Integracija chatbota u Django REST framework

Radić, Sven

Undergraduate thesis / Završni rad

2024

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

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:817454

Rights / Prava: In copyright/Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-13**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni prijediplomski studij Računarstvo

INTEGRACIJA CHATBOTA U REST FRAMEWORK

Završni rad

Sven Radić

Osijek, 2024. godina.

Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju

Ocjena završnog rada na sveučilišnom prijediplomskom studiju

Ime i prezime pristupnika:	Sven Radić
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	R4708, 28.07.2021.
JMBAG:	0165091826
Mentor:	doc. dr. sc. Hrvoje Leventić
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Integracija chatbota u Django REST framework
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada:	Izraditi web aplikaciju chatbot koja će se integrirati u Django REST framework ili neki drugi sličan razvojni okvir za kreiranje REST API-ja. Fokus rada je prikazivanje mogućnosti integracije chatbota u gotovu web aplikaciju koja sadrži prilagođeno odgovarajuće API REST sučelje. (Rezervirano: Sven Radić, 3. sveučilišni)

Datum prijedloga ocjene završnog rada od strane mentora:	18.09.2024.
Prijedlog ocjene završnog rada od strane mentora:	Izvrstan (5)
Datum potvrde ocjene završnog rada od strane Odbora:	25.09.2024.
Ocjena završnog rada nakon obrane:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	28.09.2024.

IZJAVA O IZVORNOSTI RADA

Osijek, 28.09.2024.

Ime i prezime Pristupnika:	Sven Radić
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4708, 28.07.2021.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Integracija chatbota u Django REST framework**

izrađen pod vodstvom mentora doc. dr. sc. Hrvoje Leventić

i sumentora

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

Potpis pristupnika:

SADRŽAJ

1. UVOD **1**

1.1. Zadatak završnog rada **1**

2. PREGLED PODRUČJA **2**

2.1. Što su chatboti **2**

2.2. Chatboti na tržištu **3**

2.2.1. Chatbox 3

2.2.2. Zoho SalesIQ 5

2.2.3. Botsify 6

2.2.4. ChatGPT 6

2.2.5. HubSpot Chatbot 7

2.3. OpenAI API **7**

2.3.1. Tokeni 8

2.3.2. Korištenje OpenAI API-ja 9

2.4. Usporedba ostalih chatbot aplikacija s vlastitim rješenjem **10**

3. TEHNOLOGIJE ZA CHATBOT **12**

3.1. Korištene tehnologije **12**

3.1.1. .NET 12

3.1.2. C# 12

3.1.3. React JS 13

3.1.4. JavaScript 13

3.2. React iz prespektive shadow DOM-a **14**

3.3. LLM-ovi i teorija o njima **19**

3.3.1. Arhitektura LLM-ova 21

4. REST API ZA CHATBOT **23**

4.1. REST **23**

4.2. Metode resursa **24**

4.3. Integracija chatbot API-ja s OpenAI **26**

4.4. Autentifikacija i autorizacija **29**

5. INTEGRACIJA CHATBOTA U WEB APLIKACIJU POMOĆU REST API-JA **31**

5.1. Poslužiteljski dio aplikacije **31**

5.1.1. Baza podataka i modeli 31
5.1.2. Chatbot kontroler 32
5.2. Klijentski dio aplikacije 37
5.2.1. Arhitektura 37
5.2.2. Sučelje 39
6. ZAKLJUČAK 43
Sažetak 46
Abstract 47

1. UVOD

Današnji život prosječnog studenta, zaposlenika u IT branši, profesora, učenika ili pak običnih ljudi gotovo je nezamisliv bez umjetne inteligencije i njenih benefita koje pruža kroz razne alate putem weba ili mobitela. Iako je umjetna inteligencija prisutna već desetljećima, s razvojem novih tehnologija, posebno jezičkih modela velikih razmjera (LLM-ova), poput ChatGPT-a, njezin utjecaj na svakodnevni život i rad dodatno je naglašen. Ovi modeli omogućuju računalima obavljanje zadataka koji su do sada zahtijevali ljudsku inteligenciju, kao na primjer zaključivanje i snalaženje u novim situacijama te konstantno učenje kroz razvoj. Nagli rast tehnologija na internetu i njegov globalni učinak doveli su to velike transformacije fokusa tržišta s fizičkih mjesta na ona virtualna u obliku web stranica, aplikacija, i platformi. Takav drastičan prijelaz zahtijeva veliku količinu prilagodbe s korisničke strane, što velikom broju ljudi zna stvarati probleme. Upravo u tom dijelu prilagodbe na nove tehnologije chatbotovi i umjetna inteligencija dolaze do izražaja. Cilj chatbotova je poboljšati interakciju između ljudi i računala pružajući korisnicima brze i točne odgovore na njihova pitanja. Povećanje efikasnosti produktivnosti i zadovoljstva korisnika kroz automatizaciju interakcija je svakako bitna stavka. Web aplikacija koja integrira sučelje chatbota prolazi sve navedene kriterije, uspješno je izvršena te će uz detaljno objašnjenje zadatka završnog rada biti predstavljenje glave karakteristike aplikacije. Također, upoznat će se i usporediti glavni chatbotovi na tržištu kao što su Chatbox i Zoho SalesIQ. Nakon toga slijedi upoznvanje s OpenAI API-jem koji je srž aplikacije koja se opisuje u ovom radu. U 3. poglavlju biti će opisane korištene tehnologije i teorija o velikim jezičnim modelima, a u 4. poglavlju uvod u REST i njegove karakteristike, te uvod u poslužiteljski dio aplikacije. Za kraj 5. poglavlje u kojem je detaljno opisana izrada aplikacije prvo poslužiteljski pa klijentski dio.

1.1. Zadatak završnog rada

Web aplikacija unutar koje je integriran chatbot omogućava korisnicima autentikaciju i autorizaciju te omogućuje stvaranje chatova za komuniciranje sa chatbotom, te pregled svih prijašnjih chatova. Jedan od temeljnih zadataka završnog rada bio je kreirati skriptu koja se može ubaciti u bilo koju .html datoteku bez posebne konfiguracije te će, skripta pružati sučelje chatbota. Uključivanjem skripte pojavljuje se gumb koji klikom otvara i zatvara sučelje chatbota. Skripta omogućuje normalno korištenje dokumenta ili aplikacije u koju je ubačena. Web aplikacija je implementirana korištenjem ASP.NET frameworka za poslužiteljski server, SQLite baze podataka i React Js za klijentski dio (sučelje chatbota). U radu će biti detaljno opisane funkcionalnosti i arhitektura web aplikacije, kao i način njezinog korištenja.

2. PREGLED PODRUČJA

2.1. Što su chatboti

Chatboti su softverski alati (programi) koji koriste umjetnu inteligenciju (AI) za automatiziranu komunikaciju s korisnicima kroz tekstualna ili glasovna sučelja. Korištenjem prirodnog jezika, oni oponašaju ljudski razgovor, omogućujući korisnicima interakciju s računalnim sustavima na prirodan i intuitivan način.

Chatboti mogu biti jednostavni, poput rudimentarnih programa koji odgovaraju na jednostavne upite jednom rečenicom, ili vrlo sofisticirani, poput digitalnih asistenata koji uče i razvijaju se kako bi pružili sve veće razine personalizacije putem prikupljanja i obrade informacija. Mogu se implementirati na razne platforme, uključujući web stranice, mobilne aplikacije, društvene mreže i aplikacije za razmjenu poruka. Chatboti imaju jako širok spektar funkcionalnosti, a neke on njih su:

- Pružanje odgovora na upite korisnika, gotovo sva poslovanja imaju svoje web aplikacije i na njima postoji odjeljak za upite, to se u pravilu zove dio za često postavljena pitanja (FAQ), te se u tom pogledu chatboti najčešće koriste.
- Pomažu u navigaciji i korisnike navode kroz specifične procese koji se izvršavaju na aplikaciji. Kroz korisničke inpute chatboti vrlo brzo „uče" od svojih korisnike te imaju mogučnost prilagođavanja svog tona i vrste odgovora.
- Vrlo su korisni i za automatizaciju zadataka, mogu izvršavati zadatke poput zakazivanja termina, naručivanja i pružanja podrške u stvarnom vremenu.
- Prikupljaju podatke, sakupljaju povratne informacije od korisnika i prate njihove interakcije za analizu i poboljšanje usluga.

Oracle Cloud [1] u svom članku *What is a chatbot?* kaže kako postoje dvije glavne vrste chatbota:

1. *Task-oriented* (deklarativni) chatbotovi fokusiraju se na programe s jednom svrhom koji pružaju automatizirane odgovore na korisničke upite koristeći pravila, NLP (natural-language processing) i minimalno strojno učenje. Oni su visoko specifični i strukturirani, najbolji za funkcije podrške i usluga, i često se koriste za interaktivne FAQ.
2. *Data-driven and predictive* (konverzacijski) chatbotovi poznati i kao virtualni ili digitalni asistenti, vrlo su sofisticirani i personalizirani. Koriste razumijevanje prirodnog jezika, strojno učenje i prediktivnu inteligenciju kako bi učili iz interakcija s korisnicima, nudili

personalizirane preporuke i predviđali potrebe korisnika. Primjeri uključuju Appleovu Siri i Amazonovu Alexu.

S druge strane, autorica Bella Church [2] iz IBM-a u svom članku „*5 types of chatbot and how to choose the right one for your business”* prezentira 5 vrsta chatbota:

1. Menu or button-based chatbots
2. Rules-based chatbots
3. AI-powered chatbots
4. Voice chatbots
5. Generative AI chatbots

Ovih 5 vrsta se u suštini mogu dodatno podijeliti u 2 osnovne kategorije o kojima priča Oracle Cloud, u prvu Task-oriented kategoriju bi se ubrajali 1. i 2., a u drugu Data-driven and predictive kategoriju preostale tri.

2.2. Chatboti na tržištu

Chatboti su u zadnjih par godina dosegli veliku popularnost, prema tome na tržištu već postoji velik broj chatbota koji dolaze sa svojim specifičnim svojstvima i atributima. U ovom radu biti će predstavljena 5 konkretnih primjera chatbotova.

2.2.1. Chatbox

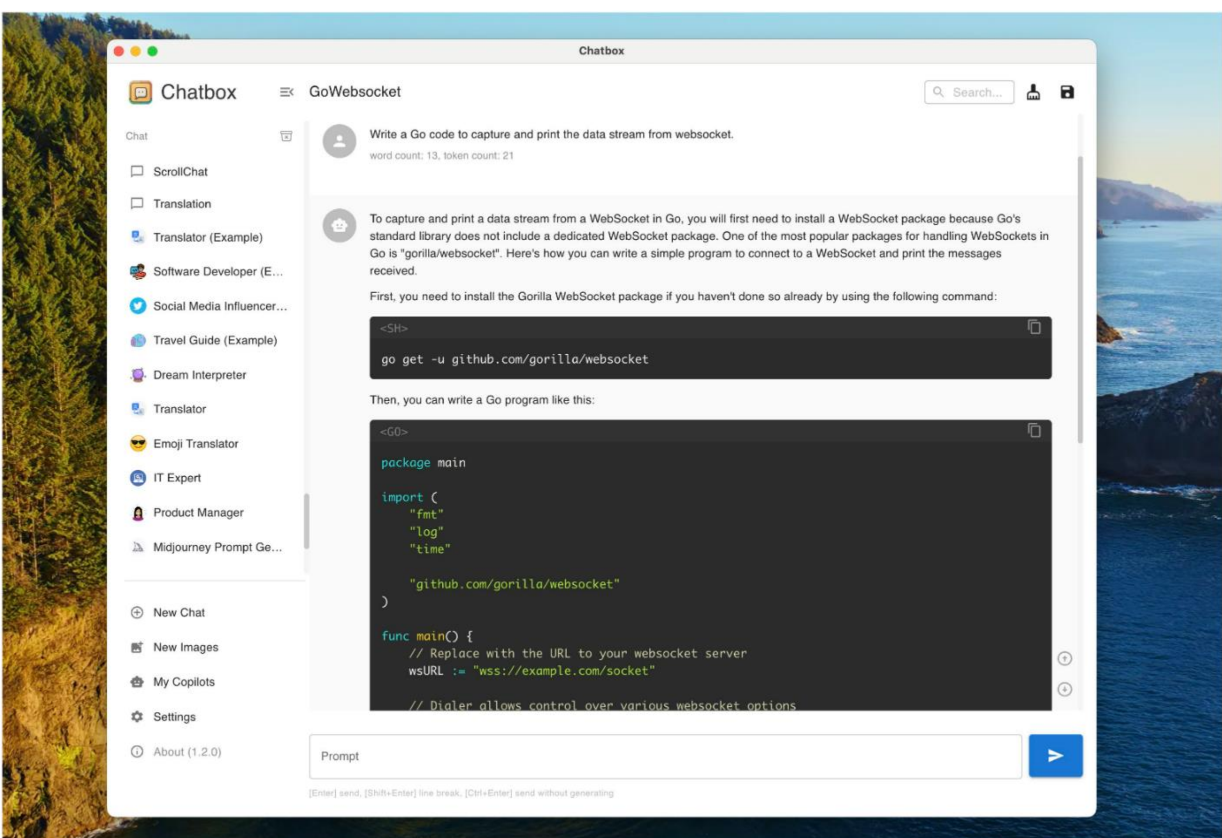
Chatbox je *user-friendly* desktop aplikacija za AI Modele i LLM-ove (detalje o LLM modelima biti će opisani u 3. poglavlju), oni uključuju GPT, Claude, Gemini, Ollama i mnoge druge. Može se jednostavno instalirati preko github repozitorija, stvorio ga je programer Bin-Huang koji je odlučio svog Chatboxa napraviti open-source, što znači da je sav kod koji čini aplikaciju javno dostupan na githubu i apsolutno svatko tko ima volje, znanja i želje može pomoći u razvoju aplikacije i dati svoj doprinos. Repozitorij Chatboxa je vrlo poznat i ima preko 20 000 github zvjezdica.

Bih-Huang [3] u opisu repozitorija piše kako je stvorio inovativnu chatbot desktop aplikaciju iz razloga što je osjetio potrebu za virtualnim asistentom dok je debuggirao svoj kod, te vjeruje da će njegov projekt pomoći mnogim ljudima koji se nalaze u sličnim situacijama. Kaže kako je i sam iznenađen s popularnosti Chatboxa te da ga ljudi koriste u raznolikim okolnostima, tako da se njegov izum ne mora koristiti samo u programerske svrhe. Aplikacija je napravljena za Windows,

MacOs i Linux kao desktop chatbot, ali daljnim unaprijeđivanjem Bin-Huang omogućio je korištenje aplikacije na mobitelu za Andriod i iOS operacijske sustave.

