

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 \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

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

Seznam uporabljenih kratic

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

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^AT_EX. 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 restavracij, vendar le malo iz med njih uporablja napredne sisteme naročanja kot npr. ena izmed večjih verig s hitro prehrano, McDonalds. V Sloveniji so obstajali projekti s podobnimi idejami, vendar z napačnimi cilji zaradi katerih so bili neuspešni. Eden izmed glavnih razlogov, da jim ni uspelo, je bilo sabotiranje sistemov s strani natakarjev, saj so misli da bo tehnologija zamenjala njegove službe. Z zavedanjem teh problematik smo se odločili narediti diplomsko 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 oziroma jedilni list (ang. menu) z možnostjo naročanja 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 neučakane na vseh področjih, bi bile hitreje in bolj kvalitetno postrežene. Tako bi imeli poleg restavracij s hitro prehrano tudi restavracije s hitro postrežbo.

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

Poglavje 2

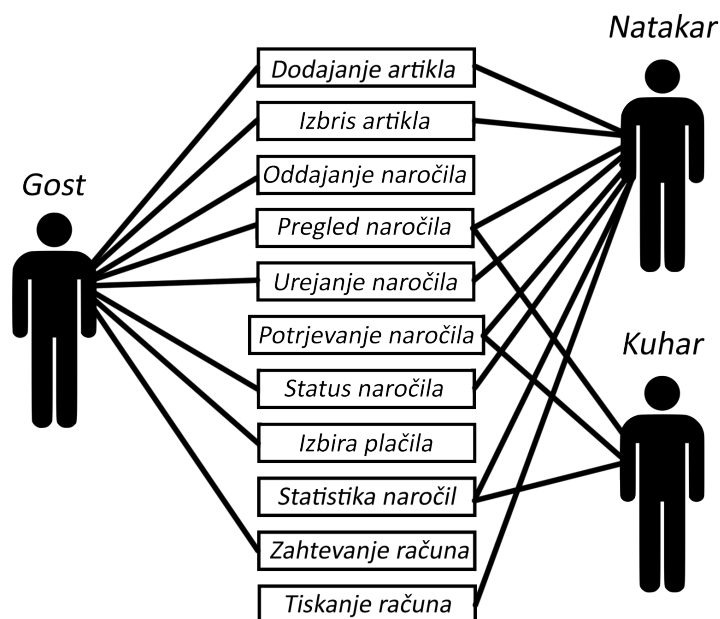
Načrtovanje in razvoj aplikacije

Razvoj aplikacije je potekal v treh delih in sicer analiza zahtev oziroma izdelava diagrama primerov uporabe, načrtovanje arhitekture in izbira tehnologij ter implementacija. Rešitev sestavljajo dve aplikaciji. 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 vsi gosti 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 zahtevanje računa. V restavraciji je lahko več natakarjev, ki svoje delo opravljajo 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 sprejeme, zavrne naročilo ali pa potrdi zaključek priprave naročila. Restavracija ima lahko več kuharjev, vendar vsi uporabljajo aplikacijo preko iste prijave. Omejitve je v podatkovnem modelu. Funkcionalnosti kuharja so: pregledovanje in potrjevanje naročila ter spremljanje statistike naročil.

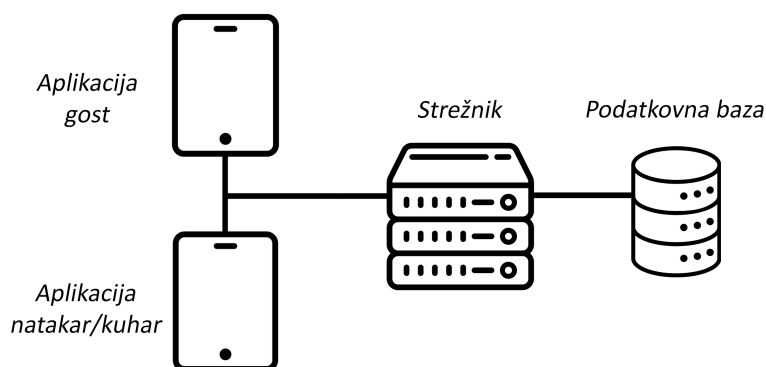
Iz analize zahtev smo za lažji pregled nad vsemi funkcionalnostmi izdelali diagram primerov uporabe, ki ga prikazuje slika 2.1.

Uporabili smo strukturo novodobnih aplikacij katero sestavljajo podat-



Slika 2.1: Diagram primerov uporabe spretnega naročanja

kovna baza, strežnik in aplikacija oziroma odjemalec. Gre za koncept, ki ga je moč prilagajati predvsem z uporabniškega vidika. Slika 2.2 prikazuje strukturo aplikacije [6]. Podatkovna baza je namenjena shranjevanju vseh podatkov za posamezno restavracijo. Strežnik implementira vmesnik RESTful, ki odjemalcu oziroma aplikaciji posreduje podatke iz podatkovne baze.

