FIPUbot: web aplikacija za automatizirano odgovaranje na često postavljena pitanja studenata

Višnjić, Matej

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:552654

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-03**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile

Fakultet informatike u Puli

Matej Višnjić

FIPUbot: web aplikacija za automatizirano odgovaranje na često postavljena pitanja studenata

Završni rad

Sveučilište Jurja Dobrile

Fakultet informatike u Puli

Matej Višnjić

FIPUbot: web aplikacija za automatizirano odgovaranje na često postavljena pitanja studenata

Završni rad

Ime i prezime studenta, JMBAG: Matej Višnjić, 0303092423

Studijski smjer: preddiplomski sveučilišni studij informatika

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Kolegij: Programiranje

Mentor: izv. prof. dr. sc.Tihomir Orehovački

u Puli, rujan 2022.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Matej Višnjić, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

M.Višnjić

U Puli, 20.09.2022

IZJAVA O KORIŠTENJU
AUTORSKOG DJELA

Ja, Matej Višnjić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj Završni rad pod nazivom „FIPUbot: web aplikacija za automatizirano odgovaranje na često postavljena pitanja studenata“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 20.09.2022

Potpis

M.Višnjić

Sadržaj

Sažetak 1

Abstract 1

1. Uvod 2

2. Kritička analiza 3

3. Opis korištenih tehnologija i razvojnog okruženja 6

3.1 Vue.js 6

3.1.1 TailwindCSS 6

3.2 Flask 6

3.3 Raspberry PI 7

3.4 Visual Studio Code 7

3.5 GNU nano i terminal 8

4. Opis funkcionalnosti 10

4.1 Chatbot 12

4.1.1 Flask API 21

4.2 Grafičko sučelje 25

4.3 Raspberry PI 34

5. Zaključak 37

6. Popis literature 38

7. Popis slika 40

Sažetak

Današnje doba gotovo je nezamislivo bez korištenja umjetne inteligencije i automatizacije. Pomoću algoritama za duboko učenje, neuronski mreža i umjetne inteligencije koncipiran je „FIPUbot“. Web aplikacija „FIPUbot“ zamišljena je za automatsko odgovaranje na često postavljena pitanja studenata Fakulteta informatike u Puli. Tema završnog rada bazira se na korištenju tehnologije za izgled i dizajn, korištenje alata za izradu „back-end-a“, te korištenje Raspberry PI računala kao web servera. Napravljen je chatbot koji odgovara na često postavljena pitanja studenata čije su prednosti da brzo i učinkovito odgovara na postavljana pitanja. „FIPUbot-om“ možemo automatizirati upite studenata, te ubrzati proces čekanja studenata na odgovor i umanjiti posao djelatnicima Sveučilišta.

KLJUČNE RIJEČI

Web aplikacija, flask aplikacija, rest api, raspberry pi, umjetna inteligencija, duboko učenje, neuronske mreže

Abstract

Today's era is almost unimaginable without the use of artificial intelligence and automation. The "FIPUbot" web application is designed for the automatic answering of frequently asked questions of the students of the Faculty of informatics in Pula. The topic of the Bachelor's thesis is based on the use of technology for appearance and design, the use of tools for creating a back-end, and the use of a Raspberry PI computer as a web server. A chatbot was created that answers frequently asked questions of students, whose advantages are that it answers questions quickly and efficiently. With "FIPUbot" we can automate inquiries, speed up the process of students waiting for an answer, and reduce the workload of University employees.

KEYWORDS

Web application, flask application, rest api, raspberry pi, artificial intelligence, deep learning, neural networks

1. Uvod

Dokumentacija predstavlja programski modul „FIPUbot“, koji je implementiran pomoću Vue.js Framework-a za korisničko sučelje („front-end“), Tensorflow za izradu „bota“ pomoću dubokog učenja, Flask web Framework za isporuku flask aplikacije („back-end“), te za hosting Flask-a korišten je Raspberry Pi model 4B.

Davne 1989. godine informatičar Tim Barness-Lee izumio je takozvani „World Wide Web“ [1]. U samim početcima Internet nije bio ni približno kao danas. HTML stranice su bile statične, nisu bile u mogućnosti da komuniciraju klijent-server, već je samo server mogao isporučivati klijentu statične poruke. HTML definira značenje i strukturu sadržaja, za izgled koristi CSS, a za funkcionalnosti JavaScript. Davne 1995. godine JavaScript, programski jezik, je osmišljen, što je dovelo do bržeg interneta i mogućnost dodavanja dinamičkih elemenata na stranicu [1]. Nedugo zatim 2005. godine Jesse James Garret je osmislio takozvani pojam Ajax koji opisuje novi pristup korištenju web tehnologija, uključujući HTML, CSS, JavaScript, DOM, XML [2]. Također, isti izvor definira definiciju za Ajax („Asynchronous JavaScript and XML“) kao skup tehnika weba koji koristi različite tehnologije na strani klijenta za stvaranje asinkronih aplikacija.

Tehnologija sve više napreduje i 2014. godine je stvoren HTML5. Podržavao je novu vrstu multimedije i dopuštao web aplikacijama da budu neovisne o platformama i internetskim preglednicima [3]. Nedugo zatim Alex Russel je osmislio „Progressive Web applications“ koje rade kao normalne web stranice, ali isto tako mogu se instalirati na mobilne uređaje i koristiti push notifikacije, raditi bez interneta i pristupati uređaju [4]. „FIPUbot“ iako ima neke značajke kao progresivne web aplikacije, no nije progresivna. Naravno, aplikacija je responzivna, brza i vrlo je ugodno ju koristiti i na mobilnom uređaju.

Sredinom prošlog stoljeća Alan Turing bio je znanstvenik i proučavao je mogućnost umjetne inteligencije, njegov prijedlog je bio da strojevi rade istu stvar kao i ljudi, to jest da strojevi isto uče od dostupnih informacija [5]. Inače, „botovi“ su najčešće istrenirani sa velikim skupovima podataka, te korištenje obrade prirodnog jezika (eng. Natural language processing*)* i dubokog učenja i neuronskih mreža stvaraju svoj „um“. U današnjem dobu, sve više stvari se automatizira, ne samo kod web aplikacija već i u drugim društvenim djelatnostima. Danas je gotovo nemoguće izbjeći „chatbot-ove“, svaka modernija aplikacija za kupovinu i slično, ima svog „bota“. Zbog toga „FIPUbot“ je koncipiran kako bi automatizirano odgovarao na često postavljena pitanja studenata na Fakultetu informatike u Puli. Samim time, znatno bi olakšao

neželjene poruke upućene nastavnicima i službama na fakultetu. „Bot“ je spreman i odgovoriti na neka druga pitanja, ovisno kako su pitanja postavljena od strane klijenta. Odgovore koje daje su nasumični i statični, što znači da za nekoliko istih upita, može dati i iste odgovore.

U sljedećim poglavljima je opisana kritička analiza, korištene tehnologije i razvojno okruženje, opis funkcionalnosti te zaključak.

„Back-end“ projekta nalazi se na Github repozitoriju: https://github.com/mvisnjic/FIPUbot-backend

„Front-end“ projekta nalazi se na Github repozitoriju: https://github.com/mvisnjic/FIPUbot

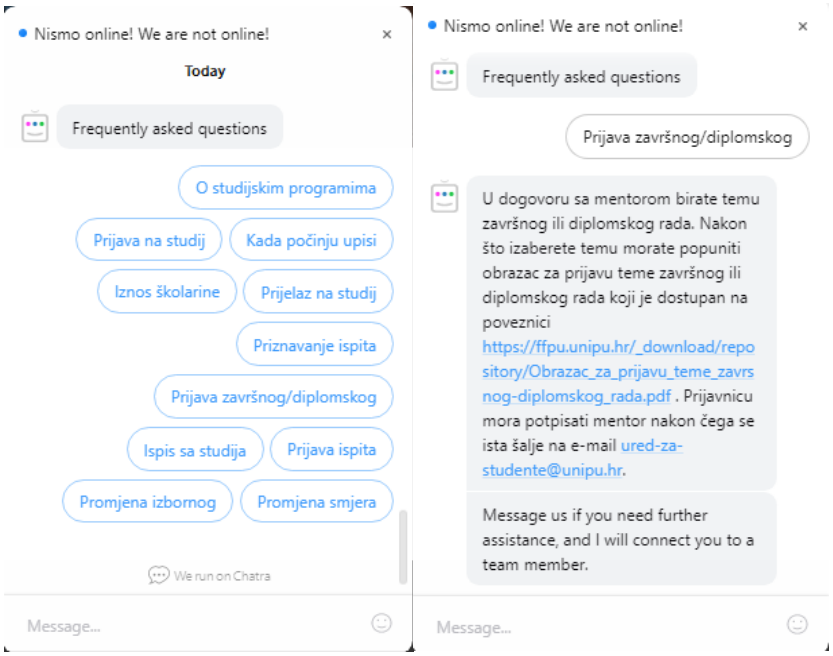
2. Kritička analiza

Tehnologija jako brzo napreduje te sve više stvari u današnjici prelaze na automatizaciju. Time bi se znatno pomoglo korisnicima i zaposlenicima. Također, automatizacije procesa na dugoročno donose velike uštede.

Današnje moderno Sveučilište ima mnogo djelatnika raspoređenih u različite kategorije za obavljanje poslova oko Sveučilišta. Gotovo svaki student, kada dobije status studenta po prvi put, ima uvijek jako puno pitanja. Na početku mu nije jasno kako uopće funkcionira proces studiranja. Većina novih studenata ima strah od postavljanja pitanja i ne žele ometati profesore sa njihovim pitanjima koja smatraju da nisu relevantna.

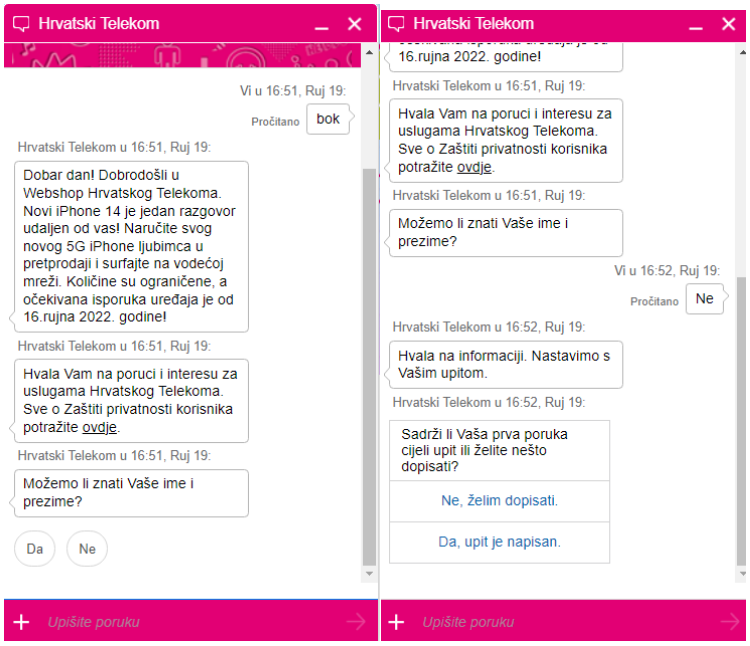
Korištenje automatiziranog i interaktivnog „FIPUbot-a“ koji ne samo da odgovara na općenita pitanja studenata, već pruža interakciju i za rješavanje nekih drugih alternativnih problema studenata, kao na primjer; „popis profesora“, „gdje se nalazi studentski restoran“, „gdje studenti vole izlaziti“ i mnoge druge ključne riječi koje se mogu i naknadno dodavati u skup podataka i imaju veze sa studentskim životom.

Trenutni „chatbot“ koji se nalazi na stranicama Sveučilišta fakulteta u Puli, koristi nekoliko statičnih najčešćih pitanja. Ukoliko korisnik nije zadovoljan odgovorom na odabrano pitanje, „bot“ mu pruža da šalje poruku direktno djelatnicima Sveučilišta. To pruža direktnu interakciju sa osobom koja je specificirana za to područje i može dobiti povratni odgovor ubrzo ukoliko, naravno, nije vikend ili korisnik nije postavio upit izvan radnog vremena osobe koja je djelatnik Sveučilišta. Na slici broj 1 prikazuje izgled trenutnog „chatbota“ sa Sveučilišta i odgovor ukoliko je pritisnut neki od statičnih odgovora.



Slika 1. „Chatbot“ Sveučilišta Jurja Dobrile u Puli

Vrlo sličan primjer „chatbota“ koristi i vodeći Hrvatski telekomunikacijski operater , Hrvatski telekom [19]. „Bot“ je isto statičan za početna pitanja, te ukoliko nastavljate daljnju komunikaciju, ima mogućnost da se priključi djelatnik korisničke službe. Vrijeme za rješavanje problema može se odužiti i do 15 minuta. S obzirom na to, da u današnje vrijeme ljudi ne vole dugo čekati, te vrlo brzo izgube strpljenje. Slika 2 prikazuje „chatbot-a“ Hrvatskog telekoma.



Slika 2. Chatbot“ Hrvatskog Telekoma

Cleverbot, „bot“ koji je je stvoren još u prošlome stoljeću i koristi umjetnu inteligenciju kako bi mogao vršiti komunikaciju sa klijentom [20]. Klijent sa „botom“ može razgovarati o mnogočemu. Također, ima dvije tipke „think about it“ i „think for me“, koje klijent koristi ukoliko ne zna što da napiše. Cleverbot podržava glasovno raspoznavanje, te nudi opcije za dijeljenje dosadašnjeg razgovora pomoću e-pošte, Facebooka, Twittera, Google+ i raznim društvenim mrežama. Slika 3 prikazuje Cleverbot i dosadašnju komunikaciju sa „botom“.

Slika 3. Prikaz Cleverbota i povijest komunikacije

S obzirom na spomenuta i prikazana slična rješenja kao „FIPUbot“, prva dva „bota“ koriste statične odgovore za početnu interakciju, dok sa Cleverbot-om se može započeti komunikacija odmah. „FIPUbot“ najviše ima sličnosti sa Cleverbot-om.

