Aplikacija za rezerviranje termina i usluga frizerskih salona

Majić, Iva

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

APLIKACIJA ZA REZERVIRANJE TERMINA I USLUGA FRIZERSKIH SALONA

Diplomski rad

Iva Majić

Osijek, 2019.

SADRŽAJ

1. UVOD

Klasična, telefonska rezervacija termina i usluga u frizerskim salonima može biti kompliciran proces: potrebno je pronaći telefonski broj salona, nazvati taj broj, zatim se djelatnik treba javiti na telefon, a možda je trenutno zauzet klijentom. Nakon što se djelatnik javi, potrebno je ugovoriti sam termin, djelatnik mora provjeravati u bilježnici slobodne termine i tek onda je cijeli postupak gotov. Osim toga, u gomili frizerskih salona ponekad je teško znati čije usluge su dobre i odgovarajuće, a čije ne. Iz tog razloga pojavila se potreba i ideja za razvojem aplikacije za rezervaciju termina i usluga u frizerskim salonima, čijim će korištenjem rezervacija biti gotova u nekoliko klikova, bez dugotrajnog procesa telefonskog poziva.

Aplikacija za rezerviranje termina i usluga u frizerskim salonima korisnicima frizerskih usluga omogućuje rezervaciju termina u frizerskim salonima, a djelatnicima tih salona upravljanje rezervacijama. Na taj način aplikacija omogućava lakše, preglednije i brže rezervacije termina. Korisnik frizerskih usluga može vidjeti popis dostupnih frizerskih salona, provjeriti detalje i ocjenu pojedinog salona, ocijeniti salon te rezervirati termin u željenom salonu za željenu uslugu. Frizerski saloni imaju uvid u rezervacije, a iste mogu izmijeniti ukoliko se ukaže potreba za time. Tako se djelatnici salona ne moraju odvajati od klijenta da se jave na telefon, nego ih sve rezervacije već čekaju u aplikaciji te između termina mogu provjeriti i potvrditi rezervaciju klijenta za željeni termin i usluge.

Ovaj rad sastoji se od četiri glavna dijela. U prvom dijelu dat će se kratak pregled korištenih tehnologija. Zatim će se u drugom dijelu obraditi struktura i funkcionalnosti aplikacije, tj. programskog koda – poslužiteljski kod i korisničko sučelje. U trećem dijelu bit objašnjeno kako je testirana ispravnost aplikacije te će se naposljetku, u četvrtom dijelu, zaključiti rad.

1.1. Zadatak diplomskog rada

U sklopu diplomskog rada potrebno je kreirati web aplikaciju koja omogućuje vođenje frizerskih salona. Aplikacija treba ispuniti sljedeće zahtjeve:

1. aplikacija treba omogućiti vođenje frizerskih salona na različitim lokacijama
2. aplikacija treba imati glavnog administratora koji može nadzirati sve lokacije, a svaka lokacija treba imati svog administratora

3. korisiniku se pri rezerviranju usluga trebaju ponuditi najbliže lokacije, a on može odabrati među njima ili ostalim lokacijama

4. korisnik može vidjeti slobodne termine i usluge

5. korisnik može vidjeti arhivu korištenih usluga

6. administrator može vidjeti sve korisnike, usluge i termine.

2. KORIŠTENE TEHNOLOGIJE

Za izradu aplikacije za rezerviranje termina i usluga u frizerskim salonima primijenjene su neke od popularnih tehnologija današnjice za izradu internetskih aplikacija. Za korisničko sučelje (engl. *front-end*) odabran je Angular *framework* temeljen na jeziku TypeScript, a za poslužiteljski kod (engl. *back-end*) korišten je Laravel *framework* temeljen na PHP-u.

2.1. HTML5

HTML (engl. *HyperText Markup Language*) opisni je jezik za stvaranje dokumenata koji se prikazuju u internetskom pregledniku, tj. web dokumenata. Internetski preglednici primaju HTML dokumente s poslužitelja ili lokalno te ih prikazuju kao multimedijske stranice [1].

Osnovna jedinica HTML-a je element. Postoje dvije vrste HTML elemenata. Prva vrsta je element koji se sastoji od otvarajuće i njemu odgovarajuće zatvarajuće oznake (engl. *tag*), između kojih je tekst ili drugi HTML element. Primjer takvih elemenata su *div, h1, p, body* i slično, primjerice * Tekst</div.* Druga vrsta HTML elemenata je element koji se sastoji od samo jedne, samozatvarajuće oznake. Takav element ne sadrži drugi element ili tekst unutar sebe, a primjeri su *img, input, br* itd. Primjerice: *
*. Nadalje, većina HTML elemenata unutar otvarajuće (ili samozatvarajuće) oznake može sadržavati atribute koji ih onda dodatno opisuju. Tako primjerice element <input /> može sadržavati atribut *type* koji može poprimiti vrijednost *text, password, number..., placeholder* koji može poprimiti bilo koju tekstualnu vrijednost itd.

Dakle, svi elementi složeni zajedno u HTML dokumentu omogućuju prikaz web stranice. Bitno je naglasiti da HTML nije programski, nego opisni jezik. Samim HTML-om nije moguće izvršavati nikakve zadatke (npr. računske operacije) – HTML datoteke su zapravo obične tekstualne datoteke koje HTML oznakama opisuju kako stranica treba izgledati, a internetski preglednik zatim obrađuje te datoteke i pretvara ih u web stranice.

HTML5 je najnovija veća inačica HTML-a, prvi put objavljena početkom 2008. godine [2], nakon HTML 4 verzije izdane još 1999. Ova verzija unijela je u HTML znatne nove promjene kako bi se prilagodila potrebama današnjih internetskih stranica i aplikacija. Neke od promjena uvedenih u HTML 5 uključuju:

- nove sekcijske elemente kao što su *article, section, header, footer*

- multimedijske elemente, kao što su *video* i *audio,* što je omogućilo znatno smanjenje korištenja dodatnih skriptnih, *plugin* programa (npr. *Flash Player*)
- razvnovrsnije forme (nove verzije elemenata za unos, više različitih opcija)
- standarde za izradu web stranica što omogućuje web stranice koje rade jednako neovisno o pregledniku i platformi.

2.2. SCSS

CSS (engl. *Cascading Style Sheets*) stilski je jezik koji se koristi za dodatno opisivanje HTML dokumenta. Točnije, CSS opisuje kako će se elementi prikazivati u pregledniku: od pozicije elementa, preko veličine elementa do boja i ostalih elemenata dizajna.

CSS pretprocesorski programi skriptni su jezici koji proširuju CSS tako što omogućuju pisanje koda u jednom jeziku, a zatim njegovo kompajliranje u CSS-u [3]. Jedan od takvih jezika je i SASS (engl. *Syntactically Awesome Style Sheets*). SASS tako omogućuje korištenje varijabli, ugnježđivanje (engl. *nesting)* CSS pravila, uvoz varijabli iz drugih datoteka i sl., čime omogućuje jednostavniji i pregledniji kod. SASS koristi dvije različite sintakse: SCSS i uvučenu sintaksu (engl. *indented syntax*) [4] Za izradu ovog rada korištena je SCSS sintaksa. SCSS sintaksa je, uz par iznimki, prošireni CSS, što znači da je svaki valjani CSS kod zapravo i valjani SCSS kod [4]. Uz već navedene dodatne prednosti SASS-a, to je čini idealnom za jednostavno korištenje za potrebe zadatka ovog rada.

2.3. MYSQL

Baza podataka strukturirani je set podataka spremljen u računalu. Relacijska baza podataka je baza podataka temeljena na relacijskom modelu podataka, gdje su podaci prikazani kao n-torke grupirane u relacije. Pojednostavljeno, relacije su tablice s određenim brojem imenovanih stupaca i *n* redova podataka.

MYSQL je sustav upravljanja relacijskim bazama podataka (engl. *Relational Database Management System*, *RDBMS*) otvorenog koda. Za ovaj projekt odabran je upravo MYSQL zbog svoje jednostavnosti, dostupnosti i široke upotrebljivosti, što znači bogatu dokumentaciju.

Kako bi se kreirala baza podataka u koju će se spremati podaci iz aplikacije, koristi se besplatni program XAMPP koji omogućuje pokretanje lokalne baze podataka na računalu.

2.4. Angular

Angular je *framework* otvorenog koda koji se koristi za izradu internetskih aplikacija, većinski razvijen od strane tvrtke Google. Zasniva se na programskom jeziku TypeScript.

TypeScript je programski jezik otvorenog koda razvijen od strane tvrtke Microsoft. To je nadskup programskog jezika JavaScript (koji se pretežno koristi pri izradi web stranica). TypeScript omogućuje korištenje tipova varijabli i funkcija, a pri kompajliranju se prevodi u JavaScript pa se može koristiti za ono što sam HTML ne može, a to je izvršavanje zadataka (računske operacije itd.), odnosno programski dio web stranica. TypeScript se može koristiti i za poslužiteljski i za klijentski kod.

Za zadatak ovog rada koristi se Angular 7, što je inačica izdana u listopadu 2018. godine. Angular se koristi za izradu klijentskog dijela koda, odnosno za izradu korisničkog sučelja aplikacije. Ovaj *framework* odabran je prvenstveno zato što je korištenjem njegovih komponenti, servisa i modela lako napisati pregledan kôd jednostavan za održavanje, a korištenjem njegovog CLI (engl. *Command-line interface* – naredbeni redak) brzo je i jednostavno kreirati navedene komponente. Nadalje, arhitektura zasnovana na komponentama koju Angular koristi omogućuje ponovno korištenje iste komponente gdje god je to potrebno, što smanjuje ponavljanje kôda, a budući da su komponente neovisne jedna o drugoj, lako ih je testirati. Osim toga, Angular je široko korišten te stoga ima opširnu dokumentaciju i velik broj vanjskih biblioteka otvorenog kôda koje se mogu s lakoćom dodati u aplikaciju i olakšati razvoj iste.

Angular aplikacija sastoji se od različitih sastavnica, od kojih je bitno spomenuti dvije koje će se najčešće spominjati u ovom radu – komponente i servise. Komponente su Angular klase koje se koriste za prikaz web stranica HTML-om i CSS-om te za programske funkcionalnosti tih stranica (JavaScript, odnosno TypeScript). Servisi se pozivaju iz komponenti korištenjem tzv. *dependency injection* koje omogućuje korištenje javnh (engl. *public*) atributa i metoda iz tog servisa. Servisi se koriste za funkcionalnosti koje su potrebne širom aplikacije, kao što je dohvaćanje i zapisivanje podataka, odnosno komunikacija s poslužiteljem.

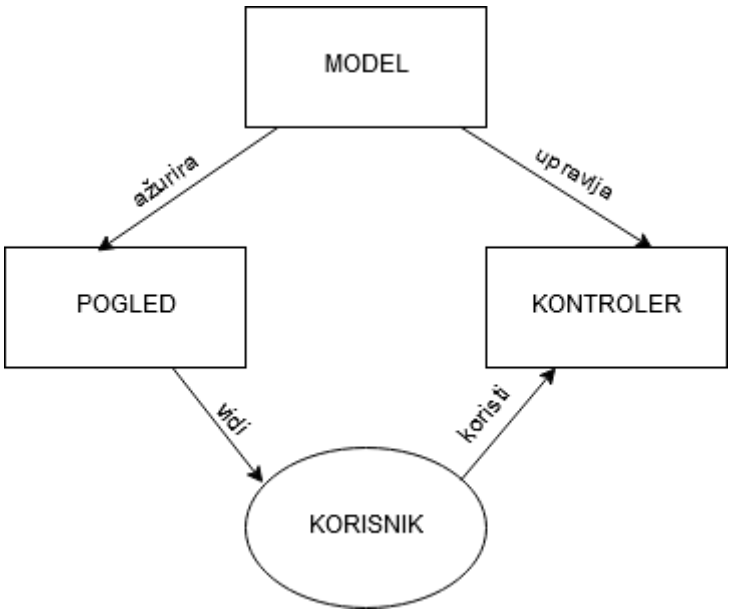
2.5. Laravel

Laravel je *framework* otvorenog koda temeljen na programskom jeziku PHP. Koristi MVC (engl. *Model-View-Controller*) arhitekturu za izradu web aplikacija, shematski prikazanu na slici 2.1. MVC arhitektura sastoji se od tri dijela:

- model – središnja komponenta koja oblikuje strukturu podataka, neovisno o korisničkom sučelju; upravlja podacima, logikom i upitima u bazu podataka

- pogled (engl. *view*) – komponenta koja prikazuje podatke proslijeđene od kontrolera – korisničko sučelje

- kontroler (engl. *controller*) – komponenta koja prima upite korisnika i pretvara ga u naredbe za model ili pogled [5].



Slika 2.1. *Shematski prikaz MVC arhitekture.*

U ovom radu koriste se model i kontroler, dok se pogled ne koristi jer je za korisničko sučelje odgovoran Angular, a Laravel se koristi za izradu poslužiteljskog dijela – RESTful API-ja (engl. ***Re****presntational* ***S****tate* ***T****ransfer* ***A****pplication* ***P****rogramming* ***I****nterface)*. Unutar Laravel kôda kreiraju se tzv. krajnje točke (engl. *endpoints*) pomoću kojih će korisničko sučelje (Angular) komunicirati s poslužiteljskim dijelom. Ovaj princip detaljnije će biti objašnjen u poglavlju 3.3.

Laravel je odabran za izradu ovog diplomskog rada prvenstveno zato što omogućuje vrlo jednostavno kreiranje i izmjenu baze podataka korištenjem migracija (engl. *migration*). Svaka migracija definira strukturu jedne tablice baze podataka: ime tablice, imena stupaca, njihove tipove te dodatna ograničenja. Također, sâm rad s bazom podataka pojednostavljen je korištenjem i nizanjem ugrađenih metoda (kao što su *get(), save(), delete(), where()*) umjesto čistih upita (engl. *query*).

