

UNIVERZA V LJUBLJANI  
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Luka Horvat

# **Elektronsko naročanje v restavraciji**

DIPLOMSKO DELO

VISOKOŠOLSKI ŠTUDIJSKI PROGRAM  
PRVE STOPNJE  
RAČUNALNIŠTVO IN INFORMATIKA

MENTORICA: doc. dr. Mira Trebar

SOMENTOR: as. dr. David Jelenc

Ljubljana, 2021

COPYRIGHT. Rezultati diplomske naloge so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavo in koriščenje rezultatov diplomske naloge je potrebno pisno privoljenje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

*Besedilo je oblikovano z urejevalnikom besedil  $\text{\LaTeX}$ .*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge: Elektronsko naročanje v restavracij

Besedilo teme diplomskega dela študent prepíše iz študijskega informacijskega sistema, kamor ga je vnesel mentor. V nekaj stavkih bo opisal, kaj pričakuje od kandidatovega diplomskega dela. Kaj so cilji, kakšne metode uporabiti, morda bo zapisal tudi ključno literaturo.



*Na tem mestu zapišite, komu se zahvaljujete za izdelavo diplomske naloge. Pazite, da ne boste koga pozabili. Utegnil vam bo zameriti. Temu se da izogniti tako, da celotno zahvalo izpustite.*



# Kazalo

Povzetek

Abstract

<b>1</b>	<b>Uvod</b>	<b>1</b>
<b>2</b>	<b>Načrtovanje in razvoj aplikacije</b>	<b>3</b>
2.1	Podatkovna baza . . . . .	5
2.2	Strežnik . . . . .	7
2.3	Odjemalec . . . . .	12
<b>3</b>	<b>Delovanje aplikacije</b>	<b>17</b>
3.1	Vmesnik za gosta . . . . .	17
3.2	Vmesnik za natakarja in kuharja . . . . .	20
<b>4</b>	<b>Diskusija</b>	<b>25</b>
4.1	Implementacija programa v realnosti . . . . .	25
4.2	Izboljšave . . . . .	26
4.3	Konkurenca . . . . .	26
<b>5</b>	<b>Sklepne ugotovitve</b>	<b>27</b>
	<b>Literatura</b>	<b>29</b>





# Seznam uporabljenih kratic

kratica	angleško	slovensko
<b>SPA</b>	single page application	aplikacija na eni strani
<b>SQL</b>	structured query language	strukturirani povpraševalni jezik za delo s podatkovnimi bazami
<b>CLI</b>	command-line interface	znakovni uporabniški vmesnik
<b>REST</b>	representational state transfer	aktualni prenos stanja
<b>URI</b>	uniform resource identifier	enotni identifikator vira
<b>SPA</b>	single-page application	aplikacija na eni strani
<b>SUPB</b>	database management system	sistem za upravljanje s podatkovno bazo



# Povzetek

**Naslov:** Elektronsko naročanje v restavraciji

**Avtor:** Luka Horvat

Pride na koncu.

**Ključne besede:** Slovenija, naročanje, neuspešni projekti, rešitev, spletna aplikacija.



# Abstract

**Title:** Diploma thesis sample

**Author:** Luka Horvat

This sample document presents an approach to typesetting your BSc thesis using L<sup>A</sup>T<sub>E</sub>X. A proper abstract should contain around 100 words which makes this one way too short.

**Keywords:** computer, computer, computer.



# Poglavje 1

## Uvod

Slovenija velja za državo z veliko število restavracij, vendar le malo iz med njih uporablja napredne sisteme naročanja kot npr. ena izmed večjih verig s hitro prehrano, McDonalds. V Sloveniji je bilo nekaj projektov s podobnimi idejami, vendar z napačnimi cilji zaradi katerih so bili neuspešni. Eden izmed razlogov da jim ni uspelo je bilo sabotiranje sistemov s strani natakarjev, saj so misli da bo tehnologija zamenjala njegove službe. V zavedanju teh problematik smo se odločil narediti diplomski nalogo na to temo.

Zamislili smo si sistem za oddajanje naročil v restavracijah, ki ne bi bil namenjen zamenjavi ljudi v strežbi, temveč kot pregledovalnik (angl. Menu) oziroma naročanju hrane in pijače. Aplikacija za stranke bi bila na tablicah, ki bi bile locirane na vsaki mizi restavracije. Stranka bi bila tista, ki bi se odločila ali želi pri naročanju uporabiti stik z osebo v strežbi ali bi naročila z uporabo aplikacije na tablici. Natakar bi tako imel več časa, katerega bi lahko posvetil pripravi pijače, kvaliteti postrežbe in ostalih dolžnosti. Tudi stranke, katere sedaj veljajo za bolj zahtevne in nečakane na vseh področij, bi bile hitreje in bolj kvalitetno postrežene. Tako bi imeli poleg restavracij z hitro prehrano tudi restavracije s hitro postrežbo.

Aplikacija podpira tri uporabniške vloge, in sicer gost, natakar in kuhar.





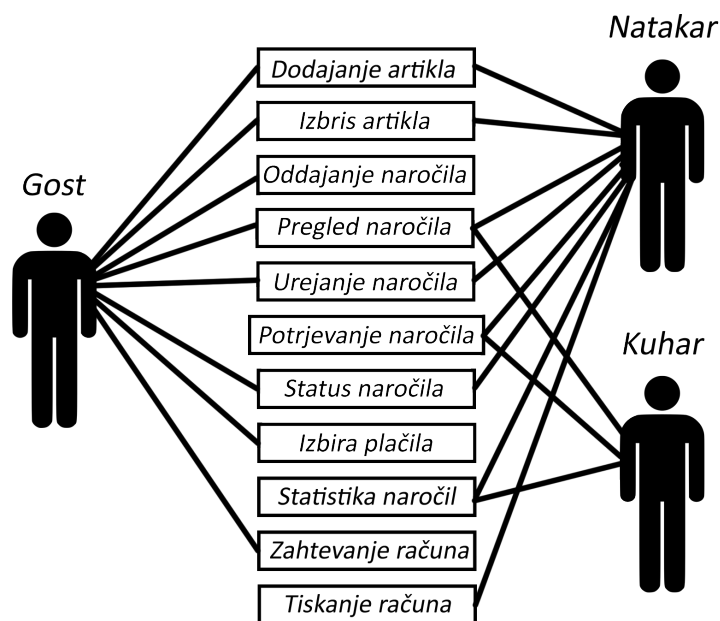
## Poglavje 2

# Načrtovanje in razvoj aplikacije

Razvoj aplikacije je potekal v treh delih in sicer analiza zahtev oziroma izdelava diagrama primerov uporabe, zamisel strukture in implementacija. Rešitev sestavljajo dve aplikaciji implementirani s pomočjo spletnih tehnologij. Ena aplikacija je namenjena gostom, druga pa natakarjem in kuharjem. Restavracija ima lahko več miz, vendar za eno mizo je lahko hkrati odprto eno naročilo. To pomeni, da morajo za mizo naročiti skupaj.

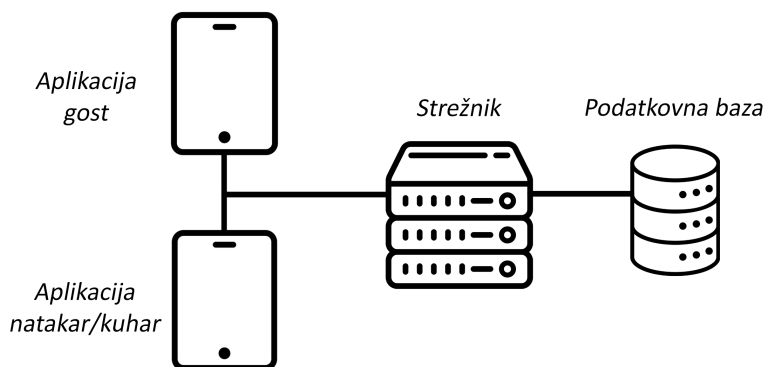
Gost lahko odda, spremeni ali zaključi naročilo. Funkcionalnosti gosta so: dodajanje in brisanje artiklov iz naročila, oddajanje, urejanje in pregledovanje naročila, spremljanje statusa naročila in možnost za zahtevanje računa. Natakar lahko sprejme, zavrne, uredi ali zaključi naročilo. V restavraciji je lahko več natakarjev, ki streže istočasno. Funkcionalnosti natakarja so: dodajanje in brisanje artiklov iz naročila, urejanje in pregledovanje naročila, spremljanje statusa naročila, spremljanje statistike naročil in možnost tiskanja računa za določeno naročilo. Kuhar lahko sprejme, zavrne ali sporoči, da je jed za neko naročilo že pripravljena. Restavracija ima lahko več kuharjev, vendar samo en kuhar hkrati lahko uporablja aplikacijo. Omejitev je v podatkovnem modelu. Funkcionalnosti kuharja so: pregledovanje in potrjevanje naročila ter spremljanje statistike naročil.

Iz analize zahtev smo za lažji pregled nad vsemi funkcionalnostmi izdelali diagram primerov uporabe, ki ga prikazuje slika 2.1.



