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

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.

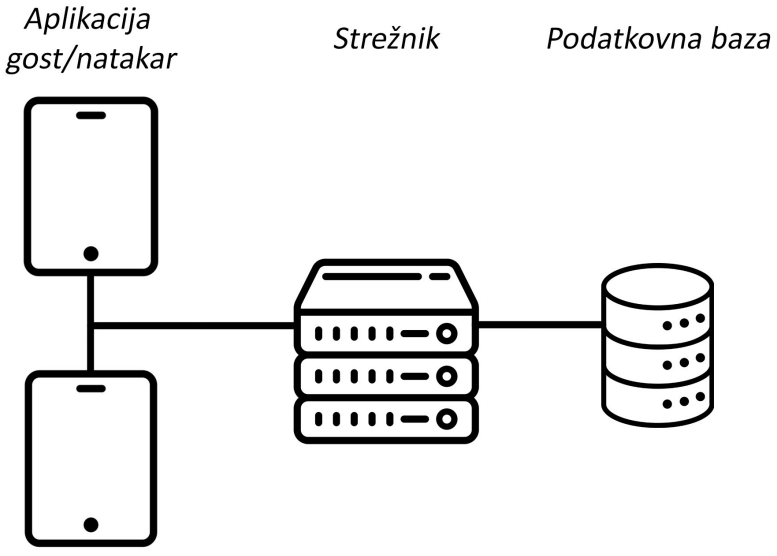
Aplikacija bi bila sestavljena iz treh osnovnih pogledov, ki bi bili namenjeni gostu, natakarju in kuharju.

Poglavje 2

Razvoj in struktura aplikacije

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

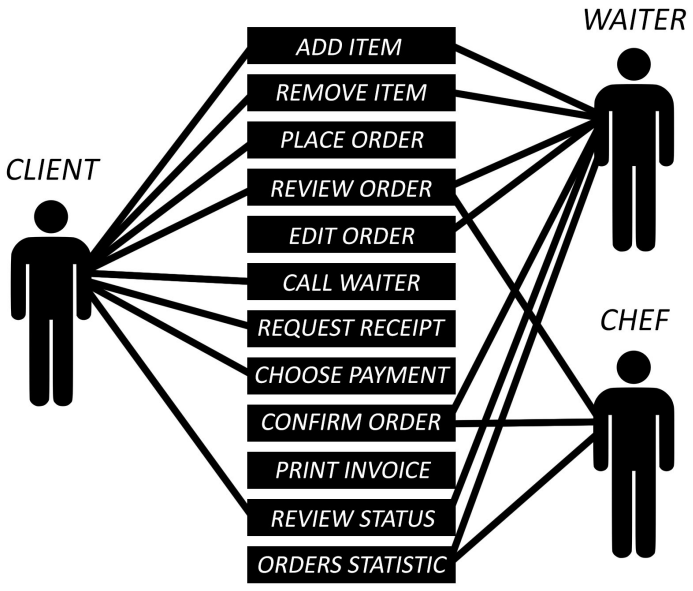
Uporabili smo strukturo novodbih aplikacij katero sestavljajo podatkovna baza, strežnik in odjemalca. Gre za koncept, ki ga je moč prilagajati predvsem z uporabniškega vidika. Slika 2.1 prikazuje strukturo aplikacije.



Slika 2.1: Struktura aplikacije

Podatkovna baza je namenjena shranjevanju vseh podatkov, ki so pomembni, da se ne izgubijo oziroma se razlikujejo za vsako restavracijo. Strežnik je kot neke vrste RESTful vmesnik, ki servira podatke iz podatkovne baze odjemalcu. Odjemaleca sta v našem primeru dve ločeni spletni aplikaciji, in sicer za gosta ter natakarja oziroma kuharja, ki je skupen.

