Aplikacija za inženjering upita razgovornih agenata

Šare, Josip

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1429

APLIKACIJA ZA INŽENJERING UPITA RAZGOVORNIH AGENATA

Josip Šare

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1429

APLIKACIJA ZA INŽENJERING UPITA RAZGOVORNIH AGENATA

Josip Šare

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1429

Pristupnik:	**Josip Šare (0036540341)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Marko Horvat

Zadatak: **Aplikacija za inženjering upita razgovornih agenata**

Opis zadatka:

Inženjering upita (engl. prompt engineering) je vještina oblikovanja precizno formuliranih upita (engl. prompts) koji učinkovito usmjeravaju razgovorne agente (engl. chatbots) u generiranju traženih odgovora. Ova tehnička vještina omogućava bolju i precizniju komunikaciju čovjeka i stroja, a danas postaje sve zanimljivija jer razgovorni agenti dobivaju nova i važna područja primjene. U ovom radu potrebno je upoznati se s osnovnim principima inženjeringa upita, uključujući kako pravilno formulirati upite koji potiču razgovorne agente na generiranje korisnih, preciznih i sigurnih odgovora. Cilj završnog rada je izraditi aplikaciju za oblikovanje upita definiranih formata i strukture na temelju principa inženjeringa upita. Pohraniti upite u relacijsku bazu podataka i preoblikovati ih. Odabrati referentne razgovorne agente i provesti eksperimentalno vrednovanje upita te statističku obradu rezultata. Radu priložiti izvorni i izvršni kod razvijenog sustava uz potrebna dodatna objašnjenja i dokumentaciju. Također, radu priložiti izrađene skupove podataka te citirati korištenu literaturu.

Rok za predaju rada: 14. lipnja 2024.

Uvod

Stručnjaci i dalje ne uspijevaju pronaći definiciju inteligencije bez da se ne vežu na pojam ljudske inteligencije. Pojam umjetne inteligencije neki poistovjećuju sa znanosti i inženjerstvom izrade inteligentnih strojeva, posebice inteligentnih računalnih programa. Finalni cilj razvoja umjetne inteligencije je doći do razine ljudske inteligencije, pa onda i nju premašiti [1]. Računalni sustavi se obično nazivaju NLP sustavima ako obrađuju jezik u tekstualnom obliku, umjesto u govornom obliku. Tekstualni unos je lakše obraditi jer su upisani znakovi jednostavni za dekodiranje programima [2]. Unos niza riječi u računalo nije dovoljan. Ni obrada rečenica nije dovoljno. Računalu se mora omogućiti razumijevanje domene o kojoj tekst govori [1]. U NLP sustavima, sintaktičko znanje obično se kodira u obliku skupa pravila, nazvanog gramatika. Gramatika se koristi za obradu rečenica; to jest, za izvođenje njihove strukture. Obrada rečenice omogućuje NLP programu da eliminira neka moguća značenja riječi u rečenici i dekodira informacije o ulogama koje riječi imaju [2]. Inženjering upita ključna je tehnika u području obrade prirodnog jezika koja uključuje projektiranje i optimizaciju upita korištenih za unos informacija u modele, s ciljem poboljšanja njihove izvedbe na specifičnim zadacima. S nedavnim napretkom u razvoju velikih jezičnih modela, inženjering upita pokazao je značajnu superiornost u različitim domenama i postao sve važnija tema proučavanja [3].

Cilj ovog rada je konstruirati web aplikaciju za inženjering upita koja bi omogućila korisnicima da razne vrste upita upotpune s više informacija u samo pak klikova. Aplikacija će koristiti vanjska LLM API sučelja kako bi procesuirala stvoreni poboljšani upit te prikazala odgovor zajedno s povezanim informacijama.

U nastavku će biti objašnjeni osnovni koncepti inženjeringa upita kao i pobliže objašnjeno što sačinjava pojedinačni upit. Bit će dano više detalja o ostvarivanju komunikacije s LLM API sučeljima kao i o procesuiranju i nadopunjavanju korisničkih upita.

1. Inženjering upita

Projektiranje odnosno inženjering upita pojavilo se kao napredni pristup u području obrade prirodnog jezika (NLP), nudeći učinkovitiji i ekonomičniji način korištenja velikih jezičnih modela (LLM). Ovaj inovativni pristup temelji se na razvoju LLM-ova, koji su revolucionirali naše shvaćanje prirodnog jezika [1] . Dijelove upita možemo kategorizirati u četiri dijela, instrukciju, kontekst, ulazne podatke i indikator izlaznih podataka. Instrukcija zadaje osnovni zadatak koji se očekuje od LLM-a nad dalje unesenim podatcima. Kontekstom zadajemo dodatne informacije koje pomažu u stvaranju šire slike te generiranju boljeg odgovora. Ulaznim podatcima zadajemo ono što mi želimo da model procesira i ovisno o instrukciji, odnosno prvom dijelu upita, LLM obrađuje te podatke. Posljednji dio je indikator izlaznih podataka daje LLM-u informaciju u kakvom se formatu očekuje odgovor [4]. Veliki jezični modeli (LLM-ovi) predstavljaju moćne alate za mnoge zadatke obrade prirodnog jezika kada se koriste odgovarajući upiti. Međutim, zbog osjetljivosti modela, pronalaženje optimalnog upita može biti izazovno, često zahtijevajući opsežne ručne pokušaje i pogreške. Projektiranje upita se ugrubo dijeli na dvije vrste, a to su ručno projektiranje odnosno mi kao korisnici sami konstruiramo upit i automatsko projektiranje koje upotrebljava takozvane meta-upite. To je vrsta upita koja se postavi LLM-u i od modela se očekuje da vrati optimalni upit za dobivanje željene informacije, drugim riječima LLM projektira upit za nas [5]. Ovaj rad će se fokusirati na prvu vrstu projektiranja upita s idejom uključivanja računalnih algoritama i grafičkog sučelja kako bi korisniku pojednostavnili posao. Uobičajena praksa za poboljšanje zaključivanja s velikim jezičnim modelima (LLM-ovima) jest generiranje međukoraka zaključivanja, koristeći metode poput lančanog upita ili razlaganja pitanja ili korištenje neke forme fraze „think step-by-step“ [6]. U ovom radu je korisniku dana opcija nadograđivanja svoj inicijalnog upita, uz ostale opcije, s frazom „Take a deep breath and think step-by-step“. Nadalje, možemo koristiti razdjelnike kako bi smo jasno označili različite dijelove unosa [7]. U ovom radu se koristi ta strategija tako što se dijeli upita odvajaju novim redovima i znakovima „#“.

2. Aplikacija za projektiranje upita

2.1. Specifikacija programske potpore

Glavna zadaća aplikacije za projektiranje upita je na temelju prvotnog upita unesenog od strane korisnika te od opcija koje je korisnik odabrao, generirati sofisticiraniji upit. Zatim se taj upit s pomoću OpenAI API te LLamaAI API arhitektura šalje na jedan od dva ponuđena LLM-modela (ChatGPT_3.5-Turbo, LLamaAI-13b-chat) te se korisniku prikazuje dobiveni odgovor.