3. Opis korištenih tehnologija i razvojnog okruženja

Korištene tehnologije su Vue.js i TailwindCSS za korisničko sučelje, Flask za implementaciju „back-end-a“, razne biblioteke za skup podataka i treniranje „bota“. Na produkciji korišten je Raspberry PI za izvedbu Flask aplikacije, a za korisničko sučelje web server koji je u vlasništvu „SETCOR d.o.o.“ [8]. Kao uređivač teksta (eng. Code editor*)* korišten je Visual studio Code za Vue.js i Flask, te je korišten terminal i GNU nano za kontrolu nad Raspberry PI-em.

3.1 Vue.js

„Vue.js“ je JavaScript framework koji služi za izgradnju izgleda i funkcionalnosti korisničkih sučelja. Najosnovnija 3 svojstva su da je pristupačan, izvodljiv i svestran, što znači da se nadovezuje na standardni HTML,CSS i JavaScript, reaktivan je sustav i rijetko zahtjeva ručnu optimizaciju te je bogat i potpuno prilagodljiv sustav [9]. CSS (Cascading Style Sheets) je stilski jezik koji služi za izgradnju izgleda pojedinih HTML elemenata.

3.1.1 TailwindCSS

Izgradnja izgleda aplikacije se odvija pomoću „TailwindCSS“. „TailwindCSS“ je CSS framework koji omogućuje programerima da vrlo jednostavno i uz srednje poznavanje CSS-a mogu vrlo brzo i razumljivo stvarati nove izglede. „Tailwind“ se koristi pomoću naziva klasa, koje su postavljene na engleskom jeziku i vrlo razumljivo je kada i kako koristiti određenu klasu. Više o tome biti će objašnjeno u poglavlju Opis funkcionalnosti.

3.2 Flask

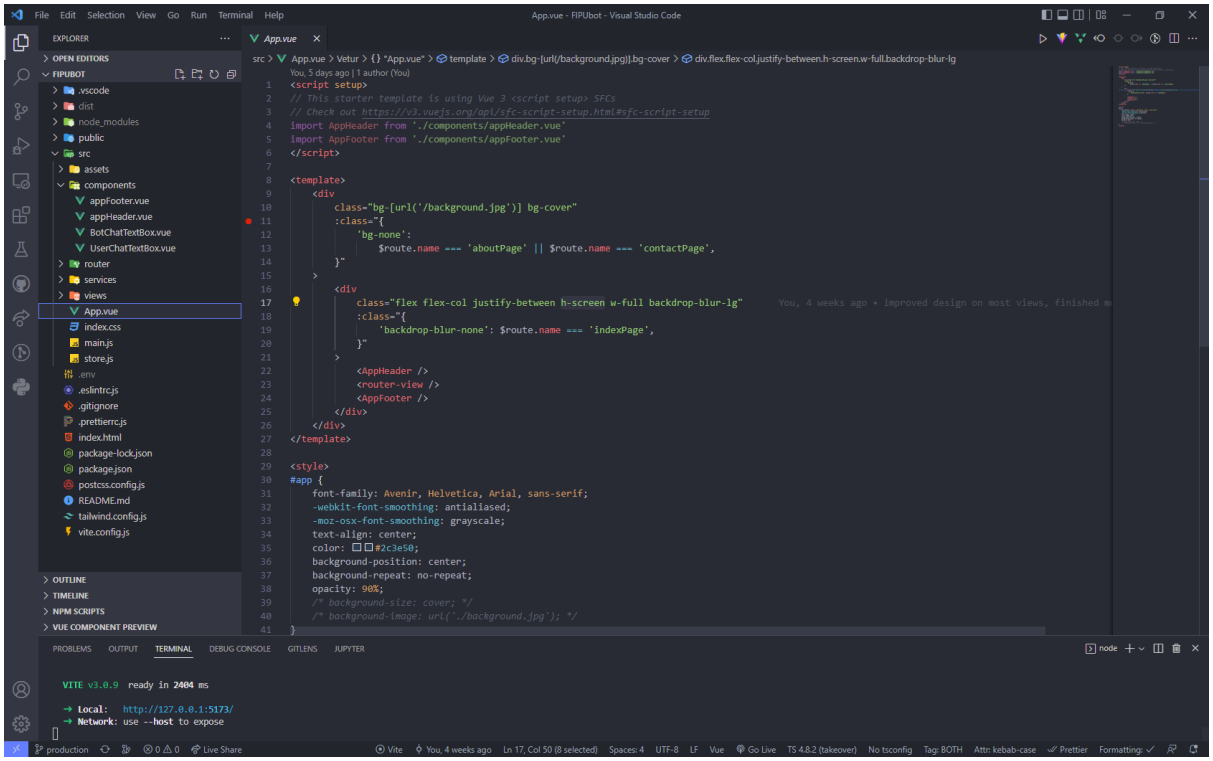
Python je interpretirani, objektno orijentiran programski jezik i ima vrlo jednostavnu sintaksu koju je lako naučiti [10]. Isto tako, podržava module i biblioteke, što potiče modularnost programa i korištenje koda iznova. Flask je jedan od mnogih Python-ovih freamwork-a, a omogućuje izgradnju web aplikacija. Trenutno, Flask služi za isporuku „back-end-a“, točnije za slanje odgovora od „back-end“ servera prema serveru koji isporučuje „front-end“.

3.3 Raspberry PI

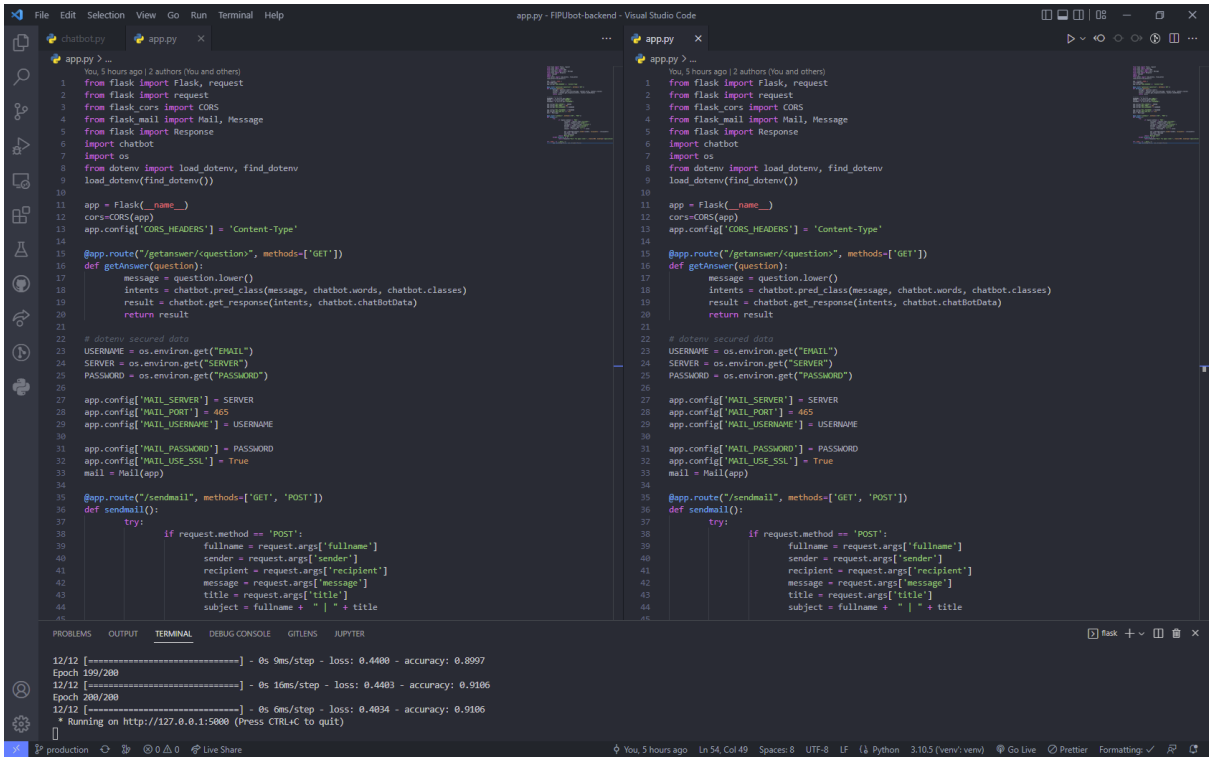
Raspberry PI je jeftino računalo veličine kartice koje omogućuje istraživanje računalstva i učenje programskih jezika, a može se koristiti i kao normalno računalo [6]. Raspberry PI danas je vrlo popularan i njegova svojstva se primjenjuju kod Internet stvari (IoT), robotike, izrada kućnih web servera, osobnog oblaka za pohranu podataka, izrada pametne kuće i još mnogih drugih mogućnosti. Raspberry PI koristi svojstveni operacijski sustav, koji je baziran na Linux distribuciji, Debian.

3.4 Visual Studio Code

Visual Studio Code, skraćeno VS Code, je uređivač teksta specijaliziran za Windows, Linux i MacOs. Uređivač ima mnoge mogućnosti i najrasprostranjeniji je tekstualni uređivač. Automatski prepoznavanje i dovršavanje koda, nudi otklanjanje greški u kodu (eng. Debugging), ima u sebi ugrađen komande za upravljanje Github-om, sadržava jako puno dodatnih proširenja kako bi pisanje koda bilo još učinkovitije [11]. Slika 4 prikazuje uređivač teksta gdje se razvija grafičko sučelje, a slika 5 prikazuje razvijanje Flask aplikacija.



Slika 4. VS code – razvijanje izgleda aplikacije

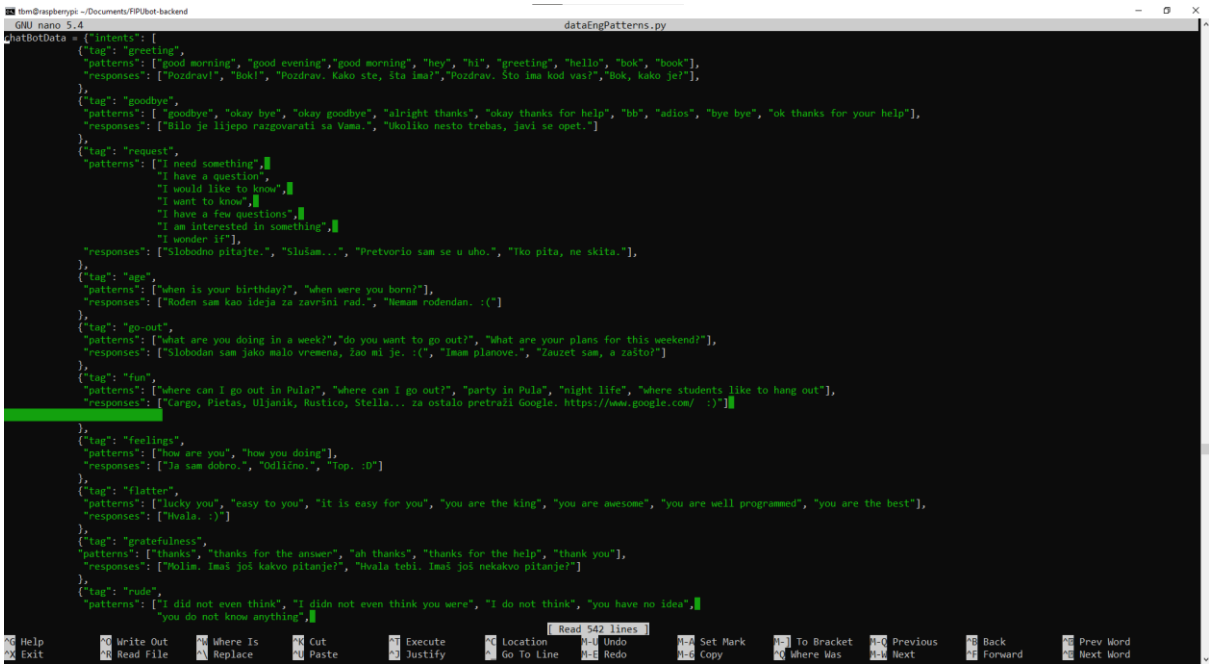


Slika 5. VS code – razvijanje „chatbota“ aplikacije

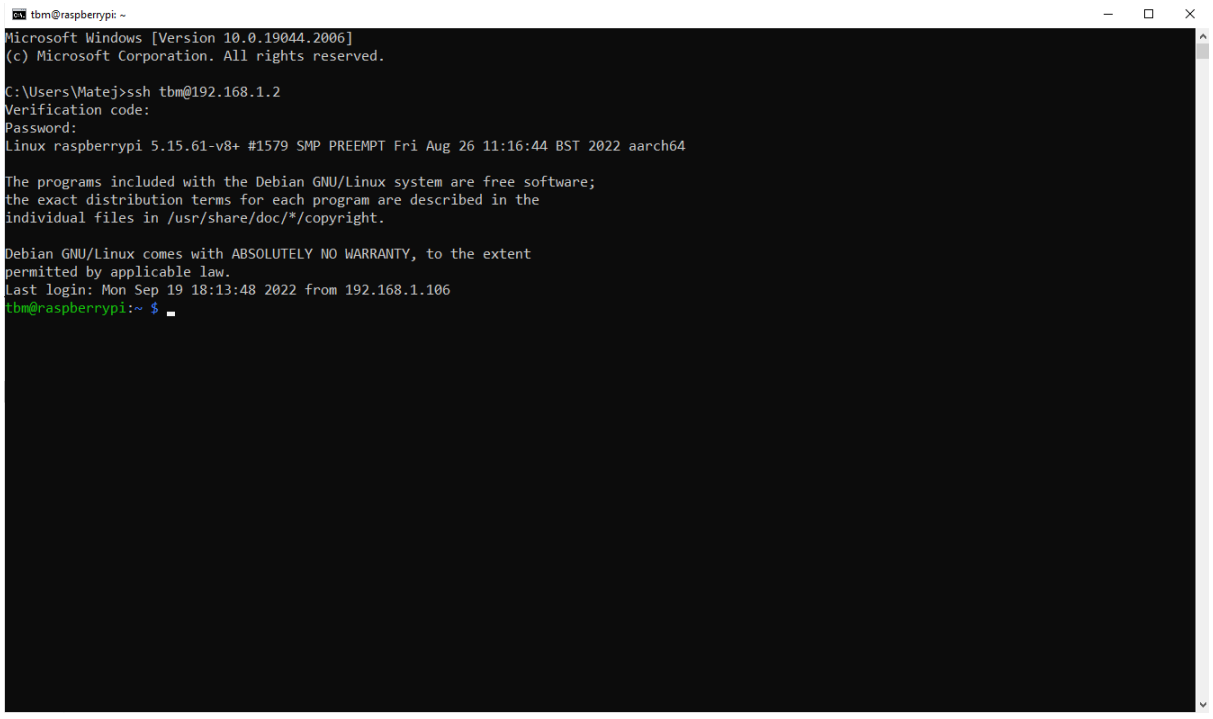
3.5 GNU nano i terminal

GNU nano je uređivač teksta koji se koristi kod sustava baziranih na Unix sustavu. Sadržava osnove operacije nad uređivanjem teksta i koristi se pogotovo kod SSH konekcije sa računala na računalo. Slika 6 prikazuje korištenje GNU nano uređivača teksta nad skupom podataka koje koristimo u „botu“.

