Razvoj Web aplikacije u programskom jeziku Python

Vrbančić, Antonio

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Vrbančić

RAZVOJ WEB APLIKACIJE U PROGRAMSKOM JEZIKU PYTHON

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Vrbančić

Matični broj: 0016123582

Studij: Informacijski sustavi

RAZVOJ WEB APLIKACIJE U PROGRAMSKOM JEZIKU PYTHON

ZAVRŠNI RAD

Mentor:

Matija Kaniški, mag. inf.

Varaždin, rujan 2020.

Antonio Vrbančić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U radu će se opisati sintaksa i elementi programskog jezika Python. Spomenut će se tehnologije i alati koje se koriste pri izradi web aplikacija. Objasnit će se komunikacija između Pythona i nerelacijske baze podatka te jezici za predloške u Pythonu. Nadalje, detaljnije će se opisati programski okvir Flask. Programski okvir Flask usporedit će se s ostalim okvirima za web programiranje u Pythonu. Također, u radu će se navesti prednosti i nedostaci Pythona i drugih jezika na strani poslužitelja. Rad se sastoji i od praktičnog dijela u kojem se izrađuje web aplikacija u programskom okviru Flask. Web aplikacija služi za olakšavanje prodaje voća i povrća između obiteljskih poljoprivrednih gospodarstava i trgovačkih centara.

Ključne riječi: Python, Flask, Web aplikacija, MongoDB, programiranje, web servis, jedinično testiranje

Sadržaj

1. Uvod

U vremenu kad je tehnologija postala dio svakodnevice, počela se javljati potreba za izradom Web aplikacija na što brži, jeftiniji i jednostavniji način. Programski jezik Python, uz osnovne tehnologije koje se koriste pri izradi Web aplikacija (HTML, CSS, JavaScript i sl.), omogućuje upravo to. Zbog svoje široke zajednice koja ga koristi, Python ima velik broj vanjskih biblioteka i paketa zbog kojih programer ne mora pisati puno programskoga koda. U početku nije bio zamišljen kao jezik za izradu web aplikacija, ali zbog svoje jednostavnosti te uz napredak tehnologije i razvoj brojnih programskih rješenja Python je postao jedan od najkorištenijih programskih jezika za izradu web aplikacija. Također, programeri u Pythonu zarađuju više od prosjeka. U Pythonu postoje brojni programski okviri za izradu web aplikacija koji olakšavaju njihovu izradu, a najpoznatiji su Django, Flask i Pyramid. Cilj ovog rada jest, kroz primjer Web aplikacije za olakšavanje prodaje voća i povrća između obiteljskih poljoprivrednih gospodarstava (kasnije u tekstu OPG) i trgovačkih centara (kasnije u tekstu Centri), pokazati proces razvoja funkcionalne Web aplikacije u Pythonu i programskom okviru Flask. Primjerima iz aplikacije pokazat će se kako se u Pythonu obrađuju podaci iz obrazaca. Također, pokazat će se na koji način se generira HTML sadržaj preko jezika za predloške. Nadalje, kako se mogu koristiti WebSocket-i u Pythonu. Još će biti opisano i prikazano kako se mogu napraviti web servisi u Pythonu te kako se koriste podaci iz vanjskog API-ja (engl. Application Programming Interface). Dodatno, bit će prikazano kako možemo upravljati greškama u Pythonu. Uz to, opisat će se i samo korištenje Pythona s bazom podataka, opisati i usporediti najpopularniji programski okviri za razvoj Weba u Pythonu te će se objasniti osnovni principi programskog jezika Python. Nadalje, usporedit će se programski jezik Python s ostalim programskim jezicima koji služe programiranju na strani poslužitelja. Na kraju će se opisati funkcionalnosti same web aplikacije.

2. Metode i tehnike rada

U razvoju web aplikacije u sklopu izrade ovog rada, korišteno je nekoliko alata, biblioteka te programskih jezika. U ovom poglavlju opisat će se korištene tehnologije u izradi web aplikacije.

2.1. HTML

HTML (engl. *Hypertext Markup Language*) je prezentacijski jezik za izradu web aplikacija. HTML ne smatra se programskim jezikom jer se za njegovo korištenje ne primjenjuje nikakva logika. Služi za oblikovanje web stranica i aplikacija, preko njega opisujemo računalu gdje se koji element mora nalaziti na zaslonu. Trenutna verzija HTML-a koja se koristi je verzija 5. Komunikacija između Pythona i HTML-a omogućena je jezicima za predloške što će biti opisano kasnije u radu.

2.2. CSS

CSS (engl. *Cascading Style Sheets*) ili jezik kaskadnog oblikovanja služi za dizajniranje i opisivanje prezentacije HTML-a. Također, omogućuje responzivan ili odazivan dizajn kako bi korisničko iskustvo bilo dobro i na manjim ekranima. Primjerice, preko CSS-a možemo mijenjati font, boje ili odabrati veće razmake između elemenata. Trenutno se koristi treća verzija CSS-a. U HTML-u se kroz oznaku <style> može ručno dodati CSS u HTML datoteku. No, puno bolji način od toga je zasebna CSS datoteka koju uključimo u HTML. Danas je sve popularniji i modularni pristup CSS dizajniranju pa se i same CSS datoteke razdvajaju u zasebne dijelove. CSS ima i svoj okvire koji se koriste za kreiranje dizajna. Neki od najpoznatijih su Bootstrap, Skeleton i Semantic. Najkorišteniji je ipak Bootsrap. Bootstrap je vrlo popularan jer uvelike olakšava programeru implementaciju responzivnosti, ali i olakšava dizajniranje weba jer programer ne mora pisati svoj CSS. U radu je korišten okvir Bootstrap za uređivanje HTML-a.

2.3. JavaScript

JavaScript je programski jezik kojim možemo manipulirati HTML elementima. Proširuje se na HTML i omogućuje interaktivnost. On manipulira HTML-om preko DOM-a. Objektni model dokumenta (engl. DOM – *Document Object Mode*l) je aplikacijsko programsko sučelje za strukturirane dokumente kao što je HTML.

Dokument i elemente dokumenta, DOM uvijek definira kao objekte, a programsko sučelje tad čine određena svojstva samog objekta i metode nad tim objektima. Takvim objektima se onda preko programskoga sučelja može dinamično pristupati, ažurirati sadržaj i izgled, mijenjati strukturu kompletnog dokumenta, neovisno o programskom jeziku [1]. Osim DOM-a postoji i BOM (engl. *Browser Object Model*). BOM omogućuje JavaScriptovu komunikaciju s preglednikom. Osim razvoja web aplikacija i stranica, JavaScript se danas koristi i za izradu mobilnih aplikacija, igara i mrežnih aplikacija. U pregledniku, JavaScript se pokreće preko tzv. JavaScript Engine-a. Npr., u Mozilli se taj engine zove SpiderMonkey, a u Google Chromeu v8. JavaScript posjeduje i brojne programske okvire. Najpoznatiji su jQuery, Angular, React i Vue.js. U ovome radu radit će se bez JavaScript okvira. JavaScript omogućuje i rukovanje podacima u realnom vremenu, bez osvježavanja stranice, a za to se koristi AJAX. U radu se JavaScript koristi za dinamičko upravljanje elementima. JavaScript se koristio i kao dio biblioteke Chart.js. Chart.js je besplatna JavaScript biblioteka otvorenog koda koja služi za vizualizaciju podataka u obliku grafikona. Uz to, JavaScript je korišten i kao dio sučelja Fetch API. Fetch API je sučelje za dohvat resursa (engl. *Resources*) [2]. Korišten je za dohvat podataka iz vanjskog API-ja te za generiranje predloška s popisom proizvoda.

2.4. MongoDB i Atlas

MongoDB je sustav za upravljanje nerelacijskom noSQL bazom podataka. On je dokumentno baziran, odnosno svoje podatke sprema u dokumente koji izgledaju poput JSON objekata [3]. Također, MongoDB je distribuirani sustav, a to su sustavi koji sadrže više različitih komponenti na različitim računalima koje komuniciraju i međusobno rade [4]. To MongoDB-u daje nekoliko prednosti. Prva je tolerancija na kvarove, koja je ugrađena u MongoDB. Pošto se podaci drže na nekoliko različitih poslužitelja, greška na jednom poslužitelju neće utjecati na rad aplikacije. Druga prednost je skalabilnost. Kako količina podataka raste, MongoDB dodaje više poslužitelja i performanse sustava ostaju na istoj razini. Uz to, spremanje podataka na više mjesta, najbolji je način za okruženje u oblaku (engl. *Cloud*). Treća prednost je ta što možemo podatke držati blizu svojih korisnika za brzo dohvaćanje podataka jer MongoDB ima glavne poslužitelje širom svijeta [5]. Nedostatak korištenja MongoDB je nepostojanje relacijske sheme. U radu se MongoDB koristi kao sustav za upravljanje bazom podataka i u njegovoj bazi pohranjeni su svi podaci aplikacije. Uz MongoDB dolazi i njegov vizualni alat koji nam pomaže pri vizualizaciji podataka, kao i lakšem pretraživanju, dodavanju, ažuriranju ili brisanju podataka.

2.5. GitHub

Za izradu aplikacije u sklopu ovog rada, kao sustav za verzioniziranje koda bit će korišteni Git i GitHub. Aplikacija je dostupna na vlastitom repozitoriju gdje su i navedene svi koraci instaliranja same aplikacije. Aplikacija je izrađena na hrvatskom jeziku, a korišten je operativni sustav Windows 10. U repozitoriju se nalazi cijela aplikacija. Podaci za spajanje na bazu priloženi su u uputama za instalaciju same aplikacije. Aplikacija je dostupna na sljedećoj poveznici: https://github.com/vrbaaa/eOPG- . Za instalaciju na vlastito računalo, potrebno je imati instaliran git ili ručno skinuti kod aplikacije s GitHuba. Ako korisnik ima instaliran git pomoću argumenta *clone* se omogućuje preuzimanje izvornog koda. Nakon argumenta *clone* dolazi repozitorij koji se želi preuzeti. Primjer cijele naredbe nalazi se u nastavku: `$ git clone https://github.com/vrbaaa/eOPG-`

2.6. Ostali alati, biblioteke i programski jezici

Ovo poglavlje će ukratko opisati ostale korištene alate, biblioteke i programski jezici. U radu su još korišteni:

- **Python** je programski jezik koji će kao glavna tema ovog rada biti detaljno opisan u vlastitom poglavlju.
- **Flask** je Python okvir za web programiranje. Detaljnije o njemu bit će opisano kroz posebno poglavlje kao što će biti i Flaskova proširenja i manji paketi.
- **Jinja2** je jezik za predloške (engl. *Template language*) kojeg koristi Flask. Također će detaljno biti opisan u razradi rada.
- **Visual Studio Code** je besplatni uređivač koda. Nastao je od strane Microsofta, ali se može koristiti i na Linuxu te macOS-u. Kao i većina ostalih uređivača, pruža potporu za mogućnosti automatskog dovršavanja naredbe, podcrtavanja sintaksnih grešaka, osjenčavanja otvarajućih i zatvarajućih zagrada i sl.
- **Postman** je kolaboracijska platforma za razvoj API-ja. Postman pojednostavljuje razvoj API-ja. U sklopu rada korišten je za testiranje kreiranog web servisa.
- **Font Awasome** je biblioteka koja je namijenjena uređivanju fonta i/ili raznih ikonica u dokumentu kako bi se korisnicima na što ljepši i moderniji način prikazao sadržaj web aplikacije. Baziran je na CSS-u i LESS-u. LESS je dinamički stilski jezik koji proširuje CSS. Primjerice, omogućuje korištenje varijabli i funkcija. Može se izvršiti na strani klijenta i na strani poslužitelja. Font Awasome najčešće se koristi s Bootstrapom. U sklopu aplikacije ovog rada, korišten je za ikone na navigaciji.

- **ReCaptcha** je besplatni sigurnosni servis koji služi zaštiti aplikacija protiv robota. Najnovija verzija je reCAPTCHA 3, a u radu se koristila verzija 2.
- **Draw.io** je besplatni alat za modeliranje dijagrama, a u njemu je izrađen UML dijagram konceptualnog modela podataka.

3. Općenito o webu i web aplikacijama

Web aplikacije su programi koji se pokreću na poslužitelju (engl. *Server*). Za razliku od stolnih aplikacija (engl. *Desktop*), pokreću se preko Web preglednika [6].
Prednosti web aplikacija su:

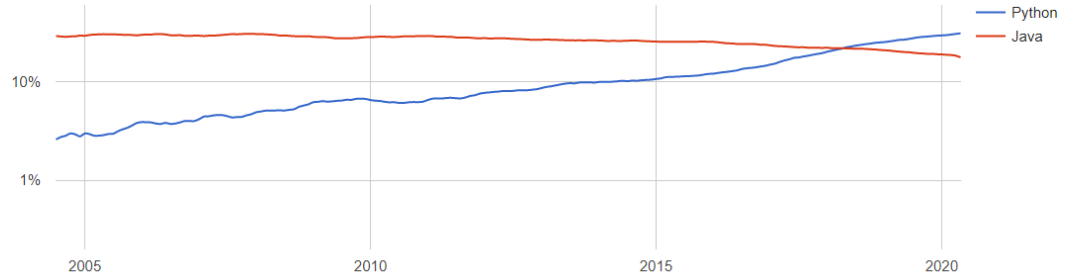
- Mogu se pokrenuti na svim uređajima koji imaju vezu na Internet te instaliran Internet preglednik (računalo, mobitel, pametni televizor, …)
- Mogu se pokrenuti na svim operacijskim sustavima
- Za njihovo pokretanje nije potrebna instalacija
- Svi korisnici koriste istu verziju aplikacije
- Manji troškovi popravaka i promjena na aplikacijama, samim time i kraće vrijeme potrebno za popravke i promjene
- Više prostora za spremanje podataka nego na lokalnom računalu

Neki od nedostataka web aplikacija su:

- Potrebna je Internetska veza za korištenje aplikacije
- Aplikacija je dostupna u gotovo cijelom svijetu što dovodi do sigurnosnih problema i mogućnosti napada
- Web aplikacije mogu biti sporije od stolnih aplikacija jer ovise o poslužitelju i brzini internetske veze korisnika
- Ovisnost o poslužitelju – osim što aplikacija može biti sporija, ako se dogodi greška na poslužitelju, aplikacija može postati nedostupna
- Potpora za preglednike – web aplikacije može koristiti puno korisnika, što znači i da će ih koristiti na različitim preglednicima i verzijama preglednika pa programer mora voditi računa o tome da njegova aplikacija radi na većini preglednika

3.1. Python kao programski jezik u profesionalnom svijetu

U ovom potpoglavlju, pisat će se o korištenju Pythona u praksi. Prema PYPL-u, online alatu koji analizira koliko se puta neki programski jezik pojavio u Google-ovoj tražilici, Python je pretekao Javu po popularnosti odnosno frekvenciji pretraživanja što vidimo na slici 1 [7].

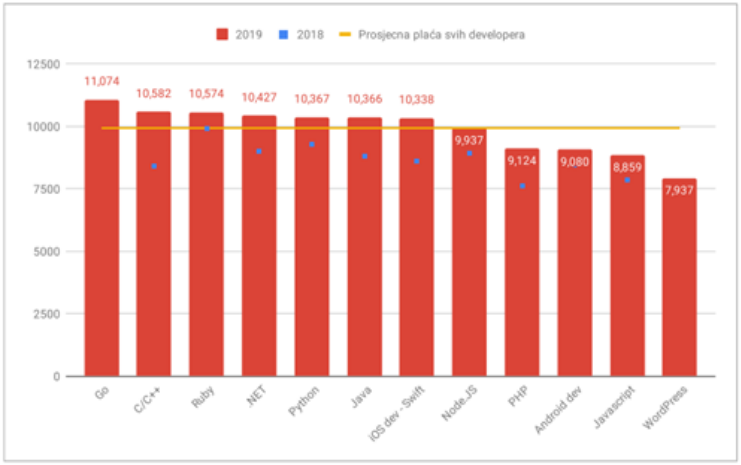


Slika 1: Popularnost programskih jezika od 2005. do danas [7]

Na slici broj 1 prikazuje se statistika upisivanja pojmova „Python“ i „Java“ u tražilicu. Treba napomenuti kako postoji 5 razina razvojnih programera. Svaka razina ovisi o iskustvu i znanju te se godine iskustva dane ispod mogu smanjivati ili povećavati, ovisno o znanju pojedinca. Također, plaća raste kako se prelazi u veću razinu. Razine su:

- Student programer – 0-1 godine iskustva
- Junior programer – 1-3 godine iskustva
- Middle programer – 3-7 godina iskustva
- Senior programer – 7+ godina iskustva
- Lead – 10+ godina iskustva

Prema istraživanju Tomislava Grubišića za 2019. godinu, prosječna neto plaća Python programera u Hrvatskoj je 10 367 kuna u odnosu na ostale programere drugih programskih jezika i platformi, što je prikazano na slici 2 [8].



Slika 2: Prosječna plaća svih programera u Hrvatskoj [8]

Na slici broj 2 prikazane su plaće svih razvojnih programera po svim tehnologijama, a vidimo da je Python peta najplaćenija tehnologija. Vidljivo je i kako je plaća Python programera za 2019. godinu porasla za oko 5 % u odnosu na 2018. godinu.

4. Python

Python je interaktivni i objektno orijentirani programski jezik nastao 1991. godine od strane Guida van Rossuma. Python svoju snagu i brzinu kombinira s vrlo jasnom i jednostavnom sintaksom što ga svrstava u sam vrh najkorištenijih programskih jezika. On spada u interpretersku skupinu programskih jezika, što znači da prevodi naredbe u trenutku izvođenja samog programa (prevodi liniju po liniju). Neki od ostalih interpreteskih jezika su PHP, Javascript i Ruby. Uz interpretere, postoje i kompilatori (engl. *Compilers*) koji prevode cijeli izvorni u strojni kod odjednom. Neki od kompilatora su C++, C# i Java. Također, Python spada i u skriptne jezike. Skriptni jezici su svi oni jezici koji se mogu ugraditi u HTML. Popularnost Pythona je toliko velika i zbog toga što se koristi u razne svrhe poput upravljanja podacima i bazama podataka, strojno učenje (engl. *Machine Learning*), izradu web stranica i aplikacija, programiranje informacijskih sustava, matematička (3D modeliranje, grafovi, projekcije) i znanstvena istraživanja. Trenutna posljednja verzija Pythona je 3.8.5, dok je u radu korištena verzija 3.8.2. Python 3 nastao je 2008. godine kao nasljednik Pythona 2, odnosno njegove posljednje verzije 2.7. Glavni razlog nastanka Pythona 3 jest rješavanje problema koji su postojali u Pythonu 2. Neke ključne razlike ovih dvaju verzija su te da je sintaksa treće verzije Pythona lakše razumljiva od sintakse Pythona 2. Također, Python 3 posjeduje mogućnost dijeljenja dva cijela broja, a da oni daju decimalni broj. Uz to, naredba *print* u trećoj verziji Pythona koristi obavezne zagrade [9]. Kako bi se koristo Python, potrebno ga je instalirati na vlastito računalo. To se može napraviti tako da se skine Python sa službene stranice https://www.python.org/downloads/. Nakon što je Python uspješno instaliran, potrebno je postaviti putanju na kojoj je instaliran Python u *$PATH* sustavnu varijablu da se Pythonu može pristupiti sa cijelog računala. Python dolazi s pip-om. Pip (engl. *Package installer for Python*) jest alat kojim možemo jednostavno instalirati pakete u PyPI (engl*. Python Package Index*). PyPI jest softverski repozitorij za Python koji pomaže pri pronalaženju i instalaciji programa stvorenih od Python zajednice [10]. Pip je jako važan jer njime instaliramo pakete, a vanjski paketi su jedan od razloga zašto je Python jedan od najpopularnijih programskih jezika današnjice. Zbog paketa, Python programer ne mora se zamarati oko funkcionalnosti koje su već rješenje jer one već postoje i može samo uvesti paket.

4.1. Virtualno okruženje

Python dolazi sa samo jednim okruženjem. To znači da ako instaliramo paket za jedan Python program, on će nam se instalirati globalno i svi će programi koristiti taj paket. To može dovesti do problema jer možda imamo program koji zahtijeva verziju 1.0 nekoga paketa, a mi smo tu verziju „prebrisali“ ponovnim pokretanjem *pip install-a*. Tada aplikacija neće raditi. Rješenje za taj problem jest instaliranje virtualnog okruženja za svaku Python aplikaciju. Virtualno okruženje je samoodržavajući direktorij koji sadrži instalaciju verzije Pythona, kao i pakete. Kako bismo kreirali virtualno okruženje, koristimo naredbu *python -m venv proizvoljnoIme-env*. Time kreiramo *proizvoljnoIme-env* direktorij ako ne postoji. Sada se još virtualno okruženje mora aktivirati. Na operacijskom sustavu Windows se to čini naredbom *proizvoljnoIme-env\Scripts\activate.bat*. Ako želimo pokazati svoj kod i rad na nekoj aplikaciji, korisnici koji će pokrenuti aplikaciju morat će instalirati sve pakete koje smo koristili. Naredbom *pip freeze > requirements.txt* stvaramo .txt datoteku u kojoj se nalaze svi paketi koje smo koristili. Kada se aplikacija želi pokrenuti na drugome računalu, samo se pokrene naredba *pip install -r requirements.txt* i svi paketi su instalirani te je aplikacija spremna za pokretanje.

4.2. Sintaksa Pythona

Glavna značajka Pythonove sintakse, u usporedbi s ostalim programskim jezicima je ta što Python za kraj naredbe ne koristi razdjelnik (engl. *Delimiter*), odnosno znak ';'. Nadalje, Python ne koristi vitičaste zagrade kod iteracija ili selekcija, međutim u tim programskim blokovima obavezno je uvlačenje redaka (indentiranje). Također, moguće je odraditi kompleksne operacije u jednoj naredbi. Uz to, takav pristup olakšava princip čistoga koda, a zbog toga su programi u Pythonu obično puno kraći nego oni napisani u C++-u ili Javi. Pisanje više naredbi u jednom redu u Pythonu omogućeno je tako da se naredbe razdvajaju razdjelnikom. Također, moguće je koristiti ugrađenu metodu ili funkciju *exec()* i linije koda razdvajati znakovima '\n', a indentiranje znakovima '\t'. Ipak, radi očuvanja principa čistoga koda, više naredbi u jednoj liniji u Pythonu se rijetko koristi. Seljedeće poglavlje će opisati sintaksne greške, komentare, varijable, tipove podataka, operatore, strukture podataka, grananja, petlje, funkcije, module, pakete i klase [11].