Glavna svojstva Chatboxa su:

- Lokalno pohranjivanje podataka
- Jednostavno preuzimanje aplikacije u obliku paketa
- Podrška za rad s različitim LLM-ovima
- Generiranje slika pomoću Dall-E-3
- Stvaranje timova i dijeljenje API resursa
- Cross-platform
- Pruža izmjenu Light i Dark Theme
- Mogučnost rada na različitim jezicima



Slika 2.1. Dizajn Chatbox-a na MacOs-u

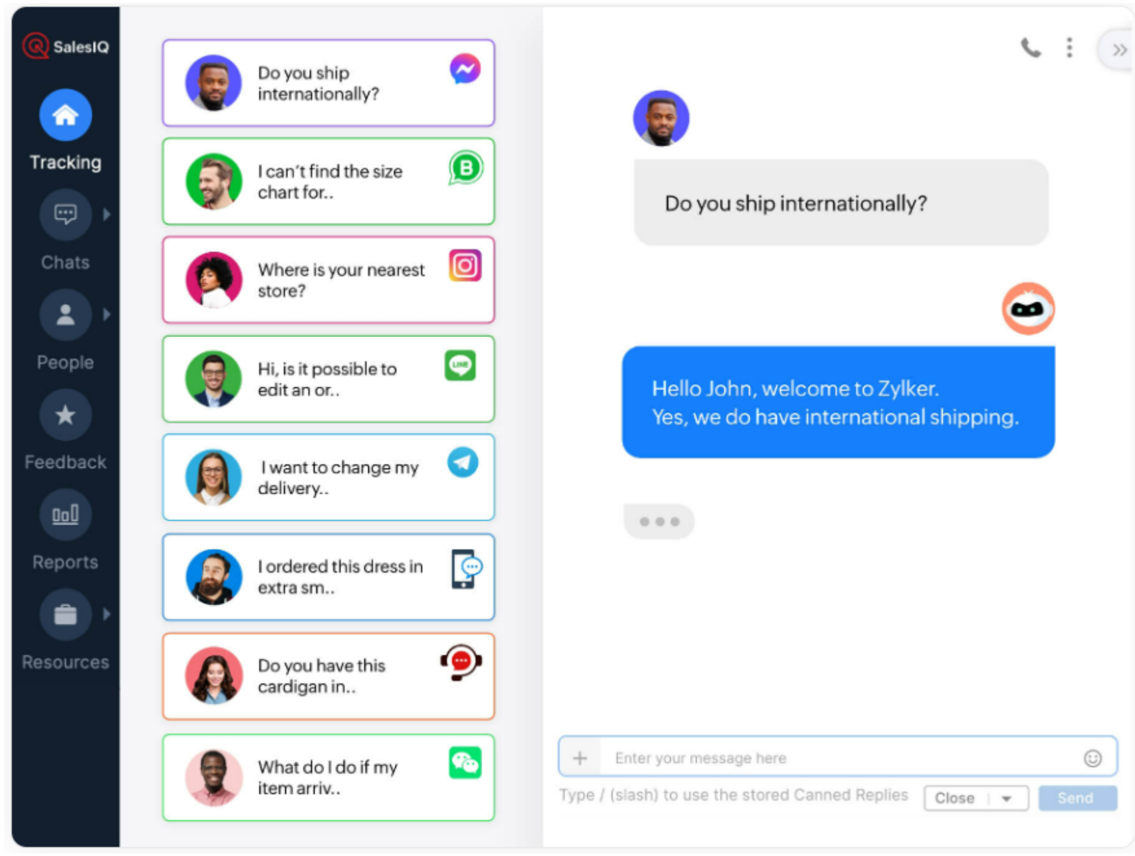
2.2.2. Zoho SalesIQ

Zoho SalesIQ [4] je platforma za angažman korisnika, dizajnirana je sa ciljem da opremi razne tvrtke alatima za interakciju s posjetiteljima web stranica, skupljanja podataka o njihovom ponašanju i pretvaranje potencijalnih kupaca u stvarne kupce. Omogućava live chat, pruža detaljnu analizu kroz jednu platformu.

Platforma omogućuje korištenje chatbota nazvanog Zobot [5] koji pomoću umjetne inteligencije automatirzira interakcije s kupcima na web stranici. Njegov zadatak je izvršavanje različitih zadataka korisničke podrške uključujući FAQ i asistiranje u specifičnim procesima, a dizajn sučelja vidljiv je na slici 2.2. Zobot ima različita specifična svojstva i funkcionalnosti, a ovo su njegove najveće kvalitete:

1. Personalizirano kreiranje chatbota: omogućuje korisnicima kreiranje vlasititog chatbota pomoću sučelja za povlačenje i spuštanje, što omogućuje tvrtkama prilagođavanje ponašanja bota vlastitm potrebama.
2. Automatizacija odgovora po pravilima ili umjetnoj inteligenciji: Zobot može biti isprogramiran koristeći automatizirana pravila koja slijede unaprije definirane skripte i logiku, ili može davati odgovore pomoću umjetne inteligencije.
3. Praćenje ponašanja: Zobot komunicira s posjetiteljima na temelju njihovih interesa te ima mogućnost pokrenuti specifične teme kada posjetitelji provedu određeno vrijeme na stranici.
4. Analitika u stvarnom vremenu i neovisnost o platformi: daje podatke u stvarnom vremenu i pruža mogučnost komuniciranja s kupcima preko mobitela, računala i društvenih mreža

Također, bitna stavka Zoho SalesIQ-a kao platforme i njihovog Zobota je to što su omogučili integriranje svih usluga kroz jednostavno dodavanje JavaScript koda u head ili body sekciju HTML dokumenta. Dodaje se par linija koda koji obavezno u sebi ima vrijednost widget koda koji se dobije prilikom pretplate na njihovoj stranici.



Slika 2.2. Dizajn Zoho SalesIQ Zobot-a

2.2.3. Botsify

Botsify [6] je platforma koja u obliku chatbota pruža VIP tretman svakom pojedinom kupcu. Ta funkcionalnost je razlog zbog kojeg se Botsify ističe od konkurencije, uz VIP tretman kupci dobivaju izuzetno brze odgovore. Transformacija web formi u topao razgovor između kupca i chatbota uvelike pomaže u prodaji. Botsify nudi i jednostavno kreiranje chatbota putem drag-and-drop editora bez potrebe za programiranjem. Podržava više jezika što omogućava tvrtkama pristup većem broju kupaca i izgradnji većeg broja klijenata. Kao i prethodni chatboti omogućuju automatizaciju različitih funkcija kao što su odgovaranje na često postavljena pitanja i pružanje osobnih poruka, uz to platforma nudi analitičke alate kroz koje se mogu pratiti performanse chatbota i razumijevanje ponašanja korisnika.

2.2.4. ChatGPT

Trenutno najpoznatiji chatbot na tržištu je ChatGPT [7], to je platforma koja uz pomoć modela umjetne inteligencije generira tekst. Pri tome koristi napredni veliki jezični model koji je razvijen od tvrtke OpenAI, nešto više od OpenAI-u u poglavlju 2.3. Glavna funkcionalnost ChatGPT-a je to što ima mogučnost generiranja odgovora u stvarnom vremenu nalik ljudskom, tako da razgovor

između računala i čovjek izgleda kao dijalog između dva živa bića. Pored jednostavnih odgovora ChatGPT može rješavati kompleksne matematičke zadatke, prevoditi tekst s jednog jezika na drugi, *debuggirati* kod i davati implementacijske detalje, pisati pjesme i priče te vršiti razne klasifikacije podataka. Jedan od problema kod ChatGPT-a je što mu je baza podataka doseže rujan 2023. godine prema najnovijim podacima, što znači da na pitanja od događajima nakon rujna 2023. ovaj močni AI model nema odgovor.

2.2.5. HubSpot Chatbot

HubSpot [8] je CRM (engl. *customer relationship management*) sustav koji dolazi s ugrađenim chatbotom za automatizaciju razgovora s posjetiteljima web stranica, generiranje potencijalnih klijenata, podršku korisnicima i povećanje prodaje. CRM je sustav koji sam po sebi sprema određene podatke o korisnicima, te iz tog razloga nudi poboljšanu personalizaciju tokova razgovora, chatbot tako može prepoznati o kojem se korisniku radi, njegovim preferencijama kao i proizvode koje je prije pregledavao. Nadalje, kada je chatbot ograničen u nekom pogledu, korisnik je preusmjeren na *live chat* s trenutnim aktivnim agentom koji odgovara na složenija pitanja.

2.3. OpenAI API

OpenAI je tvrtka koja se bavi istraživanjem i razvojem umjetne inteligencije. Krajnji cilj im je omogućiti da čovječanstvo iskusi snagu umjetne inteligencije, te na kraju ima velike koristi od nje same. CCN [9] u svom članku *What is OpenAI and what does it do?* u kojem Alisha Bains piše kako je OpenAI neprofitna organizacija koja se bavi istraživanjem umjetne inteligencije, dodaje kako je OpenAI osnovan s ciljem da kreira nekakav produkt koji poboljšava ljudski život, te da je pri tom siguran, pouzdan i konzistentan. OpenAI je to uspio do neke mjere, napravili su ChatGPT kojeg koriste gotovo svi ljudi koji imaju računalo ili mobitel u svojoj blizini. No mora se napomenuti da je to tek početak jer *GPT* u imenu je skraćenica za Generative Pre-Trained Transformer, što jasno ukazuje da je to tek početak razvoja i korištenja umjetne inteligencije od strane OpenAI kompanije.

OpenAI, kao i mnoge druge tvrtke imaju svoje API-je (Application Programming Interface) preko kojih otvaraju vrata svoje tvrtke javno na korištenje. To mogu biti API pozivi za nekakve specifične liste podataka ili nešto slično, no u ovom slučaju korišten je OpenAI API koji pri pozivu prima listu objekata koji reprezentiraju poruke, svaka poruka ima dva svojstva:

1. Role: koja opisuje je li poruka poslana od strane korisnika ili chatbota
2. Content: string zapis sadržaja poruke

Nakon toga OpenAI API radi svoj posao te šalje nazad istu tu listu poruka koja sadrži jednu poruku više od one liste koja je poslana, a ta poruka koja je višak zapravo reprezentira odgovor chatbota, u ovom slučaju ChatGPT-a jer je endpoint OpenAI API-ja koji je zadužen za odgovaranje na poruke pogonjen ChatGPT-om, ali nešto više će se govoriti o tome u poglavlju gdje se opisuje sam poslužiteljski server chatbot aplikacije koja je tema ovog rada.

Na službenoj stranici OpenAI Docs Developer quickStart [10] napisano je da OpenAI API služi kao jednostavno sučelje namjenjeno programerima za kreiranje inteligentnih slojeva u njihovim aplikacijama. Koristi se na način da se pošalje tekst pa model GPT-4o korištenjem svojih naprednih neuronskih mreža generira tekst koji šalje kao odgovor.

Naravno, kao i kod gotovo svakog drugog API-ja, potrebno se registrirati/prijaviti na OpenAI platformu, nakon toga dobiti API ključ, te pomoću tog ključa slati API zahtjeve na odabrani URL putem HTTP-a.

Za razumijevanje načina na koji OpenAI API obrađuje tekst važno je znati da on u pozadini koristi veliki jezični model (LLM) koji primljeni tekst u obliku tokena učitava u mrežu te kao izlaz generira jedan po jedan token, dok ne generira kompletan tekst odgovora. Taj generirani tekst onda OpenAI API vraća nazad korisniku. Uvod u tokene i njihovu svrhu dan je u idućem poglavlju.

2.3.1. Tokeni

U kontekstu umjetne inteligencije i GPT modela prema GPT Workspace – u [11] tokeni su elementarna jedinica za mjerenje duljine teksta. Preciznije tokeni su nakupine ili središta znakova u tekstu, to zvuči kao pojam „riječ“ pa se iz navedenog može zaključiti da token ima jednako značenje kao i riječ. U nekim situacijama tokeni uistinu jesu jednaki riječima, ali to ne mora biti stalni slučaj. Jer tokene određuje posebni mehanizam koji se zove tokenizer, a tokenizer pri određivanju tokena gleda sve znakove uključujući emotikone, razmake, interpunkcijske znakove kao potencijalne tokene, također tokenizer gleda i jezik na kojemu je tekst napisan, stoga isti tekst napisan na engleskom jeziku imati će manje tokena nego tekst napisan primjerice na hrvatskom jeziku. Engleske riječi u prosjeku imaju 1,3 tokena, dok riječi na španjolskom i francuskom jeziku sadrže oko 2 tokena. OpenAI API generira tokene u kombinaciji s parametrom max_tokens koji određuje maksimalni broj tokena generiran za svaki API zahtjev. Generairanje tokena nije

besplatno, pa tako svaki token ima cijenu koja se naplaćuje po 1000 tokena i ovisi o GPT modelu koji se koristi.

Tokenizer vrši operaciju tokenizacije nad ulaznim tekstom, tokenizacija je proces rastavljanja teksta na manje dijelove koje model može razumijeti, u ovom slučaju tokenizacija rastavlja ulazni tekst na tokene. Nakon tokenizacije model pretvara tokene u visoko-dimenzionalne vektore, taj proces zove se ugrađivanje (engl. embedding) i važan je iz razloga što omogućava modelu razumijevanje odnosa unutar teksta i složenosti tokena. Kada model razumije kontekst i ima pregled svih tokena može početi generirati tekst. To se radi token po token, koristeći primljene tokene i kontext poruke kako bi predvidio sljedeći token. Za to se koriste napredni algoritmi koji uključuju distribuciju vjerojatnosti. Proces generiranja ponavlja se dok se ne ispuni kriterij ili ne dosegne maksimalni broj tokena. Nakon toga generirani tokeni prolaze kroz proces dekodiranja gdje se tokeni spajaju u riječi i rečenice, te nastaje smisleni tekst koji se isporučuje kao odgovor.

2.3.2. Korištenje OpenAI API-ja

Proces korištenja OpenAI API-ja je vrlo jednostavan i može se opisati kroz sljedeće korake:

1. Registracija/prijava: prvo je potrebno posjedovati račun na OpenAI stranici pomoću kojeg se prati analitika tokena i pretplata koju korisnik ima u koju je uključen pregled GPT modela koji se koristi i maksimalni broj tokena.
2. API ključ: nakon registracije može se API ključ koji se koristi prilikom autentifikacije svakog zahtjeva na OpenAI API.
3. Autentifikacija: vrši se korištenjem API ključa koji se dodaje u zaglavlje HTTP zahtjeva. Primjer izgleda zaglavlja: "Authorization: Bearer YOUR_API_KEY"
4. Slanje zahtjeva: OpenAI API ima različite endpointe za različite modele na koje se mogu slati zahtjevi, endpoint korišten za potrebe ovog chatbota je "https://api.openai.com/v1/chat/completions" , zahtjev se šalje kroz post request u koji se može postaviti različiti parametri koji utječu na formirani odgovor, npr. temperature. Nad tijelom zahtjeva vrši se serializacija te se podaci šalju u JSON formatu.
5. Obrada odgovora: API odgovori dolaze u različitim formatima ovisno o endpointu koji se koristi, u obradi odgovora potrebno je odabrati informacije koje su potrebne te ih prosijediti dalje kroz aplikaciju. Tijelo odgovora dolazi u JSON formatu a tekst kojeg je GPT model generirao nalazi se u ugniježđenom objektu koji nosi ime *choices*, tip podatka je string i u primjeru sa slike 2.3. nalazi se pod svojstvom *text*.

```
1  {
2    "choices": [
3      {
4        "finish_reason": "length",
5        "index": 0,
6        "logprobs": null,
7        "text": "\n\n\"Let Your Sweet Tooth Run Wild at Our Creamy Ice Cream Shack"
8      }
9    ],
10   "created": 1683130927,
11   "id": "cmpl-7C9Wxi9Du4j1lQjdjhxBl022M61LD",
12   "model": "gpt-3.5-turbo-instruct",
13   "object": "text_completion",
14   "usage": {
15     "completion_tokens": 16,
16     "prompt_tokens": 10,
17     "total_tokens": 26
18   }
19 }
```

Slika 2.3. Primjer OpenAI API odgovora

U API odgovoru vidljive su još neke informacije koje su karakteristične za trenutni request/response pošto je API po REST arhitekturi bez stanja, ali o tome više u poglavlju 4. REST API za chatbot. Podaci koji se mogu iščitati su datum kreiranja u šiframa, id odgovora, GPT model koji se koristio i broj tokena.