2.1.1. Funkcionalni zahtjevi

Aplikacija mora zadovoljiti slijedeće funkcionalne zahtjeve:

1. **Registriranje u sustav**
   Korisnik se može registrirati u sustav koristeći svoju adresu e-pošte, ime te lozinku po želji. Pritiskom na gumb *Register* nakon napravljene prijašnje radnje korisniku se učitava aplikacija s njegovim korisničkim računom te se njegov račun sprema u bazu podataka. Ukoliko se korisnik pokuša registrirati u sustav s adresom e-pošte koja se već nalazi u sustavu o tome će biti obaviješten.

2. **Prijava u sustav**
   Korisnik se može prijaviti u sustav ukoliko već postoji njegov račun u bazi podataka. To se čini tako da se na Login ekranu upišu odgovarajuća adresa e-pošte te pripadajuća lozinka. Ukoliko je lozinka ili e-pošta krivo unesena, korisniku će biti prikazana poruka o krivom unosu no ne o tome koji je dio unosa kriv.

3. **Odabir korištenog LLM modela**
   Korisnik ima opciju biranja želi li da aplikacija za daljnje funkcionalnosti koristi ChatGPT_3.5-Turbo model ili LLamaAI-13b model. Taj se odabir može promijeniti poslije tokom uporabe aplikacije.

4. **Unos početne naredbe**
   Korisnik mora unijeti početni upit na temelju kojeg želi dobiti unaprijeđeni upit s obzirom na opcije koje je korisnik odabrao.

5. **Odabir vrste upita**

   Korisnik nakon što je unio upit odabire o kojoj se vrsti upita riječ. Kao ponuđeno ima opcije *Code* (Kod), *Teaching* (Učenje), *Rephrasis* (Preformuliranje) i *Writing* (Pisanje). Odabirom jedne od četiri ponuđene opcije korisniku se pokazuje dodatni odabir za specificiranje daljnjih uputa.

6. **Odabir količine detalja**

   Korisnik odabire jednu od tri ponuđene opcije za željenu količinu detalja koju očekuje kao ispis. Ponuđene opcije su *No detail* (Bez detalja) *, Normal detail* (Prirodna količina detalja)*, Extra detail* (Dodatni detalji). Odabirom jedne od tri ponuđene opcije se šalje odgovarajuća instrukcija LLM modelu.

7. **Odabir podvrste upita**

   Korisnik odabire jednu od više opcija podvrsta upita koje se generiraju na temelju prijašnjeg odabira vrste upita. Korisnik i dalje može promijeniti svoj odabir za vrstu upita čime se mijenjaju i ponuđene opcije za odabir podvrsta upita.

8. **Odabir temperature upita te opcije „Take a deep breath“**

   Korisnik ima opciju specificiranja temperature koju će LLM koristiti u generiranju odgovora. Korisnik može odabrati vrijednost od 0.0 do 1.0. Uz to korisnik može odabrati opciju da se u poboljšani upit uključi izraz „Take a deep breath and think step by step“ odabirom na opciju „Take a deep breath“.

9. **Slanje podataka na poslužiteljsku stranu**

   Pritiskom na gumb *Submit* nakon što je korisnik unio sve potrebne informacije, iste se šalju na poslužiteljsku stranu na daljnju obradu.

10. Nadopunjavanje prvotnog upita

Na temelju odabranih opcija za poboljšavanje upita u aplikaciji, na poslužiteljskom strani se dohvaćaju odgovarajuće nadopune upita iz baze podataka. Proces dohvaćanja nadopune upita iz baze podataka se odvija s pomoću ključnih riječi upita te informacija koje je korisnik odabrao odnosno unio.

11. Slanje poboljšanog upita na odgovarajući LLM

Nakon što je nadopunjavanje upita gotovo, poslužiteljska strana aplikacije komunicira s jednim od dva ponuđena API-a, ovisno o tome koja je opcija odabrana na klijentskoj strani aplikacije. Na odabrani API se šalje poboljšani upit koji se sastoji od prvotnog upita te od nadopuna upita koje su se uparile prije ovog koraka. Kao odgovor se očekuje sami odgovor LLM-a kao i informacije o broju tokena potrebnih za odgovor te za upit.

12. Slanje poboljšanog upita te odgovora na klijentsku stranu aplikacije

Nakon što je poslužiteljska strana aplikacije dobila odgovor od odabranog LLM-a, informacije vezane za poboljšani upit te odgovor se šalju na klijentsku stranu aplikacije.

13. Spremanje informacija u bazu podataka

U bazu podataka se spremaju informacije o početnom upitu, opcijama odabranim za generiranje poboljšanog upita, odgovoru od LLM-a te o uparivanju poboljšanog upita s nadopunama upita iz baze podataka.

14. Prikaz poboljšanog upita i odgovora korisniku

Korisniku se prikazuju poboljšan upit generiran na temelju početnog unosa upita te odabranih opcija kao i odgovor dobiven od strane odabranog LLM-a. Korisnik ima opciju kopiranja čitavog poboljšanog upita kao i čitavog odgovora.

15. **Ocjenjivanje odgovora**

    Nakon što se korisniku prikažu poboljšan upit te odgovor, korisnik ima opciju ocjenjivanja poboljšanog upita te odgovora s ocjenama od 0 do 10. Ocjene se potom na pritisak gumba *Review Response* šalju na poslužiteljsku stranu aplikacije te se spremaju uz odgovarajući odgovor.

16. **Prikaz svih upita od određenog korisnika**

    Korisnik odlaskom na prikaz *Profile* može vidjeti sve svoje upite koje je unio kao i odgovarajuće odgovore te informacije o njima kao što su zauzeće u tokenima te duljina.

2.1.2. Ostali zahtjevi

- Korisničke lozinke se moraju spremati u bazu podataka na siguran način.
- Pri prijavi korisnika u sustav uspoređuje se unesena adresa e-pošte te lozinka se podatcima u bazi podataka.
- Vrijeme od trenutka slanja podataka upita na poslužiteljsku stranu do trenutka prikaza odgovora na klijentskoj strani ne smije biti predugo (maksimalno 5 sekundi).
- Pristup bazi podataka ne smije trajati dulje od 2 sekunde.
- Grafičko sučelje mora biti jednostavno te intuitivno za korištenje.
- U upitu se ne smiju nalazi vulgarni izrazi jer bi LLM mogao odlučiti ne odgovoriti na takav upit.