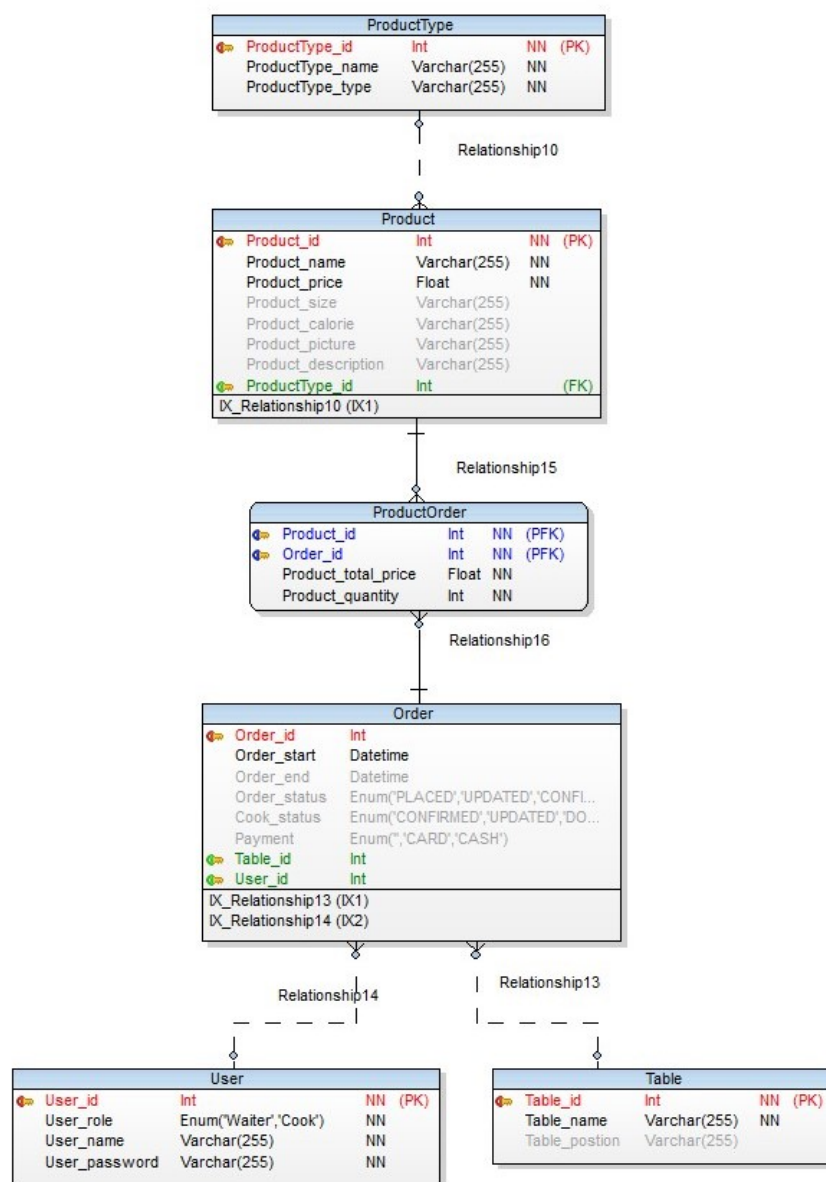


Slika 2.2: Visokonivojska arhitektura

2.1 Podatkovna baza

Na podlagi izdelanega diagrama zahtev oziroma funkcij smo najprej izdelali logičen podatkovni model (slika 2.3). Podatkovna baza je sestavljena 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) in 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 so shranjene v datotečnem sistemu spletnega strežnika.

ProductOrder tabela je namenjena količini in končni ceni 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 sporočata statusa naročila med vlogami. Uporabili smo 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 definirali 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 (ang. 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 [7]. 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 je bilo, da je sistem zanesljiv, saj brez njega odjemalec ne more delovati. Izbrali smo arhitekturo REST zaradi načel, ki so opisana spodaj [8]. 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.

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 mora vsako zahtevo odjemalca obravnavati kot popolnoma novo. V programski kodi strežnika je dobro razvidno, da ne uporabljamo nobenih globalnih spremenljivk. Vsi podatki, ki so potrebni, da strežnik odgovori na zahtevo HTTP, se nahajajo bodisi v podatkovni bazi bodisi jih odjemalec na strežnik posre-

duje z zahtevkom.

3.) *Predpomnjenje* (ang. *Cachable*) prinaša izboljšanje zmogljivosti na strani odjemalca in omogoča razširljivost 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 avtenticiranje 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 opcij-sko načelo. Strežnik na zahtevo odjemalca pošlje oziroma izvede programsko kodo na strani odjemalca. To smo vpeljali s pomočjo spletnih vtičnikov (ang. *websocket*), kjer so bolj podrobno opisani spodaj.

6.) *Enotni vmesnik* (ang. *Uniform interface*) (API) med strežnikom in odjemalcem. Vsak vir mora vsebovati povezavo, 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.

Knjižnica Flask nam je zelo poenostavila izdelavo strežnika. Flask je eno izmed najbolj popularnih spletno aplikacijskih vmesnikov (ang. *Freamwork*) [10]. 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 spletnem ogroddju Django.

Vsaka relativna povezava URI na strežniku predstavlja svoj vir podatkov iz podatkovne baze. 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 podat-

kovni bazi. Slika 2.5 prikazuje uporabo te funkcije in relativne poti drinks. 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
```

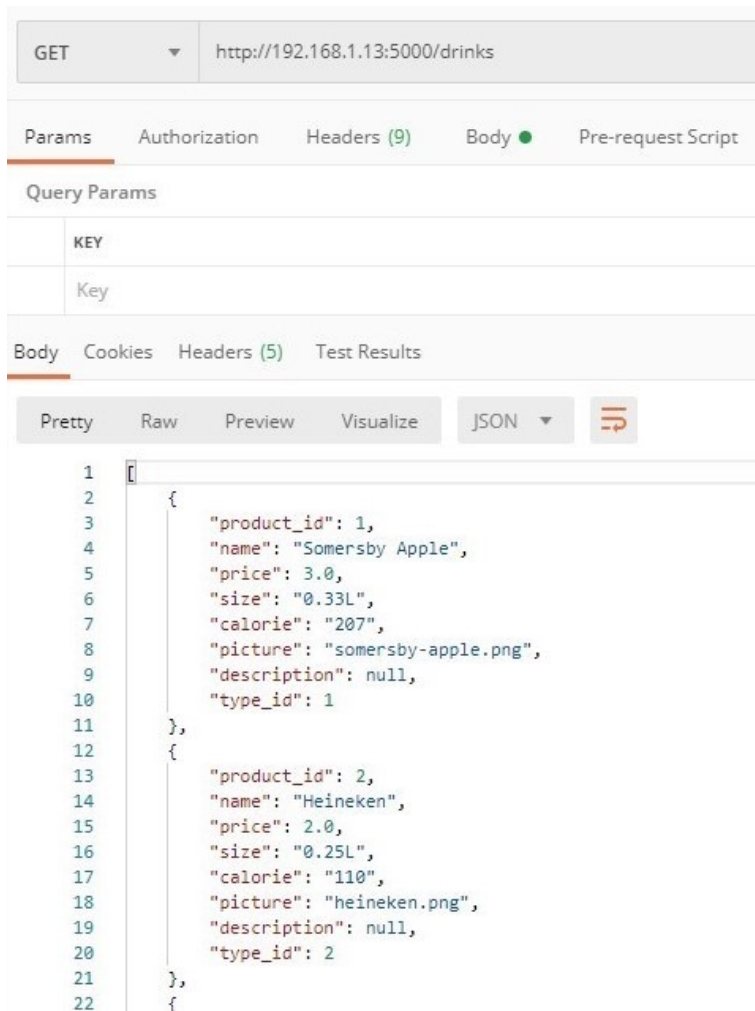
Slika 2.4: Funkcija, ki prebere podatke iz podatkovne baze