4.2.1. Sintaksne greške

Što se tiče sintaksnih grešaka (engl. *Syntax error*) u Pythonu postoji nekoliko vrsta takvih greški, a najčešće sintaksne greške su:

- IndexError – pojavljuje se kad se pokuša dohvatiti indeks koji ne postoji
- KeyError – pojavljuje se kad se pokuša dohvatiti ključ rječnika koji ne postoji
- ImportError – pojavljuje se kad neka funkcija ili modul ne može biti pronađen
- TypeError – pojavljuje se kad je neka operacija ili funkcija primijenjena na objekt krivoga tipa (npr. zbrajanje znakovnog tipa i broja)
- ValueError – pojavljuje se kad je argument funkcije krivoga tipa
- NameError – pojavljuje se kad pozivamo objekt koji nije definiran
- IndentationError - pojavljuje se kod neispravnog uvlačenja redaka

Ukoliko u programskom kodu postoji sintaksna greška, Python program neće se pokrenuti već će programera obavijestiti na kojoj liniji koda se greška nalazi.

4.2.2. Komentari

Komentari u Pythonu počinju znakom #. Komentari u više redaka počinju i završavaju s tri dvostruka navodnika (“““). Primjeri:

```
#Ovo je komentar u jednome retku

""" Ovo je komentar
U više
Redaka """
```

Važno je naglasiti kako Python ne dozvoljava ugniježđene komentare.

4.2.3. Varijable

Varijable su dijelovi programa za spremanje podataka. Za razliku od drugih programskih jezika, u Pythonu ne postoji naredba za deklariranje varijable, već se ona kreira u trenutku kad je varijabli dodijeljena neka vrijednost. Varijable u Pythonu nisu istog tipa. Na primjer, varijablu x na početku programa možemo deklarirati kao numerički tip podataka (integer), i kasnije promijeniti u znakovni tip (string) i nećemo dobiti grešku (TypeError). Što se tiče imenovanja varijabli, tu vrijede slična pravila kao u većini programskih jezika. Pravila su da ime varijable mora početi slovom ili donjom crtom, ne može početi brojkom, u sebi ne smije sadržavati posebne znakove već samo slova, brojke i donje crte te osjetljivost na mala i velika slova (broj, Broj i BROJ su tri različite varijable).

U Pythonu je moguće inicijaliziranje više varijabli u jednome retku, a isto tako moguće je u jednome retku inicijalizirati više varijabli na istu vrijednost. Primjeri:

```
a, b, c  = 'Varijabla A', 'Varijabla B' , 'Varijabla C'
a = b = c = 'Sve varijable imaju istu vrijednost'
```

Prvi redak primjera prikazuje inicijalizaciju tri varijable s različitim vrijednostima. Zarezom odvajamo imena varijabli, a znakom „=“ pridružujemo vrijednosti. U drugom retku primjera znakom „=“ pridružujemo jednaku vrijednost svim varijablama.

4.2.4. Tipovi podataka

U Pythonu postoji mnogo različitih tipova podataka. Već je prije spomenuto kako je svaka varijabla nekog tipa. Da bismo odredili kojeg je varijabla tipa, koristimo naredbu type(). Sada ćemo i nabrojiti tipove podataka: *str, int, float, complex, list, tuple, range, dict, set, frozenset, bool, bytes, bytearray, memoryview*. *Complex*, *int* i *float* jesu brojevni tipovi podataka. *Tuple, dict, set, frozenset* i *list* jesu skupovi raznih tipova podataka. Str je znakovni ili tekstualni tip podataka. *Bool* je tip podataka koji vraća True ili False. *Bytes, bytearray* i *memoryview* su binarni tipovi podataka.

4.2.5. Operatori

Operatori u Pythonu dijele se na:

- Aritmetički operatori (+, -, *, /, %, **, //)
- Operatori pridruživanja (=, +=, -=, *=, %=, //=, **=, &=, |=, ^=, >>=, <<=)
- Operatori uspoređivanja (==, !=. <, >, >=, <=)
- Logički operatori (*and, or, not*)
- Operatori identiteta (*is, is not*)
- Operatori pripadnosti (*in, not in*)
- Bitni operatori (&, |, ^, ~, <<, >>)

Aritmetički operatori i operatori pridruživanja najčešće se koriste kod izrade matematičkih funkcija. Operatori uspoređivanja i logički operatori koriste se kod ispitivanja uvjeta. Operatori identiteta koriste se za usporedbu objekata, ne za provjeru jesu li jednaki već jesu li baš doslovno isti objekt. Različito je od usporedbe znakom „==“ jer „==“ provjerava samo sadržaj. Operatori pripadnosti provjeravaju nalazi li se nešto u objektu. Primjerice, ako želimo provjeriti nalazi li se neko slovo u nizu znakova, to možemo napraviti operatorom „in“. Također ga možemo koristiti i za kretanje po iteracijama. Bitni operatori vrše operacije među bitovima. Služe za usporedbu binarnih brojeva.

4.2.6. Strukture podataka

U Pythonu postoje četiri vrste strukture podataka: liste, n-torke, skupovi i rječnici.

4.2.6.1. Liste

Liste (engl. *List*) su strukture podataka koje su promjenjive, uređene i dozvoljavaju duplikate. One su mjesta u kojima skupljamo ostale objekte kako bismo ih mogli tretirati kao grupe. Liste su vrlo moćna struktura podataka, koji bi se u nekim drugim jezicima niže razine poput C-a, morale napraviti ručno. Za inicijalizaciju listi Pythonu koristimo uglate zagrade ili ključnu riječ *list*.

```
listaProgramskihJezika = ['PHP', 'Ruby', 'Python']
listaProgramskihJezika = list(('PHP', 'Ruby', 'Python'))
```

U kodu iznad vidimo kako se na dva načina može inicijalizirati lista. Za dohvat nekog podatka iz liste, potrebno se referencirati na broj indeksa i uglatom zagradom, a indeksi kreću kao i u ostalim programskim jezicima od nule. Za iteriranje kroz listu, koristi se naredba *for in*, a za provjeravanje postoji li podatak u listi naredba *if in*. U nastavku će biti prikazan primjeri rada s listom.

```
listaProgramskihJezika = ['PHP', 'Ruby', 'Python']
for programskiJezik in listaProgramskihJezika:
      print(programskiJezik)
if 'PHP' in listaProgramskihJezika:
      print('PHP se nalazi u listi programski jezika')
```

Dodavanje nove vrijednosti u listu možemo napraviti metodama *append* i *insert*, a razlika među njima je ta što kod naredbe *append* nova vrijednost uvijek završi na kraju liste, dok kod naredbe *insert* možemo odabrati i indeks nove vrijednosti u listi. Spajanje dvije liste može se odraditi na više načina. Za to postoje naredbe *append, extend* i artimetički operator +. Od ostalih metoda u listama bitna je i *sort* metoda preko koje možemo sortirati listu. Liste su jedne od najkorištenijih tipova podataka u Pythonu pa su zato ovdje toliko detaljno opisane.

4.2.6.2. N-torke

N-torke (engl. *Tuples*) su strukture podataka koji su određeni, nepromjenjivi i dopuštaju duplikate. Incijaliziraju se oblim zagradama ili ključnom riječi *tuple*. N-torke su isto što i liste samo se njihov sadržaj ne može promijeniti.

Nepromjenljivost n-torki pruža neku sigurnost programeru da će podaci incijalizirani u torki zauvijek ostati nepromijenjeni. Ako ipak želimo dodati nove članove u n-torku potrebno je n-torku pretvoriti u listu. Dodamo vrijednost u listu i ponovno vratimo u n-torku.

```
torkaProgramskihJezika = ['PHP', 'Ruby', 'Python']
listaProgramskihJezika = list(torkaProgramskihJezika)
listaProgramskihJezika[2] = 'C#'
torkaProgramskihJezika = tuple(listaProgramskihJezika)
```

U programskom kodu iznad dan je prikaz inicijaliziranja n-torke. Zatim promjena n-torke u listu. Promjena druge vrijednosti u listi i ponovno pretvorba u n-torku.

4.2.6.3. Skupovi

Skupovi (engl. *Set*) su strukture podataka koji su neodređeni i neindeksirani. Incijaliziraju se vitičastim zagradama ili ključnom riječi set. Za dodavanje nove vrijednosti u skup koristimo naredbu *add*. U skupovima ne smiju postojati jednake vrijednosti. Rad sa skupovima možemo vidjeti u sljedećem primjeru.

```
my_set = {1, 2, 3, 4, 3, 2}
my_set.add(5)
print(my_set)
```

Primjer iznad ispisat će vrijednosti bez duplikata, odnosno ispisati „{1, 2, 3, 4, 5}“.

4.2.6.4. Rječnici

Rječnici (engl. *Dictionaries*) su strukture podataka koji su neodređeni, promjenjivi i indeksirani. Neodređeni su zato jer im ne pristupamo prema principu „lijevo prema desno“. Pristupamo im vitičastim zagradama ili ključnom riječi *dict*. U nekim drugim jezicima, oni se nazivaju asocijativnim poljima ili asocijativnim nizom. Riječnici imaju ključ i vrijednost pri čemu je važno napomenuti da ključevi moraju biti jedinstveni. Rječnici su odlična struktura podataka za brzo pretraživanje. Neke od najbitnijih ugrađenih metoda koje možemo koristiti u rječnicima su *get* preko koje dohvaćamo vrijednost za zadani ključ. Ispod je prikazano inicijaliziranje jednoga rječnika.

```
Proizvod = {"naziv": "Luk","tip": "povrće","ukupno": 1300}
```

U primjeru su rječniku naziva „Proizvod“ dodani ključevi „naziv“, „tip“ i „ukupno“. Ključevima su dodijeljene vrijednosti kojima se kasnije može pristupiti.

4.2.7. Grananja i petlje

Kao i u svakom programskom jeziku, grananja i petlje (iteracije) jedni su od najvažnijih programskih konstrukata.

4.2.7.1. *If else* instrukcija

If instrukcija u Pythonu radi ono što i u ostalim programskim jezicima, a to je ako je uvjet koji se nalazi poslije ključne riječi ispunjen, izvodi dio koda koji se nalazi ispod *if* instrukcije. Ako uvjet nije ispunjen, program se izvodi dalje. Moguće je koristiti i *else* instrukciju, a ona znači ako uvjet iz *if* instrukcije nije ispunjen, onda izvrši naredbe koje su ispod *else* instrukcije. Također, postoji i *elif* instrukcija (u većini ostalih programskih jezika *else* if). Dio koda nakon elif instrukcije izvršava se ako uvjet iz if instrukcije nije ispunjen, a uvjet iz *elif* instrukcije jest ispunjen. U uvjetima možemo koristiti logičke operatore *and*, *or* i *not*, kao i operatore uspoređivanja (>, <, ==, itd.). Zagrade kod uvjeta su opcionalne, ali se u pravilu ne koriste. Ispod je prikazan primjer jedne *if else* instrukcije.

```
if ocjena > 1.5:
   session['korisnicko_ime'] =  form.korisnicko_ime.data
   session['type'] = login_user['type']
   upisiUDnevnik("Prijava u sustav")
   flash('Prijavljeni ste.', 'success')
   return redirect(url_for('users.index'))
else:
   flash('Prijava neuspješna. Imate premalu ocjenu', 'danger')
```

Ovaj kod odradit će prijavu korisnika u sustav ako je vrijednost varijable ocjena veća od 5. Inače, prikazat će se poruka da se korisnik ne može prijaviti u sustav. Kod *if* instrukcije, svakako mora biti spomenut i ternarni operator. On je odličan izbor za jednostavnije *if* uvjete kao u sljedećem primjeru.

```
if x:
   a = 'bok'
else:
   a = 'sok'

a = 'bok' if x else 'sok'
```

Vidimo da ternarnim operatorom možemo 4 linije koda pretvoriti u jednu liniju. Time poboljšavamo čitljivost koda. Nedostatak ternarnih operatora je taj što otežavaju čitanje velikih blokova koda.

4.2.7.2. While petlja

While petlja je najgeneralnija konstrukcija iteracije u Pythonu. Samom funkcionalnošću ne razlikuje se od korištenja iste konstrukcije u ostalim programskim jezicima. Uz to, u *while* petlji postoje i naredbe *break*, *continue*, *pass*. Break naredbom izlazimo iz petlje u kojoj je pozvana naredba. Naredbom *continue* vraćamo se na vrh petlje u kojoj je pozvana ta naredba. *Pass* naredba ne radi ništa, ona doslovno govori programu da preskoči tu liniju. Primjer jedne *while* petlje gdje se provjerava je li broj prost.

```
x = y // 2
while x > 1:
      if y % x == 0:
            print(y, 'ima djelitelja', x)
            break
      x -= 1
      else:
            print(y, 'je prost')
```

Valja napomenuti kako *do while* petlja kao takva u Pythonu ne postoji, već se može ručno isprogramirati.

4.2.7.3. For petlja

For petlja u Pythonu se ipak malo razlikuje od iste naredbe u drugim programskim jezicima. Glavna razlika je ta što u Pythonu u *for* petlji možemo direktno ići od znaka do znaka u znakovnom tipu ili od indeksa do indeksa u polju. Za *for* petlju u Pythonu nisu nam potrebni brojači, iako se mogu koristiti. Na primjer, sljedeća će *for* petlja ispisati znak po znak u znakovnom tipu.

```
for znak in 'Znak':
   print(znak)
```

Ovaj kod iznad ispisat će 'Z, 'n', 'a', i 'k', a svako će slovo biti ispisano u novome redu.

4.2.8. Funkcije

Funkcije su jedan od najkorištenijih elemenata Pythona. Definiraju se ključnom riječju *def*. Nakon ključne riječi def dolazi samo ime funkcije, koje treba početi malim slovom. Nakon naziva funkcije potrebno je staviti okrugle zagrade i u njima, ako se koriste, argumente. Ispod je dan primjer funkcije koja zbraja dva broja.

```
Def zbroji(a, b):
   Return a + b
```

Naravno, funkcije ne moraju uopće primati parametre niti vratiti ikakvu vrijednost. Postoje i tzv. „pass“ funkcije koje nemaju svrhu već služe programeru za kasnije uređivanje funkcije. Postoje i anonimne (lambda funkcije). Lambda funkcije primaju bezbroj parametara, ali mogu imati samo jedan izraz. Primjer korištenja jedne lambda funkcije dan je u nastavku:

```
def lamb(n):
  return lambda a : a + n
povecajZa2 = lamb(2)
povecajZa3 = lamb(3)
print(povecajZa2(10))
print(povecajZa3(10))
```

Ovaj kod ispisat će „12“ i „13“.

4.2.9. Moduli i paketi

U Pythonu, moduli pružaju jednostavan način organiziranja komponenti kao samostalnih paketa što još nazivamo prostor imena (engl. *Namespaces*). Korištenje modula bitno je zbog ponovnog upotrjebljavanja koda. Da bismo uvezli neki modul u Python, dovoljno je znati ime tog modula, a ako se on nalazi izvan prostora u kojem ga koristimo, i putanju do tog modula. Modul se uvozi ključnom riječi *import*. Ako ne želimo koristiti cijeli modul, već samo neku funkcionalnost iz njega, to činimo naredbom *from modul import funkcionalnost.* Kad se modul uveze i kad pokrenemo program Python prvo mora pronaći taj modul, a programer da uveze modul mora znati samo njegovo ime jer sintaksa Pythona zabranjuje uvoz kompletne putanje do modula. Zatim se kod iz modula pretvara u strojni kod koji se nakon toga pokreće. Ako Python interpreter zapne na nekom od ova tri koraka, program neće raditi. Za korištenje modula, važno je i kako odrediti koji je modul većeg prioriteta. Prije nego što se pokrene Python program, Python sam generira nekoliko varijabli. Jedna od njih je i varijabla *__name__* koju Python postavlja na *'__main__*' ako se nalazimo u prvome modulu. Ako jednu Python skriptu

uvezemo u drugu, varijabla *__name__* ima vrijednost imena prvoga modula ili skripte. To nam je jako važan mehanizam jer pomoću instrukcije *If __name__ == '__main__':* možemo zaključiti pokreće li se naša skripta direktno od strane glavne skripte ili je uvezena kroz modul. Time možemo ograničiti izvođenje dijela koda koji je napisan nakon instrukcije, samo na tu skriptu u kojoj je kod napisan. Ispod je dan primjer.

```
if __name__ == '__main__':
    io.run(app)
```

Međutim, u izradi aplikacije ovakav način komunikacije između modula nije korišten, već je gore navedena if instrukcija korištena samo kod pokretanja aplikacije.

4.2.10. Klase u Pythonu

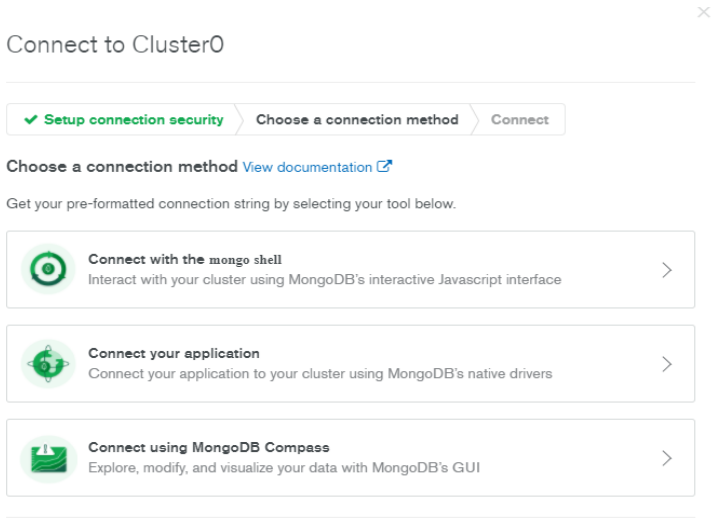
Klase u Pythonu definiraju se ključnom riječi *class*. Klasa se obično sastoji od konstruktora, atributa i metoda. U Pythonu, konstruktor je uvijek automatski pozvan kad je instanciran novi objekt klase. Konstruktor mora imati posebno ime *__init__()* i posebni parametar *self*. Svi članovi klase u Pythonu odmah su dostupni iz cijelog koda (engl. *Public*). Zaštićeni (engl. *Protected*) članovi klase dostupni su samo unutar klase i u podklasama. Zaštićeni članovi klase u Pythonu se označavaju jednom donjom crticom _. Privatni članovi (engl. *Private*) klase označavaju se s dvije donje crtice __. S obzirom da se kod izrade aplikacije u sklopu rada nisu koristile klase, osim kod korištenja formi, dan je primjer definiranja jedne klase koja ima svoje atribute i metodu, ali nema definiran konstruktor.

```
class AccountForm(FlaskForm):
 korisnicko_ime = StringField('Korisničko ime',validators=[DataRequired()])
   type = RadioField('Vrsta',  validators=[DataRequired()],
   choices=[('centar','Centar'), ('opg', 'Opg')])
   email = StringField('E-mail', validators=[DataRequired(), Email()])
   oib = StringField('OIB', validators=[DataRequired(), Length(min=13,
   max=13)])
   adresa = FormField(AdressForm)
   submit = SubmitField('Napravi korisnika')
   def validate_korisnicko_ime(self, korisnicko_ime):
       user = mongo.db.korisnici.find_one({"korisnicko_ime":
korisnicko_ime.data})
       if user:
          raise ValidationError("Korisničko ime već postoji")
```

Na primjeru je prikazano definiranje klase za kreiranje forme. U metodi „validate_korisnicko_ime“ provjeravamo postoji li već uneseno korisničko ime u sustavu.

4.3. Python i MongoDB

Python može raditi s relacijskim, nerelacijskim i graf bazama podataka. U ovome potpoglavlju, opisat će se kako Python komunicira s nerelacijskom bazom podataka MongoDB. Veliki plus rada u Pythonu, a pogotovo u Flasku je taj što je neovisan o izvoru podataka kojima programi napisani u Flasku manipuliraju. To znači da Flask dozvoljava nekoliko tipova datoteka za pohranu podataka poput .csv ili .txt datoteka. Flask nam omogućuje da radimo s onim podacima kojima nama odgovaraju i tu programer ima svu slobodu. U ovome radu koristi se MongoDB u oblaku i njegov alat MongoDB Atlas. Naravno, moguće je korištenje i lokalne baze podataka. Na slici je prikazano sučelje Atlasa za opciju *Connect*.



Slika 3: Opcija *Connect* u Atlasu (Izvor: autorski rad)

Pod opcijom *Connect* na slici 3 imamo tri opcije :

- *Connect with the mongo shell* – služi za interakciju s klasterom koristeći interaktivno sučelje
- *Connect your application* – služi za spajanje na aplikaciju u kojoj želimo koristiti podatke spremljene u bazi podataka
- *Connect using MongoDB Compass* – služi za pretragu, modificiranje i vizualizaciju podataka u alatu Compass. Compass je alat u sklopu MongoDB sustava koji analizira dokumente te prikazuje strukturu unutar kolekcija preko intuitivnoga grafičkoga sučelja [12].

Pri razvoju svake aplikacije, a pogotovo prilikom korištenja nekog sustava za upravljanje bazom podataka prvi put, poželjno je testirati upite izvan aplikacijskog okruženja. Razlog tome je taj što programer, da bi mogao koristiti podatke iz baze podataka, prvo mora znati u kakvom će se obliku vratiti podaci kad se izvrši upit.

Programer bi usto potrošio previše vremena postavljajući upite u samom kodu aplikacije. Iz tog razloga, za kreiranje upita u razvoju aplikacije korišteno je interaktivno sučelje u naredbenome retku. Da bismo mogli pisati upite u naredbenome retku, potrebno je instalirati mongo ljusku (engl. *Shell*). Za to odabiremo prvu opciju *Connect with the mongo shell*. Zatim instaliramo mongo ljusku te je dodajemo u $PATH varijablu kako bismo mogli pristupiti mongo ljuski iz bilo kojeg mjesta. Nakon toga se u naredbenom retku postavljamo na mjesto instalacije mongo ljuske i izvršavamo naredbu za povezivanje na bazu. Kad smo spojeni, možemo raditi upite nad bazom podataka. Na slici broj 4 vidimo primjer jednog jednostavnog upita u kojem dohvaćamo sve korisnike, a na slici je zbog preglednosti prikazan samo jedan korisnik.

