvo dohvaćanje je trivijalno korištenjem višetrukih poziva asinkronih funkcija, koje su detaljno objašnjene u pripadajućoj dokumentaciji. U sljedećem primjeru je prikazan isječak kôda u kojem se dohvaća jedan automobil (*ispis* 8).

```
this.oglasRepository.createQueryBuilder('o')
    .leftJoinAndSelect(„o.photos“,“p“,“p.oglas“)
    .leftJoinAndSelect(„o.vehicle“,“v“)
    .leftJoinAndSelect(„o.location“,“l“)
    .leftJoinAndSelect(„v.user“,“u“)
    .leftJoinAndSelect(„u.location“,“loc“)
    .leftJoinAndSelect(„v.chassis“,“ch“)
    .leftJoinAndSelect(„ch.color“,“c“)
    .leftJoinAndSelect(„ch.model“,“ml“)
    .leftJoinAndSelect(„ch.drivetrain“,“dt“)
    .leftJoinAndSelect(„ch.transmission“,“tr“)
    .leftJoinAndSelect(„ch.gasType“,“gt“)
    .leftJoinAndSelect(„ch.body“,“b“)
    .leftJoinAndSelect(„ml.series“,“s“)
    .leftJoinAndSelect(„s.manufacturer“,“m“)
    .where('o.PkOglas = :PkOglas', { PkOglas: pk })
    .getOne();
```

Ispis 8: *Dohvat jednog vozila koristeći TypeORM*

Kao što je već navedeno, ne treba poznavati pisanje SQL kôda, nego samo implementaciju koju pruža biblioteka. ORM omogućava da se TypeScript kod u pozadini izvrši kao SQL upit. Rezultat je dohvaćen kao jedan objekt, koji poslužiteljska aplikacija može proslijediti prema klijentu.

3.1.3. Pretraga

Korisnici imaju mogućnost za tri vrste pretrage:

- Slobodni unos
- Jednostavna
- Napredna

Biblioteka TypeORM omogućava zaobilaženje vlastitih ograničenja putem metode *query*, koja kao parametar prima sirovi (eng. raw) SQL upit. Koristeći metodu slobodnog unosa, pretraživanje se oslanja na tekstualno polje koje prihvaća bilo kakav niz znakova. Korisnik navede pretragu koja može biti primjerice „opel astra“ i takav tekst se prenosi iz klijentskog dijela aplikacije na poslužitelj i dohvaćaju se svi oglasi koji sadrže ključne riječi „opel“ ili „astra“. Važno je istaknuti kako će rezultati biti sortirani prema relevantnosti upita, što znači da će se u nizu oglasa prvo prikazati oni koji sadrže najveći broj ponavljanja svih riječi iz upita.

Funkcionalnost koja to omogućuje je *fulltext* indeks, koja grupira sav tekstualni sadržaj SQL upita u privremenu tablicu, koja omogućuje pretragu preko ključnih riječi. Rezultati takve pretrage se moraju sortirati po relevantnosti, stoga *fulltext* pretrage imaju definiran algoritam, preko kojeg se računa brojčana vrijednost koliku jedan rezultat pretrage ima prednost nad drugim.

Način rada algoritma je definiran matematičkim načelima. Početna pretpostavka nalaže ukoliko se termini pretrage postave u dvodimenzionalni prostor, može se primjeniti jednostavna aritmetika kako bi se došlo do izračuna relevantnosti pojedinog rezultata pretrage (*formula 1)*.

```
w = tf * idf
```

Formula 1: *Težina (w) je frekvencija termina pretrage (tf) što množi inverz frekvencije dokumenta (idf). Dokument je oznaka za jedan red u nizu rezulata*

Navedena formula je početna ideja, koju je bilo potrebno proširiti kako bi se mogla koristiti u *MySQL* (*formula 2*).

```
w = (log(dtf)+1)/sumdtf * U/(1+0.0115*U) * log((N-nf)/nf)
```

Formula 2: *Izračun težine u MySQL*

Pojmovi formule su redom:

- **dtf:** broj ponavljanja termina u dokumentu
- **sumdtf:** $\sum(\log(dtf) + 1)$, za svaki termin iz istog dokumenta
- **U:** broj jedinstvenih termina u dokumentu
- **N:** broj dokumenata
- **nf:** broj dokumenata koji sadrže termin

Formula sadrži tri dijela: *glavni dio, normalizacijski faktor* i *globalni multiplikator.* Glavni dio formule je jednadžba $(\log(dtf) + 1)/sumdtf$. Normalizacijski faktor je formula koja ostvaruje povećanje opće težine što je manja količina rezultata. Umnoškom glavnog dijela i normalizacijskog faktora dobije se broj koji se koristi kao indeks u *fulltext* pretragama. Globalni multiplikator napravi inverz dobivenog indeksa i tako se dobije rezultantnu frekvenciju ponavljanja termina u pretrazi.

Konačna formula, čiji se rezultat može koristiti za sortiranje prema relevantnosti je prikazana ispod (*formula* 3).

```
R = w * qf;
```

Formula 3: *Relevantnost (R) je težina (w) što množi rezultantnu frekvenciju ponavljanja termina (qf)*

Na sljedećem primjeru je prikazano kako ovaj princip radi kada se postavi upit pretrage (*ispis 9*).

```
mysql> SELECT id, title, body, MATCH (title,body)  AGAINST ('database' IN BOOLEAN MODE)
AS score FROM articles ORDER BY score DESC;
+----+-------------------------------+----------------------------------------+----------------------+
| id | title                         | body                                   | score                |
+----+-------------------------------+----------------------------------------+----------------------+
|  6 | Database, Database, Database | database database database             |  1.0886961221694946 |
|  3 | Optimizing Your Database      | In this database tutorial  ...         | 0.36289870738983154 |
|  1 | MySQL Tutorial                | This database tutorial  ...            | 0.18144935369491577 |
|  2 | How To Use MySQL              | After you went through a  ...          |                    0 |
|  4 | MySQL vs. YourSQL             | When comparing databases  ...          |                    0 |
|  5 | MySQL Security                | When configured properly, MySQL  ...   |                    0 |
|  7 | 1001 MySQL Tricks             | 1. Never run mysqld as root. 2.  ...   |                    0 |
|  8 | MySQL Full-Text Indexes       | MySQL fulltext indexes use a ..        |                    0 |
+----+-------------------------------+----------------------------------------+----------------------+
8 rows in set (0.00 sec)
```

Ispis 9: *Rezultati koristeći fulltext pretrage s terminom „database“*

Jednostavni oblik pretrage radi na principu odabira, gdje korisnik mora odabrati tri pojma, a to su proizvođač, serija i model automobila. *Fulltext* pretragama se izoliraju oni oglasi čiji nazivi se podudaraju s odabranim pojmovima. Rezultat se također sortira po relevantnosti preko gore navedene formule.

Napredan oblik pretrage omogućuje korisniku stroži način pretraživanja, gdje će dobiti točno one oglase koji ga zanimaju. Na slici 1.1 u poglavlju *Uvod* je prikazano kako izgleda napredna pretraga u aplikaciji.