```
@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

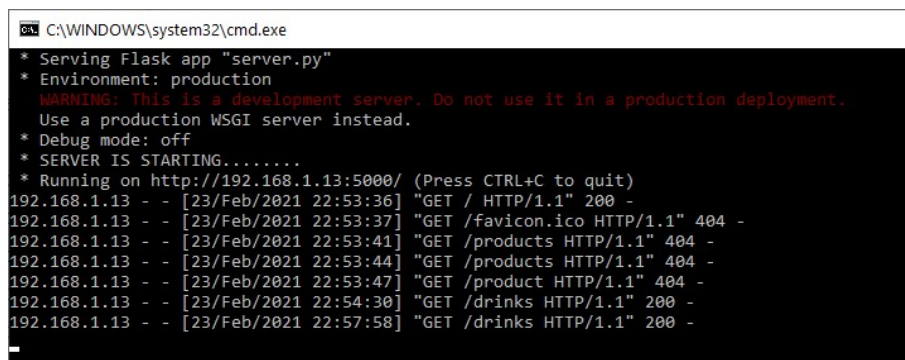
Tako smo dobili vmesnik, ki na zahtevo odjemalca odgovori s podatki v JSON formatu. Na sliki 2.6 je primer testiranja zahtevka v programu Postman, ko odjemalec zahteva podatke vseh pijač iz podatkovne baze (HTTP metoda GET). Strežnik omogoča tudi sprejemanje podatkov z metodami PUT in POST. Mi smo uporabljali metodo POST za posredovanje 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 ali napravah. Uporabili smo ga za obveščanje odjemalcov o spremembah v podatkovni bazi. Za to smo uporabljali funkcijo „emit“, ki pomeni oddaja-