Terminal je sučelje kojim se može upravljati računalom. Svaki operacijski sustav ima svoju vrstu terminala kojim može upravljati svojim računalom. Slika 7 prikazuje SSH spajanje i terminal za upravljanje Raspberry PI računalom.



Slika 6. GNU nano uređivač teksta

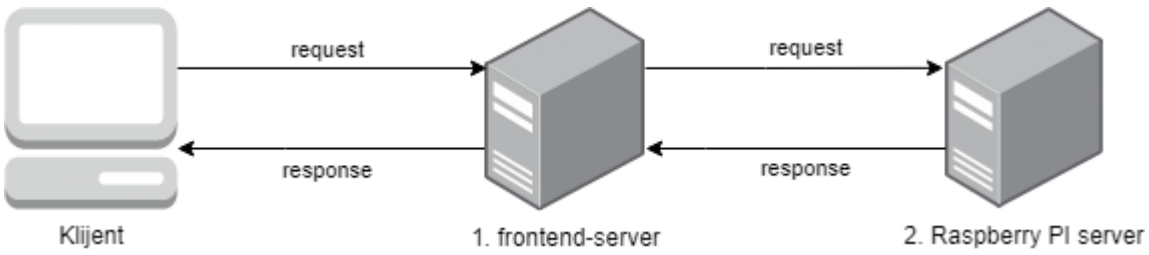


Slika 7. SSH konekcija i prikaz terminala

4. Opis funkcionalnosti

U ovome poglavlju prikazati će se sve funkcionalnosti koje aplikacija nudi, opis cjelokupne web aplikacije koja je opisana u uvodu.

Na slici broj 8 prikazana je skica izvedba aplikacije, grafički je predstavljeno kako aplikacija funkcionira.



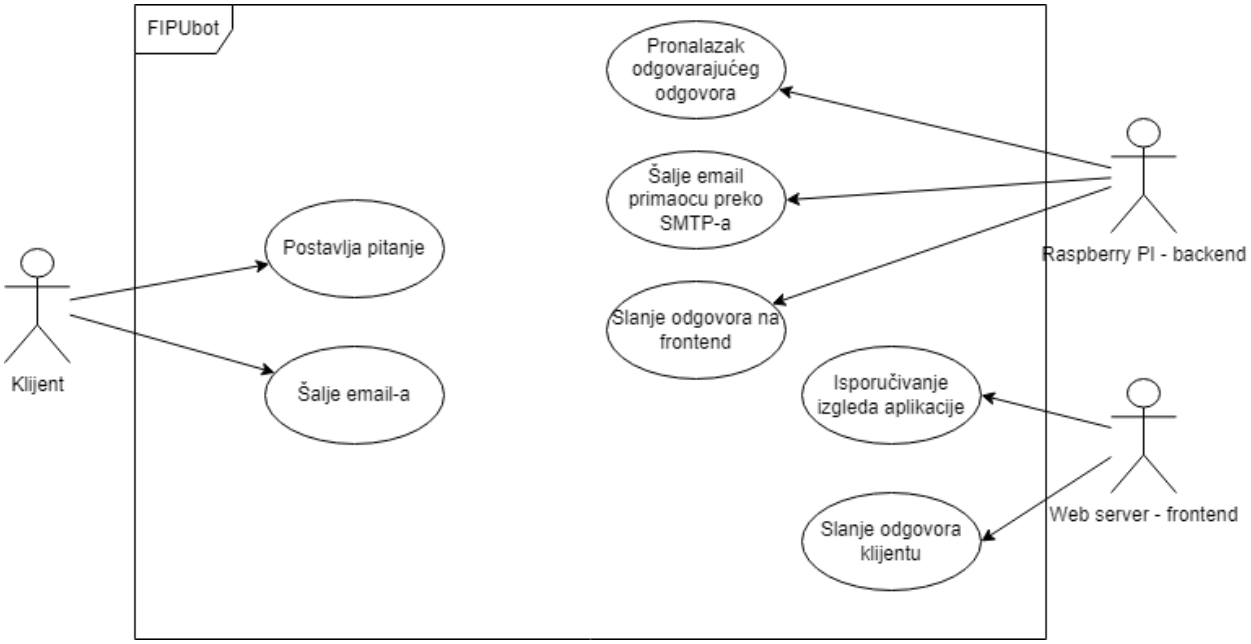
Slika 8. Prikaz skice rada aplikacije

Aplikacija je postavljena na dva različita web servera. Prvi server je u najmu i sadrži izvedbu korisničkog sučelja, „front-end“. Drugi server se izvodi na Raspberry PI uređaju i sadržava takozvani „back-end“.

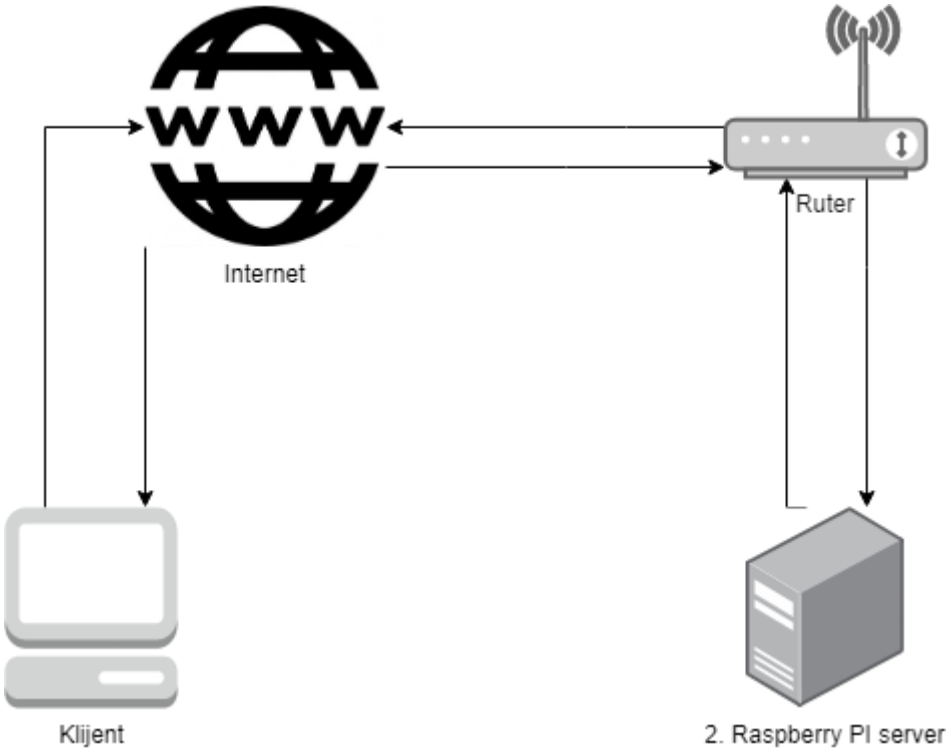
Klijent kada otvori web aplikaciju šalje zahtjev na prvi server. Prvi server vraća korisničko sučelje i klijent može vizualno prepoznati aplikaciju. Zatim, ukoliko klijent želi koristiti „bota“ ili slanje e-pošte, prvi server mora poslati zahtjev na drugi server. Ukoliko drugi server vrati odgovor na prethodni server, prvi server isporučuje klijentu odgovor od servera broj dva.

Klijent je u mogućnosti da postavlja pitanja aplikaciji i ukoliko želi može poslati upit e-poštom direktno na e-mail kreatoru. Nakon što klijent pošalje zahtjev, nebitno koji od dva moguća navedena, taj zahtjev obrađuje prvi server kao što je navedeno u odlomku iznad te ga ona šalje na drugi server. Use case dijagram prikazan je na slici broj 9.

S obzirom da Raspberry PI koristi dinamički DNS, to jest ima svoju domenu koja vodi na IP adresu od rutera na koji je uređaj spojen. Skica je prikazana na slici broj 10, a više o tome biti će objašnjeno na kraju ovog cjeline.



Slika 9. Use case diagram



Slika 10. Skica dinamičkog DNS

4.1 Chatbot

Uz pomoć članka [13] koji opisuje izradu jednostavnog „chatbot-a“ pomoću Python-a i dubokog učenja, izrađena je baza za „FIPUbot“. Kao što je navedeno u članku, za izradu „bota“ potrebno je nekoliko biblioteka, tehnika te ubaciti relevantne podatke kako bi „bot“ što učinkovitije odgovarao na upite klijenata.

Isprva treba kreirati skup podataka što bolje i poslagati ključne riječi kako bi „bot“ mogao što učinkovitije prepoznati upit od strane klijenta. Za svaku vrstu upita, postoji nekoliko različitih varijanti odgovora koji slučajnim odabirom budu odabrani, te poslani natrag klijentu.

Skup podataka je ustvari jedna velika JSON datoteka i svaki objekt ima svoju oznaku. Oznake služe kako bi kasnije „bot“ mogao prepoznavati o kakvom se upitu radi, te vratiti relevantan odgovor. Točnije, datoteka se sastoji od jednog velikog polja objekata, zatim svaki objekt ima po tri atributa: oznaku (tagg), uzorci (patterns) i moguće odgovore (responses). Primjer je prikazan na slici broj 11, koja prikazuje početni dio podataka. Manji dio podataka osmišljen je od strane kreatora aplikacije, dok su ostali podaci preuzeti sa stranice za odgovore na često postavljana pitanja Sveučilišta Jurja Dobrile u Puli [21] i stranice za odgovore na često postavljana pitanja vezano za mobilnost studenata. [22].

```
chatBotData = {"intents": [
            {"tag": "greeting",
             "patterns": ["good morning", "good evening","good morning", "hey", "hi", "greeting", "hello", "bok", "book"],
             "responses": ["Pozdrav!", "Bok!", "Pozdrav. Kako ste, šta ima?","Pozdrav. Što ima kod vas?","Bok, kako je?"],
            },
            {"tag": "goodbye",
             "patterns": [ "goodbye", "okay bye", "okay goodbye", "alright thanks", "okay thanks for help", "bb", "adios", "bye bye", "ok thanks for your help"],
             "responses": ["Bilo je lijepo razgovarati sa Vama.", "Ukoliko nesto trebas, javi se opet."]
            },
            {"tag": "request",
             "patterns": ["I need something",
                          "I have a question",
                          "I would like to know",
                          "I want to know",
                          "I have a few questions",
                          "I am interested in something",
                          "I wonder if"],
             "responses": ["Slobodno pitajte.", "Slušam...", "Pretvorio sam se u uho.", "Tko pita, ne skita."],
            },
            {"tag": "age",
             "patterns": ["when is your birthday?", "when were you born?"],
             "responses": ["Rođen sam kao ideja za završni rad.", "Nemam rođendan. :("]
            },
            {"tag": "go-out",
             "patterns": ["what are you doing in a week?","do you want to go out?", "What are your plans for this weekend?"],
             "responses": ["Slobodan sam jako malo vremena, žao mi je. :(", "Imam planove.", "Zauzet sam, a zašto?"]
            },
            {"tag": "fun",
             "patterns": ["where can I go out in Pula?", "where can I go out?", "party in Pula", "night life"],
             "responses": ["Cargo, Pietas, Uljanik, Rustico, Stella... za ostalo pretraži Google :)"]

            },
            {"tag": "feelings",
             "patterns": ["how are you", "how you doing"],
             "responses": ["Ja sam dobro.", "Odlično.", "Top. :D"]
            },
            {"tag": "flatter",
             "patterns": ["lucky you", "easy to you", "it is easy for you", "you are the king", "you are awesome", "you are well programmed", "you are the best"],
             "responses": ["Hvala. :)"]
            },
            {"tag": "gratefulness",
            "patterns": ["thanks", "thanks for the answer", "ah thanks", "thanks for the help", "thank you"],
             "responses": ["Molim. Imaš još kakvo pitanje?", "Hvala tebi. Imaš još nekakvo pitanje?"]
            },
            {"tag": "rude",
             "patterns": ["I did not even think", "I didn not even think you were", "I do not think", "you have no idea",
                          "you do not know anything",
                          "you do not understand anything",
                          "you are stupid",
                          "you are badly programmed"],
             "responses": ["U redu. :)"]
            },
            {"tag": "say-something",
             "patterns": ["say something"],
             "responses": ["Nešto.", "Nemam ništa!"]
            },
            {"tag": "feelings-answer",
             "patterns": ["you are great?", "I am fine", "I am great", "I am enjoying myself", "I feel great", "here is nothing", "nothing"],
             "responses": ["Bas mi je drago.", "Drago mi je zbog tebe."]
            },
            {"tag": "bored",
             "patterns": ["I am bored", "I do not know what to do", "such a lonely day", "what a boring day"],
             "responses": ["Kako da te zabavim?", "Kako misliš?", "Hmm. Što ćemo sada?", "Idi na https://e-ucenje.unipu.hr/ i zabavi se! :)"]
```

Slika 11. Prikaz skupa podataka

S obzirom da se za rad „chatbota“ koristi duboko učenje pomoću metode neuronskih mreža potrebno je uvesti određene biblioteke. Korištene biblioteke su: string, random, nltk, numpy, tensorflow, keras i googletrans. Datoteka od skupa podataka sadržava oko 11 000 riječi, uključujući oznake, odgovore i uzorke.

