Univerzitet u Novom Sadu
Fakultet Tehničkih nauka
Novi Sad

Izveštaj o obavljenoj stručnoj praksi

Student: Ime prezime, SFXX/YYYY

Logo firme:

Ime firme

Adresa firme

PIB: XXXXXXXX

Website:

sajt firme

Mentor prakse: Ime Prezime, mail@mail

Ime fime je kompanija koja se bavi razvojem softvera za potrebe klijenata širom sveta. Njeno sedište se nalazi u Nju Jorku, SAD. Poseduje i svoju kancelariju u Novom Sadu i ukupno ima oko 80 zaposlenih. U kompaniji se prate trendovi i na raznim projektima koriste se programski jezici kao što su JAVA, C#, PHP, PYTHON, NODE.JS. U poslednjih nekoliko godina sve više se koriste i sledeće tehnologije: Blockchain, Flutter, AWS. Neki od projekata na kojima je kompanija radila za svoje klijente su: softver za rezervaciju karte parking mesta u gradu, softver za kupovinu karata za javne događaje i mnogi drugi.

SADRŽAJ

1. Specifikacija projekta.....1
2. Tehnologije.....2
3. Način i tok izvođenja prakse.....3
4. Implementacija projekta.....4
   4.1 Baza podataka – model.....4
   4.2 Login.....7
   4.3 Security.....10
   4.4 Dodavanje obroka.....14
   4.5 Dnevni meni.....18
   4.6 Rezervisanje obroka.....20
   4.7 ChosenOne korisnik.....22
5. Zaključak.....25

1. Specifikacija projekta

Aplikacija koju smo razvijali na praksi je predviđena za interno korišćenje unutar kompanije Ime kompanije u svrhu naručivanja hrane na dnevnom nivou.

Naručivanje hrane se odvija preko Viber grupe preduzeća “Topli obrok”. Preduzeće “Topli obrok” jednom nedeljno izbacuje meni za celu nedelju. Na dnevnom nivou nudi 2 opcije obroka na meniju: Redovan obrok i Fit obrok. Postoji opcija da se “Redovan obrok” poruči u vidu velike ili male porcije, dok “Fit obrok” nema tu opciju. Postoje jela na meniju koje je potrebno naručiti dan ranije do 17h. Npr. Fit obrok koji se nudi sredom, potrebno je naručiti u utorak najkasnije do 17h. Pored glavnh jela, određenim danima koji su naznačeni na nedeljnom meniju, preduzeće nudi dezert i supu koju je takođe potrebno naručiti dan ranije do 17h.

Kako izgleda naš proces naručivanja: Svako jutro neko od zaposlenih napiše na ST Skype grupi (u kojoj se nalaze svi zaposleni) šta je taj dan na meniju. Nakon toga, zaposleni pišu da li žele da naruče taj ručak i da li žele veliku ili malu porciju. Nakon toga, neko od zaposlenih ima zadatak da sabere koliko kojih porcija ljudi žele da naruče i da izvrši porudžbinu preko Viber grupe “Topli obrok” npr. 9 velikih obroka, 3 mala obroka i 1 fit obrok, Bulevar Oslobođenja 92, treći sprat, telefon: 064/2455-264. Takođe, neko od zaposlenih uvek preuzme na sebe da sakupi novac za ručak od svih ljudi u firmi koji su naručili obrok i da plati kada isporuka stigne.

Aplikacija svakog dana u 10h zatvara mogućnost poručivanja za taj dan I randomizacijom izabere korisnika koji će manuelno proslediti porudžbinu u Viber grupu.

Tehnologije koje su korišćene razvijanjem aplikacije su:

- Angular
- Java Spring Boot
- MySQL

2. Tehnologije

Angular

AngularJS je frontend framework koji je razvio Google. To je struktuiran farmework napravljen na JavaScriptu i služi za razvijanje dinamičkih web aplikacija. Dozvoljava da se koristiti standardni HMTL i CSS, ali isto tako proširuje mogućnosti HTML-a. Angular komponente su jako korisne, i relativno lako ih je koristiti. Postoje više vrsta angulara: Angular JS I Angular 2+.

Angular 2+ su verzije koje dolaze nakon AngularJS. U pitanju je open-source frontend web platforma zasnovana na TypeScript programskom jeziku.

U mom slučaju, prilikom razvijanja aplikacije koristio sam Angular 13.

Java Spring Boot

Spring Framework je radni okvir za platformu Java koji obezbeđuje infrastrukturu za razvoj aplikacije. Omogućava lakše i brže kreiranje softvera, čistiji kod na javi, a time i lakše održavanje aplikacije. Spring je organizovan u oko 20 modula koje je moguće pojedinačno upotrebljavati zavisno od potreba aplikacije i otvorenog je koda. Prvu verziju je napisao Rod Džonson, a predstavio u oktobru 2002.

MySQL

MySQL je open source sistem za upravljanje relacionim bazama (RDBMS). To je najpopularniji open source sistem za upravljanje bazama podataka na svetu i trenutno je rangiran kao drugi najpopularniji sistem za upravljanje bazama podataka na svetu (posle Oracle Database). MySQL je zasnovan na **SQL**-u - standardnom jeziku za upis podataka unutar sistema za upravljanje relacionim bazama podataka.

3. Način i tok izvođenja prakse

Projekat sam razvijao sa još dvojicom kolega, gde sam ja radio frontend deo aplikacije koristeći **Angular** radni okvir, kao i neke funkcionalnosti na backend delu aplikacije, koristeći **Java Spring Boot** i **MySQL**. Kolege su uglavnom radile backend deo aplikacije. Sve promene smo kačili na **git** repozitorijumu koristeći **BitBucket** I **SourceTree** alate. Praksu smo radili od kuće i trajala je oko mesec dana, tačnije od 30.03.2022. do 27.04.2022. Svakog radnog dana smo imali sastanak sa mentorom koji je trajao oko 45 minuta. Na sastancima smo mu postavljali željena pitanja oko nesuglasica vezanih za projekat, ali nas je i sam mentor usmeravao za dalji tok implementacije. Sa kolegama sam komunicirao preko **Discord** grupe koju smo namestili za potrebu komunikacije oko projekta. Poslednjeg dana smo išli u firmu i prezentovali direktoru firme naš rad.

