Sustav za naručivanje u ugostiteljskim objektima

Kovačević, Filip

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:491460

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGINA OSIJEK

Sveučilišni studij

Sustav za naručivanje u ugostiteljskim objektima

Diplomski rad

Filip Kovačević

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 21.09.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Filip Kovačević
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 901 R, 27.09.2019.
OIB studenta:	85611463814
Mentor:	Izv. prof. dr. sc. Krešimir Nenadić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva 1:	Izv. prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 2:	Dr. sc. Tomislav Galba
Naslov diplomskog rada:	Sustav za naručivanje u ugostiteljskim objektima
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Opisati postojeći način naručivanja u ugostiteljskim objektima. Opisati prijedlog kako se naručivanje može olakšati osoblju i gostima. Izraditi i testirati Android aplikaciju za naručivanje. Aplikacija treba uslikati kod (QR, linijski) koji se nalazi na stolu kojim se identificira ugostiteljski objekt i stol za kojim gost sjedi. Nakon obavljene narudžbe, sadržaj iste se treba prikazati osoblju na njihovom uređaju.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	21.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

IZJAVA O ORIGINALNOSTI RADA

Osijek, 20.10.2020.

Ime i prezime studenta:	Filip Kovačević
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 901 R, 27.09.2019.
Turnitin podudaranje [%]:	4%

Ovom izjavom izjavljujem da je rad pod nazivom**: Sustav za naručivanje u ugostiteljskim objektima**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Krešimir Nenadić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1
1.1 Zadatak diplomskog rada 1
2. PRIMIJENJENE TEHNOLOGIJE 2
2.1 Django framework 2
2.2 Vue.js 3
2.3 Git Lab 4
2.4 PostgreSQL baza podataka 4
2.5 Ngrok 5
3. SUSTAV ZA NARUČIVANJE 6
3.1 Virtualno okruženje 6
3.1.1 **Aktivacija virtualnog okruženja** 7
3.1.2 **Instaliranje programskih paketa** 7
3.2 Postavljanje PostgreSQL baze podataka 8
3.2.1 **Kreiranje korisnika i same baze podataka** 9
3.3 Programsko rješenje pozadinske logike (*engl. Backend*) 10
3.3.1 **Postavljanje odgovarajućeg virtualnog okruženja** 11
3.3.2 **Pridruživanje baze i instaliranih aplikacija** 14
3.3.3 Struktura projekta 15
3.3.4 **Modeli projekta** 19
3.3.5 **Views projekta** 24
3.4 Programsko rješenje prednjeg sučelja (*engl. Frontend*) 26
3.4.1 **Proces naručivanja** 26
3.5. Razlike u odnosu na već postojeće sustave za naručivanje 30
4. ZAKLJUČAK 31
LITERATURA 32
SAŽETAK 33
ABSTRACT 34
ŽIVOTOPIS 35

1. UVOD

Cilj ovog diplomskog rada je kreiranje sustava za naručivanje u ugostiteljskim objektima, prvenstveno kafića. Trenutni problem u ugostiteljskim objektima je sporost zaprimanja narudžbe i njezinog dostavljanja. Problemi nastaju jer konobar ne primjeti gosta, ima prevelliku gužvu pa ne stignu poslužiti gosta ili su jednostavno spori. Dizajniranjem sustava naručivanja riješio bi se taj problem te bi ljudi prije dobivali narudžbe.

Bez sustava za naručivanje korisnik bi trebao pričekati konobara, reći mu svoju narudžbu te čekati da mu konobar donese naručeno. Uz uvjet da konobar ne zaboravi, narudžba bi trebala biti spremna u roku 5 minuta. Iako to zna biti i puno više zbog mogućih gore spomenutih poteškoća.

Koristeći sustav za naručivanje korisnik bi jednostavano trebao ukucati zadani URL, mobitelom skenirati QR kod te naručiti svoje piće.

Naručivanje se treba odvijati u realnom vremenu, sa što manje kašnjenja. Isto tako vrlo je bitno imati na umu sigurnost samog objekta kao i samih gostiju. Za sigurnost koristi se QR kod koji se nalazi na samom stolu ugostiteljskog objekta, kao i geo lokacija samog korisnika u odnosu na ugostiteljski objekt.

1.1 Zadatak diplomskog rada

U ovom diplomskom radu potrebno je opisati postojeći način naručivanja u ugostiteljskim objektima. Opisati prijedlog kako se naručivanje može olakšati osoblju i gostima. Izraditi i testirati sustav za naručivanje. Sustav treba uslikati QR kod koji se nalazi na stolu kojim se identificira ugostiteljski objekt i stol za kojim gost sjedi. Nakon obavljene narudžbe, sadržaj iste se treba prikazati osoblju na njihovom uređaju.

2. PRIMIJENJENE TEHNOLOGIJE

U ovom poglavlju opisivati će se sve korištene tehnologije. Isto tako biti će opisan način na koji funkcioniraju te kako se koriste u ovom radu.

2.1 Django framework

Django je besplatan i javno dostupan programska kostur. Baziran je na Python programskom jeziku te prati arhitekturu model-template-view koji je baziran na model-view-controller (MVC) modelu. Kreirali su ga u jesen 2003. godine programeri Adrian Holobaty i Simon Willison, koji su koristili Python za izradu aplikacija. Glavna zadaća Django-a je olakšati izradu kompleksnih web stranica koje koriste web baze za pohranu podataka. Sama programska okosnica zalaže ponovo korištenje komponenti, manje koda, brzo razvijanje koda te se fokusira na princip bez ponavljanja koda. Python se koristi kroz cijeli Django, čak i za postavljanje datoteka i podataka samih modela. Django, također, pruža sučelje za kreiranje, čitanje, ažuriranje i brisanje generirano dinamički preko admin modela.

Iako ima svoju nomenklaturu imenovanja objekata generiranih HTTP (*engl. Hypertext Transfer Protocol*) odgovorom, smatra se da Django slijedi MVC (*engl. Model-View-Controller*) arhitekturu modela. Sastoji se od ORM (*eng.object-relational maper*) koji posreduje između modela (kojeg predstavljaju Python klase) i relacijskih baza podataka (“Model”), te samog sustava za procesiranje HTTP odaziva (*engl. response*) sa sustavom za prikaz weba (“View”) i sustava za raspodjelu mrežne adrese (“Controller”).

Django može biti pokretan u kombinaciji sa Apache, Nginx koji koristi WSGI (*engl. Web Server Gateway Server*), Gunicorn. Službeno, Django podržava korištenje četiri osnovne baze podataka: PostgreSQL, MySQL, SQLite i Oracle. Također, Django se može povezati i sa NoSQL bazama podataka, kao što je MongoDB.

2.2 Vue.js

Vue.js javno je dostupan programski okvir za izradu sučelja i jednostraničnih aplikacija. Koristi model-view-view model model te je pisan u JavaScript programskom jeziku. Kreirao ga je Evan You, koji ga i danas održava zajedno sa aktivnim timom ljudi iz različitih kompanija kao što su Netlify i Netguru.

Glavna srž Vue.js-a je fokus na "view" sloj, te omogućuje proširenje HTML (*engl. Hyper Text Markup Language*) dijelova sa HTML atributima koji se nazivaju direktive. Direktive omogućuju funkcionalnost HTML aplikacijama. Komponente Vue.js-a proširuju osnovne HTML elemente koji obuhvaćaju korišteni kod. Na visokim razinama, komponente su prilagođeni elementi na koje Vue.js-ov kompajler pridruži određeno ponašanje. Komponenta je u samo instanca sa unaprijed definiranim opcijama.

Korištenje HMTL osnovne sintakse omogućava vezanje korištenog DOM (*engl. Document Object Model*) na instancu podatka u Vue.js. Svaki dio koda Vue.js-a može biti parsiran pomoću web preglednika. Vue.js sastavlja predloške (*engl. Template*) u virutalne DOM funkcije za iscrtavanje. Virtualni DOM omogućuje Vue.js-u iscrtavanje komponenti u memoriji DOM-a prije nego ih prikaže u web pregledniku. U kombinaciji s reaktivnim sistemom, Vue.js može izračunati minimalan broj komponenti koje treba ažurirati te primjeniti minimalnu količinu DOM manipulacija kada aplikacija promijeni stanje.

Vue.js ima svojstva reaktivnog sistema koje koristi osnovne JavaScript objekte i optimizirano učitavanje. Svaka komponenta prati stanja varijabli koje mogu promijeniti stanje. Prilikom ažuriranja, sistem zna točno kada ažurirati kao i koju komponentu ažurirati.

Vue.js komunicira sa pozadinskim (*engl. Backend*) dijelom sustava pomoću "*post*" i "*get*" zahtjeva (*engl. Requests*).

2.3 Git Lab

GitLab je platform za razvijanje programa. Omogućuje praćenje problema, kontinuirano razvijanje projekata te rad više developera na jednom projektu u isto vrijeme. Slijedi javno dostupni model razvijanja gdje je glavna funkcionalnost objavljena te joj mogu pristupiti svi. Isto tako svi mogu dodati vlastita rješenja.

GitLab alati nude mogućnost automatiziranja svih faza projekta, od planiranja, do kreacije, provjere, sigurnosnih testiranja, objavljivanja u produkciji. Omogućuje veliku dostupnost, repliciranje i skalabilnost.

2.4 PostgreSQL baza podataka

PostgreSQL (ili samo Postgres) je besplatni sistem za upravljanje bazom podataka (*engl. RDBMS – Relational Database Management System*) fokusiran na mogućnost proširivanja i usklađenost sa SQL (*engl. Structured Query Language*). Prvotno je nazvano Postgres, referencirajući se kao nasljednika Ingres baze podataka razvijenog na Berkley-u na Sveučilištu Kalifornija.

Postgres omogućuje transakcije s konzistentnim, izoliranim izdržljivim svojstvima. Dizajniran je na takav način da može podnijeti veliki raspon radnog opterećenja, od jednog računala do cijelog skladišta računala ili web servisa sa velikim brojem istovremenih korisnika. Omogućuje konkurentnost kroz MVCC (*engl. Multiversion Concurrency Control*), koji daje svakoj transakciji snimak baze te tako omogućuje promjene bez utjecaja na druge transakcije.

