# **Veb aplikacija za rezervaciju termina u restoranima**

**Profesor:** prof. dr Vladislav Mišković

**Asistent:** Milan Tair

**Student:**

Sadržaj

[**Veb aplikacija za rezervaciju termina u restoranima** 1](#_Toc54357994)

[1. Uvod 3](#_Toc54357995)

[1.1 Cilj razvoja 3](#_Toc54357996)

[1.2 Obim sistema 3](#_Toc54357997)

[1.3 Prikaz proizvoda 3](#_Toc54357998)

[1.3.1 Perspektiva proizvoda 4](#_Toc54357999)

[1.3.2 Funkcije proizvoda 4](#_Toc54358000)

[1.3.3 Karakteristike korisnika 5](#_Toc54358001)

[1.3.4 Ograničenja 6](#_Toc54358002)

[1.3.5 Definicije 6](#_Toc54358003)

[2. Reference 6](#_Toc54358004)

[3. Specifikacija zahteva 7](#_Toc54358005)

[3.1 Spoljašnji interfejsi 7](#_Toc54358006)

[3.1.1 Menadžeri 7](#_Toc54358007)

[3.1.2 Neprijavljeni korisnici 7](#_Toc54358008)

[3.2 Elementi organizacije aplikacije - MVC 9](#_Toc54358009)

[3.3 Pogodnost za upotrebu 10](#_Toc54358010)

[3.4 Zahtevane performanse 10](#_Toc54358011)

[3.5 Zahtevi baze podataka 10](#_Toc54358012)

[3.6. Spisak svih kontrolera I njihovih metoda 14](#_Toc54358013)

[3.7 Projektna ograničenja 15](#_Toc54358014)

[4. Realizacija sitema 16](#_Toc54358015)

[4.1 Detalji restorana – vreme otvaranja/zatvaranja 16](#_Toc54358016)

[Detalji restorana – vreme otvaranja/zatvaranja 16](#_Toc54358017)

[4.2 Provera slobodnih stolova pri izvršavanju rezervacija 19](#_Toc54358018)

# Uvod

Prateća dokumentacija detaljno opisuje cilj razvoja same aplikacije, obim sistema, prikaz okruženja u kojem sistem funkcioniše, način na koji sistem funkcioniše. Takođe su prikazani određeni elementi same aplkacije, dijagrami, model baze, popis svih kontrolera i njihovih metoda koji obavljaju potrebnu programsku i poslovnu logiku sa ciljem izvršavanja svih predviđenih aktivnosti aplikacije, a nevedena su i određena ograničenja koja je neophodno ispoštovati.

## Cilj razvoja

Cilj razvoja ovog sistema jeste da se stvori pouzdana, brza i jednostavna veb aplikacija namenjena vlasnicima restorana koji bi je ustupali na korišćenje svojim menadžerima za potrebe održavanja restorana, kao I svojim gostima za potrebe rezervisanja termina.

## Obim sistema

Sistem omogućava menadžerima restorana da izvršavaju kreiranje novih restorana, dodavanje stolova, radnog vremena, kao i neradnih dana za svoje restorena. Takođe mogu pregledati pristigle rezervacije i označavati ih kao prihvaćene ili odbijene.

Neprijavljeni korisnici mogu pregledati spisak svih restorana(moguće filtriranje po gradovima) i izvršavati rezervacije za izabrane restorane.

Realizacijom ovog sistema obuhvaćen je skup funkcionalnost potrebnih za osnovno funkcionisanje aplikacije za rezervaciju restorana.

## Prikaz proizvoda

Naziv ovog proizvoda je “Veb aplikacija za rezervaciju termina u restoranima”.

Osnovne funkcije ovog sistema čine:

* Administracija restorana i rezervacija od strane njihovih menadžera
* Pregled restorana i izvršavanje rezervacija u odabranom restoranu

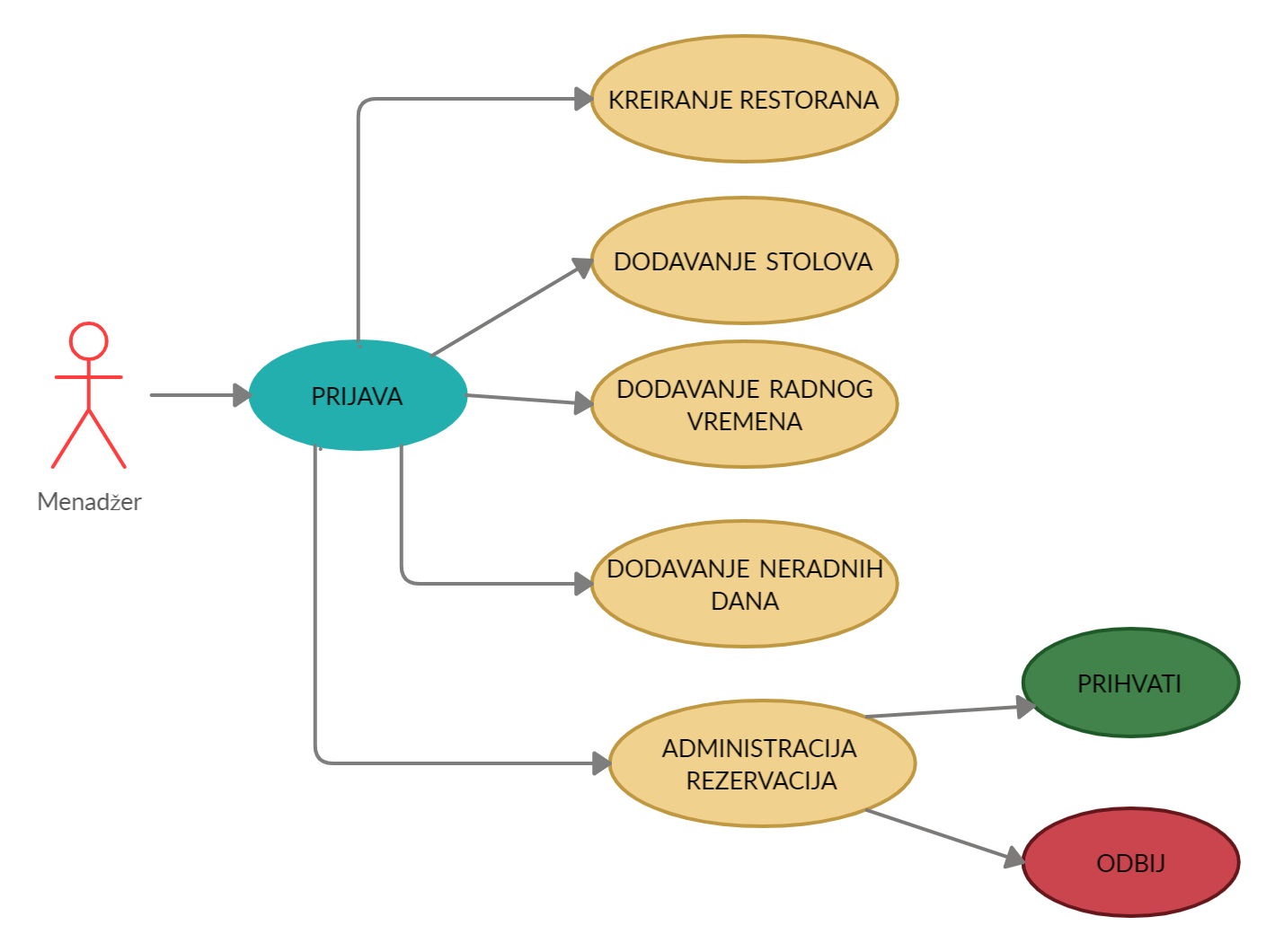
Cilj realizovanog Sistema je da se, na bezbedan i efikasan način, stvori jedinstvena veb aplikacija svih restorana. Da se omogući jednostavan I pre svega intuitivan korisnički interfejs - *GUI,* koji će biti dostupan svim korisnicima bez potrebe za bilo kakvim predznanjem ili iskustvom u radu sa ovakvim vrstama sistema. Takođe, cilj je kreirati aplikaciju koja se može sa podjednakom lakoćom koristiti na različitim uređajima sa različitim dimenzijama ekrana (*responsiveness*).

