

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 L<sup>A</sup>T<sub>E</sub>X.*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogu:

Tematika naloge: Elektronsko naročanje v restavracij

Besedilo teme diplomskega dela študent prepiše iz študijskega informacijskega sistema, kamor ga je vnesel mentor. V nekaj stawkih 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 Uvod</b>	<b>1</b>
<b>2 Načrtovanje in razvoj aplikacije</b>	<b>3</b>
2.1 Podatkovna baza . . . . .	5
2.2 Strežnik . . . . .	7
2.3 Odjemalec . . . . .	12
<b>3 Delovanje aplikacije</b>	<b>17</b>
3.1 Vmesnik za gosta . . . . .	21
3.2 Vmesnik za natakarja in kuhanja . . . . .	24
3.3 Implementacija programa v realnosti . . . . .	28
3.4 Izboljšave . . . . .	29
<b>4 Sklepne ugotovitve</b>	<b>31</b>
<b>Literatura</b>	<b>33</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>HTTP</b>	hypertext transfer protocol	protokol za prenos hiperteksta
<b>SUPB</b>	database management system	sistem za upravljanje s podatkovno bazo
<b>API</b>	application programming interface	programskega vmesnika
<b>QR</b>	qucik response	hitro odziv



# Povzetek

**Naslov:** Elektronsko naročanje v restavraciji

**Avtor:** Luka Horvat

Diplomsko delo zajema analizo, načrtovanje in razvoj spletne aplikacije, ki je namenjena elektronskemu naročanju hrane in pičače v restavracijah. Namen aplikacije je olajšati delo natakarjem in nuditi bolj kakovostno postrežbo gostom. Ključne funkcionalnosti so oddajanje naročil s strani gosta ter nadzor in upravljanje naročil s strani natakarja in kuvarja. Omogočeno je naročanje preko aplikacije ali z neposrednim stikom z natakarjem. Aplikacija je zasnovana po visokonivojski arhitekturi, ki jo sestavljajo aplikacija, strežnik in podatkovna baza. Za razvoj aplikacije se je uporabil programski jezik JavaScript, Python in MySQL za podatkovno bazo. Razvita rešitev je pravljena za implementacijo v restavracijah.

**Ključne besede:** spletna aplikacija, elektronsko naročanje, restavracije, visokonivojska arhitektura.



# 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.



# Poglavlje 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. Glavni razlog, da jim ni uspelo, je bilo sabotiranje s strani natakarjev, saj so misli da bo tehnologija zamenjala njegove službe. 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 je lahko realiziramo, smo se odločili narediti diplomsko nalogu na to tematiko.

Zamislili smo si sistem za oddajanje naročil v restavracijah, ki ne bi bil namenjen zamenjavi ljudi v strežbi, temveč kot pregledovalnik ponudbe 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 odločala 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.



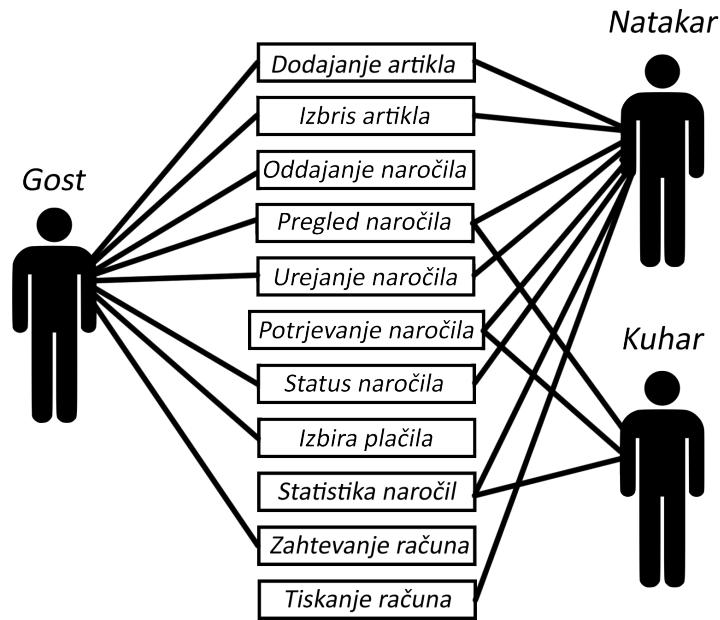
## Poglavlje 2

# 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 sestavlja dve aplikaciji. Ena aplikacija je namenjena gostom, druga pa natakarjem in kuharjem. Restavracija ima lahko več miz, vendar za eno mizo je lahko hkrati odprt eno naročilo. To pomeni, da morajo vsi gosti za mizo naročati skupaj.

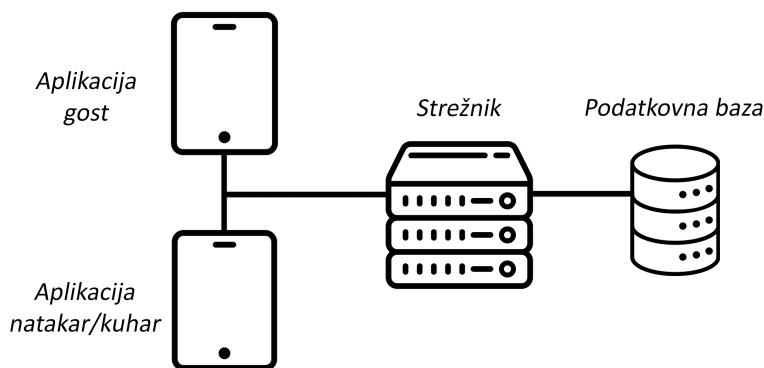
Gost lahko odda, spremeni ali zaključi naročilo. Funkcionalnosti gosta so: dodajanje in brisanje artiklov iz naročila, oddajanje, urejanje in pregledovanje naročila, spremeljanje statusa naročila in zahtevanje računa. V restavraciji je lahko več natakarjev, ki svoje delo opravljajo istočasno. Funkcionalnosti natakarja so: dodajanje in brisanje artiklov iz naročila, urejanje in pregledovanje naročila, spremeljanje statusa naročila, spremeljanje statistike naročil in možnost tiskanja računa za določeno naročilo. Kuhar lahko sprejeme ali zavrne naročilo. Restavracija ima lahko več kuharjev, vendar vsi uporabljajo aplikacijo preko iste prijave. Omejitev je v podatkovnem modelu. Funkcionalnosti kuharja so: pregledovanje in potrjevanje naročila, obveščanje natakarja o zaključku priprave jedi ter spremeljanje statistike naročil.

Uporabili smo arhitekturo 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 arhi-



Slika 2.1: Diagram primerov uporabe spretnega naročanja

tekturno 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 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*.



Slika 2.3: Logični podatkovni model

Tabela *ProductType* je namenjana 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 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 zapise ime slike, ki se prikaže v aplikaciji. Vse slike so shranjene v datotečnem sistemu spletnega strežnika.