Postgres uključuje ugrađenu binarnu replikaciju baziranu na slanju promjena repliciranim nodovima asinkrono, sa mogućnošću pokretanja čitanja upita na repliciranim nodovima. Takav način repliciranja omogućuje raspodjelu čitanja prometa efikasno među više nodova. Raniji sustav koji je omogućavao slično čitanje skaliranja, oslanjao se na dodavanje okidača replikacije na glavnu bazu, povećavajući opterećenje.

2.5 Ngrok

Ngrok je aplikacija korištena na više različitih platformi, a omogućava programerima prikaz lokalnog razvojnog servera na Internet. Aplikacija čini lokalno postavljen web server pokrenut na pod domeni ‘ngrok.com’, te samim time ne zahtjeva javnu IP adresu ili javno ime domene na lokalnom računalu. Slično se može postići i obrnutim SSH tuneliranjem, ali takav pristup zahtjeva više postavljanja kao i posjedovanje i upravljanje vlastitim serverom.

Ngrok uspjeva zaobići NAT (*engl. Network Address Translation*) mapiranje i ograničenja vatrozida kreirajući dugotrajni TCP (*engl. Transmission Control Protocol*) tunel pomoću nasumično kreirane poddomene ‘ngrok.com’ (npr. 5As741ol.ngrok.com) na lokalnom računalu. Nakon specificiranja ulaz (*engl. Port*) koji će server osluškivati, ngrok klijent program pokreće sigurnosnu konekciju na ngrok server. Kao rezultat toga, svatko može napraviti zahtjeve na lokalni server ukoliko ima jedinstvenu ngrok tunel adresu. Ngrok kreira i HTTP (*engl. Hypertext Transfer Protocol*) i HTTPS (*engl. Hypertext Transfer Protocol Secure*) krajnje točke, što ga čini korisnim za testiranje integriranja s trećim stranama.

U ovom radu korištena je besplatna verzija ngrok-a te je zbog toga potrebno promijeniti mrežnu adresu (*engl. URL*) sustava nakon što sesija završi. Plaćena verzija omogućuje korisniku specificiranje točno određenog URL-a na kojem će server biti pokrenut.

3. SUSTAV ZA NARUČIVANJE

U ovom poglavlju opisivati će se sustav za naručivanje. Prvo je objašnjeno kako se postavlja okruženje potrebno za pravilno pokretanje sustava za naručivanje. Zatim je objašnjeno postavljanje baza, virtualnog okruženja, strukturu, te programsko rješenje projekta.

3.1 Virtualno okruženje

Virtualno okruženje (*engl. Virtual Environment*) je alat koji sadrži sve programske pakete potrebne za određeni projekt u zasebnom python okruženju. Virtualno okruženje kreira se pokretanjem naredbe u terminalu. Samo okruženje koristi unaprijed određenu verziju Python programskog jezika. Ukoliko će okruženje koristiti točno određenu verziju, naredbi za kreiranje virtualnog okruženja dodaje se zastava (*engl. Flag*) i ime ili putanja do verzije Python programskog jezika (Slika 3.1.).

$$virtualenv - p\ python3.8\ venv$$

Sl. 3.1. naredba kreiranja virtualnog okruženja

Virtualenv je alat korišten za kreiranje virtualnog okruženja

-p je zastava (flag) nakon koje sustav očekuje putanju do verzije pythona koji će biti korišten u virtualnom okruženju

Venv je ime virtualnog okruženja

3.1.1 Aktivacija virtualnog okruženja

U slučaju da programer radi na više projekata, svaki projekt ima svoje programske pakete i njihove verzije potrebne za pravilan rad projekta. Na taj se način sprječava situacija u kojoj se više paketa istog imena nalazi u istoj mapi te projekt nikada neće učitati nepotrebne programske pakete. Aktivacija virtualnog okruženja moguća je pokretanjem jednostavne naredbe, prema slici 3.2., nakon pozicioniranja u mapu u kojoj se nalazi samo virtualno okruženje.

```
. venv/bin/activate
```

Sl. 3.2. Aktivacija virtualnog okruženja

Deaktivacija virtualnog okruženja omogućena je korištenjem naredbe prema slici 3.3.:

```
deactivate
```

Sl. 3.3. Deaktivacija virtualnog okruženja

3.1.2 Instaliranje programskih paketa

Iako je virtualno okruženje vezano za projekt, ono se nikada ne dodaje u GitLab repozitorij. Kako bi svatko mogao imati podatke potrebne za pravila rad projekta, svaki projekt sadrži *'requirements.txt'* datoteku koja se dodaje na GitLab. U toj datoteci, nalaze se svi programski paketi i njihove verzije potrebne za pravilan rad projekta. Upisivanjem naredbe nakon što je aktivirano virtualno okruženje, instaliraju se svi paketi u virtualno okruženje i projekt je spreman za pokretanje (Slika 3.4.).

```
pip install requirements.txt
```

Sl. 3.4. Instaliranje programa potrebnih za rad sustava

3.2 Postavljanje PostgreSQL baze podataka

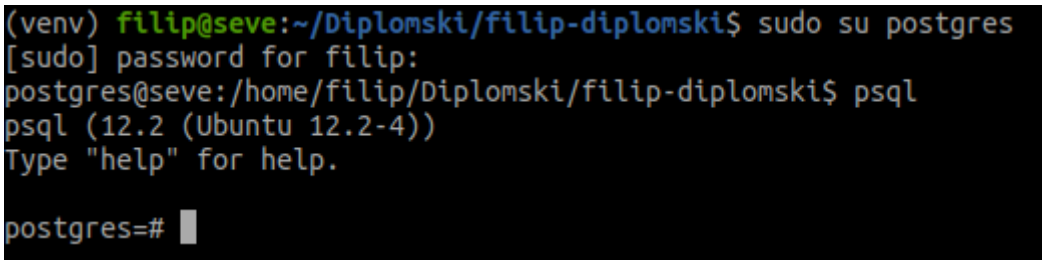
Prije pisanja samog koda diplomskog rada, bilo je potrebno organizirati bazu podataka koja će sadržavati informacije o vlasnicima ugostiteljskih objekata, produktima, narudžbama i samim ugostiteljskim objektima. U tu svrhu korištena je PostgreSQL baza podataka. Kako bi se kreirala baza podataka potrebno je spojiti se na postgres upisujući u terminal naredbu prema slici 3.5.:

sudo su postgres

Sl. 3.5. Spajanje na postgres

Gore navedena naredba spaja se na postgres koristeći '*sudo'* naredbu. Sudo naredba služi za povećavanja prava korisnika, odnosno, daje korisniku sva prava (za instalaciju programa, pristup svim mapama i programima na računalu). Nakon unosa gore navedene naredbe sustav zahtjeva lozinku računala te se spaja na postgres.

Nakon što je pokrenut postgres, potrebno je upisati naredbu '*psql'* kako bi korisnik dobio pristup samoj bazi podataka.



Slika 3.6. Prikaz spajanja na postgres

3.2.1 Kreiranje korisnika i same baze podataka

Prvenstveno je kreiran korisnik koji će imati prava pristupanja bazi te postavljanja i dodavanja tablica u samu bazu. U ovom slučaju korisnik je ‘filip’. Korisnik se kreira naredbom prema slici 3.7.:

```
CREATE USER filip WITH PASSWORD 'filip';
```

Sl. 3.7. Kreiranje korisnika “filip” sa lozinkom “filip”

Nakon kreiranja korisnika, morala se kreirati i sama baza, imena ‘*odering_app’*, korištenjem naredbe prema slici 3.8.:

```
CREATE DATABASE 'ordering_app' WITH OWNER filip;
```

Sl. 3.8. Kreiranje baze podataka sa vlasnikom “filip”

Imena svih baza podataka mogu se saznati upisivanjem naredbe *‘\l’*.

Time je proces kreiranja i postavljanja baze podataka završen. Kreiranje svih relacija objekata i dopuštenja, Django postavlja samostalno, iz napisanog koda koji predstavlja svaki model i veze među njima. Popis svih relacija i njihovih vlasnika može se vidjeti spajanjem na bazu

‘*ordering_app’* pomoću sljedeće naredbe: *‘\c ordering_app’* i pokretanjem naredbe *“\dt”* prema slici 3.9.

```
postgres=# \c ordering_app
You are now connected to database "ordering_app" as user "postgres".
ordering_app=# \dt
                  List of relations
 Schema |              Name               | Type  | Owner
--------+---------------------------------+-------+-------
 public | auth_group                      | table | filip
 public | auth_group_permissions          | table | filip
 public | auth_permission                 | table | filip
 public | auth_user                       | table | filip
 public | auth_user_groups                | table | filip
 public | auth_user_user_permissions      | table | filip
 public | dashboard_userdashboardmodule   | table | filip
 public | django_admin_log                | table | filip
 public | django_content_type             | table | filip
 public | django_migrations               | table | filip
 public | django_session                  | table | filip
 public | jet_bookmark                    | table | filip
 public | jet_pinnedapplication           | table | filip
 public | orderingApp_address             | table | filip
 public | orderingApp_category            | table | filip
 public | orderingApp_coffeeplace         | table | filip
 public | orderingApp_coffeeplacetable    | table | filip
 public | orderingApp_order               | table | filip
 public | orderingApp_orderitem           | table | filip
 public | orderingApp_owner               | table | filip
 public | orderingApp_product             | table | filip
 public | orderingApp_productcoffeeplace  | table | filip
(22 rows)

ordering_app=#
```