Nakon što je skup podataka spreman, potrebno je proći kroz dvije „for“ petlje kako bi se uklonili interpunkcijski znakovi, razmaci i slično. Također, potrebno je napuniti četiri polja sa podacima. U polje words spremljeni su podaci kojima su uklonjeni interpunkcijski znakovi i uklonjeni duplikati. U polju classes dodane su oznake, doc_X sadrži sve ključne riječi iz uzorka a doc_y pridružene oznake za sve uzorke.

Isprva, skup podataka bio je na hrvatskom jeziku. Korištenjem biblioteke za prevoditelj, skup podataka se prevodio sa hrvatskog na engleski jezik. To jest, biblioteka koja je namijenjena za treniranje podataka, ne podržava hrvatski jezik pa je bilo primorano da se riječi prvo prevode na engleski jezik, zatim onda se treniraju. Kreatori biblioteke „googletrans“ navode da; biblioteka je neslužbena i nije povezana sa Google-om, ograničenje znakova je 15 000 te zbog ograničenja verzije Google prevoditelja ne jamče da biblioteka radi ispravno u svakom trenutku [7]. Usprkos tome, skup podataka, to jest ključne riječi koje prepoznaju upite, moraju biti na engleskom jeziku kako bi trening podataka mogao imati visoku točnost. Zbog razloga gore navedenog, prevoditelj je uveden da upit klijenta prvo prevede sa hrvatskog jezika na engleski. Zatim „bot“ traži ključne riječi koje se podudaraju sa upitom klijenta. Nakon toga, „bot“ vraća odgovor koji je na hrvatskom jeziku i nalazi se u atributu „responses“.

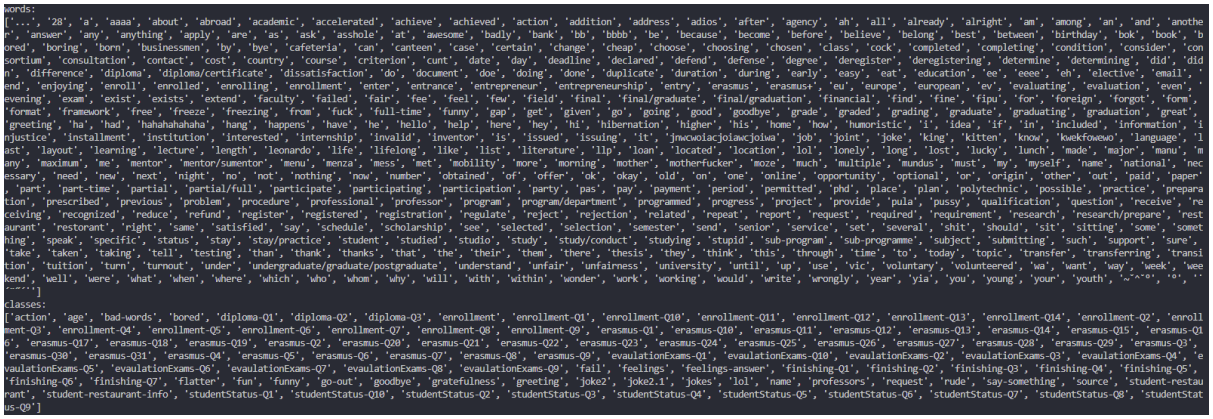
Na slici broj 12 vide se polja, koja služe za punjenje riječi iz skupa podataka, inicijalizacija prevoditelja, te prolazak kroz petlje. Nakon toga riječi se sortiraju i miču se duplikati.

```
import string
import random
import nltk
import numpy as np
from nltk.stem import WordNetLemmatizer
import tensorflow as tf
from keras import Sequential
from keras.layers import Dense, Dropout
from googletrans import Translator
from dataEngPatterns import chatBotData



nltk.download("punkt")
nltk.download("wordnet")
nltk.download('omw-1.4')
# initializing lemmatizer to get stem of words
lemmatizer = WordNetLemmatizer()

#google API translator
translator = Translator(service_urls=[
      'translate.google.com',
    ])

# Each list to create
words = []
classes = []
doc_X = []
doc_y = []
# Loop through all the intents
# tokenize each pattern and append tokens to words, the patterns and
# the associated tag to their associated list
for intent in chatBotData["intents"]:
    for pattern in intent["patterns"]:
        tokens = nltk.word_tokenize(pattern)
        words.extend(tokens)
        doc_X.append(pattern)
        doc_y.append(intent["tag"])

    # add the tag to the classes if it's not there already
    if intent["tag"] not in classes:
        classes.append(intent["tag"])
# lemmatize all the words in the vocab and convert them to lowercase
# if the words don't appear in punctuation
words = [lemmatizer.lemmatize(word.lower()) for word in words if word not in string.punctuation]
# sorting the vocab and classes in alphabetical order and taking the # set to ensure no duplicates occur
words = sorted(set(words))
classes = sorted(set(classes))
```

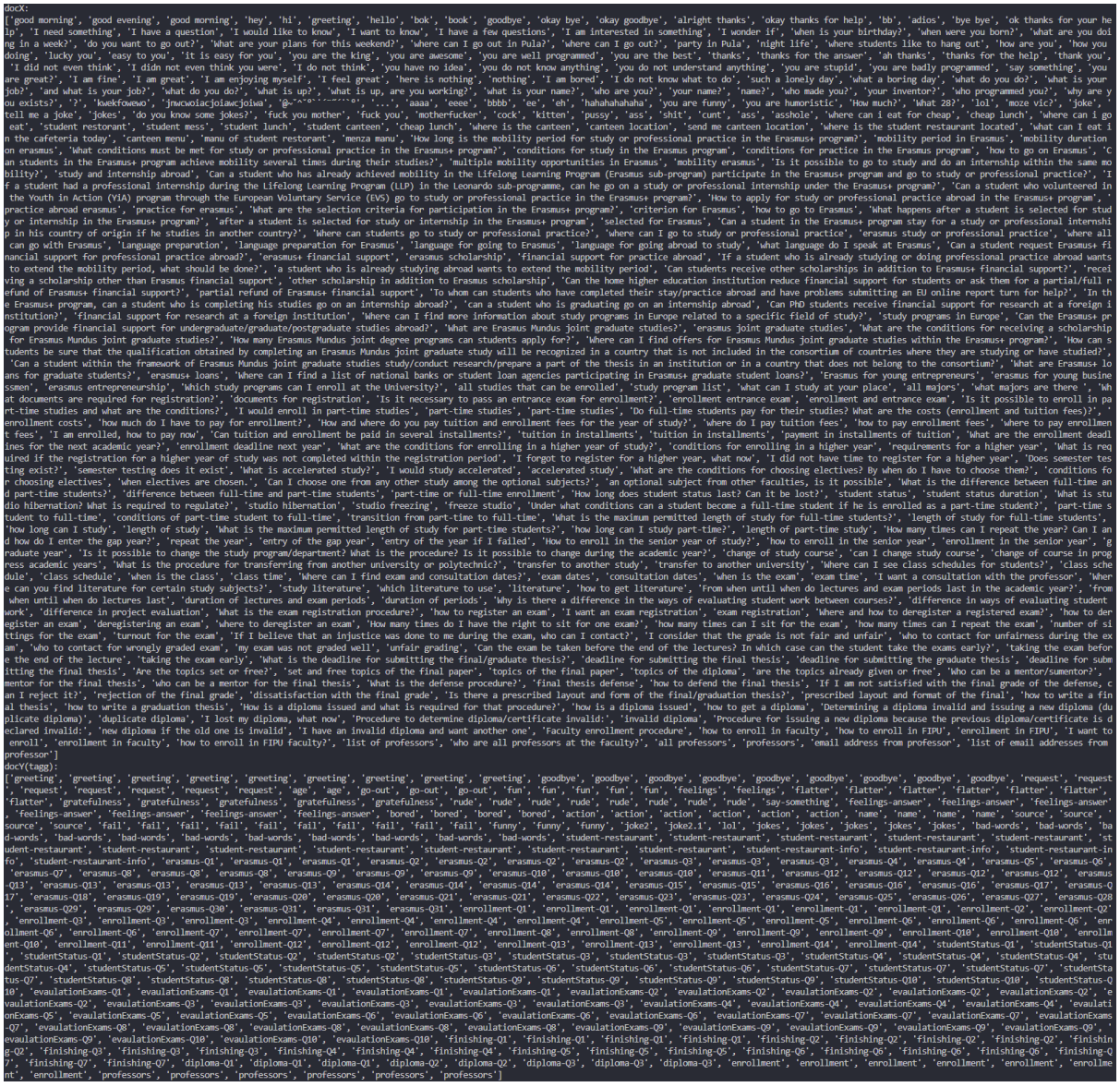
Slika 12. Prikaz početne inicijalizacije „FIPUbot-a“

Na slici 13 prikazane su riječi i oznake (classes) sortirane, nakon što su prošli kroz petlje i ne sadržavaju interpunkcijske znakove.

Na slici broj 14 prikazan je ispis polja dox_X i doc_y, što je objašnjeno ranije.



Slika 13. Prikaz polja words i polja classes



Slika 14. Prikaz polja doc_X i polja doc_y

U sljedećem odlomku pojasniti će se kako se treniraju podaci pomoću biblioteke tensorflow. Inače, tensorflow je „open-source“ biblioteka koja služi za umjetnu inteligenciju. Međutim, podaci koje treniramo, treba prvo prebaciti u numerički format, zbog toga što metoda neuronskih mreža ne može učiti iz riječi. Kasnije, pomoću funkcije bag_of_words vraća iz oblika broja u tekst. Isto tako, korištena je biblioteka numpy, kojoj je glavna svrha rad sa višedimenzionalnim nizovima i matricama. Slika 15 prikazuje način na koji su pretvorene riječi u brojeve. Slika 16 prikazuje izgled prvih nekoliko riječi koje su formatirane u numerički oblik.

```
# list for training data
training = []
out_empty = [0] * len(classes)
# creating the bag of words model
for idx, doc in enumerate(doc_X):
    bow = []
    text = lemmatizer.lemmatize(doc.lower())
    for word in words:
        bow.append(1) if word in text else bow.append(0)
    # mark the index of class that the current pattern is associated
    # to
    output_row = list(out_empty)
    output_row[classes.index(doc_y[idx])] = 1
    # add the one hot encoded BoW and associated classes to training
    training.append([bow, output_row])
# shuffle the data and convert it to an array
random.shuffle(training)
training = np.array(training, dtype=object)
# split the features and target labels
train_X = np.array(list(training[:, 0]))
train_y = np.array(list(training[:, 1]))
```

Slika 15. Prikaz načina za pretvorbu iz riječi u numerički format

```
train_x [[0 0 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 [0 0 1 ... 0 0 0]]
train_y [[0 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]
 [0 0 0 ... 0 0 0]]
```

Slika 16. Prikaz prvih nekoliko podataka neuronske mreže

Kada su podaci pretvoreni u numerički oblik, potrebno je stvoriti model dubokog učenja koji će učiti iz neuronske mreže. Kao što je već ranije navedeno, za treniranje podataka koristi se biblioteka tensorflow. Polja train_x i train_y, koja su objašnjena i prikazana na prethodnoj slici, pomoću njih će istrenirani podaci predvidjeti oznaku i odabrati nasumičan odgovor. Nasumičan odgovor se vraća ukoliko ima više od jednog odgovora navedenog u atributu „responses“. Slika 17 prikazuje treniranje podataka pomoću dubokog učenja.

```
# defining some parameters
input_shape = (len(train_X[0]),)
output_shape = len(train_y[0])
epochs = 200
# the deep learning model
model = Sequential()
model.add(Dense(128, input_shape=input_shape, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(output_shape, activation = "softmax"))
adam = tf.keras.optimizers.Adam(learning_rate=0.01, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=adam,
              metrics=["accuracy"])
model.fit(x=train_X, y=train_y, epochs=epochs, verbose=1)
```

Slika 17. Prikaz treniranja podataka pomoću dubokog učenja

Sekvencijalni model najčešće se koristi za jednostavan skup slojeva gdje svaki sloj ima točno jedan ulazni i jedan izlazni tenzor[14]. Tenzor je vektor ili matrica n dimenzija koje predstavljaju sve vrste podataka [15]. Isto tako, sve vrijednosti u tenzoru sadrže identičan tip podataka s poznatim oblikom.