Slika 2.1: Diagram primerov uporabe spretnega naročanja

Uporabili smo strukturo novodobnih aplikacij katero sestavljajo podatkovna baza, strežnik in aplikacija oziroma odjemalec. Gre za koncept, ki ga je moč prilagajati predvsem z uporabniškega vidika. Slika 2.2 prikazuje strukturo aplikacije. Podatkovna baza je namenjena shranjevanju vseh podatkov za posamezno restavracijo. Strežnik implementira RESTful vmesnik, ki odjemalcu oziroma aplikaciji posreduje podatke iz podatkovne baze.

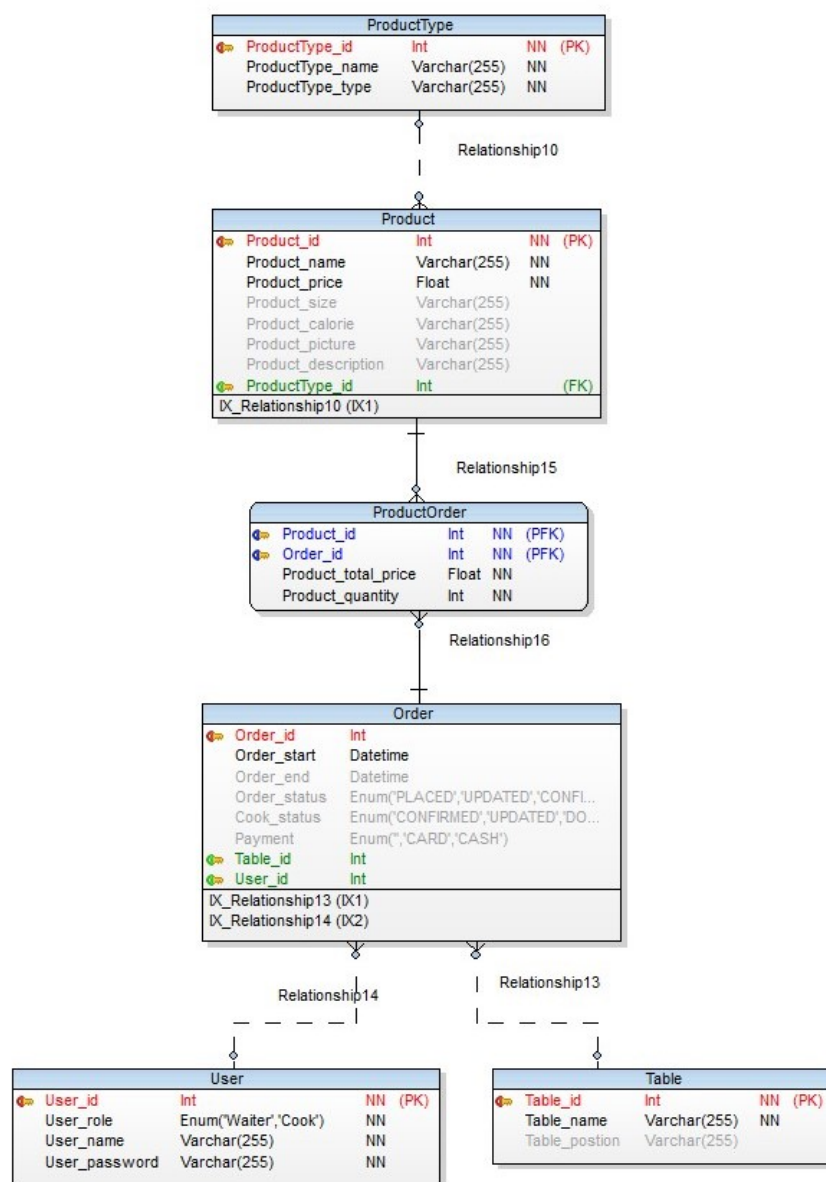


Slika 2.2: Visokonivojska arhitektura

## 2.1 Podatkovna baza

Na podlagi izdelanega diagrama zahtev oziroma funkcij smo najprej izdelali logičen podatkovni model (slika 2.3). Podatkovna baza je sestavljena je iz šestih tabel, ki so: *ProductType*, *Product*, *ProductOrder*, *Order*, *User*, *Table*.

[1,1]



Slika 2.3: Logični podatkovni model

*ProductType* tabela je namenjena zapisom za vrste jedi (predjed, glavna jed, sladica) ali pijač (sokovi, piva, vina, ...). Sestavljena je iz atributov ID, Name in Type. Atribut Type je namenjen razlikovanju hrane in pijače. Uporabljen je za razločevanje artiklov v naročilu, saj kuhar ne potrebuje pregleda nad naročili, ki vsebujejo samo pijačo.

*Product* tabela je namenjena opisu hrane in pijače ter je sestavljena iz atributov: ID, Name, Price, Size, Calorie, Picture in Description. V atribut Picture se zapiše ime slike, ki se prikaže v aplikaciji. Vse slike smo shranjevali v datotečnem sistemu spletnega strežnika.

*ProductOrder* tabela je namenjena količini in končni ceni vsake hrane in pijače v naročilu. Sestavljena je iz atributov: TotalPrice in Quantity. Zapis ne more obstajati če nima definiranega naročila. Tabela je nastala zaradi razmerja M:N t.i. mnogo-proti-mnogo med tabelo Product in Order.

*Order* tabela je namenjena zapisovanju naročila in njegovih podrobnosti. Sestavljena je iz atributov: ID, Start, End, OrderStatus, CookStatus, Payment. Vsebuje tudi tuji ključ ID od tabele Table, ki določa, na katero mizo je vezano naročilo. Atributa OrderStatus in CookStatus sta nemanjena sporočanju statusa naročila med vlogami. Uporabil sem ENUM za vrsto parametra, saj gre za vrednosti, ki se ne spreminjajo.

*Table* tabela je namenjena označevanju miz v restavracijah. Sestavljajo jo atributi: ID, Name in Position, v katerega se lahko bolj podrobno opiše lokacijo mize.

*User* tabela je namenjena predvsem aplikativnemu delu za natakarje in kuharje. Sestavljena je iz atributov: ID, Role, Name in Password.

Strukturo podatkovne baze smo naredili s pomočjo programa Toad DataModeler. To je orodje za izdelavo visokokakovostnih podatkovnih modelov [3]. Omogoča izdelavo logičnih in fizičnih podatkovnih modelov, kar pripomore k lažjemu razumevanju in razvijanju podatkovne baze. Njegova najboljša funkcionalnost je, da lahko generiramo SQL kodo v različne podatkovne sisteme kot npr. MySQL, Ingres, Microsoft Azurem, Microsoft Access in Microsoft SQL Server.

Podatkovni model smo implementirali na fizičnem nivoju v SUPB (angl. database management system oz. DBMS) MySQL. To je eden od odprtokodnih sistemov za upravljanje s podatkovni bazami, ki za delo s podatki uporablja jezik SQL [6]. Napisan je v programskem jeziku C in C++ in deluje v vseh modernih operacijskih sistemih (Windows, Linux, IOS in drugih).

## 2.2 Strežnik

Strežnik predstavlja vmesnik med podatkovno bazo in odjemalcem. Najbolj pomembno nam je bilo, da je sistem zanesljiv, saj brez njega odjemalec ne more delovati. Izbrali smo arhitekturo REST zaradi načel, ki so opisana spodaj [7]. Sama arhitektura omogoča, da odjemalec s pomočjo zahtev pridobiva podatke od strežnika, kateri jih s pomočjo URI povezav oglašuje na relativnih povezavah. Strežnik smo napisali v programskem jeziku Python z vključitvijo knjižnic Flask, MySQL, SocketIO in CORS, nameni le-teh so bolj podrobno opisani spodaj.

Načela arhitekture REST so sledeča:

1.) *Odjemalec-strežnik* (ang. *Client-server*) zahteva ločitev odjemalca od strežnika kar onemogoča odjemalcu direktno povezljivost s podatkovno bazo in s tem poenostavi razširljivost uporabniškega dela. Strežnik ne zanaša uporabniški vmesnik ali podatki, tako da je bolj enostaven in prilagodljiv za uporabo. Tako se lahko uporabniški kot strežniški del razvija ali zamenjuje neodvisno. V aplikaciji imamo dva različna odjemalca (gost in natakar/kuhar), kar pa za strežnik ne predstavlja nobenih omejitev razen na strani podatkovne baze, ki omejuje število hkratnih poizvedb. V naši aplikaciji te omejitve nismo presegli.

2.) *Brez stanja* (ang. *Stateless*) morajo biti vse interakcije med strežnikom in odjemalcem. Strežnik ne sme shranjevati nobenih stanj oziroma more vsako zahtevo od odjemalca tretirati kot popolnoma novo. V programski kodi našega strežnika je dobro razvidno, da ne uporabljamo nobenih globalnih spremenljivk. Vsi podatki, ki so potrebni, da na zahtevo HTTP odgovorimo,

rimo, se nahajajo bodisi v podatkovni bazi bodisi jih odjemalec na strežnik posreduje z zahtevkom.