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 *Order* je namenjena zapisovanju naročil in njegovih 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 ENUM podatkovnega tip (preddefinirane vrednosti) in sporočata status naročila med vlogami.

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

Tabela *User* 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 lažjem 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 sistemu za

upravljanje s podatkovno bazo (DBMS), ang. database management system (SUPB), MySQL. To je eden od odprtokodnih sistemov za upravljanje s podatkovni bazami, ki za delo s podatki uporablja jezik SQL (ang. 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. Najbolj pomembno je, da je sistem zanesljiv, saj odjemalec brez njega 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 povezav URI (ang. 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 (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 zanima 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 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 uporabljam nobenih globalnih spremenljivk. Vsi podatki, ki so potrebni, da strežnik odgovori na zahtevo HTTP

(ang. hypertext transfer protocol), se nahajajo bodisi v podatkovni bazi ali pa jih odjemalec na strežnik posreduje z zahtevkom.

3.) Predpomnjenje (ang. Cachable) prinaša izboljšanje zmogljivosti na strani odjemalca in omogoča razširljivost strežnika, ker se obremenitev zmanjša. V aplikacijah REST 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 vmesnik API (ang. application programming interface) 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 opcisko načelo. Strežnik na zahtevo odjemalca pošlje oziroma izvede programsko kodo na strani odjemalca. To smo vpeljali s pomočjo spletnih vtičnikov (ang. websocket), ki so bolj podrobno opisani spodaj.

6.) Enotni vmesnik (ang. Uniform interface) 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 poenostavila izdelavo strežnika, saj gre ze eno izmed najbolj popularnih spletno aplikacijskih vmesnikov (ang. 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 narejenih v spletnem ogrodju 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 spremenljivka *query* predstavlja poizvedbeni stavek v podatkovni bazi. Slika 2.5 prikazuje uporabo te funkcije in relativne poti *drinks*. Strežnik vsebuje 34 relativni 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

potrebnih za zapis v podatkovno bazo, na strežnik. Spremljanje zahtevkov na strani strežnika je mogoče v konzolnem vmesniku CLI, ki se uporablja pri zagonu strežnika, slika 2.7.

```

1  [
2   {
3     "product_id": 1,
4     "name": "Somersby Apple",
5     "price": 3.0,
6     "size": "0.33L",
7     "calorie": "207",
8     "picture": "somersby-apple.png",
9     "description": null,
10    "type_id": 1
11  },
12  {
13    "product_id": 2,
14    "name": "Heineken",
15    "price": 2.0,
16    "size": "0.25L",
17    "calorie": "110",
18    "picture": "heineken.png",
19    "description": null,
20    "type_id": 2
21  },
22  {

