

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, 2022

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 restavraciji

Kandidat naj v diplomskem delu analizira postopek obdelave naročil v restavraciji. Zasnuje naj spletno aplikacijo, ki omogoča gostom pregled ponudbe in izvedbo. Osebjem naj ima možnost urejanja naročil, njihovo spremljanje in izvedbo ter zaključitev postopka z izdajo računa, pri čemer naj bo omogočena interakcija v stvarnem času. Za implementiran model strežnik-odjemalec naj predstavi enostaven primer uporabe vmesnikov za gosta, natakarja in kuharja.

Zahvaljujem se mentorici doc. dr. Miri Trebar in somentorju as. dr. Davidu Jelencu za vso pomoč in nasvete. Zahvaljujem se tudi ožji družini in puncu za podporo.

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	15
3.1	Vmesnik za gosta	16
3.2	Vmesnik za natakarja in kuharja	20
3.3	Primer uporabe	24
3.4	Implementirana rešitev	31
3.5	Izboljšave	31
4	Sklepne ugotovitve	33
	Literatura	35

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	application programming interface	programski vmesnik
CLI	command-line interface	ukazno-vrstični vmesnik
DBMS	database management system	sistem za upravljanje podatkovne baze
ENUM	enumerated type	naštevni tip
HTTP	hypertext transfer protocol	protokol svetovnega spleta
JSON	JavaScript object notation	objektna notacija za JavaScript
QR	quick response	hiter odziv
REST	representational state transfer	reprezentativni prenos stanja
SPA	single page application	enostranska aplikacija
SQL	structured query language	poizvedovalni jezik SQL
URI	uniform resource identifier	enotni identifikator vira
XML	extensible markup language	razširljiv označevalni jezik

Povzetek

Naslov: Elektronsko naročanje v restavraciji

Avtor: Luka Horvat

Diplomsko delo zajema analizo, načrtovanje in razvoj spletne aplikacije, ki omogoča elektronsko naročanje hrane in pijače v restavracijah. Namen aplikacije je olajšati delo natakarjem in nuditi kakovostnejšo postrežbo gostom. Ključne funkcionalnosti so oddajanje naročil za gosta ter nadzor in upravljanje naročil za natakarja in kuharja. Omogočeno je naročanje preko aplikacije ali z neposrednim stikom z natakarjem. Aplikacija je zasnovana kot visokonivojska arhitektura (model odjemalec-strežnik), ki jo sestavljajo aplikacija, strežnik in podatkovna baza. Za razvoj aplikacije so uporabljeni ogrodje Vue in programski jeziki JavaScript, Python ter MySQL za podatkovno bazo.

Ključne besede: spletna aplikacija, elektronsko naročanje, restavracija, Vue, JavaScript.

Abstract

Title: Electronic ordering in a restaurant

Author: Luka Horvat

The diploma thesis covers the analysis, design, and development of a web application for restaurant's electronic ordering of food and beverages. The purpose of the application is to facilitate the waiter's work and provide better quality service to guests. The key functionalities are placing orders by the guest and the control and management of orders by the waiter and the chef. It is possible to order via this application or directly contact the waiter. The application is designed as a high-level architecture (client-server model), consisting of the application, server, and database. The Vue framework and JavaScript, Python, and MySQL database languages developed the application.

Keywords: web application, electronic ordering, restaurant, Vue, JavaScript.

Poglavje 1

Uvod

Slovenija velja za državo z veliko restavracijami, vendar le redke med njimi uporabljajo napredne sisteme naročanja kot npr. ena izmed večjih verig s hitro prehrano, McDonald's. V Sloveniji so obstajali projekti s podobnimi idejami, vendar do sedaj nismo zasledili takšnih rešitev. Problematiko smo vzeli kot motivacijo ter si zadali cilj, da izdelamo aplikacijo kot pripomoček za optimizacijo dela natakarjev. Ker ustrezne rešitve za ta problem nismo zaznali, nam pa se je zdelo, da jo lahko realiziramo, smo se odločili v diplomski nalogi razviti izvedbo elektronskega naročanja hrane in pijače.

Zamislili smo si sistem za oddajanje naročil v restavracijah, ki ne bi nadomestil ljudi v strežbi, temveč bi služil kot pregledovalnik ponudbe z možnostjo naročanja hrane in pijače. Aplikacija bi bila na tablicah, ki bi bile locirane na vsaki mizi restavracije. Stranka bi bila tista, ki bi se odločala, ali želi pri naročanju uporabiti stik z osebo v strežbi ali bi naročila z uporabo aplikacije na tablici. Delovanje bi bilo sledeče: stranka oziroma gost odda naročilo, natakar in kuhar ga preverita, potrdira in pripravita. Posledično bi bile stranke, ki veljajo za zahtevnejše in bolj neučakane na vseh področjih, hitreje in kakovostneje postrežene.

Diplomsko delo je razdeljeno na dva dela. V prvem delu bomo opisali načrtovanja in razvoj aplikacije. To pomeni opis komponent, ki sestavljajo samo arhitekturo aplikacije, kako se med seboj povezujejo in sestavljajo de-

lujoč sistem. V drugem delu bomo opisali delovanje aplikacije, najprej podrobneje, nato na podlagi primera. Predstavili bomo tudi omejitve naše rešitve in mogoče izboljšave. Na koncu bomo podali še sklepne ugotovitve.

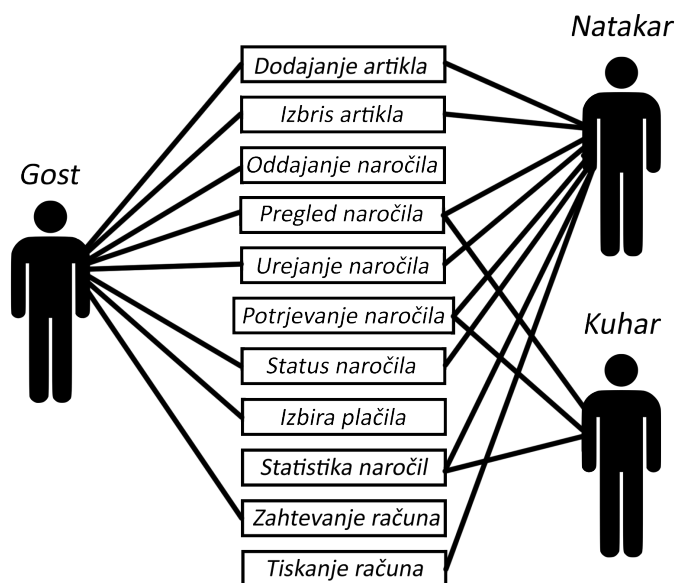
Poglavje 2

Načrtovanje in razvoj aplikacije

Načrtovanje aplikacije je potekalo v treh delih in je vključevalo opis zahtev oziroma izdelavo diagrama primerov uporabe, razvoj arhitekture in izbiro tehnologij. Predlagana rešitev je bila v nadaljevanju osnova za izdelavo dveh spletnih aplikacij. Ena aplikacija je namenjena gostom, druga pa natakarjem in kuharjem. Restavracija ima lahko več miz, vendar je za eno mizo lahko hkrati odprto eno naročilo. To pomeni, da morajo vsi gosti za mizo naročiti skupaj. Slika 2.1 prikazuje primere uporabe za izvedbo celotnega naročila.

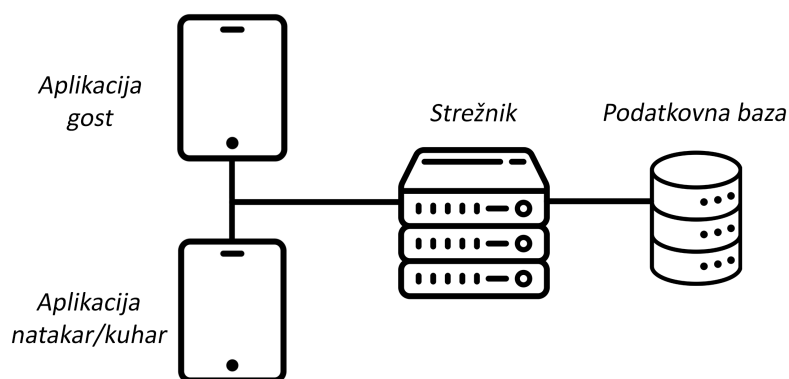
Gost lahko odda, spremeni ali zaključi naročilo. Funkcionalnost za gosta so: dodajanje artiklov v naročilo in brisanje teh iz njega, oddajanje, urejanje in pregledovanje naročila, spremljanje statusa naročila in zahtevanje računa.

Natakar lahko potrdi, zavrne, uredi ali zaključi naročilo. V restavraciji je lahko več natakarjev, ki svoja naročila vodijo preko ločene prijave. Funkcionalnosti za natakarja so: urejanje in pregledovanje naročila, spremljanje statusa in statistike naročil in možnost tiskanja računa za določeno naročilo. Kuhar lahko potrdi ali zavrne naročilo ter obvesti natakarja o zaključku priprave jedi za določeno naročilo. Restavracija lahko ima več kuharjev, vendar vsi uporabljajo aplikacijo preko iste prijave. Omejitev je v podatkovnem modelu. Funkcionalnosti za kuharja so: pregledovanje in potrjevanje naročila, obveščanje natakarja o zaključku priprave jedi ter spremljanje statistike naročil.



Slika 2.1: Diagram primerov uporabe spletnega naročanja

Uporabili smo trenutno najbolj razširjeno arhitekturo, ki jo sestavljajo podatkovna baza, strežnik in aplikacija oziroma odjemalec. To je koncept, ki ga je mogoče prilagajati predvsem z uporabniškega vidika. Slika 2.2 prikazuje arhitekturno rešitev 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.