3.) *Predpomnjenje* (ang. *Cachable*) prinaša izboljšanje zmogljivosti na strani odjemalca in boljši obseg razširljivosti strežnika, ker se obremenitev zmanjša. V REST aplikacijah se predpomnjenje uporabi za vire, ki to potrebujejo. Sami tega nismo uporabili, saj nimamo tako zahtevnih virov.

4.) *Večslojni sistem* (ang. *Layered system*) je sestavljen iz hierarhičnih slojev kjer npr. za API vmesnik uporabimo strežnik A, za shranjevanje podatkov strežnik B ter strežnik C za avtenticanje zahtev. S tem odjemalec ne more ugotoviti ali komunicira s končnim strežnikom ali s posrednikom.

5.) *Izvajanje programske kode na zahtevo* (ang. *Code on demand*) je opcijsko načelo. Strežnik na zahtevo odjemalca pošlje oziroma izvede programsko kodo na strani odjemalca. To smo vpeljali s pomočjo spletnih vtičnikov (websocket), kjer so bolj podrobno opisani spodaj.

6.) *Enotni vmesnik* (ang. *Uniform interface*) (API) med strežnikom in odjemalcem. Vsak vir mora vsebovati povezavo (HATEOAS), ki kaže na svoj relativen URI. Odjemalec te vire pridobi od strežnika v obliki zahtev, ki so lahko GET, POST, PUT ali DELETE. Za predstavitev virov se lahko uporabi poljuben format, vendar najbolj pogosta sta XML in JSON.

Zato zahtevo nam je knjižnica Flask zelo poenostavila izdelavo strežnika. Flask je eno izmed najbolj popularnih spletno aplikacijskih vmesnikov (angl. Framework) [9]. Zasnovan je tako, da omogoča hiter in enostaven začetek z možnostjo razširitve na zapletene aplikacije. Flask je prvotno zasnoval in razvil Armin Ronacher kot prvoaprilsko šalo leta 2010. Kljub taki predstavitvi je Flask postal izjemno priljubljen kot alternativa projektom narejenih v Django.

Vsaka relativna povezava URI na strežniku predstavlja svoje vir podatkov iz podatkovne baze. Te podatki so odjemalcu na voljo v JSON podatkovnem formatu. Strežnik s pomočjo knjižnice MySQL najprej prebere podatke iz podatkovne baze in jih predstavi na določeni relativni povezavi URI, ki jo določimo mi. Na sliki 2.4 je prikazana funkcija za branje podatkov iz

podatkovne baze, kjer spemenljivka query predstavlja poizvedbeni stavek v podatkovni bazi. Slika 2.5 prikazuje uporabo te funkcije in relativne poti drinks. Naš strežnik vsebuje 34 relativni povezav in 600 vrstic programske kode.

```
def SQLqueryProduct(query):  
    mycursor = mydb.get_db().cursor()  
    myresult = mycursor.execute(query)  
    myresult = mycursor.fetchall()  
    mycursor.close()  
    payload = []  
    content = {}  
    for result in myresult:  
        content = {  
            'product_id': result[0],  
            'name': result[1],  
            'price': result[2],  
            'size': result[3],  
            'calorie': result[4],  
            'picture': result[5],  
            'description': result[6],  
            'type_id': result[7]  
        }  
        payload.append(content)  
        content = {}  
    return payload
```

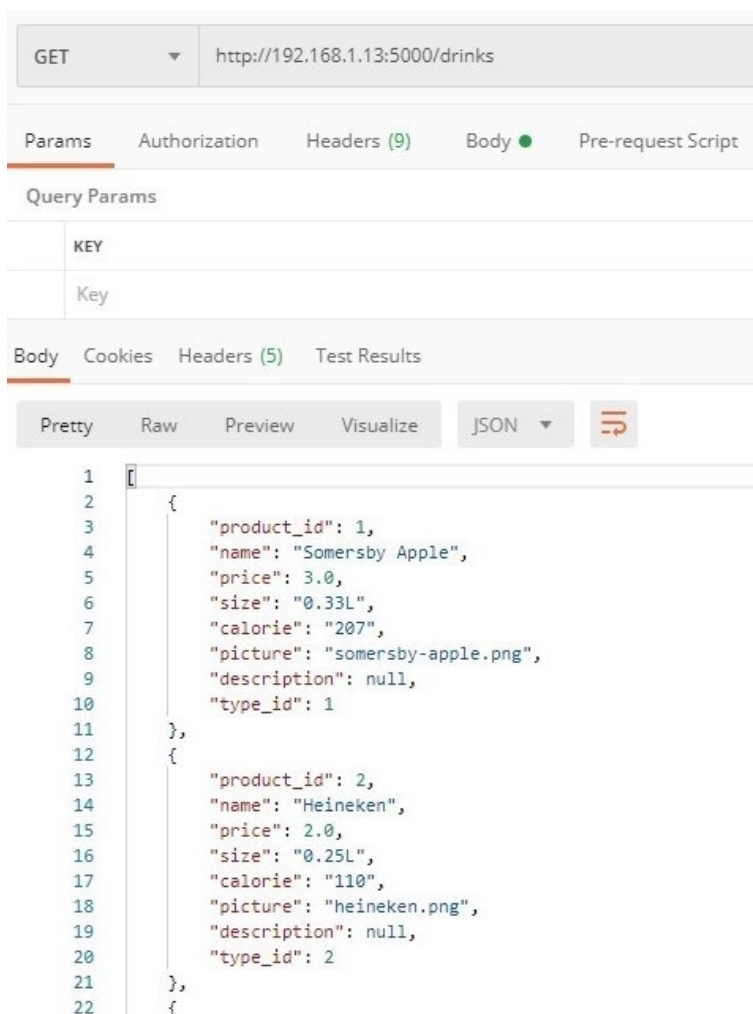
Slika 2.4: Funkcija, ki prebere podatke iz podatkovne baze

```
@app.route('/drinks')  
def allDrinks():  
    return jsonify(SQLqueryProduct("SELECT * FROM product, producttype WHERE \  
product.ProductType_id=producttype.ProductType_id AND producttype.ProductType_type='Drink'"))
```

Slika 2.5: Funkcija, ki na relativno stran /drinks oglasi podatke

Tako smo dobili vmesnik, ki na zahtevo odjemalca odgovori s podatki v JSON formatu. Na sliki 2.6 je primer testiranja zahtevka v programu Postman, ko odjemalec zahteva podatke vseh pijač iz podatkovne baze (HTTP metoda GET). Strežnik omogoča tudi sprejemanje podatkov za kar se upo-

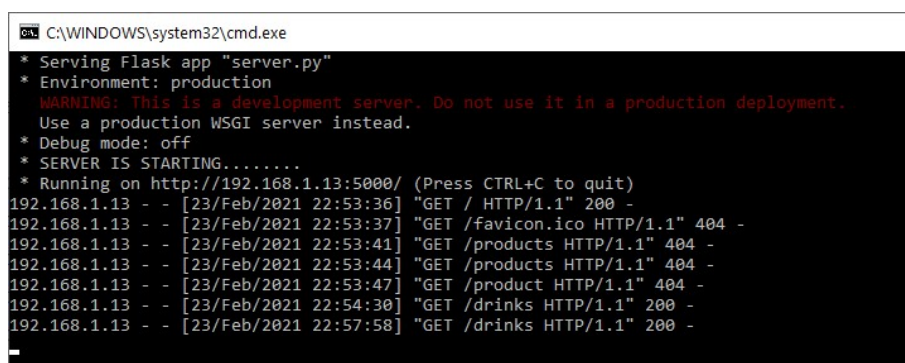
rabljati metodi PUT in POST. Mi smo uporabljali metodo POST za posredovanju podatkov strežniku, ki so bili potrebni za zapis v podatkovno bazo. Spremljanje zahtevkov na strani strežnika je mogoče v konzolnem vmesniku CLI, ki se ga uporablja pri zagonu strežnika, slika 2.7.



