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

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 prepiše 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

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

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 LaTeX. A proper abstract should contain around 100 words which makes this one way too short.

Keywords: computer, computer, computer.

Poglavje 1

Uvod

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

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

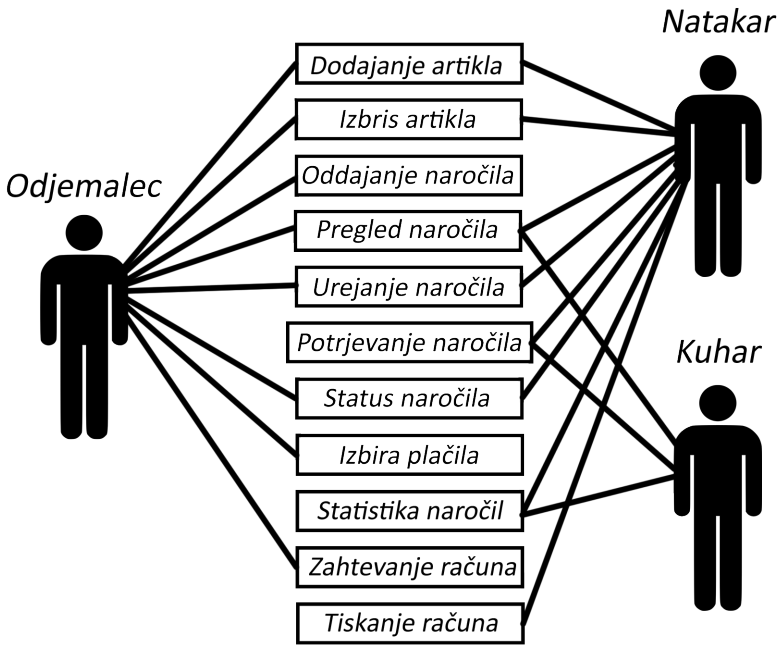
Aplikacija podpira tri uporabniške vloge, in sicer gost, natakar in kuhar.

Poglavje 2

Razvoj in struktura aplikacije

Razvoj aplikacije je potekal v treh delih in sicer analiza problematike oziroma izdelava diagrama primerov uporabe, zamisel strukture in implementacija.

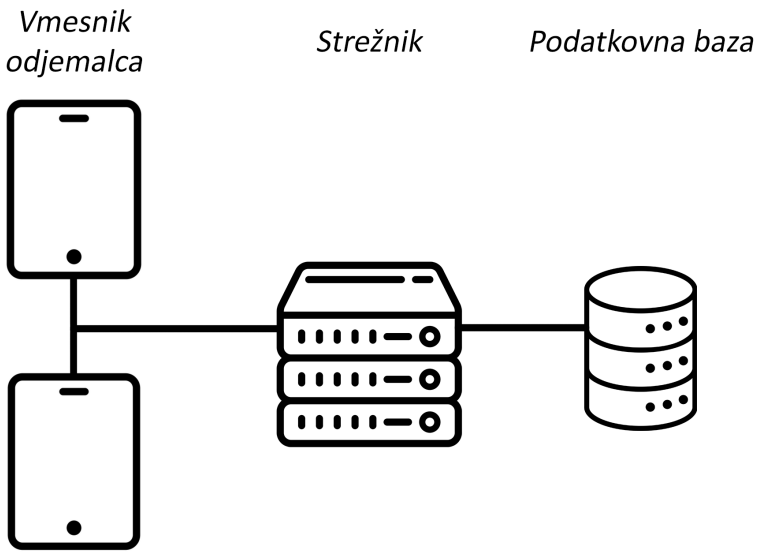
Iz analize zahtev smo za lažji pregled nad vsemi funkcionalnostmi izdelali diagram primerov uporabe, ki ga prikazuje slika 2.1.