2.1.3. Obrasci uporabe

Popis i opis obrazaca uporabe:

UC01-Registracija u sustav

- Glavni sudionik: Neregistrirani korisnik
- Cilj: Stvoriti korisnički račun
- Sudionici: Baza podataka
- Preduvjet: -
- Opis osnovnog tijeka:
    1. Korisnik bira opciju ”Make an account” na sučelju web aplikacije
    2. Korisnik unosi tražene podatke
    3. Korisnik je upisan u bazu podataka
- Opis mogućih odstupanja:
    1. Korisnik unosi neispravnu/već postojeću e-mail adresu
        a. Sustav obavještava korisnika o pogrešci
        b. Korisnik unosi ispravne podatke i registracije se uspješno završi

UC02-Prijava u sustav

- Glavni sudionik: Registrirani korisnik
- Cilj: Prijaviti se u postojeći korisnički račun
- Sudionici: Baza podataka
- Preduvjet: Registriran korisnik
- Opis osnovnog tijeka:
    1. Korisnik bira opciju ”login” na sučelju web aplikacije
    2. Korisnik unosi tražene podatke
    3. Korisnik je uspješno prijavljen u svoj račun
- Opis mogućih odstupanja:
    1. Korisnik unosi neispravnu e-mail adresu ili lozinku
        a. Sustav obavještava korisnika o pogrešci no ne specificira gdje je pogreška
        b. Korisnik unosi ispravne podatke i prijava je uspješno završila

UC03-Pregled prijašnjih upita

- Glavni sudionik: Prijavljeni korisnik
- Cilj: Dobiti prikaz svih upita učinjenih od strane prijavljenog korisnika
- Sudionici: Baza podataka
- Preduvjet: Prijavljen korisnik
- Opis osnovnog tijeka:
    1. Korisnik bira opciju ”Profile” na sučelju web aplikacije
    2. Korisnik dobije prikaz svih do tad učinjenih upita i odgovora
- Opis mogućih odstupanja: -

UC04-Unos upita te opcija

- Glavni sudionik: Prijavljeni korisnik
- Cilj: Unijeti upit te opcije u aplikaciju
- Sudionici: Baza podataka
- Preduvjet: Prijavljen korisnik
- Opis osnovnog tijeka:
    1. Korisnik odabire koji LLM želi da mu vrati odgovor
    2. Korisnik unosi svoj upit te odabire odgovarajuće opcije
    3. Korisnik pritiskom na gumb „Submit“ šalje svoj upit s odgovarajućim informacijama poslužiteljskoj strani aplikacije
- Opis mogućih odstupanja:
    1. Korisnik nije odabrao jednu od zahtijevanih opcija
        a. Sustav obavještava korisnika ili slanje upita nije moguće
        b. Korisnik ispravno unese podatke te slanje upita bude uspješno završeno

UC05-Nadopunjavanje početnog upita s nadopunama

• Glavni sudionik: Poslužiteljska strana aplikacije

• Cilj: Nadopuniti početni upit s nadopunama iz baze podatka ovisno o odabranim opcijama

• Sudionici: Baza podataka

• Preduvjet: Uspješno poslan upit i potrebne informacija na poslužiteljsku stranu

• Opis osnovnog tijeka:

1. Početni upit se sprema u bazu podataka
2. U glavnom programu za obradu se na temelju odabranih opcija prosljeđuje obrada na druge potprograme
3. Na temelju odabranih opcija se iz baze podataka dohvaćaju potrebne nadopune te se spajaju na odgovarajući način s početnim upitom
4. Spremanje nadopunjenog upita u bazu podataka
5. Nadopunjeni upit se šalje na daljnje procesiranje

• Opis mogućih odstupanja: -

UC06-Slanje poboljšanog upita LLM API-u

• Glavni sudionik: Poslužiteljska strana aplikacije

• Cilj: Poslati poboljšani upit LLM API-u

• Sudionici: LLM API

• Preduvjet: Uspješno formiran poboljšani upit

• Opis osnovnog tijeka:

1. Na temelju odabrane opcije u početnom upitu se odabire kojem će se LLM API-u poboljšani upit slati
2. Poboljšani upit se šalje na LLM API

• Opis mogućih odstupanja: -

UC07-Dohvaćanje odgovora od LLM API-a te prikaz na ekran

• Glavni sudionik: Poslužiteljska strana aplikacije

• Cilj: Dohvatiti i obraditi odgovor od LLM API-a te ga prikazati na ekranu zajedno s poboljšanim upitom

• Sudionici: LLM API, Baza podataka, Korisnik

• Preduvjet: Uspješna obrada upita na LLM API-u

• Opis osnovnog tijeka:

1. Primitak odgovora od strane LLM API-a te njegova obrada kako bi se informacije mogle poslati Korisniku
2. Spremanje obrađenog odgovora u bazu podataka
3. Slanje obrađenog odgovora te poboljšanog upita na klijentsku stranu aplikacije te prikaz na ekranu

• Opis mogućih odstupanja: -

UC08-Ocjenjivanje odgovora

• Glavni sudionik: Klijent

• Cilj: Ocijeniti posljednji odgovor i poboljšani upit te tu ocjenu pohraniti u bazu podataka

• Sudionici: Baza podataka

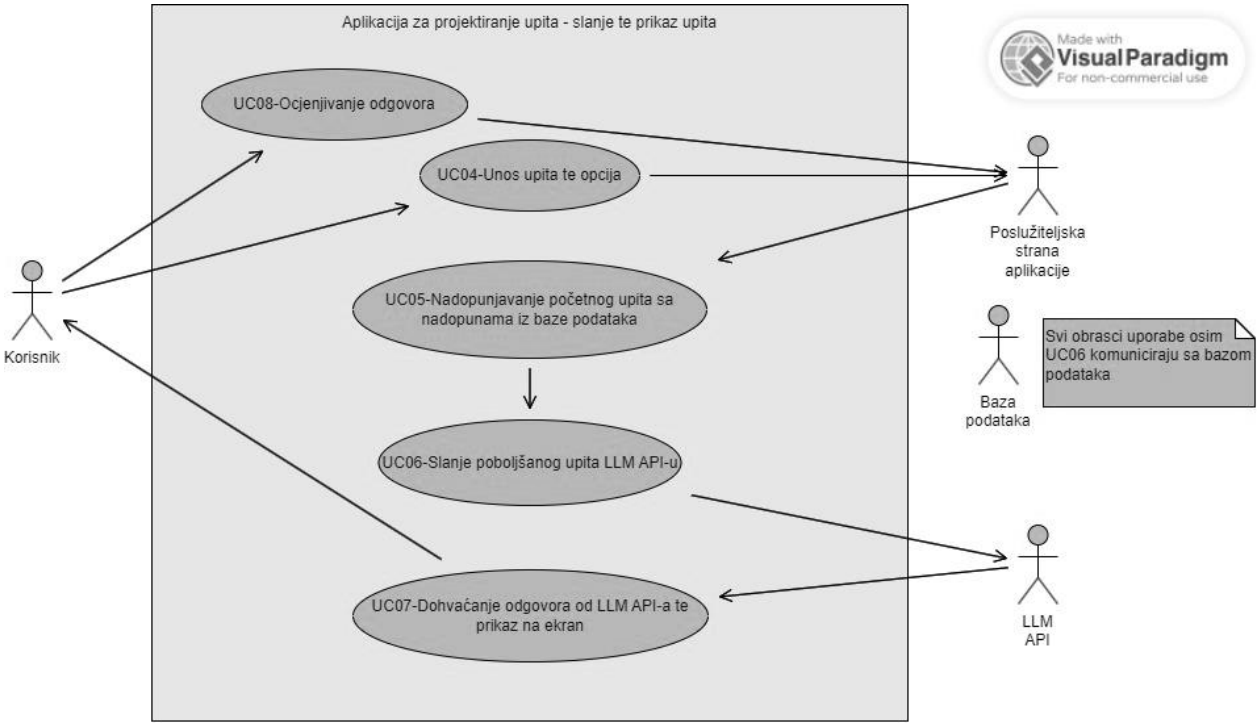
• Preduvjet: Uspješno obrađen i prikazan odgovor te poboljšani upit

• Opis osnovnog tijeka:

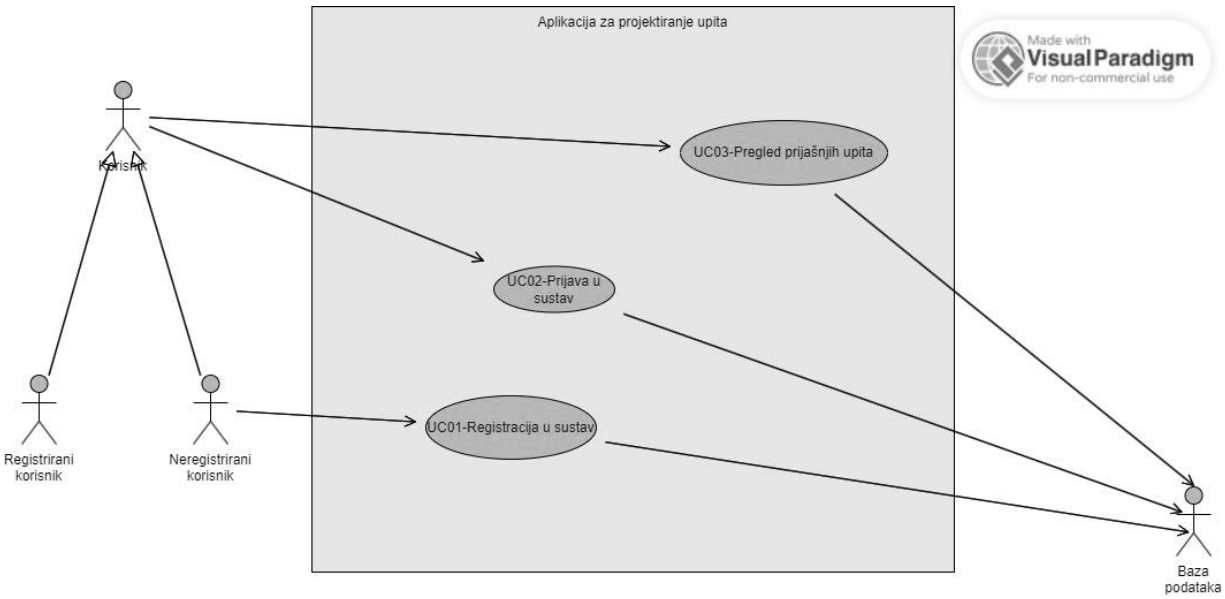
1. Korisnik nakon prikaza odgovora i poboljšanog upita odabire ocjenu od 0 do 10
2. Pritiskom na gumb „Review Response“ se ta ocjena šalje na poslužiteljsku stranu aplikacije
3. Pohranjivanje ocjene za vezani odgovor

• Opis mogućih odstupanja: -

Slijedeći grafovi su napravljeni korištenjem web-stranice VisualParadigm [10].



Slika 2.1 Dijagram obrasca uporabe za slanje te prikaz upita



Slika 2.2 Dijagram obrasca uporabe za prijavu u sustav te prikaz upita

2.2. Dizajn i arhitektura sustava

Arhitektura ovog programskog sustava može se razlučiti na sljedeće dijelove:

- Klijentska strana aplikacije
- Poslužiteljska strana aplikacije
- Baza podataka
- LLM API

Klijentska strana aplikacije služi za prikaz informacija krajnjem korisniku preko web preglednika te kao način komunikacije korisnika s ostatkom sustava. Tehnologija korištena za izradu je programski jezik JavaScript s bibliotekom React o kojima će više biti riječ kasnije u radu, točnije u poglavlju „Implementacija sustava“.

Poslužiteljska strana aplikacije služi za primanje i obradu svih informacija koje dolaze sa klijentske strane. Uz to, zadaća poslužiteljske strane je komunikacija sa bazom podataka te obrada rezultata kao i slanje tako obrađenih rezultata na klijentsku stranu aplikacije. Također poslužiteljska strana aplikacije je zaslužna za komunikaciju sa vanjskim sučeljima odnosno OpenAI API-jem i LLamaAI API-jem za ostvarivanje komunikacije s LLM-om. Tehnologija korištena za izradu je programski jezik Java u sklopu radnog okvira Spring Boot o kojima će više biti riječ kasnije u radu, točnije u poglavlju „Implementacija sustava“.

Baza podataka služi za pohranu i dohvaćanje svih relevantnih podataka vezanih za aplikaciju. Komunicira s poslužiteljskom stranom aplikacije s pomoću ORM okvira Hibernate. Tehnologija korištena za izradu je PostgreSQL. Više o detaljima implementacije će biti kasnije u radu, točnije u poglavlju „Implementacija sustava“.

Za komunikaciju s LLM modelima koriste se sljedeća dva API-a: OpenAI API i LLamaAI API. Sučelja pružaju jednostavan i intuitivan način komunikacije s pripadajućim LLM modelima. U slučaju ove aplikacije to su ChatGPT 3.5-Turbo i LLama-13b-chat modeli. Više detalja o arhitekturi i implementaciji će biti objašnjeno kasnije u radu u poglavlju „Implementacija sustava“.

2.2.1. Baza podataka

Podatci potrebni za rad aplikacije te podatci koje aplikacija kroz rad generira se spremaju u relacijsku bazu podataka. Entiteti su sljedeći:

- *user*
- *user_prompt*
- *template*
- *finished_prompt*
- *llm*
- *response*

Opis entiteta

User

Entitet *User* (Korisnik) modelira registriranog korisnika aplikacije sa svim potrebnim atributima odnosno informacijama. Sadrži sljedeće atribute: ID, e-mail adresa, lozinka, uloga, korisničko ime. Ovaj entitet je u *One-to-Many* vezom sa entitetom *user_prompt* preko svog atributa ID.

Tablica 2.1 Entitet User

ID	BIGINT	Jedinstveni identifikator korisnika
e-mail adresa	VARCHAR(255)	Jedinstvena e-mail adresa korisnika
Lozinka	VARCHAR(255)	Kriptiran zapis lozinke
Uloga	VARCHAR(255)	Uloga korisnika u sustavu
Korisničko ime	VARCHAR(255)	Korisničko ime

User_prompt

Entitet *user_prompt* (korisnički upit) se koristi za modeliranje početnog upita koji korisnik zadaje u aplikaciju. Sadrži sljedeće atribute: ID, tip, extra tip, upit, ID korisnika, datum, temperatura, take a breath. Ovaj entitet je u *Many-to-One* vezom sa entitetom *user* preko atributa ID korisnika, te *One-to-One* vezom sa entitetom *finished_prompt* preko svog atributa ID, te u *One-to-One* vezi sa entitetom *response* preko svog atributa ID.

Tablica 2.2 Entitet user_prompt

ID	BIGINT	Jedinstveni identifikator korisničkog upita
ID korisnika	BIGINT	Jedinstveni identifikator korisnika
Tip	VARCHAR(255)	Tip upita
Extra tip	VARCHAR(255)	Podtip upita
Upit	TEXT	Tekst upita
Datum	TIMESTAMP(6)	Vrijeme postavljanja upita
Temperatura	DOUBLE PRECISION	Mjera halucinacije LLM-a
Take a Breath	BOOLEAN	Oznaka koja govori želi li korisnik nadodati frazu „Take a deep breath and think step-by-step“ na kraju svog upita

Finished_prompt

Entitet *finished_prompt* (nadopunjeni upit) modelira obrađeni odnosno nadopunjeni upit koji je rezultat obrade potprograma aplikacije. Sadrži sljedeće atribute: ID, tekst, ID korisničkog upita, duljina. Ovaj entitet je u *One-to-One* vezi sa entitetom *user_prompt* preko atributa ID korisničkog upita, u *One-to-One* vezi sa entitetom *response* preko svog atributa ID te u *Many-to-Many* vezi sa entitetom *template* preko svog atributa ID.