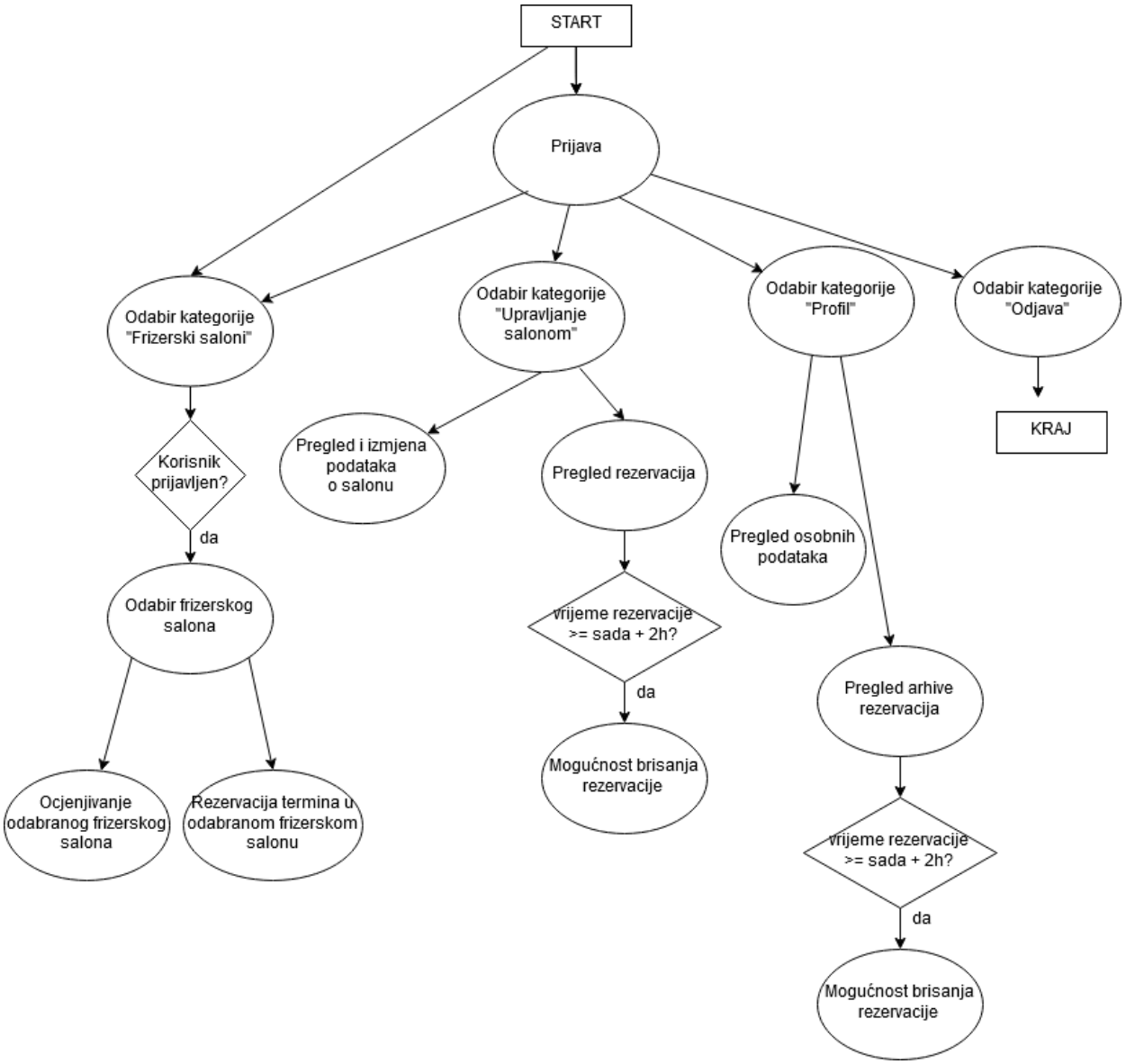
3. DIZAJN I REALIZACIJA APLIKACIJE

Aplikacija za rezerviranje termina i usluga u frizerskim salonima treba imati određene funkcionalnosti kako bi udovoljila zahtjevima postavljenima u poglavlju 1.1. Ukratko, aplikacija treba omogućiti prijavu korisnika, pregled frizerskih salona i rezervaciju termina. Svaki salon korisnik treba moći ocijeniti i ostaviti recenziju. Korisnik također treba moći dodati i vlastiti frizerski salon u aplikaciju čime postaje administrator salona i može vidjeti kalendar rezervacija, usluga i korisnika. Svaki korisnik treba moći vidjeti vlastite rezervacije, a korisnik i administrator salona mogu ukloniti rezervacije ukoliko je do termina ostalo više od dva sata. Također, u bazi podataka treba postojati korisnik koji je sustavski administrator (engl. *system admin*) koji može upravljati svim podacima – svim salonima i korisnicima.

Odlučeno je da će aplikacija imati odvojen poslužiteljski kôd i korisničko sučelje, tj. da će se koristiti različite tehnologije u dva odvojena softverska projekta, a zatim će se pomoću krajnjih točaka omogućiti komunikacija između ta dva dijela kako bi aplikacija funkcionirala kao jedna cjelina. U sljedećim poglavljima bit će objašnjeno planiranje aplikacije u vidu dijagrama toka i baze podataka te sama funkcionalnost u vidu poslužiteljskog kôda i korisničkog sučelja.

3.1. Dijagram toka

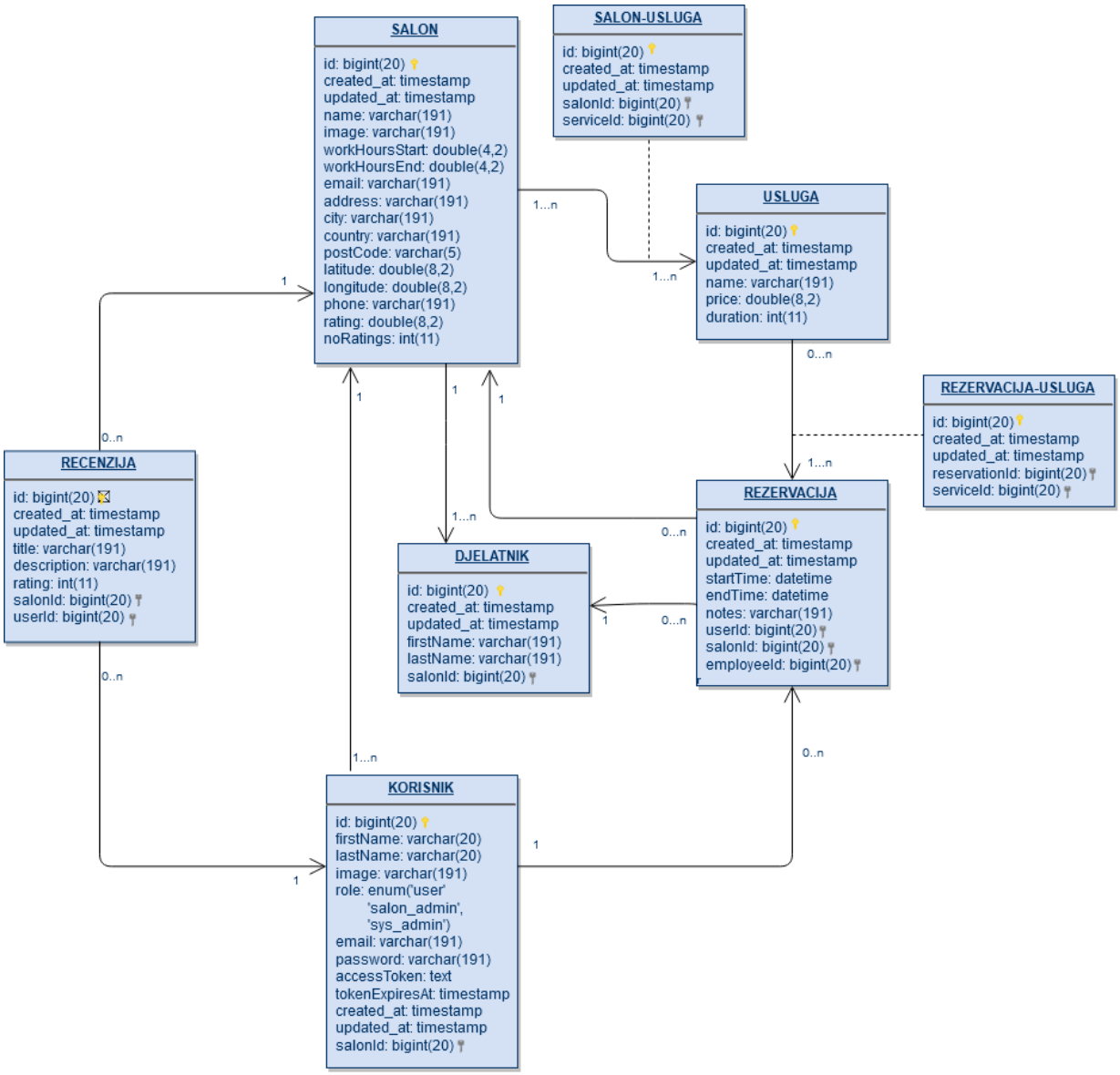
Prije samog programiranja aplikacije, potrebno je isplanirati kako će ona funkcionirati. Iz tog razloga kreiran je dijagram toka (engl. *flowchart*), prikazan na slici 3.1. Dijagram toka predstavlja pojednostavljene funkcionalnosti aplikacije, odnosno kako je planirano korištenje aplikacije. Iz dijagrama je vidljivo da su isplanirane tri glavne kategorije – *Frizerski saloni, Upravljanje salonom* i *Profil*. U nastavku rada bit će detaljnije objašnjene funkcionalnosti svake kategorije.



Slika 3.1. *Dijagram toka aplikacije.*

3.2. Baza podataka

Kako bi podaci koje je korisnik unio ostali pohranjeni i nakon što isti prestane koristiti aplikaciju, potrebno ih je spremiti u bazu podataka. Shema baze podataka ove aplikacije nalazi se na slici 3.2.



Slika 3.2. *Shema baze podataka.*

Iz sheme je vidljivo da se baza podataka sastoji od šest glavnih tablica (*Saloni, Recenzije, Djelatnici, Korisnici, Rezervacije* i *Usluge*) te dvije pomoćne tablice (*Rezervacije-usluge* i *Saloni-usluge*). Svaka tablica sadrži atribute koje je potrebno pohraniti pa tako primjerice tablica *Recenzije* sadrži:

- id – primarni ključ tablice, jedinstvena vrijednost po kojoj se svaka recenzija može prepoznati
- created_at – datum i vrijeme kreiranja retka, tj. rezervacije
- updated_at – datum i vrijeme zadnjeg ažuriranja retka, tj. rezervacije
- title – naziv recenzije koji je korisnik unio
- description – tekst recenzije koji je korisnik unio

- rating – ocjena kojom je korisnik ocijenio salon
- salonId – strani ključ koji povezuje recenziju s određenim salonom iz tablice *Saloni* pomoću njegovog primarnog ključa (odnosi se na salon za koji je pisana recenzija)
- userId – strani ključ koji povezuje recenziju s korisnikom koji ju je napisao (iz tablice *Korisnici*).

Također, iz sheme se može vidjeti da postoje određene veze između tablica. Veze mogu biti tipa *jedan-na-jedan, jedan-na-više* ili *više-na-više.* Tako se, primjerice, jedna recenzija može odnositi samo na jedan salon, ali jedan salon može imati više recenzija (*jedan-na-više* tip), dok je veza između salona i usluga *više-na-više* jer jedan salon može nuditi više usluga, ali i određena usluga može biti ponuđena u više salona. Kod *jedan-na*-više veza (i *jedan-na-jedan,* no u ovom radu ne postoje takvi primjeri) koriste se strani ključevi unutar glavnih tablica kako bi se tablice međusobno povezale. Zato svaki redak u tablici *Recenzije*, kao što je već rečeno, sadrži atribute *userId* i *SalonId* kako bi svaki redak iz te tablice bio povezan s određenim retkom iz tablice *Korisnici* te određenim retkom iz tablice *Saloni* – jer jednu recenziju može napisati točno jedan određeni korisnik i točno za jedan određeni salon. Kod veza tipa *više-na-više* situacija je nešto složenija budući da se dvije tablice ne mogu izravno povezati pomoću stranog ključa. Primjer je već spomenuta veza između salona i usluga. Svaki salon može nuditi više usluga te svaka usluga može biti ponuđena u više različitih salona, stoga bi, kad bi se koristio isti princip stranog ključa kao kod veza *jedan-na-jedan* ili *jedan-na-više,* trebao postojati atribut *salonId* u tablici *Usluge* koji bi morao moći pohraniti više ID-jeva različitih salona ili obrnuto – trebao bi postojati atribut *serviceId* u tablici *Saloni* koji bi morao moći pohraniti više ID-jeva različitih usluga. Budući da to nije moguće, koriste se pomoćne tablice. Tablica *Saloni-usluge* tako (uz svoj primarni ključ te vremenske oznake kreiranja i zadnjeg ažuriranja) sadrži strane ključeve *salonId* i *serviceId* pa su tablice *Saloni* i *Usluge* pomoću nje međusobno povezane. Svaki redak u pomoćnoj tablici predstavlja jednu vezu u vezi *više-na-više,* tj. predstavljene su sve kombinacije povezanih redaka. Npr., ako salon s id=1 nudi usluge s id=1,2 i 3, a salon s id=2 nudi usluge 2,3 i 4, tablica *Saloni-usluge* izgledat će kao što je prikazano u tablici 3.1.

Tablica 3.1. Prikaz tablice Saloni-usluge.

id	salonId	serviceId
1	1	1
2	1	2
3	1	3
4	2	2
5	2	3
6	2	4

3.3. Poslužiteljski kôd

Poslužiteljski kôd aplikacije za rezerviranje termina i usluga u frizerskim salonima koristi se za komunikaciju s bazom podataka i, kao što je već spomenuto, izrađen je u Laravelu. Za izradu poslužiteljskog kôda korištene su osnovne funkcionalnosti Laravela: migracije, modeli i kontroleri te su kreirane krajnje točke kojima će poslužiteljski kôd primati i slati podatke s korisničkog sučelja (Angular kôd).