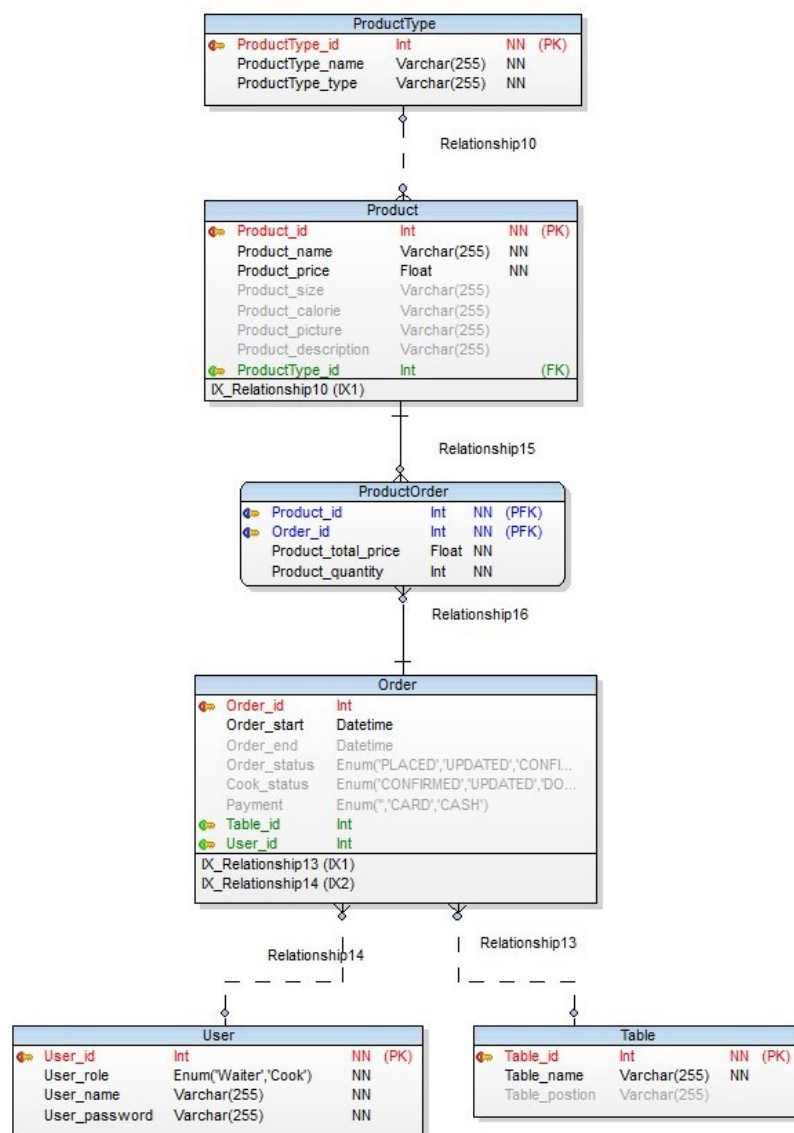


Slika 2.2: Visokonivojska arhitektura

2.1 Podatkovna baza

Na podlagi predvidenih zahtev oziroma funkcionalnosti, ki jih bo imela spletna aplikacija, 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

Tabela *ProductType* je namenjena zapisom za vrste jedi (predjed, glavna jed, sladica) in pijač (sokovi, piva, vina, itd.). Sestavljena je iz atributov: *ID*, *Name* in *Type*. Atribut *Type* je namenjen razlikovanju hrane in pijače v naročilu.

Tabela *Product* je namenjena opisu hrane in pijače ter je sestavljena iz atributov: *ID*, *Name*, *Price*, *Size*, *Calorie*, *Picture* in *Description*. V vrednost atributa *Picture* se zapiše ime slike, ki se prikaže v aplikaciji.

Tabela *Order* je namenjena zapisovanju naročil in njihovih podrobnosti. Sestavljena je iz atributov: *ID*, *Start*, *End*, *OrderStatus*, *CookStatus* in *Payment*. Vsebuje tudi tuji ključ od tabele *Table*, ki določa, na katero mizo je vezano naročilo. Atributa *OrderStatus* in *CookStatus* sta definirana kot naštevalni podatkovni tip (angl. Enumerated Type, ENUM), ki sporočata status naročila med vlogami.

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

Tabela *Table* je namenjena označevanju miz v restavracijah. Sestavljajo jo atributi: *ID*, *Name* in *Position*, v katerega se lahko podrobneje opiše lokacija mize.

Tabela *User* je namenjena le natakarjem in kuharjem. Sestavljena je iz atributov: *ID*, *Role*, *Name* in *Password*.

Strukturo podatkovne baze smo definirali s pomočjo programa Toad Data Modeler. 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, npr. MySQL, Ingres, Microsoft Azurem, Microsoft Access in Microsoft SQL Server.

Podatkovni model na fizičnem nivoju smo implementirali v sistemu za upravljanje podatkovne baze (angl. Database Management System, DBMS)

MySQL. To je eden od odprtokodnih sistemov za upravljanje podatkovnih baz, ki za delo s podatki uporablja jezik SQL (angl. Structured Query Language) [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. Najpomembnejše je, da je sistem zanesljiv, saj odjemalec brez strežnika ne more delovati. Izbrali smo arhitekturo REST (angl. Representational State Transfer) zaradi načel, ki so opisana spodaj [8]. Sama arhitektura omogoča, da odjemalec z zahtevami pridobiva podatke od strežnika, ki jih s pomočjo povezav URI (angl. Uniform Resource Identifier) 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 (angl. Client-server) zahteva ločitev odjemalca od strežnika, kar odjemalcu onemogoča neposredno povezljivost s podatkovno bazo in s tem poenostavi razširljivost uporabniškega dela. Strežnika ne zanima uporabniški vmesnik ali podatki, tako da je enostavnejši in bolj prilagodljiv za uporabo. Tako se lahko uporabniški in strežniški del razvijata ali zamenjujeta neodvisno. V aplikaciji imamo dva različna odjemalca (gost in natakar/kuhar), kar za strežnik ne predstavlja nobenih omejitev, razen pri podatkovni bazi, ki omejuje število hkratnih poizvedb. V naši aplikaciji te omejitve nismo presegli.

- 2.) Brez stanja (angl. 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 (angl. Hypertext Transfer Protocol), se nahajajo bodisi v podatkovni bazi bodisi jih odjemalec na strežnik posreduje z zahtevkom.

3.) Predpomnjenje (angl. Cachable) prinaša izboljšanje zmogljivosti za odjemalca in omogoča razširljivost strežnika, ker se obremenitev zmanjša. V aplikacijah REST se predpomnjenje uporabi za vire, ki to potrebujejo. V naši aplikaciji tega nismo uporabili, saj nimamo tako zahtevnih virov.

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

5.) Izvajanje programske kode na zahtevo (angl. Code on demand) je opcijsko načelo. Strežnik na zahtevo odjemalca pošlje oziroma izvede programsko kodo za odjemalca. To smo vpeljali s pomočjo spletnih vtičnikov (angl. Websocket), ki so podrobneje opisani spodaj.

6.) Enotni vmesnik (angl. Uniform interface) med strežnikom in odjemalcem. Vsak vir mora vsebovati povezavo, ki kaže na svoj relativni 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 sta najpogostejša XML (angl. Extensible Markup Language) in JSON (angl. JavaScript Object Notation).

Knjižnica Flask nam je poenostavila izdelavo strežnika, saj gre za eno izmed najpriljubljenejših spletno aplikacijskih vmesnikov (angl. Framework) [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, narejenim v spletnem ogrodju Django.

Vsaka relativna povezava URI na strežniku predstavlja svoj vir podatkov iz podatkovne baze. Podatki so odjemalcu na voljo v podatkovnem formatu JSON. 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 podat-

kovne baze, kjer spemenljivka *query* predstavlja poizvedbeni stavek. Slika 2.5 prikazuje uporabo te funkcije in relativne poti *drinks*. Strežnik vsebuje 34 relativnih povezav URI 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
```

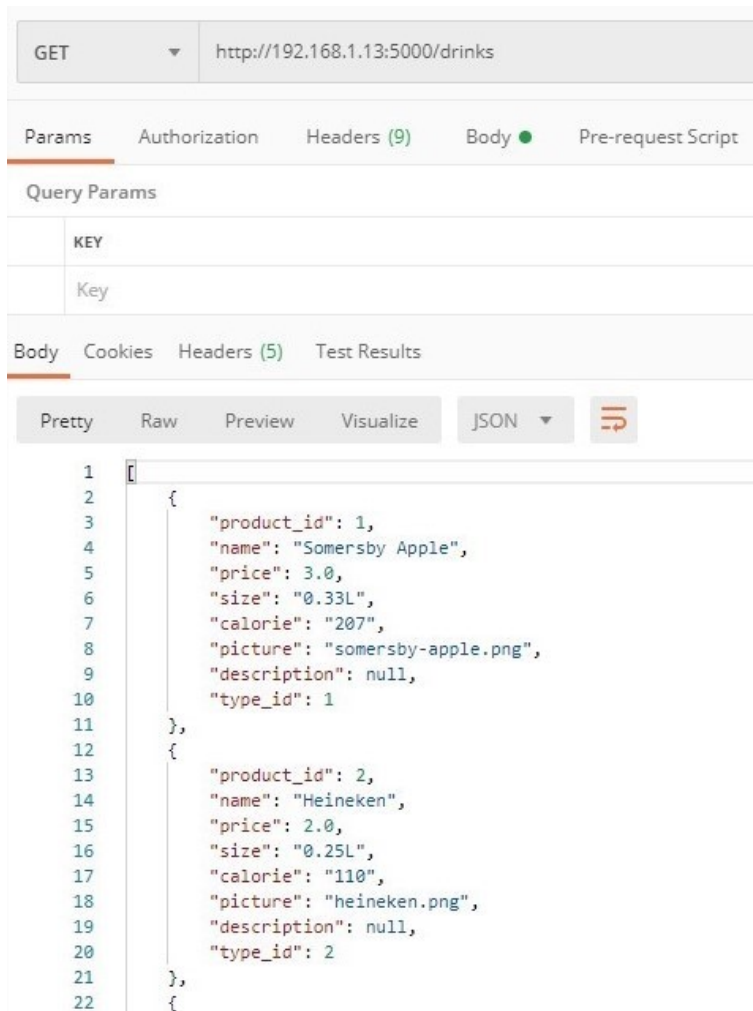
Slika 2.4: Funkcija, ki prebere podatke iz podatkovne baze

```
@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* servira podatke

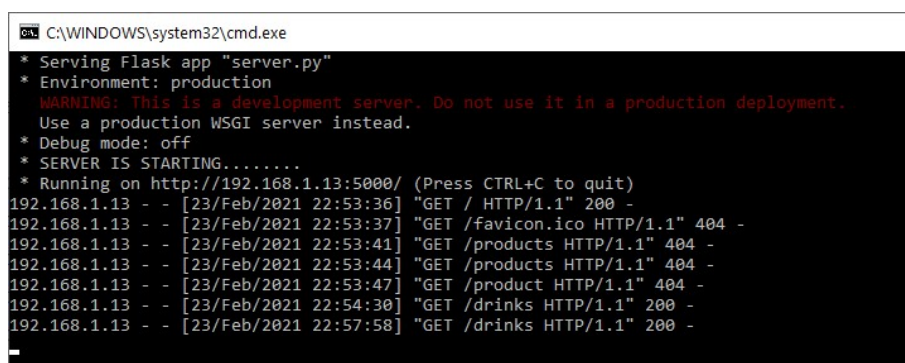
Tako smo dobili vmesnik, ki na zahtevo odjemalca odgovori s podatki v formatu JSON. Na sliki 2.6 je primer zahtevka v programu Postman, ko odjemalec zahteva podatke vseh pijač iz podatkovne baze (metoda GET protokola HTTP). Strežnik omogoča tudi sprejemanje podatkov z metodami PUT in POST. Mi smo uporabili metodo POST za posredovanje podatkov, ki so potrebni za zapis v podatkovno bazo, na strežnik. Spremljanje zahtevkov na

strežniku je mogoče v konzolnem vmesniku (angl. Command-Line Interface, CLI), ki se 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 za odjemalca, smo uporabili spletni vtičnik SocketIO. Implementirali smo ga na strežniku in odjemalcu. Zagotavlja dvosmerno komunikacijo oziroma komunikacijo na podlagi dogodkov. Deluje na vseh platformah, brskalnikih ali napravah. Uporabili smo ga zaradi medsebojnega obveščanja odjemalcev o spremembah v podatkovni bazi. Uporabili smo funkcijo *emit*, ki omogoča dodajanje podatkov in