Poslužitelj, nakon što dohvati rezultate napredne pretrage, mora obaviti provjeru koliko je puta pojedini oglas bio dodan u omiljene i napraviti odluku o tome je li trenutni korisnik aplikacije već dodao isti oglas u svoje omiljene (*ispis 10*).

```
@Get('advanced')
async getOglasiAdvancedQuery(@Request() req, @Res() res: Response) {
 let oglasi= await this.vehicleService.findOglasiByAllProps(req.query);

 this.fillOglasPhotos(oglasi).then(async ret => {
    if(req.headers?.authorization?.split('Bearer ')) {
      let token = req.headers.authorization.split('Bearer ')[1];
      ret = await this.checkFavourite(ret,token);
    } else {
      ret = await this.checkFavourite(ret,null);
    }
    res.status(HttpStatus.OK).send(ret);
  });
}
```

Ispis 10: *Prikaz dohvata oglasa i prosljeđivanje rezulata logičkih operacija*

Napomena, navedena logika iz primjera se odnosi za sve oblike pretrage, ne samo za napredne pretrage.

3.1.4. Omiljeni oglasi

Korisnici imaju mogućnost dodavanja oglasa u svoje omiljene. Funkcionalnost radi na način da postoji veza između korisnika i oglasa. Korisnik može pregledati svoje ili tuđe omiljene oglase. Svaki oglas ima broj dodavanja prikazan (*ispis 11*).

```
@UseGuards(JwtAuthGuard,RolesGuard)
@Roles('user')
@Post('/add/favourite')
async toggleToFav(@Request() req, @Res() res: Response) {
  let fav = await this.userService.oglasService.getConnection()
        .createQueryBuilder(Favourites,'f')
        .where('f.userUserId = :id', {id : req.user.userId})
        .andWhere('f.oglasPkOglas = :pkOglas',
           {pkOglas: req.body.PkOglas})
        .getOne();
  if(!fav) {
    let favourite = new Favourites();
    favourite.oglas = await this.oglasService
      .findOglasByPk(req.body.PkOglas);
    favourite.user = await this.userService.findOne(req.user.username);
    let result = await this.userService.oglasService.getConnection()
      .createQueryBuilder()
      .insert().into(Favourites)
      .values(favourite).execute();
    if(result) {
      res.status(HttpStatus.OK).send(true);
    } else {
      res.status(HttpStatus.EXPECTATION_FAILED).send();
    }
  } else {
    await this.userService.favService.getRepo().remove(fav);
    res.status(HttpStatus.OK).send(false);
  }
}
```

Ispis 11: *Prikaz funkcionalnosti dodavanja oglasa u svoje omiljene*

Samo prijavljeni korisnici mogu dodavati oglas u svoje omiljene i imati mogućnost pregleda svojih omiljenih. Provjera mogućnosti dodavanja je odrađena na klijenskom dijelu, ali kao dodatni sloj zaštite vrši se provjera i na poslužiteljskoj strani, koristeći ključne riječi *@UseGuards* i *@Roles*. Objašnjenje ovih ključnih riječi je opisano u poglavlju *Sigurnost*.

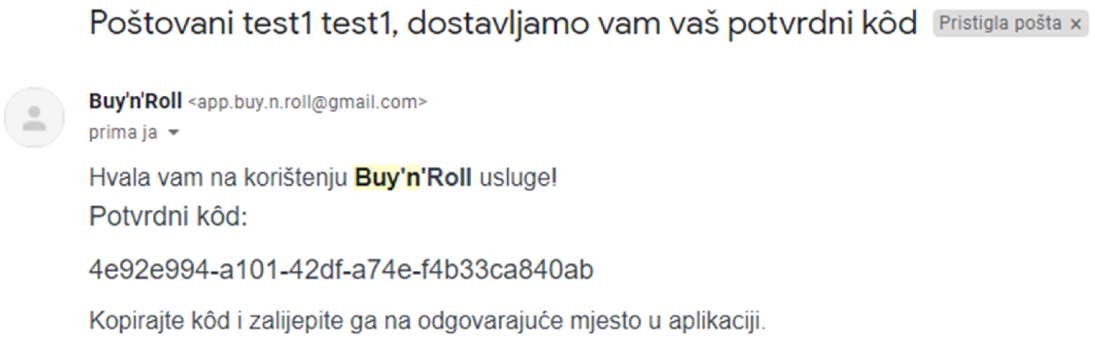
3.1.5. Korisnici

Korisnici su nužan uvjet svake aplikacije. Ova aplikacija razlikuje barem dvije vrste korisnika, a to su anonimni i prijavljeni korisnici. Anonimni korisnici mogu samo pretražiti i pregledati oglase, pregledati profile drugih korisnika gdje imaju mogućnost pregleda njihovih oglasa i omiljenih oglasa. Prijavljeni korisnici imaju sve prethodno navedene uz nekoliko dodatnih mogućnosti, a to su:

- stvaranje, pregled, izmjena i ažuriranje statusa oglasa te generiranje kupoprodajnog ugovora
- dodavanje, brisanje i pregled svojih omiljenih oglasa
- pregled korisnika koji su dodali korisnikov oglas u svoje omiljene

Kako bi korisnici postali prijavljeni, potrebno je obaviti registracijski proces koji uključuje standardnu formu koju treba popuniti, stoga primjeri nisu potrebni za razumijevanje.

Kao dodatna sigurnost autentičnosti registracije korisnika, dodana je zaštita putem e-pošte u kojoj se pošalje potvrdni kôd koji korisnik mora unijeti po završetku registracije. Prilikom prijave, ukoliko kôd nije unesen, aplikacija će tražiti unos kôda i poslati isti ponovo (*slika 9*).

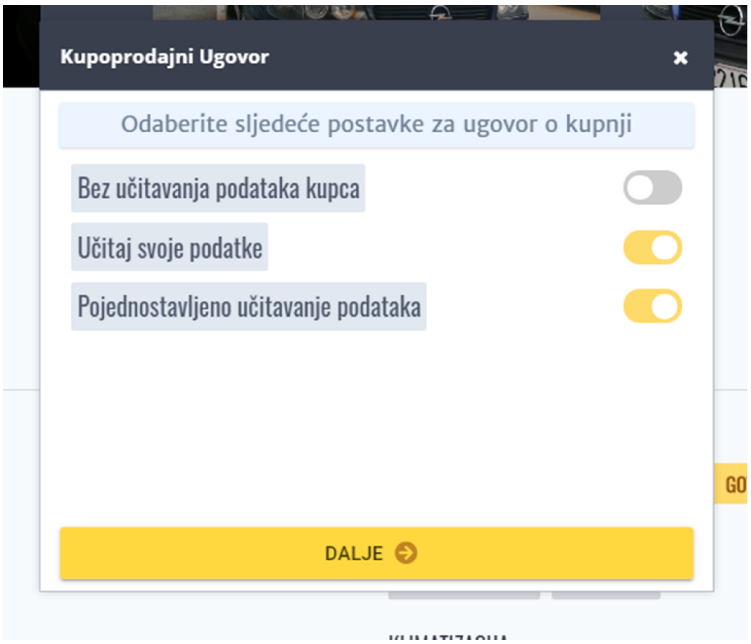


Slika 9: *Potvrdni kôd iz pretinca e-pošte gmail*

3.1.6. Kupoprodajni ugovor

Konačni uvjet za uspješnu prodaju automobila je popunjavanje kupoprodajnog ugovora između prodavača i kupca. Ova aplikacija omogućuje korisnicima generiranje popunjenog kupoprodajnog ugovora do mjere koja bi zadovoljila obje stranke.

Prilikom korištenja ove funkcionalnosti, najprije je potrebno označiti način popunjavanja ugovora. (*slika 10*).



Slika 10: *Prikaz odabira mogućih opcija za generiranje ugovora*

Aplikacija pruža tri moguće opcije prodavaču za potpunu kontrolu nad podacima koji se unose u ugovor. Može odabrati opciju učitavanja vlastitih i podataka o kupcu. Posljednja stavka označava mogućnost popunjavanja, koristeći sve podatke koje su dostavljene aplikaciji ili samo određenih koje prodavač može proizvoljno odabrati.