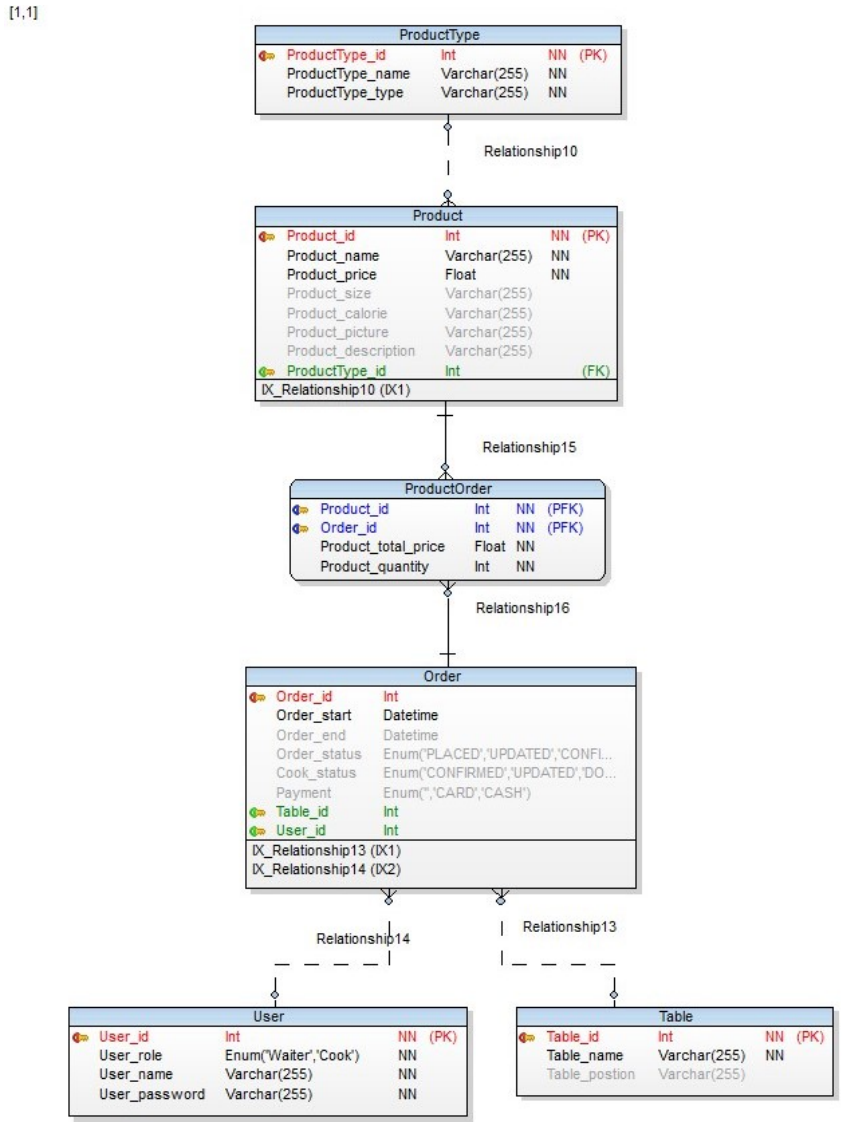
Gost lahko odda, spremeni ali zaključi naročilo. Za eno mizo je lahko hkrati odprto eno naročilo. To pomeni, da morajo za mizo naročati skupaj. V restavraciji je lahko večje število miz. Natakar lahko sprejme, zavrne, uredi ali zaključi naročilo. V restavraciji je lahko večje število natakarjev, ki streže 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.



Slika 2.2: Diagram primerov uporabe spretnega naročanja

2.1 Podatkovna baza

Na podlagi izdelanega diagrama zahtev oziroma fukcij smo izdelali fizično obliko podatkovne baze (slika 2.5), katero smo kasneje pretvorili v MySQL. Podatkovna baza je sestavljena je iz šestih tabel, ki so: *ProductType, Product, ProductOrder, Order, User, Table.*



Slika 2.3: Fizična oblika podatkovne baze

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. To nam je pomagalo pri razločevanju 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 datoteki na lokalnem strežniku Apache.

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 je zelo pomemben saj določuje 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 DataModler. To je orodje za izdelavo visokokakovostnih podatkovnih modelov [4]. Omogoča izdelavo logičnih in fizičnih podatkovnih modelov, kar pripore 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,...

Za izvoz podatkovne baze smo uporabili MySQL. To je eden od odprtokodnih sistemov za upravljanje s podatkovni bazami, ki za delo s podatki uporablja jezik SQL [7]. Napisan je v programskem jeziku C in C++ in deluje v vseh modernih 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.

Najbolj zahtevno je bilo napolniti podatkovno bazo, katero delo smo si olajšali s pomočjo phpMyAdmin. To je ime za spletni vmesnik, ki omogoča upravljane z MySQL podatkovnimi bazami. Program je vključen v XAMPP in sicer znotraj aplikacije MySQL. Je zelo močno orodje in predvsem enostavno za uporabo.