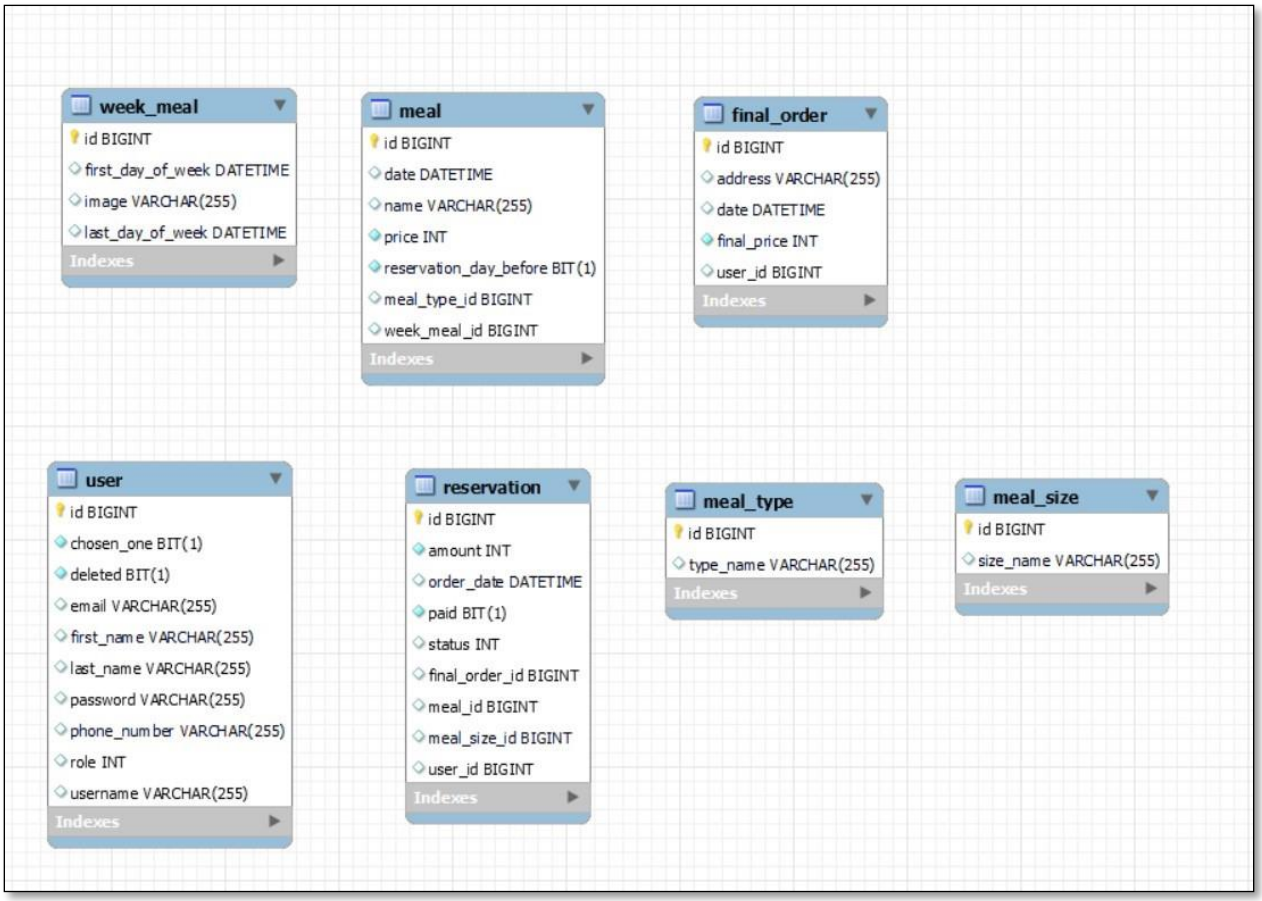
4. Implementacija projekta

4.1 Baza podataka – model

```
@Entity
public class Meal {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public Long id;


    @ManyToOne
    private MealType mealType;

    @ManyToOne
    private WeekMeal weekMeal;

    @JsonIgnore
    @OneToMany(mappedBy="meal" ,fetch = FetchType.LAZY, cascade=CascadeType.ALL)
    private List<Reservation> orders = new ArrayList<Reservation>();

    private String name;
    private int price;
    private Date date;
    private boolean reservationDayBefore;
```

Slika 1 - primer klase Meal

Bazu podataka smo generisali uz pomoć **Hibernate**-a. U ovom primeru iznad klase **Meal**, pomoću anotacije **@Entity** mapiramo naziv klase na naziv tabele u bazi podataka. Nazivi atributa će biti mapirani na kolone unutar tabele. Anotacija **@Id** označava jedinstveni (**primary**) ključ i automatski će se povećavati čuvanjem sledećeg objekta u bazu. Takođe u ovom primeru imamo i anotaciju **@OneToMany** koja označava vezu jedan prema više, što znači da jedan obrok može imati više rezervacija, dok rezervacija može imati samo jedan obrok. U tabeli **Reservation** postoji strani ključ (**meal_id**) koji vezuje te dve tabele. Isti princip je i za **@ManyToOne** veze, obrok može imati samo jedan tip i može pripadati samo jednom meniju (u tabeli **Meal** postoje strani ključevi **meal_type_id** i **week_meal_id**). Kada se učitavaju obroci, zbog FetchType.**LAZY**, njene rezervacije se neće učitavati kada i sami obroci, već će se učitavati posebno iz baze kada za to bude bilo potrebe.



Slika 2 – diagram modela

WeekMeal – predstavlja nedeljni meni. Admin unosi početni i krajnji datum nedeljnog menija, neophodno da bi se obroci dodavali posle.

Meal – predstavlja obroke koje unosi admin. Postoje i obroci koji zahtevaju da se naruče dan ranije. Mogu imati samo jedan tip i pripadati samo jednom nedeljnom meniju.

User – korisnik aplikacije. Može imati ulogu admin-a ili user-a. Takođe postoji i **chosenOne** korisnik koji se randomizacijom bira svaki dan i zadužen je za prikupljanje novca tog dana.

FinalOrder – konačna narudžbina koja sadrži datum, adresu, ukupnu cenu koja treba da se plati i **chosenOne** korisnika tog dana.

Reservation – rezervacija obroka. Ukoliko je regular obrok, sadrži veličinu, veliku ili malu porciju, dok za druge tipove obroka ne postoji ta opcija. Može imati status "**RESERVED**" ili status "**ORDERED**" u zavisnosti da li je obrok rezervisan ili naručen.

MealType – tip obroka.

MealSize – veličina obroka, koja je vezana za rezervaciju, ne za obrok.

4.2 Login

Kada se pokrene aplikacija, korisnik će automatski biti redirektovan na login stranicu. Za izgled login forme koristio sam **Angular material** biblioteku.