### Perspektiva proizvoda

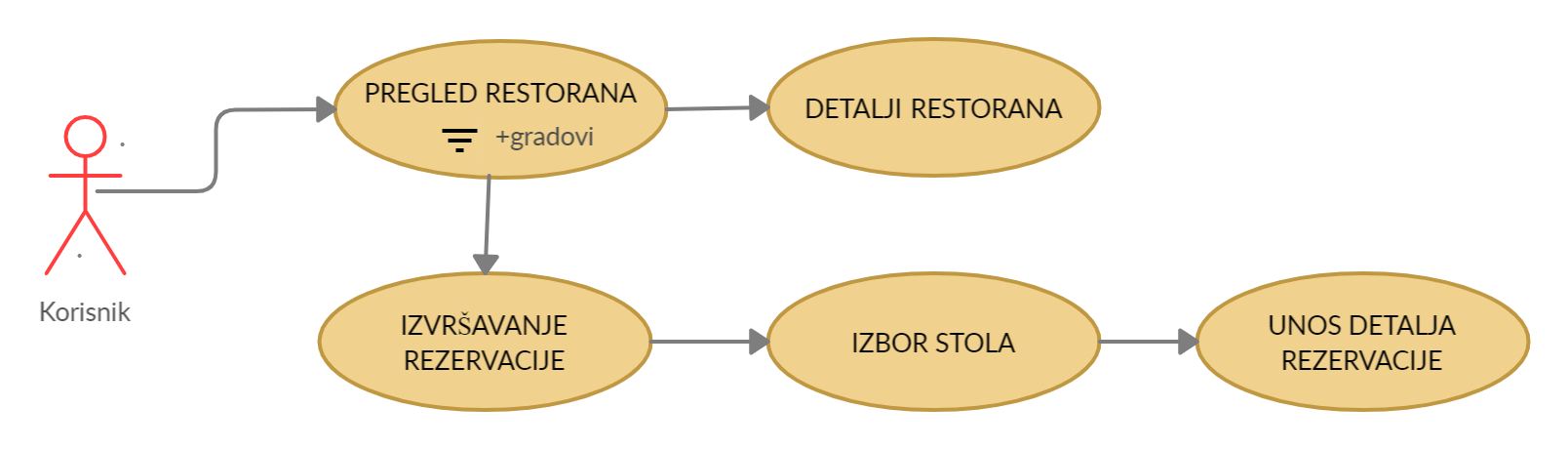
Realizovani sistem je demo verzija sistema koji bi se u budućnosti mogao kreirati i kao takav trenutno funkcioniše potpuno nezavisno.

Sistem je realizovan tako da bi u slučaju puštanja u produkciju bilo lako povezati gas a drugim sistemima. Primera radi, sistem bi se mogao povezati sa zvaničnim sajtovima restorana koji se pregledaju i mogao bi se omogućiti link ka tim sajtovima prilikom pregledanja restorana. Takođe, rezervacije bi se mogle slati i direktno restoranima na njihove sisteme koji bi te rezervacije obrađivali I skladištili u svojim bazama podataka.

### Funkcije proizvoda



*Slika 1.3.2.1, use-case dijagram menadžera*

**

*Slika 1.3.2.2, use-case dijagram korisnika*

Postoje dve vrste korisnika sistema: menadžeri i neprijavljeni korinici.

Funkcije koje mogu obavljati menadžeri su:

* Kreiranje novog restorana
* Dodavanje stolova za restoran
* Dodavanje radnog vremena restorana
* Dodavanje neradnih dana restorana
* Administracija pristiglih rezervacija za restoran
  + Potvrda rezervacije
  + Odbijanje rezervacije

Funkcije koje mogu obavljati neprijavljeni korisnici su:

* Pregled restorana
  + Filter po gradovima
* Detalji izabranog restorana
* Izvršavanje rezervacije

### 1.3.3 Karakteristike korisnika

Kao što je prethodno navedeno, postoje dve vrste korisnika sistema: menadžeri i neprijavljeni korinici.

Mendžeri su zaposleni od strane restorana i njihov posao je administracija detalja i rezervacija restorana. Moraju biti prijavljeni na sistem prilikom korišćenja aplikacije. Za svaki restoran postoji tačno jedan menadžer. Neprijavljeni korisnici mogu pregledati restorane i izvršavati rezervacije bez prijave na sistem. Interfejs je dizajniran da bude lak za korišćenje i nije neophodno nikakvo prethodno iskustvo ili obrazovanje za njegovo korišćenje.

### 1.3.4 Ograničenja

Jedino što je korisniku aplikacije potrebno jeste pristup Internetu, dok je menadžerima potrebna registracija na aplikaciji.