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 na je bilo, da je sistem zanesljiv, saj brez njega odjemalec ne more delovati. Zato smo izbrali REST arhitekturo, ki zahteva določena načela, katera so bolj podrobno opisana spodaj. 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.

1.) *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 (naročnik, kar pa za strežnik ne predstavlja nobenih omejitev razen na strani podatkovne baze, ki omogoča določeno število hkratnih poizvedb. V naši aplikaciji te omejitve nismo nikoli presegli.

2.)*Stateless* - vse interakcije med strežnikom in odjemalcem morajo biti brez stanja (stateless). Strežnik ne sme shranjevati nobenih status oziroma more vsako zahtevo od odjemalca tretirati kot popolnoma novo. V programski kodi našega strežnika je dobro razvidno, da ne uporabljamo nobenih globalnih spremenljivk. Vse kar odjemalec zahteva se prebere iz podatkovne baze in vrne odjemalcu.

3.)*Cachable* - predpomnjenje 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.)*Layered system* - slojevit sistem, sestavljen iz hierarhičnih slovej z namenom omejevanja komponent.

5.)*Code on demand (optional)* - opcijsko načelo. Strežnik na zahtevo odjemalca pošlje oziroma izvede programsko kodo na strani odjemalca.

6.)*Uniform interface* - zahteva enoten vmesnik (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 je potrebno uporabiti točno določena podatkovna formata XML in JSON.

Za to zahtevo nam je knjižnica Flask zelo poenostavila izdelavo strežnika. Vsak relativna povezav URI na strežniku predstavlja svoje vir podatkov iz podatkovne baze. Te podatki so odjemalcu na voljo v JSON podatkovnem formatu. Strežnik najprej podatke prebere iz podatkovne baze s pomočjo MySQL knjižnice in jih predstavi na določeni relativni povezavi URI, ki jo določimo mi. Na sliki LALAL je prikazana funkcija za branje podatkov iz podatkovne baze. Spremenljivka query predstavlja poizvedbeni stavek v podatkovni bazi. in izpisovanje na povezavo z imenom drinks.

Vsa zgoraj našteta navodila smo morali upoštevati pri izdelavi strežnika. Tako smo dobili vmesnik, ki na zahtevo odjemalca servira podatke v JSON formatu. Primer na sliki 2.6, ko odjemalec zahteva podatke vseh pijač iz podatkovne baze (HTTP metoda GET). Strežnik omogoča tudi urejanje in

```python
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
```