Slika 2.6: Primer serviranja podatkov na strežniku s programom Postman

Za potrebe pridobivanja podatkov v realnem času na strani odjemalca, smo potrebovali še spletni vtičnik SocketIO. Implementirali smo ga na strani strežnika in odjemalca, ki zagotavlja dvosmerno komunikacijo oziroma komunikacijo na podlagi dogodkov. Deluje na vseh platformah, brskalnikih





```
C:\WINDOWS\system32\cmd.exe
* Serving Flask app "server.py"
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* SERVER IS STARTING.....
* Running on http://192.168.1.13:5000/ (Press CTRL+C to quit)
192.168.1.13 - - [23/Feb/2021 22:53:36] "GET / HTTP/1.1" 200 -
192.168.1.13 - - [23/Feb/2021 22:53:37] "GET /favicon.ico HTTP/1.1" 404 -
192.168.1.13 - - [23/Feb/2021 22:53:41] "GET /products HTTP/1.1" 404 -
192.168.1.13 - - [23/Feb/2021 22:53:44] "GET /products HTTP/1.1" 404 -
192.168.1.13 - - [23/Feb/2021 22:53:47] "GET /product HTTP/1.1" 404 -
192.168.1.13 - - [23/Feb/2021 22:54:30] "GET /drinks HTTP/1.1" 200 -
192.168.1.13 - - [23/Feb/2021 22:57:58] "GET /drinks HTTP/1.1" 200 -
```

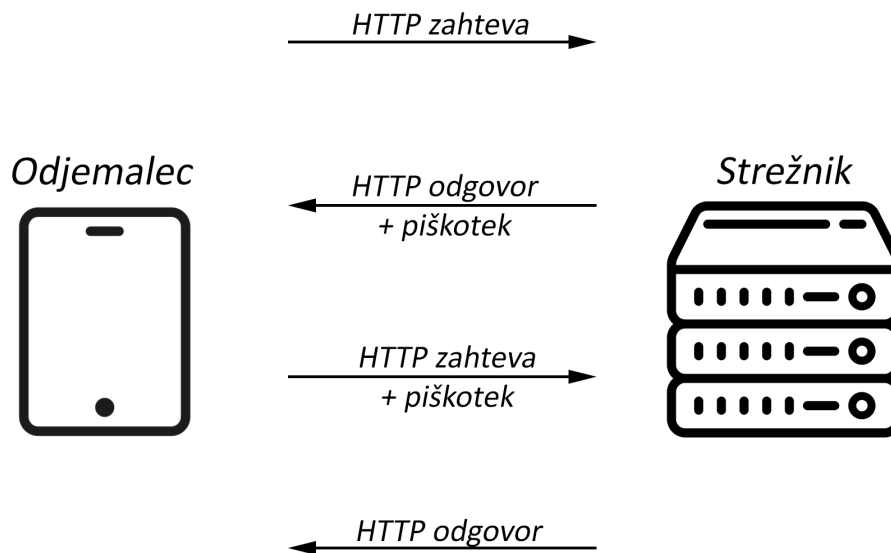
Slika 2.7: Primer spremljanja zahtevkov, ki prihajajo na strežnik

ali napravah. Uporabili smo ga za obveščanje odjemalcov o spremembah v podatkovni bazi. Za to smo uporabljali funkcijo „emit“, ki pomeni oddajanje. Omogoča dodajanje podatkov in izbiranje načina razpršenega oddajanja (ang. broadcast). To pomeni, da vsi prejemaajo te informacije ob oddajanju na strani strežnika. Gre predvsem za splošne podatke, tako da ne more priti do zlorabe. Npr. ob spremembi naročila na strani gosta se le te razlike preverijo na strežniku in vpišejo v podatkovno bazo ter o tem obvesti natakarja s funkcijo „emit“, ki vsebuje številko naročila v katerem je prišlo do sprememb.

Aplikacija omogoča urejanje določenih podatkov, zato smo morali zagotoviti ustrezno varnost za urejanje le-teh. Prvi nivo varnosti je avtorizacija odjemalca. Zahtevki, ki se pošiljajo na strežnik morajo biti omogočeni samo avtoriziranim odjemalcem, da ne pride do napadov kot npr. s posrednikom (ang. man-in-the-middle). Zato smo uporabili HTTP piškotke (ang. cookies), ki so izdelani za spletne brskalnike in so namenjeni za sledenje, prilagajanje in shranjevanje informacij o posamezni seji uporabnika. Vsi piškotki so shranjeni na strani odjemalca in so kriptografsko zaščiteni pred morebitnimi neopoblaščenimi posegi odjemalca. S tem odjemalcu preprečujemo spreminjanje podatkov v piškotkih oz. lahko nedovoljene spremembe podatkov na strežniku delectiramo. Uporabili smo Flask-Login, ki omogoča vse funkcionalnost za upravljanje uporabniških sej.

Na sliki 2.8 je prikazano delovanje HTTP piškotkov. Strežnik ob prvem

zahtevku, torej ob uspešni prijavi, odjemalcu vrne piškotek. Odjemalec ob vsakem nadaljnjem zahtevku doda piškotek s katerim strežnik overi zahtevo odjemalca.



Slika 2.8: Delovanje HTTP piškotkov med odjemalcem in strežnikom

## 2.3 Odjemalec

Odjemalca bi načeloma lahko implementirali v spletnih tehnologijah (HTML, CSS, JS) ali pa v namenski mobilni aplikaciji. Mi smo se odločili za knjižnico in ogrodje Vue, ki je implementirano v programskem jeziku JavaScript. Naredili smo odziven in reaktiven vmesnik, ki deluje v realnem času. Vue je eden izmed mnogih kot npr. Angular, Ember, React, ... poznan pa je predvsem zaradi enostavnosti za upravljanje in izvajanje testov. Vsem skupna reaktivnost, vendar v drugačnem pomenu besede. Reaktivnost [2], je programska paradigma, ki nam omogoča, da se na deklarativni način prilagodimo spremembam. Tako deluje tudi reaktivnost v aplikacijah za razliko, da je podatek lahko vezan na več funkciji oziroma delov programske kode, ki se ob spre-

membi vrednosti posodobijo. Vue je namenjen izdelavi SPA projektov, saj vsebuje samo eno datoteko HTML. To prednost smo izkoristili s pomočjo ostalih knjižnic, ki so nam olajšale izdelavo aplikacije. Uporabili smo naslednje:

**Vue CLI** velja kot standardno orodje za ekosistem Vue [4]. Zagotavlja, da že pri gradnji novega projekta poveže različne dodatke med seboj. To omogoča razviljaku, da se bolj osredotoči na programiranje in ne na povezovanje njih v projekt. Zadeva izgleda nekako tako, da preko CLI vmesnika izbereš kakšen projekt želiš. Imaš seveda že privzete nastavitve, vendar omogoča tudi nastavljanje po meri. Sam sem uporabil Vuex, Vue-Router, ESLint in Vuetify.

**Vuex** je knjižnica za shranjevanje vrednosti v aplikacijah Vue.js [8]. Služi kot centralizirana baza podatkov za vse komponente v aplikaciji.

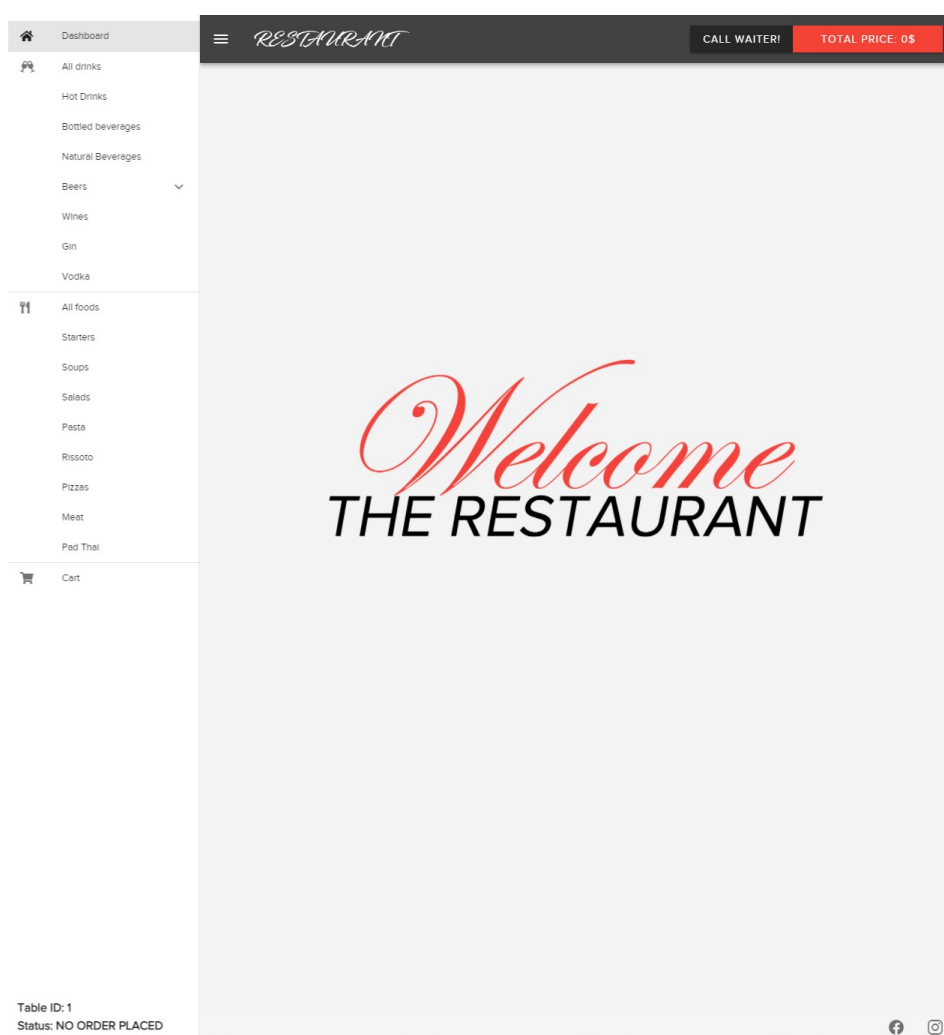
**Vue-Router** je uradni usmerjevalnik za Vue.js [5]. Integrira se globoko z jedrom Vue.js, tako da poenostavi izdelavo SPA aplikacij. Usmerjevalki je mišljen v smislu usmerjanja na druge komponente (angl. Component), ki v Vue.js predstavljajo druge poglede, lahko bi rekli podobno kot podstrani.