Slika 3 – login stranica

```
<form (ngSubmit)="onSubmit()" name="loginForm" [formGroup]="loginForm">
  <div class="usernameInput">
    <mat-form-field class="full-width" appearance="outline">
      <mat-label>Username</mat-label>
      <input
        formControlName="username"
        matInput
        placeholder="Enter username" required
      />
      <mat-error *ngIf="!loginForm.controls['username'].valid">
        Username is required
      </mat-error>
    </mat-form-field>
  </div>

  <div>

    <mat-form-field class="full-width" appearance="outline">
      <mat-label>Password</mat-label>
      <input formControlName="password" matInput [type]=" hide ? 'password' : 'text'" required />
      <button  mat-icon-button matSuffix (click)="hide = !hide" [attr.aria-label]="'Hide Password'"
      [attr.aria-pressed]="hide">
      <mat-icon>
        {{hide ? 'visibility_off' : 'visibility'}}
      </mat-icon>
  </button>
  <mat-error *ngIf="!loginForm.controls['password'].valid">
      Password is required
    </mat-error>
  </mat-form-field>
  </div>
  <button class="loginBtn" type="submit" mat-flat-button color="primary">Login</button>
```

Slika 4 – html stranica login komponente

Za preuzimanje podataka pri unosu koristio sam i importovao **Reactive forms** biblioteku u ovom slučaju.

```
export class LoginComponent implements OnInit {

  hide: boolean = false;
  login: Login;
  loginDto: LoginDto;
  loginForm: FormGroup;

  constructor(private formBuilder: FormBuilder, private loginService: LoginService,
    this.login = new Login();
    this.loginDto = new LoginDto();
  }

  ngOnInit(): void {
    this.hideTabsWhenTokenExpire();
    this.loginForm=this.formBuilder.group({
      username: ['',Validators.required],
      password: ['',Validators.required],

    });

  }

  onSubmit(){

    if (this.loginForm.invalid) {
      alert('Invalid input!');
    }else{
      this.login.username = this.loginForm.value.username;
      this.login.password = this.loginForm.value.password;

      this.loginService.login(this.login).subscribe(data =>{
        this.loginDto = data;
```

Slika 5 – login.component.ts

Deklariše se **loginForm** objekat koji je tipa **FormGroup** klase. Za svaki input je vezan **formControlName**, **username** i **password**. Preko **formBuilder**-a se pravi **loginForm**, sa inicijalnim vrednostima praznog stringa i dodaju se validatori (**required** u ovom slučaju, neophodno je da budu popunjeni). Kada korisnik unese vrednosti i pritisne dugme "**Login**", u atribute objekta login koji je tipa klase **Login** (sa atributima **username** i **password** string tipa) se stavljaju vrednosti iz input-a koje je korisnik uneo (vrednosti iz **loginForm** objekta). Nakon toga se poziva servis i putem **POST** metode se prosleđuje **login** objekat sa svojim vrednostima na server gađajući **REST** endpoint.

```
else if(this.loginDto.messageInvalidUsernameOrPassword == "no"){
  localStorage.token = this.loginDto.token;

  //alert('Successfully logged in!' + ' Token: ' + localStorage.token);
  this.alertSwal();

  if(this.loginDto.user.role.toString() === "USER"){
    this.setLoggedUser();
    localStorage.role = this.loginDto.user.role.toString();
    this.hideAndShowTabsForUser();

    timer(1500).subscribe(t=>this.goToHomePage());
    if(this.loginDto.user.chosenOne == true){
      //this.alertChosenOne();
      timer(1500).subscribe(t=>this.alertChosenOne());
    }


  }
  else if(this.loginDto.user.role.toString() == "ADMIN"){ // nzm sto nece da poredi this.
    //this.goToAdminPage();
    this.setLoggedUser();
    localStorage.role = this.loginDto.user.role.toString();
    this.hideAndShowTabsForAdmin();
    timer(1500).subscribe(t=>this.goToAdminWeekMealPage());
  }
```

Slika 6 – login.component.ts, nakon odgovora servera

Server vraća kao odgovor **loginDto** objekat koji u sebi sadrži i poruku da li je logovanje uspešno izvršeno ili ne. Ukoliko jeste, korisnik dobija poruku o tome. Nakon toga u **localStorage.role** stavlja se uloga korisnika koja je dobijena iz atributa **loginDto.user.role**. Ukoliko je uloga korisnika jednaka "**USER**", sakrivaju se i prikazuju određeni tabovi namenjeni za tu ulogu i vrši se redirekcija ka **home-page** stranici. Na isti način, ukoliko je uloga korisnika "**ADMIN**", u **localStorage.role** stavlja se uloga korisnika koja je dobijena iz atributa **loginDto.user.role** i prikazuju se određeni tabovi. Nakon toga vrši se redirekcija ka **admin-week-meal-page**.

4.3 Security

Aplikaciju mogu koristiti samo autentifikovani korisnici i u zavisnosti od uloge, određene stranice mogu posetiti samo autorizovani korisnici. Pošto je **HTTP** protokol statless protokol, odnosno ne pamti se stanje od prošlog zahteva, neophodno je čuvati podatke o klijentu. U ovom projektu smo koristili **Spring Security** sa **JWT** (**Json Web Tokenom**). Kada se korisnik uspešno prijavi, server šalje nazad generisani **JWT**. Dobijeni token se nakon toga smešta u **localStorage**.

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor(private router: Router){

  }
  intercept( req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    //const token = localStorage.token; // you probably want to store it in localStorage or something
    const token = localStorage['token'];
    if (!token) {
      return next.handle(req);
    }

    const req1 = req.clone({
      headers: req.headers.set('Authorization', `Bearer ${token}`),
    });
```

Slika 7 – AuthInterceptor

AuthInterceptor (slika 7) će u zaglavlju (**header**) svakog sledećeg zahteva upućenom serveru, da pošalje i token koji je prethodno dobio od istog. Na taj način će server svaki zahtev da filtrira i u zavistnosti od validnosti tokena dozvoliti dalje izvršavanje.