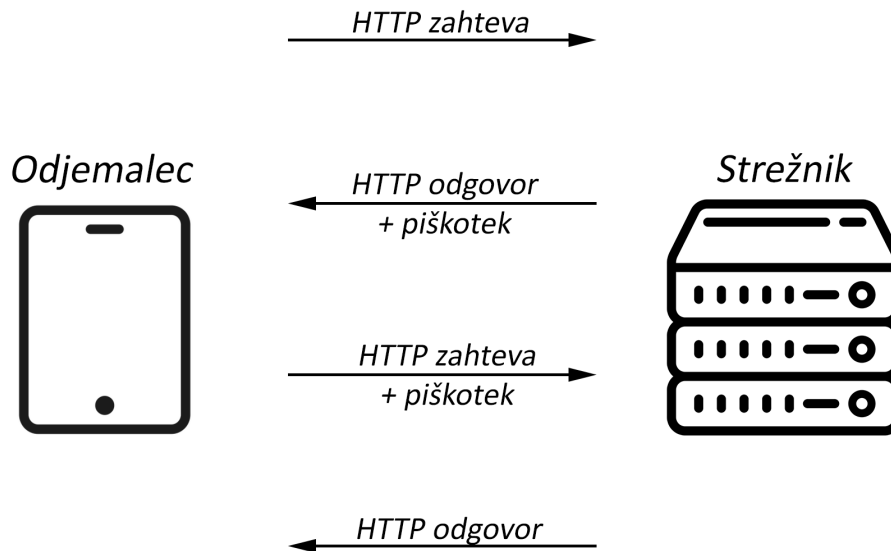
```
C:\WINDOWS\system32\cmd.exe
* Serving Flask app "server.py"
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* SERVER IS STARTING.....
* Running on http://192.168.1.13:5000/ (Press 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

nje. 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 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. 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 je 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 spremembi 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. Z uporabo vmesnika CLI se lahko izbere projekt, kjer so na voljo že privzete nastavitve, lahko pa se jih tudi nastavi po meri. Mi smo uporabili Vuex, Vue-Router, ESLint in Vuetify.

Vuex je knjižnica za shranjevanje vrednosti v aplikacijah Vue.js [9]. Služi kot centralizirana baza podatkov za vse komponente v aplikaciji.

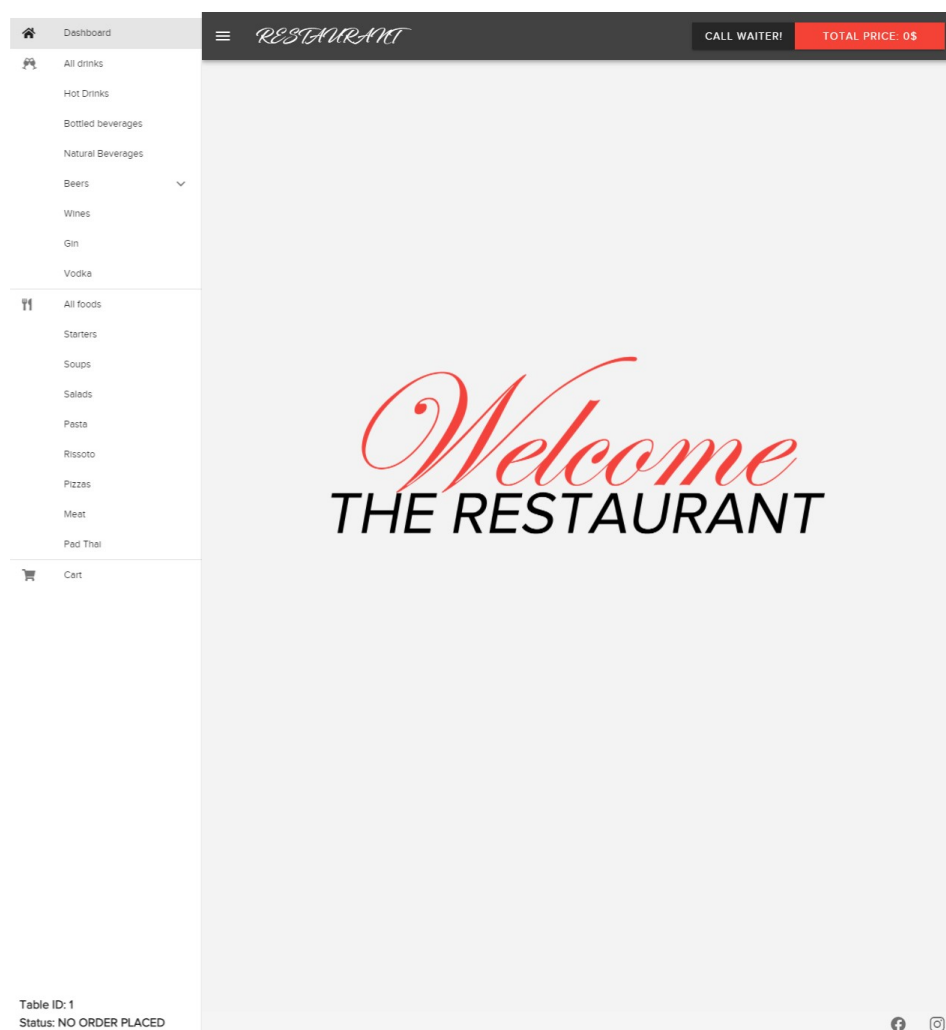
Vue-Router je uradni usmerjevalnik za Vue.js [5]. Integrira se globoko z jedrom Vue.js, tako da poenostavi izdelavo SPA aplikacij. Usmerjevalnik je mišljen v smislu usmerjanja na druge komponente (angl. Component), ki v Vue.js predstavljajo druge poglede, lahko bi rekli podobno kot podstrani.

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 [11]. Za razliko od drugih vmesnikov je Vuetify enostaven za učenje z več stotimi komponentami izdelanih po specifikacijah Material Design.

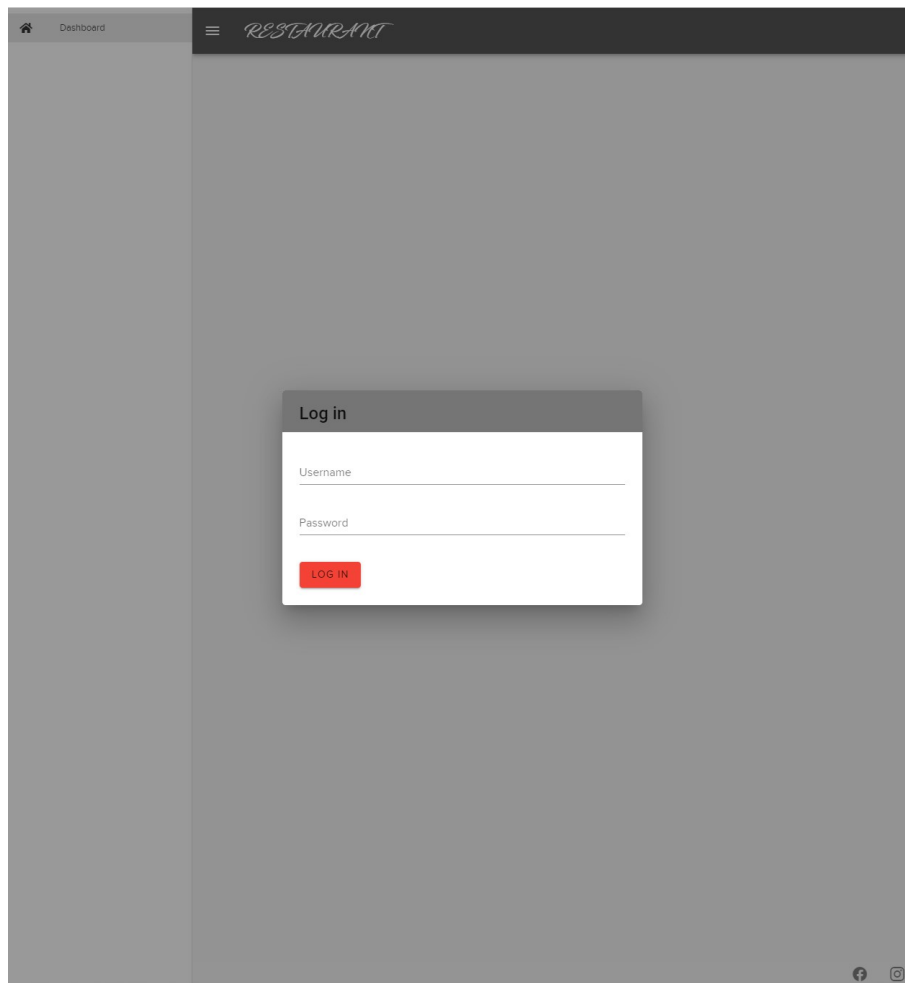
Vue-devtools je zgolj dodatek v brskalniku, ki omogoča lažje sledenje delovanja aplikacije in detektiranju napak.

Programsko izvedbo na strani 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 več 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

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, obveščanju gosta o stanju naročila, itd. Najprej smo hoteli uporabiti samodejno osveževanje na določen časovni interval, vednar je uporaba spletnih vtičnikov hitrejša in učinkovitejša. Slika 2.3 prikazuje seznam vseh



Slika 2.10: Spetni vmesnik za natakarja in kuharja

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.