Tablica 2.3 Entitet finished_prompt

ID	BIGINT	Jedinstveni identifikator nadopunjenog upita
ID korisničkog upita	BIGINT	Jedinstveni identifikator korisničkog upita
Tekst	TEXT	Tekst nadopunjenog upita
Duljina	INTEGER	Duljina nadopunjenog upita

Response

Entitet *response* (odgovor) modelira obrađeni odgovor dobiven od vanjskog sučelja OpenAI API. Odgovor u svojoj prvotnoj formi sadrži više informacija no neke nisu značajne u okviru rada ove aplikacije. Sadrži sljedeće atribute: ID, ID korisničkog upita, ID nadopunjenog upita, ID LLM-a, tekst, broj tokena odgovora, broj tokena upita, duljina, ocjena. Ovaj entitet je u *One-to-One* vezi sa entitetom *finished_prompt* preko atributa ID nadopunjenog upita, u *One-to-One* vezi sa entitetom *user_prompt* preko atributa ID korisničkog upita, te u *Many-to-One* vezi sa entitetom *llm* preko atributa ID LLM-a.

Tablica 2.4 Entitet response

ID	BIGINT	Jedinstveni identifikator odgovora
ID korisničkog upita	BIGINT	Jedinstveni identifikator korisničkog upita
ID nadopunjenog upita	BIGINT	Jedinstveni identifikator nadopunjenog upita
ID LLM-a	BIGINT	Jedinstveni identifikator LLM-a korištenog za dobivanje odgovora
Tekst	TEXT	Tekst odgovora
Broj tokena odgovora	INTEGER	Broj tokena koji je odgovor zauzeo
Broj tokena upita	INTEGER	Broj tokena koji je nadopunjeni upit zauzeo
Duljina	INTEGER	Duljina odgovora
Ocjena	INTEGER	Ocjena odgovora

Llm

Entitet *llm* (LLM) modelira LLM model korišten za dobivanje odgovora. Sadrži sljedeće atribute: ID, ime. Ovaj entitet je u *One-to-Many* vezi sa entitetom *response* preko svog atributa ID.

Tablica 2.5 Entitet llm

ID	BIGINT	Jedinstveni identifikator LLM-a
Ime	VARCHAR(255)	Ime LLM-a

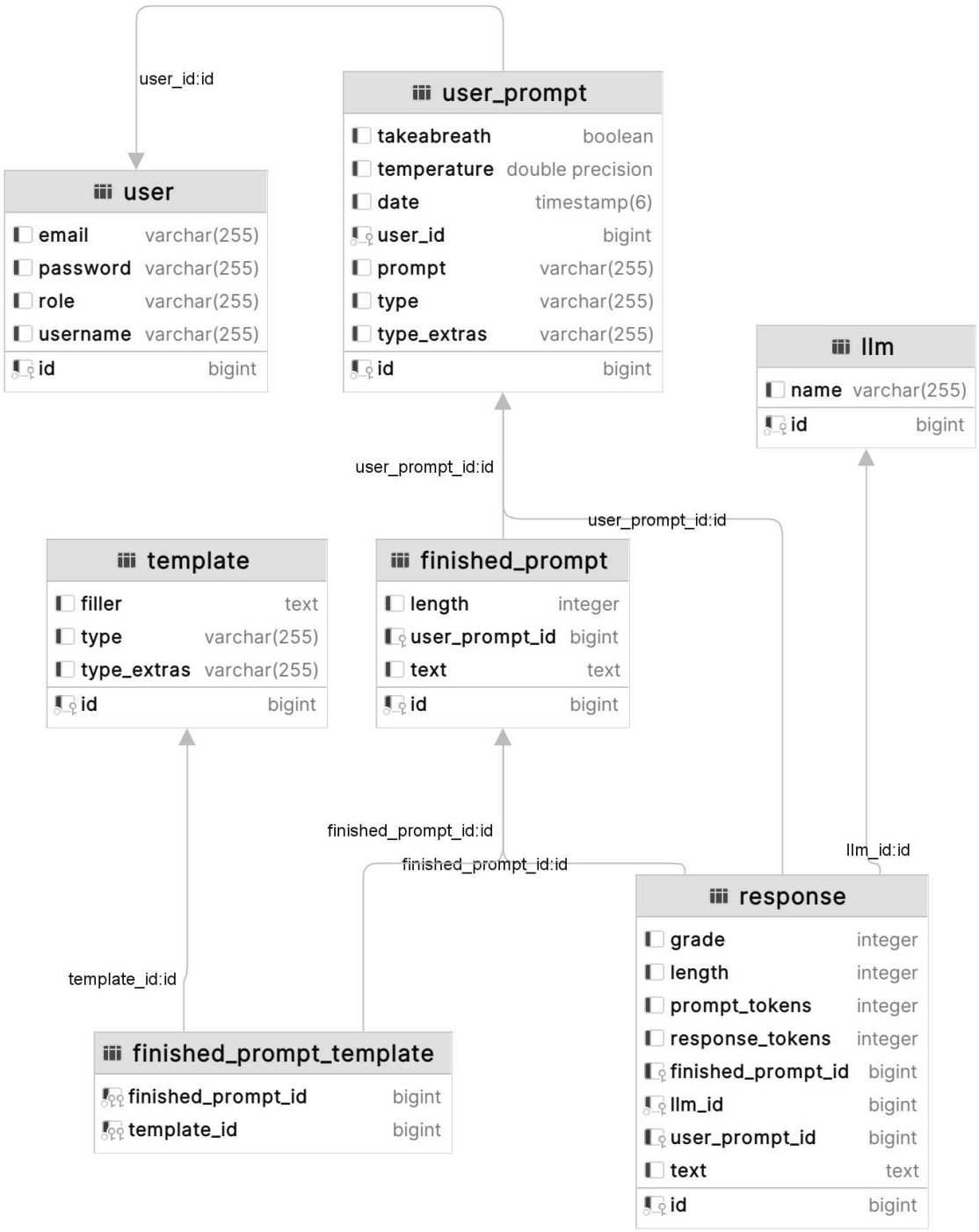
Template

Entitet *template* (nadopuna) modelira moguće nadopune korisničkog upita ovisno o izabranim opcijama. Sadrži sljedeće atribute: ID, tip, extra tip, nadopunu. Ovaj entitet je u *Many-to-Many* vezi sa entitetom *response* preko svog atributa ID.