Slika 2.1: Diagram primerov uporabe spretnega naročanja

Aplikacija oziroma odjemalec ima tri uporabniške poglede, ki so gost, natakar in kuhar. Restavracija ima lahko več miz, vendar za eno mizo je lahko hkrati odprto eno naročilo. To pomeni, da morajo za mizo naročati skupaj. Gost lahko odda, spremeni ali zaključi naročilo. Natakar lahko sprejme, zavrne, uredi ali zaključi naročilo. V restavraciji je lahko več natakarjev, ki streže istočasno. Kuhar lahko sprejeme, zavrne ali sporoči, da jed za neko naročilo že pripravljena. Restavracija ima lahko več kuharjev, vendar samo en kuhar hkrati lahko uporablja aplikacijo. Omejitev je v podatkovnem modelu.

Uporabili smo strukturo novodobnih aplikacij katero sestavljajo podatkovna baza, strežnik in odjemalca. Gre za koncept, ki ga je moč prilagajati predvsem z uporabniškega vidika. Slika 2.2 prikazuje strukturo aplikacije.

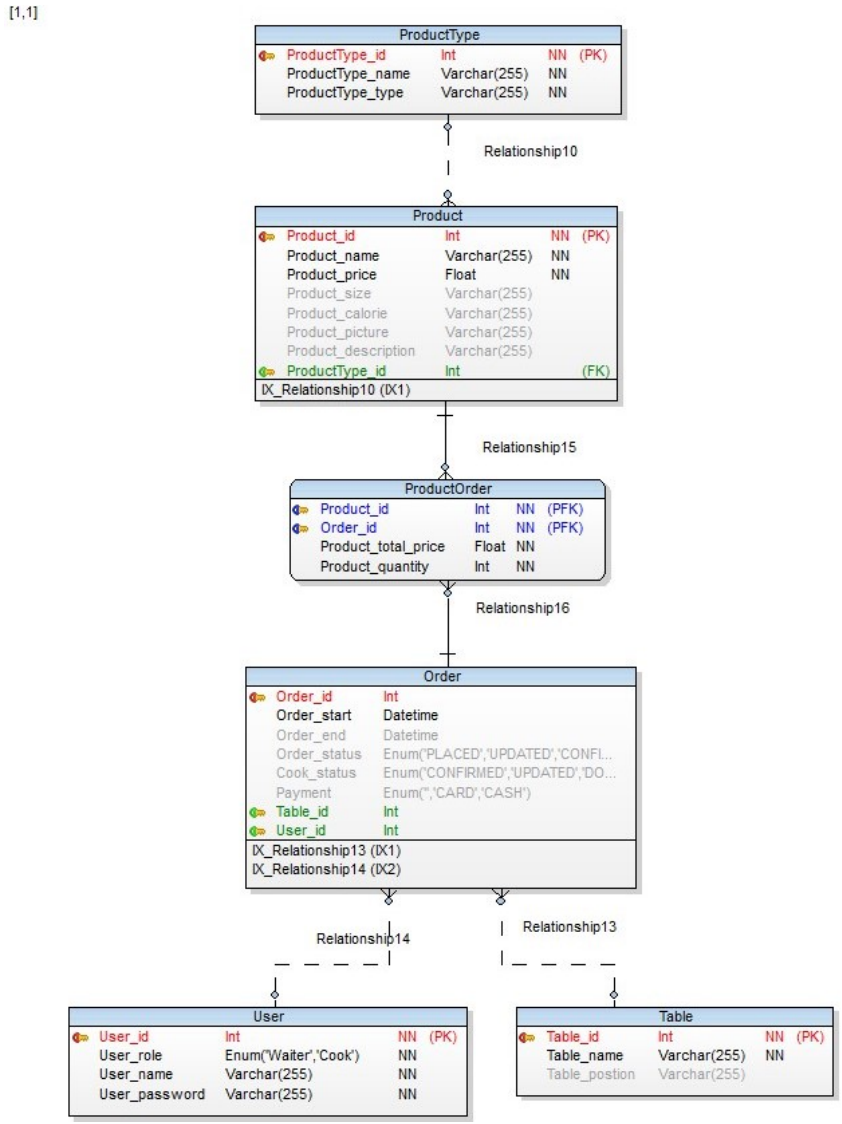


Slika 2.2: Visokonivojska arhitektura

Podatkovna baza je namenjena shranjevanju vseh podatkov, da se ne izgubijo ter se razlikujejo za vsako restavracijo. Strežnik implementira RESTful vmesnik, ki odjemalcu servira podatke iz podatkovne baze. Odjemalec sta v našem primeru dve aplikaciji implementirani s pomočjo spletnih tehnologij.