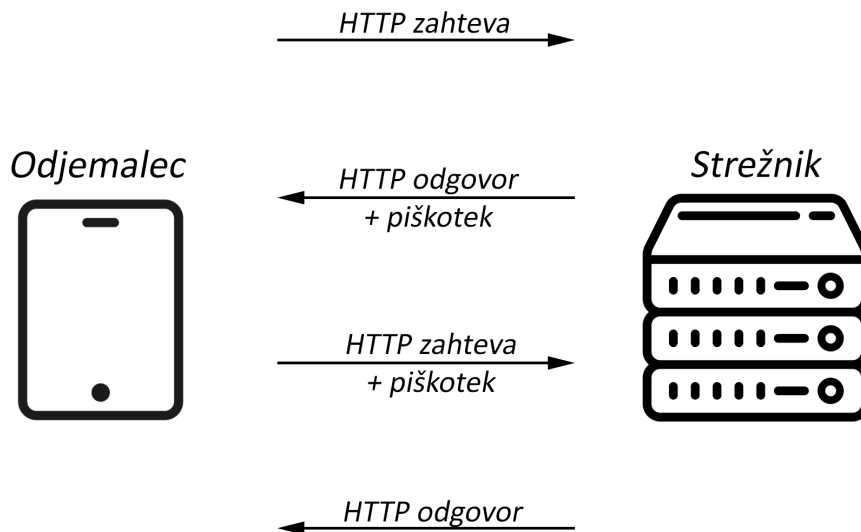
A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window displays the output of a Flask application running in production. The logs show the server starting on http://192.168.1.13:5000/. It receives several GET requests: a successful 200 response for the root path, 404 responses for /favicon.ico and /products, and successful 200 responses for /drinks. The logs also include a warning about using a development server in production and debug mode being off.

```
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

izbiranje načina razpršenega oddajanja (angl. Broadcast). To pomeni, da ob oddajanju strežnika vsi prejemaajo te informacije. Gre predvsem za splošne podatke, tako da ne more priti do zlorabe. Na primer: ob gostovi spremembi naročila se te razlike preverijo na strežniku in vpišejo v podatkovno bazo, natakarka pa se o tem obvesti 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 njihovo urejanje. Prva raven varnosti je avtorizacija odjemalca. Zahtevki, ki se pošiljajo na strežnik, morajo biti omogočeni samo avtoriziranim odjemalcem. Zato smo uporabili piškotke HTTP (angl. Cookies), ki so izdelani za spletne brskalnike in so namenjeni za sledenje, prilagajanje in shranjevanje informacij o posamezni seji uporabnika. Vsi piškotki so shranjeni pri odjemalcu in so kriptografsko zaščiteni pred morebitnimi nepooblaščenimi dostopi. S tem odjemalcu preprečujemo spreminjanje podatkov v piškotkih oz. lahko nedovoljene spremembe podatkov na strežniku zaznamo. Uporabili smo Flask-Login, ki omogoča vse funkcionalnosti za upravljanje uporabniških sej. Na sliki 2.8 je prikazano delovanje piškotkov HTTP. 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 piškotkov HTTP med odjemalcem in strežnikom

2.3 Odjemalec

Odjemalca bi lahko implementirali v spletnih tehnologijah (HTML, CSS in JavaScript) ali pa v namenski mobilni aplikaciji. Odločili smo se za spletne tehnologije, kjer je glavnina odjemalca implementirana v programskem jeziku JavaScript s pomočjo ogrodja Vue. Naredili smo odziven in reaktiven vmesnik, ki deluje v stvarnem času. Vue je eden izmed mnogih, kot npr. Angular, Ember, React, poznan pa je predvsem zaradi enostavnosti upravljanja in izvajanja 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, kjer je podatek lahko vezan na več funkcij oziroma delov programske kode, ki se ob spremembi vrednosti posodobijo. Vue je namenjen izdelavi enostranskih aplikacij (angl. Single-Page Application, SPA), saj vsebuje samo eno datoteko HTML. To prednost smo izkoristili s pomočjo ostalih knjižnic, ki so nam omogočale enostavnejšo 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 razvijalcu omogoča, da se bolj osredotoči na programiranje in ne na njihovo povezovanje 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 aplikacij SPA. Usmerjevalnik je mišljen v smislu usmerjanja na druge komponente (angl. Component), ki v Vue.js predstavljajo druge poglede oziroma odstrani.

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

Vuetify je eden izmed mnogih uporabniških vmesnikov, ki je zgrajen na vrhu Vue.js [11]. V nasprotju z drugimi vmesniki je Vuetify enostaven za učenje z več stotimi komponentami, izdelanimi po specifikacijah Material Design.

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

Programsko izvedbo za odjemalca smo razdelili v tri vloge oziroma dve aplikaciji. Ena aplikacija je namenjena natakarjem in kuharjem, ločuje se s prijavnim oknom in videzom vmesnika. Druga aplikacija je namenjena samo gostom in 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 opisani v naslednjem poglavju.

Ena izmed pomembnih stvari pri obratovanju restavracije je čim hitrejša postrežba, ki jo je mogoče izboljšati s čim hitrejšo komunikacijo. Zato smo, enako kot za strežnik, uporabili spletni vtičnik SocketIO. Vključili smo ga v obeh aplikacijah, in sicer za oddajanje naročil, posodabljanje naročil, obveščanje gosta o stanju naročila itd. Najprej smo hoteli uporabiti samodejno osveževanje na določen časovni interval, vednar je uporaba spletnih vtičnikov hitrejša in učinkovitejša. Slika 2.9 prikazuje primer vtičnikov, ki so uporabljeni za 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");  
    },  
  },  
}
```

Slika 2.9: Uporaba spletnih vtičnikov za gosta

Za potrebe pridobivanja podatkov za odjemalca smo uporabili Axios, ki je namenjen procesiranju zahtevkov HTTP. To pomeni, da podatke, ki jih oglašuje strežnik, s pomočjo te knjižnice pridobimo za odjemalca (slika 2.10).