**ESLint** je orodje za prepoznavanje in poročanje o popravkih v programski kodi [1]. Cilj je narediti kodo bolj pregledno in urejeno, kar pripomore k izogibanju napak.

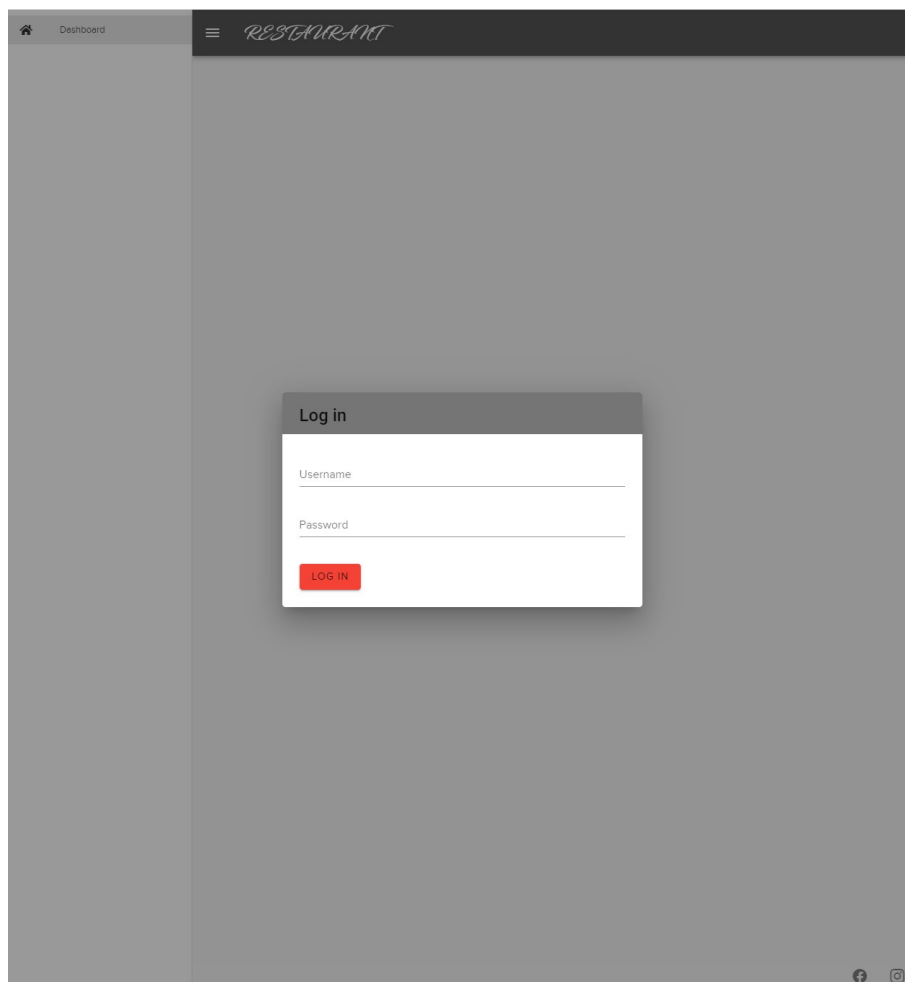
**Vuetify** je eden izmed mnogih uporabniških vmesnikov, ki je zgrajen na vrhu Vue.js [10]. Za razliko od drugih vmesnikov je Vuetiy enostaven za učenje z več stotimi komponentami izdelanih po specifikacijah Material Design.

**Vue-devtools** je zgolj dodatek v brskalniku, ki omogoča lažje sledenje delovanja aplikacije in odpravljanju napak.

Odjemalca smo razdelili v tri vloge oziroma dve aplikaciji. Ena aplikacija je namenjena natakarjem in kuharjem, ki se ločuje s prijavnim oknom in izgledom vmesnika. Druga aplikacija je namenjena samo gostom ter je sestavljena iz večih pogledov. Ločili smo jih zaradi varnosti, lažjega razvijanja in preglednosti, saj gre za dve popolnoma različni aplikaciji. Vse funkcionalnosti in delovanje ene in druge aplikacije so opisane v naslednjem poglavju. Sliki 2.9 in 2.10 prikazujeta prvi izgled obeh aplikacij.



Slika 2.9: Spletni vmesnik za gosta



Slika 2.10: Spletni vmesnik za natakarja in kuharja

Ena izmed pomembnih stvari pri obratovanju restavracije je čim hitrejša postrežba katero je mogoče izboljšati s čim hitrejšo komunikacijo. Zato smo, enako kot na strani strežnika, uporabili spletni vtičnik SocketIO. To smo vključili v obeh aplikacijah, in sicer za oddajanje naročil, posodabljanje naročil, obveščanje gosta o stanju naročila, ... Najprej smo hoteli uporabiti samodejno osveževanje na določen časovni interval, vednar je uporaba spletnih vtičnikov hitrejša in učinkovitejša. Slika 2.3 prikazuje seznam vseh vtičnikov, ki so uporabljeni na strani gosta.

```

},
SOCKET_CONFIRMED({ commit, state }, payload) {
  if (payload == state.orderID)
    commit("SET_ORDER_STATUS", "CONFIRMED BY WAITER");
},
SOCKET_SERVED({ commit, state }, payload) {
  if (payload == state.orderID) commit("SET_ORDER_STATUS", "SERVED");
},
SOCKET_CALLING_WAITER({ commit, state }, payload) {
  if (payload == state.orderID)
    commit("SET_ORDER_STATUS", "CALLING WAITER");
},
SOCKET_orderChanged({ commit, dispatch, state }, payload) {
  if (payload == state.orderID) {
    commit("SET_ORDER_STATUS", "CHANGED BY WAITER");
    dispatch("checkOrder");
  }
},
SOCKET_orderEnd({ commit, state }, payload) {
  console.log("orderEND!!!");
  if (payload == state.orderID) {
    commit("CLEAR_ALL");
    commit("SET_ORDER_STATUS", "ORDER_END");
    state.orderID = null;
    state.orderEnd = false;
  }
}
}

```

Slika 2.11: Seznam vtičnikov, ki so uporabljeni na strani gosta

Za pridobivanje podatkov na strani odjemalca smo uporabili Axios, ki je namenjen procesiranju zahtevkov HTTP. To pomeni, da podatke, ki se oglašujejo na strani strežnika s pomočjo te knjižnice pridobimo na stran odjemalca.

```

allDrinks({ commit }) {
  axios.get(`${"http://192.168.1.13:5000"}/drinks`).then(response => {
    commit("ALL_DRINKS", response.data);
  });
},
allFoods({ commit }) {
  axios.get(`${"http://192.168.1.13:5000"}/foods`).then(response => {
    commit("ALL_FOODS", response.data);
  });
},