```

},
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("orderENDD!!!");
  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

```

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

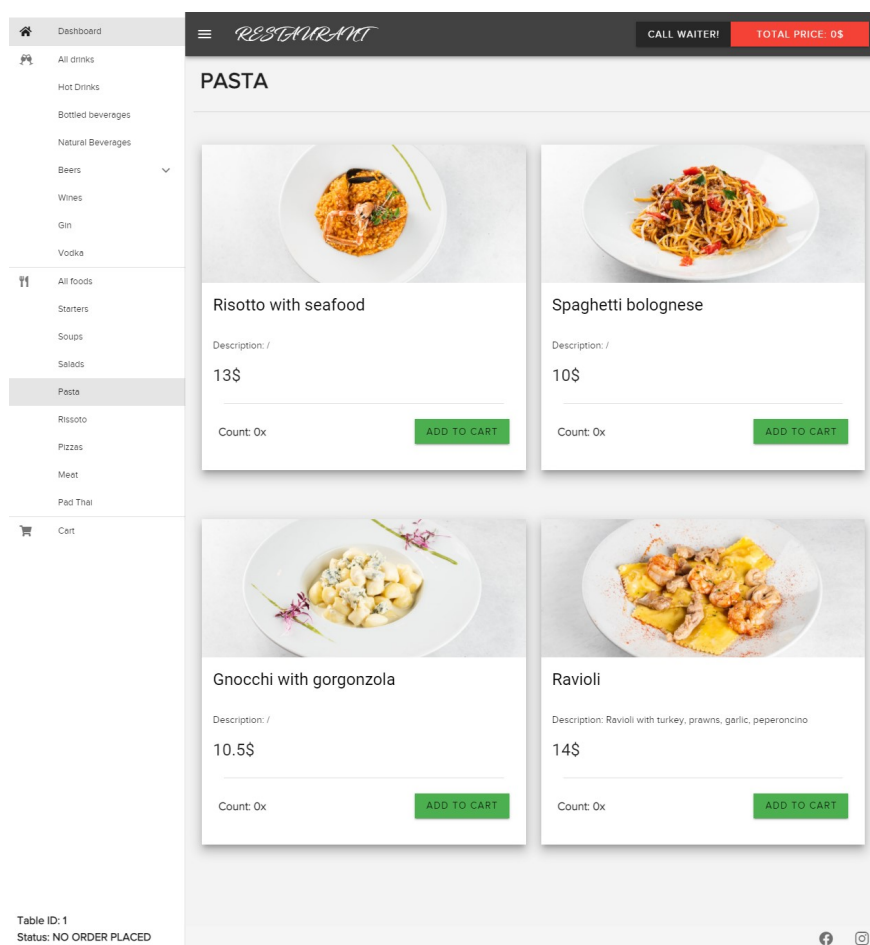
Poglavje 3

Delovanje aplikacije

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

3.1 Vmesnik za gosta

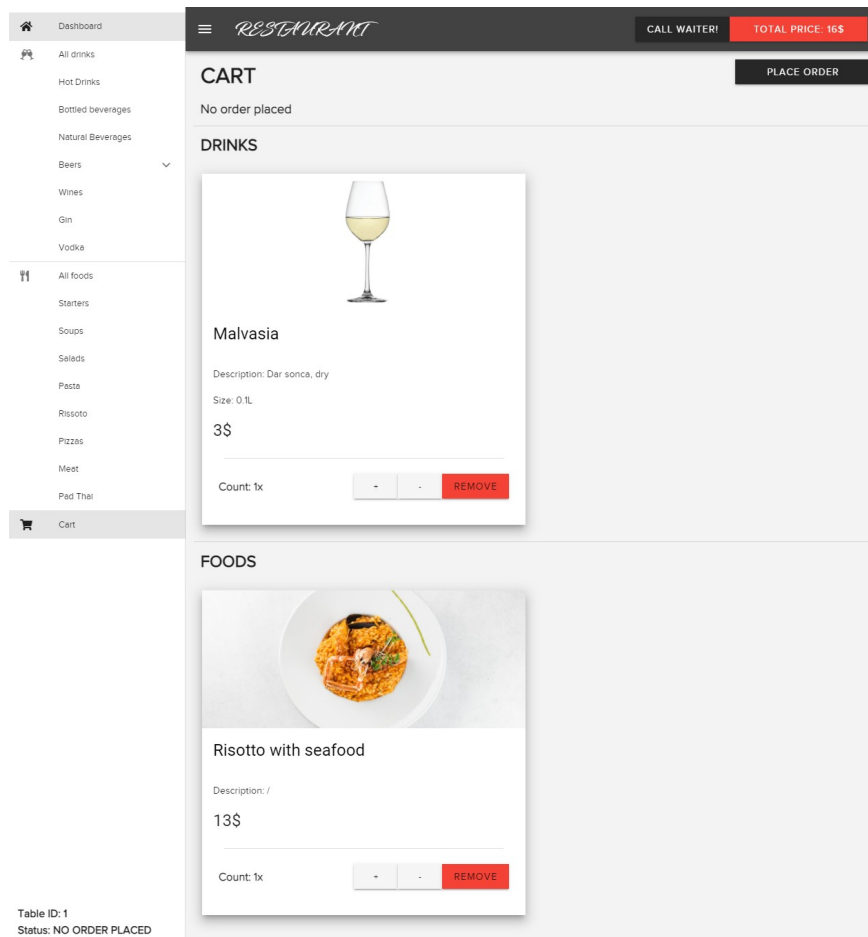
Prvi pogled vmesnika za gosta vsebuje napis za dobrodoščilo (slika 2.9) katerega bi lahko zamenjalo oglaševanja, predstavitev restavracije ali karkoli bi si potencialni kupec zaželel imeti. V zgornjem desnem kotu se nahaja gumb *call waiter* za klic natakarja in števec skupne cene artiklov v nakupovalni košarici. Tipka *call waiter* namenjena je gostom, ki aplikacije ne želijo uporabljati ali v primeru kakršnihkoli težav. V spodnjem levem kotu se nahaja status in identifikacijska številka naročila. Zavihki na levi strani predstavljajo seznam vseh vrst hrane in pijače, katere kažejo na podstrani le-teh (slika 3.1). Gostu to predstavlja jedilnik s katerim v nakupovalno košarico dodaja, briše hrano in pijačo ali spreminja njihovo količino. Vsaka hrana ali pijača vsebuje sliko in naslednje podatke: ime, opis in ceno. Poleg tega vsebuje še gumb *add to cart*, ki ob kliku izgine in prikaže tri nove gumbe *+* (povečaj količino), *-* (zmanjšaj količino) in *remove* (odstrani). Nakupovalna košarica (ang. *cart*) je skupno mesto vse hrane in pijače potencialne za naročilo (slika 3.2).



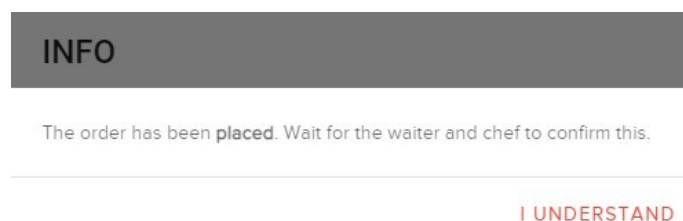
Slika 3.1: Seznam artiklov znotraj vrste špagetov

Naročilo se odda s klikom na gumb *place order*, ki gosta preusmeri na prvo stran in obvesti s pojavnim sporočilom prikazanim na sliki 3.3. Ko je naročilo oddano lahko gost ponovno dodaja hrano in pijačo v košarico, vendar je že oddani hrani in pijači onemogočeno zmanjšati količino ali jih izbrisati. Naročilo je sprejeto ko ga natakar potrdi, kar spremeni status naročila in prikaže 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 s klikom na gumb *request receipt*, ki odpre pojavno okno