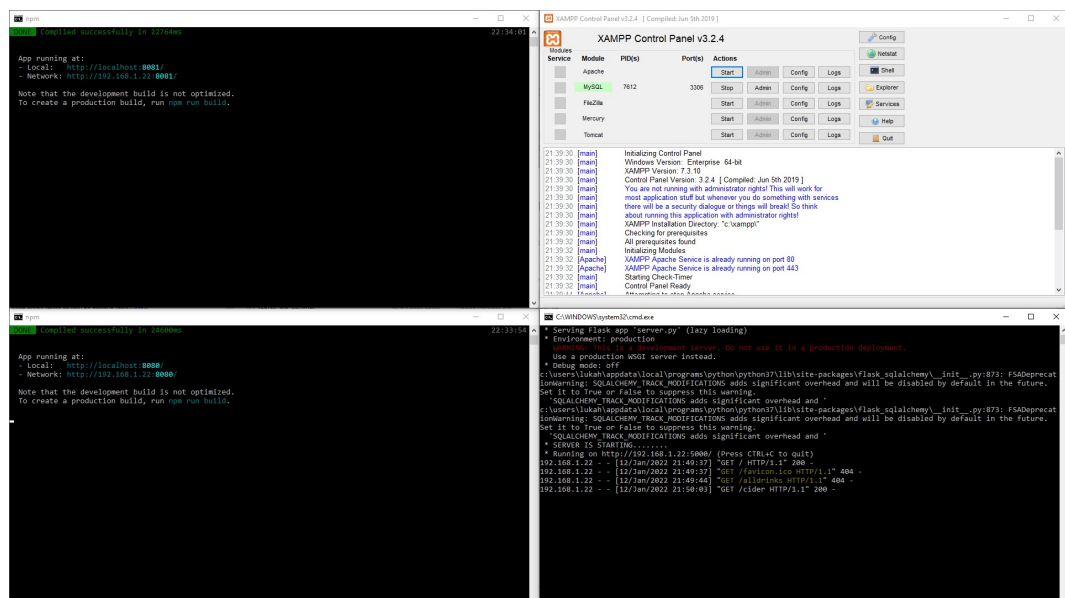
```
allDrinks({ commit }) {  
  axios.get(`${"http://192.168.1.13:5000"}/drinks`).then(response => {  
    commit("ALL_DRINKS", response.data);  
  });  
},  
allFoods({ commit }) {  
  axios.get(`${"http://192.168.1.13:5000"}/foods`).then(response => {  
    commit("ALL_FOODS", response.data);  
  });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

Slika 2.10: Način uporabe Axios v aplikaciji za gosta

Poglavje 3

Delovanje aplikacije

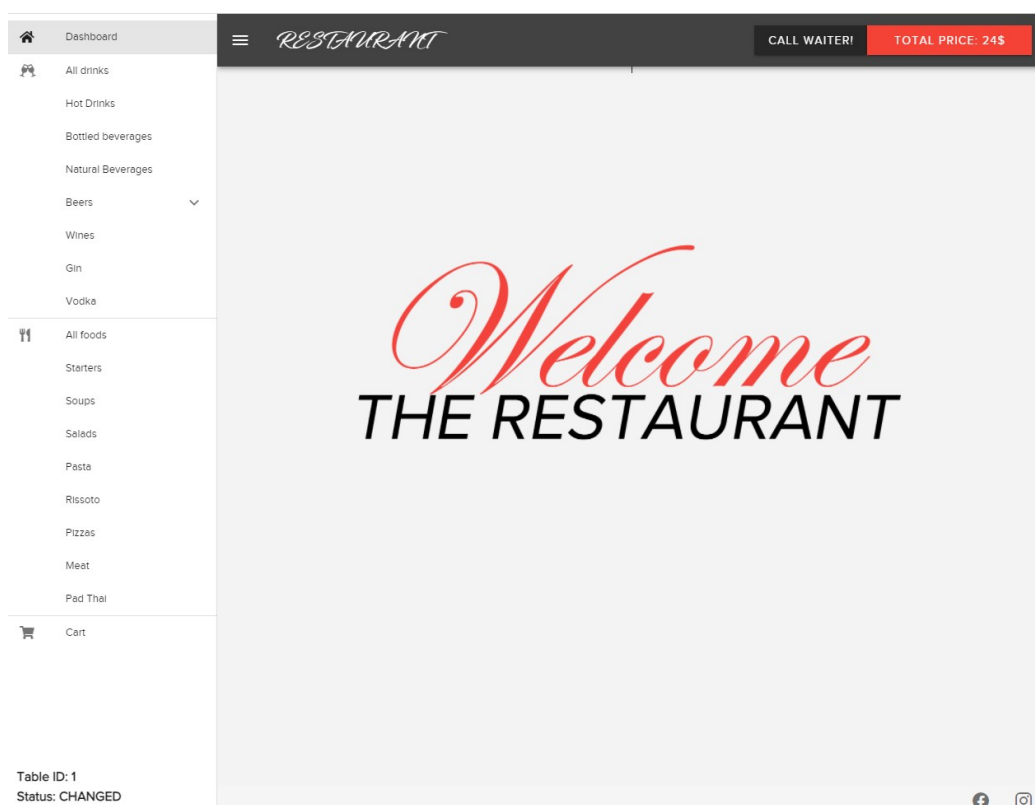
Delovanje aplikacije smo testirali v testnem okolju. Uporabili smo osebni računalnik, na katerem smo postavili podatkovno bazo MySQL, strežnik in aplikaciji za gosta in natakarja/kuharja. Slika 3.1 prikazuje testno okolje.



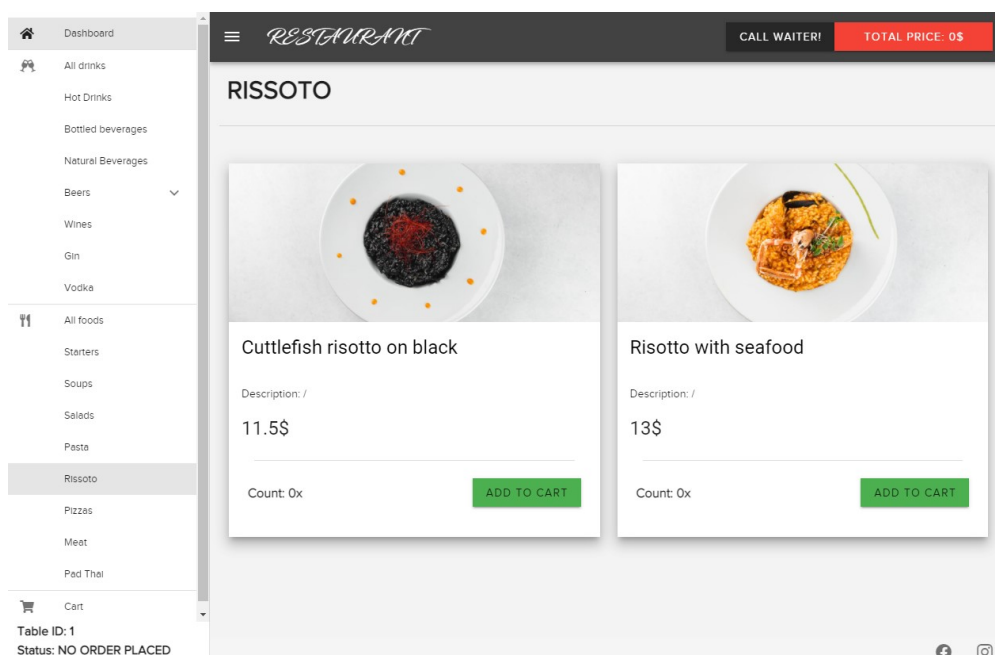
Slika 3.1: Testno okolje na osebni računalniku

3.1 Vmesnik za gosta

Začetni pogled vmesnika za gosta vsebuje napis za dobrodošlico (slika 3.2), ki bi ga lahko zamenjali oglaševanje, predstavitev restavracije ali karkoli bi si potencialni kupec zaželel imeti. V zgornjem desnem kotu se nahajata gumb *call waiter* za priklic natakarja in števec skupne cene artiklov v nakupovalni košarici. Gumb *call waiter* je namenjen gostom, ki aplikacije ne želijo uporabljati, ali v primeru pomoči, če ima gost kakšna vprašanja ali pride do kakršnihkoli težav. V spodnjem levem kotu se nahajata status in identifikacijska številka naročila. Zavihki na levi strani predstavljajo seznam vseh vrst hrane in pijače, ki kažejo na podstrani ponudbe restavracije (slika 3.3). Gostu to predstavlja ponudbo, med katero izbira pri dodajanju hrane in pijače v nakupovalno košarico.

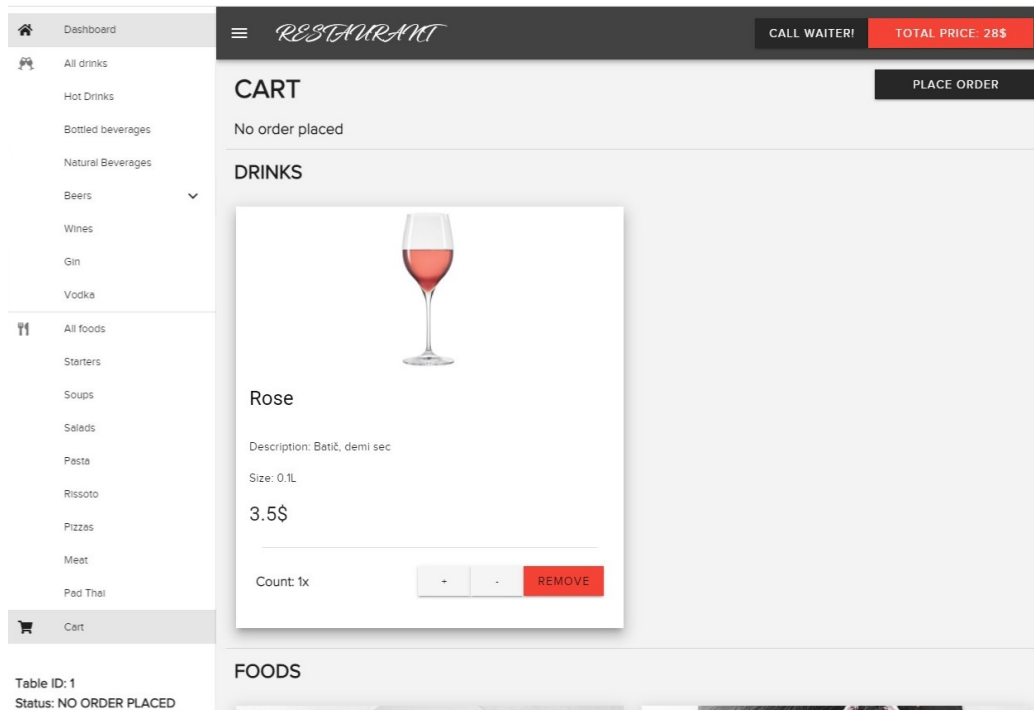


Slika 3.2: Začetni pogled vmesnika za gosta

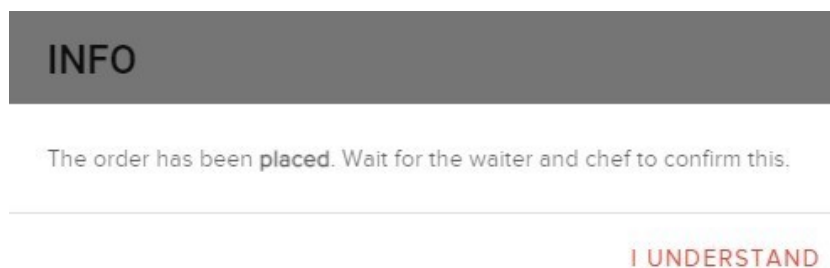


Slika 3.3: Seznam artiklov znotraj vrste rižot

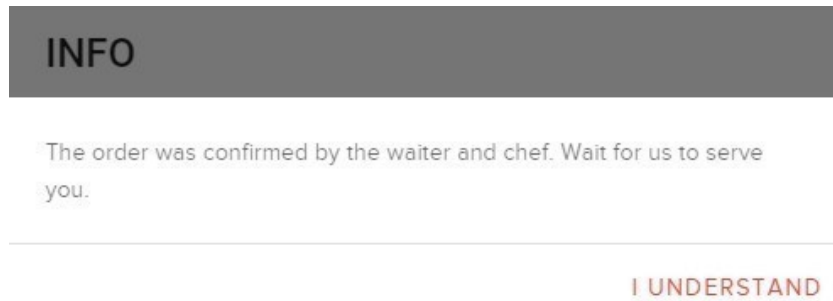
Vsaka hrana ali pijača vsebuje sliko, ime, opis in ceno. Poleg tega vsebuje še gumb *add to cart* (dodaj v košarico), ki ob kliku izgine in prikaže tri nove gumbe *+* (povečaj količino), *-* (zmanjšaj količino) in *remove* (odstrani). Nakupovalna košarica oziroma *cart* je skupno mesto vse hrane in pijače v seznamu za naročilo (slika 3.4). Vse slike hrane in pijače so shranjene v datotečnem sistemu spletnega strežnika. Naročilo se odda s klikom na gumb *place order*, ki gosta preusmeri na prvo stran in obvesti s pojavnim sporočilom, prikazanim na sliki 3.5. Ko je