## 1.3.5 Definicije

Definicije manje poznatih pojmova i skraćenica:

1. *Browser* – veb pregledač (*Google Chrome, Mozzilla Firefox*…)
2. *Client / FrontEnd* – klijentska strana, deo veb aplikacije koji se prikazuje korisniku u *browser*-u
3. *Server / BackEnd –* serverska strana veb aplikacije zadužena za komunikaciju sa bazom podataka I dopremanje podataka, koji se prikazuju na klijentskoj strain
4. *HTTP Request* – zahtev za podacima, osnovni element komunikacije između klijenta i servera
5. *GUI* , “*Graphic user interface” –* grafički korisnički interfejs
6. *Responsiveness* – prilagodljivost prikaza sadržaja različitim dimenzijama ekrana
7. *ORM, “Object relational mapper”* – objektno relacioni mapper, koristan alat pri modeliranju entiteta I relacija baze podataka
8. *MVC, Model-View-Controller* – arhitektura organizacije koda koja se zasniva na 3 celine sa uočljivo razdvojenim ulogama
9. *Separation of concerns* – podela odgovornosti

# 2. Reference

[1] <https://docs.nestjs.com/> dokumentacija rayvojnog okvira *NestJS*

[2] <https://typeorm.io/#/> dokumentacija *Type* *ORM*-a

[3] <https://reactjs.org/> dokumentacija *React* biblioteke

[4] <https://react-bootstrap.github.io/> dokumentacija *ReactBootstrap* biblioteke

[5] <https://getbootstrap.com/> *Bootstrap* dokumentacija

[6] <https://jwt.io/> *JSON Web Token* dokumentacija

[7] <https://github.com/MilanTairYouTube/React2020> *GitHub React* projekat, Milan Tair

[8] <https://developer.mozilla.org/en-US/docs/Web/JavaScript> *JavaScript* dokumentacija

[9] <https://www.w3schools.com/> *JavaScript/CSS/HTML/NodeJS* dokumentacija

[10] <https://www.stackoverflow.com/> *StackOverflow* sajt

[11] <https://www.restapitutorial.com/httpstatuscodes.html> *HTTP* status kodovi

[12] <https://www.youtube.com/channel/UCoabUK8jyHYwuxY3DFIbWcQ> *Youtube* kanal, Milan Tair

[13] https://www.npmjs.com/package/bcrypt *bcrypt* dokumentacija

# Specifikacija zahteva

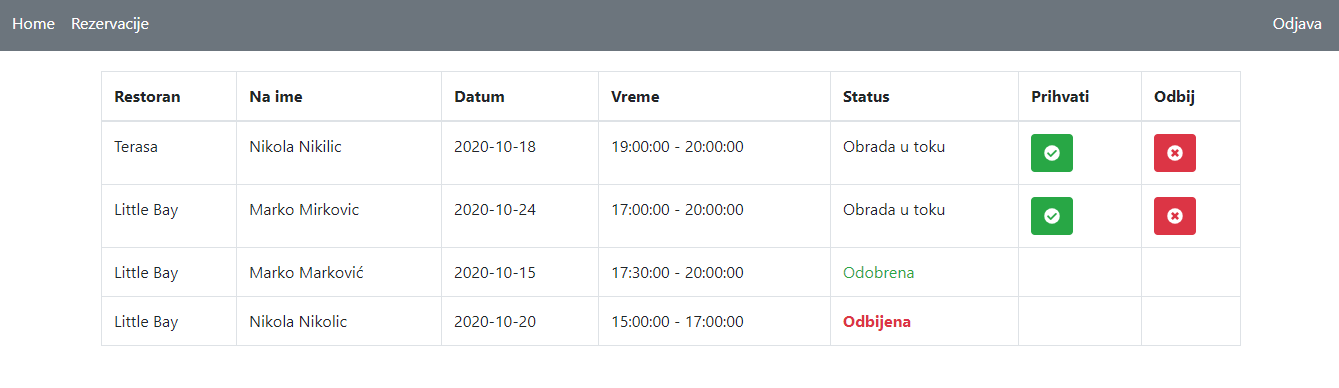
## 3.1 Spoljašnji interfejsi

Početni ekran menadžerima nudi opciju prijave ili registraije na sistem, dok se korisnicima aplikacije u svrhe rezervisanja stolova nudi opcija pregledanja svih restorana I izvršavanja rezervacija u njima. U ovom potpoglavlju prikazani su neki od izgleda stranica koje se prikazuju menadžerima i neprijavljenim korisnicima.

### 3.1.1 Menadžeri

Nakon uspešne prijave, menadžeru ima nekoliko opcija: dodavanje restorana, dodavanje stolova za restoran, dodavanje radnog vremena restorana i neradnih dana, kao i pregled i administracija svih rezervacija za svoj restoran.

Pri dodavanju restorana, menadžer unosi ime, adresu, opis i grad u kojem se restoran nalazi(izbor iz padajuće liste). Dodavanje radnog vremena podrazumeva upisivanje otvaranja i zatvaranja restorana za svaki radni dan u nedelji. Kod dodavanja stolova za restoran, unosi se broj stola, kapacitet, maksimalno trajanje rezervacije i opis stola (izbor iz padajuće liste: separe, na sred sale…). Na ekran sa rezervacijama, menadžeru je osim pregleda svih rezervacija, omogućeno da novopristigle rezervacije prihvata ili odbija (Slika 3.1.1.1).



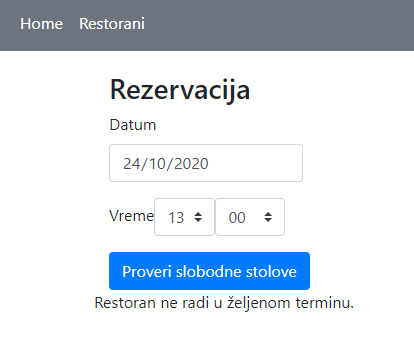
*Slika 3.1.1.1 Administracija rezervacija*