2.1 Podatkovna baza

Na podlagi izdelanega diagrama zahtev oziroma fukcij smo izdelali logičen podatkovni model (slika 2.3), katerega smo kasneje realizirali s pomočjo SUPB MySQL. Podatkovna baza je sestavljena je iz šestih tabel, ki so: *ProductType, Product, ProductOrder, Order, User, Table.*



Slika 2.3: Logični podatkovni model

ProductType tabela je namenjana zapisom vrsti jedi t.i. predjedi, glavne jedi, sokovi, piva,... 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 pijače.

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

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

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

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

User tabela je namenjena predvsem aplikativnem delu za natakarje in kuharje. Sestavljna je iz atributov: ID, Role, Name in Password.

Strukturo podatkovne baze smo naredili s pomočjo programa Toad Data-Modler. 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, Mircrosoft SQL Server,...

Podatkovni model smo implementirali na fizičnem nivoju v SUPB MySQL. To je eden od odprtokodnih sistemov za upravljanje s podatkovni bazami, ki za delo s podatki uporablja jezik SQL [6]. Napisan je v programskem jeziku C in C++ in deluje v vseh modernih sistemih npr. Windows, Linux, OS X,... Prva verzija je bil razvita leta 1995 s strani Michael Widenius in David Axmark. Kratica My izhaja iz imena prve hčerke očeta Michaela.

2.2 Strežnik

Strežnik v naši aplikaciji predstavlja vmesnik med podatkovno bazo in odjemalcem. Ko smo izdelali podatkovno bazo smo začeli z izdelavo strežniškega dela. Najbolj pomembno nam je bilo, da je sistem zanesljiv, saj brez njega odjemalec ne more delovati. Izbrali smo REST arhitekturo zaradi načel, ki so opisana spodaj [7]. Sama arhitektura omogoča, da odjemalec s pomočjo zahtev pridobiva podatke od strežnika, kateri jih s pomočjo URI povezav oglašuje na relativnih povezavah. Strežnik smo napisali v programskem jeziku Python ter s pomočjo knjižnic Flask, MySQL, SocketIO in CORS, kateri nameni so bolj podrobno opisani spodaj. Načela arhitekure 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 naši aplikaciji imamo kar dva odjemalca (gost in natakar/kuhar), kar pa za strežnik ne predstavlja nobenih omejitev razen na strani podatkovne baze, ki omejuje število hkratnih poizvedb. V naši aplikaciji te omejitve nismo presegli.

2.) *Brez stanja (ang. Stateless)* morajo biti vse interakcije med strežnikom in odjemalcem. Strežnik ne sme shranjevati nobenih stanj oziroma more vsako zahtevo od odjemalca tretirati kot popolnoma novo. V programski kodi našega strežnika je dobro razvidno, da ne uporabljamo nobenih global-

nih spremenljivk. Vse kar odjemalec zahteva se prebere iz podatkovne baze in vrne odjemalcu.

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

4.) *Večslojni sistem (ang. Layered system)* je sestavljen iz hierarhičnih slojev kjer npr. za API vmesnik uporabimo strežnik A, za shranjevanje podatkov strežnik B ter strežnik C za 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 opcijsko načelo. Strežnik na zahtevo odjemalca pošlje oziroma izvede programsko kodo na strani odjemalca. To smo vpeljali s pomočjo spletnih vtičnikov (websocket), kjer so bolj podrobno opisani spodaj.

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

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

Vsaka relativna povezava URI na strežniku predstavlja svoje vir podatkov iz podatkovne baze. Te podatki so odjemalcu na voljo v JSON podatkovnem formatu. Strežnik s pomočjo knjižnice MySQL najprej prebere podatke iz podatkovne baze in jih predstavi na določeni relativni povezavi URI, ki

jo določimo mi. Na sliki 2.4 je prikazana funkcija za branje podatkov iz podatkovne baze, kjer spemenljivka query predstavlja poizvedbeni stavek v podatkovni bazi. Slika 2.5 prikazuje uporabo te funkcije in relativne poti drinks. Naš strežnik vsebuje 34 relativni povezav in 600 vrstic programske kode.