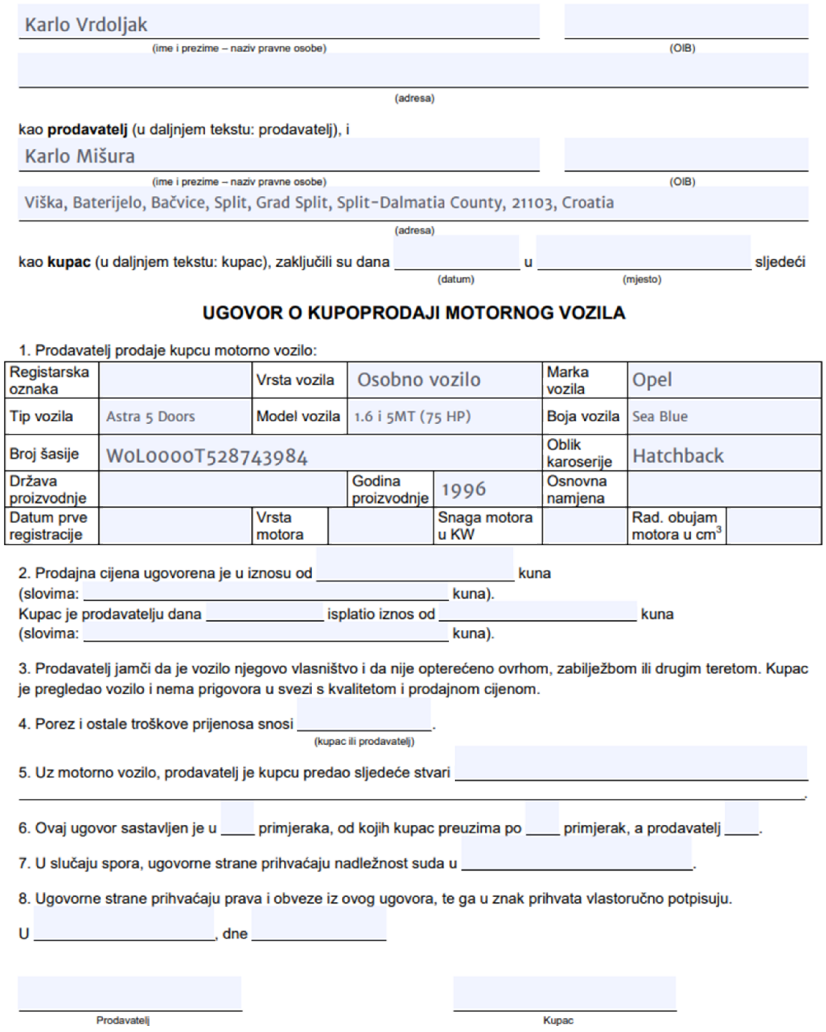
Ukoliko prodavač želi generirati prazni ugovor, potrebno je zabraniti učitavanje osobnih podataka i odabrati stavku naprednog unosa podataka te zanemariti odabir polja za unos. Na taj način će prodavač dobiti prazan kupoprodajni ugovor koji isključivo vrijedi u granicama Republike Hrvatske.

Poslužitelj za stvaranje ugovora koristi biblioteku *pdf-lib* u kojoj se učita prazni ugovor i ovisno o odabranim opcijama se dinamički popuni isti ugovor. Tako stvoren ugovor se prosljeđuje klijentu (*ispis 12*).

```
@Post('kupoprodajni')
async generatePdf(@Request() req, @Res() res: Response) {
  const doc = await this.oglasService.generateKupoprodajni(req.body);
  let stream = new Duplex();
  stream.push(doc);
  stream.push(null);
  res.setHeader('Content-Type', 'application/pdf');
  res.status(HttpStatus.OK);
  stream.pipe(res);
}
```

Ispis 12: *Prikaz generiranja kupoprodajnog ugovora*

Pozivom na metodu *generateKupoprodajni* s parametrom *req.body*, usluga *oglasService* popuni kupoprodajni ugovor sa svim potrebnim podacima koristeći biblioteku *pdf-lib*. Varijabla *stream* koja je objekt klase *Duplex*, omogućava stvaranje niza heksadecimalnih vrijednosti koje imaju funkcionalnost pisanja i čitanja multimedije. U objektu za odgovor *res* se postavlja zaglavlje *Content-Type* kojemu se pridodaje vrijednost *application/pdf*. Navedeno zaglavlje označava da će odgovor biti pdf dokument, kojeg se prosljeđuje klijentu koristeći naredbu *stream.pipe* s parametrom *res*. Na sljedećoj slici je prikazan primjer kupoprodajnog ugovora (*slika 11*).



Karlo Vrdoljak
(ime i prezime – naziv pravne osobe) (OIB)
(adresa)

kao **prodavatelj** (u daljnjem tekstu: prodavatelj), i

Karlo Mišura
(ime i prezime – naziv pravne osobe) (OIB)
Viška, Baterijelo, Bačvice, Split, Grad Split, Split-Dalmatia County, 21103, Croatia
(adresa)

kao **kupac** (u daljnjem tekstu: kupac), zaključili su dana _____ u _____ sljedeći
(datum) (mjesto)

UGOVOR O KUPOPRODAJI MOTORNOG VOZILA

1. Prodavatelj prodaje kupcu motorno vozilo:

Registarska oznaka		Vrsta vozila	Osobno vozilo	Marka vozila	Opel
Tip vozila	Astra 5 Doors	Model vozila	1.6 i 5MT (75 HP)	Boja vozila	Sea Blue
Broj šasije	WOL0000T528743984			Oblik karoserije	Hatchback
Država proizvodnje		Godina proizvodnje	1996	Osnovna namjena	
Datum prve registracije		Vrsta motora		Snaga motora u KW	Rad. obujam motora u cm^3

2. Prodajna cijena ugovorena je u iznosu od _____ kuna
(slovima: _____ kuna).
Kupac je prodavatelju dana _____ isplatio iznos od _____ kuna
(slovima: _____ kuna).

3. Prodavatelj jamči da je vozilo njegovo vlasništvo i da nije opterećeno ovrhom, zabilježbom ili drugim teretom. Kupac je pregledao vozilo i nema prigovora u svezi s kvalitetom i prodajnom cijenom.

4. Porez i ostale troškove prijenosa snosi _____.
(kupac ili prodavatelj)

5. Uz motorno vozilo, prodavatelj je kupcu predao sljedeće stvari _____
_____.

6. Ovaj ugovor sastavljen je u ____ primjeraka, od kojih kupac preuzima po ____ primjerak, a prodavatelj ____.

7. U slučaju spora, ugovorne strane prihvaćaju nadležnost suda u _____.

8. Ugovorne strane prihvaćaju prava i obveze iz ovog ugovora, te ga u znak prihvata vlastoručno potpisuju.

U _____, dne _____

_____ _____
Prodavatelj Kupac

Slika 11: *Primjer kupoprodajnog ugovora*

3.2. Klijent

U ovom poglavlju se nalazi razrada što sve korisnik može raditi u aplikaciji i opis glavnih funkcionalnosti koje se odvijaju u razvojnom okviru *Angular*. Prikazat će se kako aplikacija rukuje s promjenom stanja, iz prijavljenog korisnika u anoniman način rada i koji se sigurnosni koraci primjenjuju prilikom svakog otvaranja ekrana aplikacije.

3.2.1. Dizajn

Kao što je već navedeno, dizajn klijentske strane aplikacije je jednako važna stavka kao i skup funkcionalnosti koje sustav nudi. Aplikacija mora biti ugodna za korištenje, dizajn mora biti jednoznačan i način korištenja iste mora biti jednostavan te pristupačan.