Slika 3.9. Popis svih tablica i relacija

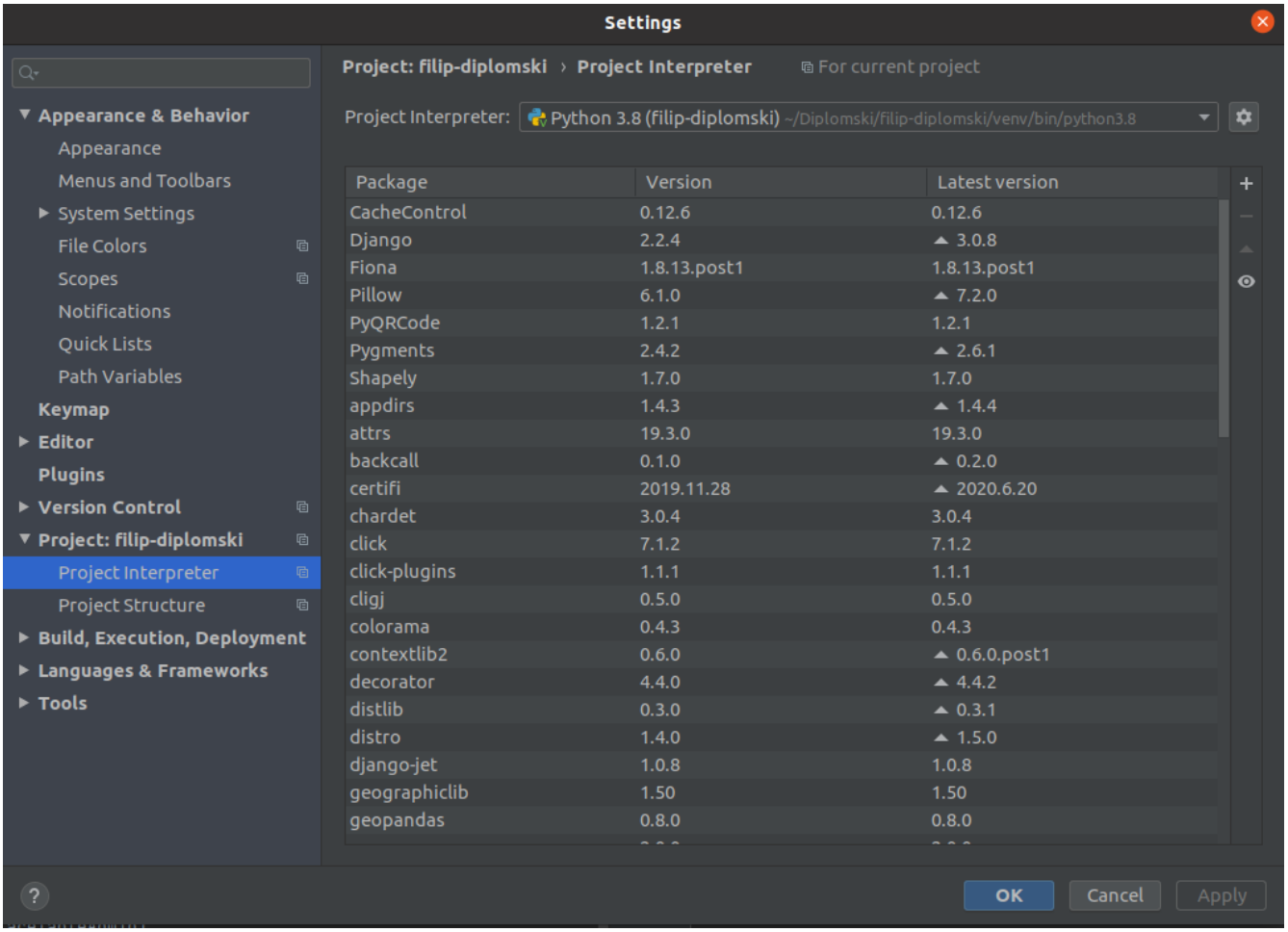
3.3 Programsko rješenje pozadinske logike (*engl. Backend*)

Za razvoj pozadinskog programskog dijela koristi se Django programski okvir. U navedenom programskom okviru piše se Python programski kod koji spaja prednji dio sustava s modelima koji predstavljaju relacije u bazi podataka.

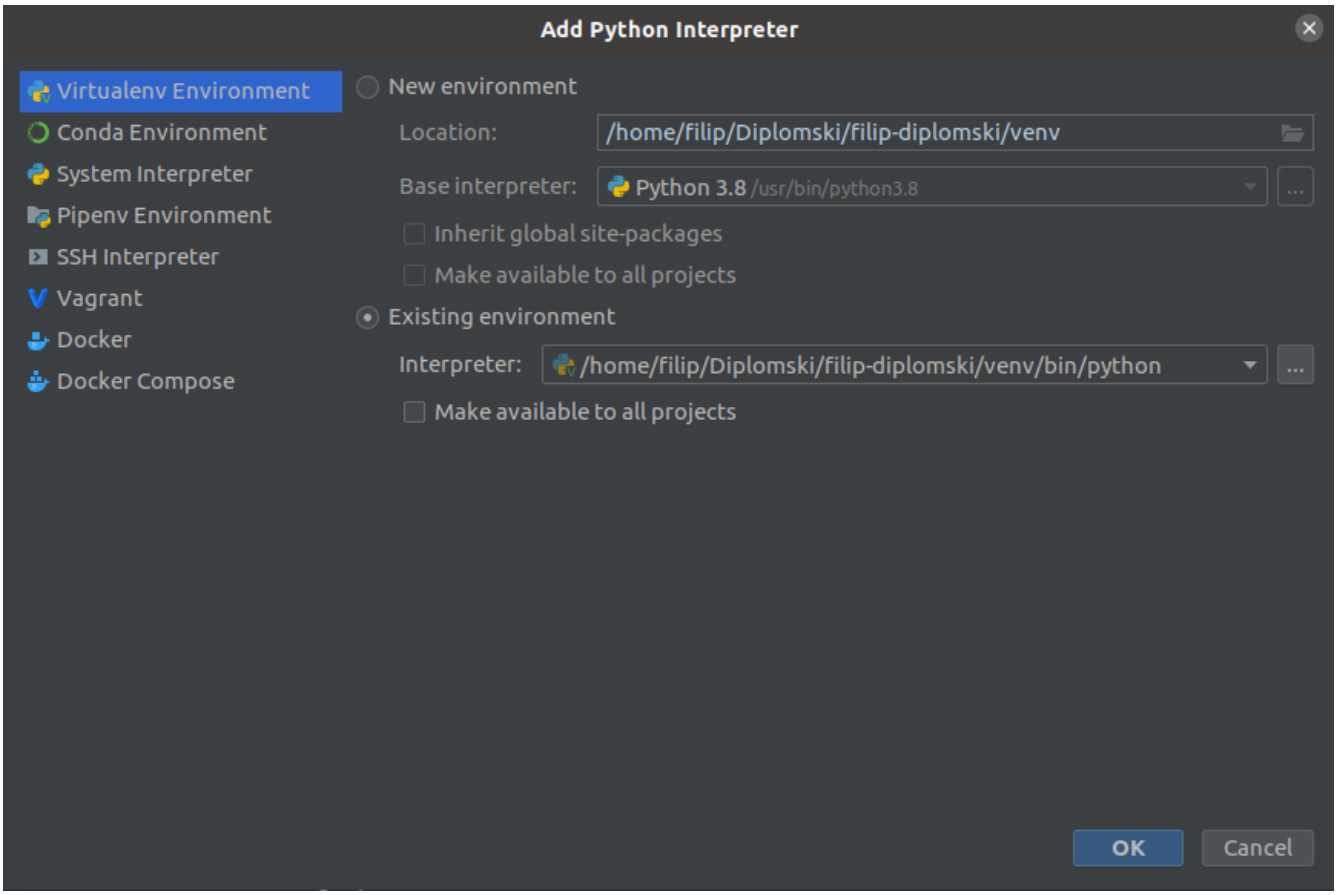
3.3.1 Postavljanje odgovarajućeg virtualnog okruženja

Svaki projekt ima svoje virtualno okruženje koje sadrži sve programske pakete potrebne za pravilan rad projekta. Virtualno okruženje u Django programskom okviru povezuje se na sljedeći način (Slika 3.10. i Slika 3.11.).

File→Settings→Project:<ime projekta>→ Project interpreter



Slika 3.10. Postavljanje interpretera za Django projekt



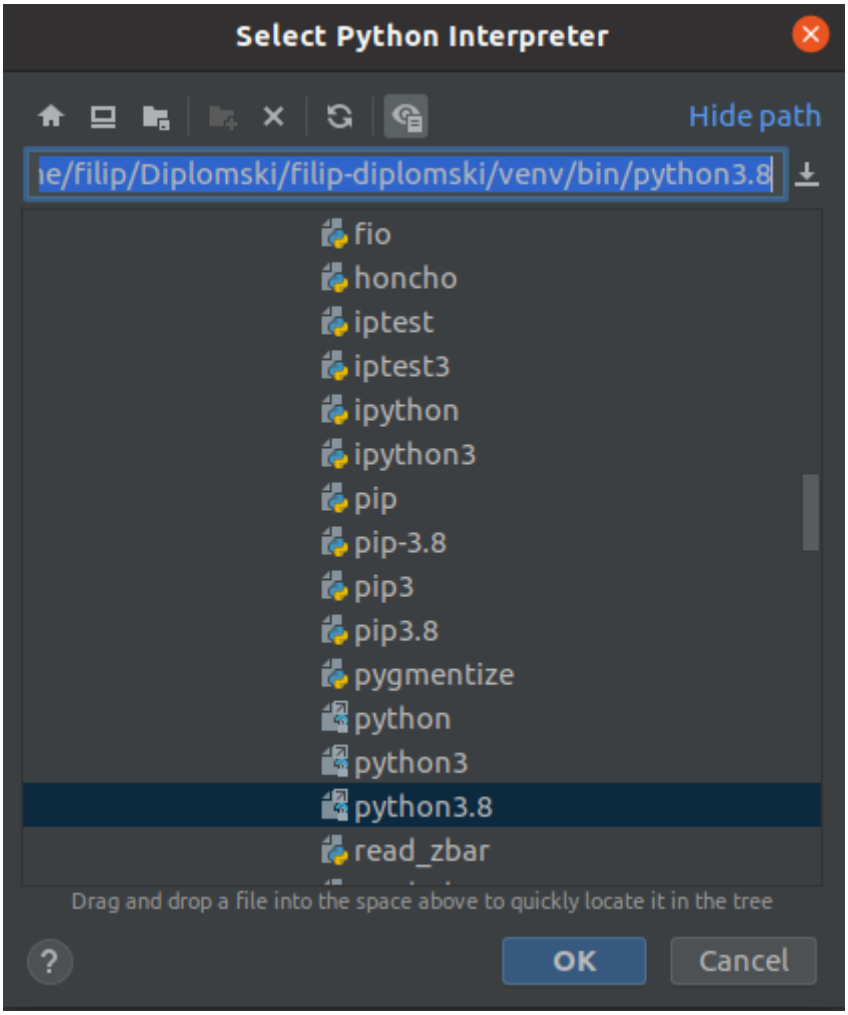
Slika 3.11. Pridruživanje točnog virtualnog okruženja

Pod ‘Project interpreter’, u pogledu Virtual environment odabire se verzija python programskog jezika koju se želi koristiti (Slika 3.12. i Slika 3.13.).