```
@Override
public LoginDTO generateToken(Login login) {
    LoginDTO loginDTO = new LoginDTO();
    User user = new User();

    try {

        user = findByUsername(login.getUsername());
        BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

        if(user.getPassword().equals(login.getPassword())) {

        }
        else if(encoder.matches(login.getPassword(), user.getPassword()) == false) {
        throw new Exception();
        };

       String token = jwtUtil.generateToken(login.getUsername());

       loginDTO = new LoginDTO(token, user, "no");
```

Slika 8 – generateToken metoda

BCryptEncoder smo koristili za čuvanje lozinke u bazi prilikom registracije korisnika. Preko metode „**matches**“ se prilikom login-a, porede uneta lozinka i enkriptovana lozinka iz baze podataka.

Kao što sam ranije napomenuo, svaki sledeći zahtev upućen serveru, prolazi kroz filter i proverava se da li je korisnik autentifikovan i validnost njegovog tokena. Token takođe ima svoje vreme trajanja, nakon toga neće više biti validan.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.cors().disable();
    http.csrf().disable().authorizeRequests().antMatchers("/api/login", "/api/user/createUser")
        .permitAll().antMatchers(HttpMethod.OPTIONS, "/**")
            .permitAll().anyRequest().authenticated()
            .and().exceptionHandling().and().sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);;
}
```

Slika 9 – SecurityConfig klasa

Jedino zahtevi upućeni ka putanji za login i registraciju neće prolaziti kroz filter, odnosno ne zahtevaju autentifikaciju i validan token.

Na frontend delu aplikacije sam uradio **AuthGuard** i **RoleGuard** servise.

```
const routes: Routes = [
  {path: '', redirectTo: 'login', pathMatch: 'full'},
  {path: 'login', component: LoginComponent},
  {path: 'registration', component: RegistrationComponent},
  {path: 'home-page', component: HomePageComponent, canActivate: [RoleGuard],
  data: {
    expectedRole: 'USER'
  }},
  {path: 'admin-page/:weekMealId', component: AdminComponent, canActivate: [RoleGuard],
  data: {
    expectedRole: 'ADMIN'
  } },
  {path: 'admin-page-week-meal', component: AdminWeekMealComponent, canActivate: [RoleGuard],
  data: {
    expectedRole: 'ADMIN'
  } },
  {path: 'user-page-active-reservations', component: UserPageActiveReservationsComponent, canActivate: [RoleGuard],
  data: {
    expectedRole: 'USER'
  } },
  {path: 'admin-page-all-user-list', component: AdminPageAllUserListComponent, canActivate: [RoleGuard],
  data: {
    expectedRole: 'ADMIN'
  } },
  {path: 'user-my-profile', component: UserMyProfileComponent, canActivate: [AuthGuard]},
  {path: 'admin-active-reservation-list', component: AdminActiveReservationListComponent, canActivate: [RoleGuard],
  data: {
    expectedRole: 'ADMIN'
  } },
  {path: 'user-page-chosen-one', component: UserPageChosenOneComponent, canActivate: [RoleGuard],
  data: {
    expectedRole: 'USER'
```

Slika 10 – app-routing-module.ts

Na slici 10 prikazane su rute koje su vezane za komponente. Svaka komponenta preko interfejsa **canActivate**, ima svoj **RoleGuard** ili **AuthGuard**. **RoleGuard** zahteva da korisnik mora imati odgovarajuću ulogu da bi mogao da joj pristupi dok **AuthGuard** zahteva da korisnik mora biti autentifikovan.

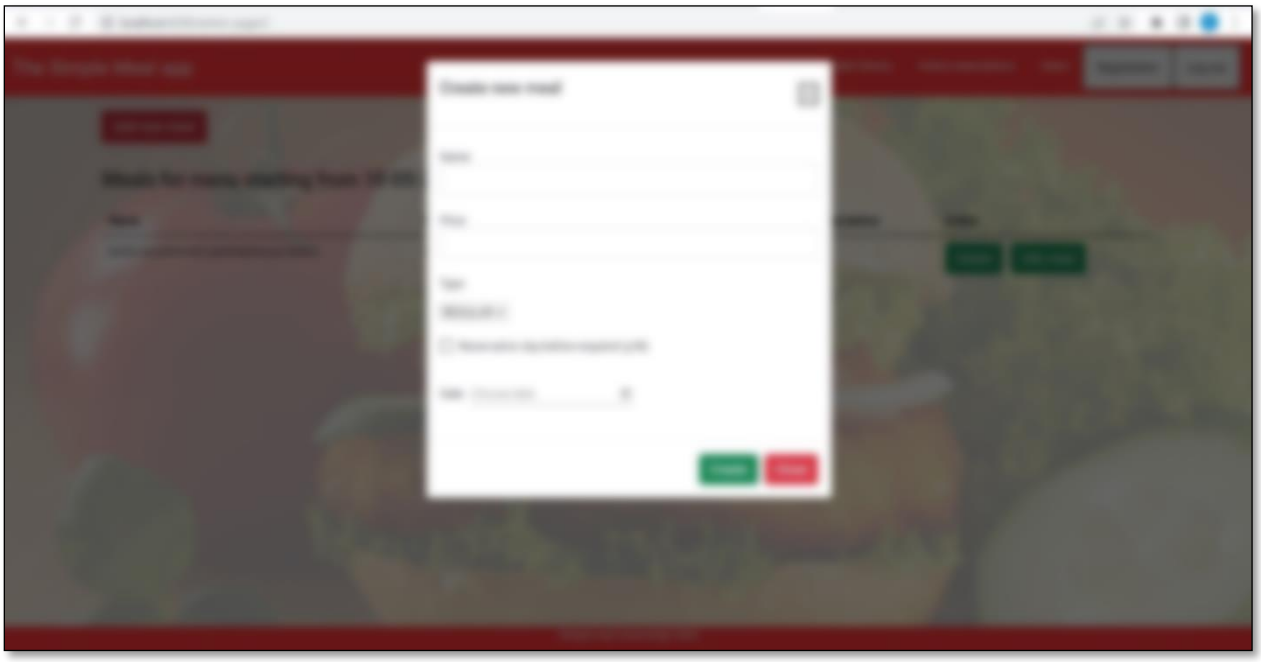
```
@Injectable()
export class RoleGuardService implements CanActivate {
  constructor(private loginComponent: LoginComponent, private appComponent: AppComponent, private previousRouteService: Previous
  canActivate(route: ActivatedRouteSnapshot): boolean {
    // this will be passed from the route config
    // on the data property
    const expectedRole = route.data.expectedRole;
    const token = localStorage.getItem('token');

  if (!this.auth.isAuthenticated() || localStorage.role !== expectedRole) {
    //ako kao ulogovan user gadja url od admina, vrati ga na home page i ponovo prikazi tabove koji su potrebni, jer ih je brisa
    if(localStorage.role == "USER"){
      this.router.navigate(['home-page']);
      this.appComponent.getHTMLElementsById(); // morao sam i ovo, jer nije radilo kako treba ponovno pokazivanje tabova
      this.loginComponent.hideAndShowTabsForUser();
      return false;

    }
    else if(localStorage.role == "ADMIN"){
      this.router.navigate(['admin-page-week-meal']);
      this.appComponent.getHTMLElementsById();
      this.loginComponent.hideAndShowTabsForAdmin();
      return false;
    }
```