```

Slika 2.6: Primer serviranja podatkov na strežniku s programom Postman

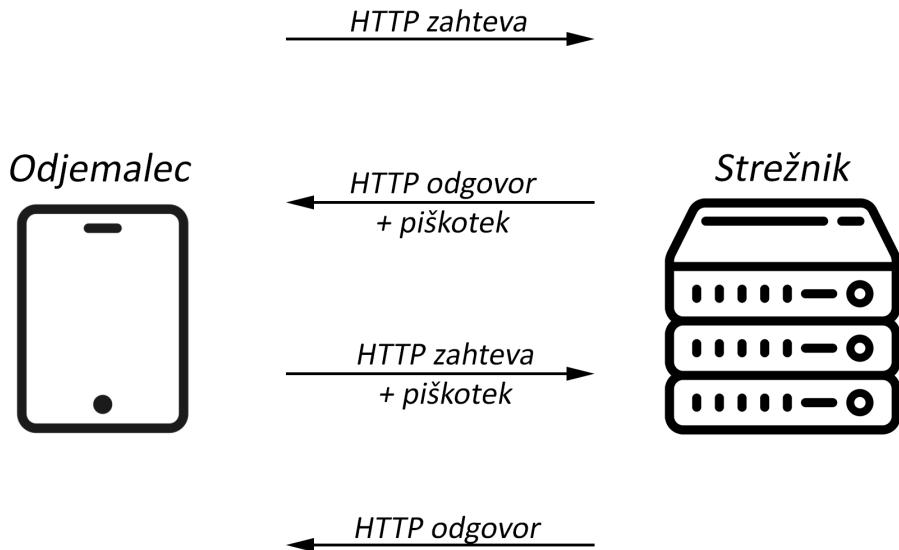
Za potrebe pridobivanja podatkov v realnem času na strani odjemalca, smo potrebovali še spletni vtičnik SocketIO. Implementirali smo ga na strani strežnika in odjemalca. Zagotavlja dvosmerno komunikacijo oziroma komunikacijo na podlagi dogodkov. Deluje na vseh platformah, brskalnikih ali napravah. Uporabili smo ga zaradi medsebojnega obveščanja odjemalcov o

```
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 spremeljanja zahtevkov, ki prihajajo na strežnik

spremembah v podatkovni bazi. Uporabili smo funkcijo *emit*, ki omogoča dodajanje podatkov in izbiranje načina razpršenega oddajanja (ang. broadcast). To pomeni, da vsi prejemajo te informacije ob oddajanju na strani strežnika. Gre predvsem za splošne podatke, tako da ne more priti do zlorabe. Npr. ob spremembah 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, prilaganje in shranjevanje informacij o posamezni seji uporabnika. Vsi piškotki so shranjeni na strani odjemalca in so kriptografsko zaščiteni pred morebitnimi neopombaščenimi posegi. S tem odjemalcu preprečujemo spremenjanje podatkov v piškotkih oz. lahko nedovoljene spremembe podatkov na strežniku zaznamo. 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 nadalnjem zahtevku doda piškotek s katerim strežnik overi zahtevo odjemalca.



Slika 2.8: Delovanje HTTP piškotkov med odjemalcem in strežnikom

## 2.3 Odjemalec

Odjemaleca 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 reaktivni 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 spremembah vrednosti posodobijo. Vue je namenjen izdelavi projektov SPA (ang. single-page application), saj vsebuje samo eno datoteko HTML. To prednost smo izkoristili s pomočjo ostalih knjižnic, ki so nam olajšale izdelavo aplikacije. Uporabili smo nasle-

dnje:

**Vue CLI** velja kot standardno orodje za ekosistem Vue [4]. Zagotavlja, da že pri gradnji novega projekta poveže različne dodatke med seboj. To omogoča razvijalcu, 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 nastavitev, 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.

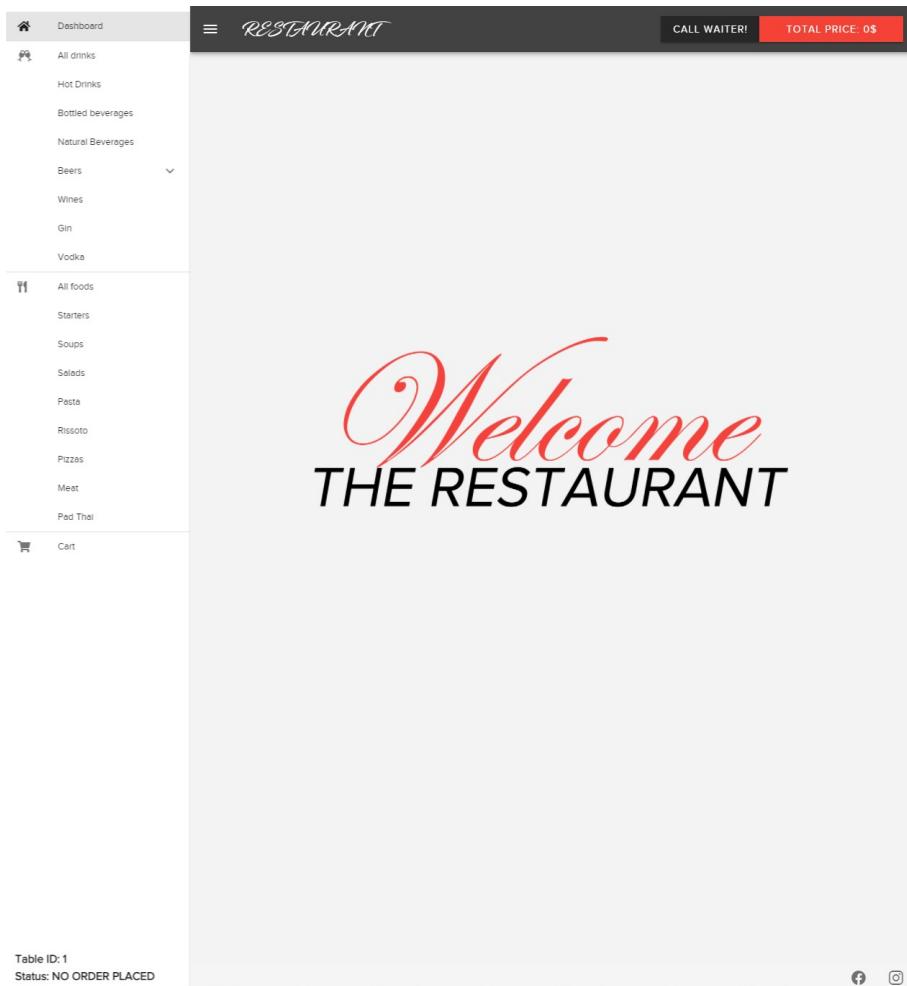
**ESLint** je orodje za prepoznavanje in poročanje o popravkih v programske kodi [1]. Cilj je narediti kodo bolj pregledno in urejeno, kar priomore k izogibanju napak.

**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.

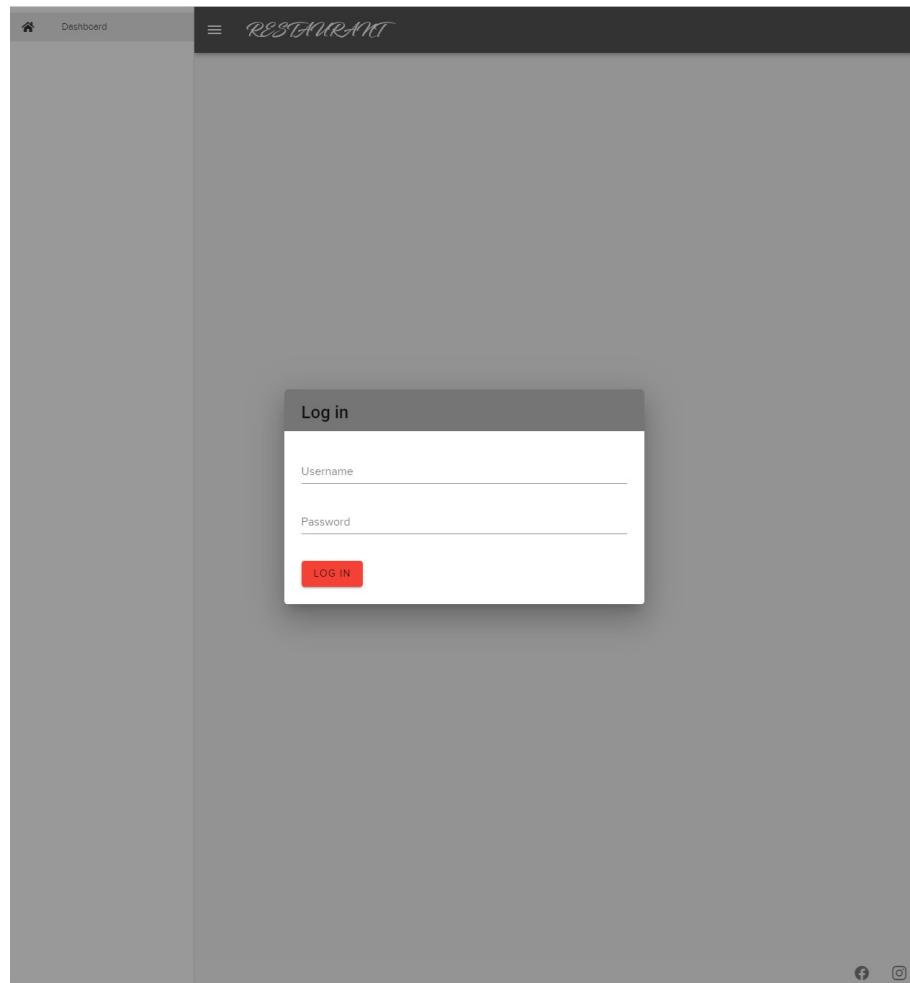
Programsko izvedbo na strani odjemalca smo razdelili v tri vloge oziroma dve aplikaciji. Ena aplikacija je namenjena natakarjem in kuharjem, ki se ločuje s prijavnim oknom in izgledom vmesnika. Druga aplikacija je namenjena samo gostom ter je sestavljena iz več pogledov. Ločili smo jih zaradi varnosti, 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 prikazujeti 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, obvečanju gosta o stanju naročila, itd. Najprej smo hoteli uporabiti samodejno osveževanje na določen časovni interval, vednar je uporaba



Slika 2.10: Spetni vmesnik za natakarja in kuharja

spletnih vtičnikov hitrejša in učinkovitejša. Slika 2.11 prikazuje seznam vseh 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 knižnjice pridobimo na stran odjemalca (slika 2.12).