Existing environment→ putanja do našeg virtualnog okruženja → bin → python3.8

Slika 3.12. Odabir prave verzija python programskog jezika

Pridruživanje virtualnog okruženja projektu potrebno je kako bi Django znao koje pakete koristi i kako bi imao pristup njihovim funkcijama.



Slika 3.13. Odabir točne python verzije

3.3.2 Pridruživanje baze i instaliranih aplikacija

Nakon pridruživanja virtualnog okruženja, Django mora znati koju bazu i koje programe koristi. Django olakšava povezivanje sa bazom i programima na način da korisnik upiše imena programa odvojena zarezom u varijablu naziva '*INSTALLED_APPS*` koja se nalazi u '*settings.py'* mapi prema slici 3.14. U istoj mapi, u varijablu imena '*DATABASES'* upisuju se podaci za pristup bazi podataka. Podaci potrebni za pristup bazi su ime baze, korisnik, korisnikova lozinka te naziv adapter prema slici 3.15.

```
INSTALLED_APPS = [
    'orderingApp.apps.OrderingappConfig',
    'jet',
    'jet.dashboard',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Slika 3.14. Popis instaliranih aplikacija

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'ordering_app',
        'USER': 'filip',
        'PASSWORD': 'filip',
        'HOST': 'localhost',
        'PORT': '',
    }
}
```

Slika 3.15. Postavljanje baze u Django programskom okruženju

Psycopg2 je PostgreSQL adapter za Python programski jezik. Glavne značajke su mu kompletna implementacija Python baze podataka te sigurnost niti (više niti može dijeliti istu konekciju). Dizajniran je za više nitne aplikacije koje stvaraju veći broj konkurentnih unosa (*engl. Insert*) i unaprjeđivanja (*engl. Update*). Pomoću njega Django može kreirati relacije u bazi podataka te u njih spremati objekte.

3.3.3 Struktura projekta

Struktura samog projekta sastoji se od glavnog direktorija (imena 'filip-diplomski'), koji predstavlja samu mapu na računalu u kojoj se nalaze sve potrebne datoteke za projekt (virtualno okruženje, requirements.txt). Unutar tog direktorija nalazi se direktorij naziva 'diplomski_rad' i direktorij virtualnog okruženja naziva 'venv'.

Prilikom kreacije direktorija 'diplomski_rad', Django sustav sam kreira direktorija istog imena. U taj direktorij sprema datoteke:

- settings.py
- urls.py
- wsgi.py

Settings.py je datoteka za inicijalizaciju svih postavki bitnih za projekt. Između ostalog postavljanje baze podataka, korištenih aplikacija, dozvoljenih domaćina (*engl. Allowed Hosts*), statičkog URL-a te vremenske zone

Urls.py je datoteka potrebna kako bi Django znao to povezati URL sa odgovarajućim akcijama. Mrežna adresa 'orderingApp/' pokazuje na datoteku urls aplikacije `orderingApp`, dok 'admin/' odgovara Django admin sučelju

Potrebno je dodati direktorij koji će predstavljati korisnikovu aplikaciju, naziva ‘orderingApp’, te direktorij naziva ‘*node_modules’*. Direktorij ‘*node_modules’* sadrži datoteke potrbne za rad sučelja prema korisniku, dok je direktorij ‘orderingApp’ ujedno i mjesto u kojem se nalazi glavni kod ovog diplomskog rada. OrderingApp u sebi sadrži sljedeće datoteke:

- migrations
- static
- templates
- admin.py
- apps.py
- consts.py
- models.py
- tests.py
- urls.py
- utils.py
- views.py

Migrations je datoteka preko koje Django zna kako će prikazati modele iz sustava te njihove relacije u bazi podataka.

Nakon što korisnik kreira modele, više o njima ću opisati kada budem objašnjavao modele datoteku, potrebno je kreirati migracije. Kreiranjem migracija Django zna kako povezati modele s bazom podataka. Migracije se kreiraju korištenjem naredbe u terminalu iz direktorija samog projekta prema slici 3.16.

```
(venv) filip@seve:~/Diplomski/filip-diplomski/diplomski_rad$ python manage.py makemigrations
Migrations for 'orderingApp':
  orderingApp/migrations/0001_initial.py
    - Create model Address
    - Create model Category
    - Create model CoffeePlace
    - Create model CoffeePlaceTable
    - Create model Order
    - Create model Owner
    - Create model Product
    - Create model ProductCoffeePlace
    - Create model OrderItem
    - Add field owner to coffeeplace
    - Add field coffee_place to category
```

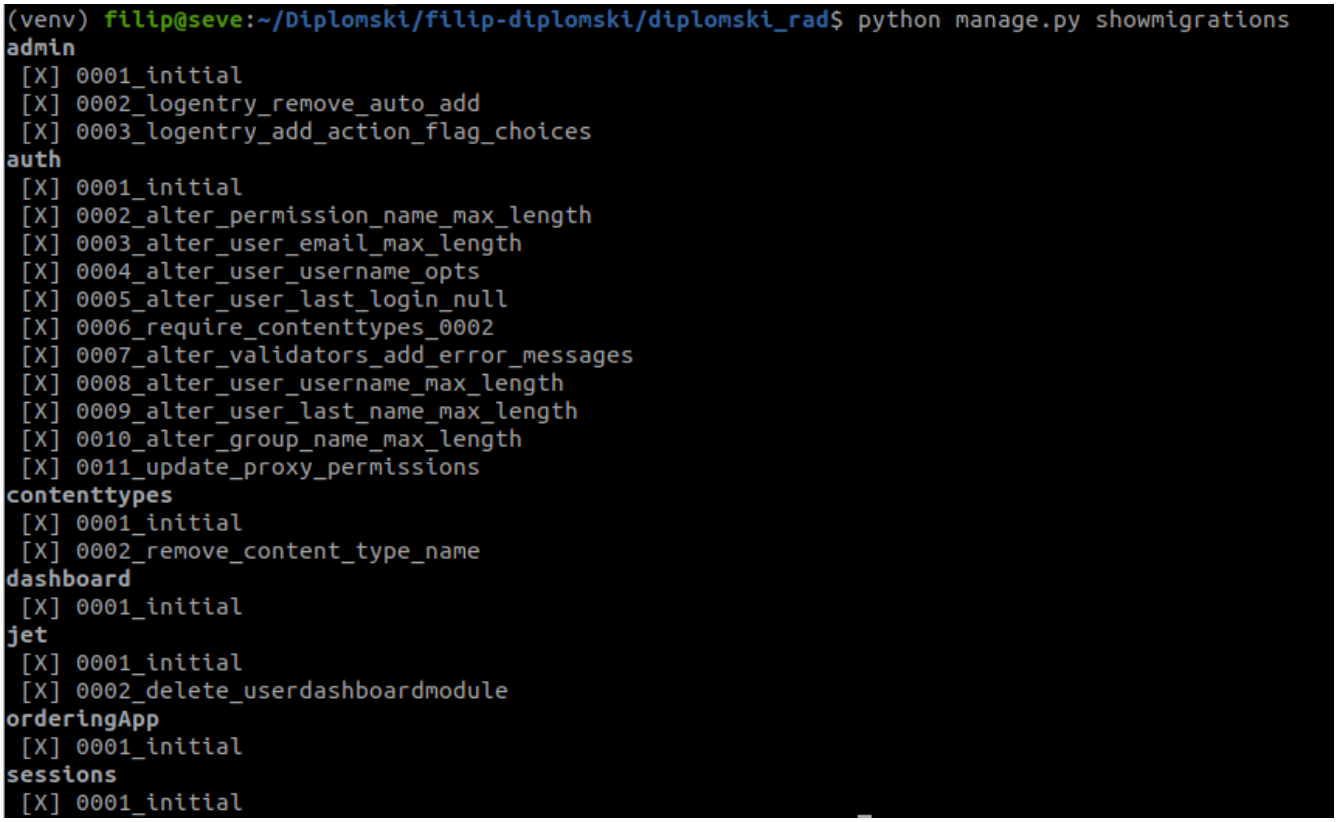
Slika 3.16. Uspješno pokrenuta naredba za kreiranje migracija

Kreiranje migracija samo kreira datoteku koja predstavlja promjene u bazi podataka, no ništa još nije postavljeno u samoj bazi. Za samo postavljanje migracija u bazi služi naredba prema slici 3.17.

```
(venv) filip@seve:~/Diplomski/filip-diplomski/diplomski_rad$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, dashboard, jet, orderingApp, sessions
Running migrations:
  Applying orderingApp.0001_initial... OK
```

Slika 3.17. Uspješno pokrenuta naredba za migriranje

Korisnik vrlo lako može vidjeti sve migracije vezane za projekt pokretanjem naredbe prema slici 3.18.



Slika 3.18. Uspješno pokrenuta naredba za ispisom svih migracija

Static je direktorij koji sadrži sve vezano za prikaz sučelja prema korisniku

Templates je direktorij koji zajedno sa '*static*' direktorijem sadrži svu logiku sučelja prema korisniku

Admin.py je datoteka za prikaz modela i svih relacija prikazanih na '*admin'* URL-u

Consts.py je u ovoj datoteci nalaze se sve konstante. Koristi se kako bi se konstante uklonile iz cjelokupnog koda. Programer lako može uvesti potrebne konstante u sustav iz ove datoteke

Models.py je datoteka koja sadrži sve modele projekta i njihove relacija

Tests.py je sadrži sve potrebne testove sustava

Urls.py je nadovezuje se na '*urls.py'* datoteku iz mape glavnog projekta. Mapira funkcije iz 'views.py' datoteke sa određenim URL-om

Utils.py je datoteka koju kreira programer. Tu se nalazi dio koda za računanje udaljenosti korisnika od ugostiteljskog objekta. Pisanjem određenih funkcija u ovu datoteku dobiva se bolja čitljivost i čistoća samo koda

Views.py je dio koda koji čita zahtjeve (*engl. Requests*) poslane od sučelja prema korisniku (*engl.Frontend*).