3.3.1. Migracije

Laravel migracije definiraju strukturu baze podataka, dakle za svaku tablicu iz baze podataka (slika 3.2.) potrebno je kreirati jednu migracijsku datoteku. Za kreiranje migracijskih datoteka može se koristiti naredbeni redak – naredbom *php artisan make:migration* kreira se nova migracijska datoteka određenog imena koju je zatim potrebno prilagoditi dodavanjem željenih atributa (stupaca za tablicu). Primjerice, za kreiranje migracijske datoteke *Recenzije* (engl. *reviews*) korištena je naredba *phpartisan make:migration create_reviews_table create=reviews*. Prilagodbom kôda kreirane datoteke dobiven je kôd prikazan listingom 3.1. Ovaj kôd opisuje kako će izgledati tablica *Recenzije* u bazi podataka. Svaki stupac tablice definiran je s *$table->*tip podataka stupca(ime stupca). Vidljivo je da će tablica *Recenzije* sadržavati stupce *id* tipa *big integer* s automatskom inkrementacijom *timestamps created_at* i *updated_at* tipa *timestamp*, *title* i *description* tipa *string*, *rating* tipa *integer* te *salonId* i *userId* tipa *unsigned big integer* – kako će ovo biti strani ključ, ovaj tip podatka mora se poklapati s tipom podatka stupca iz tablice na koji se poziva. Naposljetku se *salonId* i *userId* definiraju kao strani ključevi funkcijom *foreign()*, funkcijom *references()* definira se na koji stupac se pozivaju, a funkcijom *on()* definira se na koju tablicu se pozivaju.

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateReviewsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('reviews', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->timestamps();
            $table->string('title');
            $table->string('description');
            $table->integer('rating');
            $table->unsignedBigInteger('salonId');
            $table->unsignedBigInteger('userId');
            $table->foreign('salonId')->references('id')->on('salons');
            $table->foreign('userId')->references('id')->on('users');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('reviews');
    }
}
```

Listing 3.1. *Migracija za tablicu* Recenzije (Reviews).

Na isti način kreiraju se i ostale migracijske datoteke: create_services_table za tablicu Usluge, create_salons_table za tablicu Saloni, create_employees_table za tablicu Zaposlenici, create_users_table za tablicu Korisnici, create_reservations_table za tablicu Rezervacije, create_reservations_services_table za tablicu Rezervacije-usluge i create_salons_services_table za tablicu Saloni-usluge. Zatim se pomoću naredbenog retka i naredbe php artisan:migrate pokreće migracija te se stvara baza podataka koja u tom trenutku postaje funkcionalna – može se u nju pisati i iz nje čitati.

3.3.2. Modeli

Modeli su klase kojima su opisane tablice iz baze podataka. Pomoću njih se kreiraju, odnosno modeliraju objekti koji se koriste u daljnjem kôdu. Primjerice, kod dohvaćanja nekog salona iz baze podataka stvara se objekt klase *Salon* sa svim atributima iz te tablice.

Kako bi aplikacija funkcionirala, kreirani su modeli za sve glavne i pomoćne tablice iz baze podataka: Zaposlenik *(Employee)*, Rezervacija *(Reservation)*, Recenzija *(Review)*, Salon *(Salon)*, Korisnik *(User)*, Usluga *(Service)*, Rezervacija-usluga *(ReservationService)* te Salon-usluga *(SalonService)*. Slično kao i kod migracija, model se može kreirati naredbom naredbenog retka *php artisan make:model*. Tako je, primjerice, kreiran model *Review* za recenziju (a ostali su temeljeni na istom principu) koji nakon modifikacija izgleda kao kôd u listingu 3.2.. U polju *fillable* navode se oni atributi za koje je moguća masovna dodjela (engl. *mass assignment*), tj. oni atributi koje korisnik može izmijeniti. Ovaj korak provodi se prvenstveno radi zaštite baze podataka od neželjenih unosa. Npr., u tablici *Korisnici* postoji stupac *role* koji označava je li korisnik običan korisnik, administrator salona ili administrator čitavog sustava. Kad bi svaki korisnik mogao poslati zahtjev za izmjenom tog atributa, svaki korisnik mogao bi sebe postaviti kao administratora sustava i time ugroziti sigurnost baze jer ima pristup svim podacima – može ih mijenjati, dodavati, brisati. Ukoliko se atribut *role* ne navede u *fillable* polju modela *User*, neovisno o tome kakav objekt korisnik pošalje u nekom zahtjevu na poslužiteljski kôd i bazu podataka, neće moći izmijeniti taj atribut.

Nadalje, model *Review* sadrži i funkcije za povezivanje s drugim tablicama s kojima je tablica *Recenzije* povezana (*Saloni* i *Korisnici*). Kao što je već rečeno, jedna recenzija može biti napisana od strane točno jednog korisnika za točno jedan salon. Zato se za pronalazak tog salona i korisnika koristi funkcija *belongsTo()* koja kao parametre prima model koji se traži (*Salon* i *User*) te ime atributa, tj. stupca u tablici *Recenzije* koji je strani ključ za salon (*salonId*) i korisnika (*userId*).

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Review extends Model
{
    protected $fillable = [
        'title',
        'description',
        'rating',
```

```
            'salonId',
            'userId'
        ];

        /** Returns the salon which the review was written for. */
        public function salon() {
            return $this->belongsTo('App\Salon', 'salonId');
        }

        /** Returns the user who wrote the review. */
        public function user() {
            return $this->belongsTo('App\User', 'userId');
        }
    }
```

Listing 3.2. *Model* Recenzija *(*Review*).*

Osim *belongsTo()* funkcije, koristi se i funkcija *hasMany()* s istim parametrima koja se koristi u slučajevima kada je jedan entitet tablice (jedan objekt modela) povezan s više entiteta neke druge tablice. Tako primjerice jedan korisnik može napisati više recenzija pa je funkcija za pronalazak tih recenzija unutar *User* modela prikazana listingom 3.3.

```
    public function reviews() {
        return $this->hasMany('App\Review', 'userId');
    }
```

Listing 3.3. *Primjer funkcije* hasMany().

Naposljetku, koristi se i funkcija *belongsToMany()* koja se upotrebljava kod veza tipa *više-na-više*, kao što je veza između salona i usluga. Primjer funkcije za pronalazak usluga nekog salona prikazan je listingom 3.4. Ova funkcija također prima model koji se traži (*Service*), ali prima i ime pomoćne tablice koja povezuje dvije tablice u vezi *više-na-više* te imena atributa, tj. stupaca u toj tablici koji su strani ključevi za salon (*salonId*) te uslugu (*serviceId*).

```
    public function services() {
        return $this->belongsToMany('App\Service', 'salons_services',
'salonId', 'serviceId');
    }
```

Listing 3.4. *Primjer funkcije* belongsToMany().

Bitno je napomenuti da je u pravilu za sve tri funkcije nužan samo prvi parametar, a ostali nisu potrebni ukoliko se koristi standardna nomenklatura Laravela. Međutim, budući da je u ovom projektu nomenklatura nešto izmijenjena (npr. ne koristi se *salon_id,* nego *salonId*), program ne prepoznaje automatski o kojem se stupcu radi pa ga je potrebno dodatno navesti, kao i ime tablice u funkciji *belongsToMany*.

3.3.3. Kontroleri

Laravel kontroleri definiraju odgovor aplikacije na zahtjeve korisnika. Kada korisnik pošalje zahtjev poslužiteljskom kôdu, kontroler je taj koji taj zahtjev prima i određuje što će se izvršiti. Iako je teoretski moguće sve funkcije napisati unutar jednog kontrolera, radi preglednosti je kreirano više kontrolera: *EmployeeController* za upravljanje zaposlenicima, *ReservationController* za upravljanje rezervacijama, *ReviewController* za upravljanje recenzijama, *SalonController* za upravljanje salonima, *ServiceController* za upravljanje uslugama, *UserController* za upravljanje korisnicima, te *AuthController* za upravljanje prijavom i odjavom korisnika. Svi kontroleri osim *AuthControllera* vrlo su slični, stoga će njihov princip biti objašnjen na primjeru *SalonControllera,* dok će *AuthController* biti objašnjen zasebno.

SalonController

Unutar *SalonControllera* nalaze se sve funkcije za komunikaciju s bazom podataka. Potrebno je moći dohvatiti sve salone iz tablice ili jedan određeni salon, stvoriti novi salon, ažurirati postojeći te obrisati neki salon. Također je potrebno i dohvatiti sve recenzije određenog salona, kao i sve rezervacije za taj salon.

Dohvaćanje svih salona iz tablice *Saloni* izvršava se funkcijom *getAll()*, prikazanom kôdom u listingu 3.5. Funkcijom *Salon::get()* dohvaćaju se saloni iz tablice *Saloni* te se isti funkcijom *orderBy()* poredaju po datumu kreiranja, a budući da nema dodatnih funkcija kao što je *where()*, prvi redak funkcije *getAll()* zapravo izvršava SQL upit *SELECT * FROM salons* – dohvaća sve salone iz tablice *Saloni.* Svaki salon objekt je klase *Salon* iz modela *Salon.* Zatim se *foreach* petljom prolazi kroz svaki dohvaćeni salon te se pozivaju funkcije iz modela *Salon – services()* i *employees()* – koje dohvaćaju usluge i zaposlenike tog salona. Ti se zaposlenici i usluge zatim dodaju kao atributi tom *Salon* objektu kako bi funkcija mogla vratiti korisniku potpunu informaciju o salonu, odnosno da se ne mora s korisničkog sučelja slati novi zahtjev za informacijom o uslugama i zaposlenicima svakog salona nakon što se dobije popis salona.

```
public function getAll() {
    $salons = Salon::orderBy('created_at', 'desc')->get();
    foreach($salons as $salon) {
        $salon->services = $salon->services()->get();
        $salon->employees = $salon->employees()->get();
    }
    return $salons;
}
```

Listing 3.5. *Funkcija* getAll() *koja dohvaća sve salone.*

Na istom principu funkcionira i funkcija *getById*, koja dohvaća jedan određeni salon iz tablice *Saloni.* Funkcija je prikazana listingom 3.6. Funkcija *getById*() kao parametar prihvaća *id*, što je primarni ključ tablice *Saloni*, jedinstvena vrijednost po kojoj se svaki salon može prepoznati. Prvi redak funkcije predstavlja dohvaćanje salona iz tablice *Saloni,* što se izvršava funkcijom *findOrFail* kojoj se predaje isti parametar *id.* Dakle, izvršava se SQL upit *SELECT * FROM salons WHERE id=$id* te se dohvaća prvi rezultat. Nakon što je kreiran objekt *Salon*, kao i u funkciji *getAll()*, za taj salon se dohvaćaju njegovi usluge i zaposlenici te se potpuni objekt vraća kao rezultat funkcije.

```
public function getById($id) {
        $salon = Salon::findOrFail($id);
        $salon->services = $salon->services()->get();
        $salon->employees = $salon->employees()->get();
        return $salon;
    }
```

Listing 3.6. *Funkcija* getById() *za dohvaćanje određenog salona.*

Stvaranje novog salona izvršava se funkcijom *create()*, prikazanom listingom 3.7. Funkcija *create()* kao parametar prima varijablu *request,* koja je zapravo objekt koji sadrži sve potrebne informacije za stvaranje salona: ime, radno vrijeme, adresu itd. Funkcija prvo stvara novi objekt tipa *Salon* (iz modela *Salon*) te ga zatim popunjava vrijednostima iz objekta *request.* Budući da pri stvaranju salona isti nema nijednu recenziju, nema ni ocjena korisnika, stoga se vrijednosti atributa *rating* (ocjena) i *noRatings* (broj ocjena) postavljaju u nulu. Zatim se funkcijom *save()* izvršava SQL upit *INSERT INTO salons* sa svim atributima kreiranog *Salon* objekta te se korisničkom sučelju kao odgovor na zahtjev vraća kreirani *Salon* objekt.

```
    public function create(Request $request) {
        $salon = new Salon;
        $salon->name = $request->input('name');
        $salon->image = $request->input('image');
        $salon->workHoursStart = $request->input('workHoursStart');
        $salon->workHoursEnd = $request->input('workHoursEnd');
        $salon->email = $request->input('email');
        $salon->address = $request->input('address');
        $salon->city = $request->input('city');
        $salon->country = $request->input('country');
        $salon->postCode = $request->input('postCode');
        $salon->latitude = $request->input('latitude');
        $salon->longitude = $request->input('longitude');
        $salon->phone = $request->input('phone');
        $salon->rating = 0;
        $salon->noRatings = 0;
        if ($salon->save()) {
            return $salon;
        }
    }
```

Listing 3.7. *Funkcija za kreiranje novog salona.*

Na sličan način funkcionira i funkcija *update()* koja se koristi za ažuriranje postojećeg salona, prikazana kôdom u listingu 3.8. Funkcija kao parametar također prima *request* koja sadrži informacije za ažuriranje salona, ali uz njega prima i *id,* primarni ključ po kojem se može prepoznati određeni salon. Funkcija prvo iz baze podataka funkcijom *findOrFail()* dohvati salon pomoću njegovog ID-ja. Zatim se za svaki atribut tog salona vrijednosti mijenjaju u nove vrijednosti iz zahtjeva, ukoliko one u zahtjevu postoje, jer u određenim slučajevima korisnik ne želi izmijeniti sve vrijednosti, nego samo određene. Nakon toga, kao i kod funkcije *create()* poziva se funkcija *save()* koja sprema izmijenjeni salon u bazu podataka te se korisničkom sučelju vraća izmijenjeni *Salon* objekt.

```
        public function update(Request $request, $id) {
        $salon = Salon::findOrFail($id);
        if($request->input('name')) {
            $salon->name = $request->input('name');
        }
        if($request->input('image')) {
            $salon->image = $request->input('image');
        }
        if($request->input('workHoursStart')) {
            $salon->workHoursStart = $request->input('workHoursStart');
        }
        if($request->input('workHoursEnd')) {
            $salon->workHoursEnd = $request->input('workHoursEnd');
        }
        if($request->input('email')) {
            $salon->email = $request->input('email');
        }
        if($request->input('address') && $request->input('city')
&& $request->input('postCode') && $request->input('latitude')
&& $request->input('longitude')&& $request->input('country')) {
            $salon->address = $request->input('address');
```

```
            $salon->city = $request->input('city');
            $salon->postCode = $request->input('postCode');
            $salon->latitude = $request->input('latitude');
            $salon->longitude = $request->input('longitude');
            $salon->country = $request->input('country');
        }
        if($request->input('phone')) {
            $salon->phone = $request->input('phone');
        }
        if ($request->input('rating') && $request->input('noRatings')) {
            $salon->rating = $request->input('rating');
            $salon->noRatings = $request->input('noRatings');
        }
        if ($salon->save()) {
            return $salon;
        }
    }
```