```

    },
    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

```

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

# Poglavlje 3

## Delovanje aplikacije

Za prikaz delovanja smo pripravili testno okolje, kjer smo znotraj lokalnega omrežja postavili podatkovno bazo, strežnik in aplikaciji. Primer uporabe je opisan in prikazan spodaj, bolj podroben opis vseh funkcionalnosti pa v naslednjih poglavijih.

Gost s sprehajanjem skozi menije izbira hrano in pijačim ter jo sproti dodaja v naročilo (slika 3.1). V našem primeru gost izbere dve sokova in dve pici. Pregled naročila izvede v košarici, kjer naročilo tudi odda (slika 3.2). Natakar in kuhar se prvo prijavita, da imata omogočen pregled nad naročilo. Na seznamu se jima prikaže novo naročilo, katero morata preveriti ali imata vse sestavine za pripravo (slika 3.3). V našem primeru naročilo vsebuje hrano, tako da mora naročilo prvo potrdi kuhar (slika 3.4), da lahko natakar potrdi celotno naročilo gostu (slika 3.5). 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 3.6). Natakar je obveščen o zahtevi za račun in načinu plačila s strani gosta. S tiskanjem računa natakar zaključi naročilo, kar pomeni da se izbriše iz seznama aktivnih naročil.

**BOTTLED BEVERAGES**

- Fructal Strawberry: Description / Size: 0.33L, Price: 1.5\$, Count: 1x, Add to Cart
- Fructal Peach: Description / Size: 0.33L, Price: 1.5\$, Count: 0x, Add to Cart
- Coca Cola Zero: Description / Size: 0.25L, Price: 2.5\$, Count: 1x, Add to Cart
- Coca Cola: Description / Size: 0.25L, Price: 2.5\$, Count: 0x, Add to Cart

**PIZZAS**

- Margherita: Description: Tomato sauce, cheese, olive, Price: 8.5\$, Count: 0x, Add to Cart
- Navona: Description: Tomatoes, cheese, ham, mushrooms, artichokes, olive, Price: 9.5\$, Count: 1x, Remove
- Garibaldi: Description: Tomatoes, cheese, prosciutto, olive, Trieste sauce, Price: 10.5\$, Count: 1x, Remove
- Verdi: Description: Tomatoes, cheese, ham, Hungarian salami, olive, Price: 9.5\$, Count: 0x, Add to Cart