Slika 11 – RoleGuard klasa

Na slici 11 prikazana je klasa **RoleGuardService** koja implementira interfejs **CanActivate**. Uzima se token iz **localStorage** i očekivana uloga iz rute (na prethodnoj slici za svaku rutu postoji definisana očekivana uloga korisnika koji može da pristupi). Proverava se da li je korisnik autentifikovan (takođe i da li je token istekao ili ne) i da li je uloga iz **localStorage** jednaka očekivanoj ulozi (u ovom slučaju se proverava da li je različita, odnosno nastaviće se izvršavanje ako ovaj uslov nije tačan). Ako prijavljeni korisnik pokuša da pristupi komponenti preko url-a za koju nema ovlašćenja, biće redirektovan na svoju '**home-page**' stranu ukoliko je uloga "**USER**" ili **'admin-page-week-meal**' stranu ukoliko je uloga "**ADMIN**".

4.4 Dodavanje obroka

Obrok može da dodaje samo korisnik sa ulogom "**ADMIN**". Njemu će se jedino pojaviti ta opcija nakon uspešnog logovanja. Obrok se dodaje tako što se prethodno izabere odgovarajući nedeljni meni. Klikom na dugme "**Add new meal**" otvoriće se dialog (**modal**) koji će omogućiti unos podataka.



Slika 12 – Dialog za dodavanje obroka

Kombinacijom **Bootstrap** biblioteke i **CSS**-a implementirao sam izgled dialoga, koji ima i određenu animaciju prilikom otvaranja. Pozadina će se takođe zamračiti prilikom otvaranja dialoga. U ovom slučaju za unos podataka koristio sam **Template-driven** formu.

```
<div class="modal" id="myModal" tabindex="-1" role="dialog">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <h2 class="modal-title">Create new meal</h2>
          <button type="button" class="close" (click)="closeModal()" id="closeXBtn" data-dismiss="modal" aria-label="Close">
            <span aria-hidden="true">&times;</span>
          </button>
        </div>
        <div class="modal-body">

          <div class="form-group">
            <label>Name</label>
            <input type="text" class="form-control" id="name" name="name"
            [(ngModel)]="meal.name">
          </div>

          <div class="form-group">
            <label>Price</label>
            <input type="number" class="form-control" id="price" name="price"
            [(ngModel)]="meal.price">
          </div>

          <div class="form-group" id="comboDentistDiv">
            <div class="labelComboDentist">
              <label> Type </label>
            </div>

            <div  class ="mealType">
              <select name="mealType" [(ngModel)] = "meal.mealType">
                <option *ngFor = "let mealType of mealTypes" [ngValue] = "mealType">{{mealType.typeName}}</option>
              </select>
            </div>
```

Slika 13 – Template driven forma za unos obroka

Template driven forme rade tako što se preko **[(ngModel)]** direktive, input element direktno “binduje” (veže) za atribut objekta. Sve što korisnik unese u input element za naziv obroka će se automatski “preslikati” u atribut “**meal.name**“ i obrnutno. Sve što se bude nalazilo u određenom atributu objekta će biti “preslikano” u određeni input element (**2-way binding**).

```
<div class="checkboxReservationRequired" *ngIf="meal.mealType.typeName == 'REGULAR' || meal.mealType.typeName == 'FIT'">
  <mat-checkbox [(ngModel)]="meal.reservationDayBefore" [ngModelOptions]="{standalone: true}" class="example-margin">Res
</div>
```

Slika 14 – div element u kojem se nalazi check-box element

Ako se izabere “**SOUP**“ ili “**DESSERT**“ tip obroka, opcija za rezervaciju dan ranije će se ukloniti (slika 14), odnosno sakriće se odgovarajući div element i za ove tipove će automatski taj atribut biti setovan na **true**. Provera se vrši preko **ngIf** uslova u html stranici.

Pritiskom na dugme “**Create**”, poziva se metoda **createMeal()**. Setuje se dnevni meni sa kog je došao, odnosno kom meniju pripada (**weekMeal**).

```
createMeal(){

  //alert(this.meal.name+  " " + this.meal.mealType + this.meal.price + this.meal.reservationDayBefore)
  this.meal.weekMeal = this.weekMeal; // setujem weekmeal
  //if(this.meal.mealType.typeName === "REGULAR"){ // ako je selektovan regular, stavi da je prazan obj
  //  this.meal.mealType = new MealType();
  //  delete this.meal.mealType; -- nzm sto ovo nece isto, a mozda bi valjalo da se iskoristi

  //}
  this.ifSoupOrDessertSetReservationDayBeforeTrue();
  this.mealService.createMeal(this.meal).subscribe(data =>{

    this.responseMessage = data;

    if(this.responseMessage === "valid"){
      //alert('Successfully added meal!');
      this.alertSwal();
      this.closeModal();
      //this.getAllMeals();
      this.getMealsForWeekMeal();
      this.meal = new Meal();
      this.setDefaultMealSizeAndType();

    }
    else if(this.responseMessage === "invalid"){
      //alert('Invalid input!');
      this.alertInvalidInput();

    }
    else if(this.responseMessage === "invalidDateInWeekMealInterval"){
      //alert('Invalid date input, date of the meal is not in range of week meal (menu) dates');
      this.alertInvalidDateRangeInput();
```

Slika 15 – admin.component.ts fajl

Ako je tip “**SOUP**” ili “**DESSERT**”, setuje se atribut da je neophodna rezervacija dan ranije na true. Nakon toga se poziva mealService I prosleđuje se meal objekat, koji u svojim atributima već ima unete vrednosti iz input-a, koje su se “preslikale” preko **[(ngModel)]** direktive.

```
@Injectable({
  providedIn: 'root'
})
export class MealService {

  constructor(private httpClient: HttpClient) { }

  createMeal(meal: Meal):Observable<any>{
    return this.httpClient.post("http://localhost:8080/api/meal/createMeal", meal, {responseType: 'text' });
  }
```