Slika 4: Primjer upita u mongo ljusci (Izvor: autorski rad)

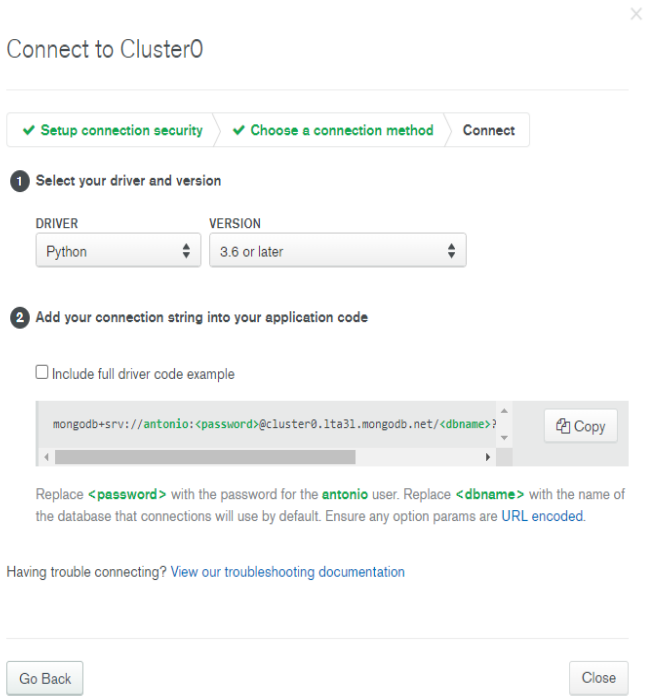
Programer iz rezultata upita na slici broj 4 može zaključiti kako je „adresa“ atribut tipa objekt, a „ocjene“ tipa polje objekata. U izradi aplikacije, korištena je biblioteka PyMongo. PyMongo jest Python biblioteka koja sadrži alate za rad s MongoDB-jem i preporučeni je način rada s MongoDB-jem u Pythonu [13]. Da bismo koristili PyMongo i MongoDB u razvoju Web aplikacije u Flasku, najprije je naravno potrebno uvesti samu biblioteku, a to se radi naredbom:

```
from flask_pymongo import PyMongo
```

Nakon toga, potrebno je postaviti vezu na bazu podataka. Pošto je u radu korištena baza podataka u oblaku, postavljen je URI (engl. *Uniform Resource Indentifier*) na lokaciju same baze podataka u konfiguracijsku varijablu aplikacije. Ovdje se u atribut *db* automatski sprema vrijednost imena baze podataka koji kasnije koristimo za pristup bazi odnosno kolekcijama u toj bazi. Ako vrijednost nije dodijeljena, tada je vrijednost atributa *mongo None*. To još znači da možemo dohvatiti sve baze podataka koje se nalaze na tom poslužitelju.

```
app.config['MONGO_URI']='mongodb+srv://KORISNIČKOIME:LOZINKA@cluster0.lt
a3l.mongodb.net/IME_BAZE?retryWrites=true&w=majority'
```

Za dobivanje podataka za povezivanje na bazu (engl. *Connection String*), moramo odabrati opciju *Connect your application*. Tada se pojavi prozor kojeg vidimo na slici 5 i u kojem biramo drajver (upravljački program) na kojem se želimo spojiti te verziju.

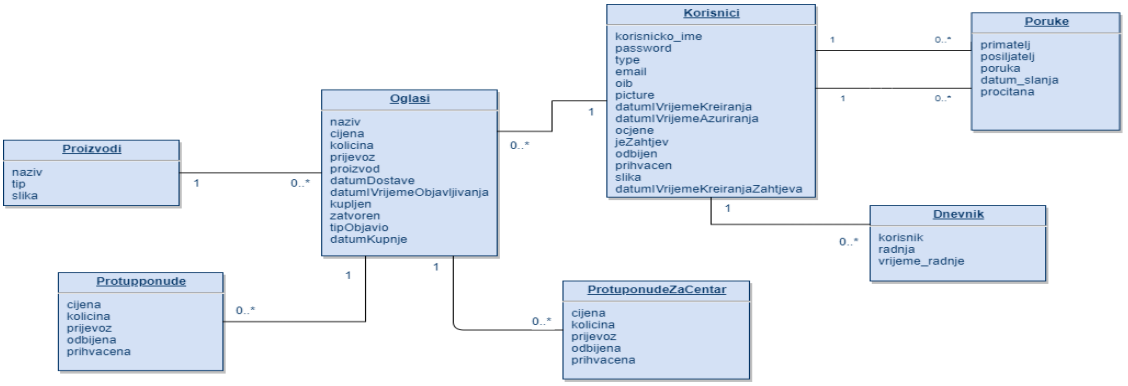


Slika 5: Prozor connect your application u Atlasu (Izvor: autorski rad)

Kad odaberemo željeni drajver, kao što vidimo na slici broj 5 dobivamo podatke za povezivanje na bazu. Za kraj, potrebno je samo pozvati PyMongo, koji se onda spaja na poslužitelj koji se nalazi u konfiguracijskoj varijabli. Vezu koristimo za dohvat kolekcija. Veza se sprema u varijablu:

```
mongo = PyMongo(app)
```

Dalje će se kroz primjere pokazati kako se u Pythonu radi s podacima iz MongoDB. Na slici 6 je prikazan UML (engl. *Unified Modelling Language*) dijagram konceptualnog modela podataka.



Slika 6: UML dijagram konceptualnog modela podataka (Izvor: autorski rad)

Na konceptualnom modelu podataka na slici 6 vidimo 7 kolekcija podataka. Iz dijagrama se vide veze između kolekcija. Jedan korisnik može poslati više poruka, dok poruka može biti poslana od samo jednog korisnika. Isto tako jedna poruka može biti poslana samo jednom korisniku, dok jedan korisnik može primiti 0 ili više poruka. Korisnik može raditi više radnji iz dnevnika, a jednu radnju u dnevniku radi samo jedan korisnik. Isto tako, korisnik može objaviti više oglasa, dok je jedan oglas objavljen od jednog korisnika. Što se tiče oglasa, oni sadržavaju točno jedan proizvod, dok se jedan proizvod može nalaziti na 0 ili više oglasa. Također, jedan oglas ima 0 ili više protuponuda ili protuponuda za centar, dok jedna protuponuda ili protuponuda za centar mogu biti od samo jednog oglasa. U kolekciji korisnici postoji zapis „ocjene“. Taj je zapis tipa *array* (polje objekata). Svaki objekt u polju sastoji se od svojih atributa, to su: „user_id“ koji označava koji je korisnik dao ocjenu te „rating“ koji predstavlja ocjenu od 1 do 5. Time se omogućuje zapis više ocjena za jednog korisnika. U kolekciji korisnici postoji zapis „adresa“. Taj je zapis tipa „Object“, što znači da zapis „adresa“ sadrži svoje „unutarnje“ zapise. Unutarnji zapisi u adresi su: „zupanija“, „opcina“, „mjesto“, „ulica“ i „kbr“. Detaljni opis kolekcija je:

- Oglasi – kolekcija koja sadrži sve oglase koji su objavljeni od strane korisnika (OPG ili centar)
- Poruke – sadrži poruke poslane preko sustava
- Proizvodi – sadrži sve proizvode koji se mogu koristiti u oglasima
- Protuponude – sadrži poslane protuponude
- ProtuponudeZaCentar - sadrži poslane protuponude
- Korisnici – kolekcija koja sadrži sve korisnike i podatke o njima
- Dnevnik – kolekcija koja sadrži podatke o korisničkim radnjama koje su dostupne administratoru

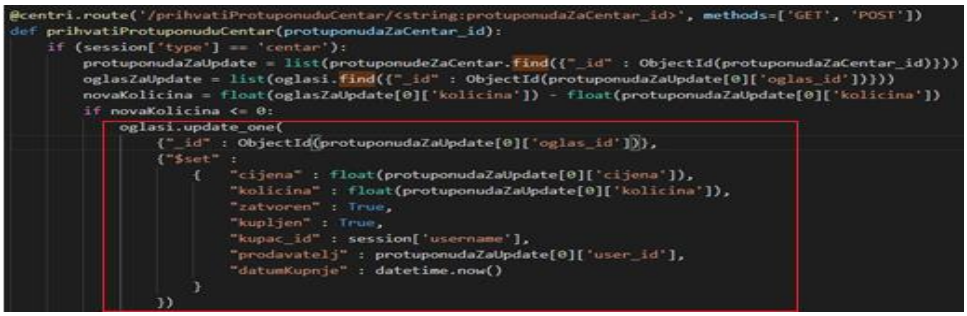
Primjer jednog jednostavnog upita, u kojem dohvaćamo sve korisnike koji nisu tipa administrator jest:

```
users = mongo.db.korisnici
korisnici = users.find({"type" : {"$ne" : "admin"}})
```

Primjer složenijeg upita u kojem dohvaćamo sve oglase OPG-ova koji nisu kupljeni i jednaki su unesenim parametrima. Treba naglasiti kako je zbog strukture nerelacijskih baza podataka, u istima rijetko pisanje ugniježđenih upita.

```
offersResults = oglasi.find({"kupljen" : False, "tipObjavio" : "opg",
"proizvod" : proizvod, "cijena" : {"$lte" : cijena}})
```

Na slici broj 7 dan je primjer ažuriranja kolekcije oglasi, nakon što je prihvaćena protuponuda OPG-a, a količina koja se nalazi u protuponudi je veća ili jednaka količini originalnoga oglasa. I ovdje, a i u kodu iznad, možemo vidjeti da ključne riječi u pisanju upita za MongoDB se moraju pisati unutar dvostrukih navodnika.



Slika 7: Primjer Mongo upita u Pythonu (Izvor: autorski rad)

Unos podataka u MongoDB bazu preko Pythona odvija se na sličan način kao i *UPDATE* ili *FIND*. U kolekciju *korisnici* unosimo podatke navedene u dvostrukim navodnicima.

```
korisnici.insert_one({
                "korisnicko_ime" : form.korisnicko_ime.data,
                "password" : form.password.data,
                "type" : form.type.data,
                "email" : form.email.data,
                "oib" : form.oib.data,
                "adresa" : form.adress.data,
                "datumIVrijemeKreiranja" : datetime.now(),
            })
```

Naravno, ovo nije jedini način na koji možemo koristiti MongoDB s Pythonom, postoji i pristup prema modelima kojim bi se kreirali posebni objekti pojedinog zapisa u kolekciji. Za to se koristi paket MongoEngine. Za veće aplikacije, bolje rješenje bilo bi koristiti modele kako bismo mogli pokušali napraviti nešto što je najsličnije MVC uzorku (engl. *Model View Controller*). U ovome radu je korištena biblioteka PyMongo. Razlog tome je što se želi zadržati sva sloboda pri rukovanju podacima koju pružaju nerelacijske baze podataka. Kod korištenja modela gubi se na slobodi i dokumenti su jednako organizirani što ipak više odgovara relacijskim bazama podataka. Nedostatak ovog pristupa jest pisanje Mongo upita u Pythonu kako bi se rukovalo podacima.

4.4. Python kao jezik za razvoj web aplikacija

U ovome radu, o Pythonu će se najviše pisati kao o jeziku za programiranje web aplikacija. Osim biblioteka, u Pythonu se za web programiranje koriste programski okviri (engl. *Framework*). Najprije, spomenut će se definicije biblioteka i programskih okvira. Biblioteka je skup datoteka, programa, skripti i funkcija koja može biti pozvana ili referencirana u programskom kodu [14]. Cilj korištenja biblioteke je smanjiti pisanje vlastitog koda. Programski okvir jest skup modula i paketa koji pomažu pri izgradnji web aplikacija. U radu s programskim okvirima, ne moramo brinuti o detaljima koji se događaju na nižoj razini poput protokola, utičnica (engl. *Socket*) ili upravljanja dretvama [15]. Okvire dijelimo na mikro programske okvire i full-stack programske okvire. U tablici broj 1 dana je usporedba između dvije vrste okvira.

Tablica 1. Usporedba vrsti okvira (Izvor: autorski rad)

Mikro okviri	**Full-stack okviri**
Jednostavni za korištenje	Kompleksni
Često korištenje	Često korištenje
Koriste se za izradu manjih i srednjih web aplikacija	Koriste se za izradu većih web aplikacija
Lagani i brzi za naučiti	Sporije učenje
Jednostavni za održavanje	Potrebno iskustvo za održavanje
Primjeri nekih okvira: Bottle, Flask, CherryPy, Dash	Primjeri nekih okvira: Web2Py, TurboGears, Django, Pyramid

Iz tablice 1 može se zaključiti kako su mikro okviri jednostavniji za učenje i prvo korištenje, ali full-stack okviri ipak su bolji izbor za izradu velikih projekata. Također, mikro okviri pružaju više slobode za izradu projekta i često se koriste za izradu mikro servisa.

4.5. Jezici za predloške u Pythonu

Kako bismo podatke generirane na poslužitelju mogli koristiti na klijentskoj strani, konkretno u HTML dokumentu, potrebno je te podatke na neki način poslati klijentu. Python se brine za slanje podataka klijentu, ali tim podacima je isto tako potrebno nekako pristupiti. Rješenje tog problema su jezici za predloške (engl. *Templating Languages*). Jezici za predloške omogućuju programerima generiranje željenog sadržaja koristeći neke programske konstrukte, poput selekcija ili petlji kako bi se manipuliralo sadržajem [16]. Predložak je tekstualna datoteka koja može biti u HTML, XML i sličnim formatima. Predložak posjeduje varijable koje se zamjenjuju vrijednostima jednom kad je predložak učitan. Također, posjeduje i oznake koji kontroliraju logiku u predlošku. Neki od jezika predložaka su: DTL, Jinja2 i Mako. Jinja2 je moderan jezik za predloške nastao na temelju DTL-a (Django Template Language). Neke od značajki Jinje2 su mogućnost nasljeđivanja predložaka, jednostavnost pronalaska greški i podesivost sintakse [17]. Detaljan rad s Jinjom2 u Pythonu opisan je u poglavlju Flask. Valja spomenuti i jezik za predloške Mako koji omogućuje programeru izravno pisanje Python koda u HTML dokumentu, što je u nekim situacijama dobro i podsjeća na korištenje PHP-a.

4.6. Okviri za Web programiranje u Pythonu

U ovome potpoglavlju bit će opisana tri okvira za web programiranje u Pythonu, a to su: Django, Tornado i Pyramid. Osim njih, postoji još tridesetak sličnih okvira, a od kojih su najpoznatiji Web2Py, TurboGears, CherryPy i Bottle. Django je odabran radi toga jer je, uz Flask koji se koristi u radu, daleko najkorišteniji. Pyramid i Tornado odabrani su jer imaju dosta različitosti od Flaska i Djanga.

4.6.1. Django

Django je okvir programskog jezika Python. Služi za brzi razvoj i čist te pragmatičan dizajn [18]. Trenutno je najpopularniji okvir za razvoj web aplikacija u Pythonu. Pomaže programeru da se maksimalno posveti razvoju Web aplikacija bez da se brine o nepotrebnim stvarima. Postoji nekoliko prednosti korištenja Djanga. Prva je ta što je brz, odnosno napravljen je da bi programerima omogućio izradu aplikacije što je prije moguće. Zatim, opremljen je desecima raznih dodataka kojima programer može koristiti za manipuliranje učestalim zadacima u razvoju za Web. Django se brine o autentifikaciji korisnika, administriranju sadržaja, RSS-u itd.

Nadalje, Django je veoma siguran i pomaže programeru da izbjegava česte sigurnosne pogreške poput SQL ubrizgavanja (engl. *SQL injection*), XSS (engl. *Cross-site scripting*), itd.

Također, izuzetno je prilagodljiv i može rukovati s Web mjestima koja imaju milijune posjetitelja, kao i s onima koje koriste deseci korisnika. Na kraju, vrlo je fleksibilan te ga koriste mnoge velike i male organizacije [18]. Django je prikladan okvir za razvoj Web aplikacija koje jako ovise o bazi podataka i njenoj strukturi. Django posjeduje svoj ORM (engl. *Object Relational Mapping*), a može raditi i automatsko generiranje formi bazirano na shemama. Jednom kad su definirani modeli, bogati Python API može biti korišten kako bi se pristupilo podacima. Uz to, Django nudi i dinamično administracijsko sučelje preko kojeg autentificirani korisnici mogu dodavati, mijenjati i brisati objekte. To omogućuje programeru da već na početku razvoja aplikacije može dodavati podatke i testirati dodane modele [19]. Django kao okvir za web programiranje koristi svoj jezik za predloške DTL, ali može koristiti i Jinju2.

4.6.2. Tornado

Tornado je okvir koji kombinira asinkrone mrežne biblioteke i Web okvira. Dobar je okvir za aplikacije koje će imati desetke tisuća korisnika istovremeno [19].Njegova namjena je da se koristi u aplikacijama koje zahtijevaju stalnu vezu za korisnike. Tornado dolazi sa svojim HTTP poslužiteljem baziranom na asinkronoj biblioteci. Iako je moguće koristiti Tornado sa WSGI (engl. *Web Server Gateway Interface*), da bi se iskoristile sve mogućnosti Tornada najbolje ga je koristiti s Web poslužiteljem. WSGI je specifikacija koja opisuje kako web poslužitelj komunicira s web aplikacijama i kako se web aplikacije mogu povezati radi obrade jednog zahtjeva [20]. U Tornado je uključen jednostavni jezik za predloške u kojem za razliku od Jinje2 nema restrikcija u smislu korištenja izraza. Također, Tornado posjeduje koncept modula korisničkog sučelja, a to su posebni pozivi funkcija za renderiranje elemenata grafičkoga sučelja u koje možemo uključiti vlastiti CSS ili JavaScript. S obzirom da je Tornado dosta specifičan okvir. Tako je on idealan za korištenje ako aplikacija treba koristiti puno *WebSocket*-a ili *long-pollinga*. *Long-polling* je pristup kod kojeg poslužitelj zadržava klijentsku vezu otvorenom koliko god dugo je moguće. Pritom šalje odgovor samo onda kad podaci postanu dostupni ili je isteklo vrijeme veze [21].

4.6.3. Pyramid

Pyramid je okvir dizajniran za jednostavnost i brzinu. Glavni cilj Pyramida je omogućiti programeru da krene s razvojem bez prevelikog znanja o okviru. Pyramid se fokusira na većinu potreba Web aplikacija; mapiranje URL-ova u kod, podržavanje najboljih Pythonovih predložaka, posluživanje statičkim dodacima i pružanje sigurnosti [19].

Pyramid dolazi sa ugrađenim bootstraping alatom koji se zove pcreate koji kreira veći kostur projekta nego kod Flaska. Zbog toga jer je Pyramid namijenjen za veće projekte, ali se može koristiti i kod manjih aplikacija. Pyramid podržava rad s velikim brojem jezika za predloške, uključujući Jinju2 i Mako. Pyramid dolazi s jezikom za predloške Chameleon [22]. Jedna od prednosti Chameleona jest ta što se kod njega ne moraju renderirati predlošci. Također, u Chameleonu se koriste TAL-ovi (engl. *Template Attribute Language*) koji omogućavaju dodavanje logike i formatiranja znakovnih tipova.

4.7. Usporedba okvira za web programiranje u Pythonu

U ovome poglavlju bit će dana usporedba između programskih okvira za web programiranje u Pythonu. Svaki okvir dobit će za pojedinu kategoriju od 0 do 4 boda. 0 bodova bi značilo da okvir uopće nije primjenjiv, 1 bod znači da je djelomično primjenjiv, 2 boda da je primjenjiv, 3 boda znači da je okvir dobar, a 4 boda da je najbolji za danu kategoriju. Bodovi za kategorije dani su prema raznim izvorima i vlastitom iskustvu, a detaljnije će biti opisani ispod tablice. U tablici 2 dana je usporedba programskih okvira.

Tablica 2. Usporedba Python okvira za web (Izvor: autorski rad)

Okviri/Kategorije	**Django**	**Flask**	**Pyramid**	**Tornado**
Brzina učenja	1	4	3	2
Dokumentacija	3	4	2	1
Popularnost	4	3	1	2
Izrada velikih aplikacija	4	2	3	1
Izrada manjih aplikacija	1	4	3	2
Krajnje performanse (brzina)	3	2	1	4
Kreiranje početne konfiguracije	2	4	1	3
Vanjski paketi i biblioteke	3	4	2	1
UKUPNO	21	27	16	16

Prva mogućnost iz tablice 2 je brzina učenja. Pošto je Flask vrlo popularan mikro okvir, može se najbrže naučiti od navedenih okvira. Pyramid zbog svoje strukture također ima dobru brzinu učenja, ali ne kao Flask.

Django je full-stack okvir pa je za njegovo učenje potrebno najviše vremena. Bodovi za dokumentaciju dodijeljeni su prema veličini dokumentacije. Možemo zaključiti kako Flask ima najveću dokumentaciju, dok Tornado ima najmanju. Bodovi za popularnost dani su na temelju [23]. Bodovi za mogućnosti izrade velikih i manjih aplikacija dani su prema [19]. Krajnje performanse ocjenjene su prema rezultatima istraživanja brzine Python web okvira [24]. Kreiranje početne konfiguracije ocijenjeno je prema razini kompliciranosti početne konfiguracije. Mogućnost ugrađivanja vanjskih paketa i biblioteka najbolja je u Flasku, što je i logično jer je on mikro okvir pa zbog toga i zahtijeva velik broj vanjskih paketa. Django zbog svoje popularnosti također posjeduje puno paketa, ali manje nego Flask. Između Pyradmida i Tornada, zaključeno je kako Pyramid pruža veću i bolju podršku za vanjske pakete pa su mu dodijeljena 2 boda. Može se zaključiti kako je Flask vjerojatno najbolji okvir za razvoj web aplikacije u Pythonu pošto ima najviše broj bodova 4. Na kraju, generalno treba zaključiti usporedbu ova četiri okvira. Django je najbolji izbor za velike projekte. Flask je odličan okvir za razvoj malih i srednjih projekata, ali njegova popularnost raste te će u bliskoj budućnosti biti i vrlo pogodan za razvoj velikih projekata. Njegova jednostavnost u kreiranju konfiguracija, pogleda i ruta omogućuje stvaranje jednostavne aplikacije u kratkom vremenskom roku. Pyramid je ipak namijenjen projektima koji nisu preveliki, ali bi u budućnosti mogli rasti. Problem kod Pyramida je taj što nije jednostavan za konfiguraciju kao ostala tri okvira. Na kraju, Tornado je zamišljen kao okvir koji će rukovati asinkronim procesima. On omogućuje korištenje metoda koje kontroliraju kako se zahtjevi ili odgovori šalju odnosno primaju [25]. Možemo zaključiti kako je odluka ipak na programeru da sam prosudi koji mu okvir najbolje odgovara za pojedini problem.

4.8. Usporedba Pythona s ostalim programskim jezicima na strani poslužitelja

Kroz ovaj rad objašnjene su i prikazane sve mogućnosti Pythona, njegove prednosti i nedostaci. U ovom poglavlju biti će napravljena kratka usporedba Pythona s odabranim programskim jezicima na strani poslužitelja PHP i Java. Ovdje je bitno navesti neke aspekte svakog programskog jezika koji bi se trebali uzeti u obzir pri analizi programskih jezika. Neki od tih aspekata su jednostavnost učenja, brzina, pronalazak grešaka (engl. *Debugging*), povezivanje s bazama, web okviri, dokumentacija, podrška zajednice, cijena. Ovo su najbitnije točke, a u potpoglavljima u nastavku tablično je prikazana usporedba sa svakim programskim jezikom koji je ovdje naveden.