2.4. Usporedba ostalih chatbot aplikacija s vlastitim rješenjem

Prilikom usporebne konkurentskih chatbot aplikacija s rješenjem koje je opisano u ovom završnom radu mogu se izvući interesantni zaključci. Autorov rad najbolje je opisati kao kombinacija između dva prethodno opisana rješenja, Zoho SalesIQ Zobot i Chatbox. Dakako, bez naprednih stavki koje Zoho SalesIQ nudi kao što su praćenje analitike web stranice, ponašanja korisnika na web stranici i prilagodljivo ponašanje chatbota ovisno o klijentovim interesima. Autorov chatbot nudi jednostavne funkcinalnosti chatbota preko OpenAI API-ja i njegovog GPT 3.5 modela, što je sličnije Chatbox-u. Sličnost i inspiracija koju je pridonijelo rješenje platforme Zoho SalesIQ je izražena u jednostavnoj integraciji chatbot sučelja, pod tim se misli na ubrizgavanje jedne linije JavaScript koda u postojeći HTML dokument koja omogučuje potpuni prikaz chatbota u bilo kojem postojećem okruženju. Također prednost autorova rada je što koristi OpenAI API koje

omogučuje napredne, sadržajne i precizne odgovore na gotova sve tipove pitanja koje korisnik može postaviti.

3. TEHNOLOGIJE ZA CHATBOT

Poglavlje opisuje poznate tehnologije za razvoj chatbota, počevši od onih koje su korištene za razvoj aplikacije koja je tema ovog završnog rada, do pojave u razvoju web-a koji se zove shadow DOM. Zatim će biti opisani alati pod nazivom LLM, njihova teorija i široka primjena.

3.1. Korištene tehnologije

3.1.1. .NET

.NET je razvojna platforma otvorenog koda koju je razvio Microsoft za kreiranje raznovrsnih aplikacija. Osnovana je na izvršnom okruženju Common Language Runtime (CLR), koje se brine o upravljanju memorijom, sigurnom pristupu memoriji i sigurnosti tipova. CLR omogućava prenosivost između različitih platformi, operativnih sistema i arhitektura, što znači da za pokretanje .NET koda nije potrebno ponovno kompajliranje, već samo instalacija odgovarajućeg izvršnog okruženja. Podržava jezike kao što su C#, F# i Visual Basic. Za dodavanje kompajliranih binarnih datoteka u projekte koristi se NuGet upravljač paketima. .NET, ranije poznat kao .NET Core, je nasljednik .NET Frameworka, razvijen s ciljem postizanja modularnosti, fleksibilnosti i prenosivosti u eri cloud računarstva [12].

3.1.2. C#

Za razvoj poslužiteljskog dijela aplikacije korišten je programski jezik C#, jedan od najpopularnijih objektno-orijentiranih jezika. C# je strogo tipiziran jezik s unificiranim sustavom tipova, gdje svi tipovi, uključujući primitivne, nasljeđuju iz osnovnog tipa „object". Podržava kako referentne, tako i vrijednosne tipove. Zbog svojih karakteristika, uključujući prikupljanje otpada, ništavne tipove i rukovanje pogreškama, C# olakšava stvaranje robusnih aplikacija. U jezik su integrirani LINQ (Language-Integrated Query) upiti koji omogućavaju uniformnu sintaksu za sve upite, neovisno o tehnologiji i bazi podataka. Ovi upiti omogućuju filtriranje, grupiranje i sortiranje uz minimalnu količinu koda [13]. Za razvoj aplikacije u ovom radu korištena je sintaksa temeljena na metodama, a primjer i usporedba s regularnom sintaksom prikazani su na slici 3.1.

```
var standardQuery = from product in products   where
                product.Id < 3
orderby product.Name               select
product;
var methodQuery = products.
Where(product      =>      product.Id      <      3).
OrderBy(product => product.Name);
```

Slika 3.1. Usporedba LINQ sintakse

3.1.3. React JS

Za razvoj klijentskog dijela aplikacije korištena je popularna biblioteka React JS [14]. React JS je JavaScript biblioteka razvijena od strane Facebooka za izradu korisničkih sučelja. Glavna prednost Reacta je njegova sposobnost stvaranja brzih i dinamičnih web aplikacija koristeći komponentni pristup. Komponente su samostalni, ponovno iskoristivi dijelovi koda koji olakšavaju razvoj složenih korisničkih sučelja. React koristi virtualni DOM (Document Object Model), što omogućuje učinkovitu manipulaciju i ažuriranje korisničkog sučelja bez potrebe za potpunim osvježavanjem stranice. Ova biblioteka podržava jednosmjerno vezanje podataka, što doprinosi predvidljivosti ponašanja aplikacije. Osim toga, React se lako integrira s drugim bibliotekama i radnim okvirima, zbog čega je i odabran za razvoj klijentskog dijela ovog rada.

3.1.4. JavaScript

JavaScript[15] je dinamički, interpretirani programski jezik široko korišten za razvoj web aplikacija. Kao jedna od temeljnih tehnologija weba, zajedno s HTML-om i CSS-om, JavaScript omogućuje interaktivnost i dinamičnost na web stranicama. Posebnost JavaScripta je njegova sposobnost izvršavanja na klijentskoj strani, unutar preglednika, čime omogućuje brzo reagiranje na korisničke akcije bez potrebe za komunikacijom s poslužiteljem. JavaScript podržava različite paradigme programiranja, uključujući objektno-orijentirano, funkcionalno i imperativno programiranje. Jedna od ključnih značajki je asinkrono programiranje putem callback funkcija, promisa i async/await sintakse, što omogućuje učinkovit rad s podacima iz različitih izvora. Također, zahvaljujući bogatom ekosustavu biblioteka i okvira poput Reacta, Node.js-a i Angulara, JavaScript je postao nezaobilazan alat u modernom razvoju web i mobilnih aplikacija. Jedna od značajki JavaScripta je što se konstantno mijenja, to jest, da se razvija izuzetnom brzinom i uvijek je u koraku s najnovijim tehnologijama te ga to čini jednim od najpopularnijih programskih jezika današnjice.

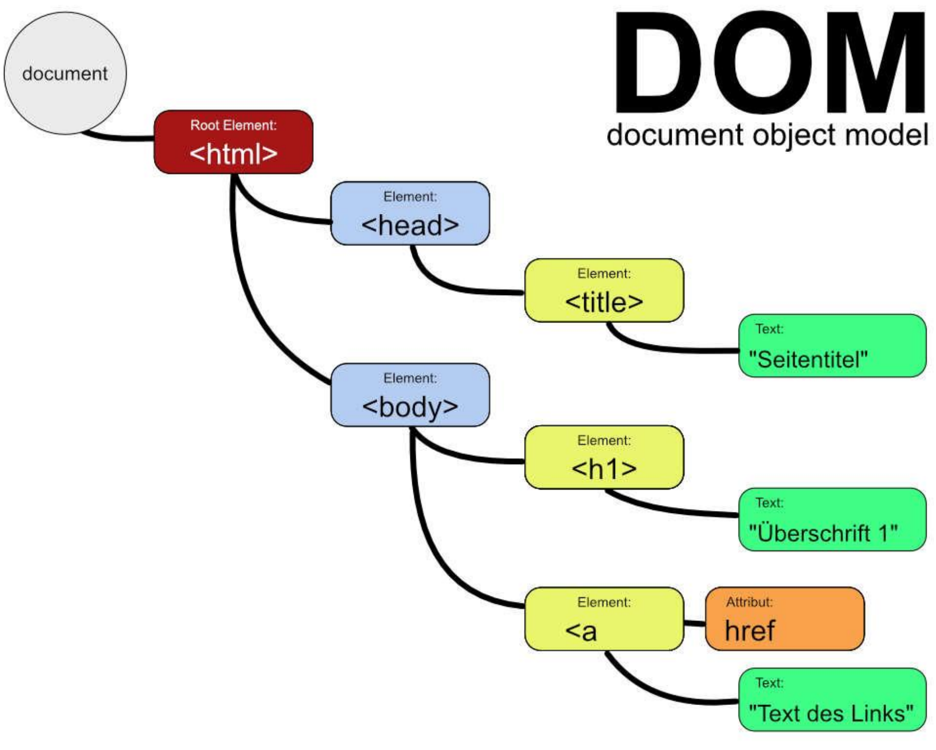
3.2. React iz prespektive shadow DOM-a

DOM (Document Object Model) [16] je stablasta struktura koja predstavlja podatkovni prikaz koji čine strukturu i sadržaj web dokumenta. Jednostavnije rečeno, DOM predstavlja jednu web stranicu s kompletnim sadržajem koji se na njoj nalazi, to je programsko sučelje za web dokumente i služi kao posrednik kako bi se stranici moglo pristupiti putem koda koristeći razne programske jezike primjerice JavaScript. Dakle, web stranica je dokument nastavka .html, iz kojega se može upravljati strukturom i izgledom stranice preko HTML i CSS jezika i to se zove statičko oblikovanje stranice. No postoji još jedan način za postizanje funckionalnosti oblikovanja strukture stranice i vrši se putem JavaScripta upravo preko DOM-a i naziva se dinamičko olikovanje stranica. Na slici ispod (Slika 3.2.) prikazan je primjer korištenja DOM-a, *document.querySelectorALL("p")* vraća polje svih HTML *p* tagova (<p>) na stranici i sprema ih u constantu *paragraphs*, te kasnije prosljeđuje ime prvog taga u nizu kao argument funckije *alert.*

```
Const paragraphs = document.querySelectorAll("p");
// paragraphs[0] is the first <p> element
// paragraphs[1] is the second <p> element, etc.
alert(paragraphs[0].nodeName);
```

Slika 3.2. Primjer korištenja DOM-a

DOM sačinjava mnoštvo povezanih API-ja. Osnovni DOM definira entitete koji opisuju bilo koji dokument i njegove objekte. To se po potrebi proširuje drugim API-jima koji dodaju nove funkcionalnosti i mogućnosti. Na primjer, HTML DOM API dodaje podršku za prikazivanje HTML dokumenata u osnovni DOM, dok SVG API dodaje podršku za prikazivanje SVG dokumenata. Kao što je prije rečeno DOM je struktura stablaste građe, što znaći predstavlja dokumente u obliku hijerarhijskog stabla, gdje su svi elementi dokumenta organizirani kao čvoru tom stablu. Najčešće je prvi izvorni, korjenski čvor <html> element i on predstavlja cijeli dokument, on se dijeli da glavu <head> i tijelo <body> tagove koji također sadržavaju svoje elemente u obliku čvorova. Čvorovi su primjerice svaki <div>, <p> ili bilo koji drugi html ili xml tagovi unutar dokumenta. Teksutalni čvorovi su primjerice <p>Hello<p> „Hello" je također element u hijerarhiji. Kada se spomene hijerarhija misli se na odnos između čvorova, gdje svaki element može imati djecu i svako dijete ima svojeg jednog roditelja, dok roditelj može imati više djece. Valjalo bi spomenuti kako varijabla *document* u programskom jeziku služi kao pokazivač na korjenski čvor te se preko nje pristupa DOM-u. Na slici ispod (Slika 3.3.) je prikazana jedna takva hijerarhija koja predstavlja DOM u nekakvom HTML dokumentu.

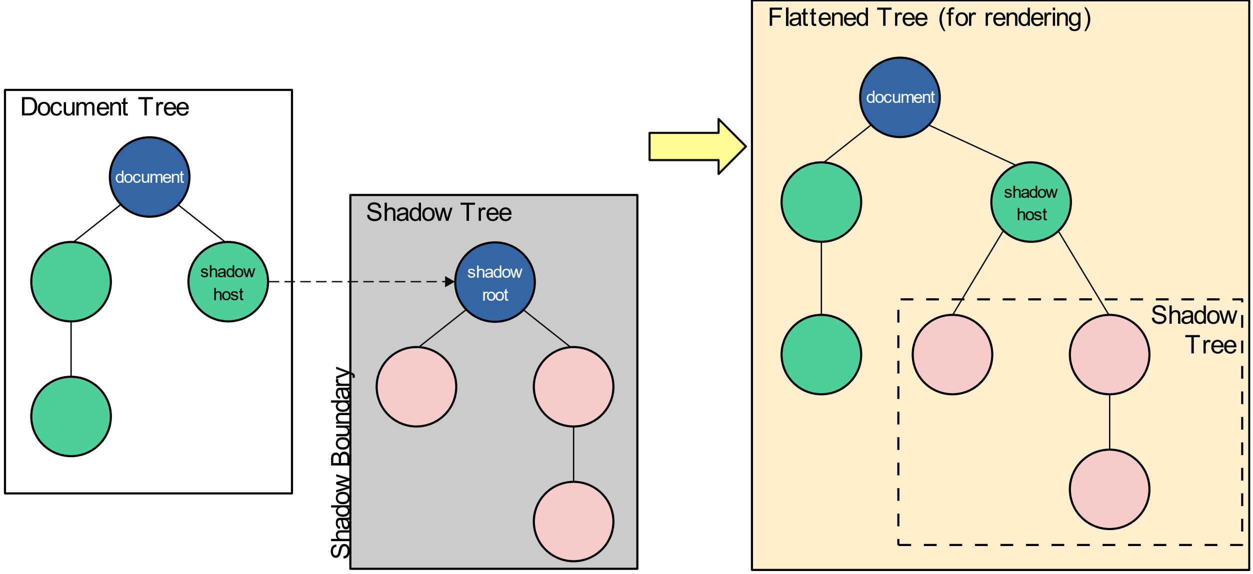


Slika 3.3. Vizualni primjer strukture DOM-a

Dakle, DOM je jedna vrlo snažna stavka i ključna tehnologija u web programiranju koja se koristi u gotovo svakoj web aplikaciji i omogućuje dinamičnu manipulaciju strukturom i sadržajem web stranica. ali postoji jedan potencionalan problem. Međutim, kako aplikacije postaju sve složenije, suočavamo se s izazovom održavanja modularnosti i izolacije između različitih dijelova aplikacije. Jedan od tih izazova jest izbjegavanje sukoba u stilovima i strukturi kada različite komponente dijele istu stranicu.

Jedan od osnovnih stupova objektno-orijentiranog programiranja (OOP) govori upravo o ovom problemu i pruža vrlo jednostavno rješenje, riječ je o enkapsulaciji. U OOP-u enkapsulacija je koncept začahurivanja svojstava, atributa i metoda u jednu klasu ili korisnički definiran tip podatka s mogučnostima skrivanja implementacijskih detalja od ostatka programa i pružanja javnog sučelja za korištenje funkcionalnosti klase. Dakako, programer sam odlučuje koje će metode, atribute i svojstva sakriti, a koje će javno otvoriti. Sličan pristup je potreban u web razvoju, gdje je potrebno da komponente budu samostalne i neovisne, kako bi se izbjegli sukobi između različitih dijelova stranice. Upravo se na enkapsulaciji temelji rješenje tog problema, a to je shadow DOM. Shadow DOM [17] je web standard koji omogučuje enkapsulaciju stila i oznaka unutar web komponenti. To znači da stilovi unutar Shadow DOM-a ne utječu na ostatak stranice, niti su pod utjecajem vanjskih stilova. Ova izolacija omogućuje razvoj modularnih i ponovno iskoristivih komponenti

koje se mogu koristiti u različitim projektima bez rizika od stilskih sukoba. Shadow DOM je dio Web Components standarda, koji također uključuje HTML predloške (templates), prilagođene elemente (Custom Elements) i HTML uvoze (HTML imports). Kreiranjem shadow DOM-a prvi put se uvodi scoped CSS, omogućujući programerima definiranje stilova i oznaka koji su enkapsulirani i izolirani od ostalih dokumentana, a funkcionira na sljedeći način. Shadow DOM stvara novu skrivenu stablastu strukturu, novi DOM koji je povezan s jednim čvorom u osnovnom DOM-u.