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

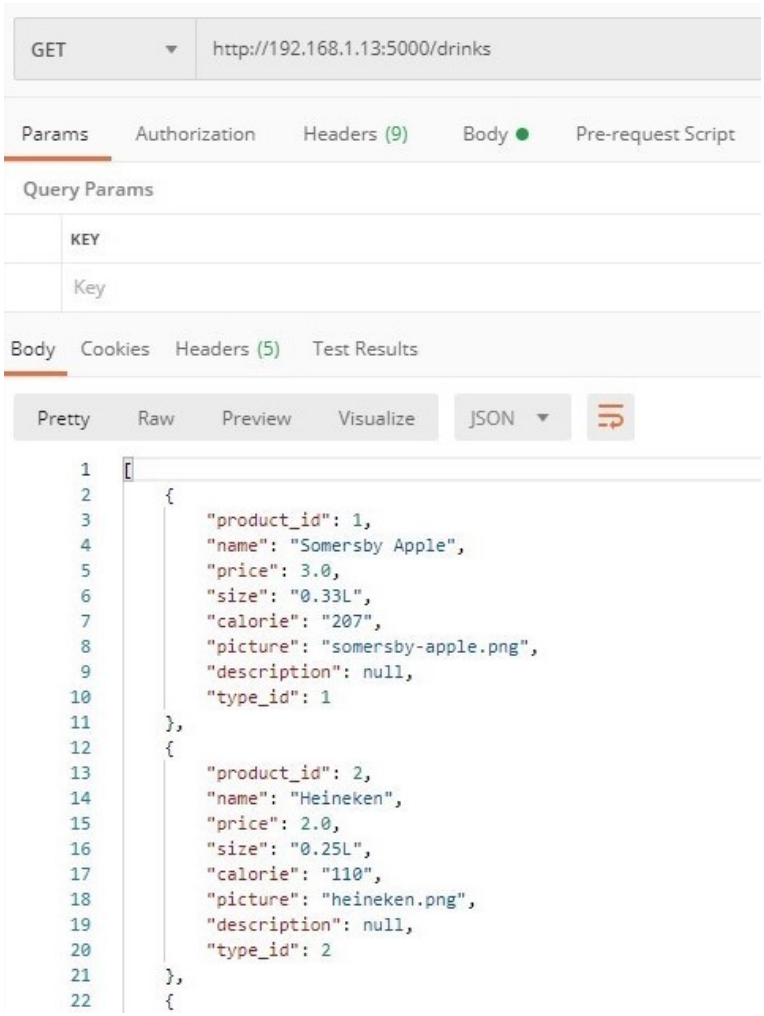
Slika 2.4: Funkcija, ki prebere podatke iz podatkovne baze

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

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

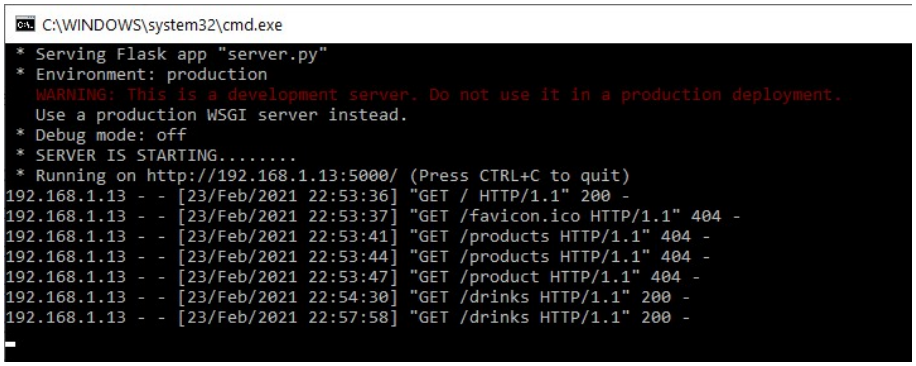
Tako smo dobili vmesnik, ki na zahtevo odjemalca odgovori s podatki v JSON formatu. Primer na sliki 2.6, ko odjemalec zahteva podatke vseh pijač iz podatkovne baze (HTTP metoda GET). Strežnik omogoča tudi sprejemanje podatkov, vendar more biti zato ustrezna metoda HTTP. Metoda POST