Listing 3.8. *Funkcia za ažuriranje postojećeg salona.*

Funkcija *delete()* omogućuje brisanje određenog salona iz baze podataka, a prikazana je kôdom u listingu 3.9. Ova funkcija također prima jedinstveni *id* salona te dohvaća taj salon iz baze podataka. Zatim se, funkcijom *delete()* brišu sve usluge tog salona iz tablice *Saloni-usluge*, sve postojeće rezervacije za taj salon iz tablice *Rezervacije-usluge* i iz tablice *Rezervacije* te svi zaposlenici tog salona iz tablice *Zaposlenici*. Ovaj korak nužan je zato što su sve navedene tablice vezane uz tablicu *Saloni* te nije moguće obrisati salon iz baze podataka ako neki redak u jednoj od tih tablica kao strani ključ *salonId* sadrži ID salona koji se pokušava obrisati. Zatim je također potrebno postaviti ulogu korisnika koji je dosad bio administrator salona *(salon_admin*) na običnog korisnika (*user*) budući da salon kojim je taj korisnik dosad upravljao više ne postoji. Nakon toga se salon konačno funkcijom *delete()* briše, odnosno izvršava se SQL upit *DELETE FROM salons WHERE id=$id* te se korisničkom sučelju vraća poruka o uspješnom brisanju salona iz baze podataka.

```
    public function delete($id) {
        $salon = Salon::findOrFail($id);
        DB::table('salons_services')->where(['salonId' => $id])->delete();
        $salon->reviews()->delete();
        foreach($salon->reservations as $reservation) {
            DB::table('reservations_services')->where(['reservationId' =>
$reservation->id])->delete();
        }
        $salon->reservations()->delete();
        $salon->employees()->delete();
        foreach($salon->admins as $admin) {
            $user = User::findOrFail($admin->id);
            if ($user->role !== "sys_admin") {
                $user->role = "user";
            }
        }
        if ($salon->delete()) {
            return "Successfully deleted.";
        }
    }
```

Listing 3.9. *Funkcija za brisanje salona iz baze podataka.*

Funkcije za dohvaćanje recenzija i rezervacija za određeni salon zasnovane su na istom principu, a u listingu 3.10. nalazi se kôd za funkciju *getReservations()* koja dohvaća rezervacije za pojedini salon. Funkcija prima ID salona kao parametar te funkcijom *findOrFail()* pronalazi taj salon u bazi podataka, unutar tablice *Saloni*. Zatim se funkcijom *reservations()* iz modela *Salon* dohvaćaju sve rezervacije za taj salon. Nakon toga se *foreach* petljom prolazi kroz sve rezervacije te se svakoj rezervaciji pridružuju atributi *user* (korisnik koji je napravio rezervaciju), *salon* (salon za koji je napravljena rezervacija), *employee* (zaposlenik koji je zadužen za rezervaciju) te *services* (usluge koje su rezervirane u sklopu rezervacije). Naposljetku funkcija vraća polje s potpunim objektima.

```
    public function getReservations($salonId) {
        $salon = Salon::findOrFail($salonId);
        $reservations = $salon->reservations()->get();
        foreach($reservations as $reservation) {
            $reservation->user = $reservation->user()->get();
            $reservation->salon = $reservation->salon()->get();
            $reservation->employee = $reservation->employee()->get();
            $reservation->services = $reservation->services()->get();
        }
        return $reservations;
    }
```

Listing 3.10. *Funkcija za dohvaćanje rezervacija za pojedini salon.*

AuthController

AuthController koristi se za upravljanje prijavom i odjavom korisnika iz sustava. Autentifikacija korištenjem standardnih obrazaca za prijavu unutar Laravel aplikacije vrlo je jednostavna, no

kako se u ovoj aplikaciji Laravel koristi samo za poslužiteljski kôd, a obrazac za prijavu napravljen je u Angularu, koristi se biblioteka *Passport* koja brzo i jednostavno implementira OAuth2 autentifikaciju – industrijski-standardni protokol za autorizaciju [6].

Nakon što je *Passport* dodan u aplikaciju te su sve postavke u aplikaciji podešene za njega, vrlo je jednostavno u *AuthControlleru* implementirati metode za prijavu i odjavu korisnika. Stoga je metoda za prijavu korisnika *login()* implementirana kôdom u listingu 3.11. Funkcija *login()* kao parametar prima *request* odnosno objekt s traženim podacima – e-mail adresom, lozinkom te varijablom *remember_me* koja je tipa *boolean,* a označava hoće li korisnik ostati prijavljen dulje vrijeme ili ne. Funkcija prvo provjerava ispravnost unesenih podataka, odnosno podudaraju li se tipovi podataka iz zahtjeva s traženim tipovima podataka. Zatim se šalje upit u bazu podataka s podacima iz zahtjeva te se pokušava prijava u sustav. Ukoliko je prijava neuspješna, funkcija korisničkom sučelju vraća pogrešku s porukom da korisnik nije autoriziran. U suprotnom, korisnik je prijavljen te se stvara jedinstveni niz znakova – *token* koji označuje trenutnu sesiju prijavljenog korisnika. *Token* se sprema u zasebnu tablicu unutar baze podataka koja je kreirana pri instalaciji *Passporta* te se također sprema u tablicu *Korisnici* u zasebnom stupcu unutar retka korisnika koji se u sustav prijavio, budući da će se taj *token* koristiti pri svakom zahtjevu nad bazom podataka. U navedene tablice također se sprema i datum i vrijeme *expires_at* koje označava kada *token* ističe, odnosno kada trenutna sesija korisnika prestaje biti validna.

```
    public function login(Request $request) {
        $request->validate([
            'email' => 'required|string|email',
            'password' => 'required|string',
            'remember_me' => 'boolean'
        ]);

        $credentials = request(['email', 'password']);

        if(!Auth::attempt($credentials))
            return response()->json([
                'message' => 'Unauthorized'
            ], 401);

        $user = $request->user();

        $tokenResult = $user->createToken('Personal Access Token');
        $token = $tokenResult->token;

        if ($request->remember_me) {
            $token->expires_at = Carbon::now()->addWeeks(1)-
>setTimezone('UTC');
        } else {
            $token->expires_at = Carbon::now()->addHours(1)-
>setTimezone('UTC');
        }
```

```
        if ($token->save()) {
            $user->accessToken = $tokenResult->accessToken;
            $user->tokenExpiresAt = Carbon::parse($tokenResult->token-
>expires_at)->toDateTimeString();
            if ($user->save()) {
                return $user;
            }
        }
    }
```

Listing 3.11. *Funkcija za prijavu korisnika u sustav.*

Uz funkciju *login()* u *AuthControlleru* postoji i funkcija *logout()* čija svrha je odjava korisnika iz sustava. Kôd iste prikazan je listingom 3.12. Funkcija *logout()* pronalazi trenutno prijavljenog korisnika, postavlja vrijednosti *tokena* i njegovog vremena isteka u tablici *Korisnici* na *null* te metodom *revoke()* povlači *token,* odnosno uklanja ga i iz tablice koju je u bazi podataka stvorio *Passport*. Ukoliko su ove dvije radnje uspješne, korisnik dobiva poruku o uspjehu, a u suprotnom dobiva poruku o grešci.

```
    public function logout(Request $request)
    {
        $user = $request->user();
        $user->accessToken = null;
        $user->tokenExpiresAt = null;
        if ($request->user()->token()->revoke() && $user->save()) {
            return response()->json([
                'message' => 'Successfully logged out'
            ]);
        }
        return response()->json([
            'error' => 'Not logged out'
        ]);
    }
```

Listing 3.12. *Funkcija za odjavu korisnika iz sustava.*

3.3.4. Krajnje točke

Kao što je već rečeno, Laravel se u sklopu ovog diplomskog rada koristi samo za poslužiteljski dio aplikacije, a korisničko sučelje realizirano je Angularom. Kako bi ove dvije cjeline mogle međusobno komunicirati, potrebno je odrediti krajnje točke (engl. *endpoints*), odnosno rute (engl. *routes*) koje će korisnik putem korisničkog sučelja posjetiti kako bi pristupio pojedinim metodama iz Laravel kontrolera. Sve rute definirane su u *api.php* datoteci unutar Laravel projekta te zapravo predstavljaju web adrese kojima korisnik pristupa i time okida određeni zahtjev, odnosno Laravel metodu iz određenog kontrolera.

Unutar *api.php* datoteke rute su podijeljene u sedam grupa: *reviews* za sve metode iz *ReviewControllera*, *salons* za sve metode iz *SalonControllera*, *employees* za sve metode iz

EmployeeControllera, services za sve metode iz *ServiceControllera*, *reservations* za sve metode iz *ReservationsControllera, users* za sve metode iz *UserControllera* te *auth* za sve metode iz *AuthControllera.* Kako su sve ove grupe slične, princip će biti objašnjen na grupi *salons,* čiji je kôd prikazan u listingu 3.13. Iz kôda je vidljivo da ova grupa ima prefiks *salons*, što znači da svaka ruta unutar grupe započinje sa *salons/*. Svaka ruta objekt je tipa *Route* nad kojim se vrši određena metoda – *get* za dohvaćanje, *post* za unos, *put* za ažuriranje te *delete* za uklanjanje iz baze podataka. Sve te metode primaju po dva parametra – prvi predstavlja dodatnu adresu rute, a drugi metodu koja će se izvršiti kad korisnik pristupi toj ruti.

```
Route::group([
    'prefix' => 'salons'
], function () {
    Route::get('', 'SalonController@getAll');
    Route::get('{id}', 'SalonController@getById');
    Route::group([
        'middleware' => 'auth:api'
    ], function() {
        Route::get('{id}/reviews', 'SalonController@getReviews');
        Route::get('{id}/reservations', 'SalonController@getReservations');
        Route::post('', 'SalonController@create');
        Route::put('{id}', 'SalonController@update');
        Route::delete('{id}', 'SalonController@delete');
    });
});
```

Listing 3.13. *Rute za* SalonController.

Primjerice, ukoliko poslužitelj radi na adresi *http://127.0.0.1:8000/api*, prva ruta *Route::get('', 'SalonController@getAll')* označava da korisničko sučelje treba poslati zahtjev *GET* na adresu *http://127.0.0.1:8000/api/salons* čime će se izvršiti funkcija *getAll()* iz *SalonControllera* koja dohvaća sve salone. S druge strane, ukoliko želi ukloniti salon s primarnim ključem *id* = 4, *Route::delete('{id}', 'SalonController@delete'),* korisnik treba poslati *DELETE* zahtjev na adresu *http://127.0.0.1:8000/api/salons/4*.

Također valja primijetiti da unutar glavne grupe postoji manja grupa koja sadrži svojstvo *middleware* čija je vrijednost *auth:api*. Svim rutama unutar te grupe moguće je pristupiti samo ukoliko je zahtjev autoriziran, odnosno ako se u zaglavlju HTTP zahtjeva pošalje potvrda autorizacije (korisnikov *token* zapisan u bazi prilikom prijave u aplikaciju). U ovom primjeru to znači da svaki korisnik (prijavljen ili ne) može pristupiti svim salonima ili određenom salonu, ali samo prijavljeni korisnici mogu vidjeti rezervacije i recenzije nekog salona, kreirati novi salon te ažurirati ili ukloniti postojeći.

3.4. Korisničko sučelje

Svaka Angular aplikacija sastoji se od jednog ili više modula. Modul zapravo predstavlja konstruktor aplikacije i u svakom modulu se nalaze komponente, servisi, uvozi drugih modula i ostale Angular sastavnice koje pojedini modul koristi i o kojima ovisi. Glavni modul zove se *AppModule* i u njemu se nalazi osnovna komponenta *AppComponent. AppComponent* sadrži samo usmjerivač (engl. *router*) koji omogućuje navigaciju po svim ostalim komponentama u aplikaciji pomoću URL-ova. Osim glavnog modula, aplikacija za rezerviranje termina i usluga u frizerskim salonima sadrži i dodatne module, a radi preglednosti aplikacija je podijeljena u četiri glavna direktorija s pripadajućim modulima, komponentama i servisima: *core, popups, shared* i *modules.* Svi ti moduli, komponente i servisi uglavnom su međuovisni te čine pet glavnih funkcionalnih cjelina:

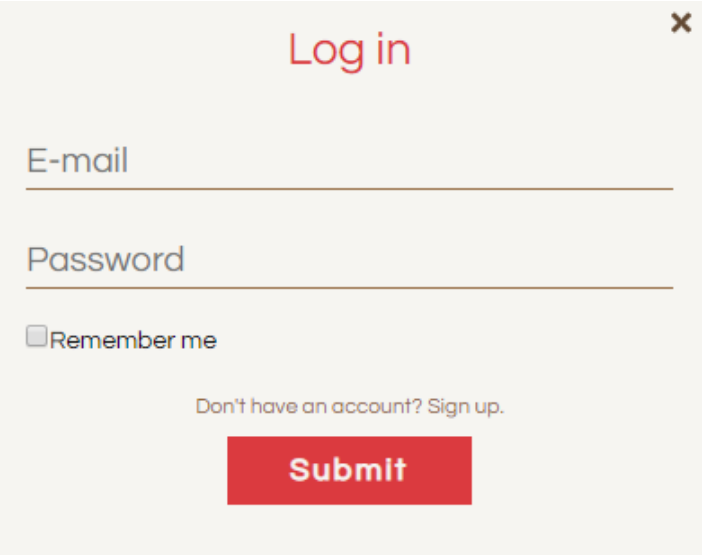
- autorizaciju korisnika
- raspored (engl. *layout*) aplikacije
- upravljanje salonom – informacijama i rezervacijama
- pregled svih salona, odabir određenog salona i rezervacija termina
- pregled arhive rezervacija.