Slika 3.4. Ilustracija usporedbe osnovnog i shadow DOM-a

Terminologija za sliku iznad (Slika 3.4.) glasi ovako:

- Shadow host: čvor osnovnog DOM-a na koji se poveže shadow DOM
- Shadow tree: DOM stablo unutar shadow DOM-a
- Shadow boundary: granice shadow i osnovnog DOM-a
- Shadow root: korjenski čvor stabla unutar shadow DOM-a

Prava svrha shadow DOM-a je da se i dalje može njime upravljati dodavati elemente i postavljati atribute unutar shadow DOM-a, samo što to mora biti učinjeno dinamički. Enkapsulacija dolazi do izražaja u činjenici da se elementi i stilovi unutar i van shadow DOM-a ne miješaju jedni s drugima čak i ako su dio iste stranice ili aplikacije, Dakle kod unutar shadow DOM-a ne utječe na vanjski kod. Shadow DOM nije oduvijek bio javno dostupan programerima za korištenje, ali preglednici su ga već tada koristili. Primjerice, <video> element koji se u osnovnom DOM-u vidi samo kao <video> element, a unutar shadow DOM-a sadrži gumbe i kontrole za korištenje koji nisu vidljivi u osnovnom DOM-u.

Na slici ispod (Slika 3.5.) vidljiva je struktura HTML dokumenta koja ima jedan div element s identifikatorom „host“ i jedan span element s nekakvim tekstom.

```
<div id="host"></div>
<span>I'm not in the shadow DOM</span>
```

Slika 3.5. Primjer HTML dokumenta

U daljnjem primjeru na slici (Slika 3.6.) bit će prikazan proces postavljanja shadow DOM-a na shadow host u JavaScriptu. Poziva se funckija *attachShadow(),* te se dodaju čvorovi na shadow DOM jednako kao što se radi kod običnog DOM elementa. U ovom primjeru doda se jedan <span> element.

```
const host = document.querySelector("#host");
const shadow = host.attachShadow({ mode: "open" });
const span = document.createElement("span");
span.textContent = "I'm in the shadow DOM";
shadow.appendChild(span);
```

Slika 3.6. Primjer postavljanja shado DOM-a

Rezultat na web pregledniku nakon prikazanog koda iznad (Slika 3.5. i Slika 3.6.) prikazan je na slici ispod (Slika 3.7.)

Play

I'm in the shadow DOM
I'm not in the shadow DOM

Slika 3.7. Rezultat na pregledniku nakon dodavanja shadow DOM-a

Vidljivo je kako imamo dva span elementa, ali u HTML dokumentu je definiran samo jedan element s tekstom „I'm not in the shadow DOM“ i taj element je drugi po redu na prikazanom rezulatatu iz razloga što je ispred njega div element koji je iskorišten za postavljanje shadow DOM-a. U shadow DOM-u dinamički je dodan novi span element s tekstom „I'm in the shadow DOM“, te je on u rezultatu na prvom mjestu. Dakle, prvi span je dio shadow DOM-a, dok je drugi span element dio osnovnog DOM-a. Kada bi se dodavali nekakvi dodatni stilovi i oni bi se razlikovali, te ne bi utjecali jedan na drugoga. Također uz enkapsulaciju shadow DOM pruža kreiranje komponenti koje se mogu koristiti u raznim aplikacijama.

Shadow DOM se u kontekstu aplikacije ovog završnog rada koristio kao element u kojeg se učitava sučelje chatbota sa svim njegovim stilovima. Upravo shadow DOM je omogućio kreiranje skripte koja učitava chatbot na HTML stranicu po želji. Sav taj kod je pisan u React JS biblioteci koja podržava rad s DOM-om. React JS vrlo je popularna biblioteka iz razloga što omogućuje kreiranje prilagodljivih, ponovno iskoristivih komponenti frontend sučelja. React iz perspektive Shadow DOM-a nešto je drugačiji od osnovne teorije o samom Shadow DOM-u. Koristeći Shadow DOM unutar React [18] biblioteke programeri mogu kreirati izolirane, samostalne komponente koje su odvojene od ostatka aplikacije. Ova izolacija sprječava sukobe stilova i osigurava da se skripte izvršavaju u kontroliranom okruženju. Postoji nekoliko prednosti korištenja shadow DOM-a za enkapsulaciju i kreiranje izoliranih komponenti, a ovo su tri glavne:

- Ograničavanje stilova: Stilovi definirani unutar shadow DOM-a ne utječu na ništa izvan njega, čime se sprječavaju neočekivani problemi sa stiliziranjem.
- Izolacija DOM-a: Sadržaj shadow DOM-a nije pod utjecajem promjena u globalnom DOM-u, što smanjuje rizik od sukoba.
- Ponovna iskoristivost: Komponente s enkapsuliranim stilovima i ponašanjem lakše je dijeliti između različitih dijelova aplikacije ili čak različitih projekata.

U priloženoj slici ispod (Slika 3.8.) prikazan je primjer integracije shadow DOM-a s React komponentama. Koristi se ista metoda *attachShadow()* koja kreira *shadow root* i povezuje *shadow tree* na taj element, metoda u ovom slučaju prima objekt sa elementom *mode* koji je string s vrijednosti *open*, a to znači da se dozvoljava pristup shadow DOM-u iz osnovnog DOM-a putem JavaScripta, dok *close* znači da pristup nije dozvoljen.

```
class ShadowComponent extends React.Component {
componentDidMount() {
this.hostElement.attachShadow({ mode: 'open' });
}

render() {
return <div ref={el => this.hostElement = el}></div>;
}
}
```

Slika 3.8. Primjer intergacije shadow DOM-a u React-u

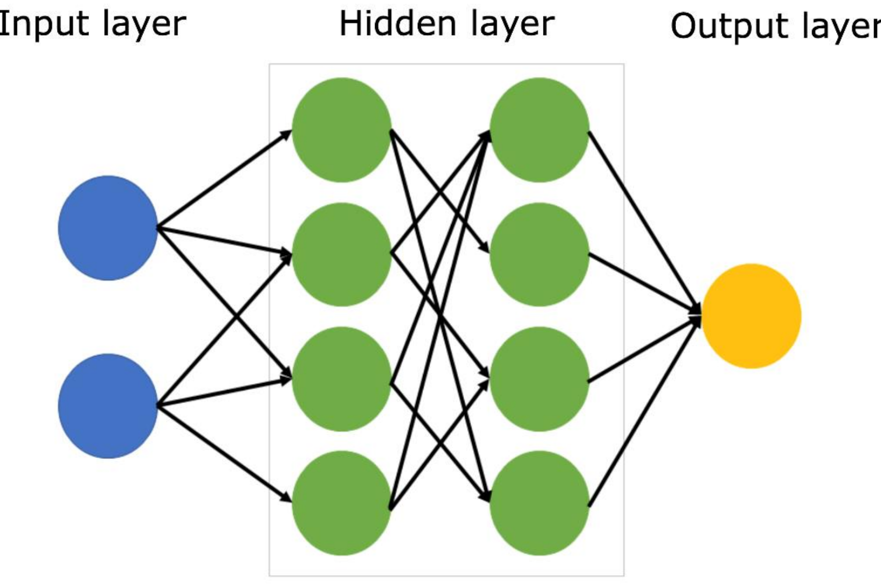
ShadowComponent klasa stvara shadow root na <div> elementu tijekom *componentDidMount* životnog ciklusa metode, a *ref* povratni poziv koristi se za vraćanje reference na taj <div> element, koji se pohranjuje u *this.hostElement*.

Iako je shadow DOM snažna i vrlo korisna web tehnologija, i ona dolazi s nekim izazovima i nedostacima. Dva glavna su upravljanje događajima (engl. event handling) i renderiranje na stani poslužitelja (engl. server-side rendering (SSR)). Problem s upravljanjem događajima iskazuje se u tome što događaji kao klik mišom ili pritisak određene tipke iz shadow DOM-a mogu imati drugačije ponašanje i u nekim slučajevima imati nepredvidljive karakteristike u usporedbi s događajima iz osnovnog DOM-a. Dok problem s SSR-om proizlazi iz toga što je shadow DOM primarno tehnologija preglednika, te nije prihvatljiva u Node.js okruženju što komplicira renderiranje sa strane poslužitelja. Rješenje se pronalazi u kreiranju komponenti koje se mogu degradirati i imati alternativne putanje za renderirane u slučaju SSR-a.

3.3. LLM-ovi i teorija o njima

LLM [19] je kratica od engleske fraze Large Language Model u prijevodu veliki jezični model, koja se odnosi na tehniku dubokog učenja kojom se izvode razni NLP (engl. natural language processing) zadaci. Koriste posebne modele koji se nazivaju transformeri koji su istrenirani na izuzetno velikim skupovima podataka. To im omogućava izvršavanje kompleksnih zadataka kao što je prepoznavanje, prevođenje, predviđanje ili generiranje nekog teksta ili nekih drugih oblika prirodnog jezika. Mnogi uspoređuju LLM-ove s ANN-ovima, iako imaju podosta sličnosti to nisu sinonimi. ANN [20] (engl. artificial neural network) ili umjetne neuronska mreža je zapravo niz ili serija algoritama koji rade skupa kako bi prepoznali teško vidljive uzorke unutar skupova podataka kroz procese koji naizgled nalikuju na način koji ljudski mozak funckionira. Naravno, to nije moguće u potpunosti postići jer je ljudski mozak kompleksan organ koji se još istražuje, ali po strukturi neuronske mreže su sistermi međusobno povezanih umjetnih neurona, umjetni neuroni su zapravo čvorovi koji primaju nekakve parametre i kroz kompleksne matematičke operacije kao rezultat daju neki broj koji se ovisno o kontekstu problema koji se rješava interpretira na razne načine, a najčesće kao vrsta vjerojatnosti. Neuronske mreže sastoje se od različitih slojeva neurona koji su međusobno povezani, ovisno o njihovoj povezanosti razlikujemo potpuno povezane od nepotpuno povezani mreža, potpuno povezana mreža je ona gdje se svaki neuron iz sloja povezuje sa svakim neuronom iz sloja ispred. Jedna od najpopularnijih i najčesćih umjetnih neuronskih mreža je unaprijedna višeslojna neuronska mreža, unaprijedna znači da su sve veze usmjerene u jednom pravcu, tj. da nema povratnih veza. Jedan takav primjer potpuno povezane unaprijedne višeslojne neurnoske mreže dan je na slici ispod (Slika 3.9.). Na slici su prikazana tri osnovna

dijela mreže: sloj za unos, skriveni sloj i sloj izlaza. Ukoliko mreža ima više skrivenih slojeva smatra se dubokom mrežom.



Slika 3.9. Umjetna neuronska mreža

LLM-ovi u usporedbi s ANN-ovima su zapravo puno veći mehanizmi koji koriste funkcionalnosti umjetnih neurnoskih mreža za rukovanje s prirodnim jezikom. Dakle neuronske mreže su neophodne za ispravan rad velikih jezičnih modela. LLM-ovi su karakteristični po tome što uz učenje ljudskih jezika za primjenu umjetne inteligencije, mogu se istrenirati za obavljanje različitih stvari poput pisanja koda ili razumijevanja bioloških pojmova poput fotosinteze. Kako bi to postigli moraju prethodno biti dobro istrenirani i podešeni za širok spektar funkcionalnosti koje pružaju. Primjena ovakvih modela je široka, a mogu se koristiti u područjima zdravstva, financija i zabave, a osnovni sustav na kojemu rade AI asistenti i chatbotovi je upravo LLM. Primjeri poznatijih LLM-ova su ChatGPT napravljen od strane OpenAI-a, BERT (Bidirectional Encoder Representations from Transformers) nastao od Google-a. A konkretni primjer LLM-a je zapravo GPT (engl. generative pre-trained transformer) koji se razvijao kroz vrijeme i trenutno postoje 4 verzije:

- GPT-1, koji je objavljen 2018. godine, sadrži 117 milijuna parametara i obuhvaća 985 milijuna riječi.
- GPT-2, koji je objavljen 2019. godine, sadrži 1,5 milijardi parametara.

- GPT-3, koji je objavljen 2020. godine, sadrži 175 milijardi parametara. Chat GPT također se temelji na ovom modelu.
- GPT-4 objavljen 2023. godine i bilijune parametara.

LLM-ovi [21] rade na principu dubokog učenja koristeći arhitekturu umjetnih neuronskih mreža za procesiranje i razumijevanje ljudskog jezika. Sastoje se od više slojeva uključujući slojeve unaprijednih mreža, slojeve ugradnje i slojeve pažnje. Pomoću mehanizma samopažnje (engl. self-attention) procijenjuju važnost različitih tokena u zapisu i prepoznaje međusobne ovisnosti.

3.3.1. Arhitektura LLM-ova

Važni elementi koji utječu na arhitekturu LLM-a su:

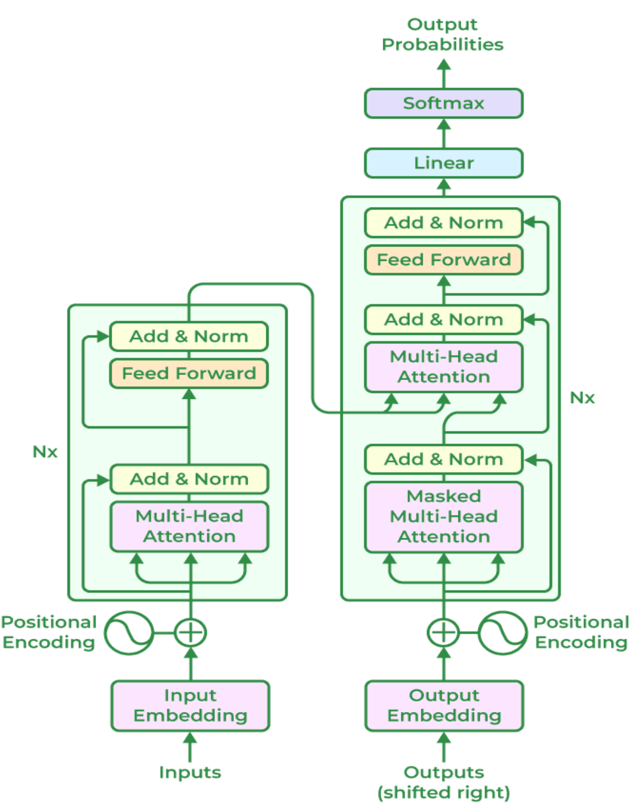
- Veličina modela i broj parametara
- Ulazne reprezentacije
- Mehanizmi samopažnje
- Ciljevi treniranja
- Računalna učinkovitost
- Dekodiranje i generiranje izlaza

LLM-ovi su zasnovani na modelima temeljenim na Transformeru koji su napravili iskorak u obradi prirodnog jezika te ga uvelike unaprijedili, arhitektura LLM-ova se bazira na transformer arhitekturi koja se sastoji od sljedećih komponenti:

1. Ulazne ugradnje (engl. input embeddings)Ulazni tekst se dijeli na manje jedinice, poput riječi ili podriječi, a svaki token se pretvara u vektor koji predstavlja njegovu značenje i sintaktičke informacije.
2. Pozicijsko kodiranje (engl. positional encoding): Dodaje se ulaznim ugradnjama kako bi model dobio informacije o redoslijedu tokena, jer Transformeri sami po sebi ne prepoznaju poredak.
3. Enkoder: Analizira ulazni tekst i stvara skrivena stanja koja čuvaju kontekst i značenje. Sastoji se od više slojeva, od kojih svaki sadrži mehanizam samopažnje i unaprijednu neuronsku mrežu.
4. Mehanizam samopažnje (engl. self-attention mechanism): Omogućuje modelu da procijeni važnost različitih tokena u nizu, uzimajući u obzir njihove međusobne odnose.

5. Unaprijedna neuronska mreža (engl. feed-forward neural network): Primjenjuje se na svaki token nakon samopažnje, koristeći potpuno povezane slojeve s nelinearnim funkcijama aktivacije za hvatanje složenih interakcija.
6. Dekoder slojevi (engl. decoder layers): Kod nekih modela dodaje se dekoder za autoregresivno generiranje, omogućujući modelu generiranje sekvencijalnih izlaza na temelju prethodno generiranih tokena.
7. Multi-Head attention: Omogućuje modelu da istovremeno prepozna različite odnose unutar ulazne sekvence.
8. Normalizacija slojeva (engl. layer normalization): Primjenjuje se nakon svakog sloja radi stabilizacije procesa učenja.
9. Izlazni slojevi (engl. output layers): Variraju ovisno o zadatku; na primjer, kod jezičnog modeliranja često se koristi linearna projekcija praćena SoftMax aktivacijom za generiranje distribucije vjerojatnosti sljedećeg tokena.