metodo smo uporabljali predvsem pri posredovanju podatkov strežniku, kateri so bili potrebni za zapis v podatkovno bazo. Spremljanje zahtevkov, ki prihajajo na strežnik, je mogoče preko vmesnika CLI, slika 2.7, ki v primeru nepopolnosti prikaže ustrezno napako.



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



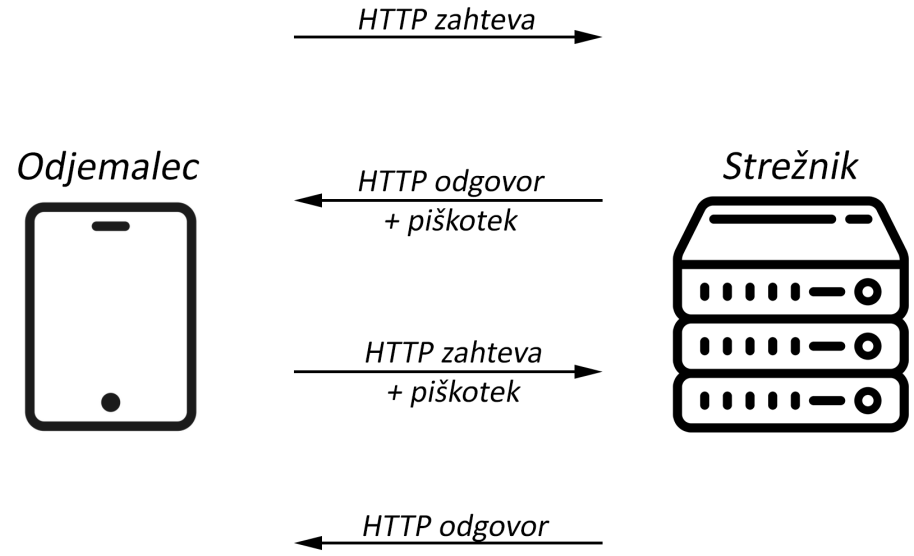
Slika 2.7: Primer spremljanja zahtevkov, ki prihajajo na strežnik

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

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

Odjemlaca 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 jezik JavaScript. Naredili smo odziven in reaktiven vmesnik, ki deluje v realnem času. Vue je eden izmed mnogih kot npr. Angular, Ember, React,... poznan pa je predvsem zaradi enostavnosti za upravljanje in izvajanje testov. Vsem skupna reaktivnost, vendar v drugačnem pomenu besede. Reaktivnost [2], je programska paradigma, ki nam omogoča, da se na deklarativni način prilagodimo spremembam. Tako deluje tudi reaktivnost v aplikacijah za razliko, da je podatek lahko vezan na več funkciji oziroma delov programske kode, ki se ob 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 razviljacu, da se bolj osredotoči na programiranje in ne na povezovanje njih v projekt. Zadeva izgleda nekako tako, da preko CLI vmesnika izbereš kakšen projekt želiš. Imaš seveda že privzete nastavitve, vendar omogoča tudi nastavljanje po meri. Sam sem uporabil Vuex, Vue-Router, ESLint in Vuetify.