### 3.1.2 Neprijavljeni korisnici

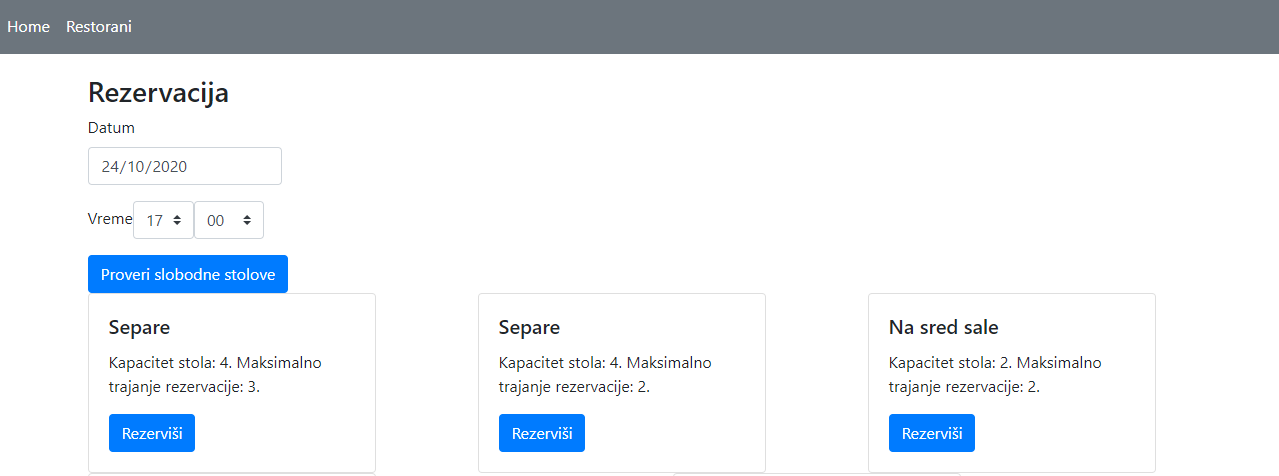
Neprijavljeni korisnici imaju opciju pregledanja svih restorana, uz opciju filtriranja restorana po gradovima (Slika 3.1.2.1). Za svaki restoran mogu pregladti stranicu sa detaljima restorana, kao i da izvršavaju rezervacije. Pri izvršavanju rezervacije, potrebno je uneti datum i vreme, a kao rezultat pretrage korisniku se prikazuju raspoloživi stolovi u datom trenutku, kao i detalji o maksimalnom trajanju rezervacije i opisu za svaki sto (Slika 3.1.2.2 i 3.1.2.3). Korisniku se klikom na dugme za rezervaciju prikazuje ekran sa formom sa ličnim detaljima (Slika 3.1.2.3), čijom se potvrdom rezervacija prosleđuje menadžeru, koji treba da je odobri ili odbije (Slika 3.1.2.4).



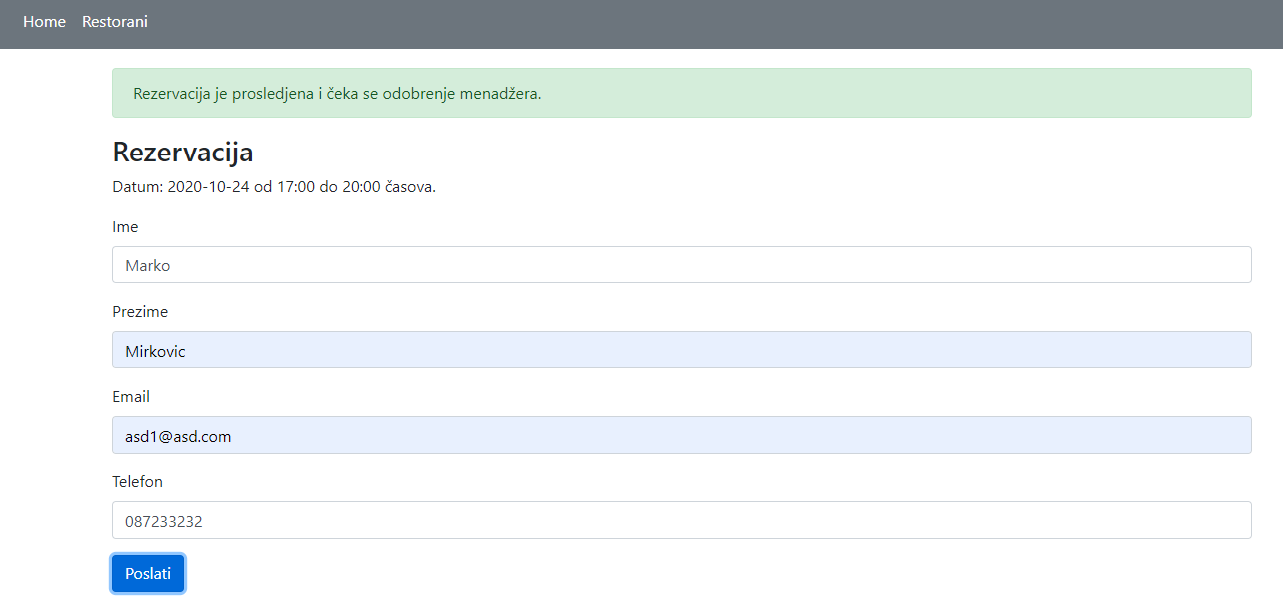
*Slika 3.1.2.1Pregled restorana po gradovima*



*Slika 3.1.2.2 Primer kada restoran ne radi u željenom terminu rezervacije*

**

*Slika 3.1.2.3 Odabir stolova koji su slobodni u željenom trenutku rezeervacije*

**

*Slika 3.1.2.4 Potvrda rezervacije i čekanje na odobrenje od strane menadžera*

## 3.2 Elementi organizacije aplikacije - MVC

Serverski deo aplikacije realizovan je *Node*-ovom razvojnom okviru, *NestJS*, dok je klijentski deo realizovan pomoću *React* rayvojnog okvira za *JavaScript*. Na klijentskoj strani se servira statički deo stranice koji dinamički doprema podatke sa servera pomoću asinhronih funkcija i ažurira stranicu koja se prikazuje korisniku. Za asinhrone pozive ka serveru, koji se odvijaju u pozadini i dopremaju podatke, korišćena je biblioteka *axios.* Ovo je realizovano pomoću *HTTP* zahteva (*eng. request*) koji se šalju na definisanu rutu predviđenu za dohvatanje potrebnih podataka. Za prijem I obradu ovih zahteva potreban je serverski ruter (*NestJS* ruter). Svaka predefinisana rutu na serveru je dodeljena različitim kontrolerima, koji pomoću svojih servisa dohvataju podatke iz baze. Komunikacija sa bazom je ostvarena pomoću modela (*Entity*-ja). Ovakvi podaci se potom prosleđuju na klijentsku stranu u sirovom obliku, gde se na osnovu neke definisane logike prikazuju korisniku na veb stranici. Na ovaj način, jasno je razdvojena uloga svakog od delova naše aplikacije (*eng. separation of concerns).* Prema tome, moglo bi se reći da na serveru postoje kontroler (*NestJS* ruter koji definiše sve postojeće rute) i model (baza podataka kojoj se pristupa putem servisa koji komuniciraju sa kontrolerom), dok na klijentskoj strani postoje komponente koje su ujedno i kontroleri (kontrolišu dohvatanje podataka sa servera) i *view*-ovi (formiraju HTML kod zadužen za prikaz dobijenih podataka sa servera).