naročilo oddano, lahko gost ponovno dodaja hrano in pijačo v košarico, vendar je že oddani hrani in pijači onemogočeno zmanjšanje količine ali brisanje. Naročilo je sprejeto, ko ga natakar potrdi, kar spremeni status naročila in prikaže pojavno sporočilo, predstavljeno na sliki 3.6. Če natakar zavrne naročilo, ga mora gost pregledati in ponovno oddati. Tudi v tem primeru je gost obveščen s statusom in pojavnim sporočilom, vidnim na sliki 3.7. Naročilo se zaključi s klikom na gumb *request receipt*, ki odpre pojavno okno (slika 3.8) za izbiro načina plačila.



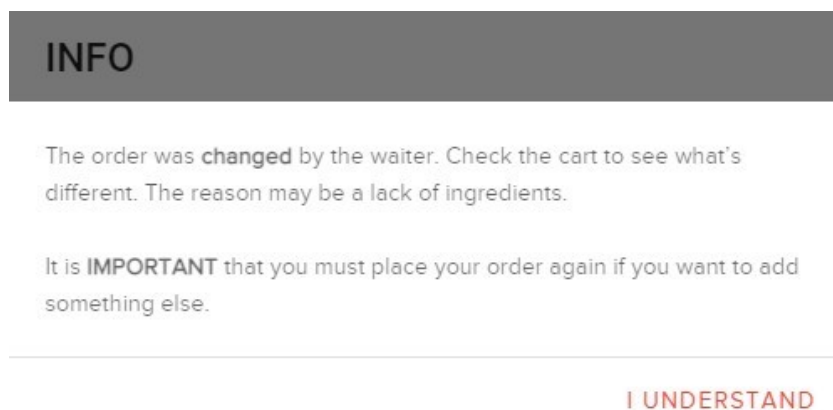
Slika 3.4: Primer naročila



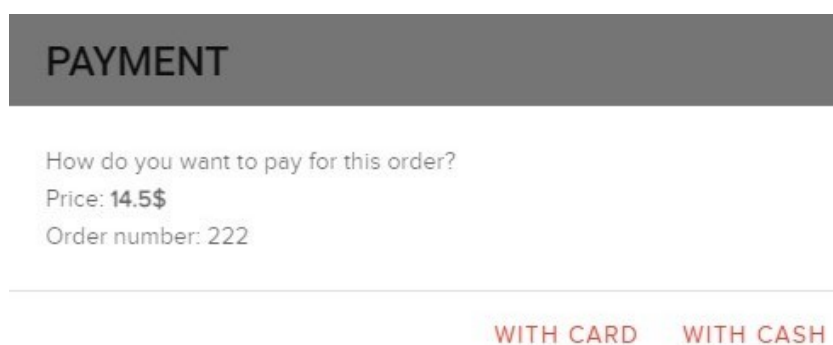
Slika 3.5: Pojavno sporočilo ob uspešni oddaji naročila



Slika 3.6: Pojavno sporočilo ob natakarjevi potrditvi naročila



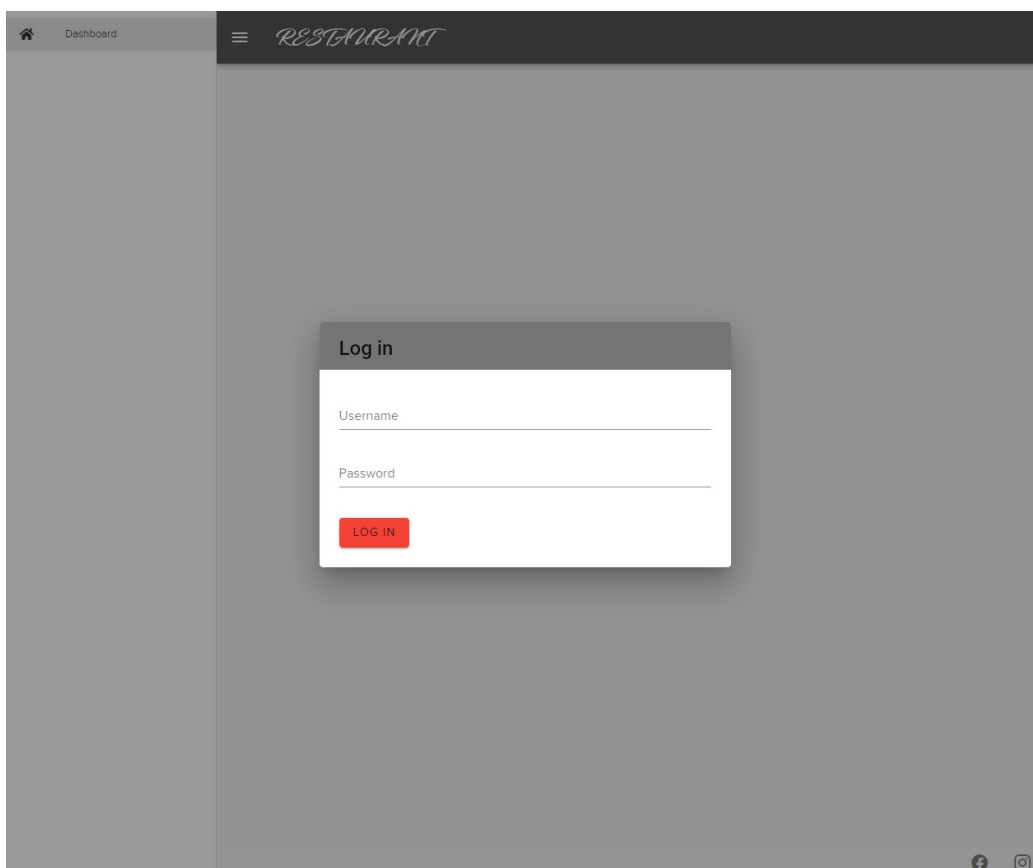
Slika 3.7: Pojavno sporočilo ob natakarjevi zavrnitvi naročila



Slika 3.8: Pojavno okno z možnostjo izbire načina plačila

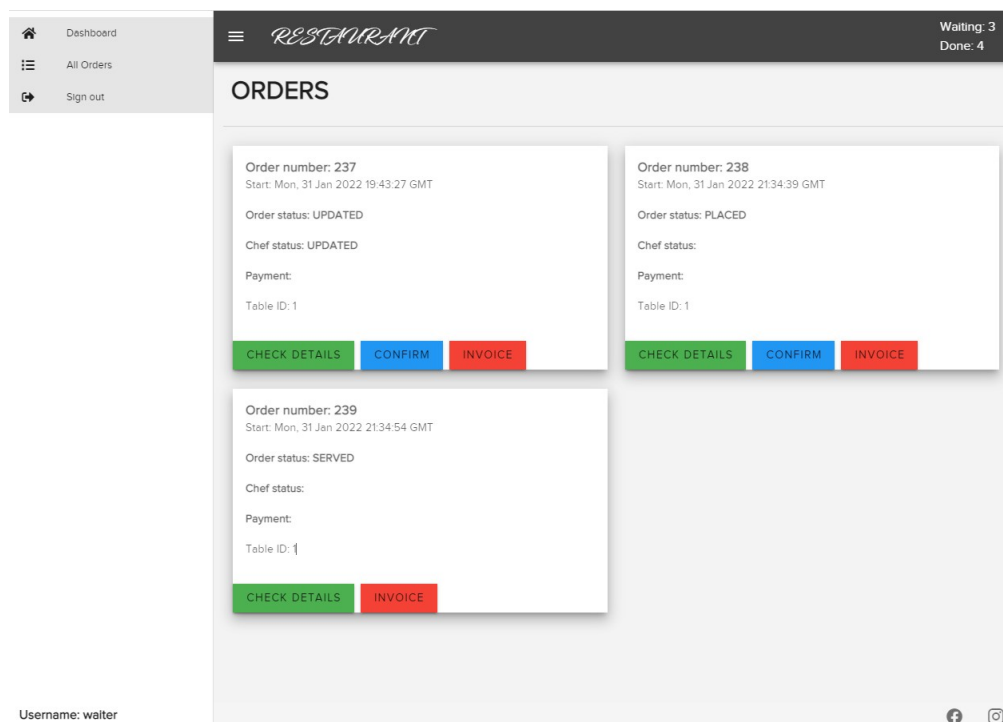
3.2 Vmesnik za natakarja in kuharja

Začetni pogled vmesnika je enak za natakarja in kuharja, saj gre za skupno aplikacijo, kjer se pogledi razlikujejo glede na vlogo uporabnika, ki je določena v podatkovni bazi. Nismo naredili ločene aplikacije, saj ni bilo potrebe, funkcije za oba uporabnika so namreč zelo podobne. Slika 3.9 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 stran ob uspešni prijavi prikazuje napis za hitrejšo razlikovanje med vlogami. Natakar ima v desnem zgornjem kotu še števec čakajočih in zaključenih naročil.

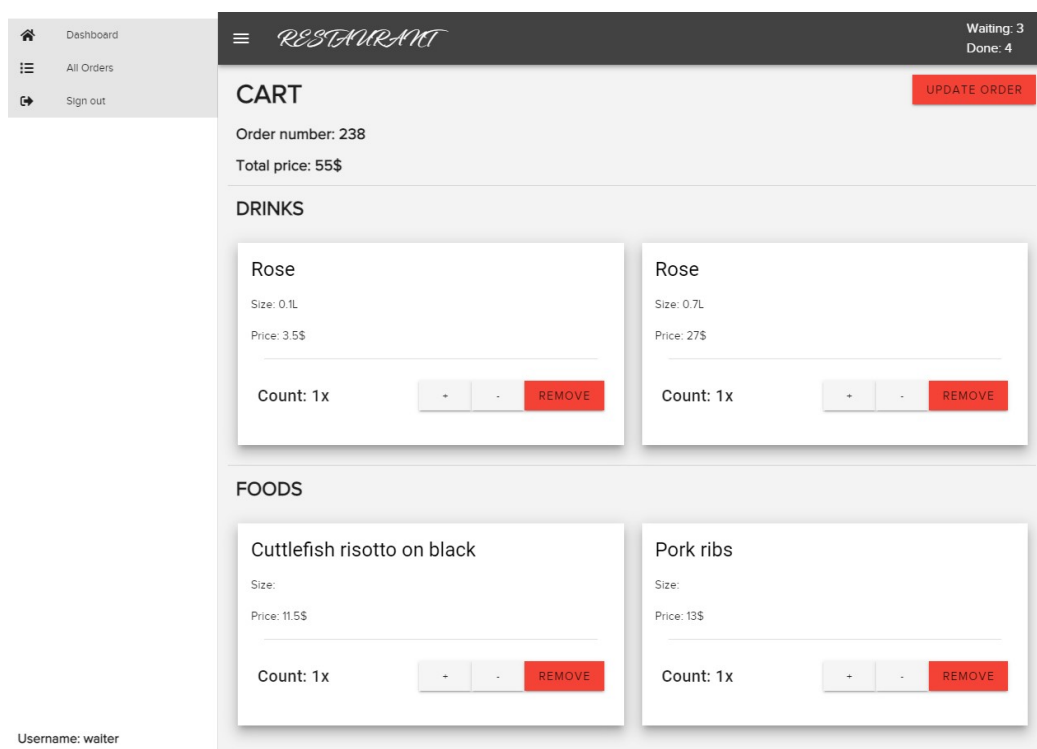


Slika 3.9: Prijavno okno za natakarja in kuharja

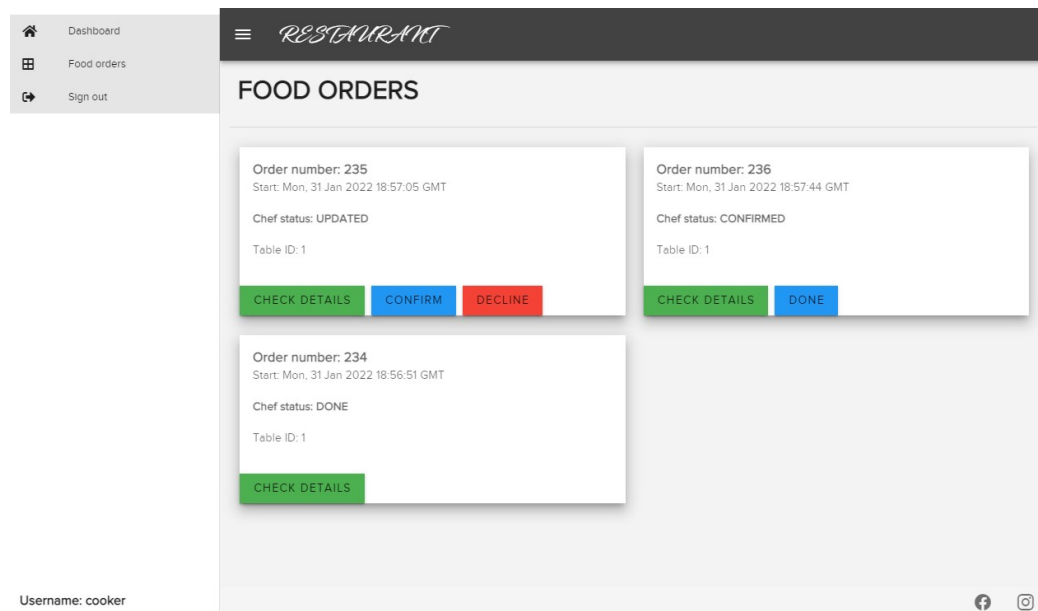
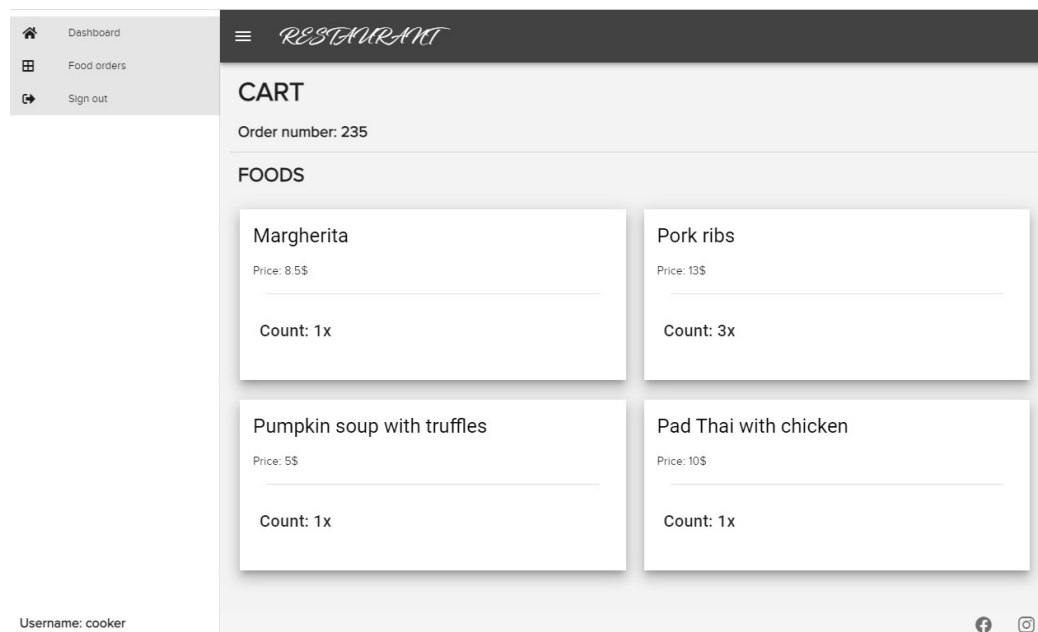
Natakarju se v zavihku *all orders* pojavijo vsa nezaključena naročila (slika 3.10). Vsako naročilo vsebuje naslednje podatke: identifikacijska številka naročila, čas oddaje naročila, status naročila, status kuharja, način plačila in številka mize. Izvaja lahko popoln nadzor nad naročili, vendar ob vsaki izvedeni akciji obvesti gosta (slike pojavnih sporočil iz prejšnjega poglavja). Novo naročilo mora natakar najprej potrditi z gumbom *confirm* ali urediti z gumbom *check details* (slika 3.11). Natakar lahko ureja celotno naročilo, kar pomeni, da lahko spreminja količino hrane in pijače ali jo briše iz naročila. Ko zaključi urejanje, mora klikniti na gumb *update order*. Celotno naročilo lahko potrdi šele, ko dobi kuharjevo potrditev o hrani. Ko je naročilo potrjeno, natakar čaka na kuharjevo potrditev o pripravi hrane, da jo lahko postreže. Natakar postreženo naročilo označi s klikom na gumb *served*. Če 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*