Slika 16 – meal-service

Šalje se **POST** zahtev sa meal objektom u svom telu zahteva i gađa se definisana lokacija na strani servera (u **REST** kontroleru). Kao odgovor klijent očekuje objekat

sa tekstualnim sadržajem, sa informacijom da li je uspešno izvršeno čuvanje obroka u bazu podataka.

```
@CrossOrigin(origins = "http://localhost:4200")
@RestController
@RequestMapping(value = "api/meal")
public class MealController {

    @Autowired
    MealService mealService;

    @RequestMapping(value = "/createMeal", method = RequestMethod.POST, consumes = MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<String> createMeal(@RequestBody Meal meal) {
        String responseToClient;
        responseToClient = mealService.isValidInput(meal);
        if (responseToClient.equals("valid")) {
            if (mealService.isValidDateInWeekMealInterval(meal).equals("valid")) {
                mealService.save(meal);
                return new ResponseEntity<String>(responseToClient, HttpStatus.OK);
            } else {
                responseToClient = "invalidDateInWeekMealInterval";
                return new ResponseEntity<String>(responseToClient, HttpStatus.OK);
            }

        } else {
            responseToClient = "invalid";
            return new ResponseEntity<String>(responseToClient, HttpStatus.OK);
        }
    }
```

Slika 17 – REST kontroler, create meal

Iznad klase se dodaje anotacija **@RestController**. Preko **@Autowired** anotacije možemo da injektujemo instancu objekta MealService i preko njega da pristupamo njenim metodama.

Kao što sam ranije naveo, sa klijentske strane se u telu zahteva šalje objekat u **JSON** formatu koji je tipa **Meal** klase. Ukoliko dobijeni objekat prođe kroz sve validacije, preko **mealService.save(meal)** metode, uz pomoć **JpaRepository** (Java Persistence API) se čuvaju (upisuju) podaci u bazu podataka.

```
    @Override
    public Meal save(Meal meal) {
        return mealRepository.save(meal);
    }
```

Slika 18 – Upisivanje objekta u bazu podataka preko mealRepository

Kao odgovor, u zavisnosti da li je uspešno upisan prosleđeni objekat, vraća se odgovor klijentu koji je string tipa, nakon čega se obrađuje odgovor i prikazuje poruka korisniku o uspešnom ili neuspešnom upisu.

4.5 Dnevni meni

Kada se korisnik prijavi, automatski se redirektuje na stranicu na kojoj može da pogleda ponudu za danas (dnevni meni), kao i jela koja je neophodno naručiti danas za sutra.

Slika 19 – dnevni meni

Za dizajn tabele i dugmadi sam koristio **Bootstrap** biblioteku. Nakon prijavljivanja korisnika, kada se pokreće **home-page** komponenta, pri inicijalizaciji obaviće se dobavljanje obroka koji su dostupni da se naruče za danas i sutra.

```
getMealsAvailableToOrderForToday(){
  this.mealService.getMealsAvailableToOrderForToday().subscribe(data =>{
    this.mealsAvailableToOrderForToday = data;
  });
}

getMealsAvailableToOrderForTomorrow(){
  this.mealService.getMealsAvailableToOrderForTomorrow().subscribe(data =>{
    this.mealsAvailableToOrderForTomorrow = data;
  });
}
```

Slika 20 – metode za čitanje dnevnog menija

Preko property-ja **mealService**, pozivamo odgovarajuću metodu i pretplaćujemo se pomoću funkcije subscribe. Klijent kao odgovor dobija **Observable** (Observablu) koja se kasnije "castuje" u listu obroka. **Observable** je tok događaja ili podataka. Slušanje toka se zove pretplaćivanje (**subscribing**). Funkcije koje reaguju na događaje se nazivaju observeri. Nakon što dobijemo odgovor sa servera, vrednosti koje smo dobili (**data**) stavljamo u listu koju smo prethodno inicijalizovali (npr **mealsAvailableToOrderForToday**).

```
public List<MealDTO> returnDailyMeals() {
    List<MealDTO> mealsAvailableToOrder = new ArrayList<MealDTO>();
    List<Meal> allMeals = mealRepository.findAll();

    Date today = new Date();
    Calendar cal1 = Calendar.getInstance();
    Calendar cal2 = Calendar.getInstance();
    cal1.setTime(today);

    for (Meal meal : allMeals) {
        cal2.setTime(meal.getDate()); // pravim kalendar od datuma za svaki meal
        if (cal1.get(Calendar.DAY_OF_YEAR) == cal2.get(Calendar.DAY_OF_YEAR)
                && cal1.get(Calendar.YEAR) == cal2.get(Calendar.YEAR) && meal.isReservationDayBefore() == false) {
            MealDTO mealDTO = MealMapper.INSTANCE.entityToDTO(meal);
            mealsAvailableToOrder.add(mealDTO);
        }
    }
    return mealsAvailableToOrder;
}
```

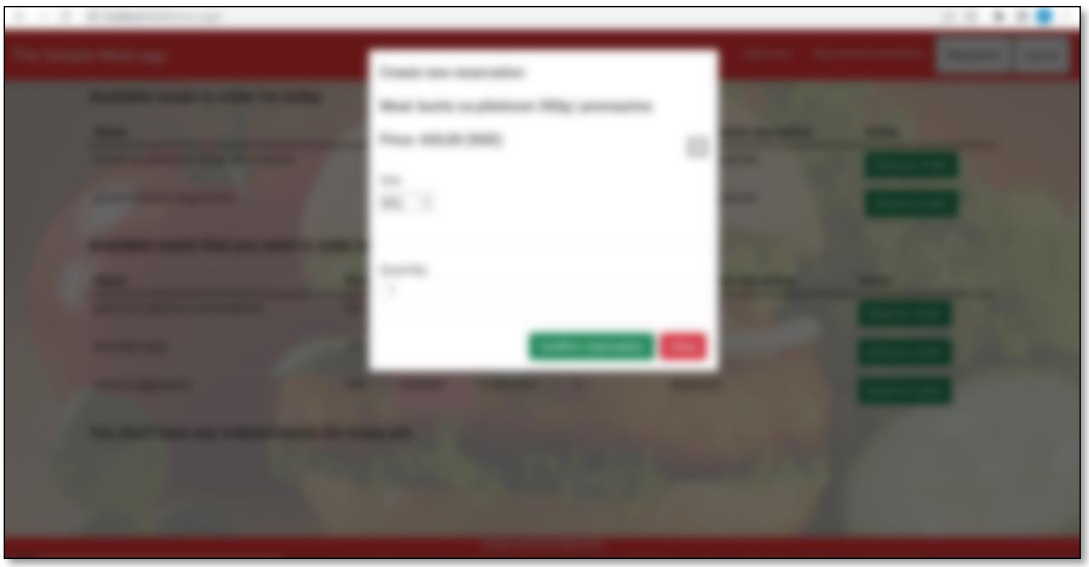
Slika 21 – vraćanje dnevnog menija (samo današnji obroci)

Na slici iznad je prikazana metoda koja vraća sve obroke koji su vezani za današnji datum. Kreiramo dve instance kalendara i jednom setujemo vreme današnjeg datuma koji je prethodno instanciran. Nakon toga prolazimo kroz niz svih obroka. Unutar petlje (niza), drugom kalendaru setujemo vreme trenutnog obroka iz liste.