4.8.1. Python i PHP

PHP je skriptni jezik koji se koristi za programiranje na strani poslužitelja. Kreiran je 1994. godine od strane dansko-kanadskog programera Rasmusa Lerdofa. Najpopularniji je programski jezik za Web programiranje zbog svoje jednostavnosti i lakog ugrađivanja u HTML. U PHP-u su napisane neke od najpopularnijih Web aplikacija poput Facebooka, Wikipedije, Yahoo itd. PHP također ima svoje okvire od kojih su najpoznatiji Laravel, Symfony i Zend. Također, u PHP-u je napisan i najpoznatiji sustav za upravljanje sadržajem odnosno CMS (engl. *Content Management System*), a riječ je naravno o WordPressu. CMS služi za upravljanje sadržaja na web aplikaciji ili stranici. PHP je najsličniji programski jezik Pythonu, a ovdje u tablici 3 je prema [26] prikazana usporedba s Pythonom prema već navedenim parametrima.

Tablica 3: Usporedba Pythona i PHP-a (Izvor: autorski rad)

PYTHON	PHP
Lak za naučiti	Lak, ali teži za naučiti od Pythona.
Brz.	Brži od Pythona.
Nudi snažni program PDB (Python Debugger)	Nudi paket XDebug.
Integracija s bazom podataka nije tako jaka kao PHP.	Pristup prema više od 20 različitih sustava za upravljanje bazama podataka.
Web okviri: Django, Flask, Pylons, Pyramid.	Web okviri: Codeigniter, Zend, Laravel, Symfony.
Dokumentacija na službenim web stranicama.	Također dobro dokumentiran.
Poznat među zajednicom pa nudi izvanrednu podršku.	Također poznat pa nudi podršku zajednice.
Besplatan.	Besplatan.

Dakle, moglo bi se zaključiti da su ova dva jezika vrlo slična, oba su laka za naučiti. Brzi su, prepoznati, integrirani s bazama podataka, svaki od njih nudi svoje web okvire i alate za pronalazak grešaka. Najveća razlika između njih je ta što PHP može biti izravno ugrađen u HTML, dok se kod Pythona za to koriste isključivo jezici za predloške. Uz to, većina okvira u Pythonu dolazi s vlastitim poslužiteljem, dok je kod Pythona potrebna instalacija nekog vanjskog poslužitelja, npr. XAMPP-a.

4.8.2. Python i Java

Java je kompilacijski, objektno orijentirani programski jezik utemeljen na sintaksi jezika C++, nastala 1995. godine. Neke od prednosti programskog jezika Java su to što je neovisna o platformi na kojoj se razvija, jednostavna je za naučiti i pokrenuti, brže se pronalaze greške napisane u kodu, stabilna je te ima ogromnu zajednicu [27]. U Javi također postoje okviri, a najpoznatiji su Spring, Hibernate i Struts. U tablici 4 je prikazana usporedba Pythona i Jave.

Tablica 4: Usporedba Pythona i Jave (Izvor: autorski rad)

PYTHON	JAVA
Lak za naučiti.	Teži za naučiti od Pythona.
Kratka sintaksa.	Duga sintaksa, teška za čitanje.
Python Debugger	jdb
Brže se pokreće, radi sporije.	Sporije se pokreće, radi brže.
Ne zahtijeva puno članova tima	Zahtijeva puno članova u timu
Interpreter.	Kompajler.
Brz razvoj.	Sporiji razvoj, ali bolja stabilnost.
U potpunosti besplatan.	Može zahtijevati plaćenu licencu za razvoj velikih aplikacija.

Uspoređujući Python i Javu može se zaključiti da Python ima jednostavniju sintaksu, što omogućuje brže učenje samog programskog jezika. Java je brža od Pythona jer je kompilacijski jezik. Java je izvrsna kada je u pitanju skaliranje vaših aplikacija. Python je automatski identificira i povezuje tipove podataka i prikladniji je za brže prototipiranje aplikacije [28].

5. Flask

Flask spada u skupinu mikro okvira, što znači da je više fokusiran na jednostavnost i slobodu u strukturiranju Web aplikacije. Flask je baziran na Werkzeugu i Jinji2. Werkzeug je biblioteka za WSGI. U Werkzeugu je uključen interaktivni *debugger* koji omogućuje pronalaženje grešaka i prikazivanje grešaka u pregledniku. Također, s pomoću Werkzeuga Flask dolazi sa vlastitim poslužiteljem te za razvoj web aplikacije u Flasku nije potrebno instalirati nikakav virtualni poslužitelj. Međutim, taj poslužitelj koristi se samo za testiranje i razvoj [29].

5.1. Nacrti

Kako ne bismo držali svu poslužiteljsku logiku u jednoj skripti, uvijek je poželjno razdvojiti pojedine funkcionalnosti aplikacije u više modula. Flask koristi koncept nacrta (engl. *Blueprints*) za izradu komponenti aplikacije. Nacrti mogu znatno olakšati rad i razvoj velikih aplikacija [30]. Programer u izradi nacrta ima svu slobodu. Nacrte može strukturirati prema funkcionalnostima aplikacije, ulogama korisnika ili onako kako on misli da je najbolje. U ovome radu koriste se nacrti prema ulogama korisnika pa stoga postoje četiri nacrta, a to su : *centri*, *opg, admin* i *users*. Svaki nacrt brine o pojedinom tipu korisnika, dok *users* nacrt brine o funkcionalnostima koje su jednake za sve vrste korisnika te o neregistriranim korisnicima. Nacrte je najbolje napraviti odmah po početku izrade aplikacije jer kasnije su potrebne promjene na gotovo svim predlošcima. U ovom radu, napravljeni su posebni direktoriji unutar aplikacije, a u svakom se nalaze datoteke ruta, forme te __init__.py datoteka kako bi Python prepoznao da se radi o posebnom paketu. Nacrti se registriraju jednostavno. Najprije je potrebno uopće uvesti paket *Blueprint* iz *flask* biblioteke. Nacrti su kreirani instanciranjem objekta klase *Blueprint*. Konstruktor za tu klasu uzima dva argumenta; ime nacrta te modul ili paket gdje se taj nacrt nalazi, a najčešće je u Pythonu ispravno dodati *__name__* varijablu [31]. Primjer izrade nacrta *users* u routes.py datoteci.

```
from flask import Blueprint
users = Blueprint('users', __name__)
```

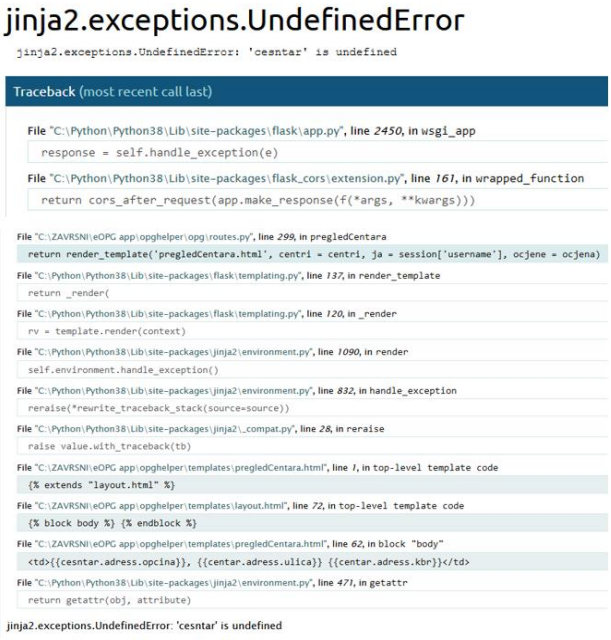
Zatim je u glavnoj __init__.py datoteci potrebno registrirati taj nacrt.

```
app.register_blueprint(users)
```

Da bi se održao normalan rad aplikacije, nacrti moraju komunicirati međusobno. Također, predlošci moraju komunicirati s nacrtima. U razvoju aplikacije s jednom skriptom nije potrebno definirati predlošku gdje se nalaze pojedine funkcije odnosno rute u routes.py datoteci jer se sve nalaze u toj jednoj datoteci. Međutim, kod korištenja nacrta, prilikom pozivanja nove rute, potrebno je toj ruti dodati i naziv nacrta u kojoj se ona nalazi.

5.2. Pronalazak grešaka u Flasku

Flask dolazi sa vlastitim *debuggerom* i programer ne treba dugo tražiti da pronađe grešku. Flask prijavljuje greške na nekoliko različitih načina. Pronalazak grešaka je jedna od najpozitivnijih prednosti Flaska jer vrlo jasno i precizno definira grešku i uvijek izbacuje točnu liniju koda na kojoj se greška nalazi. Za omogućavanje pronalaska greški, dovoljno je postaviti konfiguraciju aplikaciju u debug mod. Na slici broj 8 vidi se koju poruku dobije programer koji napravi grešku nakon pokretanja aplikacije.



Slika 8: Prikaz pogreške nastale u Jinji (Izvor: autorski rad)

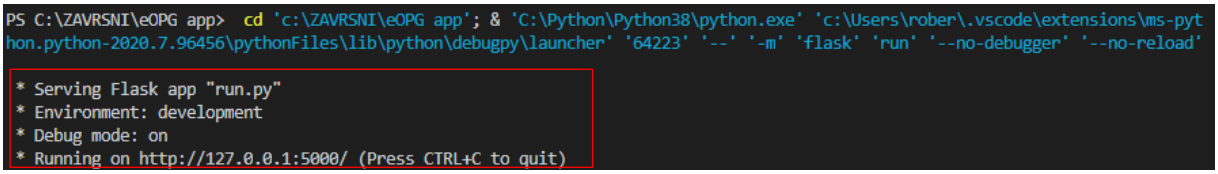
Na slici broj 8 vidimo da se greška (iznimka) se dogodila u Jinji, a dogodila se zbog tipfelera, odnosno pozvana je varijabla centar koja nije definirana ili poslana u predložak preko poslužitelja. Na debuggeru se jasno vidi njegov „tok“, odnosno on bilježi svaku točku na kojoj je programer mogao napraviti pogrešku. Također, debugger sam prepoznaje koje datoteke smo mi kreirali, a koje su kreirane od Pythona te naše datoteke označuje plavom bojom. Dakle, prva linija prikazuje koji je to dio poslužiteljskoga koda pozvao renderiranje predloška u kojem se dogodila pogreška.

Druga i treća osjenčana linija najčešće nisu toliko bitne jer one najčešće pokazuju tok kojim nasljeđujemo HTML predloške. I na kraju posljednja osjenčana linija prikazuje kod na kojem se dogodila pogreška, kao i datoteku i liniju gdje je pogreška. Pogreške, kao i neke ostale radnje koje nas zanimaju, je moguće tražiti i putem debuggera u Visual Studio Code-u. Za to je potrebno odabrati Flask kao iz liste programskih jezika kod opcije debuggiranja u Visual Studio Code-u. Tada se automatski pomoću u Visual Studio Code-a kreira datoteka launch.json koja sadrži konfiguraciju debuggiranja, a tu datoteku vidimo na slici 9.

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python: Flask",
            "type": "python",
            "request": "launch",
            "module": "flask",
            "env": {
                "FLASK_APP": "run.py",
                "FLASK_ENV": "development",
                "FLASK_DEBUG": "1"
            },
            "args": [
                "run",
                "--no-debugger",
                "--no-reload"
            ],
            "jinja": true
        }
    ]
}
```

Slika 9: datoteka launch.json (izvor: autorski rad)

U toj JSON datoteci na slici 9, uz imena, tipa i zahtjeva, nalazi se i „module“: „flask“ koji daje naredbu VS Code-u da pokrene Python s naredbom *-m flask*. Također, definira varijablu FLASK_APP kojoj se prosljeđuje naziv datoteke za pokretanje aplikacije, u ovom slučaju run.py [32]. Preostale opcije su u kojoj ćemo okolini pokrenuti aplikaciju te hoćemo li je pokrenuti u debug modu ili ne. Ispod je prikaz te JSON datoteke. Na slici 10 vidimo i koje naredbe se pokreću kada pokrenemo debuggiranje.



```
PS C:\ZAVRSNI\eOPG app>  cd 'c:\ZAVRSNI\eOPG app'; & 'C:\Python\Python38\python.exe' 'c:\Users\rober\.vscode\extensions\ms-python.python-2020.7.96456\pythonFiles\lib\python\debugpy\launcher' '64223' '--' '-m' 'flask' 'run' '--no-debugger' '--no-reload'
 * Serving Flask app "run.py"
 * Environment: development
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Slika 10: Prikaz naredbi nakon pokretanja debuggiranja (Izvor: autorski rad)

Slika 10 prikazuje detalje o pokretanju debuggiranja i aplikacije. Dakle daje nam informacije o tome koja skripta je pokrenuta, u kojem okruženju, je li uključen mod za otkrivanje pogrešaka te na kojoj je lokaciji aplikacija pokrenuta.

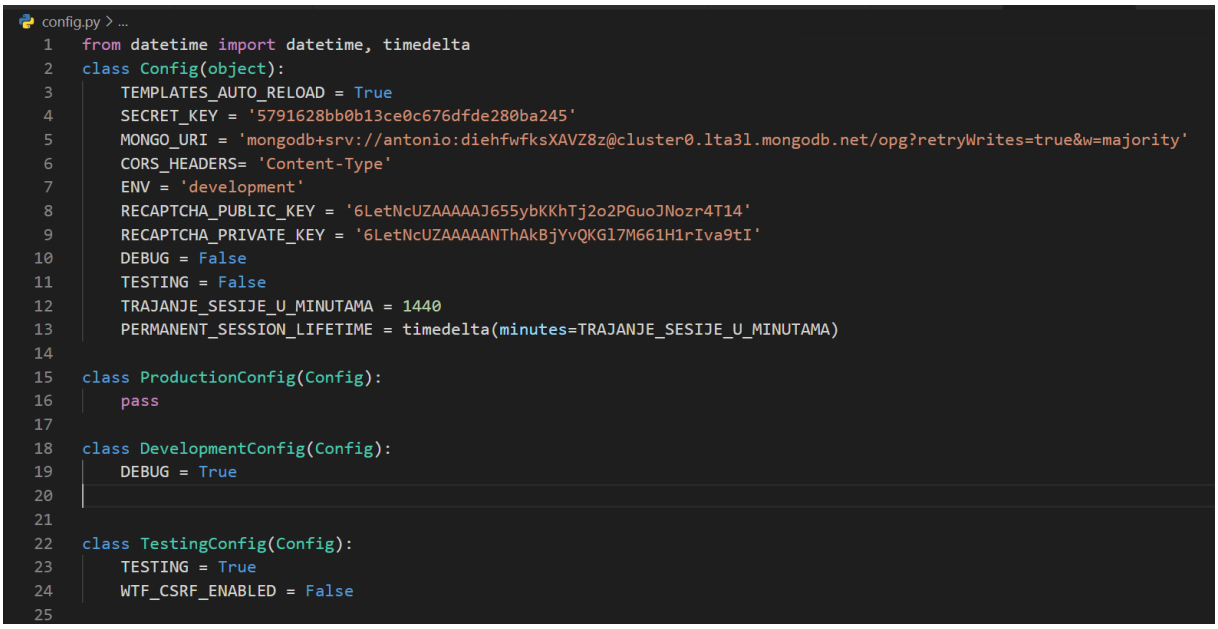
Jedan od najvažnijih alata pri korištenju VS Code debuggera jest interaktivna alatna traka (engl. *Toolbar*) kojom možemo manipulirati tijekom izvođenja programskog koda liniju po liniju ili preskakanjem linija od točke prekida (engl. *Breakpoint*) do točke prekida. Izgled alatne trake prikazan je na slici 11.

Slika 11: Alatna traka u Visual Studio Code-u (Izvor: autorski rad)

Slika 11 prikazuje izgled alatne trake u uređivaču Visual Studio Code. Funkcije će biti redom objašnjene. Prva mogućnost alatne trake su „Continue“ ili „Pause“ (ovisno o tome izvodi li se program ili ne). „Continue“ izvodi ostatak koda do iduće točke prekida ili kraja programa dok „Pause“ zaustavlja izvođenje programa kako bi programer mogao analizirati stanje sustava. Iduća je funkcija „Step Over“ kojom prelazimo na izvođenje iduće linije programskog koda. Zatim je tu funkcija „Step Into“ koju koristimo kod poziva neke funkcije i onda ulazimo u kod te funkcije također prelazeći ga liniju po liniju. Ako želimo izaći iz te funkcije, koristimo naredbu „Step out“. Opcijom „Restart“ ponovno pokrećemo debugger. Opcijom „Stop“ zaustavljamo izvođenje debuggera.

5.3. Konfiguracija aplikacije

Razvoj aplikacije započinje definiranjem okruženja u kojima će aplikacija postojati. Postoje tri okruženja u kojem aplikaciju djeluje, a to su razvojno, testno i produkcijsko. Svako navedeno okruženje ima svoju svrhu i posebne parametre koje koristi. Za kreiranje konfiguracijskih varijabli, kreirana je datoteka config.py u kojoj je definirana klasa Config kao i klase za svako pojedino okruženje koje tu klasu nasljeđuju što je prikazano na slici 12. Klase za svako pojedino okruženje također imaju specifične parametre koji ovise o svrsi klase.



Slika 12: Primjer konfigracije aplikacije (Izvor: autorski rad)

Slika 12 prikazuje konfiguracijske varijable. One koje treba spomenuti su svakako *SECRET_KEY* koji nam omogućuje upravljanje sesijama. Konfiguracijska varijabla *MONGO_URI* omogućuje spajanje na bazu. U varijabli *TRAJANJE_SEISIJE_U_MINUTAMA* određuje se koliko će sustav dozvoliti korisniku da koristi aplikaciju bez ponovne prijave. Zadana vrijednost trajanja jedne sesije u Flasku je 31 dan. Konfiguracijske varijable *ENV*, *DEBUG* i *TESTING* su one koje zapravo mijenjamo tijekom promjene okoline. Kad smo u razvojnoj okolini, poželjno je uključiti *DEBUG* jer želimo biti u mogućnosti pronaći greške. Kod testiranja, primjenjujemo *TESTING*, što podrazumijeva CSRF (engl. *Cross Site Request Forgery*) zaštitu jer nećemo biti u mogućnosti provesti testove za koje je potrebno unijeti podatke u obrazac. U radu će biti korišteno proširenje WTF (*What The Forms*). Više o samom proširenju će se spomenuti u nastavku rada. Razlog tome je jer proširenje WTF namjerno skriva te podatke kako bi obranila aplikaciju od potencijalnih napada. U kodu iznad vidimo kako je zadana vrijednost varijable *ENV* jednaka 'development'. To znači da aplikaciju pokrećemo u razvojnom okruženju. Ako želimo promijeniti okruženje, u glavnom __init__.py modulu moramo postaviti varijablu *ENV*. Ispod je dan primjer definiranja *ENV* varijable te poziva okruženja ovisno o *ENV* varijabli.

```
app.config['ENV'] = 'development'
if app.config['ENV'] == 'production':
    app.config.from_object("config.ProductionConfig")
if app.config['ENV'] == 'development':
    app.config.from_object("config.DevelopmentConfig")
if app.config['ENV'] == 'testing':
    app.config.from_object("config.TestingConfig")
```

5.4. Dohvat podataka iz vanjskog API-ja

U današnjem svijetu, postoji puno informacija i podataka na Internetu. Za izradu Web aplikacija, programeri moraju iskoristiti sve mogućnosti koje im pruža moderna tehnologija kako bi sam posao obavili što brže i kvalitetnije. Najbolji primjer za to su isti podaci koje koristi puno različitih Web aplikacija. Npr. većina Web trgovina posjeduje popis svih država na svijetu. Programer bi morao ručno unositi svaku državu u bazu podataka. Naravno, jednostavnije je pozvati API nego ručno upisivati sve države u bazu i onda ih ispisivati. U aplikaciji se koristi API s popisom hrvatskih županija, gradova i općina. API je dostupan na adresi https://tehcon.com.hr/api/CroatiaTownAPI/list.php. U svakom zapisu grada i/ili općine postoji ID županije i ime županije. Za potrebe aplikacije, općine i gradovi u pojedinoj županiji spojeni su u jedno polje. Na slici 13 nalazi se primjer dohvata podataka s vanjskog API-ja te spajanje gradova i općina pod istom županijom u jednu listu rječnika.

```
6  @admin.route('/opcina/<string:zupanija>')
7  def opcina(zupanija):
8  req = requests.get('https://tehcon.com.hr/api/CroatiaTownAPI/list.php')
9      data = json.loads(req.content)
10     gradoviIOpcine = []
11     for x in data['towns']:
12         if x['countyName'] == zupanija:
13             gradoviObj = {}
14             gradoviObj['id'] = x['name']
15             gradoviObj['name'] = x['name']
16             gradoviIOpcine.append(gradoviObj)
17
18     for x in data['communities']:
19         if x['countyName'] == zupanija:
20             opcineObj = {}
21             opcineObj['id'] = x['name']
22    opcineObj['name'] = x['name']
23             gradoviIOpcine.append(opcineObj)
24
25     return jsonify ({'gradoviopcine' : gradoviIOpcine})
```

Slika 13: Primjer dohvata podataka iz vanjskog API-ja (izvor: autorski rad)

Kao što je prikazano na slici 13, funkciji *opcina* prosljeđujemo naziv županije. U varijablu "req" spremamo GET zahtjev prema linku API-ja. Zatim, da dobijemo JSON objekt moramo deserijalizirati sadržaj zahtjeva, koji je trenutno zapis u bajtovima. Kad smo dobili sve podatke, potrebno je filtrirati i u listu unijeti samo one gradove i općine koje pripadaju toj županiji. Na svaku korisnikovu promjenu županije, okida se JavaScript događaj (engl. *Event*) koji zatim poziva ovu funkciju.

5.5. Rad s HTML predlošcima

Python programskom kodu u Jinji2 pristupamo blokom {% neki kod %}. Varijablama u Jinji2 pristupamo dvostrukim vitičastim zagradama. U nastavku će se opisati korištenje Jinje2 i Pythona kao i nasljeđivanje samih predložaka.