3.4.1. Autorizacija

Kako bi se korisnik prijavio u sustav, potrebno je poslati HTTP zahtjev na poslužitelj. Nakon što korisnik u obrazac za prijavu iz komponente *LoginComponent*, prikazanu na slici 3.3., upiše svoje podatke i klikne gumb *Submit*, poziva se funkcija *login()* čiji kôd je prikazan u listingu 3.14.

```
public login(): void {
  this.loginForm.patchValue({
    email: this.email.input,
    password: this.password.input,
  });
  this._auth.login(this.loginForm.value)
    .subscribe(
      data => {
        localStorage.setItem('user', JSON.stringify(data));
        this._popups.toggleLogin();
      }
    )
}
```

Listing 3.14. *Funkcija za prijavu korisnika unutar komponente* LoginComponent.



Slika 3.3. *Obrazac za prijavu korisnika u sustav.*

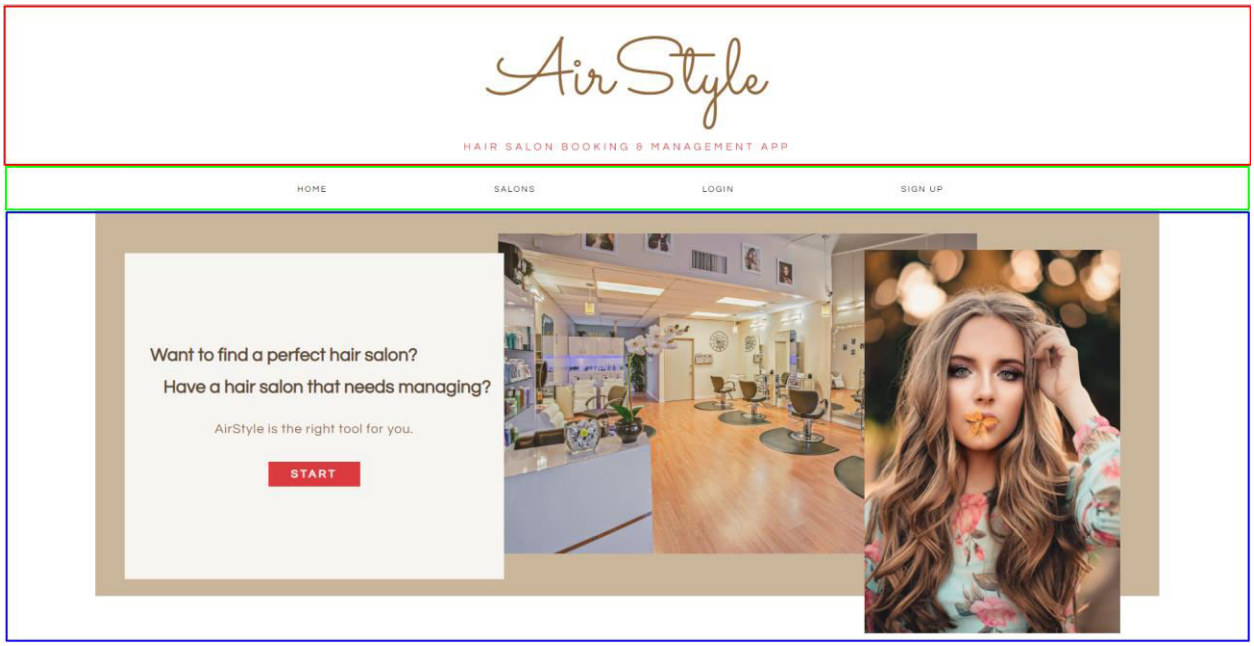
Funkcija *login()* unutar *LoginComponent* poziva funkciju *login()* unutar servisa za autorizaciju - *AuthService*. *AuthService login()* metoda prikazana je kôdom u listingu 3.15. U ovoj metodi koristi se HTTP zahtjev POST te se podaci (e-mail adresa i lozinka) na poslužiteljski dio kôda pomoću krajnje točke */login*. Kako funkcija za prijavu korisnika na poslužitelju vraća objekt s podacima o prijavljenom korisniku, nakon što metoda na klijentu primi taj objekt, on se sprema u novi objekt klase *Auth* koja je modelirana prema objektu koji vraća funkcija s poslužitelja. Funkcija vraća pohranjeni objekt komponenti *LoginComponent* koja ju je pozvala i funkcijom *subscribe()* čekala na odgovor *AuthServicea,* a time i poslužitelja. Kad dobije odgovor, funkcija pohranjuje primljeni objekt u lokalnu pohranu web preglednika (engl. *local storage*) i prijava korisnika je gotova. Sukladno tome, pri odjavi korisnika također se šalje zahtjev na poslužitelj pomoću krajnje točke */logout* te se iz lokalne pohrane uklanja objekt s podacima o korisniku.

```
public login(formData): Observable<Auth> {
  const form = new FormData();
  form.append('email', formData.email);
  form.append('password', formData.password);
  return this.http.post<Auth>(this.url + 'login', form)
    .pipe(
      map(data => {
        const obj = new Auth();
        Object.assign(obj, data);
        return obj;
      })
    )
}
```

Listing 3.15. *Funkcija za prijavu korisnika unutar servisa* AuthService.

3.4.2. Raspored aplikacije

Raspored aplikacije, odnosno način na koji će ona izgledati određen je trima koponentama, prikazanima na slici 3.4..: *HeaderComponent* za zaglavlje (na slici označeno crvenom bojom), *NavigationComponent* za navigacijsku traku (na slici označena zelenom bojom) i *MainComponent* za glavni dio prozora aplikacije (na slici označen plavom bojom). Sve tri kopmonente deklarirane su u modulu *LayoutModule*.



Slika 3.4. *Prikaz glavnog rasporeda aplikacije.*

LayoutModule koristi posebno deklarirani *LayoutRoutingModule* koji se nadovezuje na *router* iz glavne komponente i omogućuje prikaz pojedine komponente unutar *MainComponent* komponente ovisno o URL-u na kojem se korisnik trenutno nalazi, što je prikazano kôdom u listingu 3.16. Primjerice, ako je osnovna ruta aplikacije *http://localhost:4200/*, korisnik koji posjeti URL *http://localhost:4200/salons* unutar *MainComponent* vidjet će komponentu *SalonsComponent* za prikaz svih salona. Rutama se također može predati i parametar, što se vidi u ruti *salon/:id*, pri čemu je *id* parametar koji *SalonComponent* prepoznaje i može koristiti.

```
const routes: Routes = [
  {
    path: '',
    component: MainComponent,
    children: [
      {
        path: '',
        component: HomeComponent,
      },
      {
```

```
        path: 'salons',
        component: SalonsComponent
      },
      {
        path: 'manage',
        component: ManageSalonComponent,
        canActivate: [AuthGuard]
      },
      {
        path: 'salon/:id',
        component: SalonComponent
      },
      {
        path: 'profile',
        component: ProfileComponent,
        canActivate: [AuthGuard]
      }
    ]
  },
];
```

Listing 3.16. *Rute za navigaciju po aplikaciji.*

Iz kôda se također može vidjeti da neke rute sadrže svojstvo *canActivate* s poljem u kojem se nalazi vrijednost *AuthGuard. Guard* je sastavnica Angulara koja govori *routeru* može li korisnik pristupiti traženoj ruti ili ne. *Guard* uvijek implementira ugrađeno Angular sučelje *CanActivate* te proširuje njegovu metodu *canActivate()* koja vraća *true* (korisnik može pristupiti ruti) ili *false* (korisnik ne može pristupiti ruti). U slučaja *AuthGuarda* funkcija *canActivate()* provjerava postoji li u lokalnoj pohrani objekt s podacima o korisniku, odnosno je li korisnik prijavljen u sustav ili ne. Ukoliko pronađe taj objekt, funkcija vraća *true* i korisnik može pristupiti ruti, a ukoliko objekta nema, funkcija vraća *false*, ne dopušta pristup ruti i preusmjerava korisnika na početnu stranicu aplikacije.

Ruta na kojoj se korisnik nalazi se najvećim dijelom kontrolira putem komponente za navigaciju, *NavigationComponent*, čiji je HTML kôd prikazan listingom 3.17. Navigacija, tj. preusmjeravanje na neku rutu vrši se funkcijom *router.navigate()* koja kao parametar prima *string*, odnosno rutu na koju korisnika treba preuzimati. Tako se klikom na drugi gumb u navigacijskoj komponenti korisnika preusmjerava na rutu *salons* gdje će, prema *LayoutRoutingModuleu* vidjeti komponentu *SalonsComponent* kao što je već objašnjeno.

```
<div class="navigation">
    <button (click)="router.navigate([''])">Home</button>
    <button (click)="router.navigate(['salons'])">Salons</button>
    <button *ngIf="_auth.user" (click)="router.navigate(['manage'])">Manage
Salon</button>
    <button *ngIf="_auth.user"
(click)="router.navigate(['profile'])">Profile</button>
    <button *ngIf="_auth.user" (click)="logout()">Logout</button>
```

```
    <button *ngIf="!_auth.user"
(click)="_popups.toggleLogin()">Login</button>
    <button *ngIf="!_auth.user" (click)="_popups.toggleSignup()">Sign
up</button>
</div>
```

Listing 3.17. *Komponenta za navigaciju.*

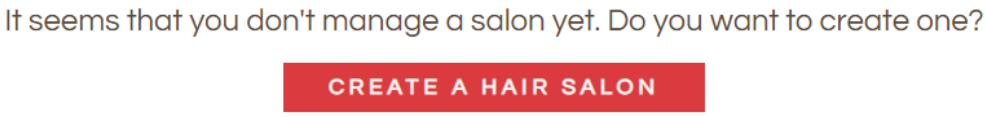
U ovom kôdu također se može uočiti i atribut **ngIf* u pojedinim gumbima. **ngIf* je jedna od ugrađenih Angular direktiva koja omogućuje uvjetno prikazivanje određenog HTML elementa. Tako će ovdje neki gumbi biti prikazani uvijek (gumbi koji ne sadrže **ngIf*), neki samo ako je korisnik prijavljen u sustav (**ngIf="_auth.user"*), dok će neki gumbi biti prikazani samo ako korisnik nije prijavljen u sustav (**ngIf="!_auth.user"*).

3.4.3. Upravljanje salonom

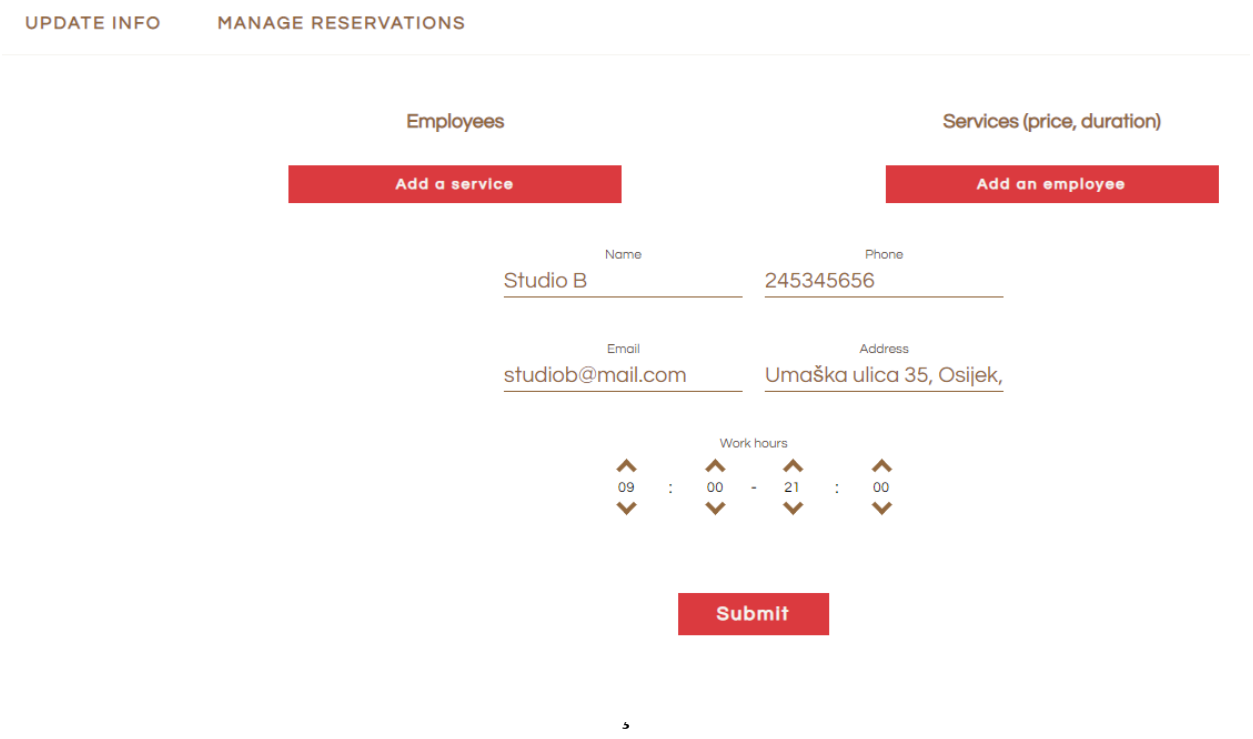
Kad se korisnik registrira u sustav, poslužiteljski kôd mu automatski dodjeljuje ulogu običnog korisnika. Međutim, ukoliko s navigacijske trake odabere opciju *Manage salon*, korisnik može dodati svoj frizerski salon u bazu podataka i time postati administrator tog salona. U sustavu također postoji i sustavski administrator koji može upravljati svim salonima. Tako *ManageSalon* komponenta različito izgleda ovisno o tome koja je uloga korisnika koji je trenutno prijavljen u sustav, što je prikazano HTML kôdom te komponente (listing 3.18.). Korištenjem **ngIf* direktive osigurano je da se ovisno o ulozi korisnika uvijek prikaže samo jedna od tri moguće komponente. Ukoliko je uloga korisnika *user* (običan korisnik), prikazuje mu se komoponeta *ManageRoleUserComponent* gdje može dodati svoj frizerski salon, ukoliko želi (slika 3.5.). Ukoliko je uloga korisnika *salon_admin* (administrator jednog salona), prikazuje mu se komponenta *ManageRoleSalonAdminComponent* koja kao atribut *salonId* prima ID salona čiji je korisnik administrator te korisnik može upravljati svojim salonom (slika 3.6.). Ako je uloga korisnika *sys_admin* (sustavski administrator), prikazuje mu se komponenta *ManageRoleSysAdminComponent* te korisnik može odabrati kojim salonom želi upravljati (slika 3.7.), nakon čega mu se također prikazuje komponenta *ManageRoleSalonAdminComponent* pri čemu je vrijednost atributa *salonId* jednaka ID-ju salona koji je korisnik odabrao.

```
<app-manage-role-user *ngIf="_auth.user.role === 'user'"></app-manage-
role-user>
<app-manage-role-salon-admin *ngIf="_auth.user.role === 'salon_admin'"
[salonId]=_auth.user.salonId></app-manage-role-salon-admin>
<app-manage-role-sys-admin *ngIf="_auth.user.role ===
'sys_admin'"></app-manage-role-sys-admin>
```