(slika 3.6) na katerem je potrebno izbrati način plačila.



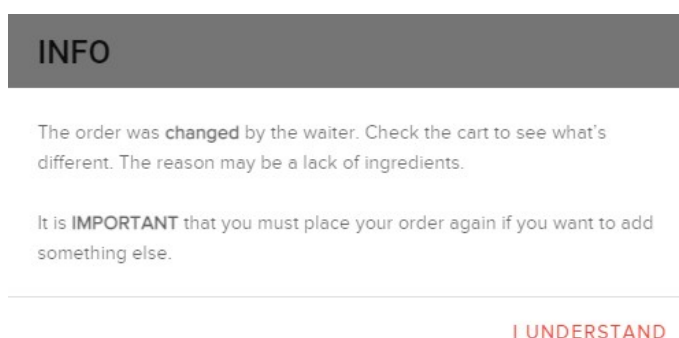
Slika 3.2: Primer naročila



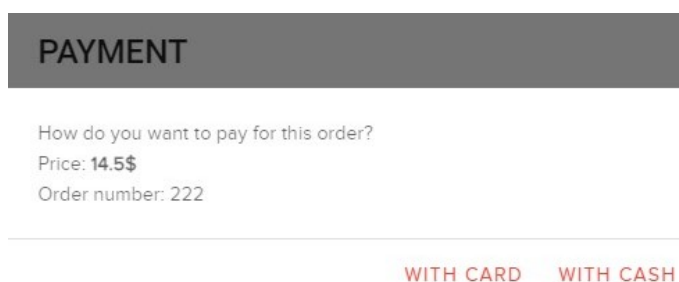
Slika 3.3: Pojavno sporočilo ob uspešni oddaji naročila



Slika 3.4: Pojavno sporočilo ob potrditvi naročila s strani natakarja



Slika 3.5: Pojavno sporočilo ob zavrnitvi naročila s strani natakarja

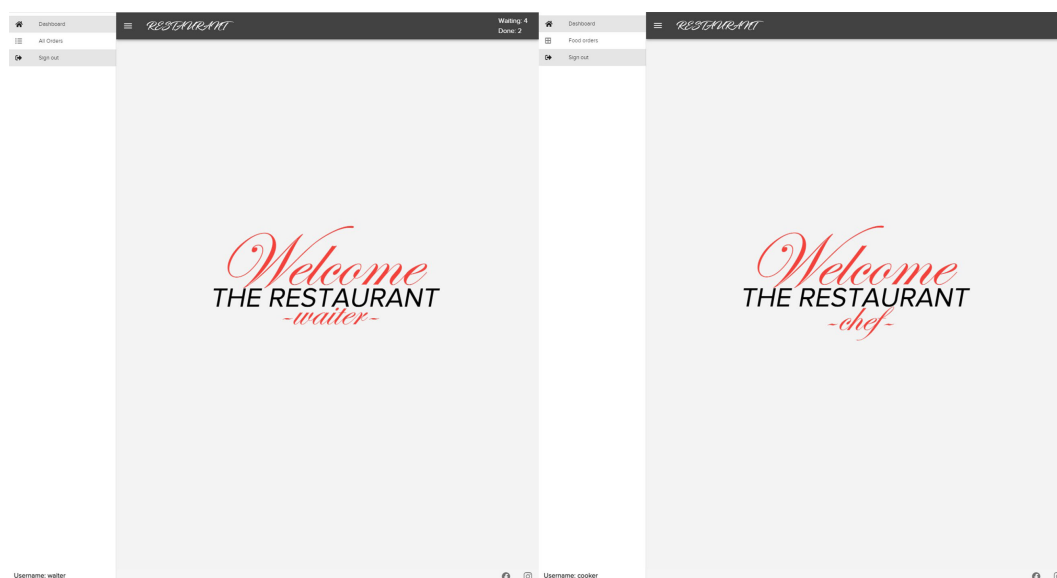


Slika 3.6: Pojavno okno z možnostjo izbire načina plačila

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 razlikujejo glede vloge uporabnika, ki je določena v

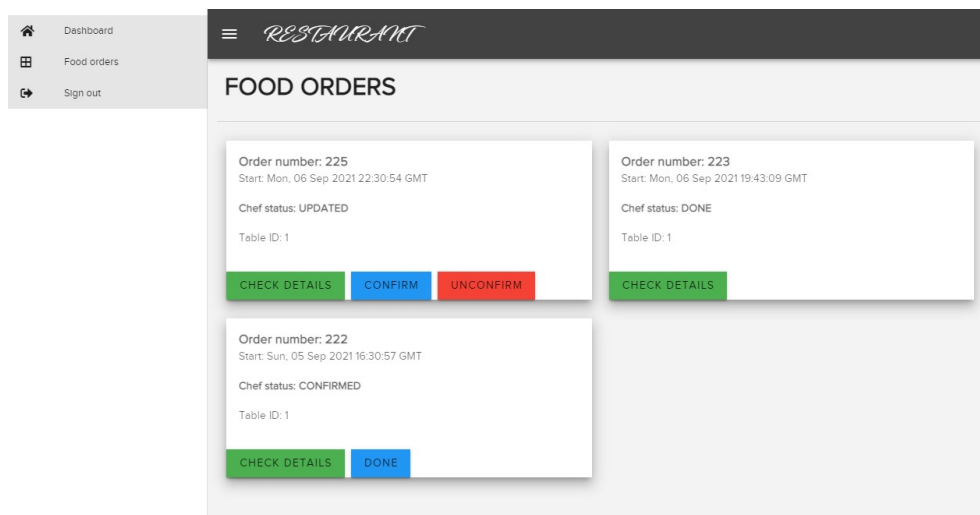
podatkovni bazi. Nismo naredili ločene aplikacije, saj ni bilo potrebe, namreč oba uporabnika imata zelo podobne funkcije. Slika 2.10 prikazuje prijavno okno. Po prijavi natarja ali kuharja se uporabniško ime izpiše v levem spodnjem kotu. Na zgornji levi strani se prikaže zavihek z dvema podstranema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



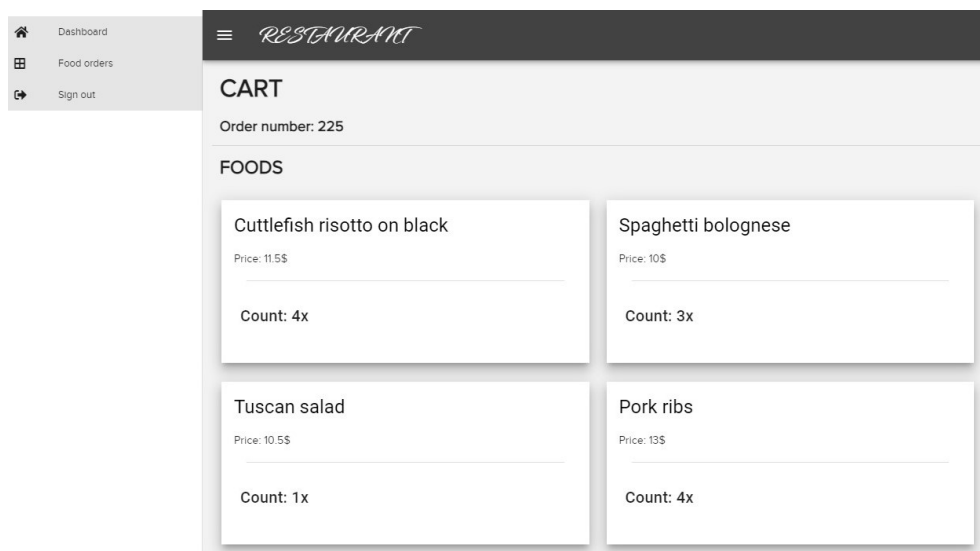
Slika 3.7: Začetni pogled ob prijavi natarja in kuharja

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 in številka mize. Kuhar mora naročilo najprej pregledati z gumbom *check details* (slika 3.9) ter ga sprejeti ali zavrniti z gumbom *confirm* in *unconfirm*. Naročilo lahko kuhar zavrne v primeru pomanjkanja sestavin. Ko kuhar zaključi s pripravo hrane o tem obvesti natarja s klikom na gumb *done*, ki izbriše naročilo iz seznama. Ob vsaki izvedeni akciji se pri vsakem naročilu spremi status ku-

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



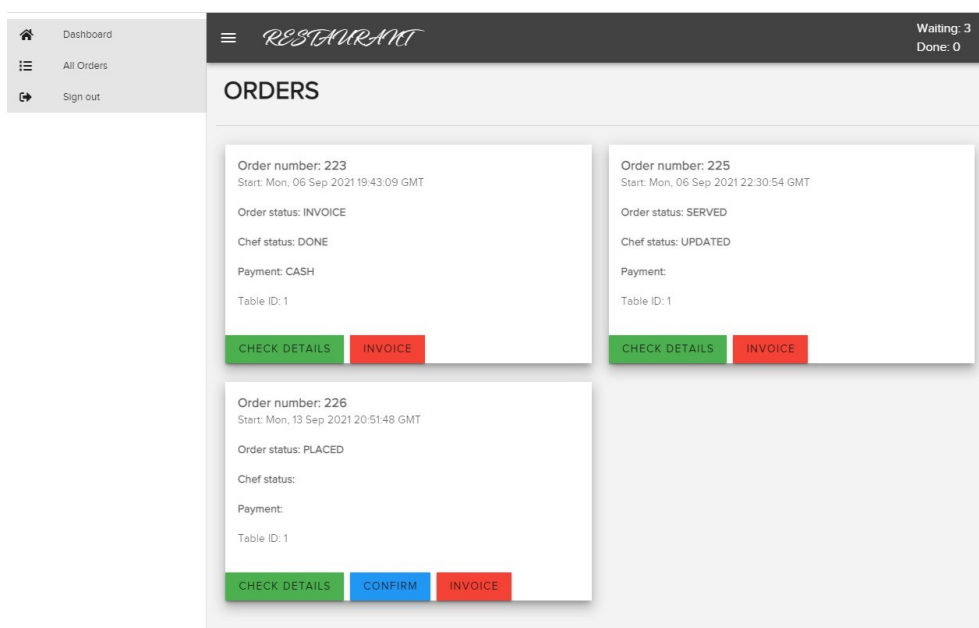
Slika 3.8: Zavihek food orders



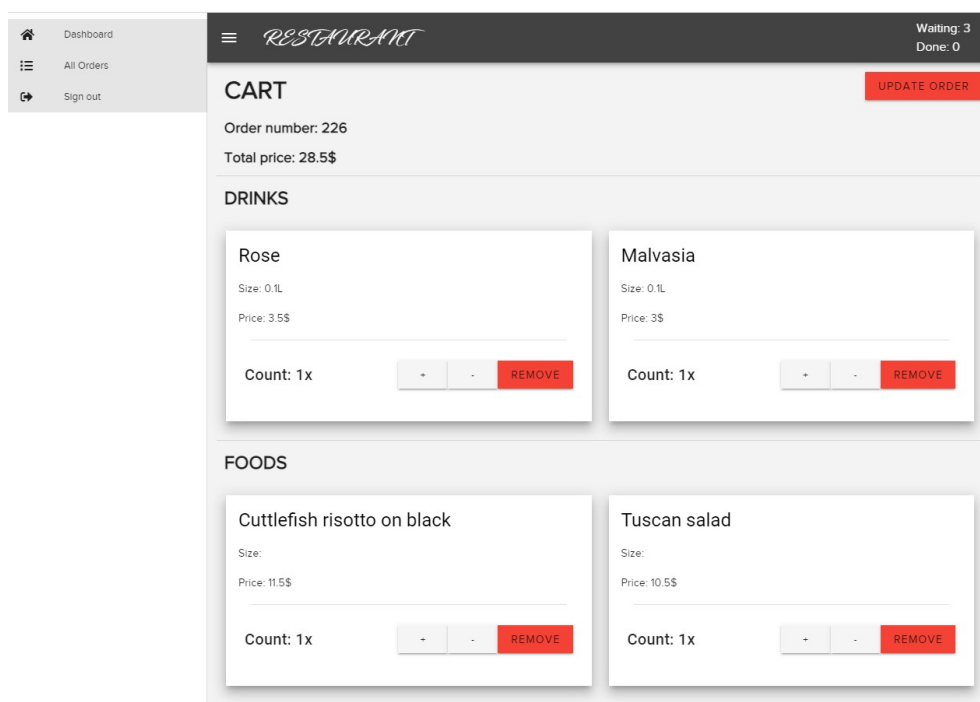
Slika 3.9: Zavihek, ki se odpre kuharju ob kliku na gumb *check details*

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