5.5.1. Pristup kodu

Preduvjet za korištenje Jinje je da je potrebno poznavati kako pristupiti dijelovima programskoga koda u Pythonu. U route() funkcijama potrebno je vratiti funkciju *render_template()* koja kao parametre prima naziv predloška te varijable koje želimo koristiti u tom predlošku. Bitno je ponovno naglasiti da se svi predlošci moraju nalaziti u datoteci templates, inače Python neće prepoznati o kojem se predlošku radi.

```
@opg.route('/povijestProdaje')
def povijestProdaje():
if (session['type'] == 'opg'):
    offersResults = oglasi.find({"kupljen" : True, "prodavatelj" : session['korisnicko_ime']})
    return render_template("povijestProdaje.html", oglasi = offersResults)
```

U ovom kodu na ruti /povijestProdaje provjerava se je li trenutno prijavljeni korisnik tipa OPG. Ako je, izvršavamo upit nad prodanim oglasima u kojima je on prodavatelj. Nakon toga funkcijom *render_template()* pozivamo renderiranje predloška koji se zove povijestProdaje.html i tom predlošku prosljeđujemo rezultate upita. Rezultate upita postavili smo u varijablu oglasi i preko te varijable, generiramo HTML sadržaj uz pomoć Jinje što je prikazano na slici broj 14.

```
{% extends "layout.html" %}
{% block body %}
    <div class="main_content">
        {%if oglasi %}
        {% for oglas in oglasi %}
        <div class="row">
            <div class="col">
                <h2 id="naslovOglasa">{{oglas.naziv}}</h2>
                <p> Cijena: {{oglas.cijena}}</p>
                <p>Proizvod: {{oglas.proizvod}}</p>
            </div>
            <div class="col">
                <p>Količina: {{oglas.kolicina}}</p>
                <p>Prijevoz: {{oglas.prijevoz}}</p>
                <p>Datum kupnje: {{oglas.datumKupnje.strftime('%d.%m.%Y.')}}</p>
            </div>
            <div class="col">
                <p>Kupac: {{oglas.kupac_id}}</p>
            </div>
        </div>
        <hr>
        {% endfor %}
        {% else %}
         <h1>Nemate prodanih oglasa</h1>
        {% endif %}
    </div>
{% endblock %}
```

Slika 14: Primjer pristupa Python varijablama i funkcijama u HTML-u pomoću Jinje2 (Izvor: autorski rad)

Dakle, kao što vidimo na slici broj 14, prvo pomoću *if* grananja provjeravamo postoje li uopće oglasi. Ako postoje, ulazimo u for petlju i za svaki pojedini oglas ispisujemo atribute. Atributima se pristupa točkom ili uglatim zagradama pa će sljedeći kod dati isti rezultat:

```
{{oglas.naziv}}
{{oglas['naziv']}}
```

Također, na primjeru ispisa datuma označenog crvenim pravokutnim područjem, možemo vidjeti kako određenim atributima možemo manipulirati preko ugrađenih Python funkcija.

5.5.2. Nasljeđivanje predložaka

Jedna od najvećih prednosti Jinje2 jest nasljeđivanje predložaka (engl. *Template inheritance*). Nasljeđivanje predložaka omogućuje razdvajanje HTML koda na više datoteka (predložaka). Na slici broj 15 prikazan je predložak layout.html. To je predložak kojeg učitavaju i koriste svi ostali predlošci.

```
<!DOCTYPE html>
<html>
    <head>
        <title>OPG Helper</title>
        <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css') }}">
        <link href="//maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" rel="stylesheet" id="bootstrap-css">
        <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">
    </head>

    <body>
        {% block login %}  {% endblock %}
        {% block zahtjev %}  {% endblock %}
        {% if session['username'] %}
                <div class="sidebar active" id="sidebar">
                    <div class="custom-menu">
                        <button type="button" id="sidebarCollapse" class="btn btn-primary">
                            <i class="fa fa-bars"></i>
                            <span class="sr-only">Toggle Menu</span>
                        </button>
                    </div>
                    <h2 id="imeKorisnika">{{session['username']}}</h2>
                    <ul>
                        {% if session['type'] == 'centar' %}
                            <li><a href="{{ url_for('users.index') }}"><i class="fas fa-home"></i>Početna</a></li>
                            <li><a href="{{ url_for('centri.dodajOglasCentar') }}"><i class="fas fa-address-card"></i>Predaj oglas</a></li>
                            <li><a href="{{ url_for('centri.oglasiOpgova') }}"></i>Pretraga oglasa</a></li>
                            <li><a href="{{ url_for('centri.mojiOglasiCentar') }}"><i class="fas fa-blog"></i>Moji oglasi</a></li>
                            <li><a href="{{url_for('centri.povijestKupnje')}}"><i class="fas fa-address-book"></i>Povijest kupnje</a></li>
                            <li><a href="{{url_for('centri.pregledOpgova')}}"><i class="fas fa-map-pin"></i>Pregled opgova</a></li>
                            <li><a href="{{url_for('users.porukeAdmina')}}"><i class="fas fa-map-pin"></i>Poruke s adminom
                                <span class="upper badge badge-danger" style="visibility: hidden">
                                </span>
                            </a></li>
                            <a class="btn btn-danger" id="down" href="{{url_for('users.logout')}}"> Odjavi se </a>
                        {% endif %}
```

Slika 15: Dio predloška layout html (Izvor: autorski rad)

U predlošku na slici 15 definirani su glavni elementi HTML-a (head i body). Nasljeđivanje se omogućuje ključnom riječi *block*. Kad otvorimo blok u koji želimo unijeti HTML iz nekog drugog predloška, tom bloku moramo dati ime i moramo taj isti blok zatvoriti ključnom riječi *endblock*. U konkretnom primjeru, ako je korisnik prijavljen tad će layout.html predložak prikazati navigaciju. Ako nije, pokreće se blok naziva login koji sadrži obrazac za prijavu. Nakon što je navigacija prikazana, ulazimo u blok body kojeg koriste gotovo svi ostali predlošci. Još valja spomenuti kako predložak layout.html posjeduje još jedan blok, a to je blok script u kojem po potrebi pišemo JavaScript. Primjer nasljeđivanja ovog predloška dan je u slici 16.

```
{% extends "layout.html" %}
{% block body %}
<div class="main_content">
    <table class="table">
        <thead class="thead-dark">
          <tr>
            <th scope="col">Korisničko ime</th>
            <th scope="col">Tip</th>
            <th scope="col">OIB</th>
            <th scope="col">Županija</th>
          </tr>
        </thead>
        <tbody>
            {%for korisnik in korisnici %}
                <tr>
                    <td><a href="{{url_for('admin.korisnik', korisnik = korisnik.username, tip = korisnik.type)}}">{{korisnik.username}}</a></td>
                    <td>{{korisnik.type}}</td>
                    <td>{{korisnik.oib}}</td>
                    <td>{{korisnik.adress.zupanija}}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
{% endblock %}
```

Slika 16: Primjer nasljeđivanja predloška (Izvor: autorski rad)

Na slici broj 16 vidimo na koji se način nasljeđuje predložak, a to je ključnom riječi *extends*. Nakon što se predložak naslijedi, jednostavno ulazimo u definirani blok i pišemo HTML po želji.

5.6. Flask proširenja

Jedna od najboljih karakteristika Pythona jest ta što ima jako puno vanjskih biblioteka koje se mogu koristiti za razvoj programskih proizvoda. Flask isto kao i Python koristi veliki broj vanjskih biblioteka i proširenja. Zbog toga što je u izradi aplikacije fokus ipak bio na samom Pythonu, korištena su samo dva proširenja Flaska: Flask-WTF i Flask-SocketIO.

5.6.1. Flask – WTF

Flask-WTF (What the Forms) je vrlo korisno proširenje za razvoj Web aplikacija u Flasku. Njome možemo definirati klase podataka kojih želimo u pojedinim obrascima. Flask-WTF podržava sve elemente HTML forme. U ovome radu, svi obrasci za unos su smještene u datoteke forms.py koje se nalaze u nacrtima. Većina obrazaca za ažuriranje unesena je ručno, bez korištenja Flask -WTF, jer se za ažuriranje koristi metoda dijaloškog okvira. Svaki nacrt ima svoju datoteku forms.py jer nam npr. za ulogu OPG nisu potrebne iste forme kao i za ulogu administratora. Primjer za korištenje WTF jest dodavanje novog korisnika u bazu podataka sa strane administratora što je prikazano na slikama 17 i 18.

```python
from flask_wtf import FlaskForm
from flask_wtf.file import FileField, FileAllowed
from wtforms import StringField, SelectField, IntegerField, FloatField,
RadioField, DateField, SubmitField, PasswordField, FieldList, FormField, TextAreaField
from wtforms.validators import DataRequired, Length, Email, ValidationError
from opghelper import  mongo

class AdressForm(FlaskForm):
    zupanija = SelectField('Županija', validators=[DataRequired()], choices=[], validate_choice=False)
    opcina = SelectField('Općina', validators=[DataRequired()], choices=[], validate_choice=False)
    mjesto = StringField('Mjesto', validators=[DataRequired()])
    ulica = StringField('Ulica', validators=[DataRequired()])
    kbr =  StringField('Kućni broj', validators=[DataRequired()])
```

Slika 17: Primjer definiranja obrasca (Izvor: autorski rad)

```python
class UserForm(FlaskForm):
    korisnicko_ime = StringField('Korisničko ime', validators=[DataRequired()])
    password = PasswordField('Lozinka', validators=[DataRequired()])
    type = RadioField('Tip korisnika',  validators=[DataRequired()], choices=[('centar','Centar'), ('opg', 'Opg')])
    email = StringField('E-mail', validators=[DataRequired(), Email()])
    oib = StringField('OIB', validators=[DataRequired(), Length(min=13, max=13)])
    adresa = FormField(adresaForm)
    slika = FileField('Profilna slika', validators=[FileAllowed(['jpg', 'png'])])
    submit = SubmitField('Napravi korisnika')
    def validate_korisnicko_ime(self, korisnicko_ime):
        user = mongo.db.korisnici.find_one({"korisnicko_ime": korisnicko_ime.data})
        if user:
            raise ValidationError("Korisničko ime već postoji")
    def validate_email(self, email):
        user = mongo.db.korisnici.find_one({"email": email.data})
        if user:
            raise ValidationError("Email ime već postoji")
```

Slika 18: Primjer definiranja obrasca - 2. dio (Izvor: autorski rad)

Na slikama 17 i 18 vidimo da nakon što odaberemo tip polja možemo odabrati i oznaku (engl. *Label*) koju želimo dodijeliti tom polju.

Također, u biblioteci postoje i validatori kojima ispitujemo ispravnost unesenih podataka. Ispod su opisani validatori korišteni u kodu iznad. Oni su:

- DataRequired – validator koji provjerava je li unesena vrijednost u polje, odnosno validator koji to polje postavlja obaveznim
- Length – validator kojim postavljamo maksimalnu ili minimalnu duljinu ako se radi o znakovnom tipu ili maksimalnu ili minimalnu vrijednost ako se radi o broju
- Email – validator kojim provjeravamo je li unesena e-mail adresa u skladu sa predloškom e-mail adrese (ime@host.domena)
- ValidationError – podiže se kad neki uvjet nije ispunjen

Implementirane metode *validate_korisnicko_ime* i *validate_email* služe za provjeru postoje li uneseno korisničko ime odnosno e-mail već u kolekciji korisnika. Kao parametre uzimaju obrazac nad kojom želimo da se metoda izvrši te polje nad kojim želimo izvršiti radnju (validaciju). U ovim slučajevima, metoda će se izvršiti nad obrascem u kojoj je i stvorena, pa koristimo parametar *self*. Nakon toga jednostavnim upitom provjeravamo postoji li ime ili email već u bazi te ako postoji, pozivamo *ValidationError*. Također, na slikama 17 i 18 vidimo da se obrasci definiraju kao klase, a te klase u sebi sadrže atribute (polja), a mogu sadržavati i metode. U obrascu *UserForm* koristimo različite tipove polja. Svaki tip polja predstavlja element forme u HTML-u. Neki tipovi polja koji su korišteni u aplikaciji prikazani su u tablici 5 zajedno s korespondirajućim HTML elementom.

Tablica 5. Usporedba tipova polja u WTForms i HTML elemenata (Izvor: autorski rad)

Tip polja	HTML element
SelectField	<select> <option></option> </select>
StringField	<input type="text>
PasswordField	<input type="password">
SubmitField	<input type="submit">

U tablici 5 s lijeve strane prikazani su tipovi polja u Flask – WTF, a s desne strane ekvivalent u HTML-u. Valja spomenuti i tip polja „FormField“. Njime pozivamo drugu obrazac koju smo prije definirali. U našem slučaju, kod dodavanja novog korisnika koristimo obrazac „UserForm“ koja nasljeđuje „AdressForm“ koja sadrži polja za unos adrese korisnika. Naravno, valja spomenuti i kako se takve obrasci uopće inicijaliziraju u kodu te onda i prikazuju u HTML-u za što je dan primjer na slici 19.

```python
from opghelper.admin.forms import  UserForm, AdressForm
users = mongo.db.users
@admin.route('/admin_dodaj_korisnika', methods=['GET', 'POST'])
def admin_dodaj_korisnika():
    if (session['type'] == 'admin'):
        …
        form = UserForm()
        if form.validate_on_submit():
            users.insert_one({
            "username" : form.username.data,
            "password" : form.password.data,
            "type" : form.type.data,
            "email" : form.email.data,
            "oib" : form.oib.data,
            "adress" : form.adress.data,
            "datumIVrijemeKreiranja" : datetime.now(),
            })
            flash('Objavili ste uspješno novi oglas!', 'success')
            return redirect(url_for('admin.admin_popis_korisnika'))
    return render_template("admin_dodaj_korisnika.html", form=form)

```

Slika 19: Isječak koda za dodavanje novog korisnika (Izvor: autorski rad)

Na slici 19 prikazan je isječak koda za dodavanje novoga korisnika. Prvo, obrazac se kreira pozivanjem imena klase u kojoj je definirana. Naravno, najprije tu klasu trebamo uvesti u ovu datoteku. Kod prvog učitavanja stranice, vraćamo predložak s obrascem za dodavanje korisnika kojem prosljeđujemo sam obrazac. Kasnije, kad se popuni podaci i obrazac šalje pritiskom na gumb „submit“, izvršava se grananje *if form.validate_on_submit(),* a to je funkcija koja vraća True ako je forma ispravno ispunjena, odnosno ako nema nikakvih validacijskih pogrešaka. Ako je uvjet ispunjen, u kolekciju korisnik dodajemo novoga korisnika s unesenim podacima. Na slici 20 će se pokazati isječak HTML koda i ispod opisati kako takav obrazac izgleda u HTML-u.

```html
<form method=POST action="">
    {{form.csrf_token}}
    {{form.adress.csrf_token}}
    <fieldset class="form-group">
    <div class="form-group">
        {{ form.username.label(class="form-control-label") }}
        {% if form.username.errors %}
            {{ form.username(class="form-control is-invalid") }}
            <div class="invalid-feedback">
                {% for error in form.username.errors %}
                <span>{{error}}</span>
                {% endfor %}
            </div>
        {% else %}
            {{ form.username(class="form-control ") }}
        {% endif %}
    </div>
    …
</form>

```

Slika 20: Primjer korištenja obrasca u HTML-u (Izvor: autorski rad)

Prije svega, za isječak na slici 20 moramo reći da koristimo prenesenu varijablu *form*. Najprije, zbog sigurnosnih razloga moramo postaviti CSRF token. CSRF je tip napada koji se događa kad maliciozna (zlonamjerna) web stranica, e-mail, blog ili aplikacija uzrokuje neželjeno ponašanje preglednika na sigurnim stranicama onda kad je korisnik prijavljen u sustav [31]. Postavljanjem tokena, na formu se dodaje posebni kriptirani ključ do kojeg napadači ne mogu doći i tako im je onemogućen bilo kakav napad takve vrste. Nakon toga krećemo s postavljanjem polja za unos i oznaka. Način na koji dolazimo do podataka iz klase obrasca jest taj da im pristupamo preko imena atributa koji su definirani u samoj klasi. Ako je došlo do greške prilikom validacije obrasca, tad želimo ispisati pogreške, a one se nalaze u atributu *errors*. Ako nema pogreške, jednostavno postavljamo polje za unos. Ako želimo dodati html elemente u obrazac (npr. klasu, id, ime, placeholder i sl.) onda to dodajemo u zagradu nakon pozivanja atributa klase i elemente odvajamo zarezom.

5.6.2. Flask – SocketIO

Za implementaciju slanja poruka u aplikaciji, korišteno je proširenje Flask-SocketIO. Flask-SocketIO Flask aplikacijama omogućuje pristup dvosmjernoj komunikaciji s malim kašnjenjem (engl. *low latency*) između klijenta i poslužitelja [33]. Proširenje je temeljeno na tehnologiji WebSocket-a. WebSocket je komunikacijski protokol koji omogućuje dvosmjernu komunikaciju između klijenta i udaljenog poslužitelja koji je zadužen za komunikaciju s tim klijentom. Cilj WebSocket-a je pružanje mehanizma Web aplikacijama koje zahtijevaju dvosmjernu komunikaciju poslužitelja koji se ne oslanja na otvaranje više HTTP (engl. *Hyper Text Transfer Protocol*) veza [34]. Da bi se moglo koristiti ovo proširenje moraju se izmijeniti neke skripte. U skripti „__init__.py“ moramo dodati samu biblioteku naredbom `from flask_socketio import SocketIO`

Nakon toga, potrebno je kreirati *Flask-SocketIO* poslužitelj, a to se napravilo naredbom
`io = SocketIO(app, cors_allowed_origins="*", manage_session=False)`
Ovdje smo sada postavili varijablu poslužitelja koju ćemo kasnije primijeniti. Prvi parametar *app* označava aplikaciju koju želimo postaviti na taj poslužitelj, a *app* je u ovom slučaju cijela aplikacija. *Cors_allowed_origins* parametar predstavlja listu podrijetala (engl. *Origin*) kojima je dopušteno povezati se sa SocketIO poslužiteljem. Ako taj parametar postavimo na '*' omogućujemo pristup svim podrijetlima. Prema zadanim postavkama pristup je omogućen samo jednakim podrijetlima. Parametar *manage_session* postavljen je na *false* jer se upravljanje sesijama na strani Flaska već koristi za login i gotovo u cijeloj aplikaciji. Kada je ovaj parametar postavljen na *false*, omogućava dijeljenje sesija između HTTP ruta i događaja Socket.IO-a. Zatim je potrebno promijeniti sam način pokretanja aplikacije u skripti run.py. Naredbom *io.run(app)* pokrećemo sami SocketIO poslužitelj.

Kad je to sve obavljeno, prelazimo na stvarno upravljanje događajima u SocketIO-u za što se koristio JavaScript i sam Flask. Najprije je u JavaScriptu potrebno napraviti varijablu u koju ćemo spremiti vezu na socket. Nakon toga kreiramo događaj koji se pokreće kad se dogodi nova veza. Tada toj vezi šaljemo podatke o imenu korisnika koje se nalazi uvijek nalazi u navigaciji pod id-jem „imeKorsinka“.

```
var socket = io.connect('http://127.0.0.1:5000');
socket.on('connect', function() {
    socket.send(document.getElementById('imeKorisnika').innerHTML);
})
```

Nakon toga na poslužitelju stvaramo rječnik aktivnih korisnika. Njega punimo u funkciji „handleMessage(msg)“ i kao ključ spremamo korisničko ime, a kao vrijednost ključ sesije (engl. *Session id*). Ključ sesije je potreban kako bismo znali kojem korisniku možemo poslati poruku.

```
korisnici = {}
@io.on('message')
def handleMessage(msg):
    korisnici[msg] = request.sid
```

Nakon što je taj dio riješen, možemo prijeći na samo slanje poruka. Na slici 21 prikazan je kod JavaScript događaja za slanje poruke.

```
var private_socket = io('http://127.0.0.1:5000/private');
document.getElementById('posalji').onclick = function() {
    var today = new Date();
    var dd = String(today.getDate()).padStart(2, '0');
    var mm = String(today.getMonth() + 1).padStart(2, '0');
    var yyyy = today.getFullYear();

    today = dd + '.' + mm + '.' + yyyy;
    var pr = document.getElementById('send_to_username').placeholder;
    var poruka = document.getElementById('poruka').value;
    var posiljatelj = document.getElementById('imeKorisnika').innerHTML;

    var mojDiv = document.getElementById('nov');
    var mojDrugi = document.createElement('div');
    const h2 = document.createElement('h2');
    const h5 = document.createElement('h5');
    const p = document.createElement('p');
    const hr = document.createElement('hr');

    mojDrugi.className = 'levo';
    h2.textContent = 'JA';
    h5.textContent = 'Datum : ' + today;
    p.textContent = 'poruka: '+ poruka;

    mojDrugi.append(h2, h5, p);
    mojDiv.append(mojDrugi);
    document.getElementById('poruka').value = '';
    private_socket.emit('private_message', {'username' : pr, 'poruka' : poruka, 'posiljatelj' : posiljatelj});
}
```

Slika 21: Primjer JavaScript događaja za slanje poruke (Izvor: autorski rad)

Prvo je potrebno kreirati novu varijablu *private_socket* jer želimo odvojiti događaj stvaranja veze od događaja slanja poruke. Kod kreiranja varijable, kreiramo i polje imena *private*. Ponovno imamo JavaScript događaj koji se pokreće nakon što smo kliknuli na gumb "Pošalji". Ovaj dio jednostavno upisuje poruku i detalje poruke koja je odmah vidljiva pošiljatelju. Također, isprazni se vrijednost elementa višeslojnog unosa (textarea) nakon što smo poslali poruku. No, ono najbitnije jest da tu poruku šaljemo upravljaču događaja na poslužitelju pozivanjem *emit* metode. Ta metoda sadrži naziv samog događaja (da poslužitelj zna kada treba upravljati) te JSON objekt koji sadrži primatelja, poruku i pošiljatelja. Nakon što je poruka poslana, krećemo s upravljanjem događaja što je prikazano na slici 22.

```python
@io.on('private_message', namespace='/private')
def private_message(input):
    if input['username'] in korisnici:
        try:
            primatelj_session_id = korisnici[input['username']]
        except Exception as inst:
            print(type(inst))
            print(inst.args)
            print(inst)
        poruka = input['poruka']
        poruke.insert_one(
            {
                "posiljatelj" : input['posiljatelj'],
                "primatelj" : input['username'],
                "poruka" : input['poruka'],
                "datum_slanja" : datetime.now()})
        emit('pokazi', {'poruka' : poruka, 'posiljatelj' : input['posiljatelj']}, room=primatelj_session_id)
    else:
        poruke.insert_one(
            {
                "posiljatelj" : input['posiljatelj'],
                "primatelj" : input['username'],
                "poruka" : input['poruka'],
                "datum_slanja" : datetime.now()})
```

Slika 22: Primjer upravljanja događajem (Izvor: autorski rad)

Na slici 22 vidimo primjer upravljanja događajem u Flasku. Metoda *private_message* prima parametar „input" koji je poslan u JavaScript događaju. Kako bi se izbjegle greške, potrebno je provjeriti status korisnika. Ako korisnik nije aktivan, samo upisujemo poruku u bazu podataka. Ukoliko je primatelj poruke trenutno aktivan prvo u varijablu „primatelj_session_id" spremamo ključ sesije za tog korisnika. Zatim u varijablu poruka spremamo vrijednost poruke i tu poruku upisujemo u bazu podataka jednostavnim MongoDB upitom. Na kraju, šalje se JavaScript događaj s nazivom „pokazi", JSON objektom u kojem se nalaze podaci o pošiljatelju i poruci te ono najbitnije, šaljemo ključ sesije primatelja kao vrijednost za parametar „room". SocketIO funkcionira na način da su kod razgovora (engl. *Chat*) svi korisnici podijeljeni u posebne prostore, sobe (engl. *Room*). Ovisno o konfiguraciji, članovi sobe mogu biti svi aktivni korisnici aplikacije, samo neki korisnici aplikacije, ili u ovome slučaju kod privatnoga slanja poruka, samo jedan korisnik aplikacije. Postavljanjem parametra „room" na ključ sesije primatelja, omogućujemo da samo taj primatelj primi našu poruku. Kako se omogućuje prikaz primljene poruke prikazano je na slici broj 23.

```
private_socket.on('pokazi', function(msg) {
    var pr = document.getElementById('send_to_username').placeholder;
    if (msg['posiljatelj'] ==  pr) {
        var today = new Date();
        var dd = String(today.getDate()).padStart(2, '0');
        var mm = String(today.getMonth() + 1).padStart(2, '0');
        var yyyy = today.getFullYear();

        today = dd + '.' + mm + '.' + yyyy;
        var mojDiv = document.getElementById('nov')
        const h2 = document.createElement('h2');
        const h5 = document.createElement('h5');
        const p = document.createElement('p');
        const hr = document.createElement('hr');

        h2.textContent = 'Šalje: ' + msg['posiljatelj'];
        h5.textContent = 'Datum : ' + today;
        p.textContent = 'poruka: ' + msg['poruka'];

        mojDiv.append(h2, h5, p, hr);
    }
});
```

Slika 23: Primjer JavaScript događaja za primanje poruke (Izvor: autorski rad)

Na kraju, poruka se prikazuje primatelju tako da se ponovno kreira događaj kojem u funkciji prosljeđujemo dobivenu poruku. Najprije je potrebno provjeriti ima li primatelj poruke otvorenu stranicu za razmjenu poruka s korisnikom koji je poslao poruku. Flask-SocketIO i njegove sobe rade na način da se može zaključiti tko treba primiti poruku, ali se ne može saznati tko ju je poslao. Dalje manipuliramo HTML-om na način koji se je već koristio kod slanja poruke na strani pošiljatelja.