Table ID: 1  
Status: NO ORDER PLACED

Table ID: 1  
Status: NO ORDER PLACED

Slika 3.1: Pregled in izbiranje hrane in pijače

**CART**  
No order placed

**DRINKS**

- Fructal Strawberry: Description / Size: 0.33L, Price: 1.5\$, Count: 1x, Remove
- Coca Cola Zero: Description / Size: 0.25L, Price: 2.5\$, Count: 1x, Remove

**FOODS**

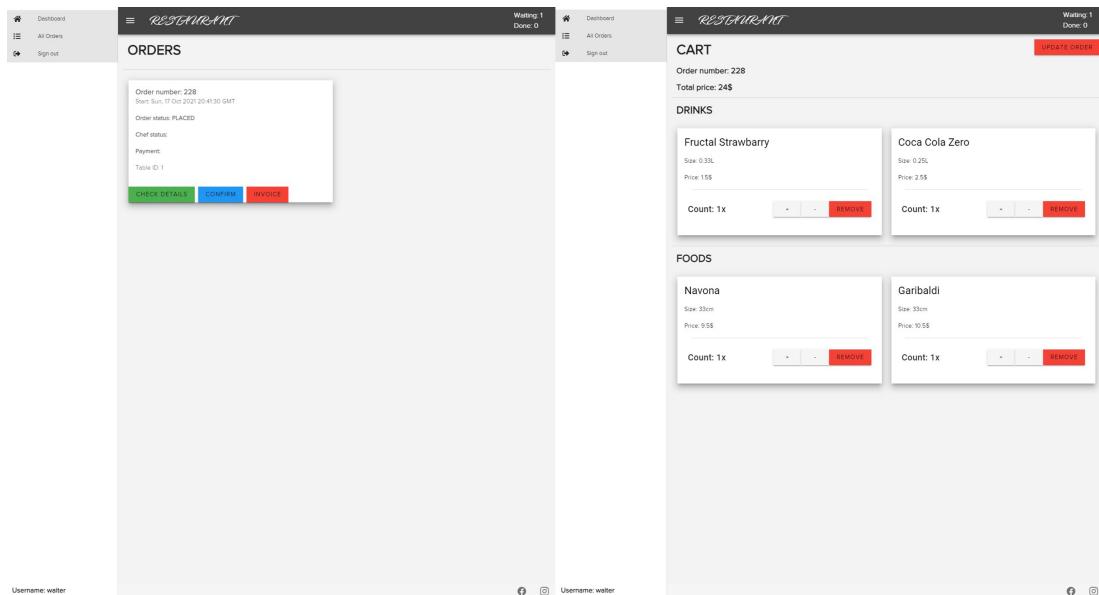
- Garibaldi: Description: Tomatoes, cheese, prosciutto, olive, Trieste sauce, Price: 10.5\$, Count: 1x, Remove
- Navona: Description: Tomatoes, cheese, ham, mushrooms, artichokes, olive, Price: 9.5\$, Count: 1x, Remove

**INFO**  
The order has been placed. Wait for the waiter and chef to confirm this.  
I UNDERSTAND

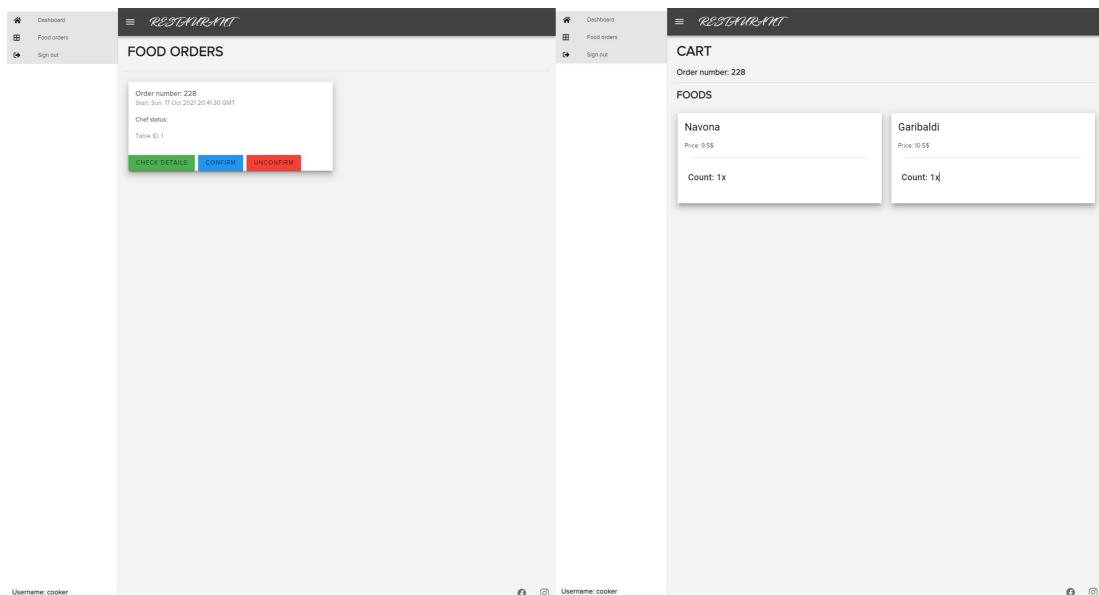
Table ID: 1  
Status: NO ORDER PLACED

Table ID: 1  
Status: PLACED

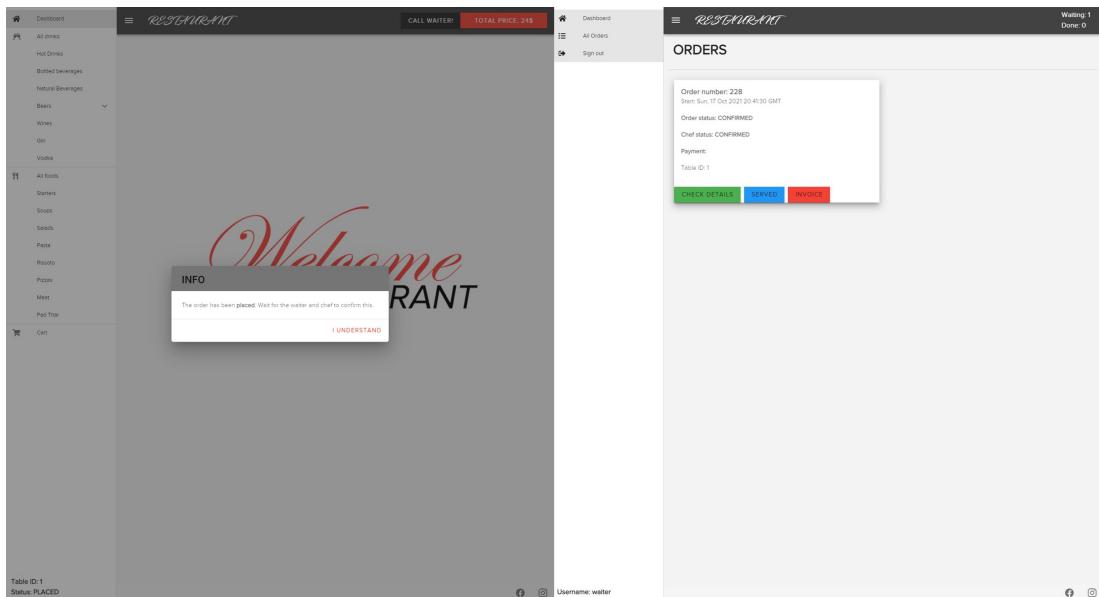
Slika 3.2: Pregled košarice in oddajanje naročila



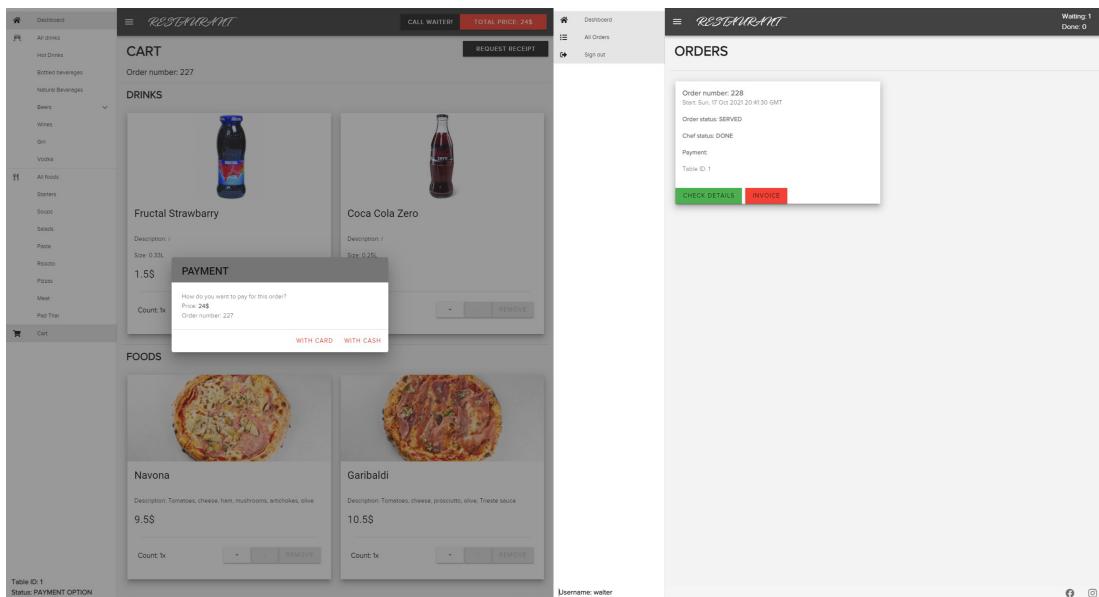
Slika 3.3: Pregled naročila na strani natakarja



Slika 3.4: Pregled naročila na strani kuharja



Slika 3.5: Obveščanje gostja o sprejemu naročila



Slika 3.6: Zahtevanje računa

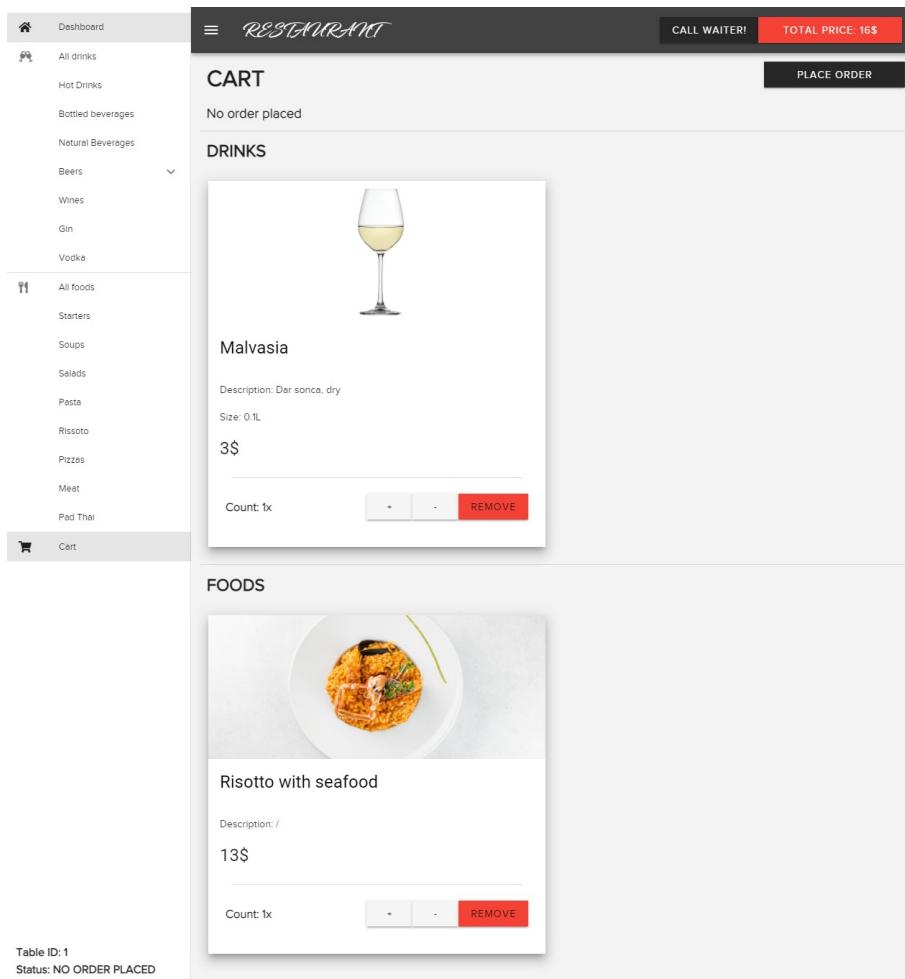
### 3.1 Vmesnik za gosta

Prvi pogled vmesnika za gosta vsebuje napis za dobrodošlico (slika 2.9) katerega bi lahko zamenjalo oglaševanja, predstavitev restavracije ali karkoli bi si potencialni kupec zaželet imeti. V zgornjem desnem kotu se nahaja gumb *call waiter* za klic natakarja in števec skupne cene artiklov v nakupovalni košarici. Gumb *call waiter* je namenjena gostom, ki aplikacije ne želijo uporabljati ali v primeru pomoči, če gost pride do kakršnihkoli težav. V spodnjem levem kotu se nahaja status in identifikacijska številka naročila.

Product	Description	Price	Count	Action
Spaghetti bolognese	Description: /	10\$	0x	<button>ADD TO CART</button>
Gnocchi with gorgonzola	Description: /	10.5\$	0x	<button>ADD TO CART</button>
Ravioli	Description: Ravioli with turkey, prawns, garlic, peperoncino	14\$	0x	<button>ADD TO CART</button>