Kako bi dizajn bio jednoznačan i ugodan, prvenstveno su korištene biblioteke *tailwindcss*, *PrimeNG* i *Angular material.*

Biblioteka *tailwindcss* je opisana u poglavlju 2.4, stoga je nepotrebno dodatno opisivanje.

PrimeNG je skup gotovih komponenti korisničkog sučelja, koje se većinski koriste u aplikaciji za korisnički unos. Biblioteka pruža dizajnerski alat kako bi korisnici mogli stvoriti vlastitu temu.

Angular material je biblioteka koju pruža *Google*, za lakšu izradu modernih Angular aplikacija. U ovoj aplikaciji su najviše korištene komponente koje pružaju *ripple effect* (hrv. *mreškanje*), *stepper module* (hrv. *modul korak po korak*) i moderne tipke.

Obzirom na to da su korištene različite biblioteke za stvaranje dizajna aplikacije, potreban je sklad boja, stoga je u sljedećem primjeru prikazana paleta boja (*ispis 12*).

```
$yellow: #feda6a !default;
$lightYellow: #ffcd39 !default;
$whiteYellow: #fffaf0!default;
$darkYellow: #AD974F !default;
$brown: #795e07!default;
$silver: #d4d4dc !default;
$lightGray: #80889b !default;
$deepGray: #393f4d !default;
$dark: #333333 !default;
$none: #00000000!default;
$darkGray: #231F20 !default;
```

Ispis 12: *Prikaz SCSS (engl. Sassy Cascading Style Sheets) varijabli korištenih za stvaranje palete boja*

3.2.2. Način korištenja

Korištenje ove aplikacije od prvog ekrana je jasno, prikazano je polje *pretraži* gdje korisnici mogu odmah pretražiti oglase bez gubljena vremena (*slika 12*).

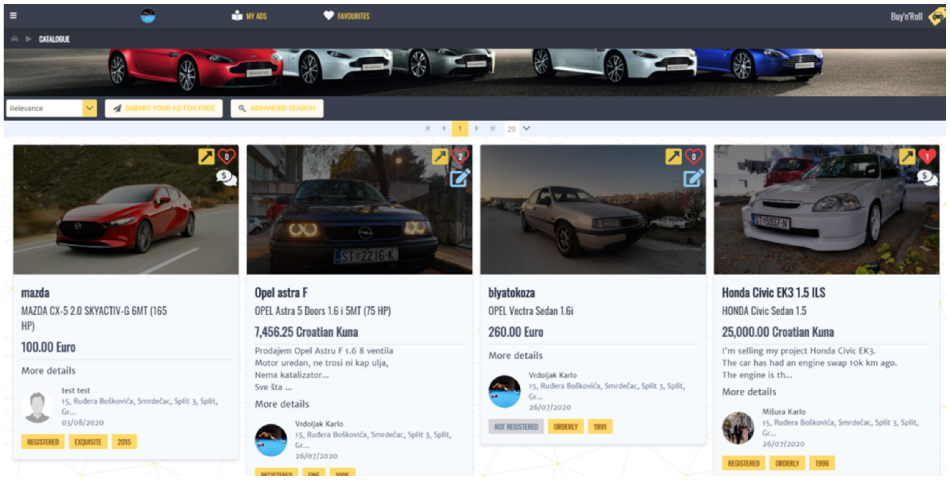
Slika 12: *Prvi ekran aplikacije*

U gornjem dijelu ekrana su općenite opcije aplikacije redom:

- Glavni meni
- Moj profil – uz prijavu
- Moji oglasi – uz prijavu
- Moji omiljeni oglasi – uz prijavu

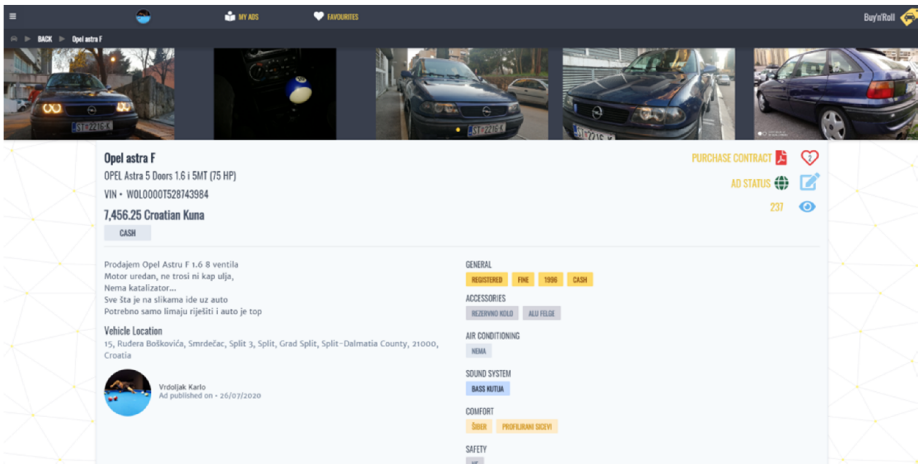
Kroz glavni meni je moguće obaviti sve važne fukcionalnosti aplikacije, a to su sve tri vrste pretraga, mijenjanje jezika, prijava ili odjava i stvaranje novog oglasa ukoliko je korisnik prijavljen.

Opcije *moj profil*, *moji oglasi* i *moji omiljeni oglasi* vode na istoimene ekrane. Sljedeća važna funkcionalnost su pretrage koje izgledaju kao na slici ispod (*slika 13*).



Slika 13: *Prikaz ekrana pretraga*

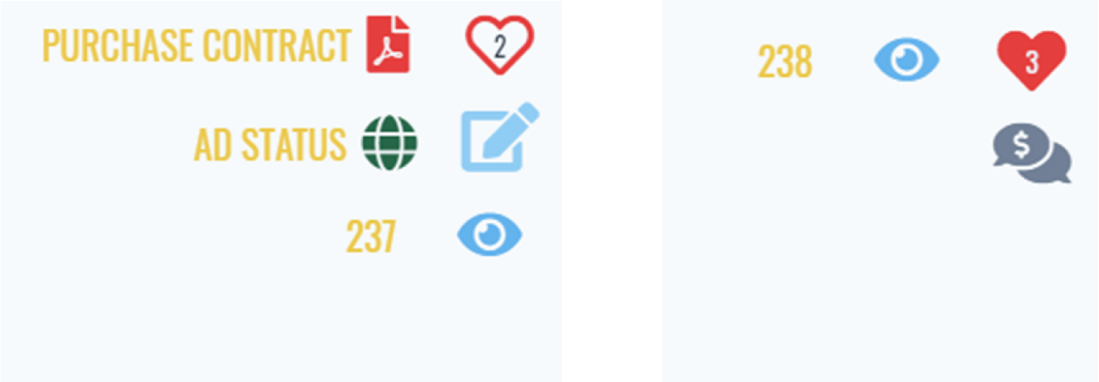
Nadalje, ulaskom u pojedini oglas nalazi se ekran kao na sljedećoj slici (*slika 14*).