```

Slika 2.12: Način uporabe Axios v aplikaciji za gosta

## Poglavje 3

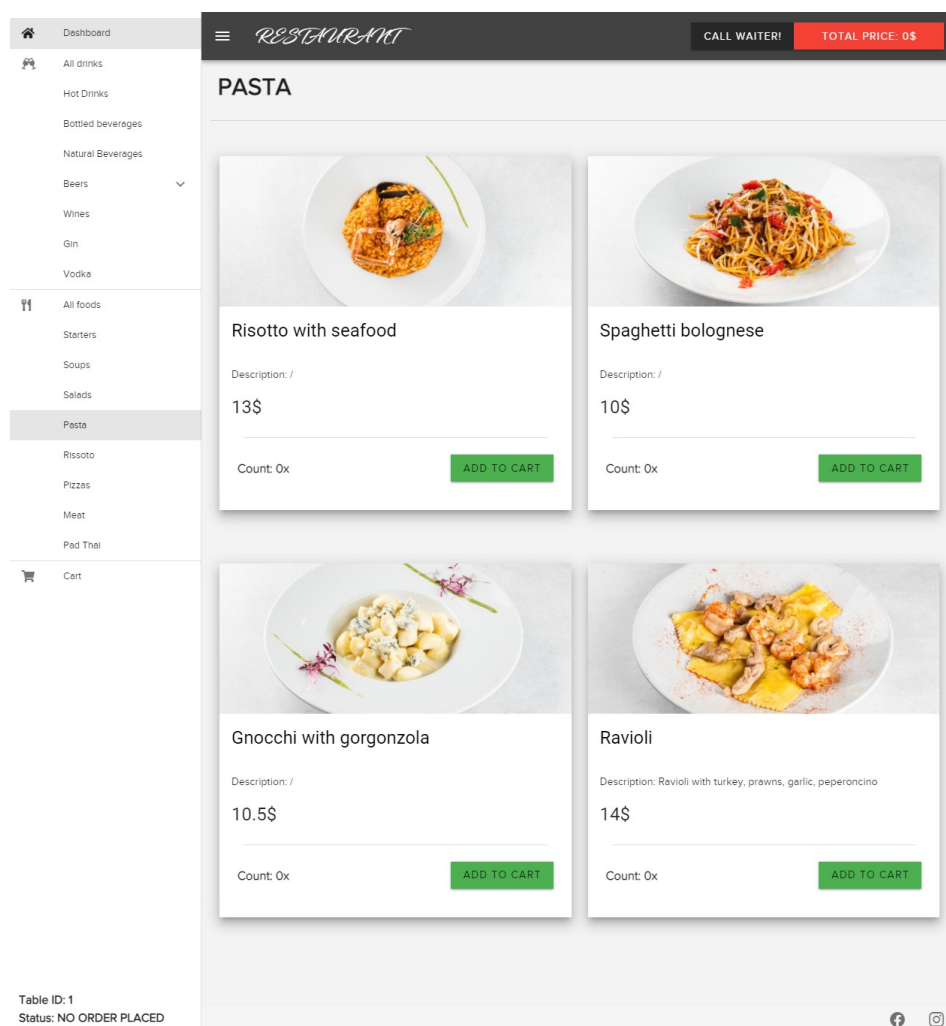
# Delovanje aplikacije

Za prikaz delovanja smo si pripravili testno okolje, kjer smo znotraj lokalnega omrežja postavili strežnik in obe aplikaciji. Vse funkcionalosti posamezne aplikacije so opisane spodaj.

### 3.1 Vmesnik za gosta

Prvi pogled vmesnika za gosta vsebuje napis za dobrodošlico, slika 2.9, kateri bi lahko zamenjalo oglaševanja, predstavitev restavracije ali karkoli bi si potencialni kupec zaželel. V zgornjem desnem kotu se nahaja gumb call waiter za klic natakarja in števec skupne cene artiklov v nakupovalni košarici. Tipka call waiter je namenjena gostom, ki aplikacije ne želijo uporabljati ali v primeru kakršnihkoli težav. V spodnjem levem kotu se nahaja status in identifikacijska številka naročila. Zavihek na levi strani je pravzaprav seznam vseh vrst hrane in pijače, ki kažejo na podstrani z artikli, slika 3.1. To gostu predstavlja jedilnik s katerim v nakupovalno košarico dodaja, briše artikle ali spreminja njihovo količino. Nakupovalna košarica (ang. cart) je skupno mesto vseh artiklov potencialnih za naročilo, slika 3.2.

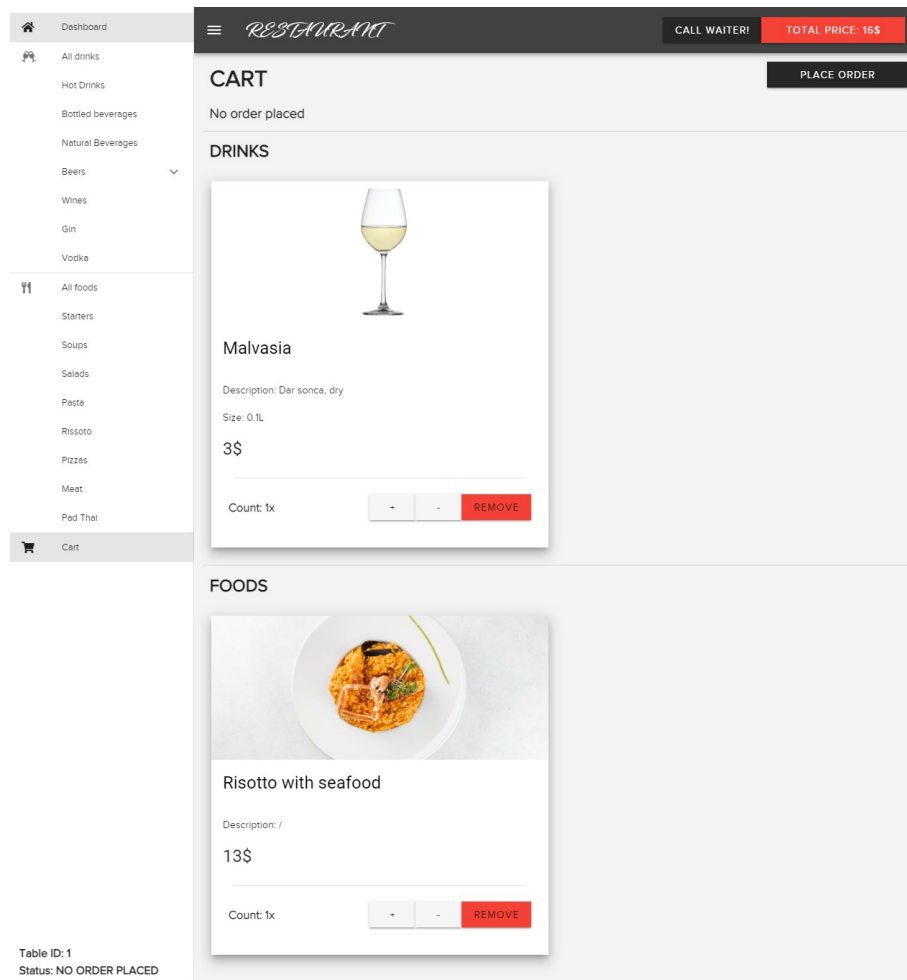
Naročilo se odda s klikom na oddaj naročilo (ang. place order), ki ob kliku gosta preusmeri na prvo stran in izpiše pojavno sporočilo prikazano na sliki 3.3. Ko je naročilo oddano lahko gost ponovno dodaja artikle v košarico,



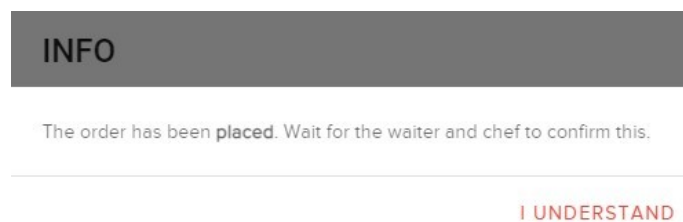
Slika 3.1: Seznam artiklov znotraj vrste špagetov

vendar je že oddanim artiklom onemogočeno zmanjšati količino ali jih izbri-  
sati. Naročilo je sprejeto ko ga natakar potrdi, s tem se spremeni statusa  
naročila in prikaže se pojavno sporočilo prikazano na sliki 3.4. V primeru, da  
natakar zavrne naročilo, ga mora gost pregledati in ponovno oddati. Tudi v  
tem primeru je gost obveščen s statusom in pojavnim sporočilom, slika 3.5.  
Naročilo se zaključi z klikom na gumb zahtevaj račun (ang. request receipt),  
ki odpre pojavno okno prikazano na sliki 3.6, kjer je potrebno izbrati način  
plačila (gotovina, kartica).

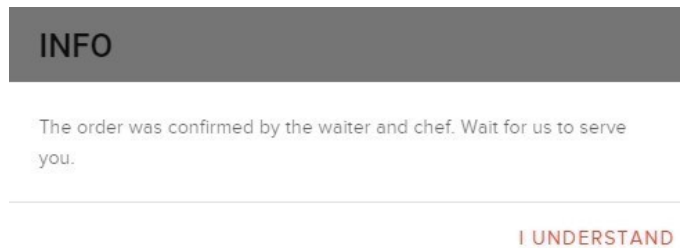




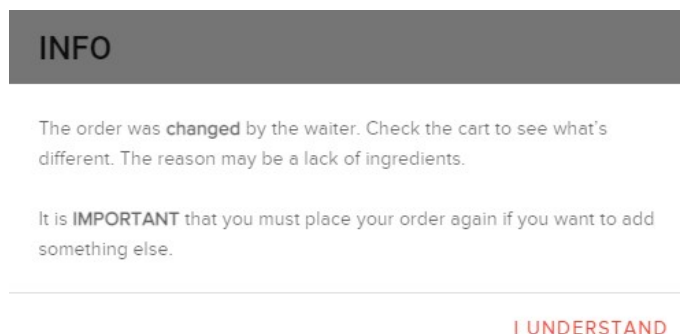
Slika 3.2: Primer nakupovalne košarice z dvema artikloma



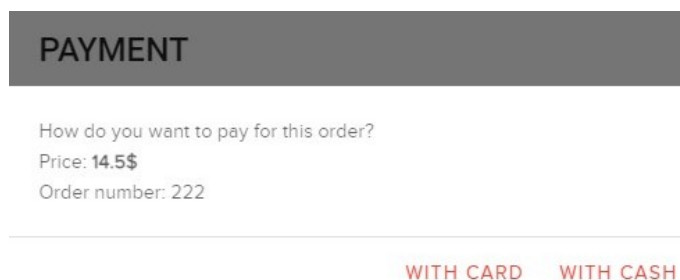
Slika 3.3: Pojavno sporočilo ob uspešni oddaji naročila