## Pogodnost za upotrebu

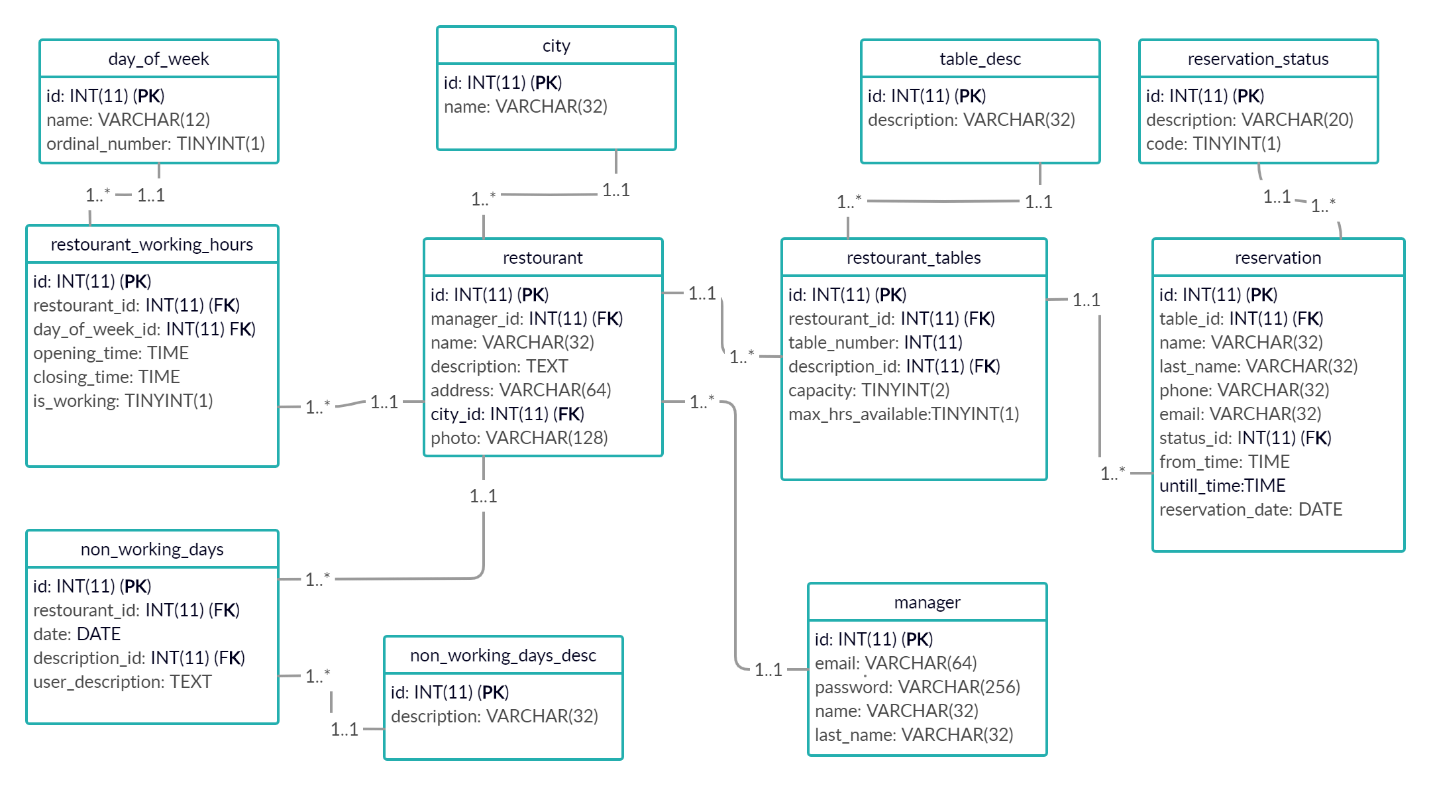
Aplikacija zadovoljava efektivnost i efkasnost korišćenja i predstavlja sistem koji pozitivno utiče na zadovoljstvo krajnjih korisnika.

## 3.4 Zahtevane performanse

Vreme odziva Web aplikacije je minimalno, a broj istovremenih korisnika ne remeti rad same aplikacije i ne predstavlja problem samom sistemu niti usporava njegov rad.

## 3.5 Zahtevi baze podataka

Za realizaciju ove veb aplikacije korišćenja je MySQL relaciona baza podataka. Na Slici 3.5.1. prikazan je model baze podataka sa svim relacijama među tabelama.



*Slika 3.5. model baze podataka*

U nastavku teksta dat je tabelarni prikaz svih tabela iz baze podataka, sa opisom svih polja, njihovih tipova, kao i ograničenja.

* manager

|  |  |  |
| --- | --- | --- |
| **Naziv kolone** | **Tip podataka** | **Ograničenja** |
| id | int(11) | PK, AI |
| email | varchar(64) | UNIQUE |
| password | varchar(250) |  |
| name | varchar(32) |  |
| password | varchar(32) |  |

* city

|  |  |  |
| --- | --- | --- |
| **Naziv kolone** | **Tip podataka** | **Ograničenja** |
| id | int(11) | PK, AI |
| name | varchar(64) | UNIQUE |

* restourant

|  |  |  |
| --- | --- | --- |
| **Naziv kolone** | **Tip podataka** | **Ograničenja** |
| id | int(11) | PK, AI |
| manager\_id | int(11) | FK references manager (id) |
| name | varchar(32) | UNIQUE |
| description | text | NULL |
| address | varchar(64) |  |
| city\_id | int(11) | FK references city (id) |
| photo | varchar(128) | NULL |

* day\_of\_week

|  |  |  |
| --- | --- | --- |
| **Naziv kolone** | **Tip podataka** | **Ograničenja** |
| id | int(11) | PK, AI |
| name | varchar(32) | UNIQUE |
| Ordinal\_number | tinyint(1) | UNIQUE |

* table\_desc

|  |  |  |
| --- | --- | --- |
| **Naziv kolone** | **Tip podataka** | **Ograničenja** |
| id | int(11) | PK, AI |
| description | varchar(32) | UNIQUE |