5.7. Web servisi u Flasku

Kako se razvoj web aplikacija sve više razvija, paralelno raste potreba za dobrim korisničkim iskustvom. Stoga se prilikom planiranja razvoja web aplikacija programeri sve više odlučuju koristiti JavaScript okvire. Za rukovanje (manipulacijom) podataka, na poslužitelju je potrebno poslati te podatke klijentskoj strani. Za to se koriste web servisi.
Flask je veoma pogodan za razvoj REST (engl. *Representational State Transfer*) web servisa. Prema [35] karakteristike REST sustava definirane su sa šest pravila:

1. Klijent – Poslužitelj (engl. *Client - Server*) – treba razdvojiti poslužitelja koji pruža neku mogućnost ili uslugu i klijenta koji je koristi
2. Bez stanja (engl. *Stateless*) – svaki zahtjev klijenta mora sadržavati sve potrebne informacije o zahtjevu kako bi poslužitelj mogao obraditi zahtjev. Drugim riječima, poslužitelj ne može spremiti informacije poslane od klijenta u jednom zahtjevu te onda te iste informacije koristiti u drugome zahtjevu.
3. Mogućnost pohrane podataka (engl. *Cachealbe*) – poslužitelj mora poručiti klijentu može li se zahtjev spremiti ili ne

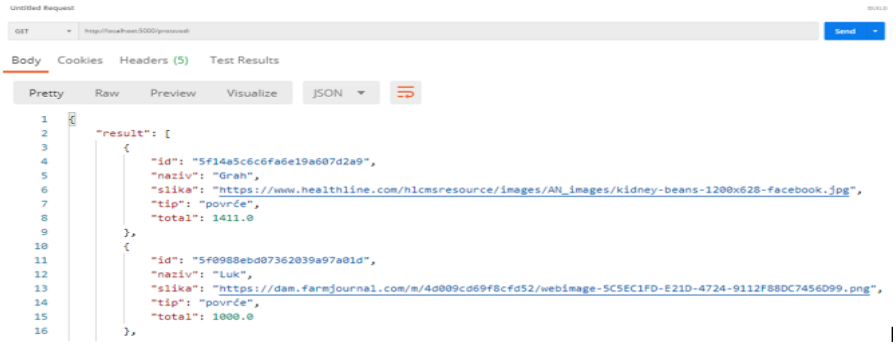
4. Slojevit sustav (engl. *Layered System*) – komunikacija između klijenta i poslužitelja mora biti standardizirana na način koji omogućuje posredniku da odgovori na zahtjeve umjesto krajnjeg poslužitelja.
5. Ujednačeno sučelje – metoda komunikacije između klijenta i poslužitelja mora biti ujednačena
6. Kod na zahtjev – poslužitelji mogu osigurati kod ili skripte za klijente

Web servisi se u ovom radu koriste za rukovanjem podataka proizvoda. Planirano je kasnije objaviti API za dohvat podataka o proizvodima. Trenutno samo korisnici aplikacije koji su prijavljeni mogu pristupiti proizvodima u aplikaciji. Međutim, neregistrirani korisnici preko Postmana mogu koristiti pregled svih proizvoda zajedno s ukupnom prodanom količinom proizvoda, pregled jednoga proizvoda, ažuriranje proizvoda te kreiranje i brisanje novoga proizvoda. Na slici 24, prikazan je kod kojim kreiramo JSON rezultat kojeg vraćamo u GET zahtjevu.

Slika 24: Primjer koda za kreiranje GET metode (Izvor: autorski rad)

Najprije se, kao što vidimo na slici 24, inicijalizira prazna lista i ta se lista onda puni podacima iz kolekcije „proizvodi“ u kojoj su zapisani svi proizvodi. Razlog zbog kojega nam je potreban id proizvoda je taj što moramo znati za koji proizvod želimo otvoriti obrazac za ažuriranje. Također dodajemo i polje 'total' kako bismo ga mogli napuniti kasnije kod proizvoda kod kojih je to potrebno. Zatim kreiramo listu u koju stavljamo sume količina svih prodanih oglasa grupirane po proizvodu. Na kraju uspoređujemo podatke iz dvije liste te pridružujemo vrijednost „total2 iz liste „producti“ u listu „jsonOutput“. Na kraju je potrebno sortirati listu tako da se prvo prikaže onaj proizvod čija je prodana količina u sustavu najveća. Putanja „/proizvodi“ vraća rezultat u JSON formatu. Valja napomenuti kako je renderiranje predloška za popis proizvoda implementiran na drugačiji način nego ostali predlošci. U tom predlošku ne koristi se Jinja2 već se preko JavaScript metode poziva putanja 'proizvodi' koja sadrži popis proizvoda u JSON obliku.

Nedostatak takvog pristupa dizajniranju predloška jest nemogućnost korištenja Jinje2 pri ispisu podataka i to što pojedini zapisi mogu postati preveliki . Na slici 25 prikazan je rezultat poslanog GET zahtjeva u Postmanu.



Slika 25: Rezultat GET zahtjeva u Postmanu (Izvor: autorski rad)

Rezultati GET zahtjeva vidljivi su u dijelu „Body“. Osim GET zahtjeva (Read), omogućene su ostale CRUD operacije preko HTTP metoda (POST metoda za Create, PUT za Update te DELETE za Delete) čiji je kod prikazan na slici 26.

```
@admin.route('/proizvodi', methods=['POST'])
def dodaj_proizvod():
    naziv = request.json['naziv']
    tip = request.json['tip']
    slika = request.json['slika']

    proizvod_id = products.insert({"naziv" : naziv, "tip" : tip, "slika" : slika})
    novi_proizvod = products.find_one({"_id" : proizvod_id})

    output = {'naziv' : novi_proizvod['naziv'], 'tip' : novi_proizvod['tip']}

    return jsonify({'result' : output})

@admin.route('/proizvodi/<string:naziv>', methods=['PUT'])
def azuriraj_proizvod(naziv):
    az_proizvod = products.update_one({"naziv" : naziv},
                                      {"$set" : {
                                          "naziv" :  request.json['naziv'],
                                          "tip" :  request.json['tip'],
                                          "slika" :  request.json['slika'],
                                      }})

    novi_proizvod = products.find_one({"naziv" : request.json['naziv']})

    output = {'naziv' : novi_proizvod['naziv'], 'tip' : novi_proizvod['tip']}

    return jsonify({'result' : output})

@admin.route('/proizvodi/<string:naziv>', methods=['DELETE'])
def obrisi_proizvod(naziv):
    products.delete_one({"naziv" : naziv})

    return redirect(url_for('admin.json_proizvodi'))
```

Slika 26: Isječak koda koji za ažuriranje, unos i brisanje proizvoda (Izvor: autorski rad)

Na slici 26 vidimo kod za metode POST, PUT i DELETE. Kod POST-a (dodavanja novog proizvoda) potrebno je primiti unesene podatke o imenu i tipu proizvoda. Nakon toga unosimo taj novi proizvod u bazu gdje se automatski generira id proizvoda. Taj novi proizvod onda vraćamo korisniku u POSTMAN. PUT metoda funkcionira na vrlo sličan način, jedino umjesto dodavanja novoga proizvoda ažuriramo stari. DELETE metoda jednostavno briše dani proizvod iz baze podataka.

5.8. Jedinično testiranje u Flasku

Iako preko debuggiranja, što je opisano u poglavlju 5.2, možemo pronaći greške koje utječu na rad i razvoj aplikacije, te greške su najčešće one koje je sam programer primijetio i koje su ga zaustavile u daljnjem razvoju. Također, programer ili tester mogu tražiti greške u aplikaciji detaljnim prolazom po aplikaciji što se naziva ručno testiranje (engl. *Manual Testing*). Međutim, postoje i načini na koji možemo automatizirati procese testiranja softvera, a jedan od tih načina jest jedinično testiranje (engl. *Unit Testing*). Jedinično testiranje jest razina testiranja softvera u kojoj su testirane individualne jedinice (komponente), gdje je jedinica ili komponenta najmanji dio sustava koji se može testirati. Cilj te vrste testiranja je potvrditi da svaka komponenta sustava radi onako kako je i zamišljena [36]. Jedinični testovi mogu ubrzati izradu aplikacije, a pogotovo njeno održavanje. Takvi testovi najčešće imaju jedan ili više ulaza te jedan ulaza preko kojega se provjerava ispravnost komponente. Kad je napisan test za jednu komponentu, i kad se napravi neka promjena na toj komponenti za koju je test već napisan, onda se ponovno provjeri prolazi li taj test i dalje. Da bismo mogli provesti jedinične testove, najprije je potrebno promijeniti okruženje iz razvojnog ili produkcijskog u testno. Nakon toga možemo započeti pisati testove za aplikaciju. Za to se kreirala nova datoteka test.py u glavnom direktoriju aplikacije. Idući je korak uvoz same aplikacije iz direktorija te uvoz Python biblioteke *unittest* koja se koristila za provođenje jediničnog testiranja. Ukupno je provedeno 9 testova, ali ovdje se neće opisivati svi. Ispod je dan primjer nekoliko testova kao i same incijalizacije istih koji će biti opisani ispod slike broj 27 na kojoj se nalazi programski kod.

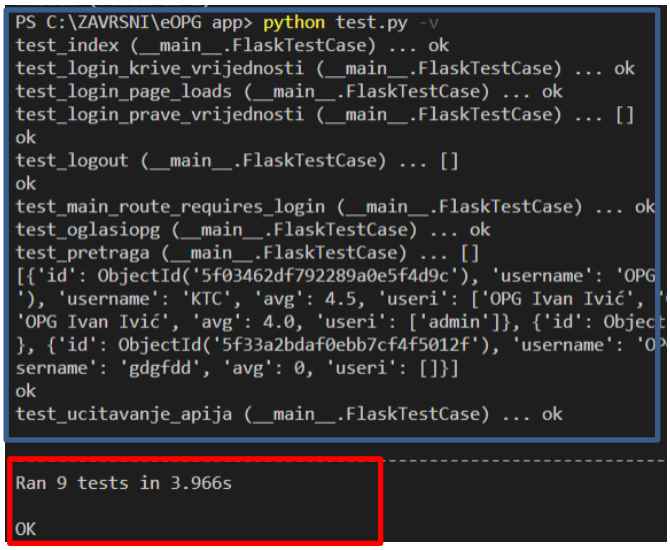
```
from opghelper import app
import unittest
    def test_login_prave_vrijednosti(self):
        tester = app.test_client()
        response = tester.post(
            '/',
            data=dict(korisnicko_ime='admin', password='administrator'),
            follow_redirects=True
        )
        self.assertIn(b'Prijavljeni ste', response.data)

    def test_logout(self):
        tester = app.test_client()
        tester.post(
            '/',
            data={'korisnicko_ime':'admin', 'password':'administrator'},
            follow_redirects=True
        )
        response = tester.get('/logout', follow_redirects=True)
        self.assertIn(b'Odjava', response.data)
```

Slika 27: Primjer dva jedinična testa (Izvor: autorski rad)

Slika 27 prikazuje primjer jediničnih testova za prijavu u aplikaciju i odjavu sa aplikacije. Dakle, potrebno je testove definirati kao funkcije u klasi proizvoljnog imena, gdje se onda

pomoću *unittest* biblioteke i njene klase „TestCase“ koja pretvara funkcije u testove. Ti testovi se onda kasnije pokreću preko „main“ funkcije. Važno je napomenuti da kod *unittesta* obavezno sve funkcije moraju počinjati riječju test. Prvi test koji ovdje vidimo jest testiranje prijave s točnim podacima za prijavu. Najprije je potrebno kreirati test klijenta za aplikaciju koju želimo testirati, a ovdje je taj klijent nazvan 'tester'. Zatim simuliramo POST zahtjev metodom post. Kao prvi parametar šaljemo rutu na koju želimo poslati podatke. Zatim šaljemo rječnik s podacima za formu, u ovome slučaju to su korisničko ime i lozinka. Još moramo postaviti „follow_redirects“ na true, budući da nakon što se korisnik uspješno prijavi, preusmjeren je na početnu stranicu. Na kraju, kako bismo potvrdili da je test prošao, moramo provjeriti je li se preusmjeravanje stvarno dogodilo. To činimo naredbom *assertIn* koja vraća True ako se neki niz znakova nalazi u odgovoru (engl. *Response*). Ovdje provjeravamo nalazi li se poruka 'Prijavljeni ste' u stranici odnosno predlošku koji se učita nakon preusmjeravanja. U drugom primjeru (test_logout), koji je vro sličan prvom, vidimo kako se testira odjava i općenito GET zahtjevi. Najprije je potrebno poslati POST zahtjev kako bismo prijavili korisnika, a zatim otići na rutu „/logout“ kako bi se isti odjavio. Budući da se nakon što se korisnik odjavi iz sustava pokaže poruka „Odjava“, provjeravamo nalazi li se taj niz znakova u predlošku koji je učitan. Nas slici broj 28 bit će prikazani rezultati jediničnih testova.



Slika 28: Prikaz rezultata jediničnih testova (Izvor: autorski rad)

Na slici broj 28 u plavom okviru vidimo rezultate svih 9 testova i za svaki dobivamo ispis samog naziva testa te povratnu poruku „OK“ što znači da su svi testovi prošli. Također, u crvenome pravokutniku vidimo da dolazi poruka o ukupnom broju samih testova i vremenu potrebnom za testiranje, kao i potvrda da su svi testovi prošli.

6. eOPG

U praktičnom dijelu izrađena je funkcionalna aplikacija za prodaju voća i povrća u programskom jeziku Python i okviru Flask.

6.1. Opis aplikacijske domene

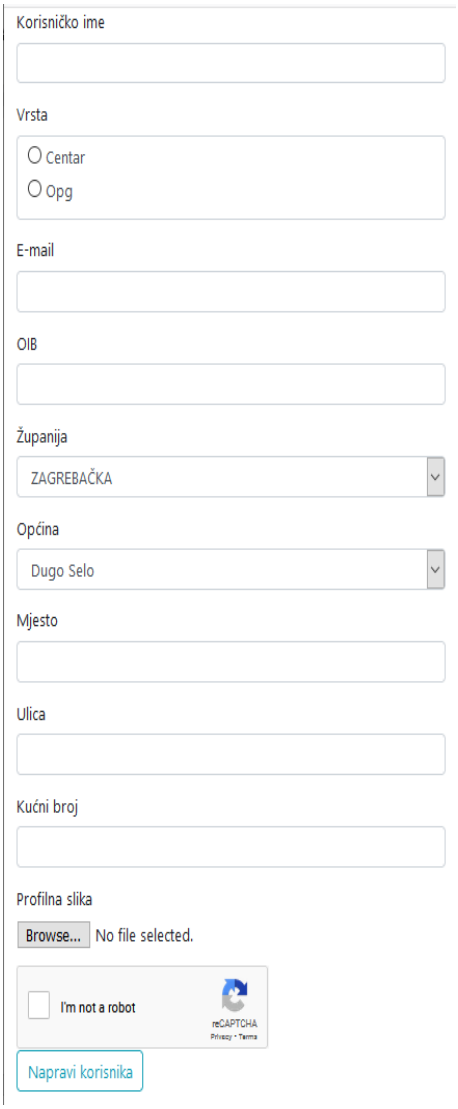
Web aplikacija služi za olakšavanje komunikacije i razmjenu robe između OPG-ova (obiteljsko poljoprivredno gospodarstvo, u daljnjem radu OPG) i kupaca njihove robe, trgovačkih lanaca (npr. Plodine, KTC, Kaufland, Konzum, itd.). Trenutno, da bi OPG prodao robu lancu, mora telefonski nazvati lanac kako bi ponudio svoju robu ili mora čekati telefonski poziv kako bi dobio narudžbu za svoj proizvod. Ako OPG surađuje s nekoliko lanaca, troškovi i vrijeme oduzeto na telefonske pozive znaju stvarati probleme vlasnicima OPG-ova, a pogotovo ako OPG posluje tijekom cijele godine. Također, telefonskim načinom dogovaranja, najčešće se ne mogu dokazati dogovorene količine i cijena. Ovom aplikacijom olakšala bi se trgovina voćem i povrćem.

6.2. Uloga neprijavljeni korisnik

Radi zaštite korisnika i specifične namjene aplikacije, korisnici se za korištenje aplikacije moraju registrirati. To se obavlja kroz poveznicu „Nemate račun? Pošaljite zahtjev za stvaranjem računa!“. Neprijavljeni korisnik može vidjeti samo obrazac za prijavu u sustav te kreirati zahtjev za kreiranjem računa. U slučaju da korisnik već ima ranije kreiran korisnički račun prijavljuje se unosom korisničkog imena i lozinke. Na slici broj 29 vidimo obrazac za prijavu.

Slika 29: Obrazac za prijavu (Izvor: autorski rad)

Nakon uspješnog unosa podataka za prijavu, korisnik se preusmjerava na početnu stranicu. Ako korisnik nema izrađen račun, klikom na poveznicu ispod obrasca prijave može poslati zahtjev za kreiranje računa. Zahtjev se šalje administratoru i on potvrđuje ili odbija zahtjev za kreiranjem računa. Na slici broj 30 vidimo obrazac za kreiranje računa.



Slika 30: Obrazac za kreiranje zahtjeva za izradom računa (Izvor: autorski rad)

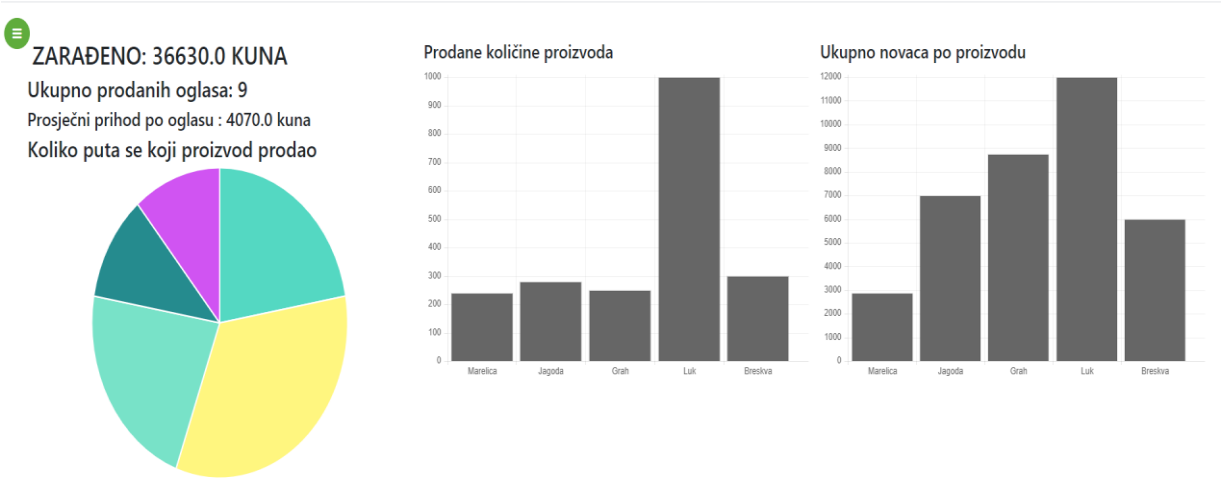
Kod kreiranja računa korisnik mora unijeti korisničko ime koje ne smije postojati u sustavu. Također, mora odabrati i vrstu korisnika. Nadalje, unosi i e-mail te OIB koji također moraju biti jedinstveni. Onda odabire adresu gdje najprije iz padajuće liste odabire županiju, a onda se prema odabranoj županiji bira općina ili grad iz generirane padajuće lista općina i gradova. Ta lista dohvaća se pomoću vanjskog API-ja. Nakon toga odabire mjesto, ulicu i kućni broj. Svi podaci u obrascu su obavezni. Na kraju, korisnik mora potvrditi svoj identitet tako da riješi test. Za to je korišteno gotovo Google-ovo rješenje za verifikaciju reCAPTCHA.