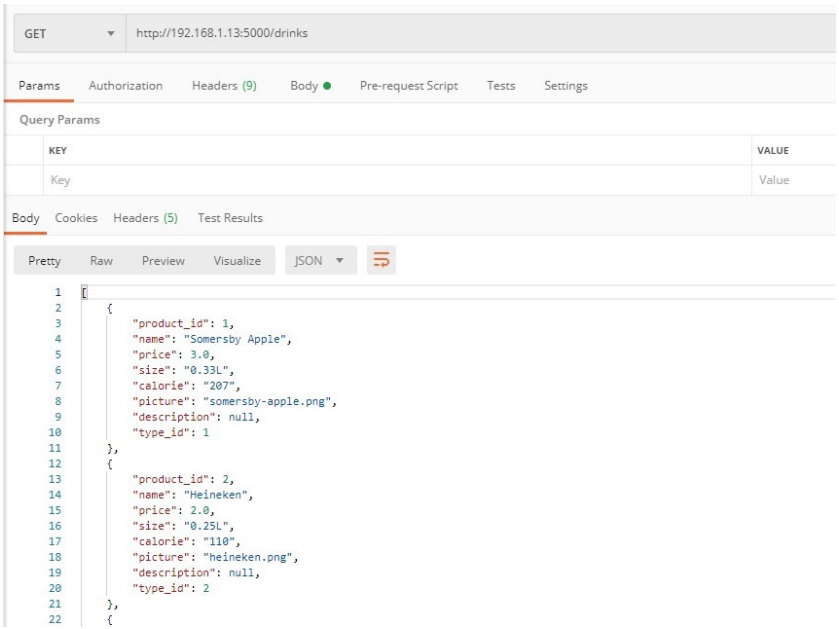
Slika 2.4: Funkcija, ki kliče

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

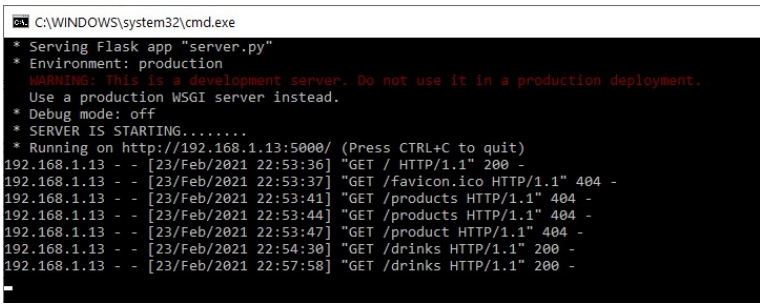
Slika 2.5: Funkcija, ki kliče

brisanje podatkov v podatkovni bazi, vendar zato more biti ustrezna HTTP metoda POST. Spremljanje zahtevkov, ki prihajajo na strežnik, je mogoče preko CLI vmesnika, slika 2.7, ki v primeru nepopolnosti servirajo ustrezno napako.

Strežnik smo napisali v programske jeziku Python in knjižnici Flask. Zaradi težav s sinhronizacijo podatkov v realnem času na strani odjemalca smo znotraj Flask uporabil še WebSocket. To je napredna tehnologija, ki omogoča odpiranje dvosmerne interaktivne komunikacijske seje med odjemalcem in strežnikom. Primer uporabe je npr. pogovor preko družbenih omrežjih. V



Slika 2.6: Primer serviranja podatkov na strežniku



Slika 2.7: Primer spremljanja zahtevkov, ki prihajajo na strežnik

naši aplikaciji so s tem lahko vsi odjemalci hkrati obveščani o spremembah na strani podatkovne baze.

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 spletnim vmesnikov 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.

Zahtevek:

```
C:\Users\lukah>curl -I http://192.168.1.13:5000/drinks
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 5066
Access-Control-Allow-Origin: *
Server: Werkzeug/0.16.0 Python/3.7.3
Date: Tue, 02 Mar 2021 20:23:57 GMT

C:\Users\lukah>curl http://192.168.1.13:5000/drinks
  {
    "product_id": 1,
    "name": "Somersby Apple",
    "price": 3,
    "size": "0.33L",
    "calorie": "207",
    "picture": "somersby-apple.png",
    "description": null,
    "type_id": 1
  },
  {
    "product_id": 2,
```

```
    "name": "Heineken",
    "price": 2,
    "size": "0.25L",
    "calorie": "110",
    "picture": "heineken.png",
    "description": null,
    "type_id": 2
  },
  {
    "product_id": 3,
    "name": "Heineken",
    "price": 3.5,
    "size": "0.5L",
    "calorie": "220",
    "picture": "heineken.png",
    "description": null,
    "type_id": 2
  },
```

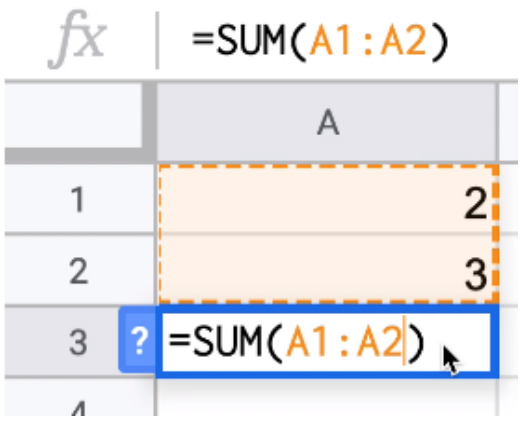

2.3 Odjemalec

Odjemlaca bi načeloma lahko implementirali v spletnih tehnologijah (HTML, CSS, JS) ali pa v namenski mobilni aplikaciji. Mi smo se odločili za programskem jezik JavaScript in knjižnico Vue. Naredili smo odziven in reaktiven vmesnik, ki deluje v realnem času. Vue je eden izmed mnogih kot npr. Angular, Ember, React,... poznan pa je predvsem zaradi enostavnosti za upravljanje in izvajanje testov. Vsem je skupna točka reaktivnost, vendar v drugačnem pomenu besede. Gre za to, da se aplikacija postopno prilagaja glede na vrednosti podatkov. To prednost sem izkoristil in dodal še knjižnico Vuex, ki služi kot centralizirana baza podatkov za vse komponente v aplikaciji. Podatke i

Da bi to dosegli ne bi potreboali JavaScript zaradi RESTful arhitekutre, ki omogoča katerokoli platformo na strani odjemalca.