„Dense“ je potpuni sloj gusto povezanih neurona u kojemu svaki izlaz ovisi o ulazu. Također, kao što je navedeno u članku [13], korišten je takozvani „Dropout“ koji sprječava prekomjerno popunjavanje modela nasumičnim postavljanjem stope ulaznih jedinica na nulu. Kako bi dobili u potpunosti funkcionalan „chatbot“ treba postojati još nekoliko funkcija koje će upite klijenata pretvarati u ulazne podatke neuronske mreže, zatim pronalaziti odgovarajuću oznaku i vraćati odgovor klijentu. Slika 18 prikazuje sažetak modela dubokog učenja i prvih 5 epoha prolaska kroz skup podataka. Ukupno ima 200 epoha, to jest broj puta koliko će algoritam učenja proći kroz skup podatka.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               60032

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 102)               6630

=================================================================
Total params: 74,918
Trainable params: 74,918
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/200
12/12 [==============================] - 1s 4ms/step - loss: 4.6310 - accuracy: 0.0108
Epoch 2/200
12/12 [==============================] - 0s 5ms/step - loss: 4.3238 - accuracy: 0.0813
Epoch 3/200
12/12 [==============================] - 0s 6ms/step - loss: 3.7545 - accuracy: 0.1572
Epoch 4/200
12/12 [==============================] - 0s 6ms/step - loss: 2.9942 - accuracy: 0.3333
Epoch 5/200
12/12 [==============================] - 0s 5ms/step - loss: 2.4290 - accuracy: 0.3957
```

Slika 18. Prikaz sažetka modela i prvih 5 pokušaja treniranja

Prvo, kada klijent pošalje upit, treba iz upita maknuti sve interpunkcijske znakove i napraviti polje koje sadržava rečenicu od upita. Slika 19 prikazuje funkciju koja uklanja sve znakove i formatira rečenicu u polje.

```
def clean_text(text):
  tokens = nltk.word_tokenize(text)
  tokens = [lemmatizer.lemmatize(word) for word in tokens]
  print("clean text", tokens)
  return tokens
```

Slika 19. Funkcija clean_text

Zatim je napravljena funkcija bag_of_words. Funkcija pretvara iz teksta u numerički oblik, kako bi se mogla napraviti usporedba sa skupom podataka. Funkcija je definirana kao pomoćna funkcija te se koristi u funkciji pred_class.

Funkcija pred_class pretvara upit od klijenta sa hrvatskog na engleski jezik. Zatim, koristi funkciju bag_of_words, čija je funkcionalnost objašnjena prethodno. Nakon što pred_class

funkcija pronađe odgovarajuću oznaku, vraća oznaku koja misli da je prikladna oznaci za upit koji je poslao klijent. Slika 20 prikazuje pomoćnu funkciju bag_of_words i pred_class.

```
def bag_of_words(text, vocab):
  tokens = clean_text(text)
  bow = [0] * len(vocab)
  for w in tokens:
    for idx, word in enumerate(vocab):
      if word == w:
        bow[idx] = 1
  print('bag_of_words', np.array(bow))
  return np.array(bow)

def pred_class(text, vocab, labels):
  translated_text = translator.translate(text, dest="en")
  lower_text = translated_text.text.lower()
  # print(lower_text)

  bow = bag_of_words(lower_text, vocab)
  result = model.predict(np.array([bow]))[0]
  thresh = 0.2
  y_pred = [[idx, res] for idx, res in enumerate(result) if res > thresh]

  y_pred.sort(key=lambda x: x[1], reverse=True)
  return_list = []
  for r in y_pred:
    return_list.append(labels[r[0]])
  print('return_list', return_list)
  return return_list
```

Slika 20. Prikaz pomoćne funkcije bag_of_words i pred_class funkciju

Na slici 21 prikazan je primjer ispisa pomoćnih funkcija clean_text i bag_of_words te ispis pred_class funkcije. Upit koji je korišten za primjer je „što je danas za jesti?“. Isto tako, u primjeru se vidi kako upit sa hrvatskog jezika, u funkciji pred_class i pomoću pomoćnih funkcija clean_text i bag_of_words upit prevodi na engleski jezik. Zadnja linija prikazuje metodu GET koja vraća klijentu odgovor, više o tome je objašnjeno u poglavlju Flask API.

```
clean text ['what', 'is', 'to', 'eat', 'today']
bag_of_words [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
1/1 [==============================] - 0s 120ms/step
return_list ['student-restaurant-info']
TAG student-restaurant-info
127.0.0.1 - - [19/Sep/2022 02:27:20] "GET /getanswer/šta%20je%20danas%20za%20jesti HTTP/1.1" 200 -
```

Slika 21. Primjer upita „Šta je danas za jesti?“

Kako bi dobili skroz funkcionalan „chatbot“ potrebna je još jedna funkcija koja vraća rezultat. Definirana je funkcija get_response koja prima parametre intents_list i intents_json. Funkcija prolazi kroz cijeli skup podataka i traži odgovarajuću oznaku. Parametar intents_list je ustvari funkcija pred_class , koja je pojašnjena u prethodnom odlomku. Slika 22 prikazuje get_response funkciju.

```python
def get_response(intents_list, intents_json):
  tag = intents_list[0]
  list_of_intents = intents_json["intents"]
  for i in list_of_intents:
    if i["tag"] == tag:
      result = random.choice(i["responses"])
      break
  return result
```

Slika 22. get_response funkcija

4.1.1 Flask API

Nakon što je logika za rad „chatbota“ postavljena, trebalo bi napraviti takozvani „REST API“, kako bi upit klijenta mogao slati u „chatbot“ pomoću funkcija navedenih prethodno. „REST API“ (Representational state transfer) služi kako bi stvorili komunikaciju između dvaju servera. Točnije, kako bi mogli slati informacije preko interneta. Prima metode GET,POST,PUT,PATCH,DELETE i tako dalje. Svaka metoda služi za nešto, kod „FIPUbota“ korištene su metode GET i POST.

Isprva, treba uvesti odgovarajuće biblioteke kako bi naš „API“ bio funkcionalan. Korištene biblioteke su: flask, flask_cors, flask_mail, os i dotenv. Pojašnjenje svake biblioteke i njene namjere biti će objašnjeno naknadno. Nakon što su odgovarajuće biblioteke dodane, postavljamo Flask aplikaciju i dopuštamo „cors policy“. „Cors policy“ kao što navodi [16], je mehanizam koji omogućuje poslužitelju da navede bilo koju domenu s koje bi preglednik trebao dopustiti učitavanje resursa. Slika 23 prikazuje početnu inicijalizaciju Flask aplikacije. Dotenv nam služi kako bi zaštitili osjetljive podatke, kao što su email, ime server i šifra.

```
from flask import Flask, request
from flask import request
from flask_cors import CORS
from flask_mail import Mail, Message
from flask import Response
import chatbot
import os
from dotenv import load_dotenv, find_dotenv
load_dotenv(find_dotenv())

app = Flask(__name__)
cors=CORS(app)      Matej, 2 months ago • in
app.config['CORS_HEADERS'] = 'Content-Type'
```

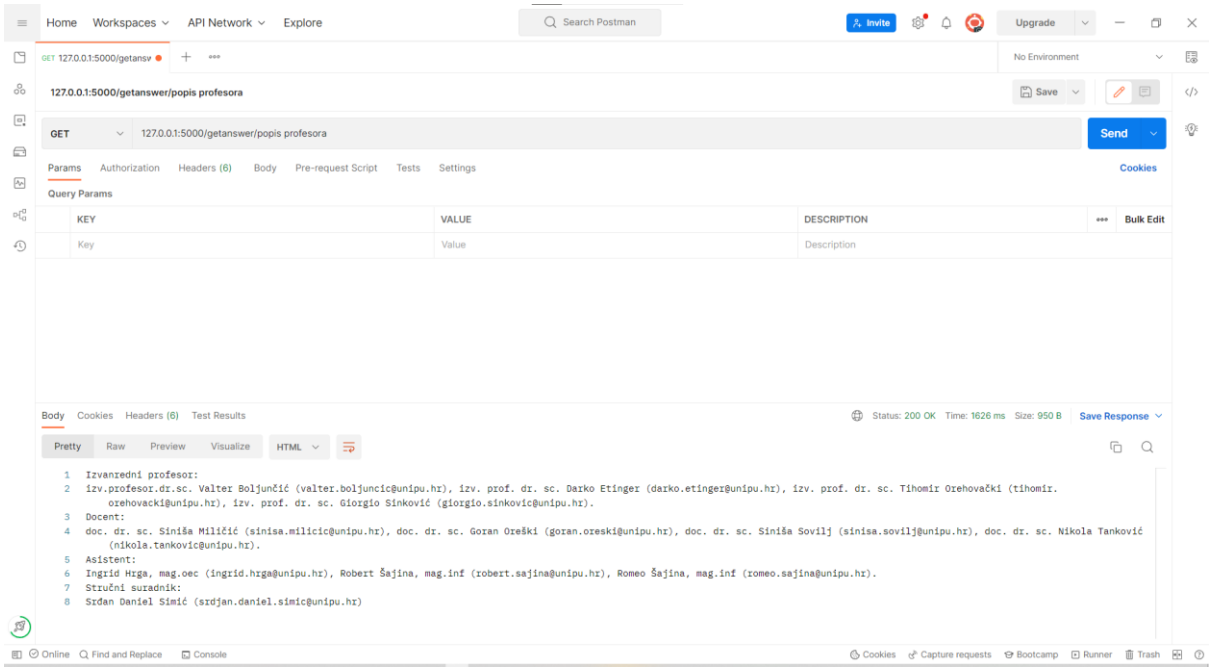
Slika 23. Inicijalizacija Flask aplikacije

Zatim, dodana je ruta „/getanswer/<question>“. Parametar „Question“ je promjenjiv zato je postavljena kao dinamička ruta. Dinamička ruta služi kako bi se omogućilo korištenje varijable „question“ u daljnjem kodu. Slika 24 prikazuje GET metodu za dobivanje odgovora od strane „chatbota“.

```python
@app.route("/getanswer/<question>", methods=['GET'])
def getAnswer(question):
        message = question.lower()
        intents = chatbot.pred_class(message, chatbot.words, chatbot.classes)
        result = chatbot.get_response(intents, chatbot.chatBotData)
        return result
```

Slika 24. GET metoda za vraćanje odgovora klijentu

Isto tako, treba napomenuti da su korištene funkcije „chatbota“, koje su uvezene pomoću „import-a“, kao što je prikazano na prethodnoj slici broj 23. Funkcije „chatbota“ objašnjene su u poglavlju Chatbot. Nakon što je flask aplikacija pokrenuta, može se dobivati odgovor od strane našeg „chatbota“. Pomoću programa Postman testirana je flask aplikacija. Slika 25 prikazuje GET zahtjev koristeći program Postman.



Slika 25. Prikaz GET metode za vraćanje odgovora koristeći Postman

Aplikacija „FIPUbot“ također nudi slanje emaila preko POST metode. Kako bi osigurali osjetljive podatke kao što je već navedeno prije, koristili smo biblioteku dotenv. Isto tako, dodan je broj porta kojim šaljemo email te SSL postavljen na True kako bi kriptirali komunikaciju. Slika 26 prikazuje podatke za slanje emaila.

```
# dotenv secured data
USERNAME = os.environ.get("EMAIL")
SERVER = os.environ.get("SERVER")
PASSWORD = os.environ.get("PASSWORD")

app.config['MAIL_SERVER'] = SERVER
app.config['MAIL_PORT'] = 465
app.config['MAIL_USERNAME'] = USERNAME

app.config['MAIL_PASSWORD'] = PASSWORD
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)
```

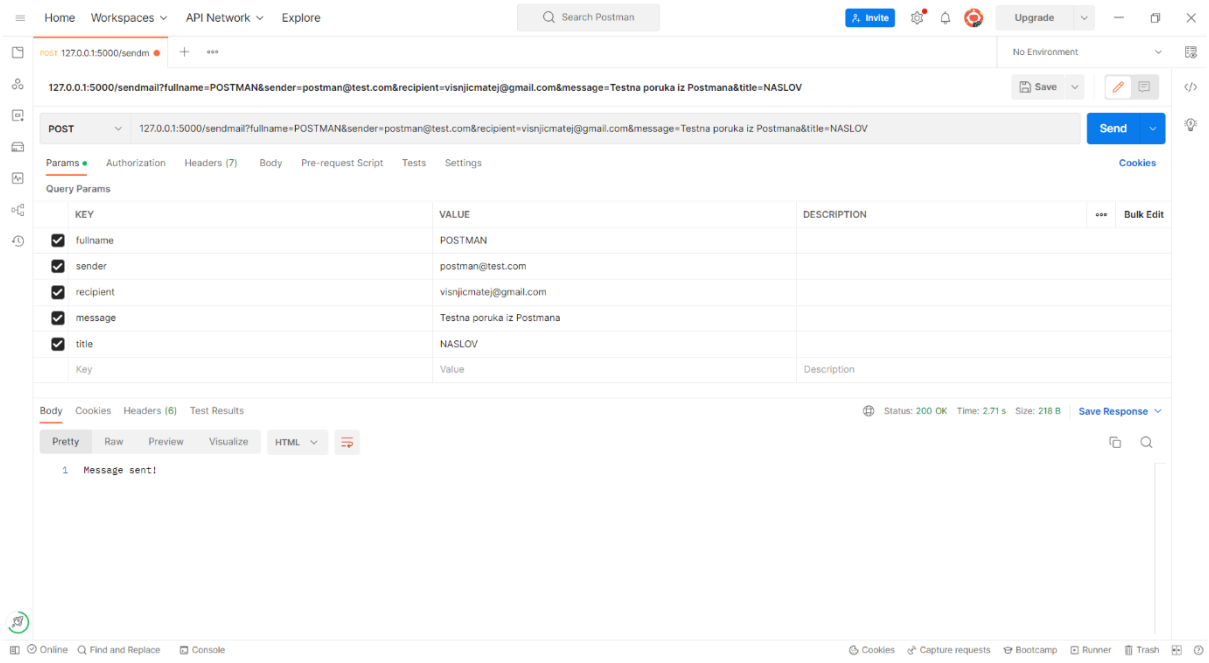
Slika 26. Podaci za slanje email-a

Nakon što su podaci osigurani, potrebna je funkcija kako bi se uspješno slao email. Ruta „/sendmail“ prima „query“ parametre. „Query string“ je dio „URL-a“ koji sadržava posebne varijable koje kasnije možemo koristiti u funkciji. Slika 27 prikazuje funkciju za slanje email-a.