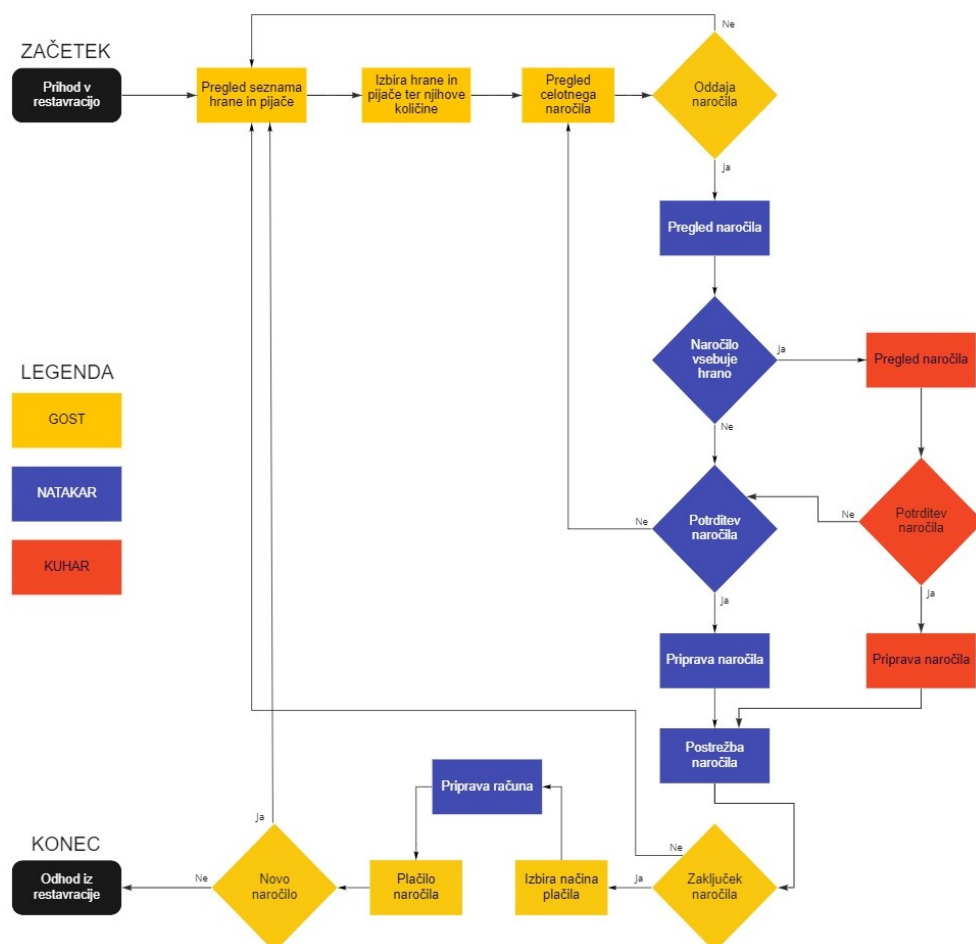
Slika 3.11: Zavihek *check details*

Kuharju se v zavihku *food orders* pojavijo vsa nezaključena naročila, ki vsebujejo hrano (slika 3.12). Vsako naročilo vsebuje naslednje podatke: identifikacijska številka naročila, čas oddaje naročila, status kuharja in številka mize. Kuhar mora naročilo najprej pregledati z gumboma *check details* (slika 3.13) ter ga sprejeti ali zavrniti z gumbom *confirm* in *decline*. Naročilo lahko kuhar zavrne, če mu zmanjka sestavin. Ko kuhar zaključi pripravo hrane, o tem obvesti natakara s klikom na gumb *done*, ki izbriše naročilo s seznama. Ob vsaki izvedeni akciji se pri vsakem naročilu spremeni status kuharja, ki je lahko: *updated*, *confirmed*, *decline* in *done*. Status *updated* se uporabi, če gost k naročilu doda izdelke.

Slika 3.12: Zavihek *food orders*Slika 3.13: Zavihek *check details*

3.3 Primer uporabe

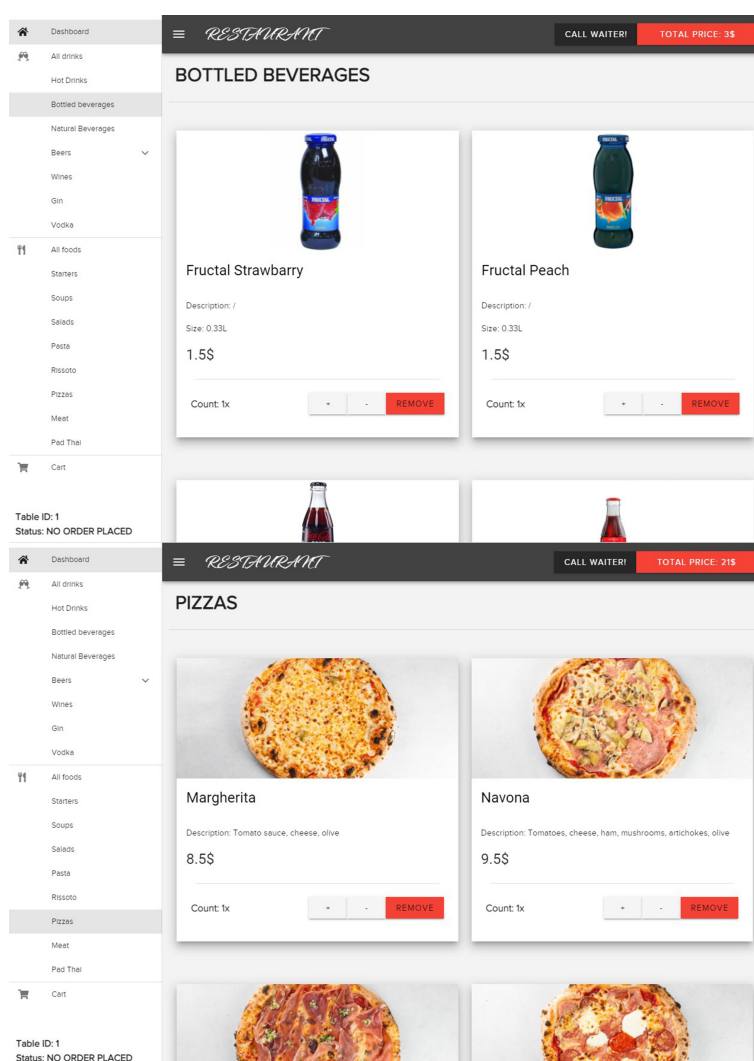
Uporaba aplikacije je prikazana na spodnjem diagramu poteka (slika 3.14).



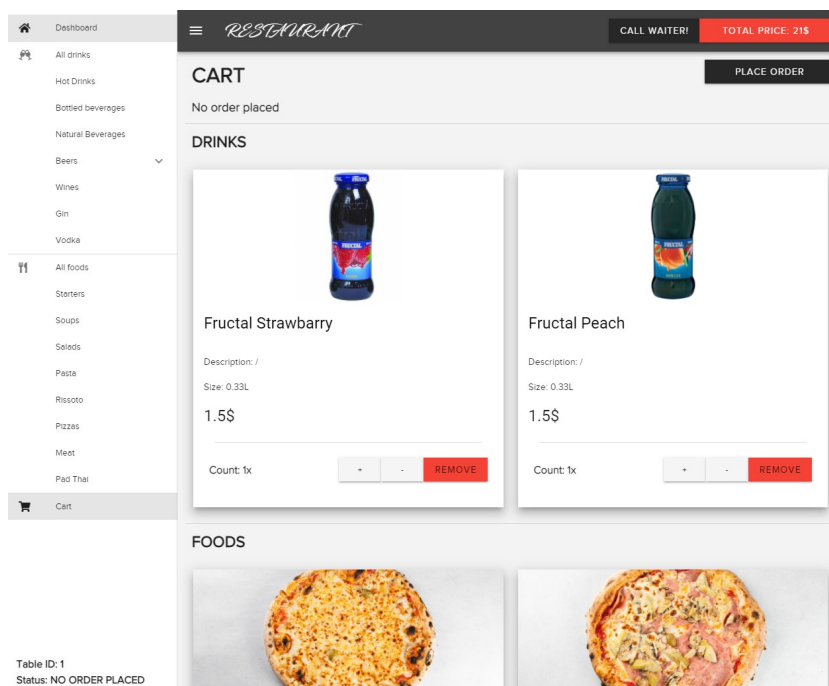
Slika 3.14: Postopek izvedbe in zaključka elektronskega naročila v restavraciji

Gost s sprehajanjem skozi menije izbira hrano in pijačo ter jo sproti dodaja v naročilo (slika 3.15). V našem primeru gost izbere dva sokova in dve pici. Pregled naročila izvede v košarici, kjer naročilo tudi odda (sliki 3.16 in 3.17). Natakar in kuhar se najprej prijavita, da se omogoči pregled naročila. Na seznamu se jima prikaže novo naročilo, pri čemer morata preveriti, ali imata vse sestavine za pripravo (sliki 3.18 in 3.19). V našem primeru naročilo vsebuje hrano, tako da mora naročilo najprej potrditi kuhar

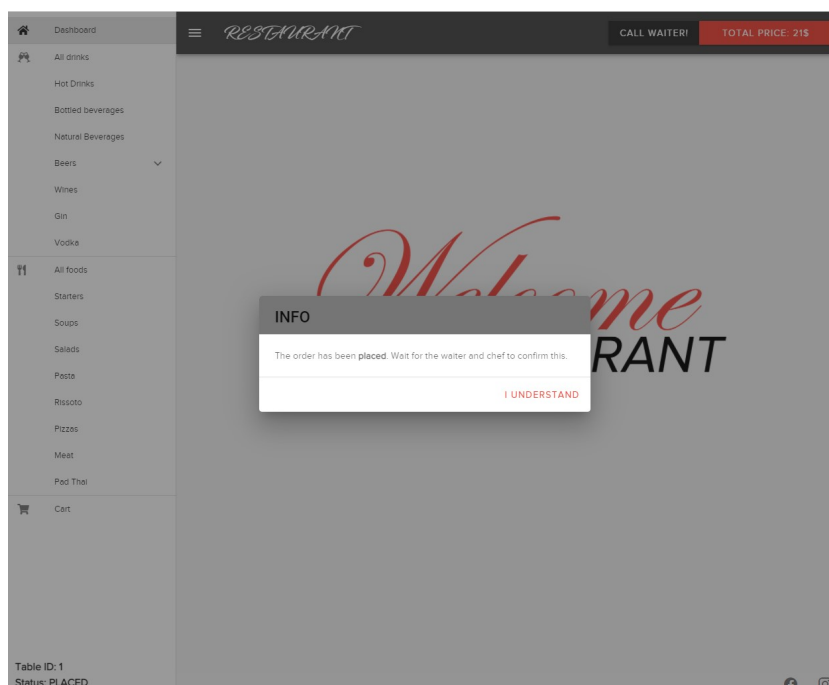
(sliki 3.20 in 3.21), da lahko natakar celotno naročilo potrdi gostu (sliki 3.22 in 3.23). Ko je hrana pripravljena, kuhar o tem obvesti natakarja. Natakar naročilo postreže in naročilo označi kot postreženo. Gost zaključi naročilo z zahtevanjem račun v zavihku košarica, kjer je naročilo tudi oddal. Način plačila izbere ob kliku na zahtevo (slika 3.24). Natakar je obveščen o zahtevi za račun in načinu plačila gosta (slika 3.25). S tiskanjem računa natakar zaključi naročilo, s čimer se to izbriše s seznama aktivnih naročil.