2.3.1 Kaj je reaktivnost?

Reaktivnost [3], je programska paradigma, ki nam omogoča, da se na deklarativni način prilagodimo spremembam. Dober primer reaktivnosti je npr. funkcija SUM, ki jo uporabljamo v Excelu. Slika 2.8 prikazuje primer v Excelu.



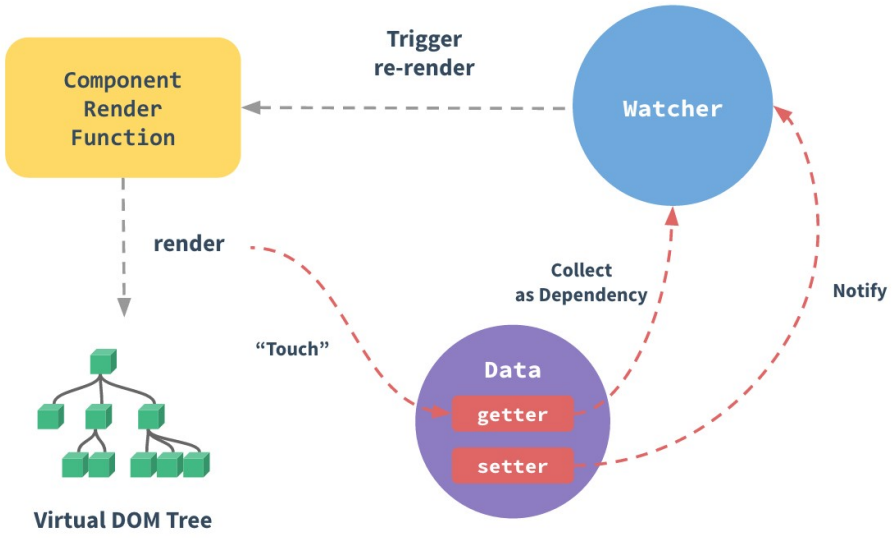
Slika 2.8: Primer fukcije SUM v programu Excel

Če vstavimo številko 2 v prvo celico in številko 3 v drugo celico ter izberemo funkcijo SUM teh dveh celic. Kot rezultat dobimo vsoto obeh številk skupaj, kar ni nič posebnega. Vendar če bomo spremenili vrednost prve celice, bo funkcija SUM avtomatsko posodobila skupno vrednost. 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.

2.3.2 Delovanje reaktivnost v Vue

Vue se torej v primerjavi z navadnim JavaScriptom sprehodi skozi podatke in njihove lastnosti (angl. Properties) pretvori v fukciji Getter in Setter, ki sta nevidni uporabniku [2]. Poglejte si sliko 2.9 za lažjo predstavo.

Torej funkcija Getter pokliče instanco Watcher z namenom odvisnosti do drugih komponent. To pomeni, če je podatek označen kot odvisen (angl. Dependency) bodo nekateri deli programske kode oziroma funkcije poklicane vsakič, ko se spremeni vrednost podatka. Funkcija Setter pa obvesti instanco Watcher, vsakič ko se podatku spremeni lastnost. Ta poskrbi, da se pokliče funkcija upodabljanja (angl. Render) tiste komponente, ki potem prikaže spremembe v samem pogledu aplikacije.



Slika 2.9: Kateri dialekt uporabljati?

2.3.3 Knjižnice in dodatki

Tako kot vsak programski jezik ima tudi Vue svoje dodatke, ki pomagajo pri razvoju aplikaciji. Spodnje smo uporabljali in so vredni omembe.

Vue CLI velja kot standardno orodje za ekosistem Vue [5]. 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 [6]. 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]. Cilje 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.

Poglavje 3

Delovanje aplikacije

3.1 Vmesnik za gosat

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] How changes are tracked? Dosegljivo: https://vuejs.org/v2/guide/reactivity.html#How-Changes-Are-Tracked. [Dostopano: 31. 10. 2020].

[3] Reactivity in depth. Dosegljivo: https://v3.vuejs.org/guide/reactivity.html#what-is-reactivity. [Dostopano: 31. 10. 2020].

[4] Toad data modeler. Dosegljivo: https://en.wikipedia.org/wiki/Toad_Data_Modeler. [Dostopano: 23. 02. 2021].

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

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

[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 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].