Nakon toga postoji **if** uslov, gde poredimo da li se godine i dani ova dva kalendara poklapaju. Isto tako proveravamo i da li je **ReservationDayBefore** atribut jednak **false**. Ukoliko se ispune svi uslovi, dodajemo objekat (obrok) u listu koju ćemo posle vratiti klijentu. Sličan princip je i za jela koja treba da se naruče dan ranije i koja se prikazuju korisniku u drugoj tabeli.

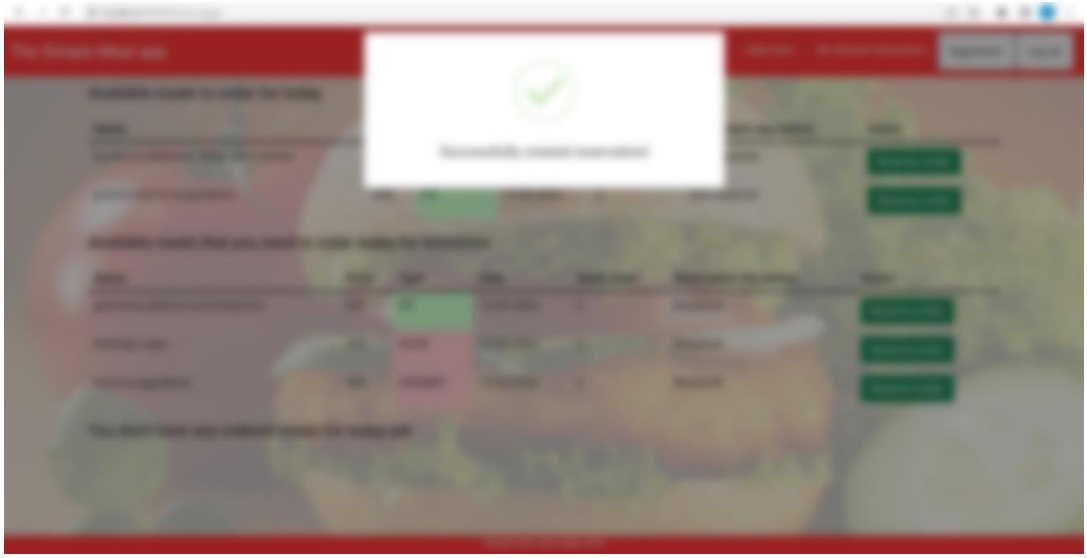
4.6 Rezervisanje obroka

Klikom na dugme “Reserve order”, otvara se modal (dialog) prikazan na slici 22.



Slika 22 – rezervisanje obroka

Ukoliko je korisnik izabrao “**REGULAR**” obrok, njemu će se pojaviti opcija za veliku I malu porciju, dok za ostale tipove obroka ne postoji ta opcija. U polje **Quantity** može da unese broj obroka koji želi da naruči.



Slika 23 – potvrda o uspešnom naručivanju

Klikom na dugme "**Confirm reservation**", korisnik potvrđuje svoju narudžbinu i ukoliko nema nikakvih grešaka prilikom rezervacije, dobija poruku o uspešnom rezervisanju. Za prikazivanje poruke (alert), importovao sam i koristio **Swal** biblioteku.

```
alertSwal(){
  Swal.fire({
    position: 'top',
    icon: 'success',
    title: 'Successfully created reservation!',
    showConfirmButton: false,
    timer: 2000
  });

}
```

Slika 24 – Swal alert

U ovoj metodi unosim naslov koji želim da se ispiše, kao i poziciju i izgled alert-a. Takođe mogu da podesim i vreme trajanja animacije, u ovom slučaju je 2000 ms.

4.7 ChosenOne korisnik

Svakog dana u 10 časova, šalju se porudžbine preduzeću od svih korisnika aplikacije.

```
// @Scheduled(cron = "0 0 10 * * MON-FRI", zone = "Europe/Paris")
@Scheduled(cron = "00 00 10 * * *", zone = "Europe/Paris")
public void sendOrderToCloseOrderingOptionsInViberGroup() {
    FinalOrder fo = createFinalOrder();
    System.out.println("Salje....");
    if (fo == null) {
        System.out.println("There are no reservations for today's status RESERVED!!!");
    } else {

        User chosenOne = fo.getUser(); // The Chosen One je zaduzen za manuelno prosledjivanje u Viber grupu i da
                                       // sakupi novac
        if (chosenOne != null) {

            System.out.println("The Chose One " + chosenOne.getFirstName() + " " + chosenOne.getLastName()
                    + " manually sends to Viber group.");

            String sender = "specijalanimejl@gmail.com";

            Session session = setPtopertiesAndCreateSession(sender);

            try {

                MimeMessage message = new MimeMessage(session);
                message.setFrom(new InternetAddress(sender));
                message.addRecipient(Message.RecipientType.TO, new InternetAddress(chosenOne.getEmail()));
                message.setSubject("Prikupljanje novca za dan " + fo.getDate());
                String str = "Postovani " + chosenOne.getFirstName()
                        + ",\n Vi ste izabrani da danas prikupite novac i prosledite poruku u Viber grupu. \n Srdacan pozdarv :
                String viberMessage = createViberGroupMessage(fo);
                System.out.println("Poruka:\n" + viberMessage);
                message.setText(str + viberMessage);
                Transport.send(message);
                System.out.println("Mail successfully sent");
```