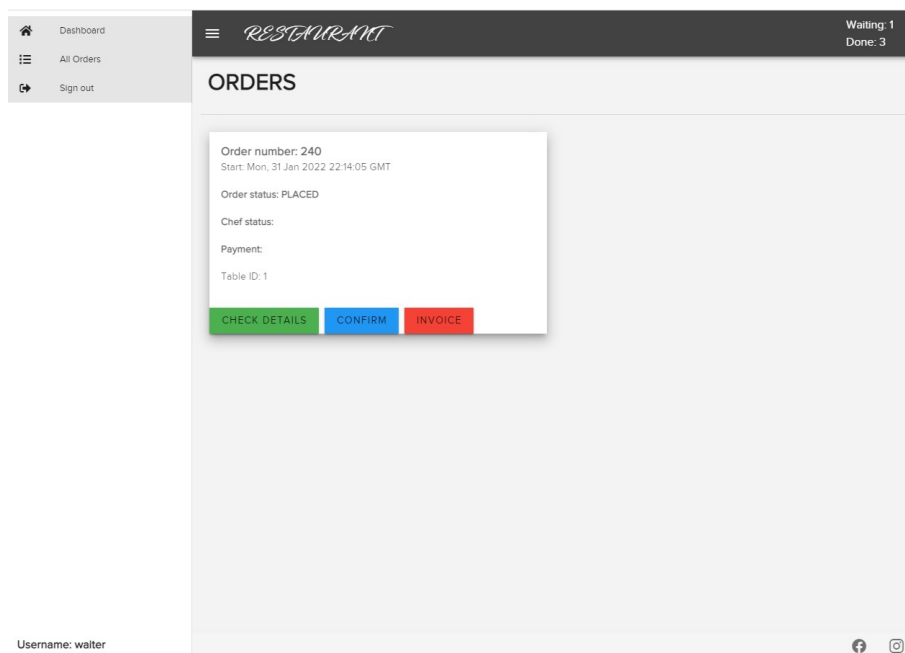
Slika 3.15: Pregled in izbiranje hrane in pijače



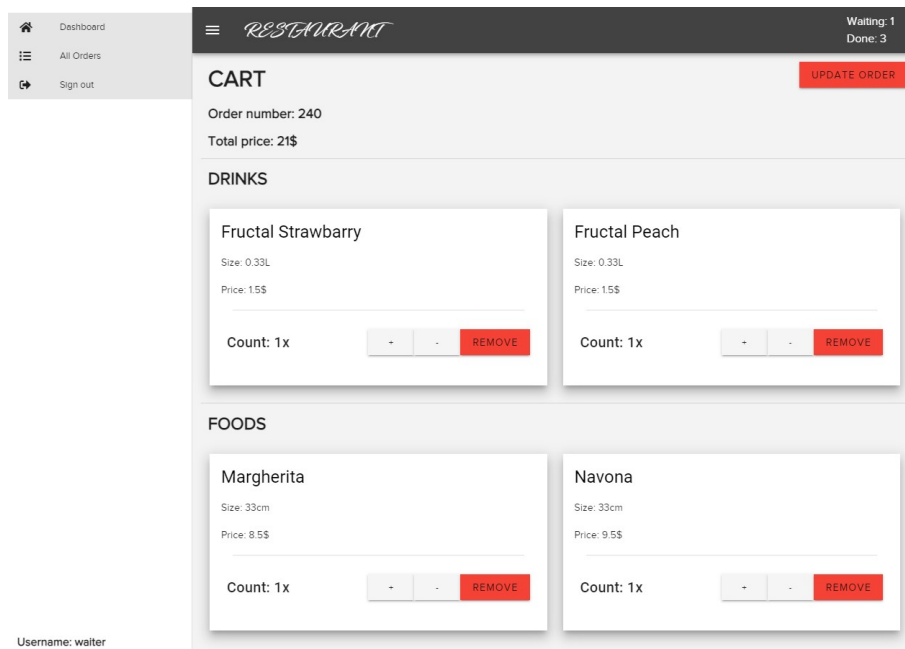
Slika 3.16: Pregled košarice



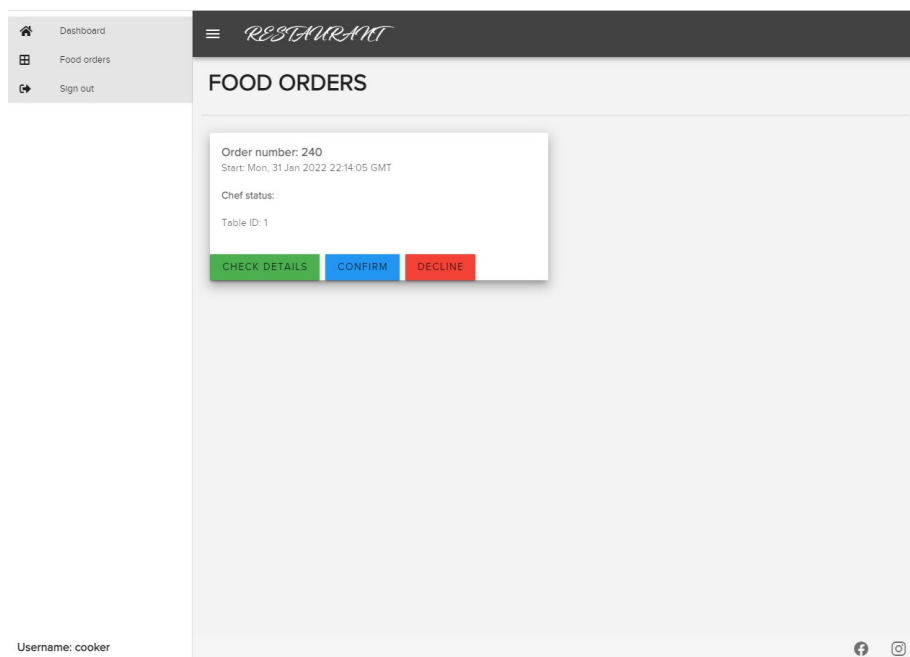
Slika 3.17: Oddajanje naročila



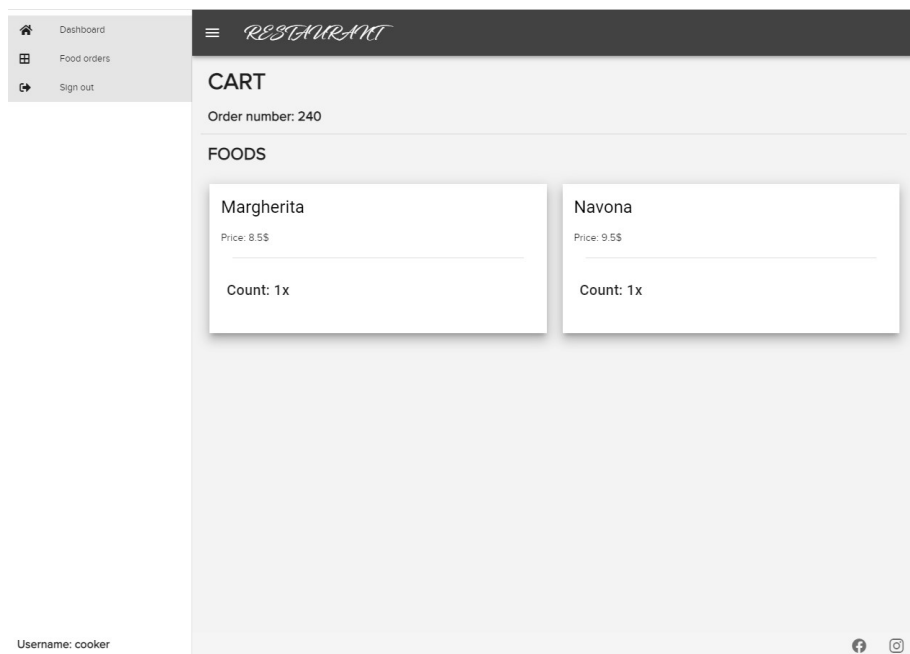
Slika 3.18: Pregled naročila, kot ga vidi natakar



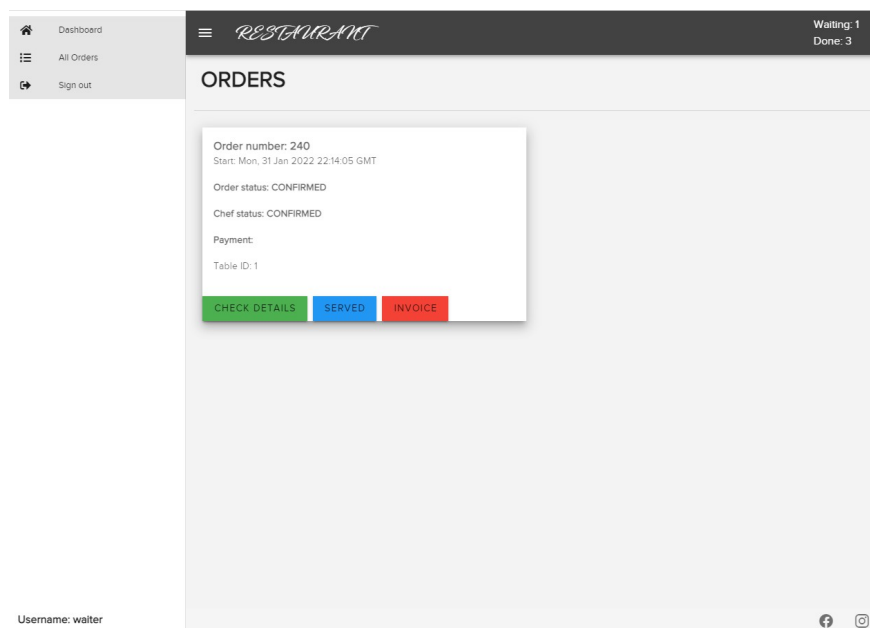
Slika 3.19: Pregled podrobnosti naročila, kot ga vidi natakar



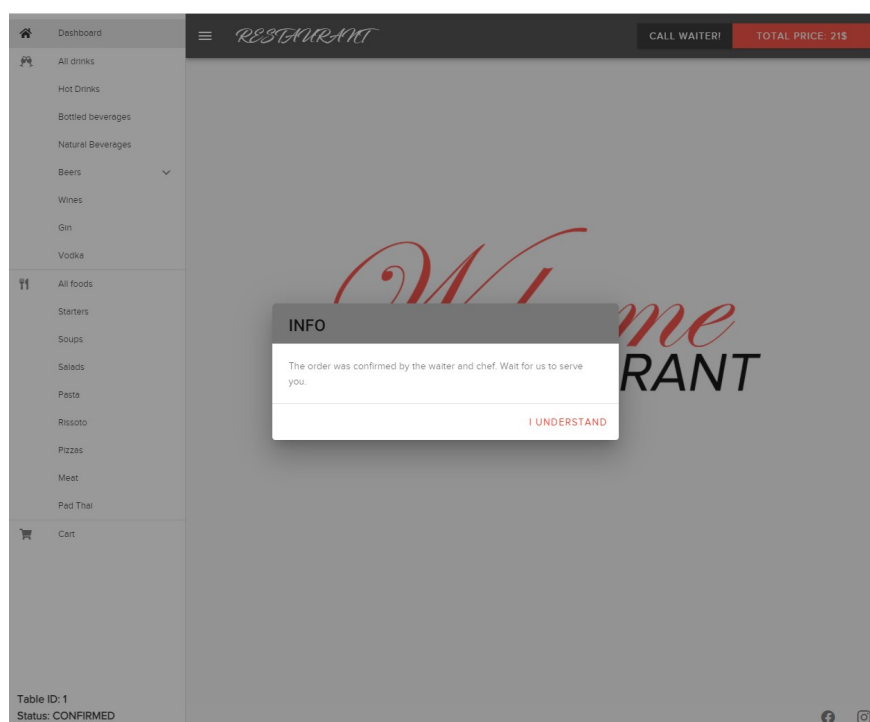
Slika 3.20: Pregled naročila, kot ga vidi kuhar



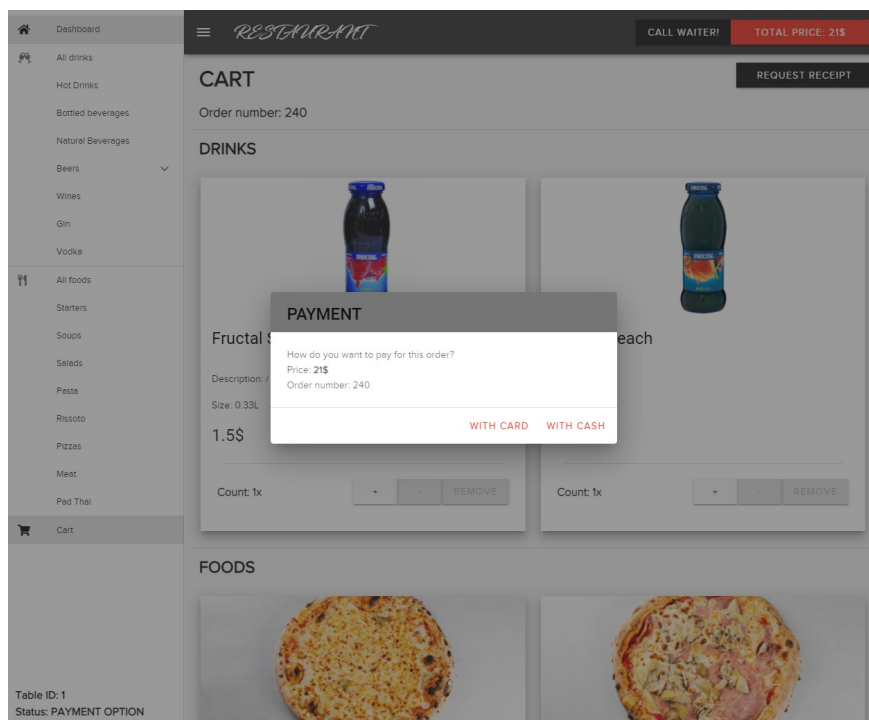
Slika 3.21: Pregled podrobnosti naročila, kot ga vidi kuhar



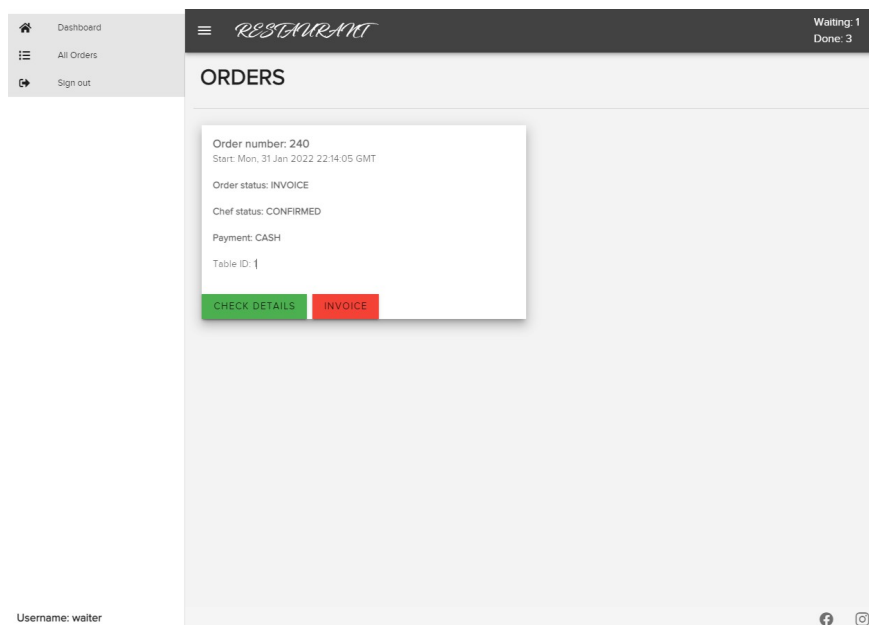
Slika 3.22: Potrjevanje naročila



Slika 3.23: Obveščanje gosta o sprejemu naročila



Slika 3.24: Zahtevanje računa in izbira načina plačila



Slika 3.25: Zaključevanje naročila s tiskanjem računa

3.4 Implementirana rešitev

Trenutna rešitev omogoča implementacijo le znotraj ene restavracije. Aplikacija bi bila nameščena na lokalnem spletnem strežniku. Naročanje bi gost izvajal s pomočjo tablic, ki bi bile nameščene na vsaki mizi v restavraciji.

Nadgradnja bi predstavljala uporabo aplikacije za naročanje z branjem kode QR (angl. Quick Response), ki bi bila nameščena na vsaki mizi v restavraciji. V kodi bi bila zapisana ime restavracije in številka mize. Kodo bi gost skeniral z mobilno napravo s preusmeritvijo v aplikacijo. Celoten sistem bi deloval v odprtem internetnem omrežju, tako za goste kot tudi natakarje in kuharje. Strežnik bi lahko implementirali za eno ali večje število restavracij in bi omogočal storitev vsem restavracijam po Sloveniji. Treba bi bilo zagotoviti, da ne bi prihajalo do fantomskih naročil, kjer bi nepridipravi oddajali neveljavna naročila. Temu bi se lahko izognili na dva načina. Prvi način bi bil, da bi moral biti uporabnik prijavljen preko lokalnega omrežja, s čimer bi overil svojo dejansko prisotnost v restavraciji. Drugi način bi bilo geslo, ki bi ga uporabnik prejel od natakarja.

3.5 Izboljšave

Potrebne izboljšave, da bi lahko aplikacijo ponudili potencialnim kupcem, so:

- 1.) Možnost, da lahko vsak posameznik za mizo dodaja hrano in pijačo k naročilu.
- 2.) Možnost hkratnega delovanja več kuharjev, kot smo to implementirali za natakarje.
- 3.) Administrativni dostop za urejanje menijev.
- 4.) Spremljanje zaloge hrane in pijače.