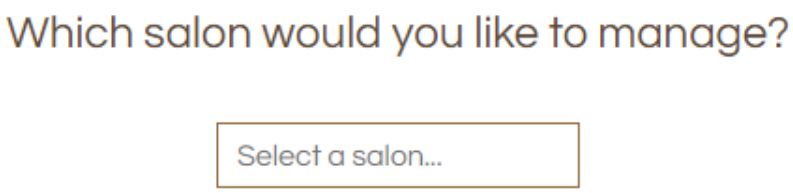
Listing 3.18. *Komponenta za upravljanje salonom.*



Slika 3.5. *Izgled* ManageRoleUserComponent *komponente*.



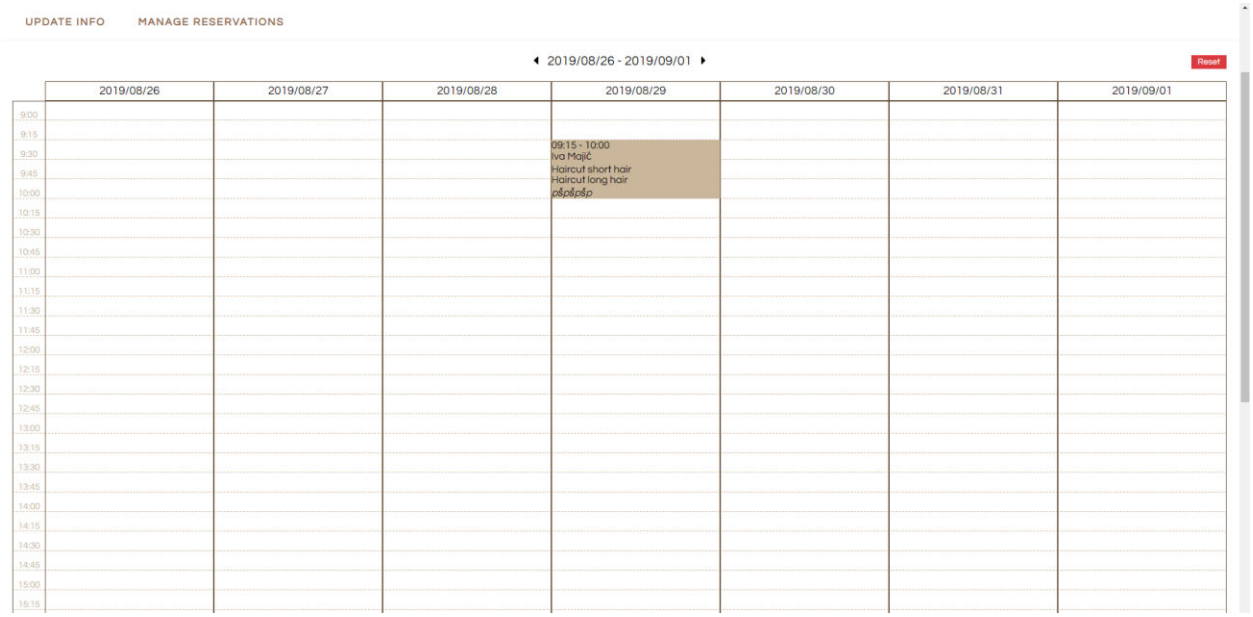
Slika 3.6. *Izgled* ManageRoleSalonAdminComponent *komponente.*



Slika 3.7. *Izgled* ManageRoleSysAdminComponent *komponente.*

Kao što je vidljivo iz slike 3.6., administrator salona i administrator sustava mogu mijenjati podatke o salonu te dodavati nove usluge ili zaposlenike u bazu podataka. Svaka od tih radnji, kao i kreiranje samog salona odvija se pozivima na poslužitelj putem za to predviđenih krajnjih točaka, na isti način kao što je objašnjeno za funkciju za prijavu korisnika u sustav – klikom na gumb *Submit* poziva se funkcija unutar komponente koja okida odgovarajuću funkciju u za to predviđenom servisu. Servis komunicira s poslužiteljem i šalje zahtjev, zatim prima odgovor i vraća ga komponenti čime je zahtjev uspješno završen. Primjerice, ako poslužitelj radi na adresi *http://127.0.0.1:8000/api,* za kreiranje salona šalje se zahtjev tipa POST na adresu *http://127.0.0.1:8000/api/salons* te se u tijelu zahtjeva predaje objekt s parametrima potrebnim za kreiranje salona.

Osim podacima o salonu, administratori mogu upravljati i rezervacijama – mogu vidjeti raspored rezervacija za taj salon (tko je rezervirao termin, koje vrijeme, koje usluge i kojeg djelatnika) te, ukoliko do vremena početka usluge ima više od dva sata, po potrebi ukloniti rezervaciju. Na slici 3.8. vidljiv je prikaz rezervacija za jedan tjedan. Prikaz rezervacija kreiran je kao tjedni kalendar na kojem se svaka rezervacija prikazuje kao smeđi element u kojem pišu informacije o rezervaciji: vrijeme, ime korisnika koji je rezervirao termin, usluge koje je korisnik rezervirao te napomene koje je korisnik ostavio.



Slika 3.8. *Prikaz tjednih rezervacija za salon pri upravljanju salonom.*

Prikaz tablice rezervacija omogućen je HTML kôdom prikazanim u listingu 3.19.

```
      <div class="reservations__table__table__body">
        <div class="reservations__table__table__body__date" *ngFor="let date
of dates">
          <div class="reservations__table__table__body__date__interval"
*ngFor="let interval of timeIntervals"></div>
          <div class="reservations__table__table__body__date__reservation"
*ngFor="let reservation of date.reservations"
            [ngStyle]="{'top': reservation.top + 'px', 'height':
reservation.height + 'px'}">
            <fa-icon *ngIf="reservation.isInFuture" [icon]="faTrash"
              (click)="deleteReservation($event, reservation.id)"></fa-icon>
            <div>{{reservation.startHours}} - {{reservation.endHours}}</div>
            <div style="margin-bottom: 3px">{{reservation.user[0].firstName}}
{{reservation.user[0].lastName}}</div>
            <div *ngFor="let service of
reservation.services">{{service.name}}</div>
            <div style="font-style: italic; margin-top:
3px">{{reservation.notes}}</div>
            <div></div>
          </div>
        </div>
      </div>
```

Listing 3.19. *HTML kôdz za prikaz tablice rezervacija.*

Za prikaz rezervacija koristi se još jedna Angular direktiva: **ngFor*. Ova direktiva omogućuje da se neki HTML element ponovi za svaki element polja definiranog u komponenti. Tako se ovdje drugi *div element* ponavlja za svaki element *date* iz polja *dates* koje predstavlja jedan datum u tjednu. Zatim je unutar tog elementa ugniježđen još jedan element s oznakom **ngFor* koji se ponavlja za svaki interval od petnaest minuta, što ukupno omogućuje tablični prikaz. Zanimljivo je napomenuti i prikaz samih elemenata rezervacija, koje su grupirane po danima te ih se može pronaći pod *div* elementom klase *reservations__table__table__body__date__reservation.* Ponovno, svaki element ponavlja se onoliko puta koliko rezervacija ima, no ovdje dodatno postoji i direktiva *ngStyle.* Ova direktiva omogućuje da se dinamički dodaje stil nekom HTML elementu. Konkretno, ovdje se svakom elementu koji predstavlja rezervaciju dodjeljuje određena pozicija u odnosu na vrh stranice (engl. *top*) te visina engl. *height*). Vrijednosti ta dva svojstva određene su kôdom prikazanim u listingu 3.20. Iz kôda se vidi da se za svaku rezervaciju računa njezina visina i položaj. Iz početka i kraja rezervacije dobivenog s poslužitelja izvuku se sati i minute te se pretvaraju u decimalni oblik (sati). Zatim se računa broj intervala od početka do kraja rezervacije (za visinu elementa) te broj intervala od početka radnog vremena salona do početka rezervacije (za položaj elementa). Svaki interval iznosi petnaest minuta (0.25 h) stoga se broj intervala računa kao razlika između dva navedena vremena podijeljena s 0.25. Naposljetku se broj intervala množi s 30 budući da svaki interval ima visinu od 30 piksela. Dobivena visina i položaj koriste se u opisanom HTML predlošku.

```
const startHours = moment.utc(reservation.startTime).hours();
const startMinutes = moment.utc(reservation.startTime).minutes();
const endHours = moment.utc(reservation.endTime).hours();
const endMinutes = moment.utc(reservation.endTime).minutes();
const start = startHours + startMinutes / 60;
const end = endHours + endMinutes / 60;
const noIntervalsHeight = (end - start) / 0.25;
const noIntervalsTop = (start - this.salon.workHoursStart) / 0.25;
reservation.height = noIntervalsHeight * 30;
reservation.top = noIntervalsTop * 30 + 30;
```

Listing 3.20. *Računanje visine i položaja rezervacije u kalendaru rezervacija.*

Kako bi program znao je li rezervacija u budućnosti, svaki objekt rezervacije koji se treba prikazati u arhivi sadrži atribut *isInFuture* koji je tipa *boolean* te mu je vrijednost *true* ukoliko do rezervacije ima još minimalno dva sata, a u suprotnom mu je vrijednost *false*. To se postiže kôdom u listingu 3.21. Za rukovanje datumima i vremenima u ovom slučaju, kao i u ostatku projekta koristi se bibiloteka *moment* koja isto znatno olakšava. U ovom slučaju biblioteka *moment* dohvaća trenutno vrijeme funkcijom *moment.utc()*, dodaje mu dva sata funkcijom *add()* te funkcijom *isAfter()* uspoređuje to vrijeme s vremenom početka rezervacije.

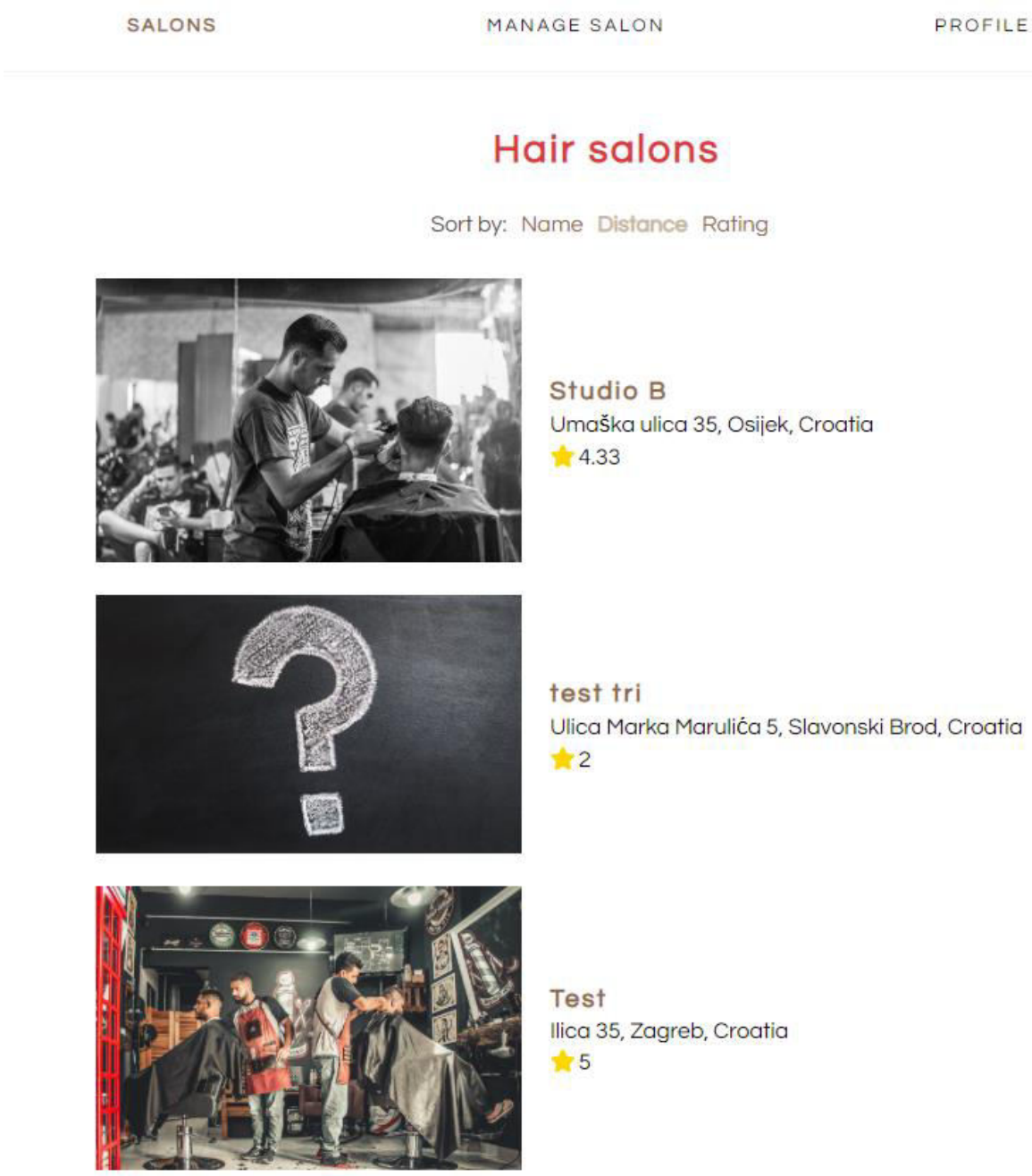
```
reservation.isInFuture =
moment.utc(reservation.startTime).isAfter(moment.utc().add(2, "hour"));
```