```
@app.route("/sendmail", methods=['GET', 'POST'])
def sendmail():
        try:
                if request.method == 'POST':
                        fullname = request.args['fullname']
                        sender = request.args['sender']
                        recipient = request.args['recipient']
                        message = request.args['message']
                        title = request.args['title']
                        subject = fullname +  " | " + title

                        msg = Message(subject,sender=sender, recipients = [recipient])
                        msg.body = message
                        mail.send(msg)
                return 'Message sent!'
        except Exception as e:
                return Response("Fail! Try again later.", status=404, mimetype='application/json')
```

Slika 27. Funkcija za slanje e-mail-a

Primjer u Postmanu za POST metodu funkcije sendmail prikazana je na slici 28, a na slici 29 prikazana je poruka na Gmail-u.



Slika 28. Prikaz slanje POST zahtjeva putem Postman-a

Slika 29. Prikaz poruke na Gmail-u

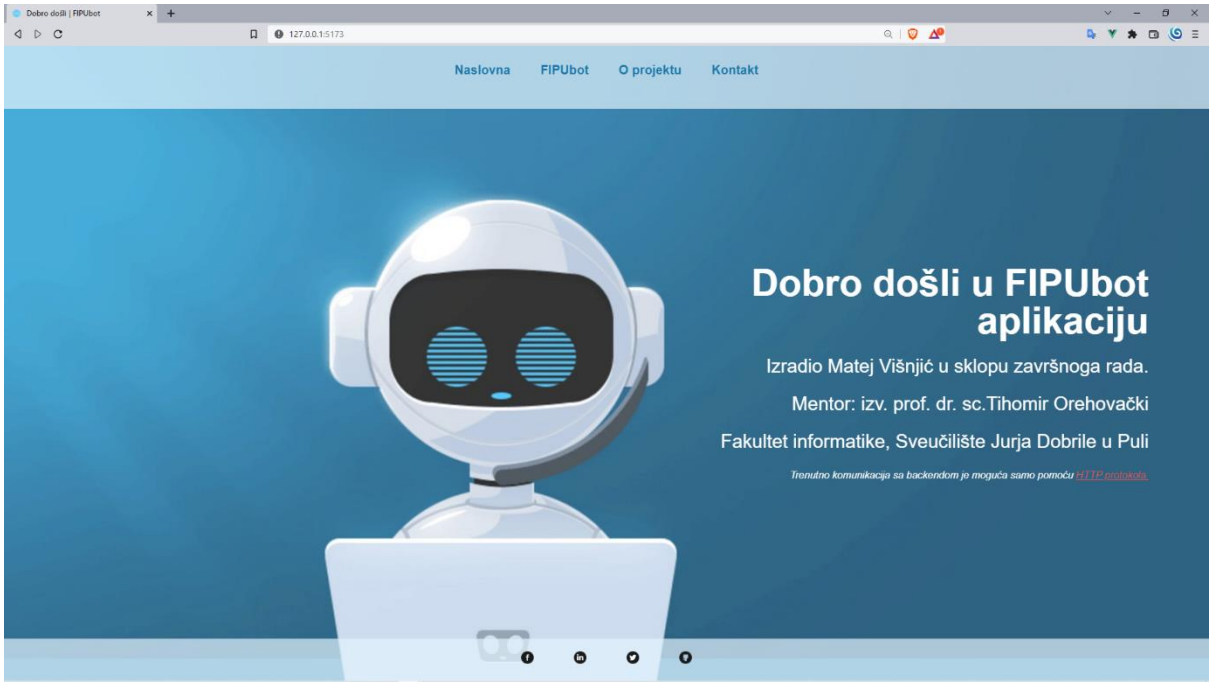
Za pokretanje aplikacije poziva se metoda „run“ nad objektom „app“ koja je prikazana na slici 30.

```
if(__name__ == '__main__'):
        app.run(debug=False, use_reloader=False)
```

Slika 30. Prikaz metode „run“ nad objektom „app“

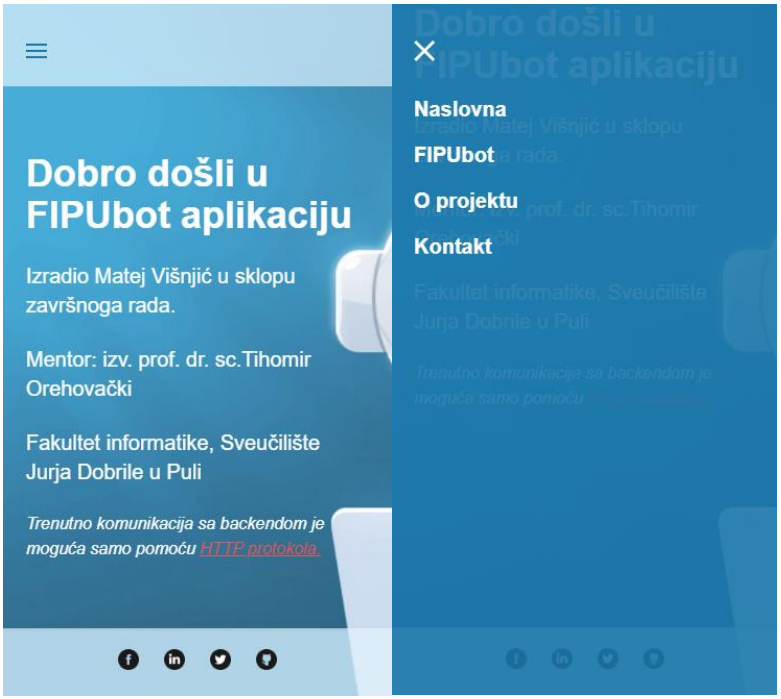
4.2 Grafičko sučelje

Izgled aplikacije stvoren je koristeći Vue.js, točnije Vite 3 i TailwindCSS. Za početak inicijalizira se Vue.js framework. Nakon inicijalizacije Vue.js-a možemo započeti raditi na izgledu aplikacije. Prvo je postavljena glavna datoteka, index.js, zatim su postavljene komponente „header“ i „footer“. Slika 31 prikazuje početnu stranicu sa komponentama „header“ i „footer“.



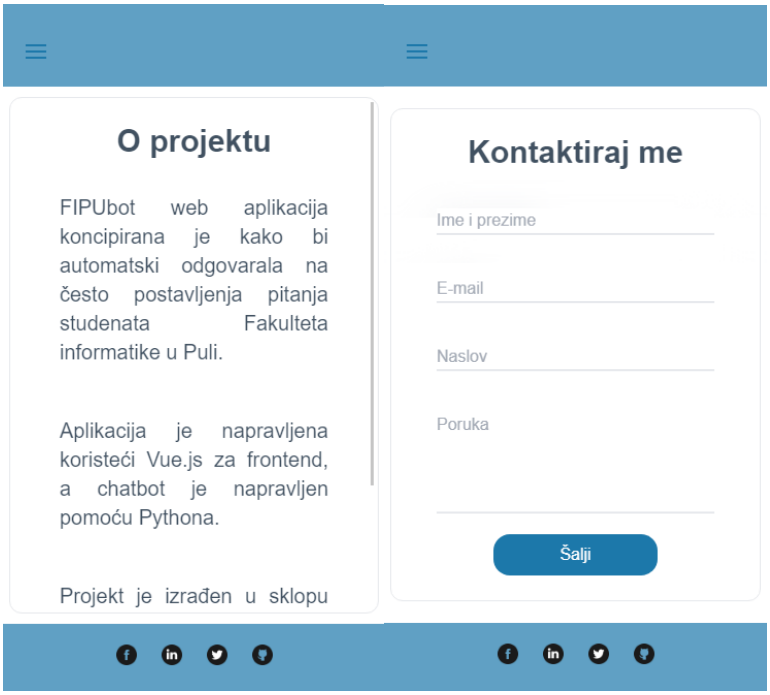
Slika 31. Prikaz izgleda početne stranice sa „headerom“ i „footerom“

Isto tako, treba napomenuti da je stranica responzivna i da ju je ugodno koristiti i na mobilnom uređaju. Slika broj 32 prikazuje početnu stranicu sa komponentama „header“ i „footer“ na mobilnom uređaju, dok je otvoren i zatvoren „header“.



Slika 32. Prikaz početne stranice na mobilnom uređaju

Na sličan način napravljene su stranice „O projektu“ i „Kontakt“. Izgled spomenutih stranica na mobilnom uređaju prikazan je na slici 33.



Slika 33. Prikaz „O projektu“ i „Kontakt“ stranice

Stranica za „chatbot“ kompleksnija je i sadržava više komponenti i funkcionalnosti. Slika 34 prikazuje izgled stranice „FIPUbot“ na mobilnom uređaju.

Slika 34. Izgled stranice „FIPUbot“

Komponente koje se također koriste u aplikaciji za slanje i primanje poruka jesu „UserChatTextBox“ i „BotChatTextBox“. Glavna svrha „UserChatTextBox“ jest da prima parametar od poruke koju je klijent poslao „botu“ i ispisuje to u obliku tekstualnog okvira. Za ispisivanje vremena poslane poruke korištena je funkcija „new Date().getHours()“ koji vraća trenutne sate. Za provjeru minuta korištena je funkcija „checkMinutes()“, kako bi ukoliko je broj minuta između 0 i 9, dodao 0 ispred broja minuta. Prikaz cjelokupne komponente prikazano je na slici broj 35.

```
<template>
    <div class="p-6 w-4/6 m-2 border bg-blue-400 rounded-2xl">
        <div class="text-l lg:text-lg text-black text-left break-words">
            {{ message }}
        </div>
        <div class="border-t border-sky-500 text-black">
            {{ hour }} : {{ minutes }}
        </div>
    </div>
</template>

<script>
export default {
    name: 'UserChatTextBox',
    props: ['message'],
    data() {
        return {
            hour: new Date().getHours(),
            minutes: this.checkMinutes(),
        }
    },
    methods: {
        checkMinutes() {
            // because new Date().getMinutes() function return single digits numbers without a zero in ahead of
            let minutes = new Date().getMinutes()
            if (minutes >= 0 && minutes <= 9) {
                return '0' + minutes
            } else return minutes
        },
    },
}
</script>
```

Slika 35. Prikaz komponente „UserChatTextBox“

Komponenta „BotChatTextBox“ se razlikuje od prethodne komponente, zbog toga što provjerava da li poruka sadrži „URL“ koju je dobio kao odgovor od „bota“. Kao parametre prima „message“ i „urls“, te navedene koristi za ispis u tekstualnom okviru. Provjera sadrži „v-if“ i „v-for“. Znači, ukoliko postoji „URL“ u poruci, onda se prikazuje „URL“, pomoću „v-for“ petlje. Također, ukoliko ima više od jednog „URL-a“, prikazat će ih sve redom. Slika 36 prikazuje komponentu „BotChatTextBox“.

Za lakšu komunikaciju sa „back-end“ serverom, kreiran je „service.js“, kojim možemo lakše upravljati zahtjevima upućenih od strane klijenta. U istoj datoteci, navodimo domenu servera. Zatim, radimo funkcije koje primaju određene parametre, a prikazani su i objašnjeni u prethodnom poglavlju. Funkcije koje smo kreirali jesu „getAnswer.getMessage“ i „sendMail-send“. Funkcija „getMessage“ prima parametar „message“, a Funkcija „sendMail.send“ prima parametre „fullname“, „email“, „subject“ i „message“. Kao primalac poruke navedena je email adresa od kreatora aplikacije. Sadržaj datoteke „service.js“ prikazan je na slici 37.

```vue
<template>
    <div class="p-6 w-4/6 m-2 bg-gray-300 border rounded-2xl">
        <div class="text-lg lg:text-lg text-left break-words">
            {{ message }}

            <div v-if="urls" class="font-bold border-t border-black mt-2">
                URLs:
                <div v-for="(url, key) in urls" :key="url" class="underline">
                    <a :href="`${url}`" target="_blank"
                        >{{ key + 1 }} - {{ url }}</a
                    >
                </div>
            </div>
        </div>

        <div class="border-t">{{ hour }} : {{ minutes }}</div>
    </div>
</template>

<script>
export default {
    name: 'BotChatTextBox',
    props: ['message', 'urls'],
    data() {
        return {
            hour: new Date().getHours(),
            minutes: this.checkMinutes(),
        }
    },
    methods: {
        checkMinutes() {
            // because new Date().getMinutes() function return single digits numbers without a zero in ahead of
            let minutes = new Date().getMinutes()
            if (minutes >= 0 && minutes <= 9) {
                return '0' + minutes
            } else return minutes
        },
    },
}
</script>
```

Slika 36. Prikaz komponente „BotChatTextBox“

```vue
import axios from 'axios'

const fipubotAPI = axios.create({
    baseURL: import.meta.env.VITE_SERVER,
    timeout: 10000,
})

let getAnswer = {
    getMessage(message) {
        return fipubotAPI.get(`/getanswer/${message}`)
    },
}

let sendMail = {
    send(fullname, email, subject, message) {
        return fipubotAPI.post(
            `/sendmail?fullname=${fullname}&sender=${email}&recipient=visnjicmatej@gmail.com&message=${message}&title=${subject}`
        )
    },
}

export { fipubotAPI, getAnswer, sendMail }
```

Slika 37. Prikaz sadržaja datoteke „service.js“

Kao što je već na slici 34 prikazan izgled „FIPUbot“ stranice, ovaj odlomak pojasniti će kako točno radi te kako šalje i prima poruke. Za čuvanje poruka dok klijent koristi web aplikaciju korišten je „store.js“ koji je reaktivan, što znači da prati promjenu i odmah ažurira odabrani HTML element. Dok korisnik ne osvježi stranicu poruke ostaju spremljene u „store-u“. Slika 38 prikazuje datoteku „store.js“

```
import { reactive } from 'vue'

export let store = reactive({
    messages: [],
    clientMessage: '',
})
```

Slika 38. Prikaz sadržaja datoteke „store.js“

Poruke se primaju i šalju pomoću funkcije „sendMessage“, koja prvo postavlja „isActive“ na True, kako bi blokirao više upita odjednom. Zatim, provjerava se duljina poruke, ukoliko je kraća od 3 znaka, vraća klijentu grešku. Ukoliko je poruka duža ili jednaka duljini od 3 znaka, funkcija poziva funkciju „getAnswer.getMessage“ koja je dio datoteke „service.js“. Ukoliko zahtjev prolazi bez greške, polje „messages“ koje je definirano u „store.js“, gura poruku od klijenta u polje. Zatim, provjerava se da li dolazna poruka sadržava „URL“. Ukoliko sadržava „URL“, u polje „messages“ dodaje se poruka od „bota“ te popis „URL-a“. Nakon toga poziva se funkcija „this.$nextTick()“, koja služi da se prozor spusti do zadnje poruke. Prikaz funkcije „sendMessage“ nalazi se na slici broj 40. Ispis poruke na grafičko sučelje napravljen je pomoću „v-for“ petlje, koja provjerava da li ima poruka od „bota“ iz polja „messages“. Ukoliko ima poruke, ispisuje na zaslon. Prikaz koda za ispisivanje poruka je na slici broj 39.