3.3.4 Modeli projekta

Modeli u Django, daju informacije o podacima. Sadrže ključna polja i ponašanja podataka koje spremamo. Općenito, svaki model mapira se na jednu tablicu podataka.

Prilikom kreiranja svaki model je Python klasa koja je podklasa ‘django.db.models.Model’. Svaki atribut modela predstavlja polje u bazi podataka. S tim informacijama Django nudi automatski generirani API (*engl.Application Programming Interface*) preko kojeg se može pristupiti bazi podataka.

```
class BaseModel(models.Model):
    date_created = models.DateTimeField(auto_now_add=True)
    date_modified = models.DateTimeField(auto_now=True)

    class Meta:
        abstract = True
```

Slika 3.19. Prikaz *Base* modela

```
class Address(BaseModel):
    street_name = models.CharField(max_length=64)
    street_number = models.CharField(max_length=10)
    city = models.CharField(max_length=25)
    country = models.CharField(max_length=25)

    def __str__(self):
        return f"{self.street_name} {self.street_number} {self.city} {self.country}"
```

Slika 3.20. Prikaz modela adrese

Za primjer modela uzeti su *"BaseModel" i "Adress"* modeli. Klasa '*BaseModel*', prema slici 3.19., predstavlja bazu koja nasljeđuje Djangov '*django.db.models.Model'*. BaseModel na sebi ima dva polja, '*date_created*' i '*date_modified'*.

Iz Djangove dokumentacije može se vidjeti da su oba polja tipa `*DateTime*` što znači da sadrže datum i vrijeme. Djangov '*DateTime*' field ima dodatnih argumenata. Argument '*auto_now*' postavlja vrijednost polja na trenutno vrijeme i datum kada je objekt spremljen. Takvo polje postaje korisno ukoliko se želi imati informaciju o promjeni objekta, pogotovo jer se polje ne može prepraviti.

Argument '*auto_now_add'* služi za postavljanje vrijednosti polja na trenutni datum i vrijeme. Za razliku od '*auto_now'*, '*auto_add_now'* se postavlja kada je model prvi puta kreiran. Također kao i kod prethodno spomenutog polja, datum i vrijeme ne mogu biti prepravljeni.

Razlog za kreaciju ovog modela je taj što sada svaki objekt nasljeđuje '*BaseModel'* te tako na sebi automatski imati datum i vrijeme kreiranja, kao i datum i vrijeme kada je objekt ažuriran. Zbog toga sustav omogućuje filtriranje objekata po datumima, te vlasnik ugostiteljskog objekta može vidjeti profit od određenog datuma, sve narudžbe toga datuma.

U '*Address'* klasi, prema slici 3.20., vidljivo je nasljeđivanje modela '*BaseModel*'. Osim polja '*date_created*' i '*date_modified*', sam model sadrži polja '*street_name*', '*street_number*', '*city*' i '*country*'. Sama imena polja opisuju koju informaciju sadrže. Sva polja ovog modela su tipa

‘*CharField*’ što predstavlja string vrijednosti, a broj u zagradi označava maksimalnu duljinu polja u znakovima. Možda izgleda čudno i kontraintuitivno što je polje ‘*street_number*’ tipa string umjesto integer, ali razlog tome je što neke adrese imaju slovo uz broj ulice (npr. 5B). Ukoliko se ugostiteljski objekt nalazi na sličnoj adresi, a polje zahtjeva unos broj doći će do pucanja samog koda.

Model može sadržavati funkciju, koja se može pozvati te vratiti određene varijable. Na većini modela sustava kreirana je funkcija ‘*__str__*’ koja na poziv vraća string vrijednosti polja modela. Znači kada se model kreira, u bazi će biti vidljivo ono što će funkcija ‘*__str__*’ vratiti. Poanta iza ‘*__str__*’ funkcije je prikaz podataka pomoću kojih odmah možemo reći o kojem se objektu radi.

Ostali modeli sustava:

- Owner
- CoffePlace
- CoffeePlaceTable
- Product
- Category
- ProductCoffeePlace
- Order
- OrderItem

Owner je model na sebi ima polja ‘*first_name*’ i ‘*last_name*’. Polja su sama po sebi opisna, sadrže ime i prezime vlasnika ugostiteljskog objekta. Također, funkcija ‘*__str__*’ vraća iste vrijednosti u formatu “<*Ime vlasnika*> <*Prezime vlasnika*>”

CoffeePlace je model predstavlja podatke o samom kaficu (ugostiteljskom objektu). Sadrži polja ‘*name*’, ‘*address’* i ‘*owner’*. Polje ‘name’ je tipa *string* te predstavlja ime kafića, dok je ‘*address’* poveznica na adresu kafića, a ‘*owner’* poveznica na vlasnika kafića. Oba polja su tipa ‘*ForeignKey’*, što predstavlja relaciju više prema jedan (*engl. Many-to-One*). Odnosno, više kafića može se nalaziti na jedno adresi, te više kafića može imati jednog vlasnika. Polje tipa ‘*ForeignKey’* prihvaća različite argumente koji određuju kako relacija funkcionira. Argument

‘*on_delete’* određuje što će se dogoditi kada se objekt, na koji se ‘*coffee_place’* polje referencira, obriše. Argument ‘*on_delete’* polja ‘*address’* ima vrijednost ‘*models.CASCADE’*, što znači da će se objekt tipa ‘*CoffeePlace’* obrisati ukoliko se obriše objekt tipa ‘*Address’*. Ima smisla da kafić ne može postojati na nepostojećoj adresi. Sa druge strane, vrijednost argumenta ‘*on_delete’* je ‘*models.SET_NULL’* što znači da ukoliko se obriše objekt vlasnik, kafić i dalje postoji na određenoj lokaciji, te se vrijednost polja ‘*owner’* postavlja na nul vrijednost. Uz ‘*SET_NULL’* moramo omogućiti da polje smije sadržavati ‘*null’* vrijednost. CoffeePlace model na sebi uz *‘__str__’* funkciju koja vraća ime kafića, sadrži i ‘*get_orders’* funkciju, koja dohvaća te vraća sve objekte modela ‘*Order’* za taj kafić. Na taj način izbjeglo sa dodatno filtriranje u daljnjem kodu. Umjesto filtriranja, čime se dobije objekt modela ‘*CoffeePlace’,* možemo odmah dohvatiti sve narudžbe za taj kafić.

CoffeePlaceTable – svaki kafić ima svoje stolove. Svaki kafić može imati više stolova. Klasa ‘*CoffeePlaceTable’* predstavlja stol u ugostiteljskom objektu, te sadrži 3 polja. Polje ‘*number’*, tipa *string*, koji predstavlja broj stola, ‘*QR_code’,* tipa string, koji predstavlja QR kod, nalazi se na fizičkom stolu koji sadrži šifru stola, te ‘*coffee_place’*, relacija više prema jedan (*ForeignKey*). Polje ‘*coffee_place’* služi kao veza između stolova i kafića kojem pripadaju. Ukoliko se obriše objekt tipa ‘*CoffeePlace’*, briše se i svi stolovi toga kafića, jer stolovi ne mogu postojati ako ne postoji kafić u kojemu se stolovi nalaze. Funkcija *‘__str__’* vraća ime, string reprezentaciju imena kafića i broja stola.

Product je model koji predstavlja piće. Sadrži samo *ime*, tipa string, što predstavlja ime pića. Svaki kafić će preko određenih modela pokazivati na ovaj model te će se na taj način povezati piče i kafić u kojem se to piće poslužuje. Funkcija *‘__str__’* vraća ime objekta.

Category je model koji predstavlja kategoriju pića. Na sebi sadrži polje ‘*name’,* tipa *string*, i ‘*coffee_place’*. Polje ‘*Name’* sadrži ime kategorije pića dok ‘*coffee_place*‘ predstavlja vezu jedan naprema više, tj. jedan kafić može imati više kategorija. Ukoliko objekt tipa ‘*CoffeePlace’* više ne postoji ne postoje niti objekti tipa ‘*Category’* za taj kafić. Metoda *‘__str__’* vraća string reprezentaciju imena kategorije i imena kafića.

ProductCoffeePlace je model koji predstavlja produkte, tj. pića, jednog kafića. Sadrži polja ‘*product’*, ‘*coffee_place’*, ‘*price’*, ‘*category’* i ‘*quantity’*. Polja ‘*product’* i ‘*coffee_place’* relacije

su jedan prema više na '*Product'* i '*CoffeePlace'* objekte. Ukoliko se obriše '*Product'* ili '*CoffeePlace'* objekt, brišu se i '*ProductCoffeePlace'* objekti. Polje '*Price'* predstavlja cijenu samog produkta (pića) te je stoga tipa *float*, a zadana vrijednost mu je nula. Polje '*category'*, baš kao i polja '*product'* i '*coffee_place'*, je relacija tipa jedan prema više, koji pokazuje na objekt tipa '*Category'*. Ukoliko se obriše objekt kategorije, samo '*Category'* polje se postavlja na '*null'* vrijednost. Polje koje govori o količini naručenog produkta je '*quantity'*. Tip '*quantity'* polja je integer, a zadana vrijednost mu je nula jer ništa još nije naručeno kada se korisnik poveže na menu kafića. Sam model ima dvije metode, *'__str__'* koja vraća string reprezentaciju produkta, cijene i količine produkta, te '*to_dict'* metoda koja vraća ključ – vrijednost (*engl. key – value*) reprezentaciju objekta kako bi mu bilo lakše pristupiti prilikom obrade u Vue.js.

Order je model koji predstavlja koji stol u kojem kafiću je naručio piće. Sam objekt tipa '*Order'* ne predstavlja što je naručeno, nego pomoću metode '*get_order_items'* dohvaća '*OrderItem'* objekt koji na sebi sadrži ime produkta, trenutnu cijenu i produkt i količinu koja je naručena. Na sebi sadrži relacije jedan prema više na '*CoffeePlaceTable'* i '*CoffeePlace'* objekte. Ukoliko se obriše '*CoffeePlaceTable'* objekt, '*coffee_place_table'* polje se postavlja na '*null'* vrijednost, dok se vrijednost polja '*coffee_place'* briše ako se obriše '*CoffeePlace'* objekt. Polje 'status' označava status same narudžbe. Status može imati vrijednosti '*ordered'*, '*served'* ili '*paid'*. Zadana vrijednost status polja je '*ordered'* budući da se objekt kreira kada korisnik naruči piće. Vrijednost samog polja ovisi o tome je li konobar poslužio narudžbu gostu kao i je li gost platio narudžbu. Uz metodu *'__str__'* koja vraća string prezentaciju broja kafića, '*Order'* objekt sadrži metodu '*get_order_items'*. Get_order_items metoda vraća sve '*OrderItem'* objekte za '*Order'* objekt.