Listing 3.21. *Određivanje je li rezervacija u budućnosti.*

3.4.4. Pregled salona i rezervacija termina

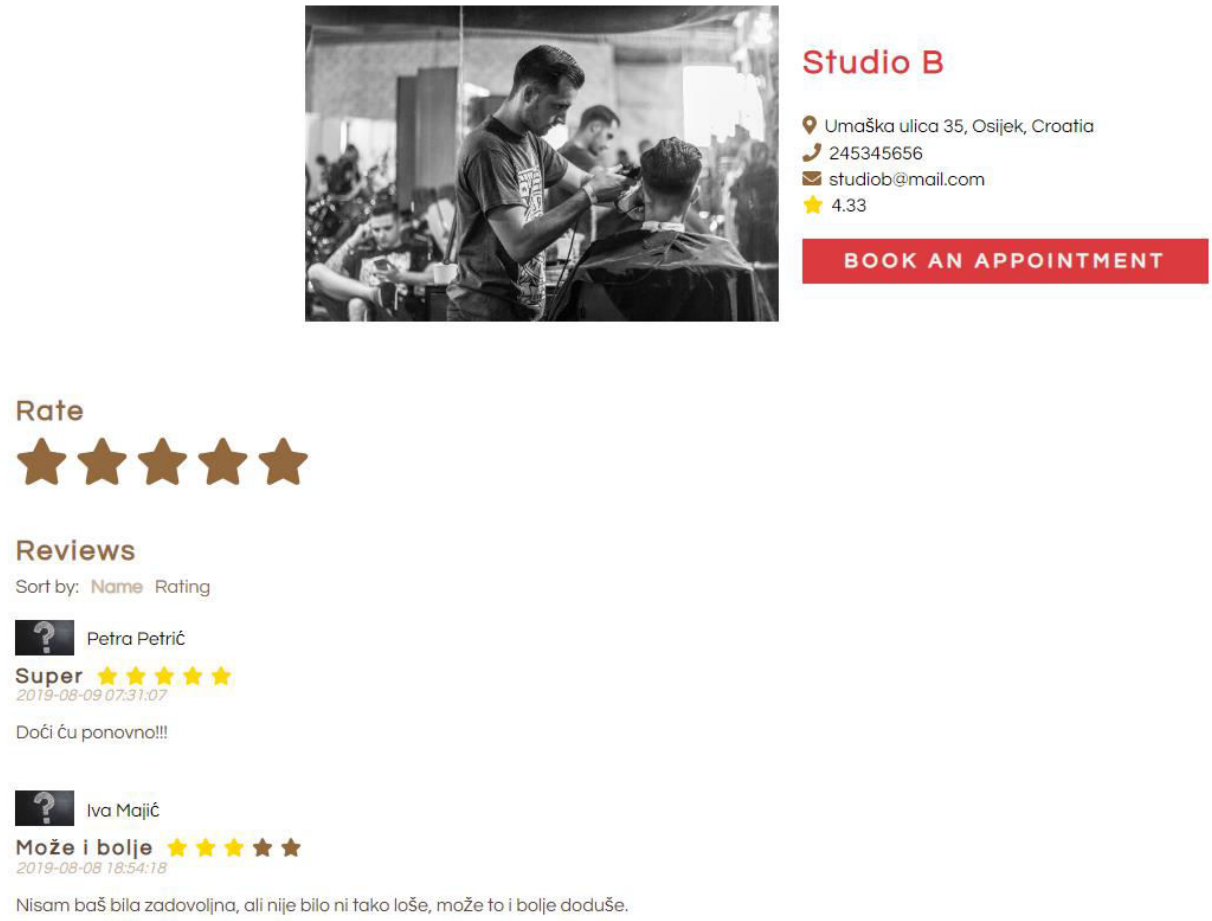
Kada korisnik klikne na gumb *Salons* na navigacijskoj traci, prikazuju mu se svi saloni iz baze podataka, što je prikazano na slici 3.9. Saloni su inicijalno poredani po udaljenosti kako bi se korisniku prvo ponudili njemu lokacijski najbliži saloni. Kako bi program znao poredati frizerske salone po lokaciji, potrebna mu je trenutna lokacija korisnika, što se postiže korištenjem funkcije web preglednika *navigator.geolocation.getCurrentPosition()*. Ta funkcija vraća koordinate (latitudu i longitudu) korisnikove trenutne lokacije i zatim se, budući da su u bazi podataka za svaki salon spremljene njegove koordinate, pomoću formule za udaljenost između dvije točke (formula (3-1)) računa udaljenost između korisnika i svakog salona te se saloni prema tom svojstvu sortiraju. Kako bi korisnikova lokacija mogla biti očitana, korisnik mora dopustiti aplikaciji pristup lokaciji putem iskočnog prozorčića (engl. *pop-up*) koji se automatski pojavljuje pri učitavanju komponente *SalonsComponent* budući da se koristi ugrađena funkcija *getCurrentPosition()*. Ukoliko korisnik odbije dati pristup lokaciji, saloni će biti sortirani abecednim redom.

$$d(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \qquad (3\text{-}1)$$



Slika 3.9. *Prikaz svih salona.*

Nakon što korisnik klikom odabere određeni salon, *router* ga preusmjerava na već spomenutu rutu */salon/:id*, gdje je *id* ID salona na koji je korisnik klliknuo. Time se učitava komponenta *SalonComponent* te se s poslužitelja dohvaćaju podaci o odabranom salonu koji se zatim korisniku prikazuju kao što je vidljivo na slici 3.10.



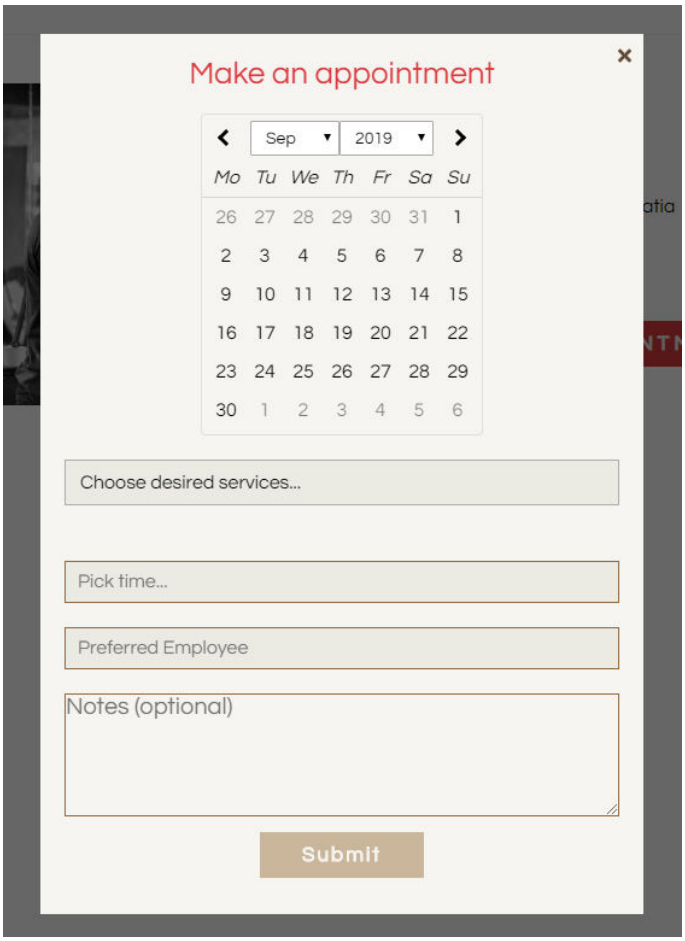
Slika 3.10. *Prikaz detalja o salonu.*

Kad korisnik klikne na gumb *Book an Appointment*, otvara mu se iskočni prozor prikazan na slici 3.11. Korisnik prvo mora odabrati željeni datum te mu se otvara mogućnost odabira željenih usluga (slika 3.12.). Kako svaka usluga nekog salona ima atribut *duration* koji označava procijenjeno trajanje te usluge, nakon što korisnik odabere sve željene usluge zbrajaju se ta vremena i time se računa ukupno vrijeme rezervacije *totalDuration.* Nakon što je odabrao željene usluge, korisnik može birati željeni termin (slika 3.13.). Ovo polje u početku je onemogućeno budući da se ovisno o željenom datumu i procijenjenom trajanju odabranih usluga računaju slobodni termini, i to funkcijom *calculateTime()* prikazanom u listingu 3.20. Funkcija *calculateTime()* prvo prolazi kroz sve rezervacije salona i filltrira samo one čiji je datum jednak odabranom datumu. Zatim *for* petljom prolazi kroz svako potencijalno vrijeme početka termina (bilo koje vrijeme od početka do kraja radnog vremena salona s razmakom od petnaest minuta – ako salon počinje raditi u 9:00, potencijalna vremena početka su 9:00, 9:15, 9:30, 9:45, 10:00,...) i ovisno o procijenjenom vremenu trajanja rezervacije *totalDuration* računa za svako potencijalno vrijeme početka njemu upareno potencijalno vrijeme završetka, s time da je posljednje potencijalno vrijeme početka upravo ono za koje potencijalno vrijeme završetka

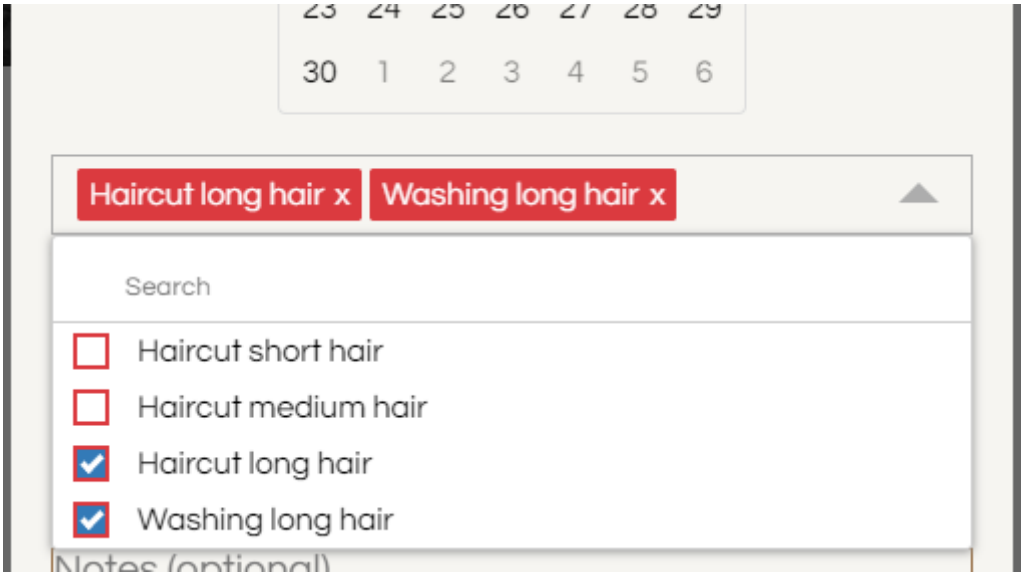
odgovara kraju radnog vremena salona. Dakle, ako salon radi do 20:00, a korisnik je odabrao uslugu koja traje 45 minuta, posljednje potencijalno vrijeme početka neće biti 19:45, nego 19:15 kako bi usluga stigla biti obavljena do završetka radnog vremena. Nakon toga funkcija provjerava ima li u vremenskom periodu između potencijalnog početka i završetka usluge već neka rezervacija (iz filtriranog polja s početka funkcije) ili je termin slobodan. Ako je termin slobodan, dodaje se u polje sa slobodnim terminima i prikazuje korisniku (slika 3.13.) te ga isti može odabrati.

```
  public calculateTime(): void {
    this.freeTerms = [];
    const dateReservations = this.salonReservations.filter(reservation =>
reservation.startTime.year() === this.selectedDate.year &&
reservation.startTime.month() + 1 === this.selectedDate.month &&
reservation.startTime.date() === this.selectedDate.day);
    for (let i = this.salon.workHoursStart; i <= this.salon.workHoursEnd -
this.totalDuration / 60; i += 0.25) {
      const potentialEnd = i + this.totalDuration / 60;
      let invalidFlag = false;
      dateReservations.forEach(reservation => {
        if (i < reservation.endTime.hour() && potentialEnd >
reservation.startTime.hour()) {
          invalidFlag = true;
        }
      });
      if (!invalidFlag) {
        const wholeStart = Math.floor(i);
        const decimalStart = (i - Math.floor(i)) * 60 === 0 ? "00" :
Math.round((i - Math.floor(i)) * 60);
        const wholeEnd = Math.floor(potentialEnd);
        const decimalEnd = (potentialEnd - Math.floor(potentialEnd)) * 60 ===
0 ? "00" : Math.round((potentialEnd - Math.floor(potentialEnd)) * 60);
        const term = {
          start: i,
          end: potentialEnd,
          display: wholeStart + ":" + decimalStart + " - " + wholeEnd + ":" +
decimalEnd,
        }
        this.freeTerms.push(term);
      }
    }
  }
```