Primjer ovakve arhitekture dan je na slici ispod (Slika 3.10.)



Slika 3.10. LLM arhitektura [21]

4. REST API ZA CHATBOT

U ovom poglavlju biti će opisano REST API za chatbot, uključujući općenitu teoriju o REST arhitekturi kao i samom REST API-ju, nakon toga biti će prikazana integracija autorovog API-ja s OpenAI API-jem, te na kraju autentifikacija korisnika s poslužiteljske strane.

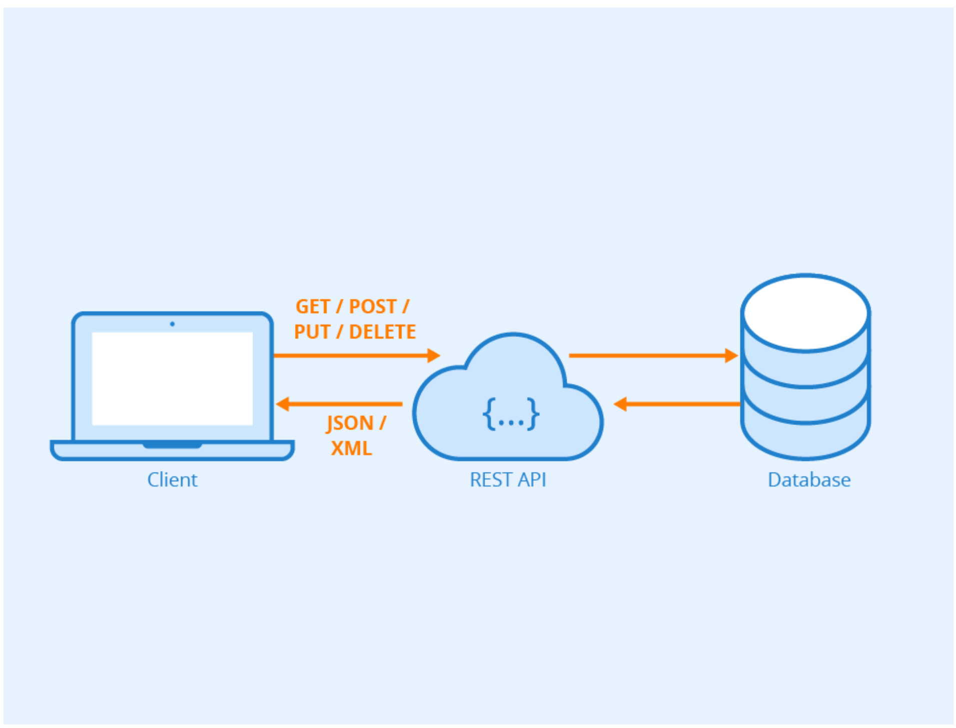
4.1. REST

REST [22] je akronim engleske fraze Representational State Transfer koji označava pojam arhitekturalnog stila za distribuciju resursa putem WEB-a. Resurs je ključna apstrakcija informacije u REST-u, to je bilo kakav tip podatka, dokument ili slika, vremenska usluga ili zbirka nekakvih drugih resursa. Reprezentacija resursa sasoji se od: podatka, metapodatka koji opisuje podatke i hipermedijske veze. Nastao je u početnog 21. stoljeća i od tada je najčesće korišten pristup za izgradnju web API-ja. REST je baziran na nekim ograničenjima i pravili koje pospješuju jednostavnost, skalabilnost u dizajnu. Šest osnovnih pravila REST-a su:

1. **Jedinstveno sučelje**: u razvoju aplikacija bitno je da postoji jednostavna i dosljedna interakcija između klijenta i poslužitelja, a to se postiže prateći nekoliko principa: identifikacija resursa (svaki resurs mora imati svoj identifikator), manipulacija resursa kroz reprezentacije (resursi imaju jedinstvenu i ujednačenu reprezentaciju koje klijenti koriste za promjenu stanja na serveru), samoopisane poruke i hipermedija kao pokretač stanja aplikacije što znaći da klijent treba imati samo početni URI (engl. uniform resource identifier), dok ostale resurse upravlja pomoću hiperveza u aplikaciji, primjer URI-ja: „https://requestb.in/1ix963n1".
2. **Klijent-Poslužitelj**: osigurava se odvajanje odgovornosti klijentskog korisničkog sučelja od pohrane podataka čime se poboljšava prenosivost i skalabilnost aplikacije
3. **Nepostojanje stanja (engl. statelessness)**: svaki zahtjev klijenta prema poslužitelju treba sadržavati sve potrebne informacije za izvršavanje zahtjeva, poslužitelj ne smije koristiti informacije iz prijašnjih zahtjeva.
4. **Keširanje (engl. cacheable)**: odgovor poslužitelja mora implicitno ili eksplicitno označiti je li keširan ili nije. Ako je keširan klijentska aplikacija može ponovo koristiti podatke odgovora za iste zahtjeve na određeno vrijeme.

5. **Slojeviti sustav**: ograničava ponašanje komponenti tako što uvodi hijerarhiju u sustav gdje svaka komponenta vidi samo sloj s kojim vrši interakciju. Primjer slojevitog sustava je MVC (engl. model – view – controller) obrazac koji jasno odvaja odgovornosti.
6. **Kod na zahtjev (opcionalno)**: omogućuje proširanje funkcionalnosti klijenta preuzimanjem i izvršavanjem koda u obliku appleta ili skripti

API koji implementira i slijedi ovih šest koncepata i pravila naziva se REST API. Za transport resursa kroz aplikacije koristi se proces serializacije i deserializacije, što ukratko znaći da se podatak koji se šalje mora serializirati, odnosno konvertirati u transportni objekt, najčešće JSON ili XML, i kada stigne na odrediše treba se deserializirati, odnosno konvertirati u nekakav specifičan objekt koji je potreban.



Slika 4.1. Ilustracija REST arhitekture

4.2. Metode resursa

Metode resursa [23] su jedna od važnijih stavki u kontekstu REST-a. Ove metode koriste se za izvršavanje željenog prijelaza između dva stanja nekog resursa. Metode resursa nisu ekvivalentne HTTP metodama (GET/PUT/POST/DELETE/...), HTTP metode su samo oznaka i smjernica HTTP-a da olakša interpretaciju pojedinog zahtjeva. Osnivači REST-a nisu ni u kojem trenutku točno precizirali koja metoda što radi, sve za što se oni zalažu je da sučelje bude dosljedno i

jedinstveno. Funkcionalnost metode POST je da stvara nove resurse u aplikaciji ali može biti da ažurira stanje nekog resursa ako se tako dogovore programeri koji razvijaju aplikaciju, dakle, HTTP metode služe kao univerzalni jezik kako bi se ljudi mogli lakše sporazumijevati, a ne kao striktno pravilo. Sve ide ka tome da REST aplikacija mora moći funkcionirati bez upotrebe prethodnog znanja, osim početnog URI-ja. Stvar je u tome što HTTP metode izuzetno dobro opisuju funkcije u RESTfull aplikaciji pa se u pravilu koriste kao standardni korak u razvoju REST API-ja, HTTP metode su metode koje specificira klijent kada šalje zahtjev prema poslužitelju, najpopularnije metode su:

1. GET: koristi se isključivo za dohvaćanje resursa, što znaći da nije moguća promjena stanja postojećeg pohranjenog resursa
2. POST: koristi se za kreiranje novog resursa i pohraniti ga u kolekciju resursa
3. PUT: koristi se za ažuriranje ili mijenjanje stanja postojećeg resursa
4. DELETE: koristi se za brisanje resursa, određenog jedinstvenim identifikatorom
5. PATCH: koristi se za djelomično ažuriranje resursa
6. HEAD: traži odgovor identičan GET zahtjevu ali bez tijela, tj. očekuje samo header odgovora

Također, u REST odgovorima mora postojati način na koji se opisuje posao kojega je poslužitelj odradio. Kijent kada šalje zahtjev, u njemu opisuje što želi napraviti putem HTTP metode, tako i poslužitelj kada vraća odgovor uz tijelo koje vraća, tijelo može biti u obliku nekog objekta ili opisnog teksta, mora dostaviti statusni kod [23] koji opisuje odrađeni posao. Najčešće koristeni statusni kodovi su:

- 200 OK: zahtjev je uspješno obrađen i tijelo sadrži ispravne resurse
- 404 Not Found: poslužitelj nije u mogučnosti pronaći resurs, najvjerojatnije iz razloga što je klijent krivo utipkao URI ili zatražio krivi resurs
- 400 Bad Request: poslužitelj nije u mogučnosti procesirati zahtjev zbog greške u zahtjevu klijenta
- 500 Internal Server Error: greška na poslužitelju
- 302 Found (Moved Temporarily): traženi resurs je privremeno premješten na drugi URL
- 401 Unauthorized: zahtjev zahtjeva korisničku autentifikaciju, klijent se nije prijavio u svoj račun

4.3. Integracija chatbot API-ja s OpenAI

U ovom pododlomku biti će opisan proces integracije autorova API-ja s OpenAI API-jem. Kako bi se to uopće omogućilo mora postojati klasa koja nosi odgovornost rukovanja sa stranim API-jem, u ovom specifičnom slučaju klasa je imenovana OpenAIHelper, jednostavna je i sadrži jedno privatno svojstvo, konstruktor i dvije važne metode koje se često koriste tijekom izvođenja aplikacije.

```
4 references
public class OpenAIHelper
{
    private readonly string apiKey;

    0 references
    public OpenAIHelper()
    {
        apiKey = "sk-RO*********************************************";
    }
```

Slika 4.2. Klasa OpenAIHelper

Privatno svojstvo apiKey inicijalizirano kroz konstruktor važna je stavka za interakciju s javim OpenAI API-jem, bez ispravne vrijednosti API ključa API zahtjevi nisu pravilo autorizirani i OpenAI poslužitelj vraća 401 Unauthorized odgovor. API ključ korisnik može pronaći na OpenAI stranici nakon registracije i prijave. Nadalje ostaju još dvije metode koje imaju svaka svoje specifične odgovornosti, prva od njih je *ProcessMessagesToOpenAI* koja prima listu objekata *Message* klase, ti objekti kao što ime kaže reprezentiraju poruke u razgovoru, bile one poslane od strane korisnika ili od strane chatbot asistenta, i također prima nullable int parametar *threadId* koji označava identifikator trenutnog threada, u ovom kontekstu razgovora, ako je korisnik autentificiran. U slučaju da korisnik nije autentificiran taj parametar nosi vrijednost null. Povratni tip metode je Message omotan u Task iz razloga što je API poziv asinkrona operacija, a Message je zbog toga što se u tijekom aplikacije na poziv ove metode očekuje odgovor chatbota i upravo taj Message objekt reprezentira odgovor. Nadalje, lista poruka koja je primljena kroz parametar metode prvo se konvertira u listu poruka oblika kakav odgovora OpenAI API-ju, a to je poruka s dva opisna atrubuta: *role* i *content*. *Role* atribut ozbnačava tko je poslao poruku, korisnik ili asistent, a *content* je sadržaj poruke. Zatim se generira početna poruka namjenjena sistemu i zato nosi vrijednost role atributa system. Sistem porukom se virtualnom asistentu ukratko objasni kako bi se trebao ponašati, tj kakav ton bi trebao imati, u ovom slučaju mu je rečeno da razgovara kao dvadesetogodišnji student. Nakon predobrade poruka i pripreme za API zahtjev slijedi konfiguracija HTTP zahtjeva u obliku kreiranja tijela u koje se ubacuje GPT model, poruke i

autorizacijski API ključ. Zatim se tijelo zahtjeva serializira, enkodira UTF8 standardom i šalje na specifični URI. Ako je zahtjev valjan dobiva se odgovor u obliku JSON objekta koji se treba deserializirati, OpenAI API u odgovoru vraća jednu poruku spremljenu u *data.choices[0].message.content* varijablu. U završnom koraku se kreira novi Message objekt te se šalje kao povratni podatak ove metode.

```
public async Task<Message> ProcessMessagesToOpenAI(List<Message> messages, int? threadId)
{
    try
    {
        var apiMessages = messages.Select(message =>
        {
            return new
            {
                role = message.Role == "user" ? "user" : "assistant",
                content = message.Content
            };
        }).ToList();

        var systemMessage = new
        {
            role = "system",
            content = "Speak like a 20 year old student"
        };

        var apiRequestBody = new
        {
            model = "gpt-3.5-turbo",
            messages = new[] { systemMessage }.Concat(apiMessages)
        };

        using (var client = new HttpClient())
        {
            client.DefaultRequestHeaders.Add("Authorization", $"Bearer {apiKey}");

            var jsonRequestBody = JsonConvert.SerializeObject(apiRequestBody);
            var stringContent = new StringContent(jsonRequestBody, Encoding.UTF8, "application/json");

            var response = await client.PostAsync("https://api.openai.com/v1/chat/completions", stringContent);

            if (!response.IsSuccessStatusCode)
            {
                throw new Exception($"Failed to get response from OpenAI API. Status code: {response.StatusCode}");
            }

            var responseData = await response.Content.ReadAsStringAsync();
            dynamic data = JsonConvert.DeserializeObject(responseData);

            Message message = new Message();
            if (threadId != null)
            {
                message = new Message
                {
                    Role = "assistent",
                    Content = data.choices[0].message.content,
                    ThreadId = threadId
                };
            }
            else
            {
                message = new Message
                {
                    Role = "assistent",
                    Content = data.choices[0].message.content,
                };
            }


            return message;
        }
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

Slika 4.3. ProcessMessagesToOpenAI metoda

Druga metoda je slična prvoj, ali s jednom razlikom. Prva metoda služi za komunikaciju sa chatbotom, dok druga metoda ima zadaću dati ime trenutnom razgovoru po uzoru na prvu poruku koju je korisnik poslao. Dakle, metoda *GetThreadName* prima jednan Message objekt, te se generira jedna poruka s tekstom "Please give me a short name for this message: '{message.Content}' (only name without quotation marks)", *message* objekt proizlazi iz liste parametara. Nakon toga svi koraci su jednaki kao u prvoj *ProcessMessagesToOpenAI* metodi, generiranje tijela zahtjeva, postavljanje API ključa, serializacija tijela zahtjeva i slanje na isti URI. Povratni tip metode je string omotan u Task iz razloga što je API poziv asinkron, a string predstavlja ime trenutnog threada. Napomena, ova metoda se koristi samo kod korisnika koji su prethodno registrirani i prijavljeni na chatbot domenu.