6.3. Uloga registrirani korisnik

U aplikaciji postoje dvije vrste registriranih korisnika. To su OPG i Centar. Jedan korisnik pripada samo jednoj vrsti. Svaki registrirani korisnik (jedan OPG ili jedan Centar) ima mogućnost prijave s jednim korisničkim imenom i lozinkom. Razlika između te dvije vrste registriranih korisnika je ta što oni vide oglase i podatke za drugu vrstu registriranog korisnika. Drugi riječima, Centri vide oglase i podatke OPG-ova, a OPG-ovi vide oglase i podatke Centara. U ovom poglavlju opisat će se uloga registriranog korisnika vrste OPG. Uloga registriranog korisnika vrste Centri imaju iste funkcionalnosti kao i OPG-ovi ali rade sa drugačijim podacima. Nakon uspješne prijave, korisnik koji se registrirao sa vrstom OPG na početnoj stranici vidi navigacijsku traku. Navigacija se prikazuje i sakriva klikom na gumb u gornjem lijevom kutu zaslona koji je označen kružnim područjem na slici 31. Na navigacijskoj traci nalaze se poveznice do glavnih funkcionalnosti aplikacije prijavljene uloge za koje ima prava sa mogućnošću odjave. Navigacijska traka prikazana je na slici 31.

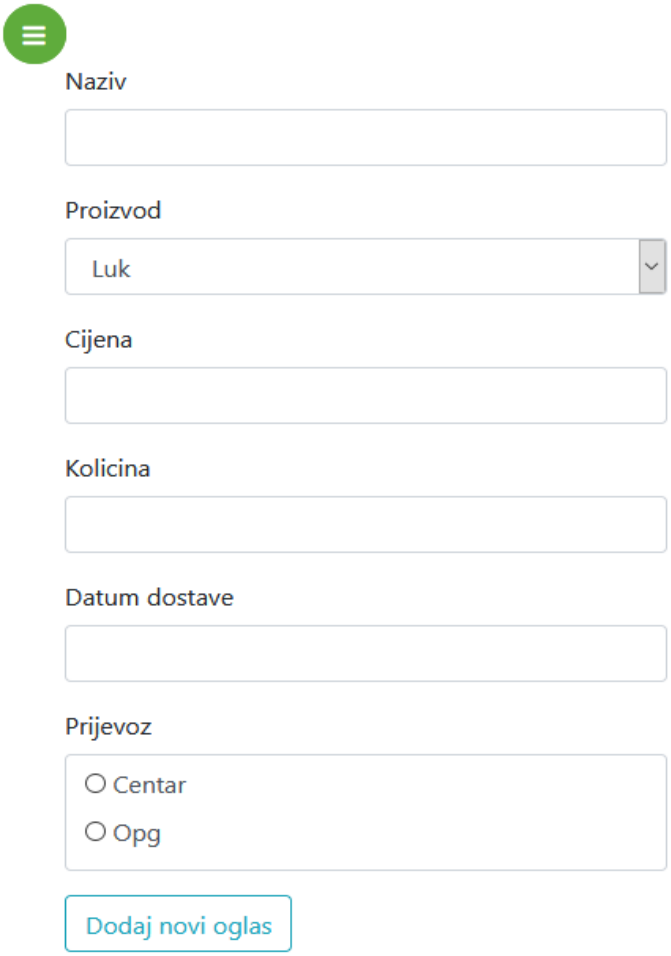
Slika 31: Navigacijska traka (izvor: autorski rad)

Na slici 31 vidimo navigaciju aplikacije za registriranog korisnika vrste OPG. Tako vidimo da on može predati ponudu (objaviti oglas). Zatim, može pretraživati aktivne oglase Centara i pregledavati vlastite aktivne oglase. Također, vidi i povijest svoje prodaje. Može i pregledavati sve Centre iz sustava te poslati poruku administratoru i vidjeti ranije poruke s administratorom. Klikom na opciju „Poruke“, označenom crvenim pravokutnikom, može pogledati koji su mu korisnici do sad poslali poruke. Tu također vidimo i broj nepročitanih poruka aktivnog korisnika koji je implementiran pomoću WebSocket-a. Klikom na „Početna“, prijavljeni korisnik vidi svoju statistiku prikazanu u obliku grafova što je vidljivo na slici 32.



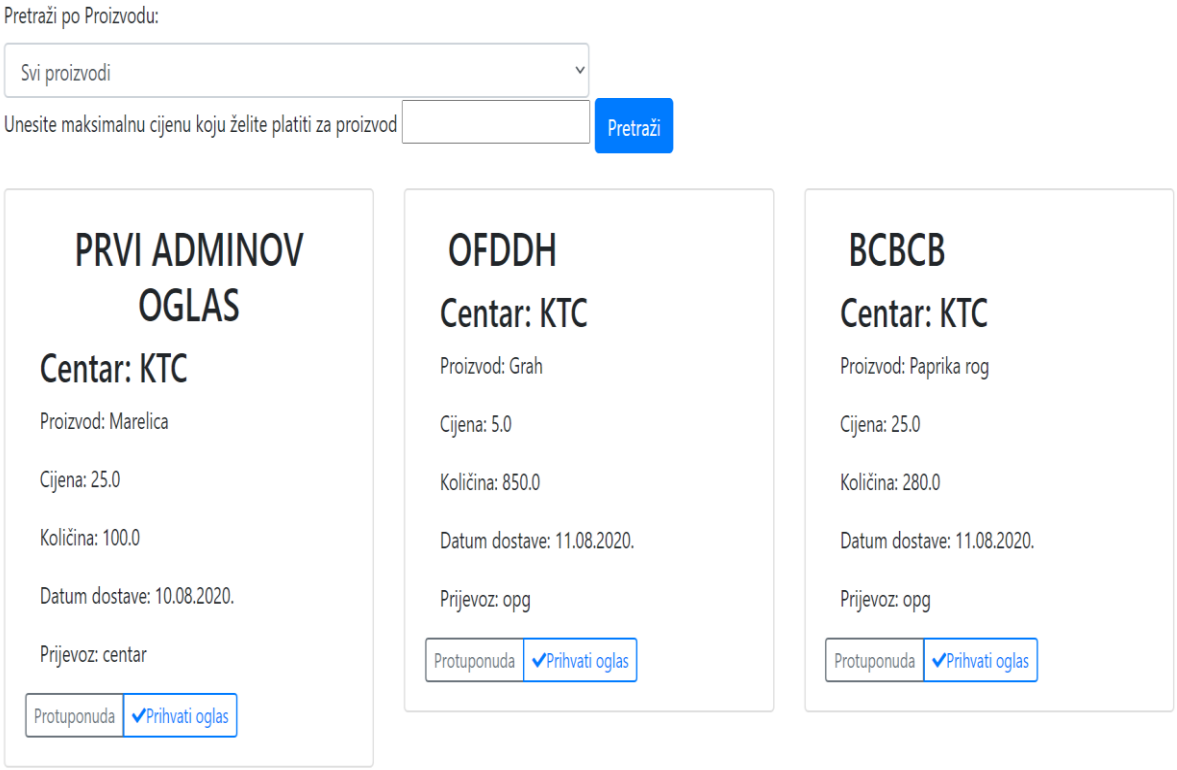
Slika 32: Početna stranica nakon prijave (Izvor: autorski rad)

Prijavljeni korisnik najprije, kroz kružni dijagram (engl. *Pie chart*) vidi koliko je puta prodao pojedini proizvod. Prelaskom preko obojanih područja dijagrama vidi naziv proizvoda i koliko ga je puta prodao. Veličina područja se generira automatski prema broju prodanih proizvoda. Na prvom stupčastom dijagramu (engl. *Bar chart*) vidi količine prodanih proizvoda u kilogramima. Na drugom stupčastom dijagramu vidi se koliko je prijavljeni korisnik ukupno zaradio po pojedinom proizvodu. Sljedeća funkcionalnost registriranog korisnika vrste OPG jest objava novog oglasa. Obrazac za kreiranje novog oglasa prikazana je na slici broj 33.



Slika 33: Obrazac za dodavanje novog oglasa (Izvor: autorski rad)

Oglas sadržava naziv, proizvod za koji se oglas objavljuje, cijenu po kilogramu, količinu proizvoda u kilogramima, datum dostave te vrstu prijevoza. Vrsta prijevoza znači hoće li OPG izvršiti prijevoz do skladišta ili prijemnog centra, ili će Centar pokupiti robu kod OPG-a. Valja naglasiti kako su sva polja u obrascu obavezna za ispuniti. Unosom novog oglasa OPG-a provjeravaju se aktivni oglasi centara. Ako postoji centar koji ima veću ili jednaku količinu za taj proizvod u aktivnom oglasu (aktivni oglasi su svi oglasi koje autor oglasa ili administrator nisu izbrisali), tom centru se automatski šalje mail s detaljima oglasa o samom oglasu i OPG-u kako bi mogli dogovoriti prodaju. Nadalje, registrirani korisnik vrste OPG vidi aktivne oglase koje su objavili registrirani korisnici tipa Centar pod akcijom "Pretraga oglasa" koja se nalazi u navigaciji što vidimo na slici 34.

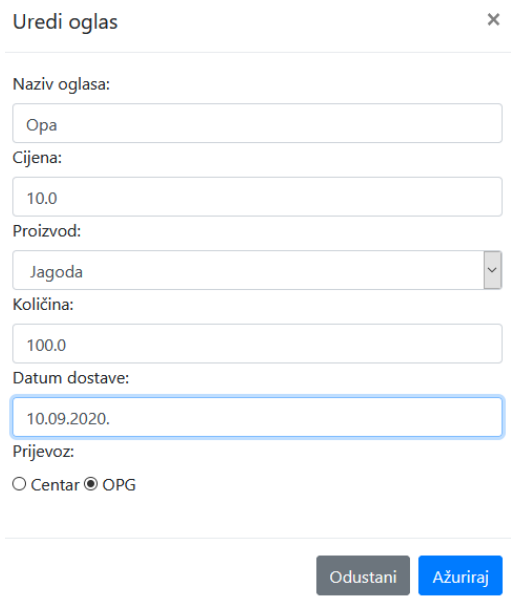


Slika 34: Lista aktivnih oglasa Centara (Izvor: autorski rad)

Slika 34 prikazuje sve detalje o oglasu. Oglase može filtrirati prema proizvodu i/ili prema minimalnoj cijeni za koju želi prodati kilogram svog proizvoda. Na te oglase može poslati protuponudu centru za pojedini oglas. Na primjer, u oglasu Centra tražena količina za rajčicu jest 5000 kilograma. Ako OPG ima samo 3000 kilograma rajčice, može poslati protuponudu Centru s tom količinom kojom raspolaže. Osim količine, u protuponudi može predložiti i novu cijenu te vrstu prijevoza. Osim slanja protuponude ima i mogućnost prihvaćanje oglasa čime je dogovorena prodaja proizvoda Centru. Centru se tad na mail šalje poruka kako je proizvod kupljen i kako je potrebno dogovoriti detalje s OPG-om. Na slici 35 prikazana je sljedeća funkcionalnost, a to je pregled vlastitih aktivnih oglasa. Uz aktivne, postoje i kupljeni oglasi koje registrirani korisnik vidi pod opcijom „Povijest prodaje“ kojima može pristupiti preko navigacije. Obrisani oglasi brišu se iz baze podataka.

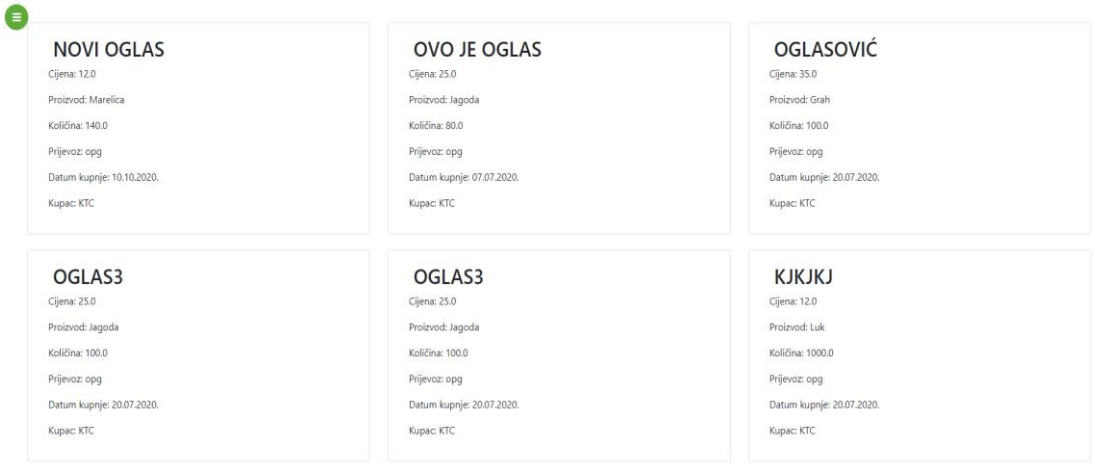
Slika 35: Lista vlastitih aktivnih oglasa (Izvor: autorski rad)

Na slici 35 prikazani su aktivni oglasi prijavljenog korisnika vrste OPG. Korisnik vrste OPG oglase može uređivati, brisati i pregledavati protuponude Centra za taj oglas. Klikom na uredi oglas otvara se obrazac za ažuriranje koji sadrži sve u podatke odabranog oglasa. I sve te podatke prijavljeni korisnik vrste OPG može mijenjati što je prikazano na slici 36.



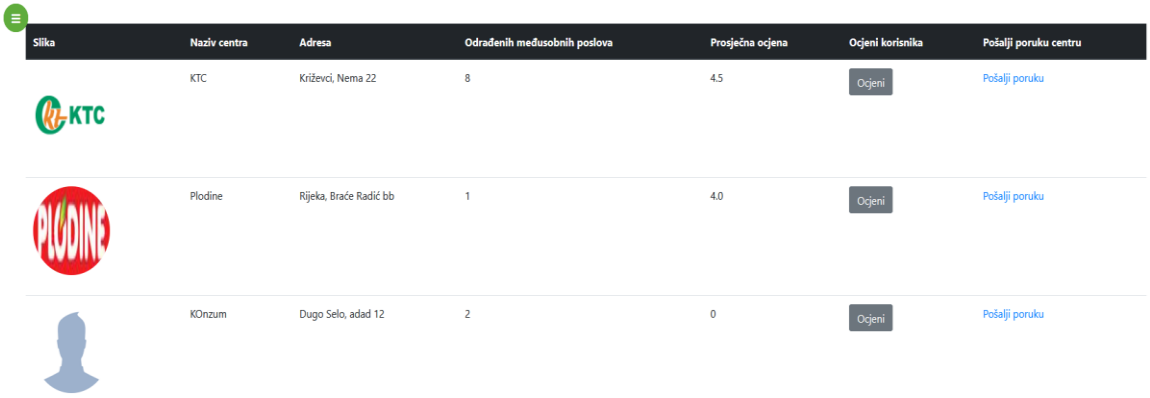
Slika 36: Obrazac za uređivanje oglasa (Izvor: autorski rad)

Na slici 36 vidimo obrazac za uređivanje odabranog oglasa. Sljedeća funkcionalnost je pregled povijesti prodaje u kojoj se nalazi svi oglasi u kojima je sudjelovao korisnik, bez obzira radi li se o njegovom oglasu, ili oglasu centra na koji se on javio. Detalji o oglasu uz datum kupnje i samog kupca vidljivi su na slici 37.



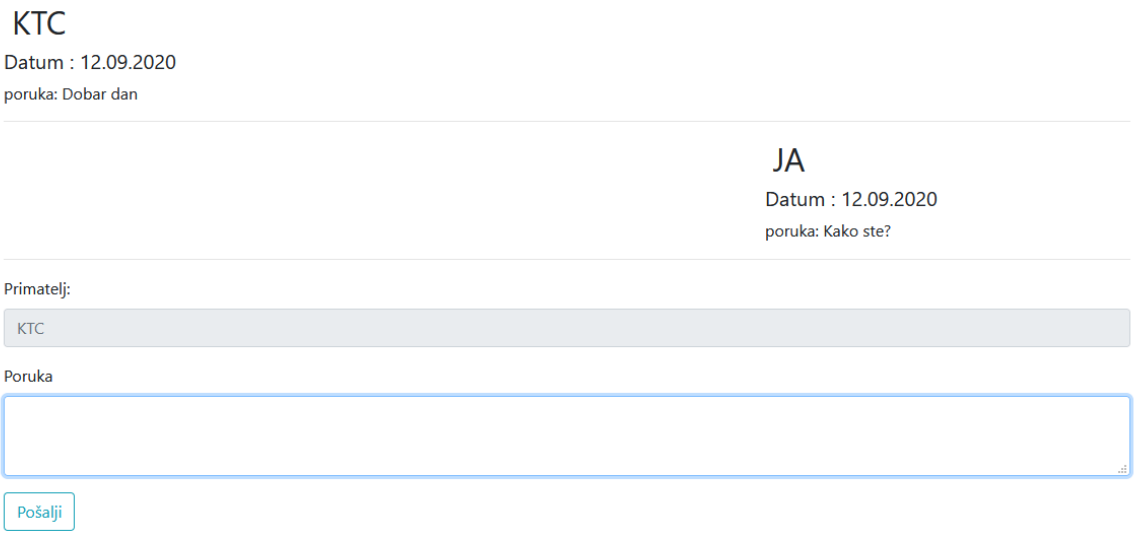
Slika 37: Povijest prodaje (Izvor: autorski rad)

Na slici 37 prikazana je povijest prodaje prijavljenog korisnika vrste OPG, a svaka prodaja sadrži podatke o proizvodu, količini, cijeni, prijevozu, datumu kupnje i samom kupcu. Sljedeća funkcionalnost je prikaz svih Centara što je prikazano na slici 38.



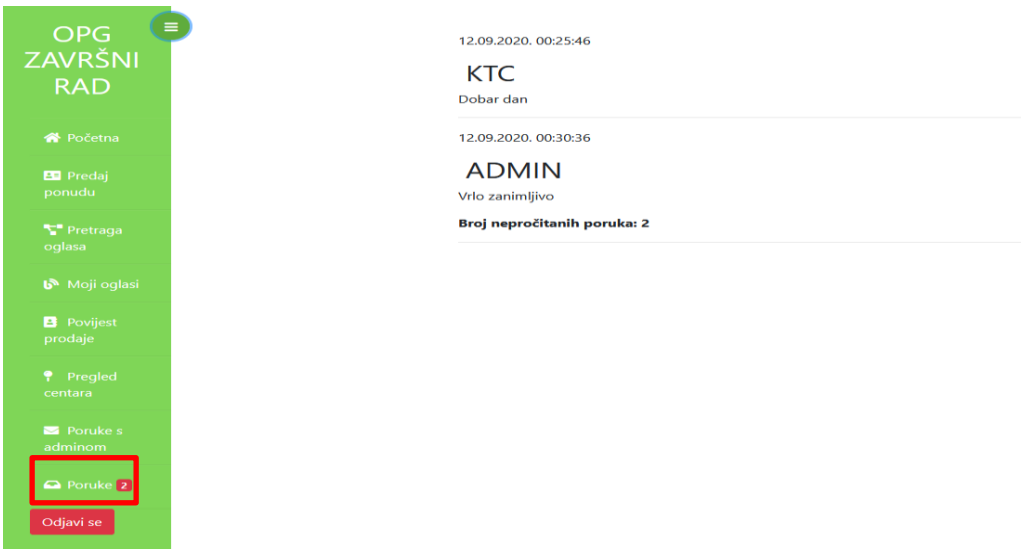
Slika 38: Popis Centara (Izvor: autorski rad)

U tabličnom prikazu se prikazuje slika i ispisuje naziv i adresa centra kao i broj međusobno odrađenih poslova te prosječna ocjena Centra u sustavu. Također, omogućeno je ocjenjivanje Centra ocjenom od 1 do 5. Prijavljeni korisnik tipa OPG preko popisa Centra može poslati poruku Centru. Kod ocjenjivanja, onemogućeno je ocjenjivanje korisnika tipa Centar s kojima nismo odradili niti jedan posao (nismo mu prodali niti jedan oglas). Sljedeća funkcionalnost aplikacije je slanje poruka. Klikom na pošalji poruku otvara se nova stranica na kojoj vidimo sve ranije poruke s tim centrom (slika 39).



Slika 39: Slanje poruke između Centara i OPG-a (Izvor: autorski rad)

Na slici 39 vidimo kako možemo poslati poruku Centru kao i primiti poruku od samog Centra. Korisnik može primiti i poslati više poruka uzastopno. Prijavljenom korisniku umjesto njegovog korisničkog imena prikazuje naslov 'JA'. Također, prikazuju se i datum i vrijeme slanja poruke. Ova funkcionalnost implementirana je pomoću WebSocket-a i Flask ekstenzije Flask-SocketIO. Ako dobijemo poruku, a ne nalazimo se na stranici za razmjenu poruka s korisnikom koji je poslao poruku, tada se poruka smatra nepročitanom. Notifikacijom u navigaciji pod opcijom „Poruke“ obaviješteni smo o broju nepročitanih poruka. Klikom na opciju „Poruke“, otvara nam se stranica na kojoj vidimo posljednju poruku koja nam je poslana od pojedinog korisnika te broj nepročitanih poruka po korisniku. Kako to izgleda možemo vidjeti na slici 40.

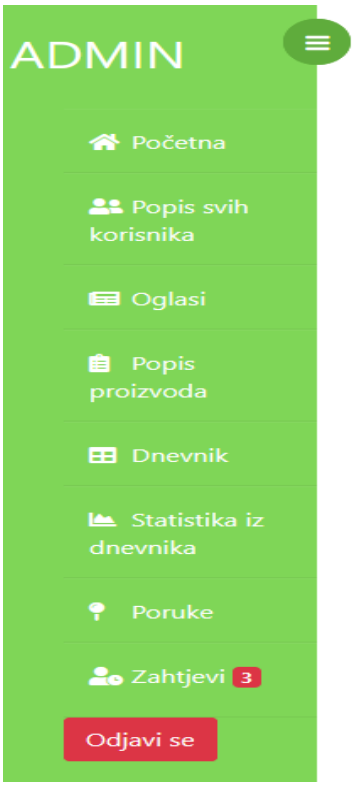


Slika 40: Prikaz primljenih poruka i notifikacija (izvor: autorski rad)

Slika 40 prikazuje stranicu s dobivenim porukama. Valja naglasiti kako se i ova stranica ažurira dinamički. Primjerice, Centar KTC nam pošalje novu poruku kad se nalazimo na ovoj stranici i ažurirat će se broj nepročitanih poruka. Također, ako nam neki drugi korisnik vrste Centar pošalje poruku, dodat će se novi zapis na stranicu sa svim podacima. Automatski se ažurira i broj nepročitanih poruka u notifikacijskom okviru koji je prikazan pravokutnim područjem.