OrderItem je model koji povezuje narudžbe s produktom (pićem), količinom produkta, kafićem i cijenom produkta u određenom kafiću. Sadrži polja '*order'*, '*product_coffee_place'* i '*current_price'*.

Polja '*order'* i '*product_coffee_place'* su relacija na objekte narudžbi i produkata određenog kafića. Ukoliko se objekti, na koje se ta polja referenciraju obrišu, briše se i '*OrderItem'* objekt. Polje '*current_price'* je tipa float te je jednak vrijednosti cijene produkta sa '*ProductCoffeePlace'* objekta. Samo polje se koristi u slučaju da vlasnik promijeni cijene pića, kako bi se moglo usporediti dobit prije i promjene pića.

3.3.5 Views projekta

"*Views"* su python funkcije koje primaju mrežne (*engl. Web*) zahtjeve (*engl. Requests*) te vraćaju Web odgovor (*engl. Response*). Odgovor može biti HTML (*engl. Hypertext Markup Language*) sadržaj, 404 greška (*engl. Error*), XML (*engl. Extensible Markup Language*) dokument čak i slika. Sam '*view'* sadrži svu logiku potrebnu za vraćanje toga odgovora. Iako se kod za obradu zahtjeva može nalaziti bilo gdje u projektu, dokle god je u Python putanji (*engl. Path*), konvencija Djanga nalaže da se nalazi u `*views.py*` datoteci.

Funkcije koje se nalaze u pogled (*engl. Views*) datoteci, pozivaju se u '*urls.py'* datoteci. Svaka funkcija odgovara specifičnom url-u. Kada korisnik pristupi određenom url-u poziva se funkcija mapirana na taj url.

U pogled datoteci ovoga projekta, nalazi se pet funkcija:

- Index
- Result
- Products
- Order
- result

Svaka funkcija prima zahtjev (*engl. Request*) kao argument, koja onda, ukoliko je to potrebno, vraća obrađene podatke nazad ili vraća sami HTML sadržaj. Funkcije '*index'* i '*result'* vraćaju HTML sadržaj, koji onda uzima podatke iz '*app.js*' (index funkcija) ili '*result.js'* (result funkcija).

Funkcija products dobiva podatke poslane od '*app.js'* putem POST zahtjeva (*engl. Request*). Podaci koje prima sadrže dekodiran qr kod, te geografsku dužinu i širinu korisnika. Kada primi podatke, funkcija prvo pretvara tijelo (*engl. Body*) zahtjeva iz json formata u Python riječnik (*engl. Dictionary*) što je ključ – vrijednost (*engl. Key – Value*) struktura podataka gdje se preko ključa pristupa podacima. Iz tijela zahtjeva dobiva se vrijednost qr koda te lokacija korisnika.

Sustav zatim dohvaća sve objekte modela '*CoffeePlaceTable'*, koji predstavljaju stolove kafića a koji odgovaraju qr kodu. To je uvijek točno određeni stol jer je qr kod stola jedinstven za svaki stol. Nakon toga, preko objekta stola kafića, dohvaća se kafić kojem odabrani stol pripada. Preko objekta kafića, sustav ima pristup kategorijama kafića, te dohvaća sve kategorije za specifičan kafić.

Prije nego sustav krene u kreaciju menija kojeg će prikazati korisniku, potrebno je odrediti udaljenost korisnika od stola kafića. Udaljenost se računa kao neka vrsta sigurnosti da je korisnik stvarno za stolom. Budući da bi QR kod stajao na stolu, vrlo lako je uslikati kod te ga skenirati na udaljenoj lokaciji, napraviti narudžbu a kada konobar donese piće za stolom nema nikoga. Kako bi se to spriječilo koristila se sigurnost preko geo lokacije. Pomoću Python programskog paketa '*geo.py'* dobiva se geo lokacija kafića preko njegove adrese. Zatim se uzima geo lokacija samog korisnika iz *post* zahtjeva, te se računa udaljenost korisnika od stola. Prvotno se udaljenost računala pomoću Haversine formule, no rezultati nisu bili zadovoljavajući. Haversine formula uzima u obzir da je Zemlja sfera što rezultira u pogrešci od 0,5%. Kako bi se popravila preciznost, korištena je Vincentova formula koja (formula korištena u geodieziji za računanje udaljenosti između dvije točke na površini zaobljene sfere). Sama formula koristi precizniji elipsoidni model kao WGS-84 i implementirana je u '*geo.py'*. Nakon što se dobije udaljenost korisnika od stola kafića provjerava se je li udaljenost u granicama dogovorene udaljenosti za određeni kafić. Ovisno o terasi kafića, odobrena udaljenost može se razlikovati, što je odlučeno u dogovoru sa vlasnikom kafića. Ako je udaljenost korisnika veća od dozvoljene udaljenosti, funkcija '*products'* vraća varijablu vrijednosti 'false' kao JsonResponse. Ukoliko se sustav uvjeri da je korisnik dovoljno blizu stola, kreće u kreaciju menija koji će biti prikazan korisniku. Kada je meni kreiran sustav vraća meni kafića, broj stola, qr kod stola i potvrdu da je udaljenost u dopuštenim granicama.

Druga funkcija u pogled (*engl. Views*) datoteci je 'orders' funkcija. Orders funkcija pokreće se nakon što korisnik odabere pića te pritisne tipku (*engl. Button*). Nakon pretvaranja json formata u Python riječnik, funckija orders sadrži podatke o naručenim pićima, qr kodu stola te broju samog stola. Tada sustav dohvaća stol na kojem se nalazi dobiveni qr kod. Sustav zatim

provjerava je li broj pića veći od nula, te ukoliko je vrijednost veća on nule dohvaća objekt tipa '*ProductCofeePlace'* na kojem se nalazi stol sa dobivenim qr kodom.

Sustav prolazi kroz svaki objekt narudžbe te za svaki objekt kreira objekt tipa '*Order'* u kojem se nalaze podaci o stolu na kojem je narudžba, kafiću kojem stol pripada te postavlja status narudžbe na '*ordered'*. Nakon toga kreira se objekt tipa '*OrderItem'* u koji sprema narudžbu, povezuje produkte narudžbe sa produktima kafića i trenutnu cijenu naručenog produkta. Kada su objekti kreirani i spremljeni, zaposlenici kafića mogu na zasebnoj mrežnoj adresi (*engl. URL*) vidjeti produkte narudžbe, ukupnu cijenu cijele narudžbe te broj stola na koji trebaju dostaviti narudžbu.

Prilikom posjete radnika mrežnoj adresi *'/orderingApp/result'* sustav pokreće '*result.html'* sadržaj. Čim se '*result.html'* sadržaj pokrene, sustav pokreće '*result.js'* datoteku koja šalje get zahtjev na server. Server zahtjev prosljeđuje '*get_orders'* funkciji u pogled datoteci. Get_orders funkcija služi za dohvaćanje svih narudžbi vezane za određeni kafić. Sustav dohvaća sve kafiće te onda među njima traži narudžbe koje su vezane za stol kafića. Kada pronađe stol sa kojeg je naručeno piće, sustav prolazi kroz svs produkte narudžbe te kreira Python riječnik u kojeg sprema ime produkta, njegovu cijenu te količinu. Nakon što je sustav prošao kroz sve produtke, dobiveni riječnik šalje kao Json odgovor na '*result.js'*.

3.4 Programsko rješenje prednjeg sučelja (*engl. Frontend*)

3.4.1 Proces naručivanja

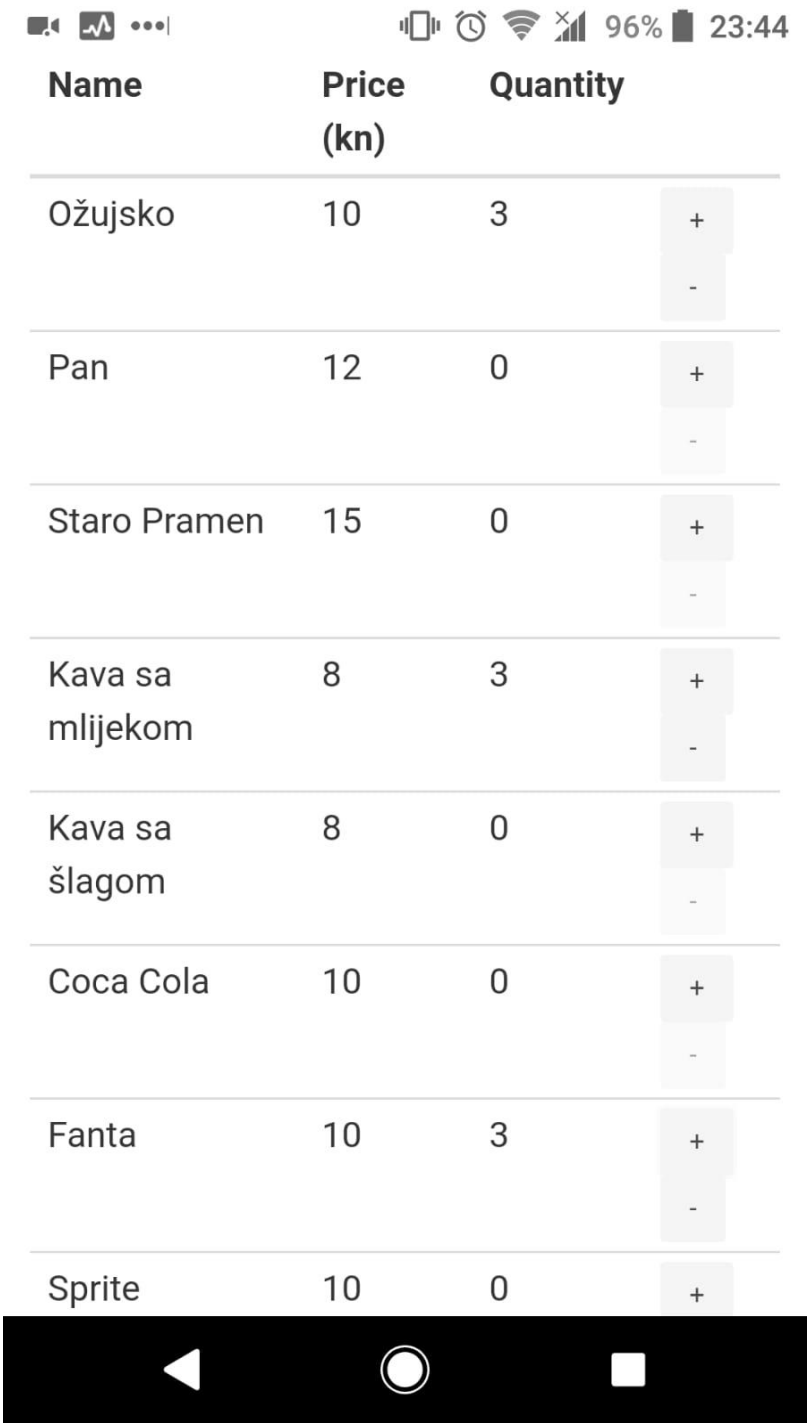
Za rješenje prednjeg sučelja korištena je kombinacija Vue.js programski okvir koji omogućuje povezivanje sa HTML elementima i datotekama. Kada korisnik otvori u svom Web pregledniku mrežnu adresu sustav pokreće views.py funkciju koja prikazuje *index.html*. Pokretanjem html dokumenta pokreće se '*app.js'* datoteka.