Tablica 2.6 Entitet template

ID	BIGINT	Jedinstveni identifikator nadopune
Tip	VARCHAR(255)	Tip nadopune
Extra tip	VARCHAR(255)	Podtip nadopune
Nadopuna	TEXT	Tekst nadopune

Dijagram baze podataka



Slika 2.3 Dijagram baze podataka

2.2.2. Vanjsko sučelje LLM API

Za ostvarivanje komunikacije sa vanjskim sučeljem koje pruža ulogu LLM u sklopu ove aplikacije korišteni su API koje nude kompanije OpenAI i Meta, točnije njihov *Chat Completions API.* Komunikacija sa tim sučeljima se događa na poslužiteljskoj strani aplikacije radi bolje sigurnosti i jednostavnosti. Sa sučeljem se komunicira na specificiran način propisan u dokumentaciji tog API-a pozivanjem na URL „https://api.openai.com/v1/chat/completions“ odnosno „https://api.llama-api.com/chat/completions“ ovisno o tome za koji se LLM korisnik odlučio. Primjer jednog poziva API-a izgleda ovako:

```
curl https://api.openai.com/v1/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
    "model": "gpt-3.5-turbo",
    "messages": [
      {
        "role": "system",
        "content": "You are a helpful assistant."
      },
      {
        "role": "user",
        "content": "Who won the world series in 2020?"
      },
      {
        "role": "assistant",
        "content": "The Los Angeles Dodgers won the World
Series in 2020."
      },
      {
        "role": "user",
        "content": "Where was it played?"
      }
    ]
  }'
```

U području `Headers` se pod poljem `Authorization` mora nalaziti tekst u obliku „`Bearer <Open AI ključ>`“ odnosno „`Bearer <LLamaAI ključ>`“. Taj ključ se koristi za autorizaciju upita na poslužitelju OpenAI odnosno LLamaAI. U području `Body` se nalaze informacije o modelu te o porukama koje šaljemo modelu na obradu. Pod `model` se očekuje oznaka modela sa kojim se želi uspostaviti komunikacija, u slučaju ove aplikacije tu mogu biti „`llama-13b-chat`“ ili „`gpt-3.5-turbo`“. Pod `messages` se nalaze poruke koje šaljemo API-u na obradu. Svaka poruka može imati jednu od tri uloge odnosno `role` a to su: `user`, `assistant` i `system`. Uloga `system` se može koristiti za davanje uputa LLM-u na koji način da interpretira poruke, no slanje te poruke nije nužno jer se u protivnom pretpostavi *default* stanje. Uloga `user` signalizira da se sljedeća poruka interpretira kao upit LLM-u te da se na nju očekuje odgovor. Isključivo ova uloga se koristila u ovom radu odnosno za izradu aplikacije. Uloga `assistant` se koristi kad eksplicitno zadati LLM-u koji odgovor od njega očekujemo. Navođenje poruke sa ulogom `assistant` također nije potrebno za ispravno pozivanje API-a [8].

Odgovor LLM API-a je u ovom obliku:

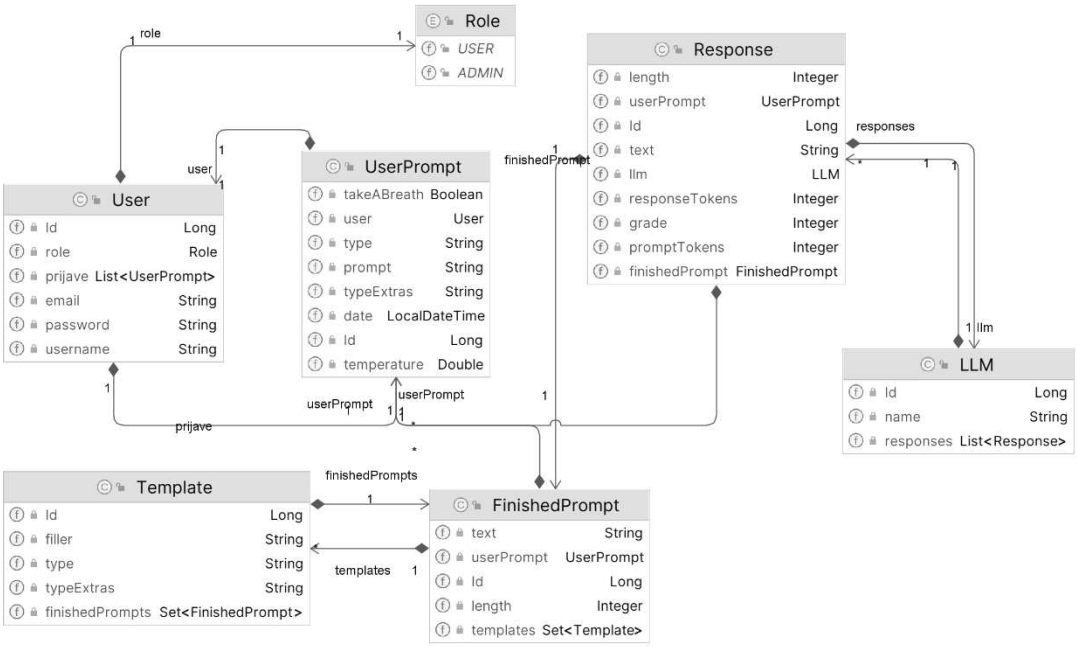
```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content": "The 2020 World Series was played in Texas
at Globe Life Field in Arlington.",
        "role": "assistant"
      }
    }
  ],
  "created": 1677664795,
  "id": "chatcmpl-7QyqpwdfhqwajicIEznoc6Q47XAyW",
  "model": "gpt-3.5-turbo-0613",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 17,
    "prompt_tokens": 57,
    "total_tokens": 74
```

```
    }
}
```

Pod poljem `choices` se nalaze mogući odgovori od LLM-a. U praksi se tu najčešće nalazi samo jedan član, no moguće ih je i više. Svaki član polja `choices` sadrži stavke `finish_reason`, `index`, `message`. Stavka `finish_reason` označava razlog prestanka generiranja odgovora. Moguće vrijednosti su sljedeći: `stop`, `length`, `function_call`, `content_filter` i `null`. Ako LLM smatra da je odgovor dovršen i da zadovoljava upit korisnika vraća vrijednost `stop`, ako odgovor zauzima previše tokena, odnosno predug je, vraćena vrijednost će biti `length`. Vrijednost `function_call` označava da je LLM pozvao neku prije definiranu funkciju u toku odgovora, dok `content_filter` označava da je odgovor prekinut jer ne zadovoljava kontekstualne filtre API-a. Vrijednost `null` znači da se odgovor još generira ili odgovor nije potpun. Stavka `message` se sastoji od stavke `content` u kojoj je sadržan odgovor od LLM-a u tekstualnom obliku, te od stavke *role* koja će u ovom slučaju biti `assistent`. Za ovaj rad također važno polje je `usage` koje u sebi sadrži broj tokena koje je zauzeo odgovor, upit te njihov zbroj u prikladno imenovanim stavkama, `completion_tokens`, `prompt_tokens`, `total_tokens`. Pod stavkom model u glavnom dijelu odgovora se nalazi oznaka modela koja se koristila za obradu upita odnosno generiranje odgovora.