* restourant\_tables

|  |  |  |
| --- | --- | --- |
| **Naziv kolone** | **Tip podataka** | **Ograničenja** |
| id | int(11) | PK, AI |
| restaurant\_id | int(11) | FK references restourant (id) |
| table\_number | int(11) | UNIQUE |
| description\_id | int(11) | FK references table\_desc (id) |
| capacity | tinyint(2) |  |
| max\_hours\_available | tinyint(1) |  |

* restourant\_working\_hours

|  |  |  |
| --- | --- | --- |
| **Naziv kolone** | **Tip podataka** | **Ograničenja** |
| id | int(11) | PK, AI |
| restaurant\_id | int(11) | FK references restourant (id) |
| day\_of\_week\_id | int(11) | FK references day\_of\_week (id) |
| opening\_time | time | FK references table\_desc (id) |
| closing\_time | time |  |
| is\_working | tinyint(1) | DEFAULT: 1 |

* non\_working\_days\_desc

|  |  |  |
| --- | --- | --- |
| **Naziv kolone** | **Tip podataka** | **Ograničenja** |
| id | int(11) | PK, AI |
| description | varchar(32) | UNIQUE |

* non\_working\_days

|  |  |  |
| --- | --- | --- |
| **Naziv kolone** | **Tip podataka** | **Ograničenja** |
| id | int(11) | PK, AI |
| restaurant\_id | int(11) | FK references restourant (id) |
| date | int(11) | FK references car\_make (cm\_id) |
| description\_id | int(11) | FK references non\_working\_days\_desc (id) |
| user\_description | text |  |

* reservation\_status

|  |  |  |
| --- | --- | --- |
| **Naziv kolone** | **Tip podataka** | **Ograničenja** |
| id | int(11) | PK, AI |
| description | varchar(20) | UNIQUE |
| code | tinyint(1) | UNIQUE |

* reservation

|  |  |  |
| --- | --- | --- |
| **Naziv kolone** | **Tip podataka** | **Ograničenja** |
| id | int(11) | PK, AI |
| table\_id | tinyint(1) | FK references restaurant\_tables (id) |
| name | varchar(32) |  |
| last\_name | varchar(32) |  |
| phone | varchar(20) |  |
| email | varchar(32) |  |
| status\_id | tinyint(1) | FK references reservation\_status(id) |
| from\_time | time |  |
| until\_time | time |  |
| reservation\_date | date |  |

## 3.6. Spisak svih kontrolera I njihovih metoda

U ovom potpoglavlju prikazan je spisak svih kontrolera i njihovih metoda koji obavljaju potrebnu programsku logiku sa ciljem izvršavanja svih predviđenih aktivnosti aplikacije. U nastavku su tabelarno prikazani svi serverski kontroleri ruta, realizovani u *NestJS* razvojnom okviru.

* utility.controller.ts

|  |  |  |
| --- | --- | --- |
| **Ruta**  utility/ | **HTTP Metod** | **Metoda kontrolera** |
| /cities | GET | findAllCities() |
| /days-of-week-desc | GET | findAllDaysOfWeekDesc() |
| /non-working-days-desc | GET | findAllNonWorkingDaysDesc() |
| /reservation-status-desc | GET | findAllReservationStatusDesc() |
| /table-desc | GET | findAllTableDesc() |

* restourant.controller.ts

|  |  |  |
| --- | --- | --- |
| **Ruta**  restaurant/ | **HTTP Metod** | **Metoda kontrolera** |
| /find | GET | findAllRestaurants(@Query(‘cityId’)) |
| /:id | GET | findRestaurant(@Param('id')) |
| /available-tables | POST | findAvailableTables(@Body()) |
| /add | POST | addRestaurant((@Body()) |
| /tables/add | POST | addTables((@Body()) |
| /working-time/add | POST | addWorkingTime((@Body()) |
| /non-working-days/add | POST | addNonWOrkingDays((@Body()) |

* reservations.controller.ts

|  |  |  |
| --- | --- | --- |
| **Ruta**  reservations/ | **HTTP Metod** | **Metoda kontrolera** |
| /find | GET | findAllReservations(@Query(‘statusCode)) |
| /add | POST | addReservation((@Body()) |
| /confirm/:id | GET | confirmReservation((@Param('id')) |
| /reject/:id | GET | rejectReservation((@Param('id')) |

* auth.controller.ts

|  |  |  |
| --- | --- | --- |
| **Ruta**  auth**/** | **HTTP Metod** | **Metoda kontrolera** |
| /manager/login | POST | loginManager(@Body()) |
| /manager/register | POST | registerManager(@Body()) |

## 3.7 Projektna ograničenja

Grafički korisnički interfejs je realizovan tako što se korisniku servira statički deo stranice koji dinamički doprema podatke sa servera pomoću asinhronih funkcija i ažurira stranicu koja se prikazuje korisniku. Serverski deo projekta je realizovan na *NodeJS* platformi korišćenjem *NestJS* razvojnog okvira. Sam kod je organizovan prema pravilima *MVC* arhitekture. Baza podataka je relaciona i korišćena je *MySQL* baza odataka. Klijentska strana je realizovana u *ReactJS* biblioteci, dok je za potrebe responsive dizajna korišćen *Bootstrap*, tj. *ReactBootstrap*. Izrada projekta je sprovedena korišćenjem alata za verziranje koda *Git* i celokupan kod projekta nalazi se na *GitHub*-u na linku: [https://github.com/](https://github.com/stex123/rent-a-car)cekara5.

# Realizacija sitema

U ovom poglavlju pojašnjeno je kako su neke od najkompleksnijih funkcionalnosti sistema realizovane programski.

## 4.1 Detalji restorana – vreme otvaranja/zatvaranja

Pri pregledu restorana, potrebno je prikazati da li je trenutno restoran otvoren ili zatvoren.

## Detalji restorana – vreme otvaranja/zatvaranja

Pri pregledu restorana, potrebno je prikazati da li je trenutno restoran otvoren ili zatvoren. U slučaju kada je restoran otvoren potreno je prikazati vreme zatvaranja, dok je u slučaju da je restoran zatvoren potrebno prikazati vreme i dan kada se restoran otvara, uzimajući u obzir i neradne dane restorana.