- 5.) Dodatno spremljanje hrane in pijače za kuharja/natakarja, kot npr. katera pijača in hrana je že bila postrežena.

Napredne izboljšave:

- 1.) Statistika za lastnika restavracije, ki bi poleg vseh podatkov izračunala predvideno oceno nabave za prihodnji mesec.

2.) Neposredno brezstično plačevanje s plačilno kartico za gosta.

3.) Če bi aplikacija delovala v odprtem internetnem omrežju, bi lahko restavracije oglaševale dostavo ali sprejemale naročila za dostavo hrane in pijače.

Poglavje 4

Sklepne ugotovitve

Namen diplomske naloge je bil izdelati enostavno aplikacijo, ki bi olajšala delo natakarjem in nudila kakovostnejšo postrežbo gostom. Mislimo, da je aplikacija v okviru diplomskega dela realizirana v skladu z zahtevami, pričakovanji in cilji. Uporaba aplikacije gostom ponuja drugačno uporabniško izkušnjo, ki omogoča tudi klasično izbiro naročanja z neposrednim stikom z natakarjem. S tem smo želeli poudariti, da aplikacija ni namenjena nadomestitvi delovnih mest natakarjev, vendar omogoča digitalizacijo procesov pri izvajanju njihovega dela. Natakarjeva glavna naloga bi še vedno bila postrežba gosta in priprava pijače.

Aplikacija bi bila s predstavljenimi izboljšavami primerna prototipna rešitev za uporabo. Uporabljali bi jo lahko v vsaki restavraciji, če ne drugače kot pomoč ob veliki zasedenosti. Trenutno, v času krize, povezane z epidemijo covida-19, bi takšna aplikacija omogočala določeno podporo za delovanje restavracij tudi pri spletnem naročanju.

Literatura

- [1] About eslint. Dosegljivo: <https://eslint.org/docs/about/>. [Dostopano: 1. 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. 2. 2021].
- [4] Vue cli overview. Dosegljivo: <https://cli.vuejs.org/guide/>. [Dostopano: 1. 11. 2020].
- [5] Vue router. Dosegljivo: <https://router.vuejs.org/>. [Dostopano: 1. 11. 2020].
- [6] Web application architecture: Best practices and guides. Dosegljivo: <https://lanars.com/blog/web-application-architecture-101>. [Dostopano: 14. 6. 2021].
- [7] What is mysql? Dosegljivo: <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>. [Dostopano: 23. 2. 2021].
- [8] What is rest. Dosegljivo: <https://restfulapi.net/>. [Dostopano: 1. 5. 2021].
- [9] What is vuex? Dosegljivo: <https://vuex.vuejs.org/>. [Dostopano: 1. 11. 2020].

- [10] Why is flask good? Dosegljivo: <https://www.fullstackpython.com/flask.html>. [Dostopano: 1. 11. 2020].
- [11] Why vuetify? Dosegljivo: <https://vuetifyjs.com/en/introduction/why-vuetify/>. [Dostopano: 1. 11. 2020].