Slika 3.7: Seznam artiklov znotraj vrste špagetov

Zavihki na levi strani predstavljajo seznam vseh vrst hrane in pičače, katere kažejo na podstrani ponube, ki jo nudi restavracija (slika 3.7). Gostu to predstavlja ponudbo s katero v nakupovalno košarico dodaja, briše hrano in pičačo ali spreminja njihovo količino.



Slika 3.8: Primer naročila

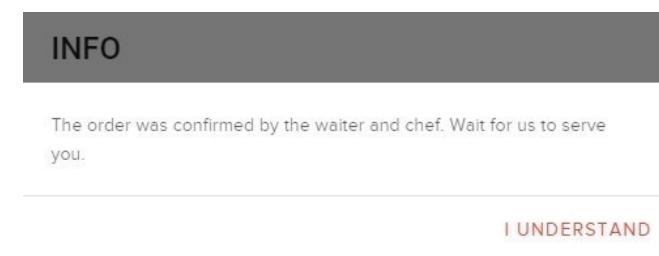
Vsaka hrana ali pičača vsebuje sliko, ime, opis in ceno. Poleg tega vsebuje še gumb *add to cart* (dodaj v košarico), ki ob kliku izgine in prikaže tri nove gumbe + (povečaj količno), - (zmanjšaj količino) in *remove* (odstrani). Nakupovalna košarica ozziroma *cart* je skupno mesto vse hrane in

pijače potencialne za naročilo (slika 3.8).

Naročilo se odda s klikom na gumb *place order*, ki gostu preusmeri na prvo stran in obvesti s pojavnim sporočilom prikazanim na sliki 3.9. 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 3.10. 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 na sliki 3.11. Naročilo se zaključi s klikom na gumb *request receipt*, ki odpre pojavno okno (slika 3.12) na katerem je potrebno izbrati način plačila.



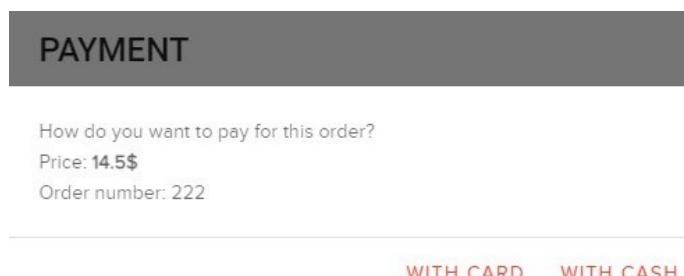
Slika 3.9: Pojavno sporočilo ob uspešni oddaji naročila



Slika 3.10: Pojavno sporočilo ob potrditvi naročila s strani natakarja



Slika 3.11: Pojavno sporočilo ob zavrnitvi naročila s strani natakarja



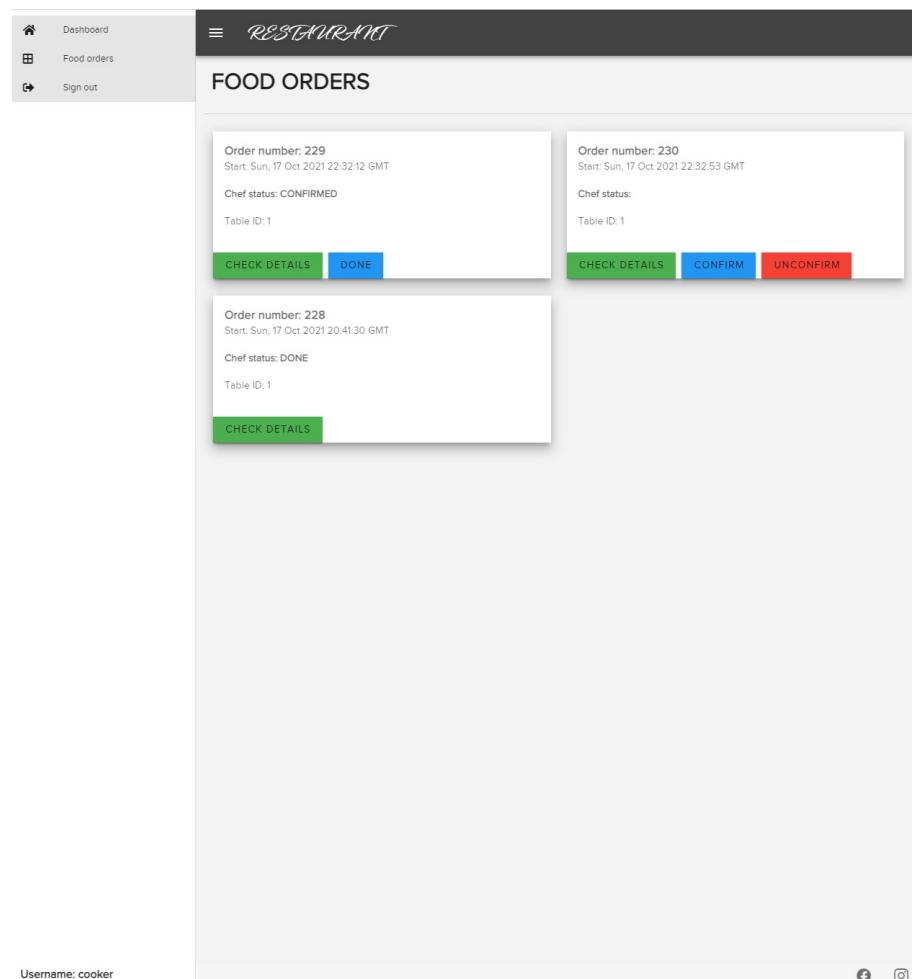
Slika 3.12: Pojavno okno z možnostjo izbire načina plačila

## 3.2 Vmesnik za natakarja in kuharja

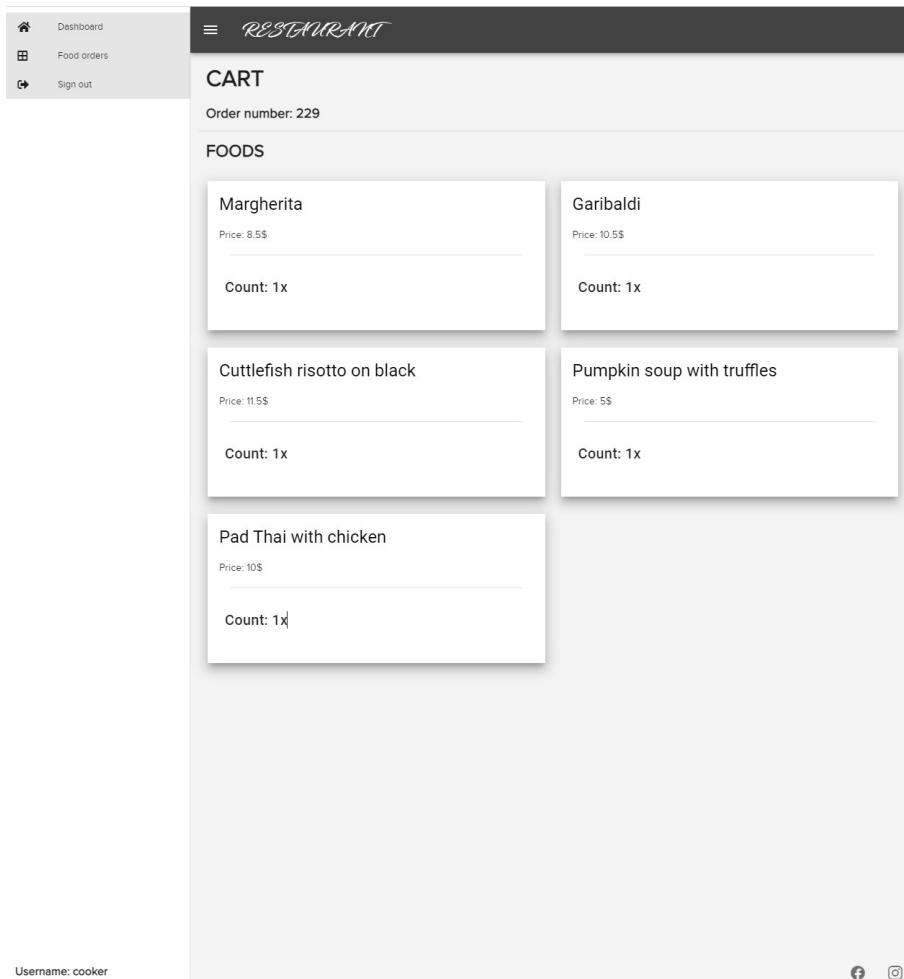
Prvi pogled vmesnika je enak tako za natakarja kot kuharja, saj gre za skupno aplikacijo kjer se pogledi razlikujejo glede vlogo uporabnika, ki je določena v podatkovni bazi. Nismo naredili ločene aplikacije, saj ni bilo potrebe, namreč oba uporabnika imata zelo podobne funkcije. Slika 2.10 prikazuje prijavo 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 *sing out*. Prva podstran imenovana *dashboard* ali prva stran ob uspešni prijavi, prikazuje napis za hitrejšo razlikovanje med vlogami. Natakar ima poleg tega v desnem zgornjem kotu še stevec čakajočih in zaključenih naročil.