Vsako naročilo vsebuje naslednje podatke: identifikacijska številka naročila, čas oddaje naročila, status naročila, status kuharja, način plačila in številka mize. Izvaja lahko popoln nadzor nad naročili, vendar ob vsaki izvedeni 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, to pomeni da lahko spreminja količino hrane in pijačo ali jo izbriše iz naročila, vendar ne more pa dodajati hrane in pijače, ki ni na naročilu. Ko zaključi urjanje mora klikniti na gumb *update order*. Celotno naročilo lahko sprejme šele ko dobi kuharjevo potrditev naročila o hrani (Chef status označen s confirmed). Ko je naročilo potrjeno, natakar čaka na kuharjevo potrditev o pripravi hrane, da jo lahko postreže (Chef status označen z done). Natakar postreženo naročilo označi s klikom na gumb *served*. V primeru, da gost zahteva račun, se natakarju izpiše način plačila v zadevi payment. Naročilo se zaključi, ko natakar natisne račun s klikom na gumb *invoice*.



Slika 3.10: Zavihek all orders



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

Poglavje 4

Diskusija

4.1 Implementacija programa v realnosti

Naša aplikacijo predstavlja osnovno rešitev, vendar bi jo lahko z ustrezno nadgradnjo ponudili potencialnim kupcem.

Prva različica, ki bi bila cenejša, bi predstavljala aplikacijo v obliki QR kode, ki bi bila locirana na vsaki mizi v restavraciji. V kodi bi bilo zapisano ime restavracije in številka mize. Kodo bi gost skeniral z mobitelom in bil preusmerjen v aplikacijo. Celoten sistem bi bil na Internetu, tako za goste kot tudi natakarje in kuharje. Strežnik bi bil postavljen za celotno Slovenijo in bi omogočal storitev vsem restavracijam po Sloveniji. Potrebno bi bilo zagotoviti, da ne bi prihajalo do fantomskih naročil, kjer bi nepridipravi oddaljeno oddajali neveljavna naročila. Ena rešitev bi bila, da bi moral biti uporabnik prijavljen preko lokalnega omrežja s čimer bi overili svojo dejansko prisotnost v restavraciji. Druga rešitev bi bilo geslo, ki bi ga uporabnik prejel od natakara.

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 boljše. Restavracija bi potrebovala poleg tablic še lokalni spletni strežnik. Prav tako bi lahko uproabili QR kode, vendar bi mora.