App.js datoteka ima funckiju imena '*mounted'* koja se pokreće automatski na pokretanje same '*app.js'* datoteke. Funkcija mounted pokreće dvije funkcije. Prva je '*locateMe*', funkcija koja za cilj ima odrediti lokaciju korisnika koristeći GPS (*engl. Global Positioning System*). U tu svrhu funkcija traži od korisnika dopuštenje za pristup lokaciji koju korisnik omogućiti kako bi mogao koristiti sustav za naručivanje. Nakon što korisnik odobri pristup GPS uređaju, sustav traži dopuštenje kameri kako bi mogao skenirati QR kod. Ukoliko korisnik odbije dati dopuštenje, bilo za GPS ili za kameru, neće moći koristiti sustav za naručivanje.

Nakon što korisnik odobri pristup kameri i GPS uređaju, otvara mu se skener pomoću kojeg korisnik može skenirati qr kod. Kada korisnik skenira qr kod, '*App.js'* pokreće funkciju imena load, kojoj kao argument predaje skenirani kod. Funkcija šalje post zahtjev na mrežnu adresu *'/orderingApp/products'*. Slanjem zahtjeva sa podacima o skeniranom QR kodu i lokaciji korisnika, pozadinski dio sustava određuje udaljenost korisnika od stola kafića. Rezultat obrade, zajedno sa menijem i brojem stola vraća, se onda vraća nazad u '*load'* funkciju. Sa dobivenim podacima od '*products'* funkcije, '*app.js'* kreira meni kojeg pokazuje korisniku.

Korisniku je prikazan meni koji sadrži prema slici 3.21.:

- ime pića
- cijenu pića
- količinu naručenog pića
- + tipka (*engl. Button*) – za povećanje količine
- - tipka – za smanjenje količine
- Order tipka – za naručivanje pića



Sl.3.21. Izgled menija kafića sa odabranim pićima

Kada korisnik odaber piće koje želi, i količinu koju želi može stisnuti tipku za naručivanje. Ukoliko ni jedno piće nije odabrano tipka za naručivanje se ne može pritisnuti. Također korisnik ne može naručiti manje od jednog pića, jer tipka ‘-‘ ne može se pritisnuti ako je količina pića

jednaka nuli. Kada korisnik naruči piće te klikne na '*Order'* tipku, '*App.js'* pokreće funkciju '*sendOrder'* koja šalje *POST* zahtjev na *'orderingApp/order'*.

Order funkcija u pozadinskom dijelu sustava prima zahtjev, kreira objekte tipa '*Order'* i '*OrderItem'* sa pićima iz narudžbe. Time završava cijeli tok naručivanja.

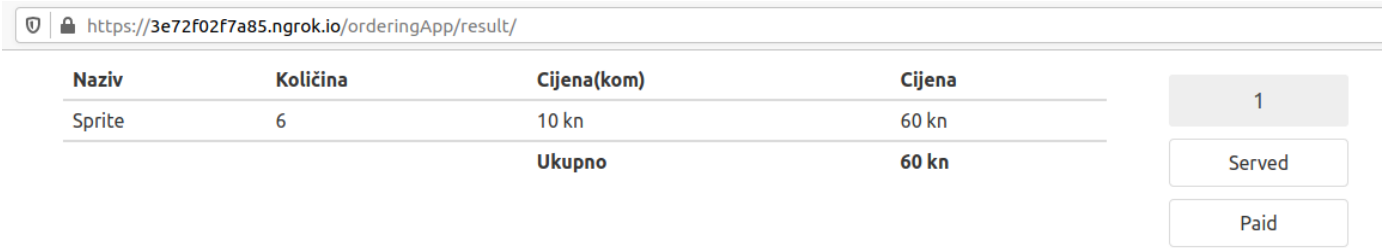
Zaposlenik kafića kod sebe na računalu, tabletu ili mobitelu ima otvorenu stranicu na mrežnoj adresi *'orderingApp/result'*. Slanjem upita za mrežnu adresu '*orderingApp/result'* sustav učitava '*result.html'* datoteku koja pak pokreće '*getOrder'* funkciju u '*result.js'* datoteci. Funkcija '*getOrder'* prolazi kroz sve objekte narudžbe te kreira tablicu koja sadrži prema slici 3.22.:

- Ime pića
- Količina
- Cijena(kom)
- Ukupna cijena količine pića
- Ukupna cijena računa

Slika 3.22. Prikaz narudžbe za stol broj 4 i broj 1

Uz tablicu koja predstavlja narudžbe, kraj svake tablice piše broj stola na koji narudžba treba stići kao dvije tipke, '*Served'* i '*Paid'*. Tipke mijenjaju status same narudžbe, kako bi konobar

znao koju narudžbu je korisnik dobio a koju još treba odnijeti. Ukoliko je korisnik platio narudžbu, konobar stisne tipku '*Paid'*, narudžba se ne prikazuje na ekranu prema slici 3.23.



Slika 3.23. Prikaz kada je narudžba za stol broj 4 plaćena

3.5. Razlike u odnosu na već postojeće sustave za naručivanje

Trenutno postoji više različitih verzija sustava za naručivanje u ugostiteljskim objektima. Jedna od tih je i verzija kreirana i u Hrvatskoj. Sustav *"Smart Service"* omogućuje korisnicima brzo i jednostavno naručivanje preko aplikacije koja skenira QR kod. Smart Service sustav puno je grafički razvijeniji od moga sustava te nudi više opcija. Pregled dijelova menija je puno jednostavnije i preglednije, meni je podijeljen u dijelove prema tipu pića. Sustav nudi mogućnost poziva konobara, te ostavljanja dodatnog naputka konobarima. Uz jelo, sustav sadrži informacije o samom jelu kao i koji su mogući alergeni.

Sam *"Smart Service"* nije loš, ali ne vidim nigdje način na koji štite korisnike i ugostitelja. Sustav opisan u ovom radu nije toliko grafički razvijen ali nudi bolju zaštitu ugostitelja i korisnika. Sustav onemogućuje naručivanje ukoliko korisnik nije na dozvoljenoj udaljenosti od kafića (dogovoreno u razgovoru sa vlasnikom kafića). Ovdje opisani sustav onemogućuje plaćanje karticama te na taj način ne postoji šansa krađe identiteta kartica. Dodatne pogodnosti kao pozivanje konobara za stol, davanje naputaka su nešto što bi se isplatilo dodati.

4. ZAKLJUČAK

Ovim radom prikazana je kreacija pozadinskog sučelja i prednjeg sučelja prema korisniku za rješavanje problema kreiranja sustava za naručivanje u ugostiteljskim objektima.

Pozadinsko sučelje kreirano je pomoću Django programskom okruženja, koji koristi Python programski jezik. Django omogućuje komunikaciju sa različitim bazama podataka, no za ovaj rad odabrana je PostgreSQL baza podataka. Sama komunikacija sa bazama odvija se preko modela, gdje svaki model predstavlja tablicu baze podataka. Model je jedini izvor istine informacija o podacima. Komunikacija pozadinskog i prednjeg sučelja odvija se preko zahtjeva i odgovora. Pozadinsko sučelje obrađuje podatke, koje dobije od prednjeg sučelja, te ih vraća prednjem sučelju preko odgovora (JsonResponse ili HTTPResponse).

Prednje sučelje kreirano je pomoću Vue progresivnog okruženja, koji koristi JavaScript programski jezik. Iako Vue omogućuje lako pokretanje jednostraničnih (engl. Single-Page) aplikacija, ovdje je korišten zbog svog osnovnog dijela knjižnice (engl. library). Vue kao osnovni dio library-a fokusiran je na pogled sloj, koji Django koristi kako bi spremio podatke u bazu podataka. Podaci se, sa prednjeg sučelja, šalju stražnjem sučelju preko zahtjeva.

Iako se trenutno verzija diplomskog rada smatra spremnom za rad, postoji mnoštvo dodatnih funkcionalnosti koje se mogu implementirati. Neke od njih su registracija konobara gdje svaki ugostiteljski objekt može imati više konobara te na računu dodati mogućnost ispisa konobara koji je obavio narudžbu. Mogućnost poziva konobara, ukoliko gost ima dodatna pitanja može pozvati konobara bez da radi dodatnu narudžbu. Hashiran qr kod, ova mogućnost odnosi se na sigurnost podataka, jer svako skeniranje koda pomoću pametnih telefona rezultira kodom koji može biti korišten kako bi se napravio zahtjev na server. Dodavanje vremena rada kafića je još jedna opcija zaštite ugostitelja. Ukoliko bilo tko skenira kod, moguće je naručiti narudžbu, čak iako je radno vrijeme objekta odavno prošlo. Na ovaj način onemogućilo bi se naručivanje ukoliko je radno vrijeme prošlo.