```
public async Task<string> GetThreadName(Message message)
{
    try
    {
        var apiMessage = new
        {
            role = message.Role,
            content = $"Please give me a short name for this message: '{message.Content}' (only name without quotation marks)"
        };

        var apiRequestBody = new
        {
            model = "gpt-3.5-turbo",
            messages = new[] { apiMessage }
        };

        using (var client = new HttpClient())
        {
            client.DefaultRequestHeaders.Add("Authorization", $"Bearer {apiKey}");

            var jsonRequestBody = JsonConvert.SerializeObject(apiRequestBody);
            var stringContent = new StringContent(jsonRequestBody, Encoding.UTF8, "application/json");

            var response = await client.PostAsync("https://api.openai.com/v1/chat/completions", stringContent);

            if (!response.IsSuccessStatusCode)
            {
                throw new Exception($"Failed to get response from OpenAI API. Status code: {response.StatusCode}");
            }

            var responseData = await response.Content.ReadAsStringAsync();
            dynamic data = JsonConvert.DeserializeObject(responseData);

            string name = data.choices[0].message.content;


            return name;
        }
    }
    catch (Exception e)
    {
        throw;
    }
}
```

Slika 4.3. GetThreadName metoda

Pozivi ovih metoda vidljivi su iz endpointova aplikacije. Endpoint je krajnja točka API na koji se vrši API poziv. Endpoint koji poziva ove metode je POST endpoint ProcessMessageseToOpenAI, koji će detaljno biti opisan u 5. poglavlju.

4.4. Autentifikacija i autorizacija

Autentifikacija i autorizacija korisnika izvršena je uporabom predefiniranog sustava za autentifikaciju od strane .NET-a koji se zove ASP.NET Core Identity [24]. To je mini API koji podržava registraciju i login korisnika, Identity se brine o spremanju korisnika, zaporki, role-ova, emaila i svih metapodataka koji se tiču korisnika. Također pruža u korisničko sučelje za registraciju i prijavu u obliku Razor stranica. Identity je konfiguriran uz SQL Server bazu podataka gdje su spremljenji svi korisnici, poruke i razgovori. Za interakciju s bazom podataka korišten je ORM (engl. object-relational mapper) alat kojeg pruža .NET zvan *EntityFramework*. To znatno olakšava komunikaciju s bazom jer prilikom koritenja Entity Frameworka nije potrebno pisanje SQL skripti već se bazi pristupa objektno orijentiranim jezikom. Identity sprema korisnike u obliku objekata u bazu, objekt je definiran klasom *ApplicationUser* koja nasljeđuje IdentityUser, *IdentityUser* je predodređena klasa sa predodređenim atributima za korisnika, u ovom slučaju bilo je potrebe za dodatnim svojstvom koje User ima, a to je kolekcija threadova.

```
99+ references
public class ApplicationUser : IdentityUser
{
    2 references
    public string FirstName { get; set; }
    1 reference
    public string LastName { get; set; }
    1 reference
    public ICollection<Models.Thread> Threads { get; set; }
}
```

Slika 4.4. ApplicationUser

Identity se također brine i pohranjivanju lozinke i njenom hashiranju, lozinke korisnika nikada se ne pohranjuju u običnom tekstu iz sigurnosnih razloga. Koristi se kriptografski algoritam PBKDF2 (engl. password-based key derivation function 2) za hashiranje lozinke. To je algoritam koji primjenjuje više SHA-256 hash funkcija kako bi se dobio konačni hash koji se pohranjuje u bazu podataka. Kada se korisnik želi ulogirati, upisuje svoju lozinku koja također hashira i uspoređuje s hashiranom lozinkom u bazi. Proces hashiranja se vrši iz razloga što je hash funcija karakteristična po tome što male promjene na ulazu prouzrokuju iznimno velike promjene na izlazu, što znaći da je skoro pa nemoguće iz hashiranog izlaza dobiti ulaz. Dakle iz hashiranog zapisa se nikako ne može dobiti početna lozinka, jedino se mogu uspoređivati dva hashirana zapisa i biti će jednaki ako i samo ako su unese lozinke jednake. Radi dodatne sigurnosti Identity tijekom registriranja korisnika, prije hashiranja dodaje nasumični niz znakova kako bi otežao pokušaje

napada. Prilikom provjere lozinke Identity dodaje isti taj niz znakova kako bi se proizveo isti hash. Proces dodavanja nasumičnih znakova prije hashiranja naziva se salting. Autorizacija korisnika vrši se preko kolačića (engl. cookie-based authentication) [25] koji se u literaturi naziva authentication cookie. Kada se korisnik uspješno prijavi u svoj račun na web aplikaciji, poslužitelj u tom trenutku generira kriptografski zapis koji sprema na klijentskoj strani, na pregledniku kao autentifikacijski kolačić. Time se kreira sesija između klijenta i poslužitelja, sesija je sekvenca interakcija između klijenta i poslužitelja, tj. određeni broj zahtjeva koji klijenti mogu poslati poslužitelju i očekivati odgovor od njega. Kolačići su izuzetno korisni kod principa autorizacije jer HTTP protokol sam po sebi, a i po načelima REST-a mora biti stateless, što znači da je svaki zahtjev zaseban i sve informacije koje poslužitelju trebaju, moraju se nalaziti unutar samog zahtjeva. Iz tog razloga je uveden pojam autentifikacije preko kolačića. Pristup je vrlo jednostavan i učinkovit samo što dolazi s nekim ograničenjima poput činjenice da su kolačići vezani za domenu, što znaći da preglednik može slati kolačiće samo na poslužitelja koji je kreirao taj kolačić. To osigurava značajan nivo sigurnosti i kod rada sa MVC aplikacijama ne predstavlja problem jer se kod MVC aplikacija i klijentska i poslužiteljska strana nalazi na istoj domeni pa se autentifikacija preko kolačića čini kao opcija bez mana. No u slučaju izgradnje aplikacije koja u sebi integrira chatbot koji dolazi s druge domene predstavlja veliki problem. Jedno od rješenja ovog problema autor rada pronašao je u postavljanju web aplikacije tako da se *main* skripta koja sadrži izkompajlirani JavaScript kod koji stvara shadow DOM i u njega postavlja kompletan chatbot učita iz iste domene. Dakle, po završetku klijentskog dijela aplikacije datoteka koja sadrži iskompajlirani JS kod kopirana je i zalijepljena u folder na putanji ~/wwwroot/js/chatbot-frontend zajedno sa svim kreiranim komponentama i css datotekama koje su se koristile u izradi chatbot sučelja. Na taj način kolačići mogu nesmetano kružiti imeđu klijenta i poslužitelja zajedno sa svim potrebnim podacima. Bolji način bio bi vršenje autorizacije preko JWT-a (engl. JSON web token), što je sličan pristup kao cookie ali omogućava dijeljenje sadržaja između domena, jer za razliku od kolačića ne sprema podatak u pregledniku gdje je javno vidljiv, već je skriven negdje unutar klijentske aplikacije. Korištenje JWT-a za autorizaciju može se smatrati kao buduće unaprijeđenje za ovu aplikaciju. Tada ne bi bilo potrebno kompajlirani JS zalijepiti na istu domenu s koje dolazi poslužitelj, tj. omogućila bi se međudomenska komunikacija. Preciznije omogućilo bi se dijeljenje podataka od strane poslužitelja prema klijentu.

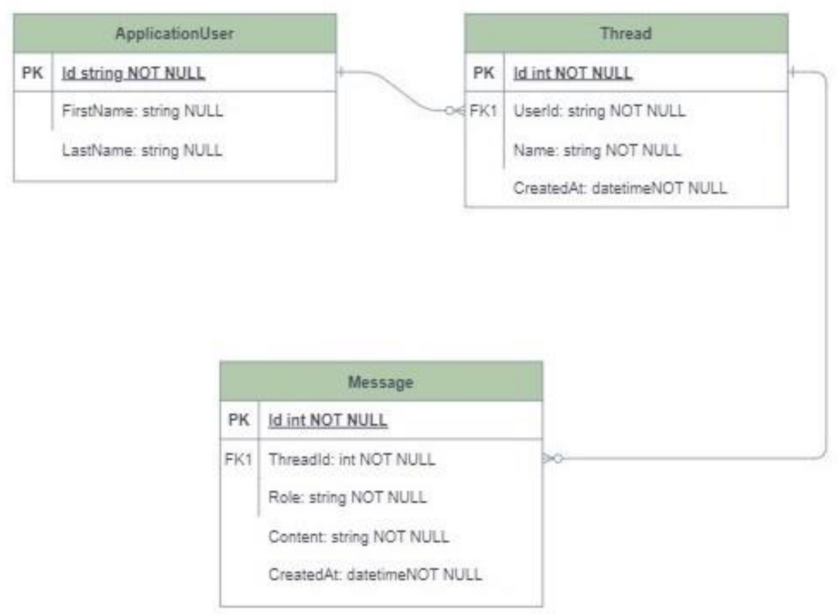
5. INTEGRACIJA CHATBOTA U WEB APLIKACIJU POMOĆU REST API-JA

Web aplikacija u koju je integriran chatbot izrađena je kao MVC aplikacija unutar ASP.NET razvojnog okvira. MVC [26] (engl. model – view – controller) je arhitekturalni obrazac koji razdvaja aplikaciju u tri bitne komponente, čime razdvaja odgovornosti u aplikaciji te postiže visoki nivo strukturiranosti. Korisnički zahtjevi su usmjereni prema kontroleru koji radi nekakve operacije nad podacima koji su predstavljeni modelom, modeli su ništa drugo nego klase koje definiraju strukturu i ponašanje podataka u aplikaciji, nakon što kontroler odradi akciju, rezultate prikazuje korisniku tako što prikazuje odgovarajući view.

5.1. Poslužiteljski dio aplikacije

5.1.1. Baza podataka i modeli

U ovom radu se koristi ASP.NET razvojni okvir za izradu MVC aplikacija i poslužiteljskog API-ja, za bazu podataka koristi se lokalna SQL Server relacijska baza za pohranu podataka o modelima. U nastavku na Slici 5.1. biti će prikazan relacijski dijagram modela.



Slika 5.1. Relacijski model baze podataka

Dakle u aplikaciji postoji tri tipa modela: ApplicationUser, Thread i Message:

- ApplicationUser predstavlja korisnika aplikacije koji ima svoje osnovne atrubute poput imena i prezimena, također uz njega se vežu pripadajući thread objekti koje je kreirao
- Thread predstavlja jedanu razgovornu kutiju (engl. chatbox) koja u sebi ima pohranjene sve poruke iz tog razgovora, ApplicationUser je vezan za Thread vezom 1 naprema više
- Message predstavlja poruke unutar razgovora, Thread je vezan za Message vezom 1 naprema više

5.1.2. Chatbot kontroler

Kontroleri u kontekstu ASP.NET web API-ja i MVC aplikacija su klase koje obrađuju i procesuiraju HTTP zahtjev. Dakle, kada korisnik pošalje zahtjev na nekakav API, kontroler zadužen za tu nekakvu specifičnu granu se aktivira i obrađuje zahtjev u obliku odrađivanja nekog posla, ili najčesće proslijedi zadatak određenom servisu koji vraća rezultat kontroleru, koji taj isti rezultat vraća klijentu u obliku HTTP odgovora. Korištena je MVC aplikacija za izradu web aplikacijskog poslužitelja u ovom radu, MVC dolazi sa mogučnošću korištenja web API-ja, pa u nastavku kada se spominje poslužitelj, MVC aplikacija ili chatbot API misli se na istu stvar. Kao što je rečeno API dolazi integriran s MVC aplikacijom, da bi to bilo moguće mora se kreirati posebna klasa kontrolera. U ovom slučaju ista klasa nazvana je *ChatbotController* i ona služi kao API kontroler s rutom „api/message/" za funcionalnosti koje pruža chatbot API, u ovoj aplikaciji nije korištena multilayer arhitektura i kontroler obavlja sav posao koji se inače prosljeđuje servisu. Kontroler sadrži nekoliko krajnjih točaka (engl. endpoint), od kojih je prva *GetUser*, kao što ime kaže to je Get endpoint koji vraća podatke o autoriziranom korisniku. Pomoću SignInManagera provjerava se je li korisnik uopće autoriziran, ukoliko je pomoću *User* globalne varijable unutar ASP.NET okvira koja predstavlja autoriziranog korisnika s njegovim *Claim* tipovima, to su podaci pohranjeni u hashirani kolačić poput userId-a ili emaila ili nekog drugog važnog podatka za korisnika. Trenutno bitan podatak je samo userId, *userManager* koristi userId kako bi pronašao konkretnog korisnika u bazi. Nakon što se pronađe korisnik, traže se svi njegovi *thread*-ovi u posljenih 7 dana te se skupni objekt vraća u obliku ActionResult<ApplicationUser> povratnog tipa omotanog u Task jer je metoda asinkrona. Na Slici 5.2. u nastavku biti će prikazan endpoint GetUser.

```
[HttpGet("current-user")]
0 references
public async Task<ActionResult<ApplicationUser>> GetUser()
{

    try
    {
        if (userManager == null)
        {
            return StatusCode(StatusCodes.Status500InternalServerError, "UserManager is not initialized");
        }
        if (!signInManager.IsSignedIn(User))
        {
            return StatusCode(StatusCodes.Status500InternalServerError, "User not signed in");
        }
        var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
        if (userId == null)
        {
            return StatusCode(StatusCodes.Status500InternalServerError, "UserId not found");
        }
        var currentUser = await userManager.FindByIdAsync(userId); ;
        if(currentUser == null)
        {
            return StatusCode(StatusCodes.Status500InternalServerError, "User not found");
        }
        var startDate = DateTime.Today.AddDays(-7);
        var user = currentUser.ToUserResponse();
        user.Threads = new List<ThreadResponse>();
        var threads = db.Threads.Where(t => t.UserId == userId).Select(t => t.ToThreadResponse()).ToList();
        threads = threads.OrderByDescending(t => t.CreatedAt).ToList();
        if(threads.Count > 0)
        {
            user.Threads = threads;
            foreach (var thread in user.Threads)
            {
                var messages = db.Messages.Where(m => m.ThreadId == thread.Id).Select(m => m.ToMessageResponse()).ToList();
                thread.Messages = messages;
            }
        }

        return Ok(user);

    }
    catch(Exception e)
    {
        return StatusCode(500, $"Internal server error: {e.Message}");
    }
}
```

Slika 5.2. GetUser endpoint

Također je korišten MessageMapper za konverziju Message objekta u DTO (engl. data transfer object) MessageResponse koji sadrži samo neka svojstva osnovnog Message modela. Svrha ovoga leži u tome što klijentu ne trebaju svi podaci iz modela, pa je dobra praksa raditi DTO-ove za korištene modele. U nastavku na Slici 5.3. slijedi implementacije mappera za Message.

```
3 references
public static MessageResponse ToMessageResponse(this Message message)
{
    return new MessageResponse
    {
        Role = message.Role,
        Content = message.Content,
        ThreadId = message.ThreadId,
    };
}
```

Slika 5.3. Implementacija Message mappera

Nadalje slijedi POST enpoint pod imenom *CreateThread*, on služi za kreiranje praznog razgovora od strane autoriziranog korisnika. Provjerava se trenutno autoriziran korisnik i pronalazi se njegov userId iz User Claims-a kao i u prijašnjoj krajnjoj točci. Zatim se kreira novi Thread objekt koji se šalje kao odgovor. Na slici 5.4. prikazana je implementacija CreateThread metode.

```
[HttpPost("create-thread")]
0 references
public async Task<IActionResult> CreateThread()
{
    var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if(userId == null)
    {
        return StatusCode(StatusCodes.Status500InternalServerError, "User not signed in");
    }

    Thread newThread = new Thread
    {
        Name = "New chat",
        UserId = userId,

    };

    db.Threads.Add(newThread);
    await db.SaveChangesAsync();
    var threadDTO = newThread.ToThreadResponse();
    threadDTO.Messages = new List<MessageResponse>();
    return Ok(threadDTO);
}
```

Slika 5.4. ChreateThread endpoint

Također je korišten mapper koji konvertira Thread objekt u ThreadResponse i prikazan je u nastavku na slici 5.5.