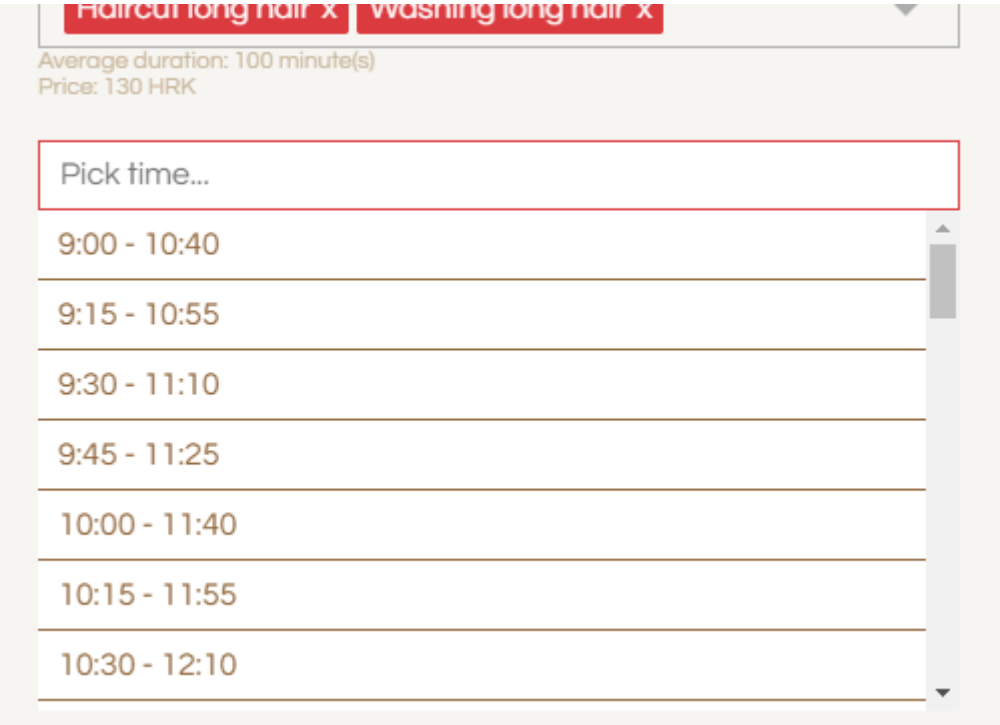
Listing 3.22. *Funkcija za određivanje slobodnih termina.*



Slika 3.11. *Iskočni prozor za kreiranje rezervacije.*



Slika 3.12. *Odabir željenih usluga pri rezervaciji termina.*



Slika 3.13. *Odabir željenog termina pri rezervaciji termina.*

Nakon što je odabrao željeni termin, korisnik može odabrati željenog zaposlenika koji će biti zadužen za uslugu i po želji dodati nekakav komentar, odnosno napomenu. Klikom na tipku *Submit* poziva se funkcija *createReservation()* koja poziva istoimenu funkciju unutar servisa za rezervacije, koja pomoću za to predviđene krajnje točke šalje *POST* zahtjev na poslužitelj kako bi se rezervacija spremila u bazu podataka.

3.4.5. Pregled arhive rezervacija

U osnovnim zahtjevima navedeno je da korisnik mora moći vidjeti svoju arhivu rezervacija. To je postignuto dodavanjem rute *profile* na kojoj korisnik može vidjeti sve svoje rezervacije te, ukoliko je do početka termina ostalo više od dva sata, iste po potrebi ukloniti. Arhiva rezervacija jednog korisnika prikazana je na isti način kao što je to postignuto za upravljanje rezervacijama nekog salona. Jedina razlika je što se vremenski intervali ne računaju od početka do kraja radnog vremena salona, nego u sklopu jednog dana, dakle od 0 do 24 sata.

4. PLANIRANA UNAPRJEĐENJA APLIKACIJE

Aplikacija za rezerviranje termina i usluga u frizerskim salonima ispunjava sve zahtjeve zadane u sklopu ovog diplomskog rada, no svejedno se može dodatno unaprijediti u aspektima koji nisu obuhvaćeni u vremenskom okviru stvaranja diplomskog rada.

Primjerice, trenutni sustav rezervacija termina podrazumijeva da u jednom vremenskom terminu može postojati samo jedna rezervacija, što nije uvijek slučaj jer u nekim frizerskim salonima u određenim radnim satima radi više osoba istovremeno. Poslužiteljski kôd i baza podataka morali bi se promijeniti kako bi se prilagodili tome i kako bi svaki salon mogao unijeti točan broj slobodnih zaposlenika u svaki svoj radni sat. Također vezano uz radno vrijeme i slobodne termine, u ovoj verziji aplikacije postoje samo radni sati – u koje vrijeme salon počinje raditi i kad završava s radom – ne uzimaju se u obzir stanke, vikendi i praznici ili drugačija radna vremena kroz tjedan (ukoliko salon, primjerice, radi ponedjeljkom, srijedom i petkom prijepodne, a utorkom i četvrtkom poslijepodne). U budućim verzijama aplikacije to se također može unaprijediti promjenama poslužiteljskog kôda i baze podataka tako da svaki administator pri kreiranju salona može unijeti u koje točno sate tijekom tjedna salon radi. Za praznike i blagdane može se dodati upit administratoru salona da za nadolazeći praznik/blagdan odabere radi li salon (i u koje radno vrijeme) na taj dan pa se budući termini rezervacija računaju po tome. Također je moguće dodati opciju da administrator po potrebi može odrediti da na određeni dan salon ne radi (u slučaju bolovanja, godišnjeg odmora, renovacija ili bilo kojeg drugog razloga).

Osim toga, kako aplikacija podrazumijeva suradnju s frizerskim salonima, tako je planirana i suradnja s različitim markama za proizvode za kosu. To bi podrazumijevalo novu kategoriju aplikacije – promocije – u kojoj bi korisnik mogao vidjeti koje mu se promocije nude te ih, poput kupona iskoristiti za kupovinu proizvoda s popustom. Svaka promocija imala bi svoje trajanje pa bi tako u sklopu neke promocije npr. svi korisnici aplikacije tijekom listopada imali popust od 10% na sve šampone za kosu marke s kojom je suradnja ostvarena. Ovu opciju jednostavno je implementirati u postojeću aplikaciju, no za to je prvo potrebno dogovoriti suradnje s markama, za što je preduvjet širenje aplikacije, odnosno da se ona počne koristiti u većem broju frizerskih salona.

5. ZAKLJUČAK

Izrada aplikacije za rezerviranje termina u frizerskim salonima zahtijevala je detaljno upoznavanje s tehnologijama koje će se za izradu koristiti. Za razvoj aplikacije odabrana su dva *frameworka* – Laravel (PHP) za poslužiteljski kôd te Angular (TypeScript) za korisničko sučelje. Oba *frameworka* imala su prednost nad drugim rješenjima zbog svoje jednostavnosti, bogate dokumentacije, velikog broja dodatnih biblioteka i široke upotrijebljenosti te su se pokazala kao idealan odabir za razvoj ove aplikacije – svaki aspekt aplikacije mogao se bez problema realizirati koristeći ta dva *frameworka* i njihove biblioteke.

Aplikacija za rezerviranje termina u frizerskim salonima pruža korisnicima informacije o obližnjim (ali i ostalim) frizerskim salonima te im omogućuje da po ocjenama i recenzijama prepoznaju koji salon će biti najbolji za njihove potrebe te za taj salon rezerviraju termin. Iako postoje nedostatci pri rezervaciji termina koji nisu mogli biti obuhvaćeni u vremenskom opsegu ovog diplomskog rada, kao što su nemogućnost različitih radnih vremena tijekom tjedna ili neosjetljivost na blagdane i praznike, aplikacija obavlja ono za što je namijenjena, a to je jednostavno i brzo rezerviranje termina, bez podizanja telefonske slušalice i dugačkih dogovora, dok se nedostatci vrlo lako mogu nadoknaditi u nekoj od sljedećih inačica aplikacije. Osim što korisnici mogu rezervirati termin u salonu, svaki korisnik, ukoliko je vlasnik salona, može otvoriti i „profil“ za svoj salon, tj. dodati svoj salon u bazu podataka kako bi korisnici mogli i za njega rezervirati termin. Takav korisnik postaje administrator salona te može vidjeti kalendar rezervacija za svoj salon, odnosno koji korisnik je rezervirao koji termin za koje usluge te tako upravljati salonom bez bilježnica i zapisivanja termina. U skladu s time svaki korisnik ima svoju arhivu korištenja usluga gdje može vidjeti svoje prethodne rezervacije, ali i one buduće kako bi se podsjetio na budući termin ukoliko zaboravi koje vrijeme je dogovorio.

Uzimajući u obzir sve navedeno, može se zaključiti kako aplikacija za rezerviranje termina u frizerskim salonima funkcionira na razini planiranoj u sklopu ovog diplomskog rada i kako su svi zahtjevi zadani u zadatku diplomskog rada uspješno implementirani. Kao što je napomenuto, postoje nedostaci u radu aplikacije koji su bili izvan opsega ovog diplomskog rada, no isti mogu biti nadoknađeni u nekoj od sljedećih inačica aplikacije. Trenutna verzija aplikacije (obuhvaćena ovim diplomskim radom) potpuno je funkcionalna te se može početi koristiti u frizerskim salonima čiji su voditelji zainteresirani za njezino korištenje.

LITERATURA

[1] W3, HTML, https://www.w3.org/TR/REC-html40-971218/conform.html#deprecated (pristupljeno 29.6.2019.)

[2] W3, HTML5, https://www.w3.org/TR/2008/WD-html5-20080122/ (pristupljeno 29.6.2019.)

[3] CreativeBloq, What is SASS, https://www.creativebloq.com/web-design/what-is-sass-111517618 (pristupljeno 29.6.2019.)

[4] Sass, Syntax, https://sass-lang.com/documentation/syntax (pristupljeno 29.6.2019.)

[5] Reenskaug, Trygve; Coplien, James O.: "The DCI Architecture: A New Vision of Object-Oriented Programming"

[6] https://oauth.net/2/ (pristupljeno 3.9.2019.)

SAŽETAK

Tema ovog diplomskog rada izrada je aplikacije za rezerviranje termina i usluga u frizerskim salonima. Aplikacija je izrađena korištenjem *frameworka* Laravel (PHP) za poslužiteljski kôd i Angular (TypeScript, HTML5, SCSS) za korisničko sučelje. Aplikacija pruža korisniku informacije o njemu najbližim (ali i ostalim) frizerskim salonima te mu pomoću sustava recenzija i ocjena omogućuje da odabere onaj koji najbolje odgovara njegovim potreba, nakon čega u tom salonu može rezervirati svoj termin. Aplikacija također omogućuje korisniku koji je koristi da doda svoj salon u bazu podataka kako bi drugi korisnici mogli u njemu rezervirati termin, pri čemu taj korisnik može na jednostavan način upravljati rezervacijama za svoj salon. Osim toga, svaki korisnik ima svoju arhivu korištenja usluga gdje može vidjeti prethodne rezervacije koje je napravio u sustavu, ali i one buduće kako bi se mogao podsjetiti koji termin je rezervirao i po potrebi buduću rezervaciju otkazati.

Ključne riječi: Angular, Laravel, TypeScript, PHP, HTML5, SCSS, web aplikacija, frizerski salon, rezervacija termina

ABSTRACT

Application for booking appointments and services in hair salons

The topic of this Master's Thesis is the making of an application for booking appointments and services in hair salons. The application was created using Laravel framework (PHP) for the back-end code and Angular (TypeScript, HTML5, SCSS) for the front-end code. This app provides the user with information about the closest hair salons (and others as well) and, using the system of reviews and ratings, allows them to choose the one that best suits their needs, after which they can book the appointment in that salon. The app also allows the user to add their salon to the database so that other users can book an appointment there and said user can easily manage bookings for their salon. In addition, each user has their own archive of services where they can see the previous bookings they made through the app, as well as the future ones, so that they can be reminded what time of the reservation they have booked or cancel the reservation if necessary.

Keywords: Angular, Laravel, TypeScript, PHP, HTML5, SCSS, web app, hair salon, booking

ŽIVOTOPIS

Iva Majić rođena je 6. lipnja 1996. godine u Đakovu, Hrvatska. Pohađa Osnovnu školu Vladimira Nazora u Đakovu do 2010., kad upisuje općí smjer Gimnazije Antuna Gustava Matoša Đakovo. Sva četiri razreda srednje škole završava s odličnim uspjehom te 2014. godine upisuje preddiplomski studij računarstva na tadašnjem Elektrotehničkom Fakultetu Osijek, danas Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. 2017. godine završava preddiplomski studij s vrlo dobrim uspjehom te upisuje diplomski studij računarstva, smjer programsko inženjerstvo, također na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

Ima izvrsnu razinu znanja opisnog jezika HTML, programskog jezika JavaScript i TypeScript te *frameworka* Angular i React. Ima srednju razinu znanja programskog jezika PHP i *frameworka* Laravel. Također ima i srednju razinu znanja programskih jezika C, C++ i Java. Osim toga, izvrsno se služi engleskim jezikom, Microsoft Office alatima te Adobe paketom (Premiere Pro, After Effects i Photoshop).

Zahvaljujući vrlo dobrom uspjehu tijekom studiranja, dobitnica je stipendije za deficitarna zanimanja za potporu učeničkom i studentskom standardu za svih pet godina studiranja. Prvo radno iskustvo stekla je kao agentica u teleprodajnim centrima za Studio Moderna d.o.o. u Osijeku i Hrvatski Telekom d.d. također u Osijeku. Nakon toga, u svibnju 2018. godine odrađuje praksu te se zapošljava kao front-end developer u tvrtci EM2 d.o.o. u Osijeku gdje ostaje do ožujka 2019. godine kad se zapošljava u tvrtci Kod Savjetovanje d.o.o. u Vukovaru, također kao front-end developer.

Iva Majić