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

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

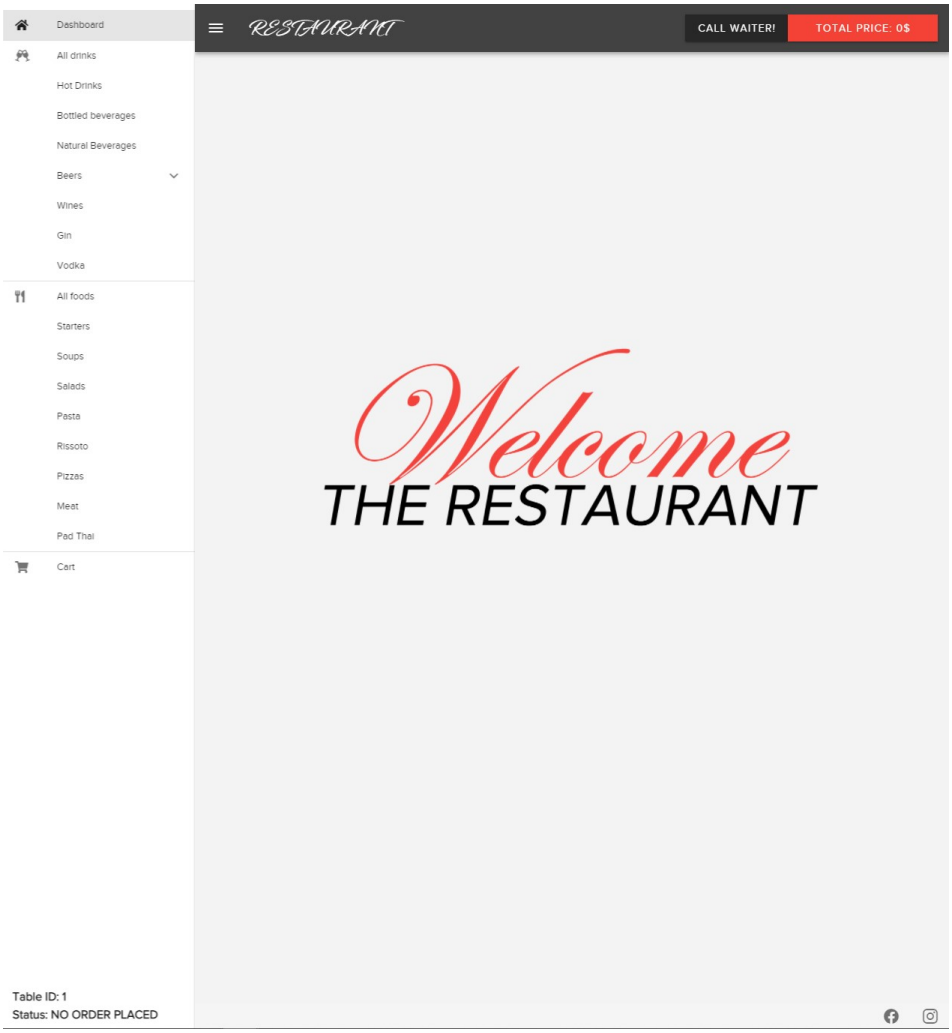
ESLint je orodje za prepoznavanje in poročanje o popravkih v programski kodi [1]. Cilj je narediti kodo bolj pregledno in urejeno, kar pripomore k izogibanju napak.