```
2 references
public static ThreadResponse ToThreadResponse(this Models.Thread thread)
{
    return new ThreadResponse
    {
        Id = thread.Id,
        Name = thread.Name,
        CreatedAt = thread.CreatedAt,

    };
}
```

Slika 5.5. Implementacija Thread mappera

Treći i zadnji endpoint ove web aplikacije je *ProcessMessagesToOpenAI* koja prima listu Message objekata i int? threadId , najvažniji i najkompleksniji POST enpoint koji služi za komunikaciju sa chatbotom, u ovom služaju služi za prosljeđivanje zahtjeva na već prethodno opisanu klasu

OpenAIHelper, ali to vrši s određenim redoslijedom i određenom logikom. Dakle na početnu moralo se uzeti u obzir kako komunikaciju s chatbotom mogu ostvariti i autorizirani i neuatorizirani korisnici, pa se po tom prvo treba služiti uvjetnim grananjem. Slučaj kada korisnik nije autoriziran je jednostavniji, nema potrebe voditi brigu o threadu u kojemu se trenutno razgovara ili o kreiranju threada, pa se samo proslijede parametri na *openAIHelper.ProcessMessagesToOpenAI(messages, threadId)*, nakon toga se lista poruka, zajedno s novom porukom prebacuju u MessageResponse i šalju klijentu kao odgovor. Slučaj kada je korisnik autoriziran je malo kompleksniji, na početku se mora pronaći korinsik i thread u kojemu se razgovara, nakon toga se dohvaćaju sve poruke iz baze koje su vezane za taj specifični thread, ukoliko nema poruka i thread je prazan potrebno je stvoriti ime threada pomoću OpenAIHelper klasa i njene metode GetThreadName, nakon toga se nova poruka sprema u bazu i zajedno sa ostalim porukama (ako ih ima) šalje OpenAIHelper klasu na procesiranje poruke i traženje chatbot odgovora. Kad chatbot odgovori nova poruka se sprema u bazu, sve poruke se prebacuju u MessageResponse putem MessageMappera nakon čega se šalju klijentu kao odgovor. Na Slici 5.6. prikazan je enpoint ProcessMessagesToOpenAI.

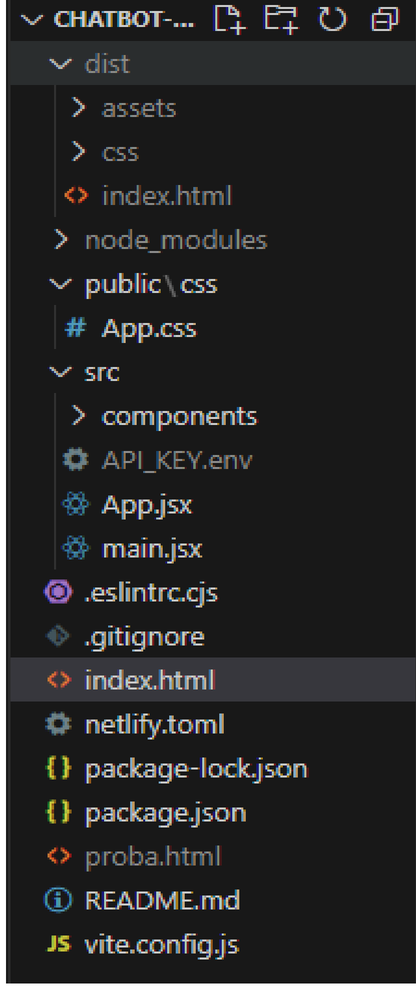
```
[HttpPost("process-messages/{threadId?}")]
0 references
public async Task<ActionResult<List<Message>>> ProcessMessagesToOpenAI([FromBody] List<Message> messages, [FromRoute] int? threadId)
{
    try
    {
        if (!signInManager.IsSignedIn(User))
        {

            if(threadId < 0)
            {
                threadId = null;
            }

            Message message = await
                openAIHelper.ProcessMessagesToOpenAI(messages, threadId);

            messages.Add(message);

            var messagesDTO = messages.Select(s => s.ToMessageResponse()).ToList();
            return Ok(messagesDTO);
        }
        else
        {
            var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
            var thread = db.Threads.Find(threadId);

            if (thread == null)
            {
                return NotFound();
            }

            if (!(userId == thread.UserId))
            {
                return BadRequest();
            }

            var messagesToProcess = db.Messages.Where(m => m.ThreadId == thread.Id).ToList();
            if (!messagesToProcess.Any())
            {
                thread.Name = (await openAIHelper.GetThreadName(messages.Last())).ToString();
                if (thread.Name == null)
                {
                    throw new Exception("Failed to get thread name from OpenAI API");
                }
                db.Threads.Update(thread);
                await db.SaveChangesAsync();
            }
            messagesToProcess.Add(messages.Last());
            db.Messages.Add(messages.Last());
            await db.SaveChangesAsync();

            Message message = await
               openAIHelper.ProcessMessagesToOpenAI(messagesToProcess, threadId);
            db.Messages.Add(message);

            await db.SaveChangesAsync();
            messages.Add(message);

            var messagesDTO = messages.Select(s => s.ToMessageResponse()).ToList();
            return Ok(messagesDTO);
        }


    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Internal server error: {ex.Message}");
    }
}
```

Slika 5.6. ProcessMessagesToOpenAI endpoint

5.2. Klijentski dio aplikacije

5.2.1. Arhitektura

Klijentsko sučelje pisano je u JavaScript jeziku koristeći React JS biblioteku upotrebom ponovno iskoristivih komponenti i Shadow DOM tehnologije. React aplikacija koristeći alat pod imeno Vite, to je moderni, brzi razvojni okvir za izradu JavaScript aplikacija. Kao i za većinu JavaScript aplikacija korištena je Node.js open-source platforma koja omogućava razvoj frontend aplikacija i korištenje JavaScript koda izvan web preglednika. Na Slici 5.7. u nastavku vidljiva je struktura direktorija klijentskog dijela aplikacije koja prati smjernice postavljenje od strane Vite zajednice.



Slika 5.7. Struktura direktorija klijentske aplikacije

Unutar direktorija .../src/components/ nalaze se komponente Message.jsx i ChatbotWindow.jsx, Message.jsx reprezentira poruku unutar razgovora, dok ChatbotWindow.jsx reprezentira čitavo chatbot sučelje. Prilikom izrade chatbot sučelja korišten je lokalni server „http://localhost:5173“ ,

na njemu se nalazi HTML dokument s jednim div elementom i učitanom main.jsx skriptom koja kreira shadow DOM i učitava App.jsx skriptu u taj shadow DOM. Način na koji se učitava shadow DOM je posve dinamički, takav pristup je nužan zbog glavne mogućnosti i potrebe chatbot sučelja, a to je da se osposobi ubacivanje sučelja u bilo koji HTML dokument, dinamičko stvaranje elementa i postavljanja shadow DOM-a je nužan korak u toj izvedbi. U nastavku na Slici 5.8. slijedi prikaz implementacije dinamičkog postavljanja shadow DOM-a, tj. implementacija main.jsx komponente.

```
import React, { version } from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';
import 'bootstrap-icons/font/bootstrap-icons.css';

const root = document.createElement('div');
root.id = 'root';
document.body.append(root);

document.addEventListener('DOMContentLoaded', () => {

const shadowRoot = root.attachShadow({ mode: 'open' });
const link = document.createElement('link');
link.rel = 'stylesheet';
link.href = '../../css/App.css';


shadowRoot.appendChild(link);
const reactApp = document.createElement('div');
reactApp.id = 'react-app';
shadowRoot.appendChild(reactApp);

ReactDOM.createRoot(reactApp).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
});
```

Slika 5.8. main.jsx komponenta

Dakle, kreira se div element imena root, na koji se dodaje slušatelj događaja (engl. event listener) zvan *DOMContentLoaded*, to je poseban event listener koji se aktivira nakon što se osnovni DOM inicijalizira u pregledniku, nakon što se on aktivira inicijalizira se shadow root na host root. Unutar shadow root-a prvo se postavi link element na css stil, zatim se dodaje div element naziva react-app u kojeg se učitava App.jsx. Unutar App.jsx nalazi se gotovo sva logika za rukovanje sa

sučeljem chabota, također App.jsx inicijalizira *ChatbotWindow* komponentu na pritisak gumba *open chatbot* na stranici. Za uspostavu s chatbot API-jem koristila se JavaScript funckija fetch, te se HTTP zahtjev podešavaju po potrebi. Primjer korištenja funckije fetch biti će objašnjen prikazom funkcije *processMessagesToChatGPT* koja se spaja na endpoint ProcessMessagesToOpenAI koji je objašnjen u prethodnom poglavlju. Odgovor API-ja sprema se u constantu data iz koje se poruke mappiraju u željeni objekt i na kraju se postavljaju u stanje ChatMessages koristeći setter setChatMessages() funckiju nakon koje se ponvono renderira stranica i prikazuje poruke zajedno s novom porukom od strane chatbota. Na Slici 5.9. vidi se funcija *processMessagesToChatGPT*.

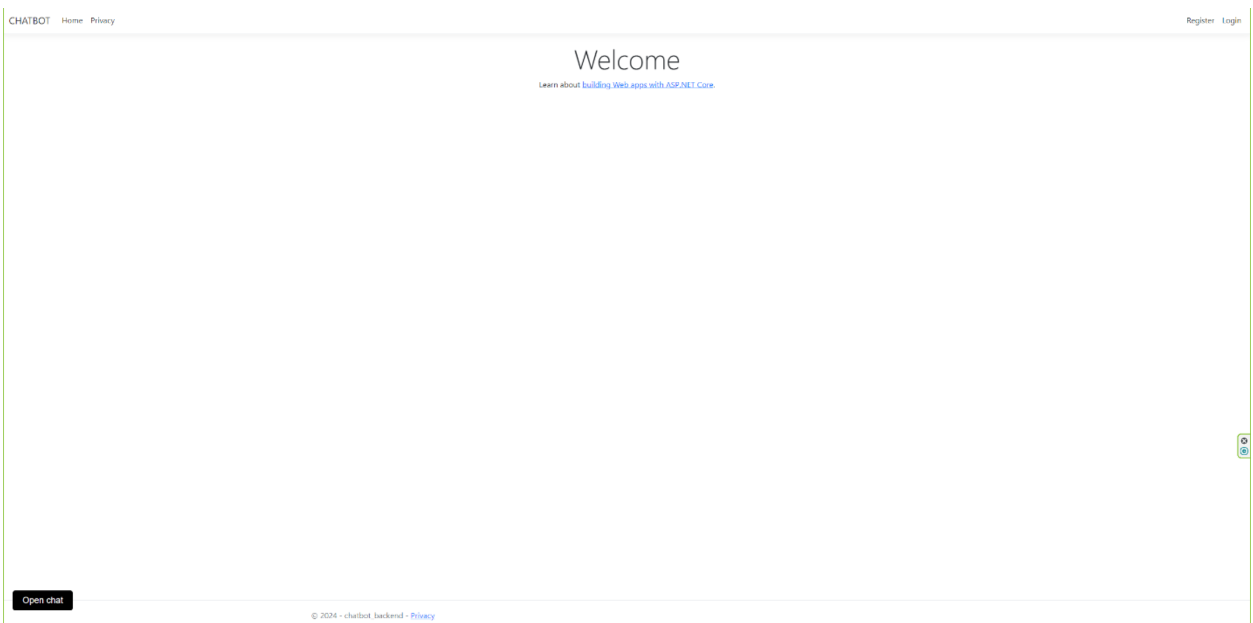
```
async function processMessagesToChatGPT(newMessages){
  try {

    const response = await fetch(`https://localhost:7030/api/message/process-messages/$
    {currentThread? currentThread.id: -1}`, {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify(newMessages)
    });

    if (!response.ok) {
        throw new Error('Failed to process messages');
    }

    const data = await response.json();

    const extractedMessages = data.map(message => ({
      role: message.role,
      content: message.content,
      threadId: message.threadId? message.threadId: null

    }));
    setChatMessages(extractedMessages);