Serverska funkcija kojom je implementirano dohvatanje potrebnih podataka zove se *getOpenigDetails* i njen potpis je prikazan na Slici 4.1.1.

export function getOpeningDetails(

    dateToCheck: Date,

    restourantWorkingHours: RestourantWorkingHours[],

    restourantNonWorkingDays: NonWorkingDays[],

    daysOfWeek: DayOfWeek[],

    recursiveCall: boolean

): OpeningDetails

*Slika 4.1.1*

Funkcija je implementirana tako što se rekurzivno poziva, za sutrašnji dan, svaki put kada je dan za koji se proverava radno vreme (*dateToCheck)* neradan dan restorana ili ako je današnje radno vreme završeno i potrebno je pronaći prvi sledeći radni dan restorana. U svakom slučaju rekurzivnog poziva potrebno je zapamtiti da se radi o rekurzivnom pozivu jer ta informacija nam govori da je restoran sigurno zatvoren u datom trenutku(samo je cilj pronaći vreme i datum otvaranja restorana).

if (!todaysWorkingTime) { // ako nije ubaceno radno vreme restorana

        return null;

    }

    if (!todaysWorkingTime.isWorking) { // ako ne radi restoran tog dana idemo dalje

        return getOpeningDetails(tomorrow, restourantWorkingHours, restourantNonWorkingDays, daysOfWeek, true);

    }

U slučaju da se restoran trenutnog dana zatvara posle ponoći(ako je prva cifra ‘0’), potrebo je dodati +24h na vreme zatvranja kako se ne bi greškom dobilo da je restoran zatvoren. Primera radi radno vreme je 10:00-02:00.

// ako je closingTime "02:00" dodaj +24sata jer je to sledeci dan i pritom opening time nije manji "02:00"

    if (todaysWorkingTime.closingTime.split(':')[0].charAt(0) === '0' && closingInMins < openingInMins) {

        closingHrs += 24;

        closingInMins = closingHrs \* 60 + closingMin;

    }

if (recursiveCall) {

        if (!nonWordkingDayToday) { // ako nije neradni dan

            const openingTimeRecursive = todaysWorkingTime.openingTime.split(':')[0] + ":" + todaysWorkingTime.openingTime.split(':')[1];

            const openingDayRecursive = daysOfWeek.find(day => day.ordinalNumber === ordinalNumberOfDay).name;

            return { isOpened: false, openingTime: openingTimeRecursive, openingDay: openingDayRecursive, closingTime: null };

        }

        // ako jeste neradni dan nastavi dalje dok ne dodje do prvog radnog

        return getOpeningDetails(tomorrow, restourantWorkingHours, restourantNonWorkingDays, daysOfWeek, true);

    }

    let isOpened = !nonWordkingDayToday && currentTimeInMins >= openingInMins && currentTimeInMins <= closingInMins;

U slučaju kada je restoran trenutno otvoren, tj. Kada je trenutno vreme izmedju radnog vremena za trenutni dan, funkcija odmah vraća rezultat da je restoran trenutno otvoren, sa vremenom zatvaranja restorana trenutnog dana.

Takođe, u slučaju kada je restoran naizgled trenutno zatvoren, potrebno je proveriti i mogućnost da je restoran prethodnog dana zatvara posle 00:00, a da je vreme za trenutni dan manje od vremena zatvaranja restorana od prethodne večeri, te je restoran trenutno i dalje otvoren.

let openingTime: string = null;

let openingDay: string = null;

let closingTime: string = null;

if (isOpened) {

        closingTime = todaysWorkingTime.closingTime;

    }

    else { // if it is closed

        if (!nonWordkingDayToday) { // ako nije neradni dan

            if (currentTimeInMins < openingInMins) { // i ako je vreme otvaranja posle trenutnog, proveriti da li je u toku radno vreme od prethodnog dana: npr. sada je cetvrtak 01:00, a restoran sredom radi 10:00-02:00 => treba da vrati true

                if (!nonWordkingDayYesterday) {

                    const yesterday = ordinalNumberOfDay === 0 ? 6 : ordinalNumberOfDay - 1; // u slucaju da je nedeljea(now.getDay()=0) danas-> juce je nedelja (now.getDay()=0)

                    const yesterdayWorkingTime = restourantWorkingHours.filter(rwh => rwh.dayOfWeekId === yesterday)[0];

                    const yesterdayClosingTimeInHrs = (parseInt(yesterdayWorkingTime.closingTime.split(':')[0]));

                    const yesterdayClosingTimeMins = (parseInt(yesterdayWorkingTime.closingTime.split(':')[1]));

                    const yesterdayClosingTimeInMins = yesterdayClosingTimeInHrs \* 60 + yesterdayClosingTimeMins;

                    // treba nam i vreme otvaranja da bismo proverili da li je uopste juce restoran radio posle ponoci

                    // sto bi znacilo da je vreme zatvaranja manje od vremena otvaranja => yesterdayClosingTimeInMins < yesterdayOpeningTimeInMins

                    const yesterdayOpeningTimeInHrs = (parseInt(yesterdayWorkingTime.openingTime.split(':')[0]));

                    const yesterdayOpeningTimeMins = (parseInt(yesterdayWorkingTime.openingTime.split(':')[1]));

                    const yesterdayOpeningTimeInMins = yesterdayOpeningTimeInHrs \* 60 + yesterdayOpeningTimeMins;

                    if (yesterdayClosingTimeInMins < yesterdayOpeningTimeInMins && yesterdayClosingTimeInMins >= hrsNow \* 60 + minsNow) { // ukoliko jeste zatvara se u yesterdayClosingTime

                        isOpened = true;

                        closingTime = yesterdayWorkingTime.closingTime;

                    }

                    else { // trenutno zatvoren -> otvara se u danasnjih openingHrs

                        openingTime = todaysWorkingTime.openingTime.split(':')[0] + ":" + todaysWorkingTime.openingTime.split(':')[1];;

                        openingDay = daysOfWeek.find(day => day.ordinalNumber === ordinalNumberOfDay).name;

                    }

                }

                else { // ako je bio neradan dan juce

                    openingTime = todaysWorkingTime.openingTime.split(':')[0] + ":" + todaysWorkingTime.openingTime.split(':')[1];;

                    openingDay = daysOfWeek.find(day => day.ordinalNumber === ordinalNumberOfDay).name;

                }

            }

            else { // ako je trenutno vreme posle vremena zatvaranja, nastavi proveri za sutradan

                return getOpeningDetails(tomorrow, restourantWorkingHours, restourantNonWorkingDays, daysOfWeek, true);

            }

        }

        else { // ako je neradni dan danas, nastavi proveri za sutradan

            return getOpeningDetails(tomorrow, restourantWorkingHours, restourantNonWorkingDays, daysOfWeek, true);

        }

    }

    return { isOpened, openingTime, openingDay, closingTime };

}

## 4.2 Provera slobodnih stolova pri izvršavanju rezervacija

Kada neprijavljeni korisnici žele da izvrše rezervaciju u izabranom restoranu u izabranom terminu, potrebno je pronaći sve stolove datog restorana koji u datom trenutku nemaju potvrđenu rezervaciju. Pritom je potrebno i prikazati maksimalno moguće vreme trajanja rezervacije, na koje, osim maksimalnog trajanja dužine rezervacije izabranog stola, može uticati i ukoliko je taj sto rezervisan u nekom narednom trenutku. U tom slučaju je, potrebno smanjiti maksimalno vreme trajanja dužine rezervacije datog stola na razliku vremena prve sledeće rezervacije i željenog termina rezervacije, u slučaju da je ta razlika manja od maksimalne dužine trajanja rezervacije tog stola.