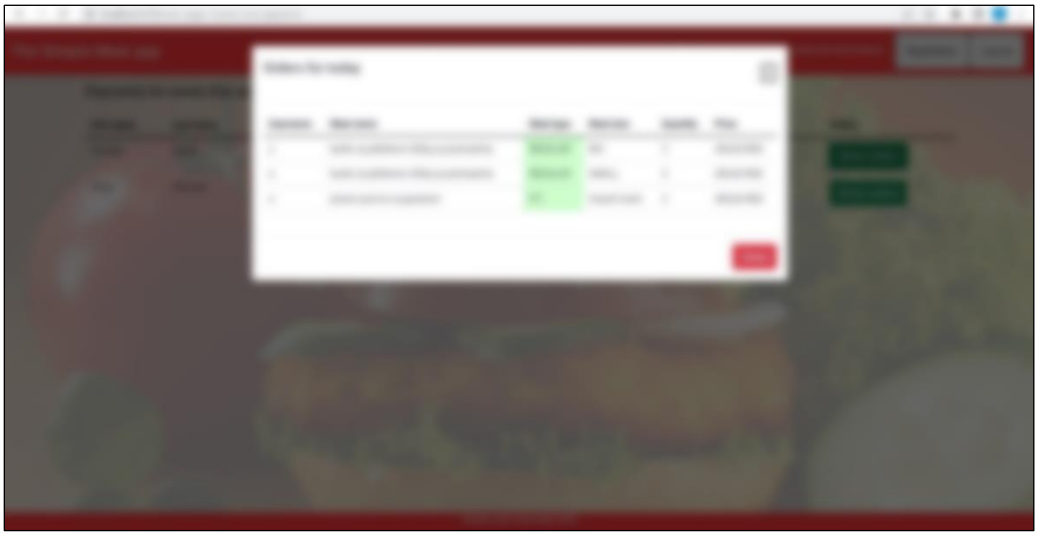
Slika 25 – slanje porudžbina (final order)

Preko anotacije **@Scheduled**, funkcija će se izvršavati svakog radnog dana u setovano vreme (u ovom slučaju 10h). Nakon što se pošalju porudžbine, bira se randomizacijom **chosenOne** korisnik koji je tog dana zadužen za prikupljanje novca. Takođe svakom korisniku koji je naručio hranu tog dana, dobija email poruku sa potvrdom I obrocima koje je naručio. **ChosenOne** korisnik takođe dobija email poruku da je on zadužen za prikupljanje novca.

Slika 26 – chosenOne alert

Sledeći put kada se prijavi, dobija dodatna dva taba. Na **Payments** tabu mu se prikazuju svi korisnici koji su za današnji dan naručili hranu. Pored imena se nalazi i dugme koje se pritisne ukoliko je korisnik doneo pare. Pritiskom na dugme '**Show orders**' prikazuju se obroci koji su naručeni.

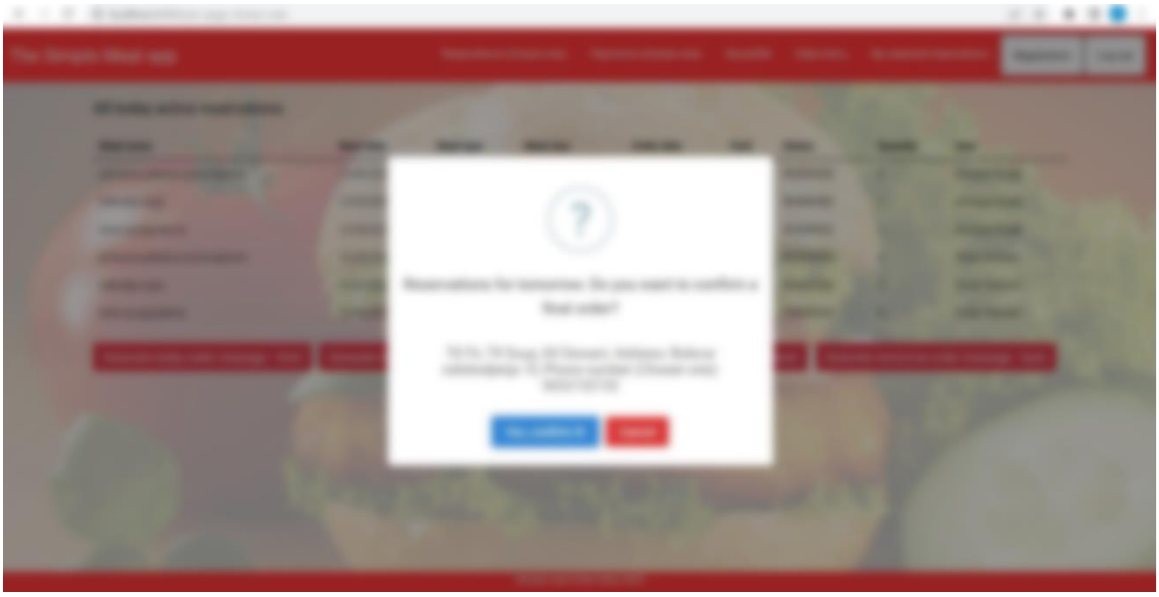
Slika 27 – Payments tab



Slika 28 – Payments tab – show orders

Na **Reservation** tab-u, prikazuju se sve aktivne rezervacije, odnosno one koje jos nisu naručene preduzeću, sa statusom '**reserved'**. **ChosenOne** korisnik ima mogućnost I manuelno da poruči hranu, tako što će generisati poruku I kopirati je u viber grupu. Klikom na dugme '**Yes, confirm it**!', na backend-u će se instancirati **finalOrder** objekat, setovati određeni atributi i upisaće se u bazu podataka. Nismo

stigli da implementiramo Viber bot-a, pa je neophodno manuelno da se kopira poruka i pošalje u Viber grupu.



Slika 29 – Reservations tab, generisanje poruke za narudžbinu

5. Zaključak

Veoma sam zadovoljan praksom, trajala je oko mesec dana. Praksa je bila remote i svakog radnog dana smo imali sastanak oko 13h sa mentorom. Mentor Ime prezime se dobro snašao i uvek je bio spreman da nam odgovori na pitanja i da nas usmeri za dalji rad, iako je i njemu ovo bila prva praksa koju je vodio. U početku smo imali problema sa **git**-om i korišćenjem **SourceTree** alata i nismo baš najbolje bili organizovani oko podele poslova. Na kraju smo se jako dobro uklopili, kolege su bile podjednako aktivne kao i ja. Radio sam frontend deo aplikacije i neke delove na backend delu, koje sam uglavnom radio na početku. Imao sam osnovno znanje za korišćenje Angular radnog okvira, ali sam ga definitivno unapredio tokom ove prakse. Do sada nisam nikada radio **HttpInterceptor**, **AuthGuard** i **RoleGuard** na frontend-u, ali sam ih uz savete mentora uspešno implementirao. Zadatak je bio vrlo dobro osmišljen i imao je dosta komplikovanih delova, koje smo većinom uspešno realizovali. Mentor se trudio da nam objasni i prikaže kako funkcioniše rad na realnim projektima u firmi koliko je mogao. Svideo mi se i rad u timu, imali smo dobru komunikaciju preko **Discord** grupe svakog dana. Nadam se da ću opet imati priliku da učestvujem na nekom ozbiljnijem projektu u skorijoj budućnosti.