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 hrkatnega delovanja več kuharjev, da bi lahko vsak vedel kaj more delati. Tako kot smo implementirli za natakarje.
- 2.) Dodatno spremljanje hrane in pijače na strani kuharja/natakarja – katera pijača in hrana je že bila postrežena.
- 3.) Dodajanje novih artiklov v naročilo na strani natakarja.
- 4.) Pisanje opomb pri vsakem naročilo, kar bi natakarju in kuharju olajšalo delo ob povečanemu številu gostov.

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. Aplikacija bi imela svoj ločen uporabniški račun, ki bi omogočal tudi nastavitve v pogledu za gosta (npr. lahko bi spreminjal številko mize).
- 2.) Brezstično plačevanje s kartico neposredno na strani gosta.
- 3.) Če bi aplikacija delovala na centralnem strežniku, bi lahko za vsako restavracijo omogočali dostavo hrane z enakim pogledom, ki bi bil dostopen na skupni spletni strani npr. kot Glovo in Wolt. Tak sistem v restavracijah zmanjša iskanost natakarjem. Podobne želje so imeli v LarsSven.

4.3 Konkurenca

Največja trenutna konkurenca je McDonalds, ki rešitev uporablja že nekaj let. Gre za rešitv pri kateri se naročilo izvede elektronsko pri vhodu, ni mogoče ponovnega hitrega naročila,...

Velika konkurenca so tudi spletne aplikacije, ki ponujajo dostavo vseh restavracij po Sloveniji npr. Glovo 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] Web application architecture: Best practices and guides. Dosegljivo: <https://lanars.com/blog/web-application-architecture-101>. [Dostopano: 14. 06. 2021].
- [7] What is mysql? Dosegljivo: <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>. [Dostopano: 23. 02. 2021].
- [8] What is rest. Dosegljivo: <https://restfulapi.net/>. [Dostopano: 01. 05. 2021].
- [9] What is vuex? Dosegljivo: <https://vuex.vuejs.org/>. [Dostopano: 01. 11. 2020].

- [10] Why is flask good? Dosegljivo: <https://www.fullstackpython.com/flask.html>. [Dostopano: 01. 11. 2020].
- [11] Why vuetify? Dosegljivo: <https://vuetifyjs.com/en/introduction/why-vuetify/>. [Dostopano: 01. 11. 2020].