Prvo se proverava pomoću funkcije *getOpenigDetails* (4.1) da li restoran radi u datom trenutku.

const apiResponse = new ApiResponse();

        // prvo proveriti da li restoran radi u datom vremenu

        const date = new Date(findAvailableTablesDto.reservationDate);

        const hh = parseInt(findAvailableTablesDto.fromTime.split(":")[0]);

        const mm = parseInt(findAvailableTablesDto.fromTime.split(":")[1]);

        const timeToCheck = new Date(date.getFullYear(), date.getMonth(), date.getDate(), hh, mm);

        const daysOfWeek = await this.dayOfWeekRepository.find();

        const options: any = {

            relations: ['nonWorkingDays', 'restourantWorkingHours']

        };

        const restourantInfo = await this.restourantRepository.findOne(findAvailableTablesDto.restourantId, options);

        const openingDetails = getOpeningDetails(

            new Date(timeToCheck),

            restourantInfo.restourantWorkingHours,

            restourantInfo.nonWorkingDays,

            daysOfWeek,

            false

        );

        if (!openingDetails.isOpened) { // zatvoren

            apiResponse.data = {

                isOpened: false,

                message: "Restoran je u izabranom vrmenu zatvoren.",

                tables: []

            };

            return Promise.resolve(apiResponse);

        }

Ukoliko radi, prvo se dohvataju sve rezervacije datog stola u datom trenutku, kako bi se isključile iz pretrage slobodnih stolova i id-evi ovih stolova se čuvaju u nizu *arrayWithIdsOfReservedTables*.

 const reservedTables = await getRepository(RestourantTables)

            // dohvataju se svi rezervisani stolovi u zeljenom trenutku rezervacije

            .createQueryBuilder("tables")

            .innerJoinAndSelect("tables.reservations", "reservation", "tables.restourantId = :id", { id: findAvailableTablesDto.restourantId })

            .where("reservation.statusId = :statusId", { statusId: 2 }) // odobrena

            .andWhere("reservation.fromTime <= :fromTime", { fromTime: findAvailableTablesDto.fromTime }) // nije odobrena

            .andWhere("reservation.untillTime >= :fromTime", { fromTime: findAvailableTablesDto.fromTime })

            .andWhere("reservation.reservationDate = :date", { date: findAvailableTablesDto.reservationDate })

            .select("tables.id")

            .getMany();

        let arrayWithIdsOfReservedTables = [];

        reservedTables.forEach(rt => arrayWithIdsOfReservedTables = [...arrayWithIdsOfReservedTables, rt.id]);

// dohvataju se svi raspolozivi stolovi u zeljenom trenutku

        const restourantTables = await getRepository(RestourantTables)

            .createQueryBuilder("tables")

            .innerJoinAndSelect("tables.description", "description")

            .where("tables.restourantId = :id", { id: findAvailableTablesDto.restourantId })

            .andWhere("tables.id NOT IN (:...reserved)", { reserved: arrayWithIdsOfReservedTables })

            .getMany();

        const reservedTablesLater = await getRepository(RestourantTables)

            // dohvataju se svi rezervisani stolovi koji imaju rezervaciju kasnije tog dana

            .createQueryBuilder("tables")

            .innerJoinAndSelect("tables.reservations", "reservation", "tables.restourantId = :id", { id: findAvailableTablesDto.restourantId })

            .where("reservation.statusId = :statusId", { statusId: 2 }) // odobrena

            .andWhere("reservation.fromTime > :fromTime", { fromTime: findAvailableTablesDto.fromTime }) // nije odobrena

            .andWhere("reservation.reservationDate = :date", { date: findAvailableTablesDto.reservationDate })

            .getMany();

Kada su dostupni svi dostupni stolovi i sve rezervacije za dati dan u datom restoranu, kod kojih je vreme početka rezervacije posle željenog trenutka rezervacije koja se pravi.

Sada je potrebno za svaki sto proći kroz njegove rezervacije, naći prvu sledeću rezervaciju, izračunati maksimalno vreme između ove dve rezervacije, I ukoliko je to vreme kraće od maksimalnog trajanja rezervacije za dati sto, ađurirati to vreme na novoizraćunato vreme.

// prolazak kroz sve slobodne stolove->prolazak kroz sve rezervisane kasnije->trazenje prve sledece rezervacije i po potrebi azuriranje maxHrsAvailable polja

        restourantTables.forEach(rt => {

            reservedTablesLater.forEach(rtl => {

                if (rt.id === rtl.id) {

                    const reservationFromTimeInMins = parseInt(findAvailableTablesDto.fromTime.split(":")[0]) \* 60 + parseInt(findAvailableTablesDto.fromTime.split(":")[1]);

                    let fromTimeReservedTable: number = null; // vreme pocetka prve sledece rezervacije

                    // prodji kroz sve rezervacije datog stola i nadju najblizu ovoj rezervaciji po vremenu

                    rtl.reservations.forEach(res => {

                        const fromTimeReservedTableInMins = parseInt(res.fromTime.split(":")[0]) \* 60 + parseInt(res.fromTime.split(":")[1]);

                        if (reservationFromTimeInMins < fromTimeReservedTableInMins && (!fromTimeReservedTable || fromTimeReservedTableInMins < fromTimeReservedTable)) {

                            fromTimeReservedTable = fromTimeReservedTableInMins;

                        }

                    });

                    if (fromTimeReservedTable !== null) {

                        const differnceInHrs = (fromTimeReservedTable - reservationFromTimeInMins) / 60;

                        console.log(differnceInHrs)

                        if (rt.maxHoursAvailable > differnceInHrs) {

                            rt.maxHoursAvailable = differnceInHrs;

                        }

                    }

                }

            })

        })