LITERATURA

[1] N., Murray; A., Lerner; H., Djirdeh, *Fullstack Vue : The Complete Guide to Vue.Js*, Createspace Independent Publishing Platform, 2018

[2] D.R., Greenfeld, *Two Scoops of Django 1.11 : Best Practices for the Django Web Framework*, Two Scoops Press, 2017

[3] B., Slatkin, *Effective Python: 59 Specific Ways to Write Better Python*, Pearson Education, 2015

[4] D., Herman, *Effective JavaScript : 68 Specific Ways to Harness the Power of JavaScript*, Pearson Education, 2012

[5] M., Lutz, *Programming Python, Powerful Object-OrientedProgramming*, O'Reilly Media, 2013

[6] D., Herman, *Effective JavaScript : 68 Specific Ways to Harness the Power of JavaScript*, Pearson Education, 2012

[7] D., Flanagan, *JavaScript:The DefinitiveGuide:ActivateYour Web Pages*, O'Reilly Media, 2011

[8] D., Crockford, *JavaScript:The Good Parts*, O'Reilly Media, 2008

SAŽETAK

U ovom radu opisano je rješenje kreiranja sustava za naručivanje u ugostiteljskim objektima. Opisano je rješenje pozadinskog sučelja napisan u Django razvojnom okruženju. Opisano je i okruženje prema korisniku napisano u Vue.ja razvojnom okruženju. Također je razrađeno korištenje cijelog dijela sustava te način na koji sučelje prema korisniku i sučelje prema programeru međusobno komuniciraju kao i način na koji dijele podatke.

Ključne riječi: Django, Vue.js, modeli, zahtjevi, Python

ABSTRACT

System for ordering in coffe places

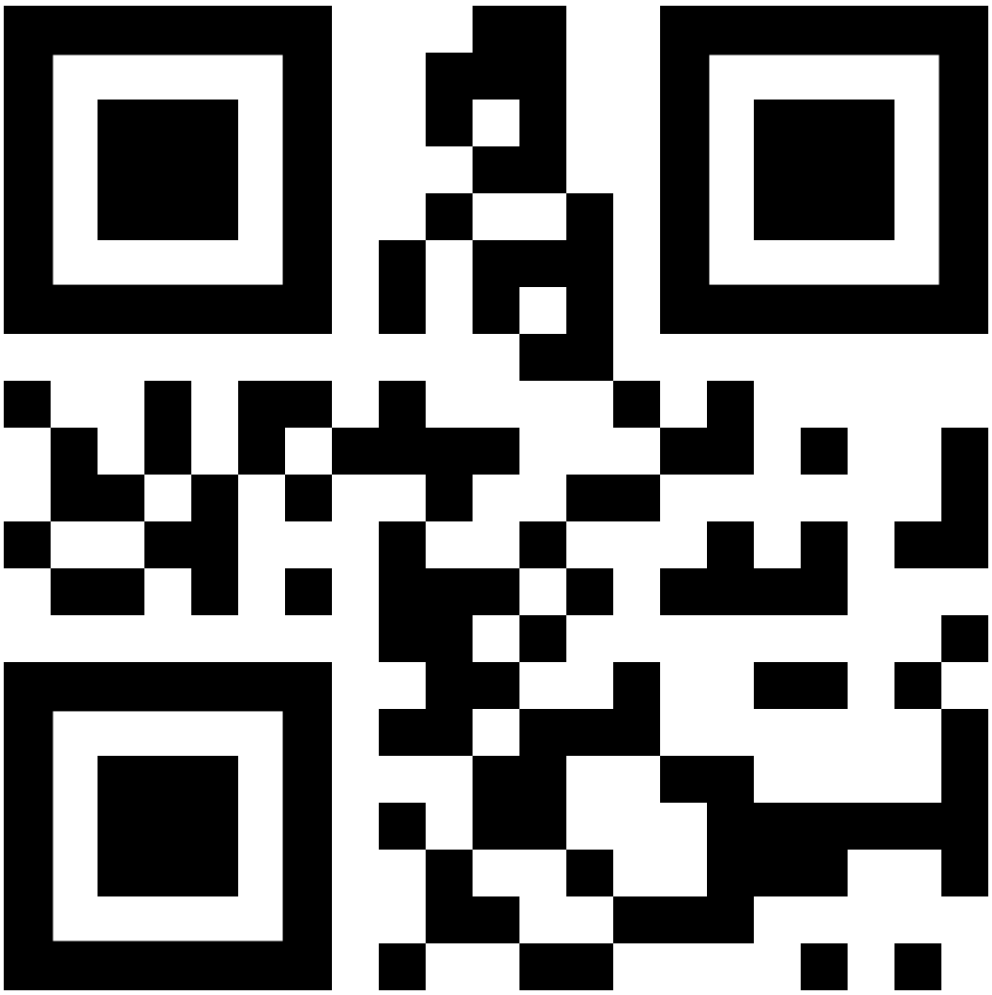
This paper describes solution of creating system for ordering in catering. It is described in details backend part of system written in Django framework as well as frontend written in Vue.js framework. Also is described how entire system works and the way the backend and fronted communicate to each other and the way they share data. This paper also contains instructions on how to set up computer system to be able to run project. It also contains ideas for future upgraded versions.

Key words: Django, Vue.js, models, requests, Python

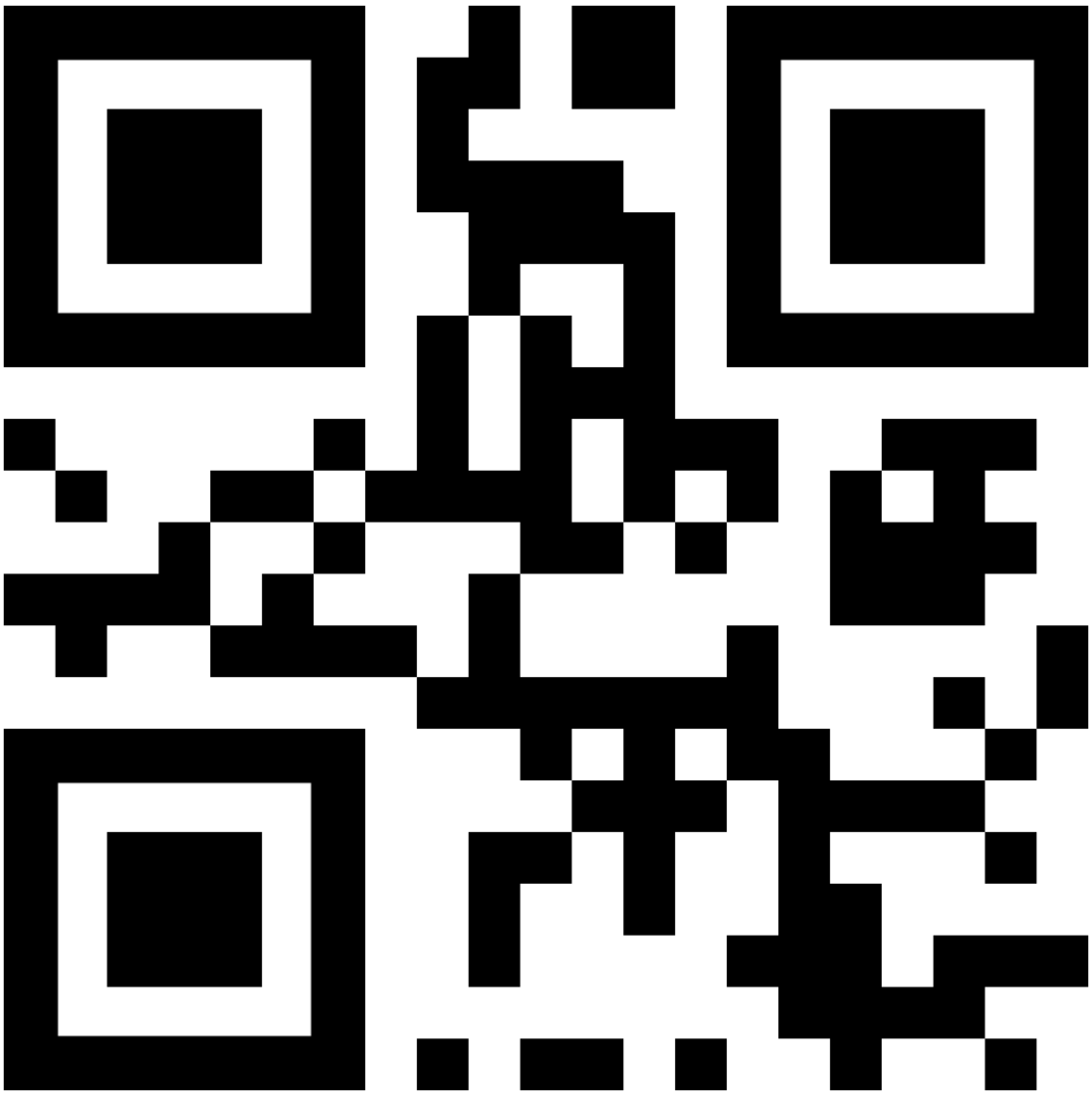
ŽIVOTOPIS

Filip Kovačević rođen je 4. veljače 1994. g. u Zagrebu od oca Emila i majke Milene. Živi sa majkom u Zagrebu. U Lužanima pohađa osnovnu školu “Ljudevit Gaj”. Nakon završetka osnovne škole, upisuje Klasičnu gimnaziju fra Marijana Lanosovića u Slavonskom Brodu. U srpnju 2012. g. upisuje se na Elektrotehnički fakultet u Osijeku, preddiplomski sveučilišni studij računarstva. U rujnu 2016. g. upisuje diplomski studij računarstva, smjer informacijske i podatkovne znanosti. U siječnju 2019. g. zapošljava se u tvrtci “Vingd”, koja početkom 2020. g. mijenja ime u “Mindsmiths” gdje radi i danas.

PRILOG



Prilog 1. QR kod teksta ‘hrvoje4’ – označava kod stola broj 4 ugostiteljskog objekta ‘Kod Hrvoja’



Prilog 2. QR kod teksta ‘dario1’ – označava kod stola broj 1 ugostiteljskog objekta ‘Kod Daria’