Slika 3.4: Pojavno sporočilo ob potrditvi naročila s strani natakarja



Slika 3.5: Pojavno sporočilo ob zavrnitvi naročila s strani natakarja

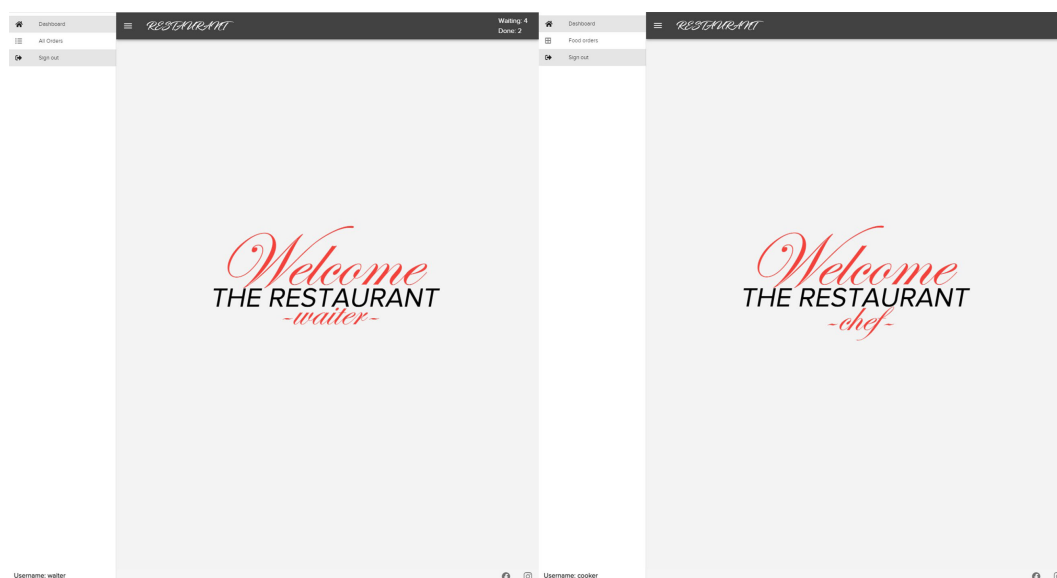


Slika 3.6: Pojavno okno z možnostjo izbire načina plačila

## 3.2 Vmesnik za natakarja in kuharja

Prvi pogled vmesnika je enak tako za natakarja kot kuharja, saj gre za skupno aplikacijo kjer se pogledi razlikuje glede vloge uporabnika, ki je določena v

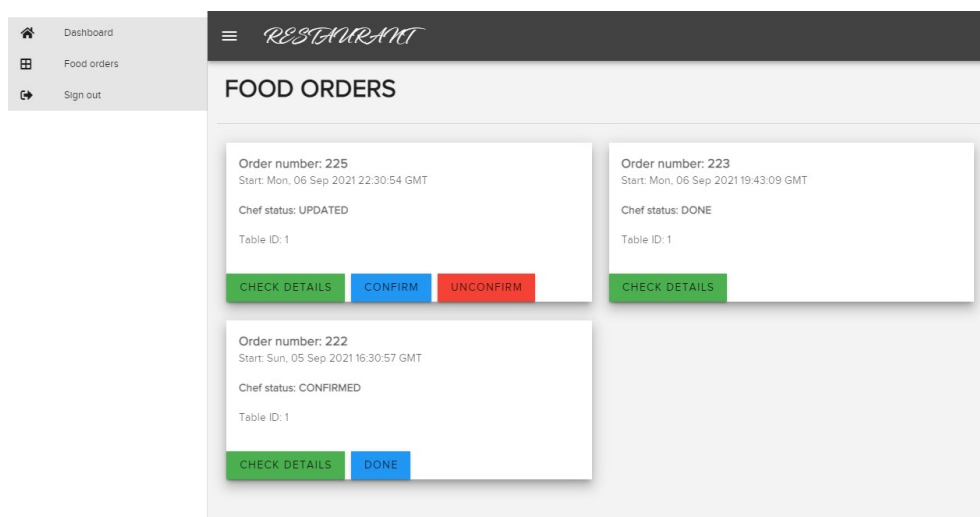
podatkovni bazi. Nismo naredili ločene aplikacije, saj ni bilo potrebe, namreč oba uporabnika imata zelo podobne funkcije. Slika 2.10 prikazuje prijavno okno. Po prijavi natakarja ali kuharja se uporabniško ime izpiše v levem spodnjem kotu. Na zgornji levi strani se prikaže zavihek z dvema podstranema in odjavnim gumbom sign out. Prva podstran imenovana dashboard ali prva strani ob uspešni prijavi, prikazuje napis za hitrejše razlikovanje med vlogami (slika 3.7). Natakar ima poleg tega v desnem zgornjem kotu še števec čakajočih in zaključenih naročil.



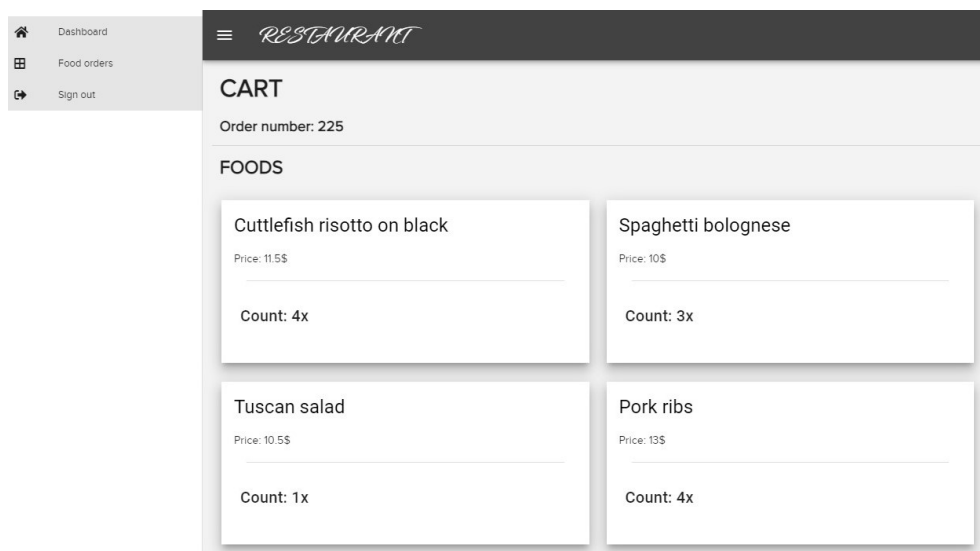
Slika 3.7: Pojavno okno z možnostjo izbire načina plačila

Kuharju se v zavihku food orders pojavijo vsa nezaključena naročila, ki vsebujejo hrano (slika 3.8). Vsako naročilo vsebuje naslednje podatke: identifikacijska številka naročila, čas oddaje naročila, status kuharja (Chef status) in številka mize. Kuhar more naročilo najprej pregledati z gumbom check details (slika 3.9) ter ga sprejeti ali zavrniti z gumbom confirm in unconfirm. Naročilo bi lahko zavrnil v primeru pomankanja sestavin. Ko kuhar zaključi s pripravo hrane o tem obvesti natakarja s klikom na gumb done, ki izbriše naročilo iz seznama. Ob vsaki izvedeni akciji se pri vsakem naročilu spremi

status kuharja, ki je lahko: updated, confirmed, unconfirmed in done. Status updated je v primeru dodajanja artiklov k naročilu s strani gosta.