  } catch (error) {
    console.error('Error processing messages:', error.message);
  }
}
```

Slika 5.9. Primjer korištenja funkcije *fetch*

5.2.2. Sučelje

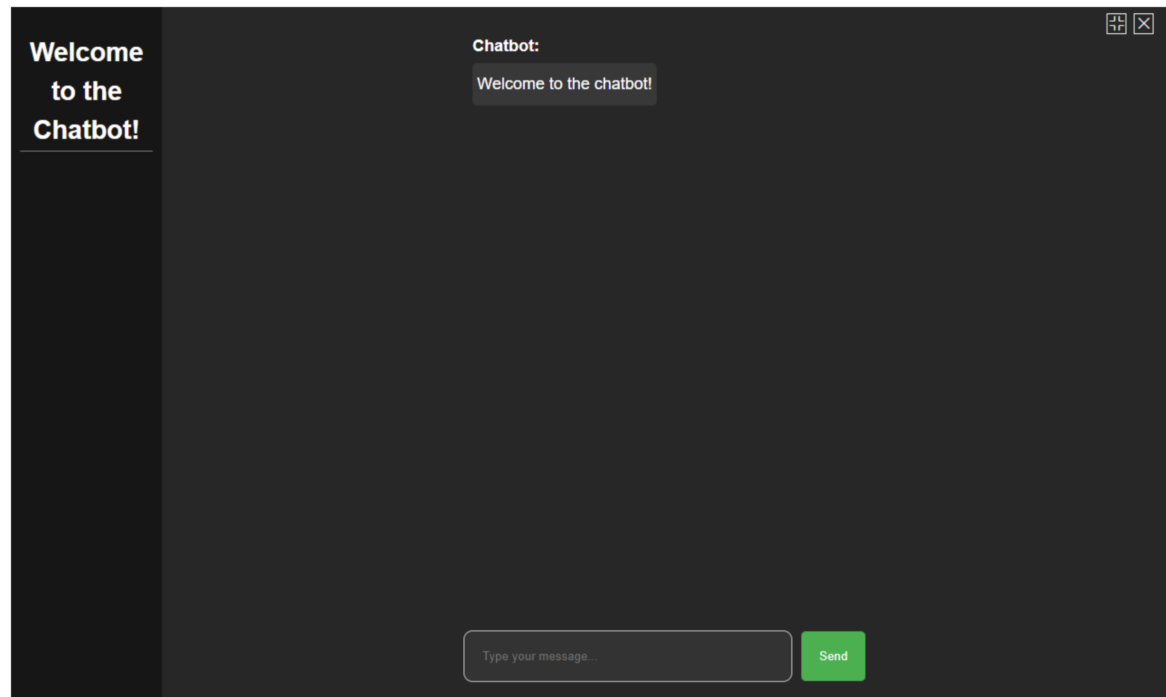
Konačna integracija chatbot sučelja unutar REST web aplikacije prikazana je u ovom poglavlju gdje će ukratko biti objašnjen izgled sučelja i korištenje aplikacije. Ulaskom na početnu stranicu

aplikacije prikazan je predodređeni izgled ASP.NET MVC aplikacije s gumbom u doljnjem lijevom kutu koji otvara i zatvara chatbot sučelje. Na Slici 5.10. prikazana je početna stranica aplikacije.



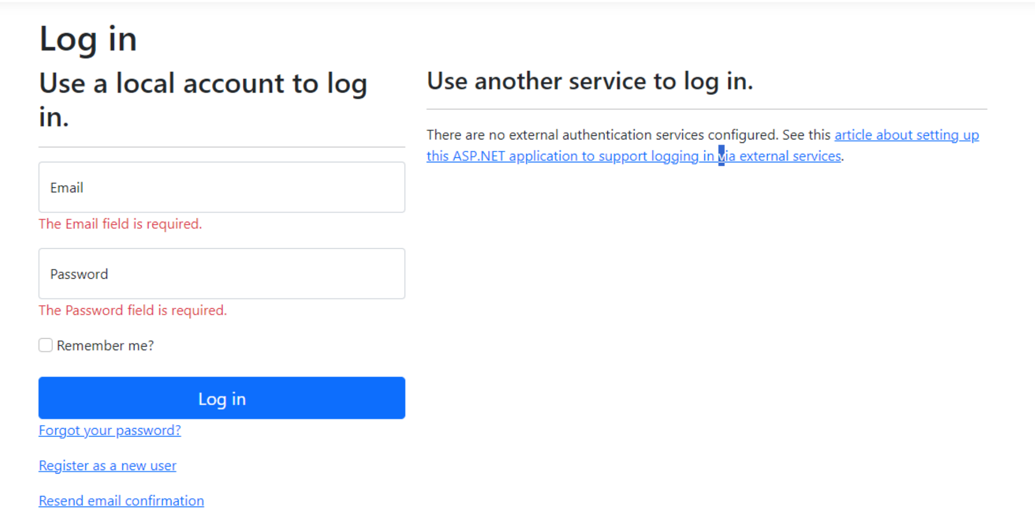
Slika 5.10. Početna stranica

Pritiskom na gumb Open chat u doljnjem lijevom kutu otvara se prozor na kojemu je prikazano chatbot sučelje, inspiracija za dizajn je preuzeta od ChatGPT official stranice. Pri otvaranju sučelja ostatak stranice postaje zatamljen kako bi se chatbot istaknuo. Na Slici 5.11. prikazano je sučelje chatbota.



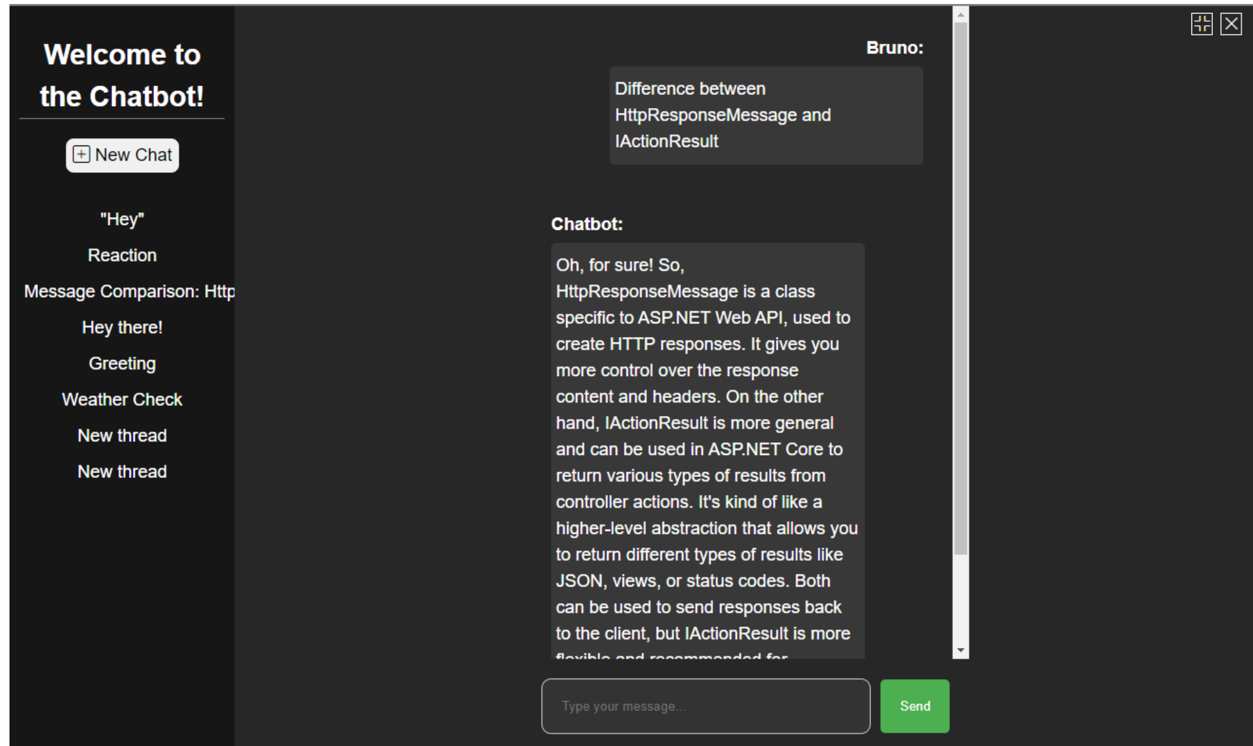
Slika 5.11. Chatbot sučelje

Važno je napomenuti kako je to izgled sučelja kada korisnik nije prijavljen u svoj račun, s lijeve strane nema prijašnjih razgovora korisnika, niti gumb za kreiranje novog razgovora. Kada se pritisne X u gornjem desnom kutu zatvara se sučelje i dobiva se ponovo izgled početne stranice. Kada se korisnik prijavi u svoj račun i klikom na Login gumb u gornjem desnom kutu te unese svoje podatke u formu oblika prikazanog na Slici 5.12. korisnik postaje autoriziran i može započeti korištenje aplikacije.



Slika 5.12. Login sučelje

Nakon što se korisnik autorizira chatbot sučelje dobiva dodatne funkcionalnosti. To je gumb koji kreira prazan razgovor te se automatski prebacuje na njega, u gumb dolazi i lista prethodnih razgovora koje je vodio autorizirani korisnik sa chatbotom. Također odabirnom na pojedini razgovor, prikazuju se poruke tog razgovora. Na slici 5.13. vidljivo je sučelje chatbota kada je korisnik autoriziran.



Slika 5.13. Sučelje chatbota kada je korisnik autoriziran

Frontend server je hostan na Netlify platformi za hostanje JavaScript web aplikacija, tamo se nalazi iskompajlirana verzija main.jsx skripte pomoću koje se chatbot sučelje ubacuje u bilo koji HTML dokument. Sve što je potrebno za postizanje te funkcionalnosti je ubaciti sljedeću liniju koda u tijelo HTML dokumenta:

```
<script type="module" src="https://sven-chatbot.netlify.app/assets/main-dhwnwfzk.js"></script>
```

6. ZAKLJUČAK

U ovom radu teorijski je pokriveno područje uvoda u umjetnu inteligenciju i razlozi potrebza istom. Opisana je struktura chatbota i integracija umjtene inteligencija u mehanizam chatbota, također opisane su karakteristike i funkcionalnosti modernih chatbotova, neke od njih su: pružanje odgovora na upite korisnika, pomaganje u navigaciji i navođenje korisnika kroz specifične procese, automatizacija zadataka i prikupljanje podataka od korisnika, to su samo neke od mnogih. Prikazani su dva popularna chatbota na tržištu: Chatbox i Zoho SalesIQ, njihova međusobna usporedba, te po ćemu se razlikuju od autorovog rješenja. Spomenut je i OpenAI API, njegove prednosti i način na koji generira tekst, a sa strane tehnologija opisane su glavne tehnologije korištene u radu, to su .NET, React JS, te popularni DOM i nešto manje popularni Shadow DOM koji služi za enkapsulaciju stilova i komponenti u području razvoja web-a.

Zadatak završnog rada bio je razviti chatbot aplikaciju koja pruža sve navedene funkcionalnosti chatbotova, ali s dodatnom mogučnošću pristupanju chatbotu iz bilo kojeg odabranog HTML dokumenta na webu. Aplikacija je realizirana kao kombinacija funkcionalnosti, pristupanje iz web aplikacije preko koje se nudi praćenje prijašnjih starih i kreiranje novih razgovora i pristupanje izvan domene aplikacije preko chatbot sučelja koje nudi jedinstveni trenutni razgovor bez potrebe za logiranjem i praćenjem podataka. Aplikacija je razvijena unutar .NET i React JS okvira. U usporedbi s opisanim aplikacijama na tržištu ima prednosti i nedostatke poput manjka funcionalnosti u usporedbi s naprednijim Zoho SalesIQ Zobotom i cross-platform dizajnom koje pruža Chatbox. Autorova aplikacija može se okarakterizirati kao spoj ova dva rješenja, u kojemu se na jednom mjestu nalazi korisnički pristupačno, intuitivno sučelje zajedno s jednostavnosti korištenja i fleksibilnosti upotrebe i postavljanja chatbota u odgovarajuće okruženje.

Aplikaciju je moguće dodatno unaprijediti uporabom JWT autentifikacije koja omogućava dijeljenje podataka kroz različite domene, kao i jaču sigurnonost. Također unaprijeđenje izgleda i dizajna aplikacije je moguće poput uvođenja slika na profil korisnika ili nešto slično. Nadalje moguće je implementirati algoritam da ograniči područe znanja ovisno o domeni s koje se pristupa chatbotu, na primjer ukoliko zahtjevi dolaze s www.mefos.hr domene, ograničiti chabtota na medicinske upite i pojmove. To daje dodatnu osobnosti chatbotu i kreira okruženje specifično njegovim korisnicima.

LITERATURA

[1] B. Church, »5 types of chatbot and how to choose the right one for your business,« IBM, September 2023. [Mrežno]. Available: https://www.ibm.com/blog/chatbot-types/. [Pokušaj pristupa August 2024].

[2] Oracle Cloud, »What is a Chatbot?,« [Mrežno]. Available: https://www.oracle.com/chatbots/what-is-a-chatbot/. [Pokušaj pristupa August 2024].

[3] Bin-Huang, »chatbox,« GitHub, May 2024. [Mrežno]. Available: https://github.com/Bin-Huang/chatbox/tree/main.

[4] Zoho, »Zoho SalesIQ: Features,« [Mrežno]. Available: https://www.zoho.com/salesiq/features.html?src=headermenu. [Pokušaj pristupa August 2024].

[5] Zoho, »Zoho SalesIQ: Chatbots,« [Mrežno]. Available: https://www.zoho.com/salesiq/chatbot.html?src=headermenu. [Pokušaj pristupa August 2024].

[6] Botsify, »Features,« [Mrežno]. Available: https://botsify.com/features. [Pokušaj pristupa August 2024].

[7] University of Central Arkansas, »Chat GPT: What is it?,« [Mrežno]. Available: https://uca.edu/cetal/chat-gpt/. [Pokušaj pristupa August 2024].

[8] HubSpot, »Free AI Chatbot Builder,« [Mrežno]. Available: https://www.hubspot.com/products/crm/chatbot-builder?hubs_content=blog.hubspot.com/service/customer-service-bots&hubs_content-cta=HubSpot%20Chatbot%20Builder&hubs_post=blog.hubspot.com/service/customer-service-bots&hubs_post-cta=HubSpot%20Chatbot%20Builde. [Pokušaj pristupa August 2024].

[9] OpenAI, »Docs/Developer quickstart,« [Mrežno]. Available: https://platform.openai.com/docs/quickstart. [Pokušaj pristupa August 2024].

[10] A. B. CCN, »What is OpenAI and what does it do?,« March 2024. [Mrežno]. [Pokušaj pristupa August 2024].

[11] GPT Workspace, »Understanding OpenAI GPT Tokens,« [Mrežno]. Available: https://gpt.space/blog/understanding-openai-gpt-tokens-a-comprehensive-guide. [Pokušaj pristupa August 2024].

[12] gewarren, ».NET (and .NET Core) - introduction and overview,« March 2023. [Mrežno]. Available: https://learn.microsoft.com/en-us/dotnet/core/introduction. [Pokušaj pristupa August 2024].

[13] BillWagner, »A tour of C# - Overview,« May 2023. [Mrežno]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. [Pokušaj pristupa August 2024].

[14] React, »React Reference Overview,« April 2024. [Mrežno]. Available: https://react.dev/reference/react. [Pokušaj pristupa August 2024].

[15] Mdm web docs, »JavaScript,« March 2024. [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript#reference. [Pokušaj pristupa August 2024].

[16] Mdn web docs, »Document Object Model; Introduction to the DOM,« 26 July 2024. [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. [Pokušaj pristupa August 2024].

[17] Imperva, »Learning Center: Shadow DOM,« [Mrežno]. Available: https://www.imperva.com/learn/application-security/shadow-dom/. [Pokušaj pristupa August 2024].

[18] D. Patel, »The Role of React Shadow DOM in Creating Truly isolated Components,« DhiWise, 29 January 2024. [Mrežno]. Available: https://www.dhiwise.com/post/the-role-of-react-shadow-dom-in-creating-isolated-components. [Pokušaj pristupa August 2024].

[19] Elastic, »What is a large language model (LLM)?,« [Mrežno]. Available: https://www.elastic.co/what-is/large-language-models. [Pokušaj pristupa August 2024].

[20] J. Chen, »What Is a Neural Network,« Investopedia, July 2024. [Mrežno]. Available: https://www.investopedia.com/terms/n/neuralnetwork.asp. [Pokušaj pristupa August 2024].

[21] GeeksforGeeks, »What is a Large Language Model (LLM),« January 2024. [Mrežno]. Available: https://www.geeksforgeeks.org/large-language-model-llm/. [Pokušaj pristupa August 2024].

[22] L. Gupta, »What is REST?,« REST API Tutorial, 12 December 2023. [Mrežno]. Available: https://restfulapi.net/. [Pokušaj pristupa August 2024].

[23] T. P. Team, »What Is a REST API? Examples, Uses, and Challenges,« Postman, 28 June 2023. [Mrežno]. Available: https://blog.postman.com/rest-api-examples/. [Pokušaj pristupa August 2024].

[24] Microsoft, »Introduction to Identity on ASP.NET Core,« 26 April 2024. [Mrežno]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio. [Pokušaj pristupa August 2024].

[25] A. Chiarelli, »Cookies, Tokens, or JWTs? The ASP.NET Core Identity Dilemma,« Auth0, 23 November 2023. [Mrežno]. Available: https://auth0.com/blog/cookies-tokens-jwt-the-aspnet-core-identity-dilemma/. [Pokušaj pristupa August 2024].

[26] Microsoft, »Overview of ASP.NET Core MVC,« 26 September 2023. [Mrežno]. Available: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-8.0&WT.mc_id=dotnet-35129-website. [Pokušaj pristupa August 2024].

Sažetak

Naslov: Integracija chatbota u REST framework

Sažetak: Glavni problem ovog završnog rada dijeli se na dva dijela, prvi problem je kreiranje chatbot sučelje koje pruža sve osnovne funkcionalnosti chatbota poput razgovora, a drugi problem koji se nadovezuje na prvi je omogućiti njegovo korištenje diljem weba kroz ubacivanje skripte u HTML dokument. Kako bi se to postiglo bilo je potrebno istražiti teoriju chatbotova, njihove karakteristike i kako rade, zatim istražiti tržište i provjeriti postoji li slično rješenje. U radu su opisana dva postojeća chatbota Zoho SalesIQ Zobot i Chatbox koji su najsličniji autorovu rješenju. Nadalje, integracija OpenAI API-ja s autovorim API-jem je ključan dio u osposobljavanju funkcionalnosti chatbota, također je u radu opisan princip na koji se generira tekst od strane LLM-ova (engl. large language model). Pred kraj je opisan arhitekturalni stil weba REST kojeg se bilo potrebno držati tijekom cijele izrade praktičnog dijela rada, dakako, opisane su krajnje točke web aplikacije na poslužiteljskom dijelu i kompletno sučelje i arhitektura klijentskog dijela aplikacije. Na poslijetku je zadatak rada u potpunosti izvršen uz mogučnost unaprijeđenja kompletne aplikacije u pogledu bolje autorizacije korisnika i dizajna sučelja.

Ključne riječi: chatbot, klijent, poslužitelj, sučelje, web aplikacija

Abstract

Title: Chatbot integration with REST framework

Abstract: The main problem of this thesis is divided into two parts. The first problem is creating a chatbot interface that provides all the basic functionalities of a chatbot, such as conversation. The second problem, which builds on the first, is enabling its use across the web by embedding a script into an HTML document. To achieve this, it was necessary to research the theory of chatbots, their characteristics, and how they work, followed by market research to check for similar solutions. The thesis describes two existing chatbots, Zoho SalesIQ Zobot and Chatbox, which are most similar to the author's solution. Furthermore, the integration of the OpenAI API with the author's API is a crucial part of enabling the chatbot's functionalities. The thesis also describes how text is generated by LLMs (Large Language Models). Toward the end, the architectural style of the web, REST, is described, which needed to be followed throughout the practical part of the work. Additionally, the endpoints of the web application on the server side, along with the complete interface and architecture of the client side of the application, are described. Finally, the task of the thesis was fully accomplished, with potential improvements in the overall application regarding better user authorization and interface design.

Keywords: chatbot, client, interface, server, web application