Slika 14: *Prikaz ekrana pojedinog oglasa*

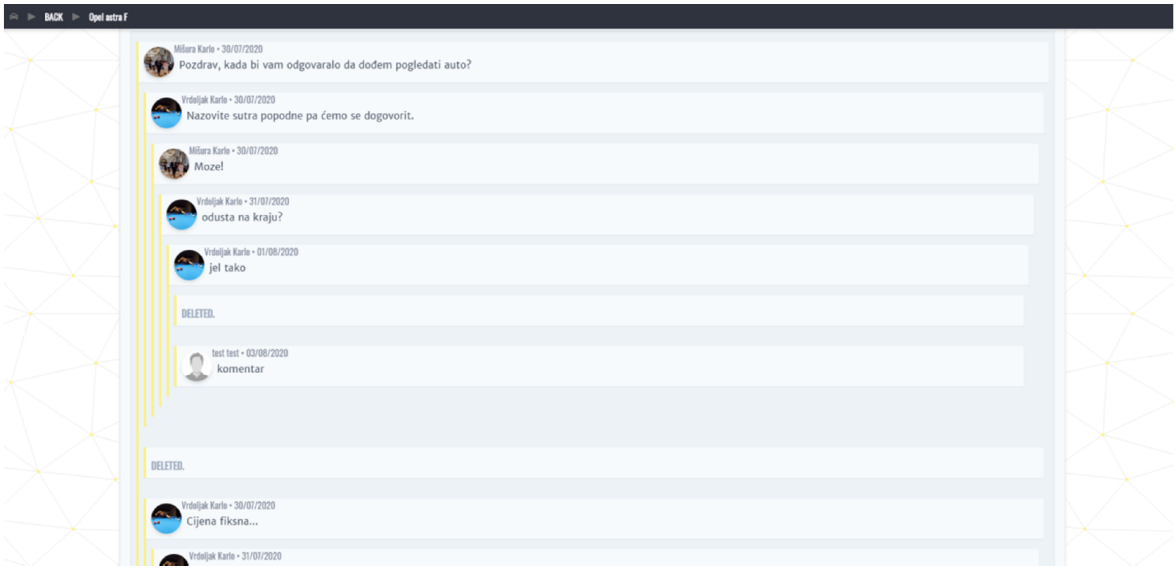
Svaki oglas ima prikaz svih unesenih podataka prilikom stvaranja, broj pregleda i broj koji označava koliko je puta dodan u omiljene. Pregled oglasa se mijenja u ovisnosti o tome je li korisnik prijavljen i je li taj prijavljeni korisnik vlasnik oglasa. Mogućnosti su prikazane na sljedećoj slici (*slika 15*). Na ovom ekranu izmjene se zapravo pronalaze kod

akcijskih tipki u gornjem desnom uglu ispod slika automobila.



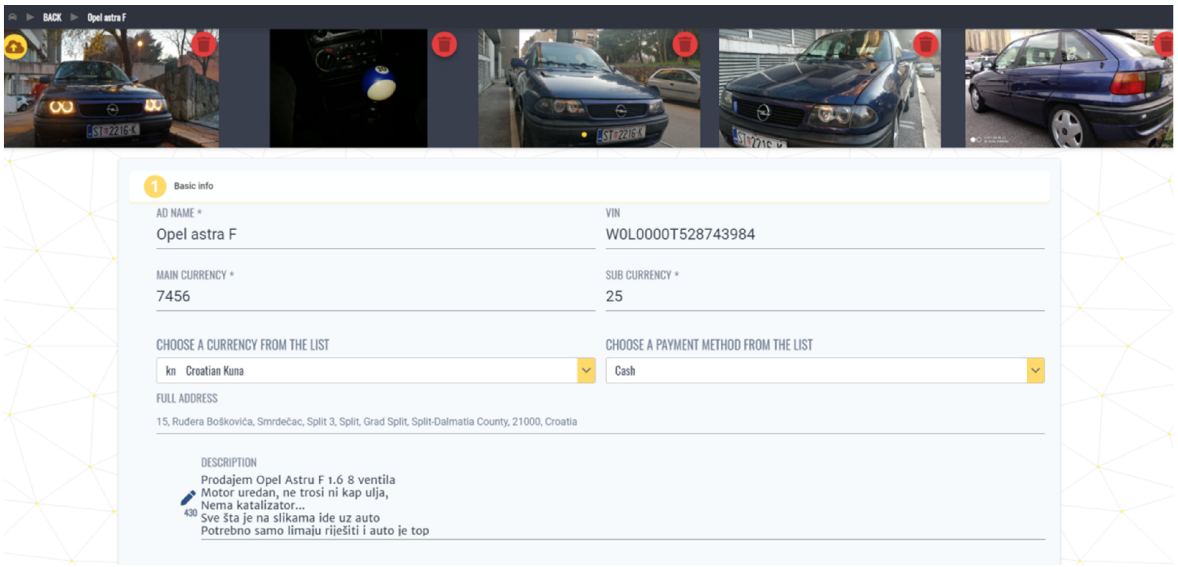
Slika 15: *Promjena mogućnosi ovisno o statutsu prijave gdje na slici lijevo je korisnik prijavljen i vlasnik, a na slici desno nije vlasnik*

Svaki oglas ima svoje komentare koji su na dnu oglasa, struktura je stablasta i na svaki komentar se može odgovoriti kao što je na slici (*slika 16*). Korištenje i značaj tipke za generiranje kupoprodajnog ugovora je već detaljno opisan u poglavlju *Kupoprodajni ugovor* kao i značenje statusa oglasa u poglavlju *Uvod*.



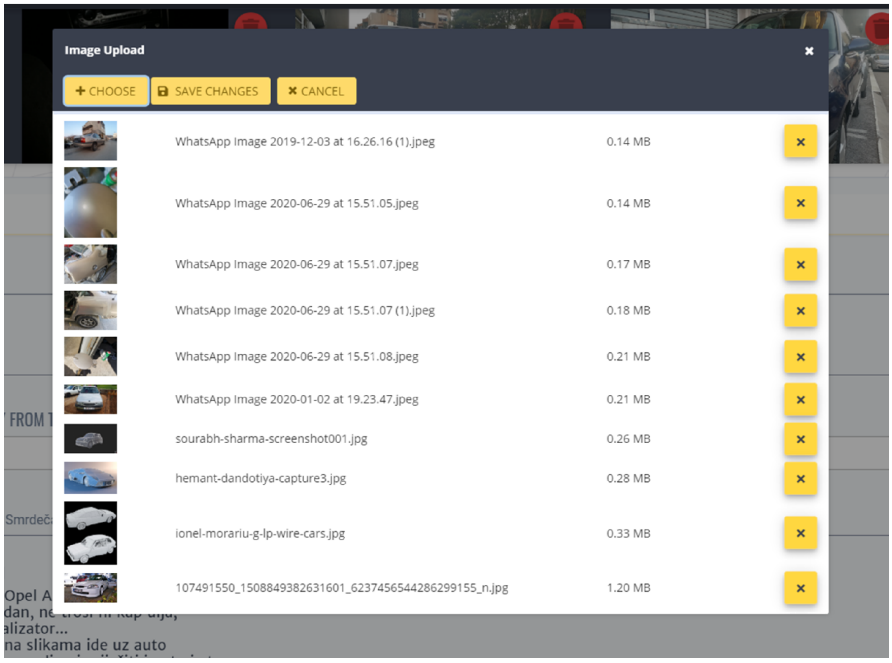
Slika 16: *Prikaz strukture komentara*

Ekrani stvaranja i izmjene oglasa su gotovo identični, stoga je na sljedećoj slici prikazan samo ekran izmjene oglasa. Za kreirati oglas jedini je uvjet da korisnik nije anoniman, odnosno da je uspješno registriran i aktivan član aplikacije (*slika* 17).



Slika 17: *Ekran izmjene oglasa*

Unos slika se obavlja koristeći žutu tipku u gornjem lijevom uglu i dobije se sljedeći prikaz (*slika 18*).