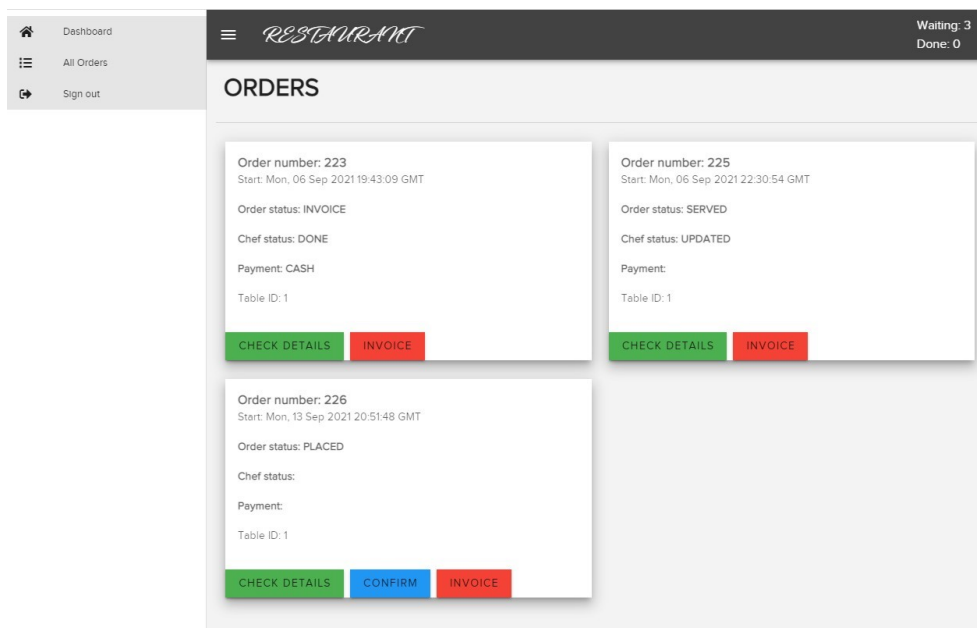
Slika 3.8: Zavihek food orders



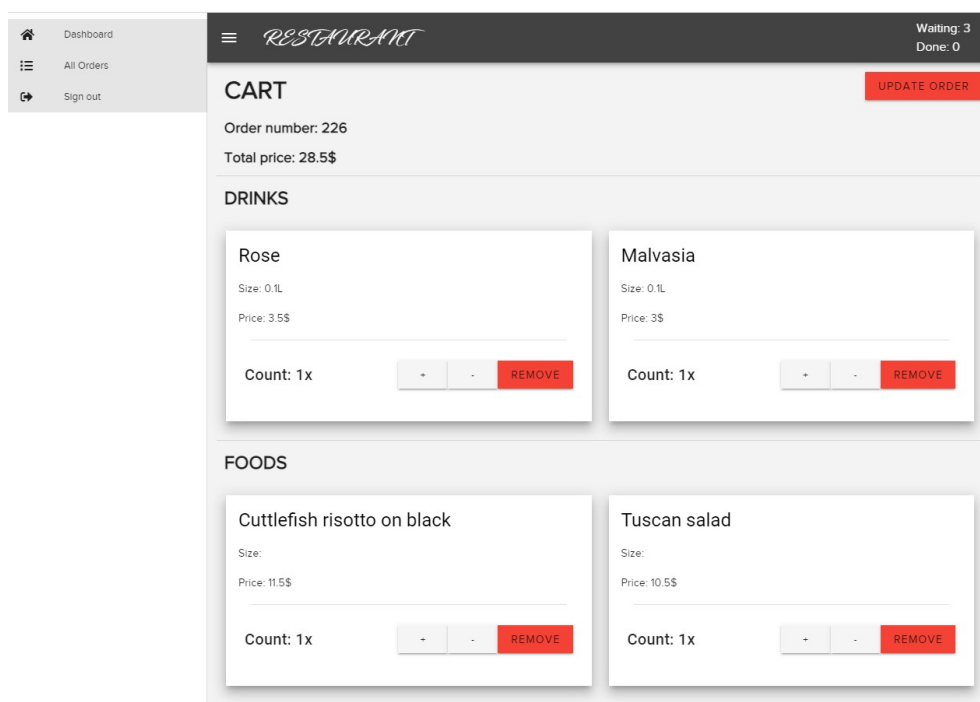
Slika 3.9: Zavihek, ki se odpre kuharju ob kliku na gubm check details

Natakarju se v zavihku all orders pojavijo vsa nezaključena naročila (slika 3.10).

Vsako naročilo vsebuje naslednje podatke: identifikacijska številka naročila (Order number), čas oddaje naročila, status naročila (Order status), status kuharja (Chef status), način plačila (Payment) in številka mize (Table ID). Izvajajo lahko popoln nadzor nad naročili, vendar ob vsaki storjeni akciji obvesti gosta (slike pojavnih sporočil iz prejšnjega poglavja). Novo naročilo mora natakar najprej potrditi ali zavrniti z gumbom confirm ali unconfirm oziroma urediti v z gumbom check details (slika 3.11). Ureja lahko celotno naročilo, vendar ne more dodajati novih artiklov. Ko zaključi urjanje more klikniti na gumb update order. Celotno naročilo lahko sprejme šele ko dobi kuharjevo potrditev/zavrnitev naročila o hrani (confirmed/unconfirmed). Ko je naročilo potrjeno, natakar čaka na kuharjevo potrditev o pripravi hrane, da jo lahko postreže. Natakar to v naročilu primerno označi z klikom na gumb served. V primeru, da gost zahteva račun, se natakru izpiše način plačila v zadevi Payment. Natakar ob kliku na gumb invoice zaključi naročilo.



Slika 3.10: Zavihek all orders



Slika 3.11: Zavihek, ki se odpre natakarju ob kliku na gubm check details

## Poglavje 4

## Diskusija

### 4.1 Implementacija programa v realnosti

Našo aplikacijo bi potencialnim kupcem ponudili v dveh različicah.

Prva različica, ki bi bila cenejša, bi predstavljala aplikacijo v obliki QR kod na vsaki mizi v restavraciji. V kodi bi bilo zapisano ime restavracije in številka mize. Kodo bi gost skeniral z mobilnim telefon in bil preusmerjen v aplikacijo. Celoten sistem bi bil na javnem Internetu tako za goste kot tudi natakarje in kuharje. Strežnik bi bil postavljen za celotno Slovenijo, in bi omogočal storitev vsem restavracij po Sloveniji. Potrebno bi bilo zagotoviti, da ne bi prihajalo do »fantomskih« naročil, kjer bi nepridipravi oddaljeno oddajali neveljavna naročila. To bi lahko zagotovili na primer, da bi moral biti uporabnik prijavljen preko lokalnega omrežja s čimer bi overili njegovo dejansko prisotnost v restavraciji. Seveda bi pri tem morali zagotovi še vrsto drugih varnostnih mehanizmov. Ena od rešitev bi bilo geslo, ki se spreminja za vsako naročilo in ga lahko izda samo natakar.

Druga oziroma dražja različica bi bila uporaba tablic za vsako mizo. Sistem bi bil dostopen samo lokalno, znotraj restavracije, kar bi bilo iz vidika varnosti precej bolj varno. Restavracija bi potrebovala poleg tablic potrebuje še lokalni spletni strežnik.

## 4.2 Izboljšave

Izboljšav za to aplikacijo je veliko, ker obstaja veliko zadev, ki bi aplikacijo dvignilo na naslednji nivo, vendar v okviru naše implementacije bi lahko izboljšali naslednje stvari:

- 1.) Možnost hkratnega delovanja več kuharjev, da bi lahko vsak kuhar vedel kaj more delati. Tako kot smo implementirali za natakarje.
- 2.) Dodatno spremljanje hrane in pijače na strani kuharja/natakarja – katera pijača in hrana je že bila postrežena.
- 3.) Dodajanje novih artiklov s strani natakarja.
- 4.) Pisanje opom na strani natakarja in kuharja, saj bi s tem preprečili daljše klepete.

Še par izboljšav, ki bi obstoječo aplikacijo dvignilo na naslednji nivo:

- 1.) Statistika za lastnika restavracije, ki bi poleg vseh podatkov lahko računala oceno nabave za prihodnji mesec. Ta bi imel svoj ločen uporabniški račun, ki bi omogočal tudi nastavitve v pogledu za gosta –lahko bi spreminjal številko mize.
- 2.) Brezstično plačevanje s kartico direktno na strani gosta.
- 3.) Če bi aplikacija delovala na centralnem strežniku, bi lahko za vsako restavracijo omogočali tudi dostavo hrane z enakim pogledom, ki bi bil vedno dostopen na skupni spletni strani – npr. kot Glovo in Wolt. Vpeljava sistema v restavracije brez potreb po natakarju. Podobne želje so imeli v LarsSven.

## 4.3 Konkurenca

McDonalds, ki rešitev uporablja že nekaj let. Prednosti in slabosti – naročile se lahko izvede elektronsko samo na vhodu, ni mogoče ponovnega hitrega naročila,... Spletne aplikacije, ki ponujajo dostavo vseh restavracij po Sloveniji npr. ehrana in wolt. To bi bilo mogoče narediti tudi za mojo rešitev. Prednost bi bila, da bi mi lahko to restavracijam ponujal kot paket.



## Poglavje 5

### Sklepne ugotovitve



# Literatura

- [1] About eslint. Dosegljivo: <https://eslint.org/docs/about/>. [Dostopano: 01. 11. 2020].
- [2] Reactivity in depth. Dosegljivo: <https://v3.vuejs.org/guide/reactivity.html#what-is-reactivity>. [Dostopano: 31. 10. 2020].
- [3] Toad data modeler. Dosegljivo: <https://www.quest.com/products/toad-data-modeler/>. [Dostopano: 23. 02. 2021].
- [4] Vue cli overview. Dosegljivo: <https://cli.vuejs.org/guide/>. [Dostopano: 01. 11. 2020].
- [5] Vue router. Dosegljivo: <https://router.vuejs.org/>. [Dostopano: 01. 11. 2020].
- [6] What is mysql? Dosegljivo: <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>. [Dostopano: 23. 02. 2021].
- [7] What is rest. Dosegljivo: <https://restfulapi.net/>. [Dostopano: 01. 05. 2021].
- [8] What is vuex? Dosegljivo: <https://vuex.vuejs.org/>. [Dostopano: 01. 11. 2020].
- [9] Why is flask good? Dosegljivo: <https://www.fullstackpython.com/flask.html>. [Dostopano: 01. 11. 2020].

- [10] Why vuetify? Dosegljivo: <https://vuetifyjs.com/en/introduction/why-vuetify/>. [Dostopano: 01. 11. 2020].