Kuharju se v zavihku *food orders* pojavijo vsa nezaključena naročila, ki

vsebujejo hrano (slika 3.13). Vsako naročilo vsebuje naslednje podatke: identifikacijska številka naročila, čas oddaje naročila, status kuhanja in številka mize. Kuhar mora naročilo najprej pregledati z gumbom *check details* (slika 3.14) 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 natakarja s klikom na gumb *done*, ki izbriše naročilo iz seznama. Ob vsaki izvedeni akciji se pri vsakem naročilu spremi status kuhanja, ki je lahko: *updated*, *confirmed*, *unconfirmed* in *done*. Status *updated* je v primeru dodajanja artiklov k naročilu s strani gosta.



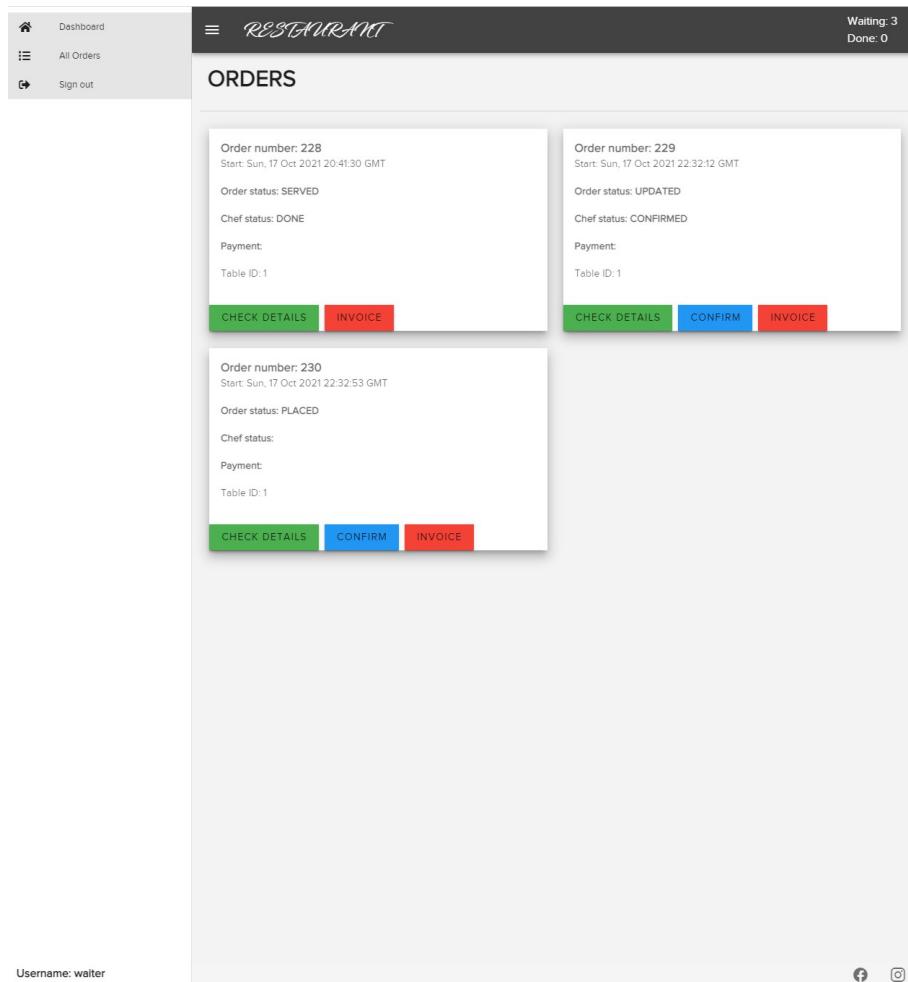
Slika 3.13: Zavihek *food orders*



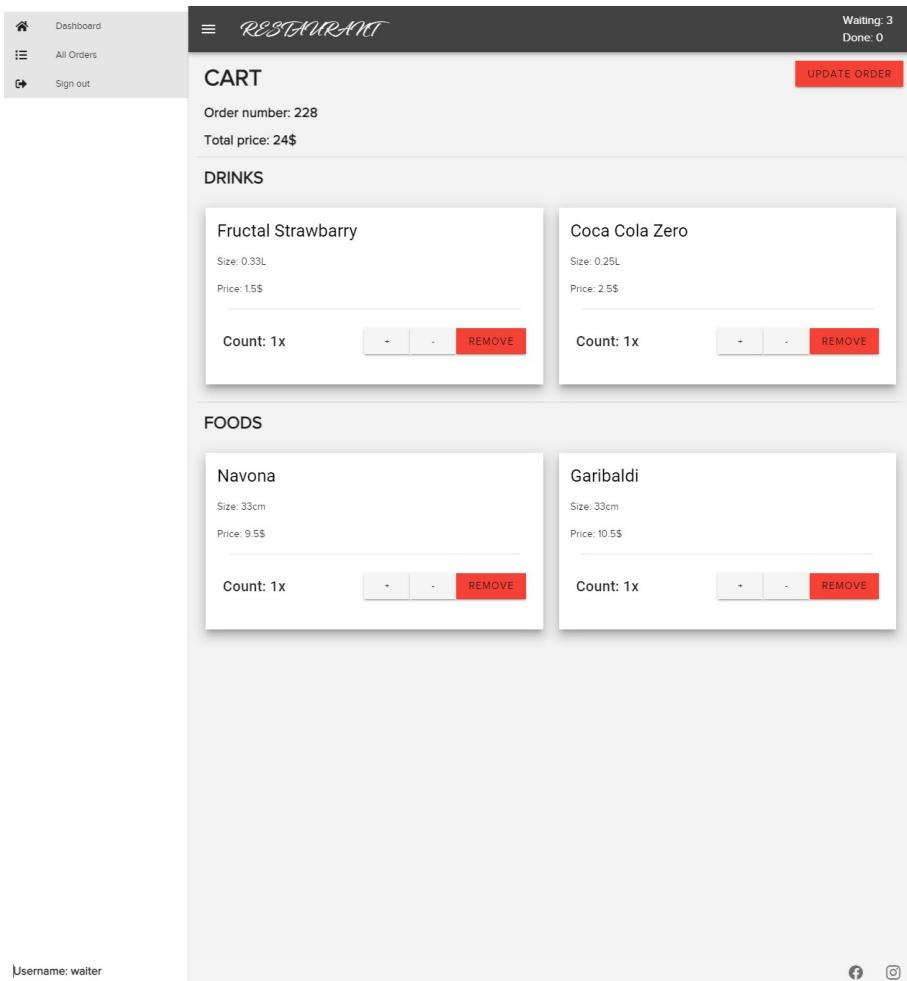
Slika 3.14: Zavihek, ki se odpre kuharju ob kliku na gumb *check details*

Natakarju se v zavihku *all orders* pojavijo vsa nezaključena naročila (slika 3.15). 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 z gumbom *check details* (slika 3.16). Natakar lahko ureja celotno naročilo, kar pomeni da lahko spreminja količino hrane in pijače ali jo briše iz naročila. Ko zaključi urjanje 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*. 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 3.15: Zavihek *all orders*



Slika 3.16: Zavihek, ki se odpre natakarju ob kliku na gumb *check details*

### 3.3 Implementacija programa v realnosti

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

Nadgrajena bi predstavljal uporabo aplikacije za naročanje z branjem kode QR (ang. quick response), 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 bi bil preusmerjen v aplikacijo. Celoten sistem

bi deloval v odprtem internetnem omrežju, 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 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 overili svojo dejansko prisotnost v restavraciji. Drugi način bi bilo geslo, ki bi ga uporabnik prejel od natakarja.