Slika 18: *Prikaz funkcionalnosti prenošenja fotografija*

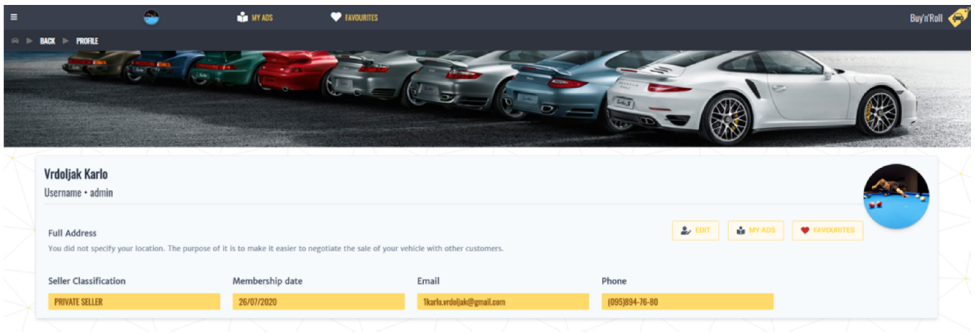
Kao što se vidi na slici, moguć je unos višestrukih fotografija. Prikaz ekrana *moji oglasi* i *moji omiljeni* oglasi su dizajnom slični kao i ekran pretrage.

U ekranu moji oglasi, korisnik dobije prikaz svih svojih oglasa nad kojima može sve raditi bez restrikcija.

Moji omiljeni su prikaz svih omiljenih korisnika, gdje je moguće brisati pojedini oglas iz svog popisa.

Profil korisnika je ispis svih osobnih podataka tog korisnika, uz tipke koje vode na ekrane njegovih oglasa ili omiljenih. Ukoliko je korisnik prijavljen u aplikaciju i taj profil

je njegov, onda je omogućena izmjena podataka profila (*slika 19*).



Slika 19: *Prikaz profila korisnika, na slici je vlasnikov profil*

3.2.3. Sigurnost

Najvažnija stavka svake aplikacije je sigurnost za korisnike. Sigurnost se postiže na način da klijent i poslužitelj zajedno brinu o toj stavci. Dakle ne može se postaviti restrikcija korisnicima na neki ekran, tako što ne postoji tipka ili poveznica koja ih vodi tamo. Treba se kvalitetno osigurati svaka pojedina putanja, kako bi joj mogli pristupiti samo oni korisnici koji na to imaju pravo.

Kao što je već ranije rečeno, aplikacija koristi autorizaciju putem JWT tokena. Kada se korisnik uspješno prijavi u aplikaciju, poslužitelj odgovara s potpisanim tokenom. Dekodirajući token mogu se dobiti podaci korisnika koji je prijavljen, koje prihvaća klijentska aplikacija i sprema u lokalnu pohranu (engl. *local storage*).

Aplikacija će svakom promjenom konteksta putanje provjeriti je li korisniku token i dalje vrijedi. Ukoliko to nije slučaj, obavlja se proces odjave iz aplikacije za tog korisnika i aplikacija nastavlja s anonimnim radom. Provjera se izvršava tako da klijent postavi upit prema poslužitelju, gdje pošalje token i očekuje pozitivan odgovor.

Provjera pri svakoj promjeni konteksta je omogućena zahvaljujući tehnici autentikacijskog čuvara (engl. *auth guard*) koja izvršava metodu *canActivate()*, koja obavezno mora vratiti rezultat *true* ili *false*. Ukoliko je rješenje *false* korisnik nema prava otvaranja te putanje, stoga neće dopustiti promjenu konteksta (*ispis 13*).

```
canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
      let auth = this.storage.get('auth');
      if(auth) {
        return this.userService.checkToken().then(result => {
          if(result == true) {
            return true
          } else {
            this.router.navigate(['/login'],{ skipLocationChange: true});
            return false;
          }
        });
      }
```

```
    this.router.navigate(['/login'], { skipLocationChange: true });
    return false;
  }
```

Ispis 13: *Prikaz provjere valjanosti tokena*

Kako bi korisnici imali dojam brzog rada aplikacije, svi potrebni podaci za prikaz željenog ekrana se dohvate prije samog iscrtavanja istog. Pritom, ukoliko se dogodi HTTP greška, ekran se neće otvoriti i korisnik će ostati gdje je dosad bio. Ova funkcionalnost je moguća zahvaljući tehnici rješavača gledišta (engl. *view resolver*) koja izvrši metodu *resolve()* iz klase rješavača. Rezultati poziva prema poslužitelju se pohrane u niz i proslijede na pripadajući ekran (*ispis 14*).

```
resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot){
  if(!route.params.query) {
    const state: RouterState = this.router.routerState;
    this.errorHandler.handleRouterState(state,true);
    return;
  }
  let prevRoute = this.helperService.getLastNavigation();
  return forkJoin (
    this.userService.findUsersFavouritedOglas(route.params.query),
    of(prevRoute),
    this.translate.get(this.translationProvider.getRegistration()),
  ).pipe(
    catchError(error => {
      const state: RouterState = this.router.routerState;
      this.errorHandler.handleRouterState(state);
      return this.errorHandler.handleError;
  }));
}
```

Ispis 14: *Za ekran popisa omiljenih oglasa resolve() metoda*

Sigurnost na strani poslužitelja je jednostavnija nego na strani klijenta, zbog toga što što se predefinira koje putanje kontrolera moraju biti pod nekom vrstom zaštite. Implementacija JWT token sigurnosti je navedena u poglavlju *NestJS*.

Propagiranje zaštite kroz poslužiteljsku aplikaciju svodi se na to da je potrebno navesti ključnu riječ *@UseGuards* i/ili *@Roles* kako bi zaštitilu pojedinu putanju kontrolera (*ispis 15*).

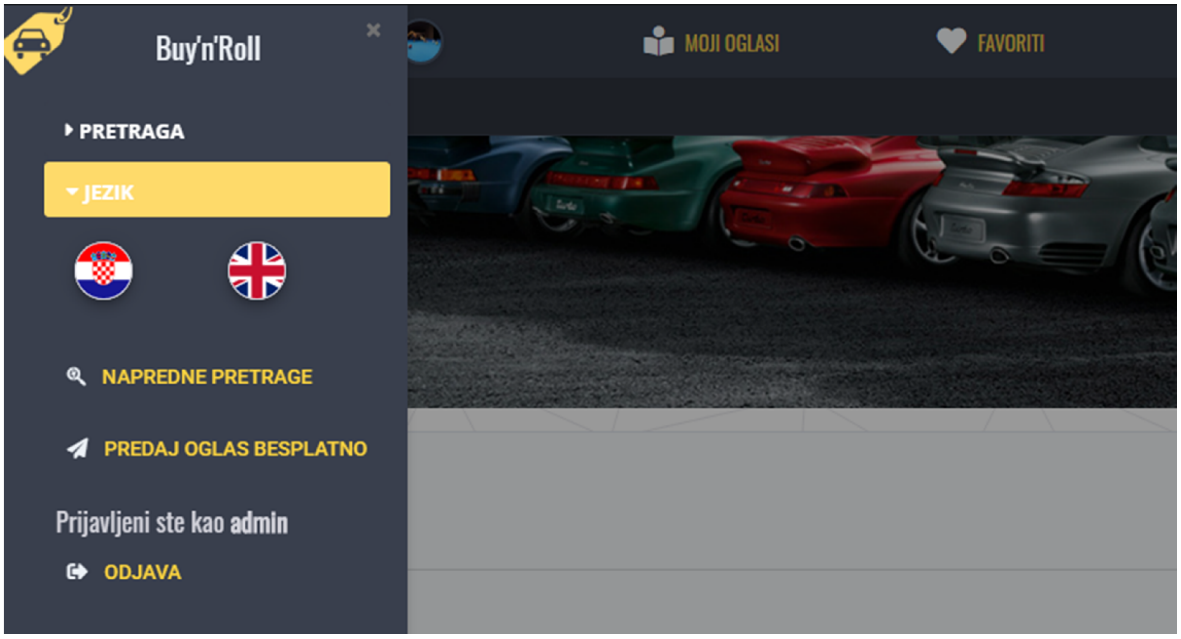
```
@UseGuards(JwtAuthGuard,RolesGuard)
@Roles('user')
@Post('/add/favourite')
async toggleToFav(@Request() req, @Res() res: Response) { ... }
```