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

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

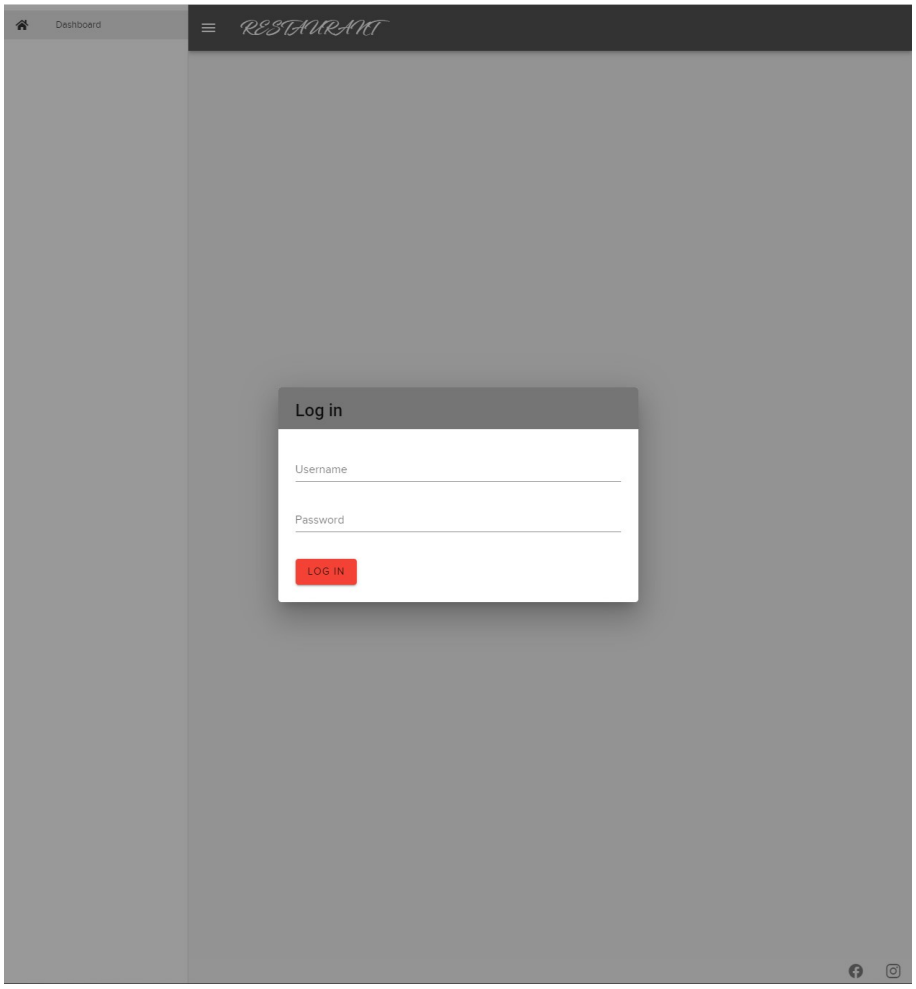
Odjemalca smo razdelili v tri vloge oziroma dve aplikaciji. Ena aplikacija je namenjena natakarjem in kuharjem, ki se ločuje s prijavnim oknom

in izgledom vmesnika. Druga aplikacija je namenjena samo gostom ter je sestavljena iz večih pogledov. Ločili smo jih zaradi varnosti, lažjega razvijanja in preglednosti, saj gre za dve popolnoma različni aplikaciji. Vse funkcionalnosti in delovanje ene in druge aplikacije so opisane v naslednjem poglavju. Sliki 2.9 in 2.10 prikazujeti prvi izgled obeh aplikacij.



Slika 2.9: Spletni vmesnik za gosta

Ena izmed pomembnih stvari pri obratovanje restavracije je čim hitrejša postrežba katero je mogoče izboljšati s čim hitrejšo komunikacijo. Zato



Slika 2.10: Spetni vmesnik za natakarja in kuharja

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,... Najprej smo hoteli uporabiti samodejno osveževanje na določen časovni interval, vednar je uporaba spletnih vtičnikov precej bolj zanesljiva in točna. 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

```
  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: Spletni vtičniki v aplikaciji za gosta

oglašujejo na strani strežnika s pomočjo te knižnjice preberemo na strani odjemalca.

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

Slika 2.12: Način uporabe Axios v aplikaciji za gosta

Poglavje 3

Delovanje aplikacije

3.1 Vmesnik za gosta

3.2 Vmesnik za natakarja

3.3 Vmesnik za kuharja

Poglavje 4

Diskusija

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://en.wikipedia.org/wiki/Toad_Data_Modeler. [Dostopano: 23. 02. 2021].

[4] Vue cli overview. Dosegljivo: https://cli.vuejs.org/guide/. [Dostopano: 01. 11. 2020].

[5] Vue router. Dosegljivo: https://router.vuejs.org/. [Dostopano: 01. 11. 2020].

[6] What is mysql? Dosegljivo: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html. [Dostopano: 23. 02. 2021].

[7] What is rest. Dosegljivo: https://restfulapi.net/. [Dostopano: 01. 05. 2021].

[8] What is vuex? Dosegljivo: https://vuex.vuejs.org/. [Dostopano: 01. 11. 2020].

[9] Why is flask good? Dosegljivo: https://www.fullstackpython.com/flask.html. [Dostopano: 01. 11. 2020].

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