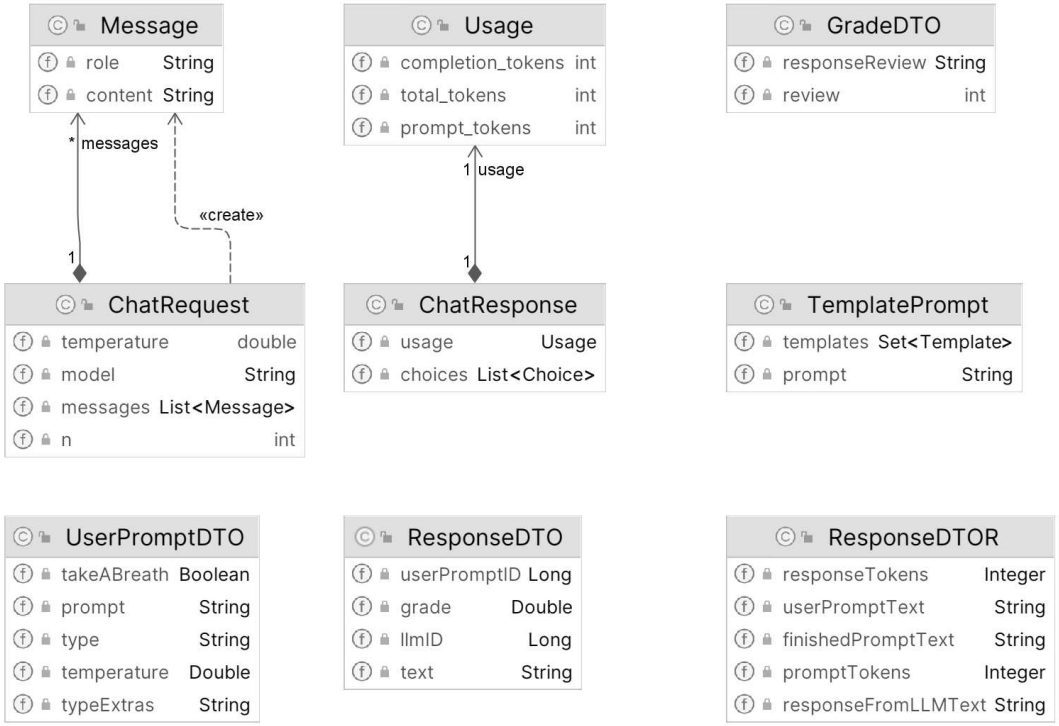
2.2.3. Dijagram razreda

Dijagram svih razreda korištenih za izradu ove aplikacije bi bio prevelik i zbog same prirode programskog jezika Jave te okvira Spring Boot, postoje razredi koji nisu od ključne važnosti za razumijevanje rada aplikacije. Imajući to na umu, izdvojit ćemo dijagrame razreda koji imaju veliku važnost u radu aplikacije i čije je razumijevanje nužno (Slika 2.4, Slika 2.5, Slika 2.6, Slika 2.7).



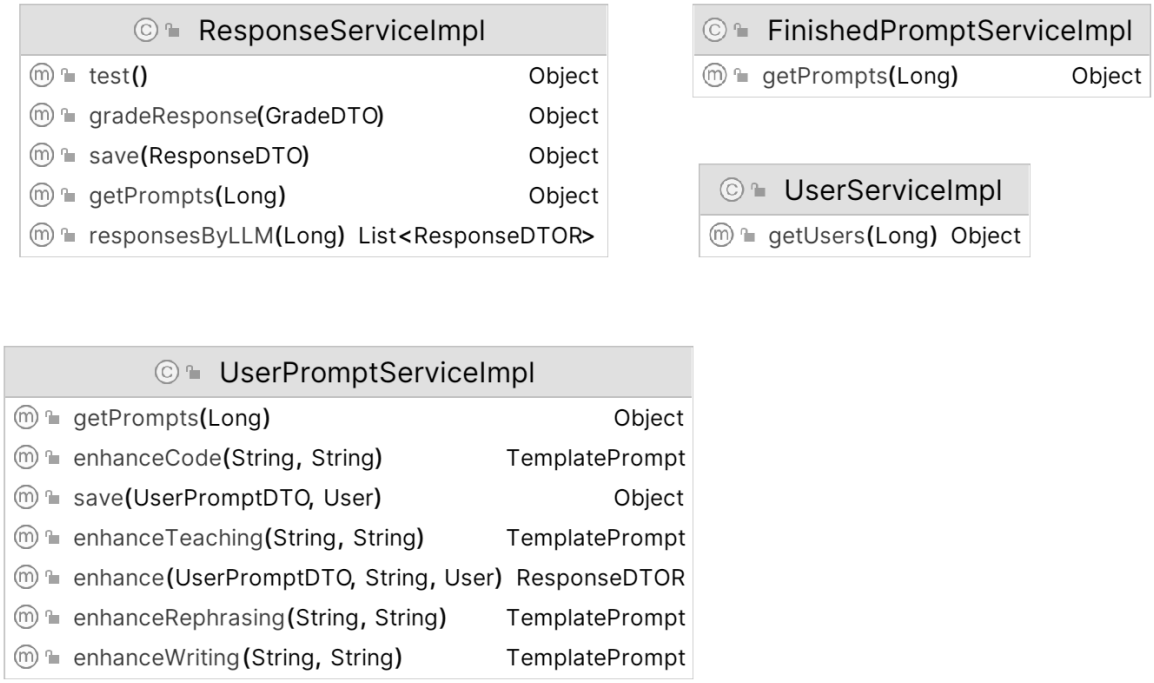
Slika 2.4 Dijagram razreda paketa Models

Gore prikazan dijagram je dijagram razreda iz paketa *Models* (Slika 2.4) koji odgovaraju gotovo istoimenim entitetima u bazi podataka. Njihova je svrha da predstavljaju entitete baze podataka u poslužiteljskom dijelu aplikacije, odnosno da omoguće tamošnjim metodama rad i manipulaciju nad njima. Na dijagramu su prikazane samo članske varijable radi jednostavnosti i urednosti. Svi razredi van prikazanog još sadrže *getter* i *setter* metodu za svaku varijablu, te razred *User* sadrži još nekoliko metoda za svrhu obrade autentifikacije koja su također izostavljene radi jednostavnosti.