Ispis 15: *Prikaz osiguravanja putanje „/add/favourite“ kako bi joj mogli pristupiti samo prijavljeni korisnici kojima je uloga barem „user“*

3.2.4. Višejezičnost

Kroz poglavlja ovog rada pokrivene su sve važne stavke koje prilikom korištenja

aplikacije mogu doprinjeti velikom broju korisnika. Obzirom da je riječ o internet aplikaciji, očekivano je da će biti posjetitelja iz raznih država, stoga je uvedena višejezičnost gdje korisnici u glavnom izborniku, pod stavkom jezik, mogu birati između hrvatskog i engleskog jezika(*slika 20*).



Slika 20: *Prikaz odabira jezika*

Internacionalizacija se postiže koristeći modul *ngx-translate* koji pruža mogućnost učitavanja više *JSON* (engl. *JavaScript Object Notation*) datoteka koje sadrže niz ključ-vrijednost parova, gdje su ključevi jednoznačni za svaki prijevod, a vrijednosti su stvaran prijevod (*ispis 16*).

```
{
  ...
  "LANDING_SECTION_ONE_HEADER": "Pick your ride",
  "LANDING_SECTION_ONE_HEADER_A": "We'll set you up!",
  "MANUFACTURER":"Manufacturer"
  ...
}
{
  ...
  "LANDING_SECTION_ONE_HEADER": "Odaberite vozilo",
  "LANDING_SECTION_ONE_HEADER_A": "Pronaći ćemo ga!",
  "MANUFACTURER": "Proizvođač"
  ...
}
```

Ispis 16: *Prikaz JSON datoteka s prijevodima, engleski gore i hrvatski dolje*

Prevođenje se odvija na klijentskog strani aplikacije.

Prednosti su sljedeće:

- prijevod se odvija jako brzo
- nema potrebe za ponovnim učitavanjem pogleda
- mogućnost korištenja stotine jezika

Koristi se na sljedeći način (*ispis 17*).

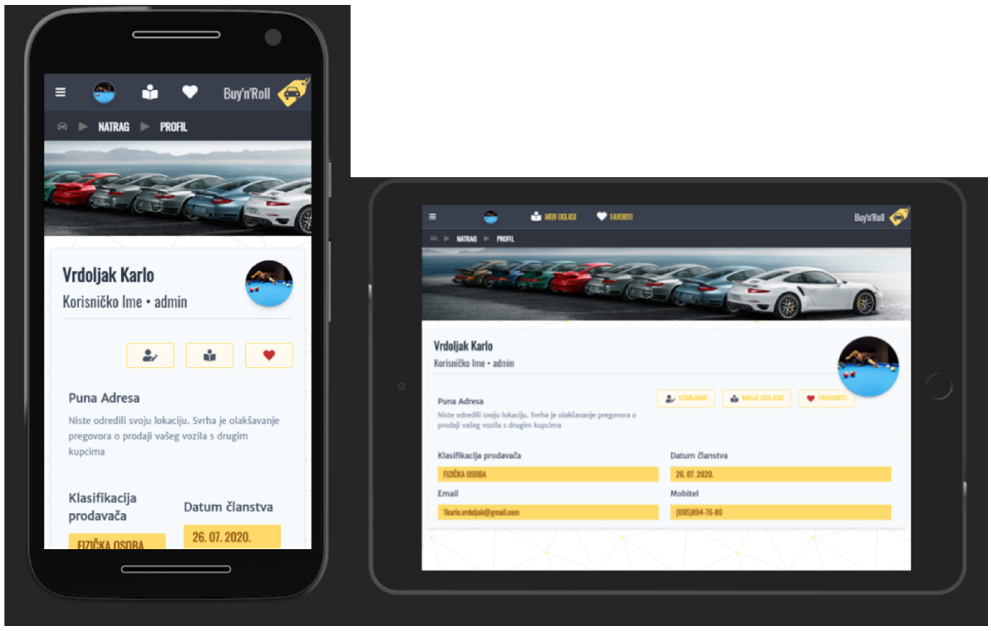
```
<div class="item header-text">
      {{ 'LANDING_SECTION_ONE_HEADER' | translate }}
</div>
<div class="item header-text smaller">
      {{ 'LANDING_SECTION_ONE_HEADER_A' | translate }}
</div>
```

Ispis 17: *Prikaz korištenja internacionalizacije u HTML datoteci*

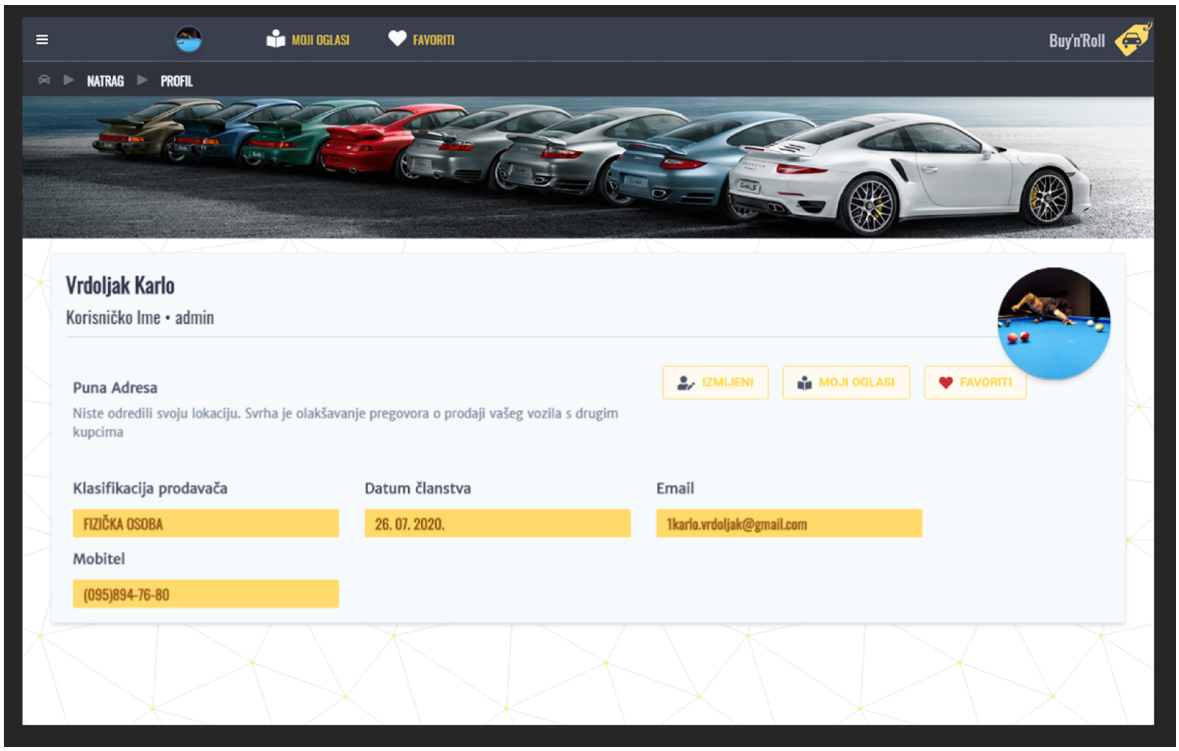
Kao što se vidi iz primjera koristi se tako da se *interpolira* ključ iz *JSON* datoteke uz ključne riječi | *translate*.