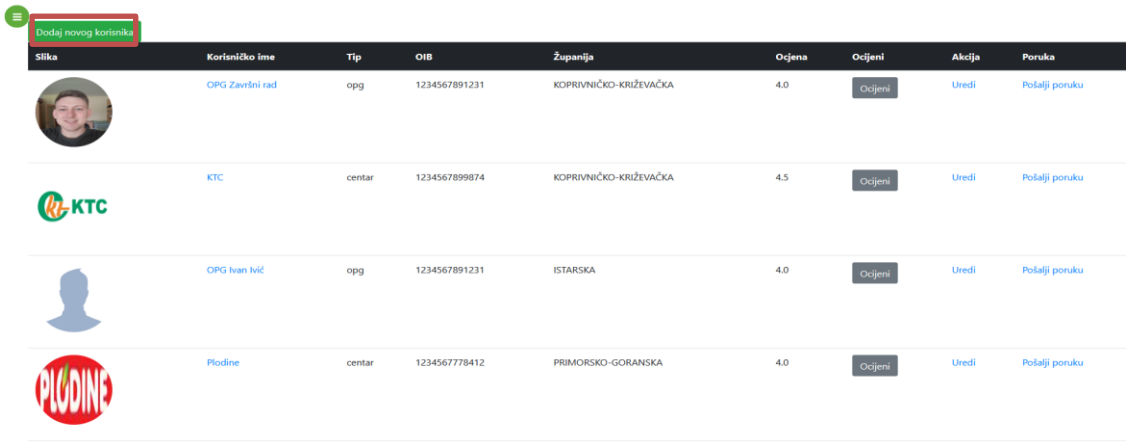
6.4. Uloga administrator

Nakon prijave, administrator u navigaciji vidi početnu stranicu, popis svih korisnika, oglase, popis proizvoda, dnevnik, statistiku iz dnevnika, poruke i zahtjeve. Administrator može sve što i registrirani korisnici. Uz to, može dodavati i ažurirati korisnike sustava. Vidi sve oglase bez obzira je li ih objavio registrirani korisnik vrste OPG ili vrste Centar. Također, vidi popis proizvoda, a može i dodati novi proizvod. Sljedeća funkcionalnost administratora koja se razlikuje od registriranoga korisnika je pregled korisničkih radnji iz dnevnika. Također, administrator vidi i statističke podatke iz dnevnika. Uz to, administratoru je dozvoljeno slanje poruka objema vrstama registriranih korisnika. Na kraju, administrator upravlja zahtjevima neregistriranih korisnika. Navigacija prikazana je na slici 41.



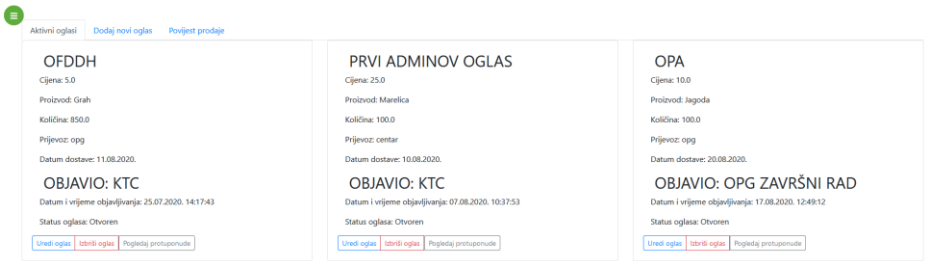
Slika 41: Navigacija administratora (izvor: autorski rad)

Na slici 41 vidimo navigaciju administratora. Da bi se olakšalo kretanje kroz aplikaciju, smanjen je broj opcija na navigaciji. Dodatna horizontalna navigacija dostupna je u nekim opcijama. Sljedeća funkcionalnost administratora je pregled korisnika sustava. Na slici 42 je prikazana tablica s popisom korisnika.



Slika 42: Prikaz popisa korisnika (Izvor: autorski rad)

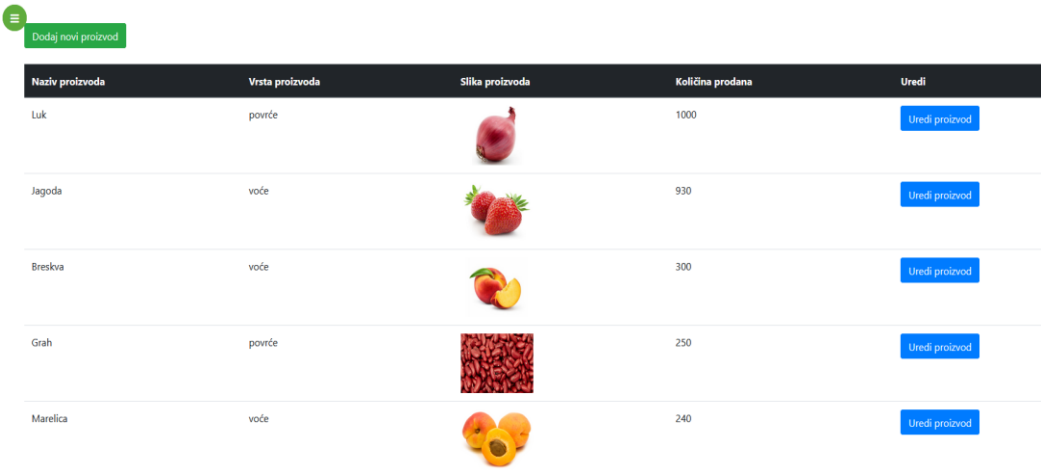
Klikom na korisničko ime u tablici otvara se stranica sa statistikom korisnika koji je odabran, odnosno vidi se njegova statistika iz početne stranice za tog korisnika. Može i uređivati korisnika te poslati poruku korisniku. Kod uređivanja, otvara se nova stranica i obrazac koja je popunjena trenutnim podacima o korisniku. Na tom dijelu može blokirati korisniku pristup do aplikacije. Klikom na gumb označen pravokutnim područjem, odlazi na obrazac za dodavanje novog korisnika koji izgleda kao obrazac na slici 30. Pod opcijom „Oglasi“, administratoru se najprije otvaraju aktivni oglasi Centara i OPG-ova što je prikazano na slici 43.



Slika 43: Prikaz opcije „Oglasi“ (Izvor: autorski rad)

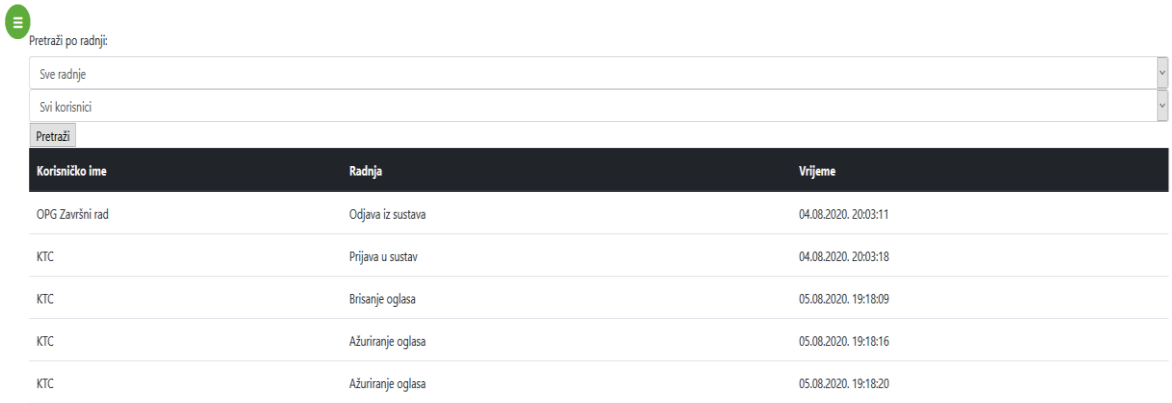
Kao što vidimo na slici 43, može uređivati, brisati, te pregledavati protuponude za aktivne oglase. Slično kao i registrirani korisnici, samo što administrator to može raditi za sve vrste registriranih korisnika. Drugom poveznicom na horizontalnoj navigaciji. „Dodaj novi oglas“ može dodati novi oglas kao i svi korisnici, pri čemu dodatno odabire korisnika umjesto kojeg objavljuje oglas.

Tako da je obrazac jednak kao na slici broj 33 uz dodatni padajući izbornik korisnika za kojeg želi objaviti taj oglas. Također Putem poveznice „Povijest prodaje“ vidi povijest prodaje svih oglasa. Sljedeće što administrator može jest upravljati proizvodima što je prikazano na slici 44.



Slika 44: Prikaz popisa svih proizvoda (Izvor: autorski rad)

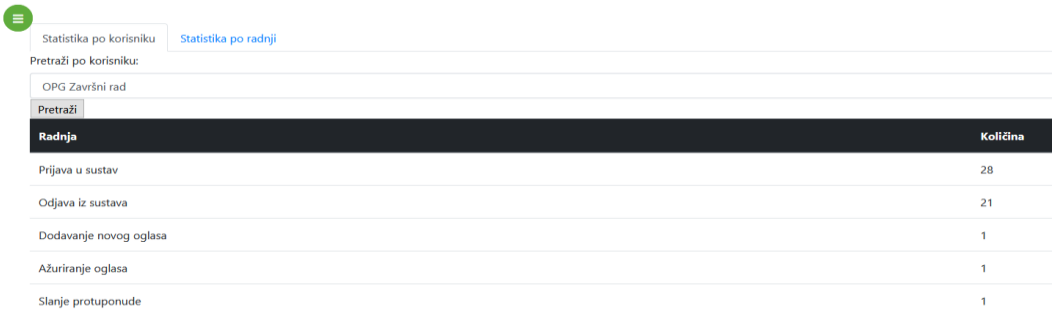
Vidi popis svih proizvoda, zajedno s prodanom količinom te može dodavati i ažurirati proizvode. Ovdje valja istaknuti kako se popis proizvoda ne generira tradicionalnim putem preko Jinje, već je korišten RESTful servis i dohvat podataka odvija se preko JavaScripta. Nadalje, administrator pod opcijom „Dnevnik“ vidi sve radnje koje su korisnici radili u aplikaciji i u koje vrijeme što je prikazano na slici 45.



Slika 45: Prikaz korisničkih radnji iz dnevnika (Izvor: autorski rad)

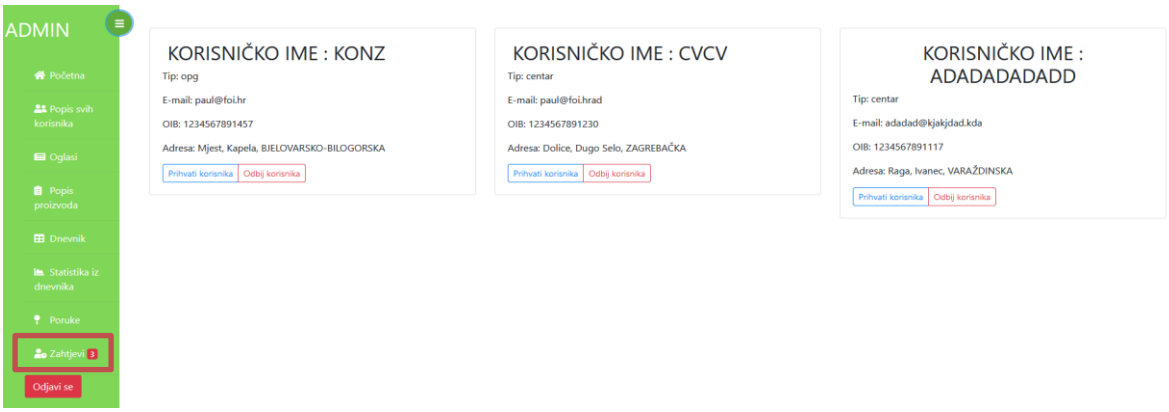
Radnje koje se bilježe u sustavu jesu: prijava u sustav, odjava iz sustava, objavljivanje novoga oglasa, ažuriranje oglasa, kupnja oglasa, brisanje oglasa, slanje protuponude, prihvaćanje protuponude i odbijanje protuponude. Podatke može filtrirati prema radnji i/ili prema korisniku koje odabire iz padajućeg izbornika.

Razlog tome je taj da bi se time narušila potreba za korištenjem nerelacijskih baza podataka već samo dnevnik. Radnje dobivamo upitom u kojem odabiremo polje 'radnja' u zapisima iz dnevnika bez duplikata (engl. *Distinct*). Administrator još može pregledavati statističke podatke iz dnevnika. Statistički podaci dijele se na statistiku prema korisniku ili prema korisničkoj radnji. Za statistiku prema korisniku, za određenog korisnika prikazuju mu se radnje koje je taj korisnik napravio te broj radnji. Za statistiku prema radnji, prikazuje mu se svi korisnici koji su napravili tu radnju i koliko puta su je napravili. Na slici 46 prikazana je statistika po korisniku.



Slika 46: Prikaz statistike po korisniku (Izvor: autorski rad)

Posljednja, ali ne manje važna mogućnost administratora je rukovanje zahtjevima neregistriranih korisnika što je prikazano na slici 47.



Slika 47: Prikaz zahtjeva za kreiranjem računa (Izvor: autorski rad)

Odabirom opcije „zahtjevi“ otvara se stranica gdje su prikazani svi zahtjevi koji nisu niti prihvaćeni niti odbijeni. Može odlučiti želi li prihvatiti zahtjev korisnika ili odbiti. Ako prihvati zahtjev, korisnik koji je kreirao zahtjev na svoj mail dobiva potvrdu o prihvaćanju zahtjeva te mu se dodjeljuje automatski generirana lozinka s kojom se može prijaviti u sustav. Također, u navigaciji se prikazuje broj neodgovorenih zahtjeva. Taj dio prikazan je pravokutnim područjem.

7. Zaključak

Cilj ovog rada bio je prikazati programski jezik Python kao jezik za izradu web aplikacija. Na početku je analizom plaća programera dana motivacija zašto bi Python bio dobar izbor za buduće programere. Nadalje, primjerima u radu prikazana je sintaksa Pythona. Uz to, opisano je kako Python komunicira s nerelacijskom bazom podataka MongoDB. Razlog tome je što su nerelacijske su baze podatka sve popularnije. Također, opisani su i uspoređeni programski okviri za web programiranje u Pythonu. Najpopularniji web programski okviri za razvoj web aplikacije u Pythonu su Flask i Django. U usporedbi s drugim jezicima na strani poslužitelja, Python posjeduje velik broj vanjskih paketa i biblioteka te posjeduje iznimno jednostavnu sintaksu što omogućuje brži razvoj aplikacija. Na kraju samog rada prikazana je i izrađena aplikacija u kojoj su demonstrirane mogućnosti programskog jezika Python i okvira Flask. Programski okvir Flask odabran je za izradu aplikacije jer je vrlo jednostavan za učenje. Naravno, kod izrade aplikacije koristile su se i neophodne tehnologije poput HTML-a, CSS-a i JavaScripta. U aplikaciji je kroz primjere prikazano korištenje proširenja u Flasku. Time se je pokazala jednostavnost korištenja WebSocket-a i generiranje obrazaca tom programskom okviru. Iako u Pythonu postoje brojni jezici za predloške, poput DTL-a ili Maka, kod izrade aplikacije korišten je jezik za predloške Jinja2. Jinja2 dolazi zajedno s Flaskom i zato je najbolji izbor za uređivanje predložaka. Moderne web aplikacije teško je zamisliti bez web servisa, pa su i oni prikazani u radu, kao i otkrivanje i rukovanje pogreškama. Također, demonstrirano je i jedinično testiranje koje je vrlo važno provjeru ispravnosti nekih funkcionalnosti koje se mijenjaju prilikom razvoja aplikacije. Nakon izrade funkcionalne aplikacije, može se zaključiti kako sam razvoj web aplikacija u programskom jeziku Python omogućuje programeru da maksimalno skrati vrijeme razvoja jer omogućuje korištenje raznih proširenja. Sukladno tome, može se zaključiti kako je Python zbog svoje popularnosti i velike zajednice definitivno dobar programski jezik za razvoj web aplikacija.

Popis literature

[1] G. Popović, JavaScript i DOM [Završni rad]. Fakultet organizacije i informatike, Varaždin, Sveučilište u Zagrebu, 2014..

[2] MDN contributors, „Fetch API“, 2020. [Na internetu]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API [pristupano 14.09.2020.].

[3] MongoDB (bez dat.) *The database for modern applications* [Na internetu]. Dostupno: https://www.mongodb.com/ [pristupano 10.08.2020.]

[4] R. Gibb, "What is a Distributed System?", [Blog post]. 2019. [Na internetu]. Dostupno: https://blog.stackpath.com/distributed-system/ [pristupano 10.08.2020.]

[5] E. Horowitz, (13.02.2018.) "MongoDB in 5 Minutes" *Youtube* [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=EE8ZTQxa0AM [pristupano 10.08.2020.]

[6] „Web Application“, (2014). Techterms. Dostupno: https://techterms.com/definition/web_application [pristupano: 14.09.2020.]

[7] PYPL –službeni podaci o popularnosti programskih jezika [Slika] (bez dat.). Dostupno: http://pypl.github.io/PYPL.html [pristupano 30.06.2020.]

[8] T. Grubišić, *Analiza plaća developera za 2019. godinu* [Slika] (bez dat.). Dostupno: https://www.developeri.place/pdf [pristupano 30.06.2020.]

[9] "What Should I Learn as a Beginner: Python 2 or Python 3?" (08.08.2018.) Learn to Code With Me [Na internetu]. Dostupno: https://learntocodewith.me/programming/python/python-2-vs-python-3/ [pristupano 07.08.2020.]

[10] Python Package Indeks (PYPI) (bez dat.) PYPI. Dostupno: https://pypi.org/ [pristupano 30.06.2020.]

[11] M. Lutz, Learning Python, Fifth Edition. United States of America, O'Relly Media. 2013.

[12] MongoDB (bez dat.) *MongoDB Compass* [Na internetu]. Dostupno: https://www.mongodb.com/products/compass [pristupano 10.08.2020.]

[13] PyMongo (bez dat.) *PyMongo 3.11.0 Documentation* [Na internetu]. Dostupno: https://pymongo.readthedocs.io/en/stable/ [pristupano 10.08.2020.]

[14] "Library" (2017.) u *Computer Hope.* Dostupno: https://www.computerhope.com/jargon/l/library.htm [pristupano 10.08.2020.]

[15] M. Waseem, "Python Frameworks: What Are The Top 5 Frameworks In Python?", [Blog post]. 2020. [Na internetu]. Dostupno: https://www.edureka.co/blog/python-frameworks/ [pristupano 30.06.2020.]

[16] Full Stack Python (bez dat.) *Template Engines* [Na internetu]. Dostupno: https://www.fullstackpython.com/template-engines.html [pristupano 10.08.2020.]

[17] Jinja (bez dat.) *Jinja* [Na internetu]. Dostpuno: https://jinja.palletsprojects.com/en/2.11.x/ [pristupano 10.08.2020.]

[18] Django (bez dat.) *Meet Django* [Na internetu]. Dostupno: https://www.djangoproject.com/start/overview/ [pristupano 13.08.2020.]

[19] C. de la Guardia, Python Web Frameworks. United States of America, O'Relly Media, 2016.

[20] WSGI (bez dat.) *What is WSGI?* [Na internetu]. Dostupno: https://wsgi.readthedocs.io/en/latest/what.html [pristupano 10.08.2020.]

[21] Ably (bez dat.) *Long Polling – Concepts and Considerations* [Na internetu]. Dostupno: https://www.ably.io/concepts/long-polling [pristupano 10.08.2020.]

[22] R. Brown, "Django vs Flask vs Pyramid: Choosing a Python Web Framework", [Blog post]. (bez dat.) [Na internetu]. Dostupno: https://www.airpair.com/python/posts/django-flask-pyramid [pristupano 10.08.2020.]

[23] Y. Petlovana, "Top 13 Python Web Framewroks to Learn in 2020", [Blog post]. (bez dat.) [Na internetu]. Dostupno: https://steelkiwi.com/blog/top-10-python-web-frameworks-to-learn/ [pristupano 03.09.2020.]
[24] "Python's Web Framework Benchmarks" (bez dat.) GitHub [Na internetu]. Dostupno:https://klen.github.io/py-frameworks-bench/07.2015.html#results [pristupano 03.09.2020.]
[25] N. Hunt-Wlaker, "An introduction to the Tornado Python web app framework", [Blog post]. 2018. [Na internetu]. Dostupno: https://opensource.com/article/18/6/tornado-framework [pristupano 03.08.2020.]
[26] A. Goel, "Python vs PHP in 2020", [Blog post]. 2020. [Na internetu]. Dostupno: https://hackr.io/blog/python-vs-php?fbclid=IwAR2V7aK4fBpBhDTcBa8LwADsRUPqncOkw09i6wbBjQP_oUGZ5YWumSgunmM [pristupano 01.09.2020.]
[27] "The Good and the Bad of Java Programming" (09.08.2018.) Altexsoft [Na internetu]. Dostupno: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-java-programming/ [pristupano 10.08.2020.]
[28] A. Stempniak, "Pytjon vs. Java: Comparing the Pros, Cons, and Use Cases", [Blog post]. (bez dat.) [Na internetu]. Dostupno: https://www.stxnext.com/blog/python-vs-java-comparison/ [pristupano 01.09.2020.]
[29] The Pallets Projects (bez dat.) *Werkzeug* [Na internetu]. Dostupno: https://palletsprojects.com/p/werkzeug/ [pristupano 10.08.2020.]
[30] The Pallets Projects (bez dat.) *Modular Applications with Blueprints* [Na internetu]. Dostupno: https://flask.palletsprojects.com/en/1.1.x/blueprints/ [pristupano 09.08.2020]
[31] *Flask Web Development.* [Brošura] (bez dat.) M. Grinberg. O'Reilly. Dostupno: https://coddyschool.com/upload/Flask_Web_Development_Developing.pdf [pristupano 22.08.2020.]
[32] Visual Studio Code (bez dat.) *Flask Tutorial in Visual Studio Code* [Na internetu]. Dostupno: https://code.visualstudio.com/docs/python/tutorial-flask [pristupano 18.08.2020.]
[33] Flask-SocketIO (bez dat.) *Flask-SocketIO* [Na internetu]. Dostupno: https://flask-socketio.readthedocs.io/en/latest/ [pristupano 03.08.2020.]
[34] Internet Engineering Task Force (IETF) (2011.) *The WebSocket Protocol* [Na internetu]. Dostupno: https://tools.ietf.org/html/rfc6455 [pristupano 03.08.2020.]
[35] M. Grinberg, "Designing a RESTful API with Python and Flask", [Blog post]. 2013. [Na internetu]. Dostupno: https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask [pristupano 03.08.2020.]
[36] Software testing fundamentals (bez dat.) Unit Testing? [Na internetu]. Dostupno: http://softwaretestingfundamentals.com/unit-testing/ [pristupano 03.08.2020.]

Popis slika

Popis tablica