```
<div class="flex justify-start pt-28">
    <BotChatTextBox :message="welcomeMsg" />
</div>
<div v-for="(message, i) of store.messages" :key="i">
    <div class="flex justify-start">
        <BotChatTextBox
            :message="message.msg"
            :urls="message.urls"
            v-if="message.author === 'server'"
        />
    </div>
    <div class="flex justify-end">
        <UserChatTextBox
            :message="message.msg"
            v-if="message.author === 'client'"
        />
    </div>
</div>
```

Slika 39. Prikaz koda za ispis poruka

```
async sendMessage() {
    // function for sending a message
    this.isActive = true
    if (store.clientMessage.length <= 3) {
        store.clientMessage = ''
        this.isActive = false
        return alert('Prazna ili prekratka poruka!')
    } else {
        await getAnswer
            .getMessage(store.clientMessage.toLowerCase())
            .then((res) => {
                store.messages.push({
                    msg: store.clientMessage,
                    author: 'client',
                })
                store.clientMessage = ''

                if (typeof res.data === 'string') {
                    //checking for urls in string
                    var stringUrls =
                        res.data.match(/\bhttps?:\/\/\S+/gi)
                }
                store.messages.push({
                    msg: res.data,
                    urls: stringUrls,
                    author: 'server',
                })
                this.isActive = false
            })
            .catch((e) => {
                console.error(e)
                alert(e)
                store.clientMessage = ''
                this.isActive = false
            })
        this.$nextTick(() => {
            this.$refs.chatbot.scrollTop =
                this.$refs.chatbot.scrollHeight
        })
    }
},
```

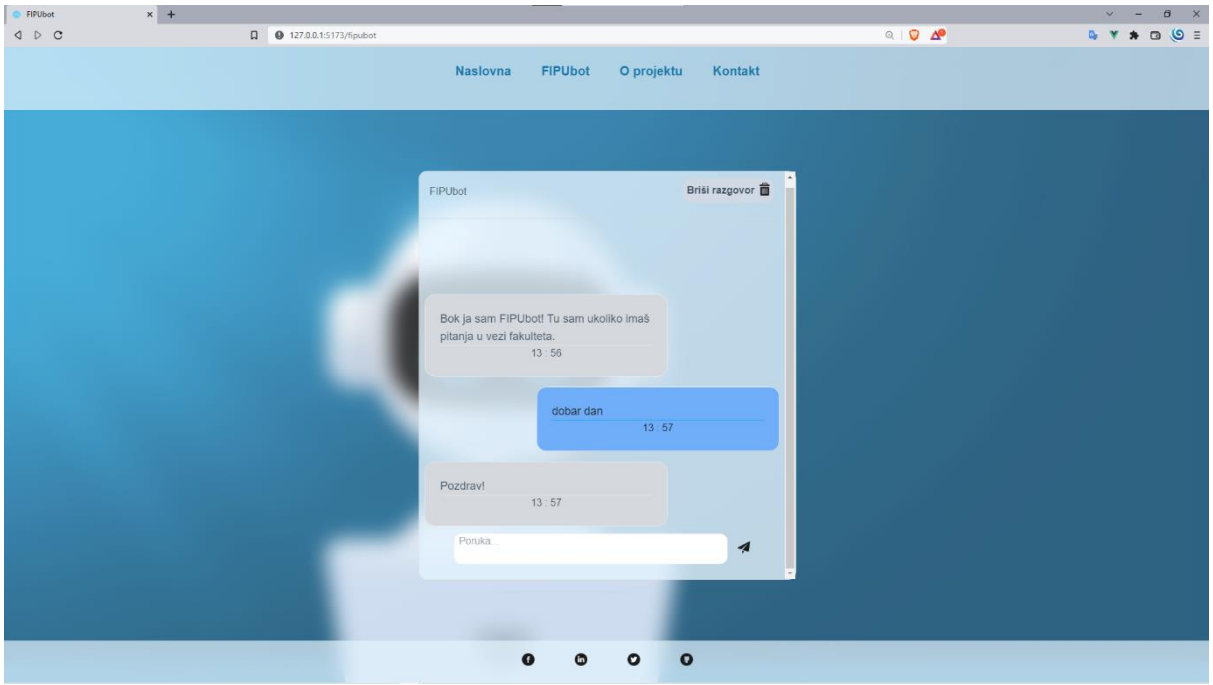
Slika 40. Prikaz funkcije „sendMessage“

Za brisanje trenutnih poruka koje su razmjenili klijent i „bot“, koristi se funkcija „clearChat()“, koja prvo provjerava da li je duljina polja „messages“ manja od jedan. Ukoliko je manja od jedan, funkcija se ne pokreće, to jest brisanje nije moguće. To sprječava pozivanje funkcije ukoliko je polje prazno. Prikaz funkcije za brisanje poruke je na slici 41.

```
clearChat() {
    // doesn't call function if chat is already cleared.
    if (store.messages.length < 1) {
        return 0
    } else {
        store.messages = []
    }
},
```

Slika 41. Prikaz funkcije za brisanje poruke

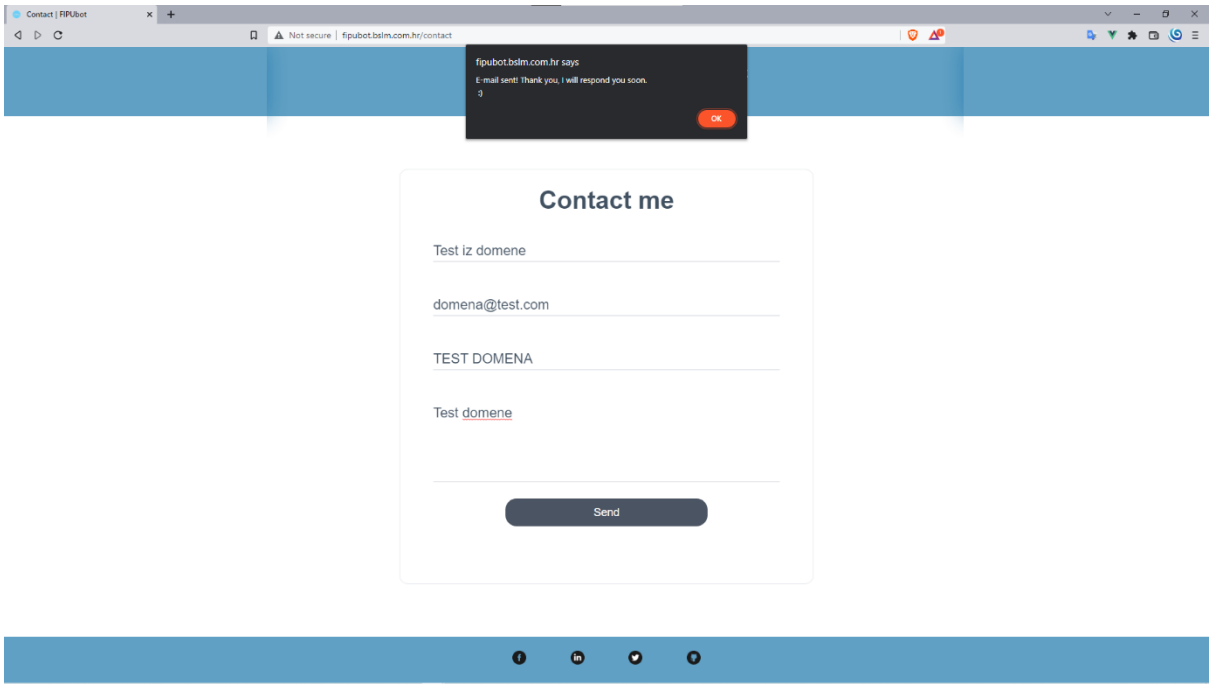
Prikaz primjera kako radi „FIPUbot“ prikazan je na slici 42, gdje je prikazana poruka od strane klijenta, te odgovor koji je primio klijent od strane „bota“.



Slika 42. Prikaz razmjenjivanja poruka između klijenta i „bota“

Za slanje e-pošte preko „Kontakt“ stranice korištena je funkcija „sendMail()“, koja također postavlja „sendBtnClicked“ na „True“. S time ograničavamo da klijent može stiskati gumb nakon što je stisnut prvi put. Zatim se poziva funkcija „sendMail.send“ koja je definirana u „service.js“. Ukoliko je email uspješno poslan, klijent dobiva skočni prozor u kojem piše pozitivna poruka, a ukoliko je došlo do greške, klijent dobiva skočni prozor sa greškom, te mu je omogućeno ponovno stiskanje gumba, kako bi mogao ispraviti krive parametre te pokušati

poslati e-mail ponovno. Na slici 43 prikazano je slanje emaila putem stranice „Kontakt“. Zatim, na slici 44 prikazan je dolazni e-mail.



Slika 43. Prikaz slanja emaila pomoću „Kontakt“ stranice

Slika 44. Prikaz dolaznog e-mail-a

4.3 Raspberry PI

Raspberry PI služi za isporučivanje Flask aplikacije. Korišten je apache2 server i pomoću „mod_wsgi“ postavljeno je da služi Flask aplikaciju. Raspberry ima podešen dinamički DNS, koji svakih 15 minuta prati IP adresu. Postavljen je Google autentifikator, koji traži verifikacijski kod prije upisivanja lozinke. Isto tako, podešen je „fail2ban“ koji blokira određenu IP adresu nakon 2 kriva pokušaja ulaska u server. Ime domene, sa kojom se vrši komunikacija između „front-end-a“ i „back-end-a“, kupljena je od strane kreatora aplikacije. Za kompletnu instalaciju i podešavanja korištena je dokumentacija od apache2 [17], a za instalaciju mod_wsgi korištena je dokumentacija [18].

Nakon instalacije apache2 servera, trebalo je otvoriti portove od rutera. Otvoreni portovi su port 22 i port 80. Port 22 služi za SSH komunikaciju, a port 80 za serviranje aplikacije. Zatim, pomoću „sudo ufw“ komande, koja upravljanja vatrozidom, je namješteno dopuštanje prometa kroz određene portove. Status vatrozida prikazan je na slici 45.

```
tbm@raspberrypi:~ $ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
SSH                        ALLOW       Anywhere
WWW Full                   ALLOW       Anywhere
WWW Secure                 ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
SSH (v6)                   ALLOW       Anywhere (v6)
WWW Full (v6)              ALLOW       Anywhere (v6)
WWW Secure (v6)            ALLOW       Anywhere (v6)
```

Slika 45. Status vatrozida Raspberry-ja

Nakon što su portovi otvoreni i dozvoljen je promet na Raspberry-ju, pomoću dokumentacije navedene iznad se postavljaju datoteke koje su nužne za rad servera. U mapi „apache/sites-available“ podešavamo konfiguraciju za naš server. Upisivanje komande „sudo apache2ctl configtest“ provjerava se da li je konfiguracijska datoteka prošla bez greški. Primjer konfiguracije možete pogledati na slici 46. Zbog sigurnosti podataka servera, nisu prikazani podaci od trenutnog servera, nego primjer iz dokumentacije [18].