3.2.5. Responzivnost

S obzirom na razvoj interneta, bilo kakve usluge koje se pružaju na spomenutoj platformi trebaju biti dostupne nadohvat ruke te lake za korištenje. U svrhu toga potrebne su responzivne, odnosno prilagodljive aplikacije, koje se mogu koristiti ne samo putem računala već i korištenjem mobilnih uređaja na bilo kojem mjestu, u bilo koje vrijeme. Ova aplikacija je napravljena u skladu sa navedenim potrebama (*slike 21 i 22*).



Slika 21: *Prikaz aplikacije koristeći mobitel (lijevo) i tablet (desno)*



Slika 22: *Prikaz aplikacije koristeći računalo*

Responzivnost se najjednostavnije može postići korištenjem biblioteka korisničkog sučelja, koje uglavnom sadrže *CSS* klase koje pomažu prilikom stvaranja responzivnog dizajna (*ispis 18*).

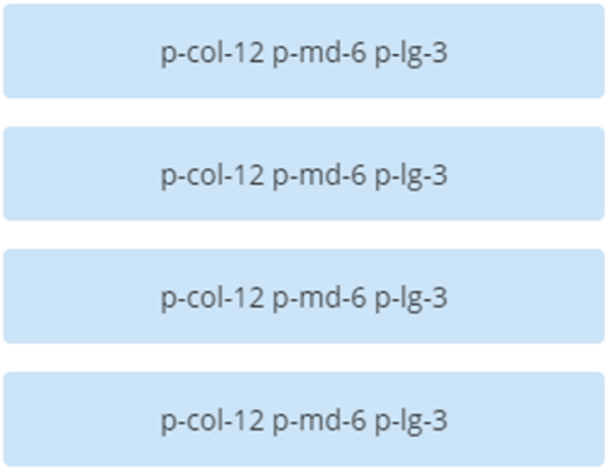
```
<div class="p-grid">
    <div class="p-col-12 p-md-6 p-lg-3">
        <div class="box">p-col-12 p-md-6 p-lg-3</div>
    </div>
    <div class="p-col-12 p-md-6 p-lg-3">
        <div class="box">p-col-12 p-md-6 p-lg-3</div>
    </div>
    <div class="p-col-12 p-md-6 p-lg-3">
        <div class="box">p-col-12 p-md-6 p-lg-3</div>
    </div>
    <div class="p-col-12 p-md-6 p-lg-3">
        <div class="box">p-col-12 p-md-6 p-lg-3</div>
    </div>
</div>
```

Ispis 18: *Prikaz responzivnog dizajna*

Klasa *p-grid* označava da će sljedeći sadržaj biti obuhvaćen u koordinatnoj mreži gdje popratne klase *p-col-12* označavaju da je mreža ekran u 12 jednakih podjela. Brojevi uz *p-md, p-lg* označavaju djeljenika broja 12 gdje je rezultat broj koji množi jednu dvanaestinu trenutne širine ekrana (*slika 23*).

1.

2. 

3. 

Slika 23: *Prikaz promjene raspodjele sadržaja ovisno o širini ekrana*

Navedeni sadržaj na velikim ekranima se može smjestiti u jedan red zbog toga što su četiri elementa širine četvrtine ekrana (*slika 23.1*). Kod ekrana srednje širine, kao što su tableti, raspodjela nalaže da će elementi biti širine polovine ekrana (*slika 23.2*), dok elementi na mobilnim uređajima će obuhvatiti potupunu širinu ekrana (*slika 23.3*).

4. Zaključak

Konačni rezultat ovog završnog rada je aplikacija za kupnju i prodaju osobnih automobila. Sustav omogućuje korisnicima pretraživanje i stvaranje oglasa te dodavanje pojedinog oglasa među svoje omiljene. Nadalje, ima opciju objave fotografija, kako bi oglasi bili što detaljniji. Kao proširenje sustavu oglasnika omogućena je usluga ispisa kupoprodajnog ugovora među korisnicima. Kao društvena komponenta aplikacije, dodano je postavljanje komentara na oglas i mogućnost odgovora na iste. Kako bi aplikacija mogla namjenjena za svakog korisnika interneta, responzivnost i ugodan dizajn te brzina sustava su bile početne točke razvoja usluge. Korisnicima je omogućeno sigurno korištenje aplikacije.

Prostor za napredak postoji u svakom sustavu, pa tako i u ovom. Bilo bi dobro dodati opciju razmjena tekstualnih poruka između kupca i prodavača kao dodatno olakšanje prilikom prodaje, odnosno kupnje automobila. Nadalje, aplikacija nudi mogućnost odabira proizvođača, serije i modela vozila. Ponuđeni izbori mogu zastarjeti s obzirom na proizvodnju novih vozila. Primjerice, u aplikaciji je najnoviji *Volkswagen Golf 7* iako je činjenica da postoji nasljednik *Golf 8*. Kao rješenje ovog slučaja potrebna je veća uloga od korisničke gdje će zadužena osoba ažurirati aplikaciju s novim vozilima.

Korištene tehnologije *Angular* i *NestJS* pokazale su se kao izvrsna kombinacija za izradu internetske aplikacije. Obzirom da imaju gotovo identičnu organizaciju arhitekture, programeri koji poznavaju *Angular* mogu vrlo lako koristiti *NestJS* okvir poslužitelja. Glavni problem koji odvlači neke programere od korištenja *Angulara,* analogno tome i *NestJS*, nalazi se u tome što je *krivulja učenja Angular* aplikacija osjetno gora u odnosu na druge razvojne okvire.

Literatura

[1] *ACEA_Report_Vehicles_in_use-Europe_2019.pdf*, https://www.acea.be/uploads/publications/ACEA_Report_Vehicles_in_use-Europe_2019.pdf#page=4 (*posjećeno 6.8.2020.*)

[2] *Car Ownership Statistics (2020 Report),* https://www.valuepenguin.com/auto-insurance/car-ownership-statistics (*posjećeno 6.8.2020.*)

[3] *Full-Text Search*, https://dev.mysql.com/doc/internals/en/full-text-search.html (*posjećeno 7.8.2020.*)

[4] *Documentation | NestJS*, https://docs.nestjs.com (*posjećeno 7.8.2020.*)

[5] *TypeORM*, https://typeorm.io (*posjećeno 7.8.2020.*)

[6] *Introduction to the Angular Docs*, https://angular.io/docs (*posjećeno 7.8.2020.*)

[7] *PrimeNG*, https://www.primefaces.org/primeng/showcase/#/setup (*posjećeno 7.8.2020.*)

[8] *Getting started with Angular Material*, https://material.angular.io/guide/getting-started (*posjećeno 8.8.2020.*)

[9] *Tailwindcss*, https://tailwindcss.com (*posjećeno 8.8.2020.*)

[10] *Angular Gallery v5*, https://github.com/MurhafSousli/ngx-gallery/wiki (*posjećeno 8.8.2020.*)

[11] *Swiper*, https://swiperjs.com/get-started/ (*posjećeno 8.8.2020.*)

[12] *TypeScript Documentation*, https://www.typescriptlang.org/docs/ (*posjećeno 8.8.2020.*)