### **3.4 Izboljšave**

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

- 1.) Možnost dodajanja hrane in pijače k naročilu s strani vsakega posameznika za mizo.
- 2.) Možnost hkratnega delovanja več kuharjev, da bi lahko vsak vedel kaj more delati. Tako kot smo to implementirli za natakarje.
- 3.) Dodatno spremjanje hrane in pijače na strani kuharja/natakarja, kot npr. katera pijača in hrana je že bila postrežena.
- 4.) Pisanje opomb pri vsakem naročilu, kar bi natakarju in kuharju oljašalo delo ob povečanemu številu gostov.

Napredne izboljšave:

- 1.) Statistika za lastnika restavracije, ki bi poleg vseh podatkov računala oceno nabave za prihodnji mesec.
- 2.) Brezstično plačevanje s plačilno kartico neposredno na strani gosta.
- 3.) Če bi aplikacija delovala v odprtem internetnem omrežju, bi s pomočjo skupne spletne strani omogočali dostavo hrane za vsako restavracijo.



# Poglavlje 4

## Sklepne ugotovitve

Namen naše diplomske naloge je bil ustvariti enostavno aplikacijo, ki bi olajšala delo natakarjem in nudila bolj kakovostno postrežbo gostom. Uporaba aplikacije gostom ponuja drugačno uporabniško izkušnjo, ki vseeno omogoča klasično naročanje, in sicer z neposrednim stikom z natakarjem. S tem smo želeli poudariti, da aplikacija ni namenjena nadomestitvi delovnih mest natakarjev, vendar kot pripomoček pri izvajanju njihovega dela. Natakarjeva glavna naloga bi še vedno bila postrežba gosta ter priprava pihače.

Aplikacija je dovršena za uporabo, vendar smo med razvojem dobili veliko idej za izboljšavo. Uporabljali bi jo lahko v vsaki restavraciji, če ne drugače kot pomoč ob veliki zasedenosti. Trenutno, v času COVID-19 krize, bi lahko ta aplikacija omogočala nemoteno delovanje restavracij in barov.

Mislimo da je aplikacija v okviru diplomskega dela realizirana v skladu z zahtevami, pričakovanji in cilji.



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