```
<VirtualHost *>
    ServerName example.com

    WSGIDaemonProcess yourapplication user=user1 group=group1 threads=5
    WSGIScriptAlias / /var/www/yourapplication/yourapplication.wsgi

    <Directory /var/www/yourapplication>
        WSGIProcessGroup yourapplication
        WSGIApplicationGroup %{GLOBAL}
        Order deny,allow
        Allow from all
    </Directory>
</VirtualHost>
```

Slika 46. Primjer izgleda datoteke za konfiguraciju servera

Prema dokumentaciji [18], koja govori da za razvijanje (eng. Deploy) Flask aplikacije, treba se koristiti „mod_wsgi“, koji služi za pokretanje aplikacije. Na slici 47 prikazan je primjer kako datoteka app.wsgi treba izgledati.

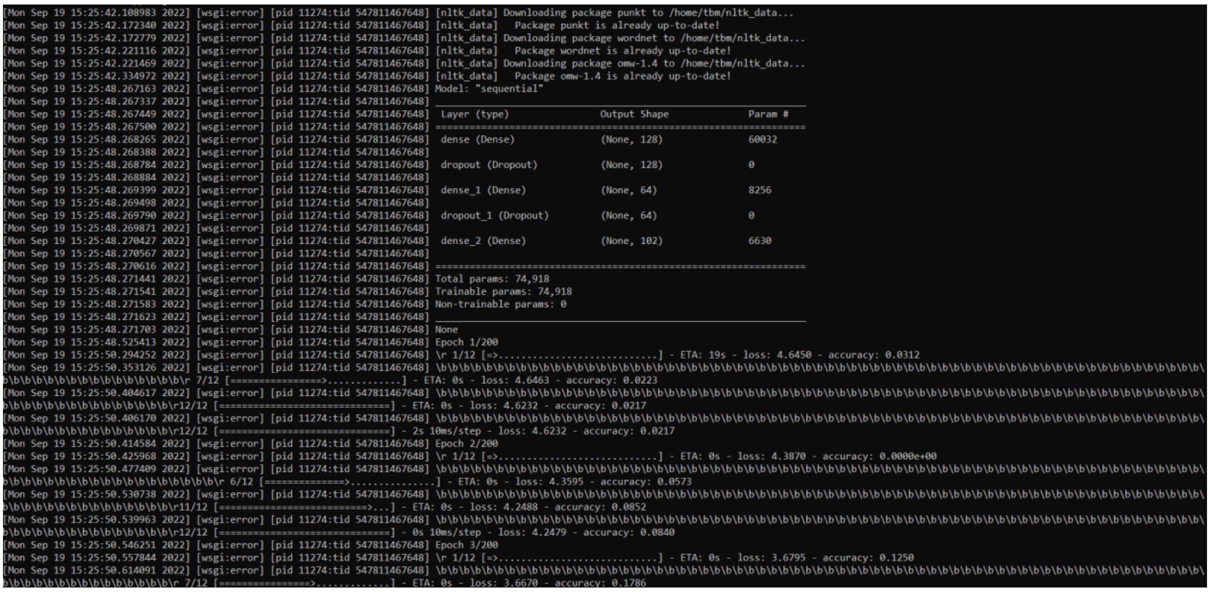
```
from yourapplication import app as application
```

If a factory function is used in a `__init__.py` file, then the function should be imported:

```
from yourapplication import create_app
application = create_app()
```

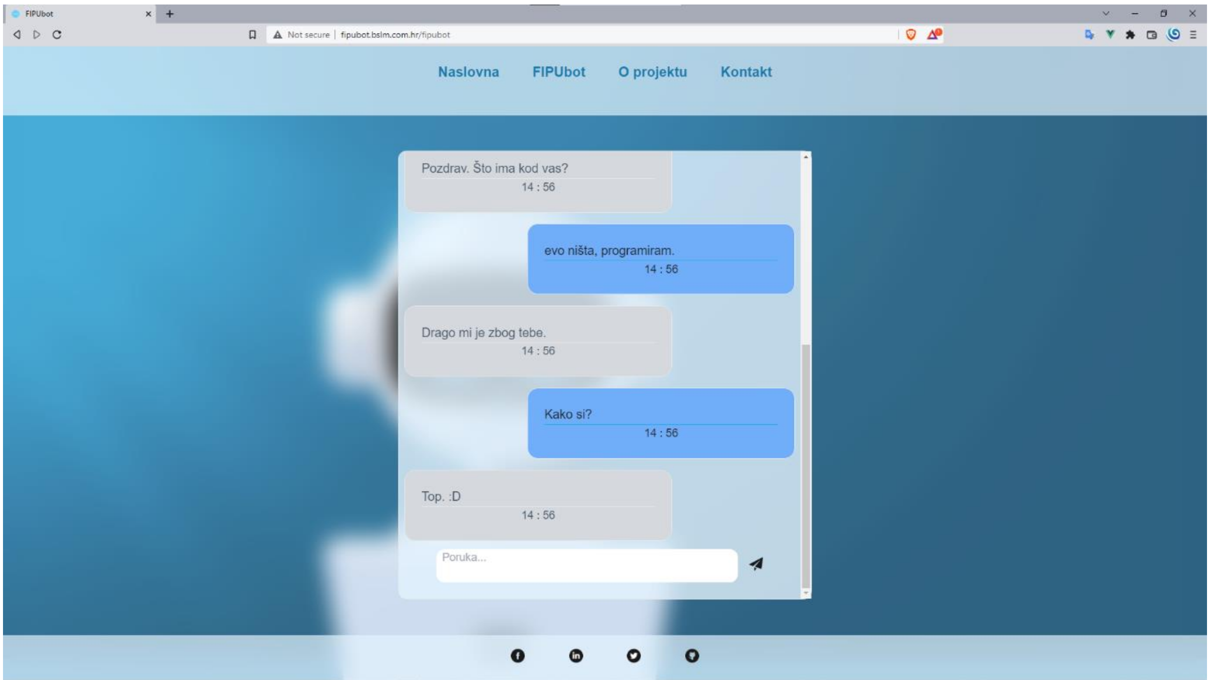
Slika 47. Primjer app.wsgi datoteke

Nakon što je apache2 i mod_wsgi uspješno konfiguriran, sljedeći korak je pokretanje apache2 servisa, sa komandom „sudo systemctl start apache2“. Međutim, prema dokumentaciji [17], koja govori da se koristi više procesa na Unix mašini, skripta se pokrene nekoliko puta, što obično traje 5 minuta. Nakon toga server je u mogućnosti slušati upite klijenata. Slika 48 prikazuje prvih nekoliko linija iz terminala koje pokazuju uspješno pokretanje skripte.



Slika 48. Prikaz pokrenutog apache2 server na Raspberry-ju

Nakon što je apache2 uspješno pokrenut, „back-end“ sluša na upite klijenata, te vraća odgovor. Na slici 49 prikazan je primjer korištena aplikacije preko domene http://fipubot.bslm.com.hr/. Trenutna komunikacija je moguća samo pomoću HTTP protokola.



Slika 49. Primjer korištenja „FIPUbot-a“ preko domene

5. Zaključak

„FIPUbot“ je postavljen na server koji je ujedno i Raspbery PI, koji nema instaliran SSL certifikat. Grafičko sučelje, takozvani „frontend“, izvodi se na drugome serveru koji podržava HTTP i HTTPS protokol. Zbog nemogućnosti instaliranja SSL certifikata na Raspberry PI, komunikacija između dva servera mora se odvijati isključivo pomoću HTTP protokola, u suprotnom dolazi do greške. Također, Raspberry PI ima svoju domenu te je namještena provjera IP adrese svakih 15 minuta. Ukoliko se adresa promjeni, automatski se šalje nova IP adresa koja se referencira na ime domene. Loša stvar je što se svaka 24 sata skripta se pokreće ispočetka, pa se podaci treniraju iznova. Iako postoji mogućnost da se uvijek isti podaci pokreću, premda ne mora značiti da će istrenirani podaci biti bolji od prethodnih.

Jedno od ograničenja „FIPUbot“ aplikacije jesu ta da vraća nasumične odgovore, koje su navedene u skupu podataka. Točnije, ukoliko ima 5 različitih vrsta odgovora na određenu vrstu pitanja, za svaki odgovor vjerojatnost je 20% da će vratiti isti odgovor kao prethodni. Aplikacija ne prati pitanja koja su već postavljena, niti odgovore koje je već klijent slao. Isto tako, ponekad dolazi do greške, jer nije uspio pronaći odgovarajući odgovor na postavljeno pitanje. Aplikacija ima i mogućnost slanja e-mail-a. Trenutno je postavljeno da e-mail zaprima samo kreator aplikacije. Također, jedan od problema je ne korištenje neprekidnog napajanja za Raspberry PI, stoga ukoliko dođe do vremenskih neprilika i slično, može doći do nedostupnosti servera.

S obzirom na sve gore navedeno i ukoliko bi se usavršile stvari koje trenutno ne rade kako treba, „Fipubot“ bi u bliskoj budućnosti mogao znatno poboljšati poslovanje za djelatnike Sveučilišta Jurja Dobrile u Puli, koji moraju odgovarati na upite studenata. Naravno, ne odnosi se na sve zahtjeve studenata, nego one koji se mogu automatizirati. To bi uštedilo vrijeme djelatnika koje rade na Sveučilištu Jurja Dobrile u Puli.

6. Popis literature

[1] Uryutin, O. (2018b, September 13). A brief history of web app - Oleg Uryutin. Retrieved September 19, 2022, from https://oleg-uryutin.medium.com/a-brief-history-of-web-app-50d188f30d

[2] Ajax - Developer guides | MDN. (2022, September 9). Retrieved September 16, 2022, from https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

[3] Uryutin, O. (2018, September 13). A brief history of web app - Oleg Uryutin. Retrieved September 16, 2022, from https://oleg-uryutin.medium.com/a-brief-history-of-web-app-50d188f30d

[4] A. (2020, March 5). Progressive Web Apps: Everything You Need to Know. Retrieved September 16, 2022, from https://www.csschopper.com/blog/progressive-web-apps-everything-you-need-to-know/

[5] Anyoha, R. (2020, April 23). The History of Artificial Intelligence. Retrieved September 16, 2022, from https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/

[7] googletrans. (2020, June 14). Retrieved September 17, 2022, from https://pypi.org/project/googletrans/

[8] Mydataknox.hr. (2022, September 5). Mydataknox.hr. Retrieved September 17, 2022, from https://mydataknox.hr/

[9] Vue.js - The Progressive JavaScript Framework | Vue.js. (n.d.). Retrieved September 17, 2022, from https://vuejs.org/

[10] What is Python? Executive Summary. (n.d.). Retrieved September 17, 2022, from https://www.python.org/doc/essays/blurb/

[11] What is a raspberry pi? (2015, august 20). retrieved september 17, 2022, from https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/

[12] Visual Studio Code - Code Editing. Redefined. (2021, November 3). Retrieved September 17, 2022, from https://code.visualstudio.com/

[13] Pykes, k. (2022, june 28). build a simple chatbot in python with deep learning. retrieved september 17, 2022, from https://towardsdatascience.com/a-simple-chatbot-in-python-with-deep-learning-3e8669997758

[14] Team, K. (n.d.). Keras documentation: The Sequential model. Retrieved September 19, 2022, from https://keras.io/guides/sequential_model/

[15] Johnson, D. (2022, September 17). TensorFlow Basics: Tensor, Shape, Type, Sessions & Operators. Retrieved September 22, 2022, from https://www.guru99.com/tensor-tensorflow.html#1

[16] Cross-Origin Resource Sharing (CORS) - HTTP | MDN. (2022, September 13). Retrieved September 19, 2022, from https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[17] Documentation Group. (n.d.). Welcome! - The Apache HTTP Server Project. Retrieved September 19, 2022, from https://httpd.apache.org/

[18] Welcome to Flask — Flask Documentation (2.0.x). (n.d.). Retrieved September 19, 2022, from https://flask.palletsprojects.com/en/2.0.x/

[19] Hrvatski telekom. (n.d.). Retrieved from https://www.hrvatskitelekom.hr/

[20]Tudury, L. (2021). Cleverbot. In Dictionary.com. Retrieved from https://www.dictionary.com/e/slang/cleverbot/

[21] Često postavljana pitanja studenata. (n.d.). Retrieved September 22, 2022, from https://www.unipu.hr/studenti/faq

[22] FAQ. (n.d.). Retrieved September 22, 2022, from https://www.unipu.hr/medunarodna-suradnja/erasmus_program/mobilnost_studenata/faq

7. Popis slika

Slika 1. „Chatbot“ Sveučilišta Jurja Dobrile u Puli 4
Slika 2. Chatbot“ Hrvatskog Telekoma..... 4
Slika 3. Prikaz Cleverbota i povijest komunikacije 5
Slika 4. VS code – razvijanje izgleda aplikacije..... 7
Slika 5. VS code – razvijanje „chatbota“ aplikacije 8
Slika 6. GNU nano uređivač teksta..... 9
Slika 7. SSH konekcija i prikaz terminala..... 9
Slika 8. Prikaz skice rada aplikacije..... 10
Slika 9. Use case diagram..... 11
Slika 10. Skica dinamičkog DNS..... 11
Slika 11. Prikaz skupa podataka 12
Slika 12. Prikaz početne inicijalizacije „FIPUbot-a“..... 14
Slika 13. Prikaz polja words i polja classes..... 15
Slika 14. Prikaz polja doc_X i polja doc_y 15
Slika 15. Prikaz načina za pretvorbu iz riječi u numerički format..... 16
Slika 16. Prikaz prvih nekoliko podataka neuronske mreže..... 16
Slika 17. Prikaz treniranja podataka pomoću dubokog učenja..... 17
Slika 18. Prikaz sažetka modela i prvih 5 pokušaja treniranja 18
Slika 19. Funkcija clean_text..... 18
Slika 20. Prikaz pomoćne funkcije bag_of_words i pred_class funkciju 19
Slika 21. Primjer upita „Šta je danas za jesti?“ 20
Slika 22. get_response funkcija 20
Slika 23. Inicijalizacija Flask aplikacije..... 21
Slika 24. GET metoda za vraćanje odgovora klijentu..... 22
Slika 25. Prikaz GET metode za vraćanje odgovora koristeći Postman 22
Slika 26. Podaci za slanje email-a..... 23
Slika 27. Funkcija za slanje e-mail-a 23
Slika 28. Prikaz slanje POST zahtjeva putem Postman-a 24
Slika 29. Prikaz poruke na Gmail-u 24
Slika 30. Prikaz metode „run“ nad objektom „app“ 24
Slika 31. Prikaz izgleda početne stranice sa „headerom“ i „footerom“..... 25
Slika 32. Prikaz početne stranice na mobilnom uređaju 26

Slika 33. Prikaz „O projektu“ i „Kontakt“ stranice 26
Slika 34. Izgled stranice „FIPUbot“..... 27
Slika 35. Prikaz komponente „UserChatTextBox“..... 28
Slika 36. Prikaz komponente „BotChatTextBox“ 29
Slika 37. Prikaz sadržaja datoteke „service.js“..... 29
Slika 38. Prikaz sadržaja datoteke „store.js“ 30
Slika 39. Prikaz koda za ispis poruka..... 30
Slika 40. Prikaz funkcije „sendMessage“..... 31
Slika 41. Prikaz funkcije za brisanje poruke 32
Slika 42. Prikaz razmjenjivanja poruka između klijenta i „bota“..... 32
Slika 43. Prikaz slanja emaila pomoću „Kontakt“ stranice..... 33
Slika 44. Prikaz dolaznog e-mail-a 33
Slika 45. Status vatrozida Raspberry-ja 34
Slika 46. Primjer izgleda datoteke za konfiguraciju servera 35
Slika 47. Primjer app.wsgi datoteke..... 35
Slika 48. Prikaz pokrenutog apache2 server na Raspberry-ju..... 36
Slika 49. Primjer korištenja „FIPUbot-a“ preko domene..... 36