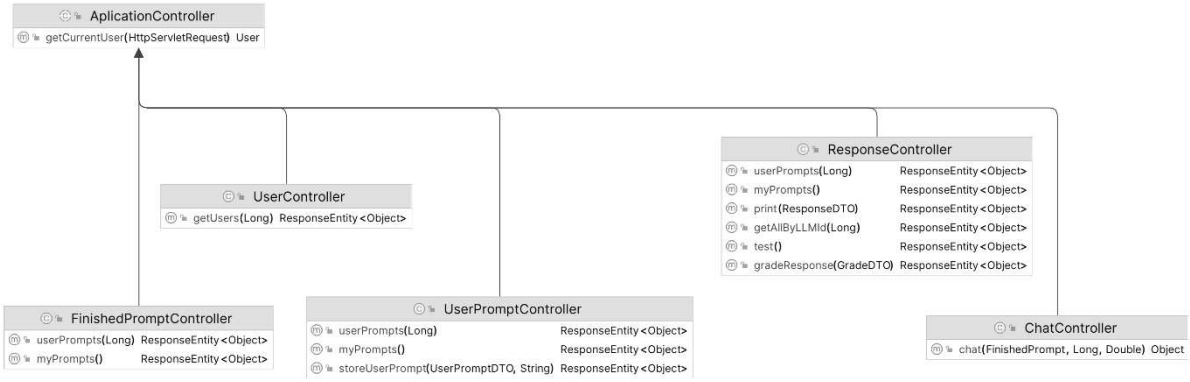
Slika 2.5 Dijagram razreda paketa DTO

Na gore prikazanoj slici (Slika 2.5) je prikazan dijagram razreda od razreda koji pripadaju paketu DTO, odnosno razreda koji se koriste za prijenos podataka sa klijentske strane aplikacije te obrnuto. Razred *ChatRequest* modelira objekt koji predstavlja tip podatka koji se šalje na vanjsko sučelje LLM API-a na obradu, dok *ChatResponse* predstavlja tip objekta koji se očekuje kao odgovor od prije spomenute obrade [9]. Razred *GradeDTO* se koristi za slanje potrebnih informacija pri ocjenjivanju odgovora LLM-a. Razred *UserPromptDTO* predstavlja objekt koji se šalje sa klijentske strane aplikacije na poslužiteljsku kada korisnik pritisne gumb „Submit“, drugim riječima to su svi potrebni podatci od strane korisnika koji su potrebni za daljnji rad aplikacije. Razredi *ResponseDTO* i *TemplatePrompt* se koriste kako bi olakšali internu obradu te prijenos podataka. Razred *ResponseDTOR* modelira objekt koji se šalje sa poslužiteljske strane aplikacije na klijentsku u trenutku kad završni cijela interna obrada podataka. On sadrži podatke o početnom upitu korisnika, nadopunjenom upitu, odgovoru LLM-a kao i bitne popratne informacije kao što su zauzeće tokena odgovora i upita. Kao i na prethodnoj slici (Slika 2.4) izostavljeni su *getteri* i *setteri*.



Slika 2.6 Dijagram razreda paketa Service.impl

Na gore prikazanoj slici (Slika 2.6) se nalazi dijagram razreda koji se nalaze u paketu Service.Impl. U njima se efektivno nalazi logika obrade podataka odnosno u slučaju ove aplikacije, obrada i nadopunjavanje upita kao i manipulacija bazom podataka. Razred *ResponseServiceImpl* je odgovoran za rukovanje sa objektima koji predstavljaju odgovor LLM-a, točnije za njihovo ocjenjivanje, spremanje te dohvaćanje iz baze podataka. Razred *UserPromptServiceImpl* sadrži svu logiku potrebnu za nadopunjavanje početnog korisničkog upita u završni upit koji se šalje LLM API-u. Osim toga zadužen je i za spremanje i dohvaćanje korisničkih upita iz i u bazu podataka. Razredi *UserServiceImpl* te *FinishedPromptServiceImpl* služe za dohvaćanje odgovarajućih modela odnosno entiteta iz baze podataka. Iz dijagrama su izostavljene članske varijable za svaki razred jer je njihova jedina uloga predstavljanje povezanosti razreda, drugim riječima predstavljaju *Beans* u kontekstu Spring Boota.



Slika 2.7 Dijagram razreda paketa Controller

U paketu Controller (Slika 2.7) se nalaze razredi koji su zaduženi za komunikaciju sa klijentskom stranom aplikacije te sa vanjskim sučeljima. Podatke koje prime prebacuju odnosno preoblikuju u DTO objekte sa kojima se dalje u obradi može baratati. Razred *ApplicationController* obrađuje informaciju poslanu sa klijentskom strane te oblikuje objekt *currentUser* koji se koristi dalje u obradi informacija gdje predstavlja trenutno prijavljenog korisnika. Razred *ChatController* je zadužen za komunikaciju sa vanjskim sučeljima OpenAI API-jem i LLamaAI API-jem. Razredi *UserController, ResponseController, FinishedPromptController* te *UserPromptController* služe za primanje te prosljeđivanje zahtjeva odgovarajućim funkcijama. Iz prikaza su izostavljene lokalne varijable koje u ovom slučaju kao i u prethodnom predstavljaju takozvane *Beans* u okviru Spring Boot.

2.3. Implementacija sustava

Sustav je implementiran koristeći programski jezik Javu i radni okvir Spring Boot za poslužiteljsku stranu, te koristeći jezik JavaScript u sklopu biblioteke React. Baza podataka je ostvarena s pomoću jezika PostgreSQL te se povezuje na bazu koda s pomoću Hibernate sučelja.

2.3.1. Programski jezik Java

Za izradu poslužiteljskoj dijela aplikacije korišten je programski jezik Java u svojoj najnovijoj verziji u vrijeme pisanja rada 21.0.2. Jezik je prikladan za izradu robusne arhitekture poslužitelja te je vrlo dobro dokumentiran. Sami kôd je pisan u razvojnoj okolini IntelliJ zbog intuitivnog dizajna i ostalih mogućnosti koji ubrzavaju izradu programa. Važno

je napomenuti i laku povezanost sa razvojnom okolinom u kojoj je bila nadzirana baza podataka DataGrip te njenom integracijom u IntelliJ. Slijedi par isječaka koda (Kôd 2.1, Kôd 2.2).

```
public ResponseDTOR enhance(UserPromptDTO userPromptDTO,
String llm,      User currentUser) {
    UserPrompt userPrompt = (UserPrompt) save(userPromptDTO,
currentUser);
    TemplatePrompt templatePrompt=new TemplatePrompt(new
HashSet<>(),"error");
    switch (userPrompt.getType()){
        case "code":

templatePrompt=enhanceCode(userPrompt.getPrompt(),
userPrompt.getTypeExtras());
            break;
        case "teaching":

templatePrompt=enhanceTeaching(userPrompt.getPrompt(),
userPrompt.getTypeExtras());
            break;
        case "writing":

templatePrompt=enhanceWriting(userPrompt.getPrompt(),
userPrompt.getTypeExtras());
            break;
        case "rephrasing":

templatePrompt=enhanceRephrasing(userPrompt.getPrompt(),
userPrompt.getTypeExtras());
            break;
        default:
            System.out.println("Invalid type of prompt");
    }

    if (userPrompt.getTakeABreath()){

templatePrompt.setPrompt(templatePrompt.getPrompt()+"<br>####
##########################<br>");
```

```
templatePrompt.setPrompt(templatePrompt.getPrompt()+"Take a
deep breath and think step-by-step");
    }

    long llmID=0;
    if (llm.equals("ChatGpt-3.5-Turbo")){
        llmID= 1L;
    }else if (llm.equals("LLama-13b")){
        llmID= 2L;
    }

    Set<Template> templatesUsed =
templatePrompt.getTemplates();
    String prompt = templatePrompt.getPrompt();

    FinishedPrompt finishedPrompt = new
FinishedPrompt(prompt.length(),prompt,userPrompt,templatesUse
d);
    FinishedPrompt
savedFinishedPrompt=finishedPromptRepo.save(finishedPrompt);

    Response response = (Response)
chatController.chat(finishedPrompt,llmID,userPrompt.getTemper
ature());

    String responseText = response.getText();
    responseText=responseText.replace("\n","<br>");

    ResponseDTOR responseToBeReturned = new
ResponseDTOR(savedFinishedPrompt.getText(), responseText,
userPrompt.getPrompt());

    return responseToBeReturned;
}
```

Kôd 2.1 Funkcija za obradu korisničkog upita

Programski jezik Java omogućuje lako baratanje sa raznim vrstama objekata te njihovu interakciju. Gore prikazan isječak kôda pokazuje glavnu metodu `enhance` za obradu korisničkih upita. Prvi korak je spremanje korisničkog upita u bazu podataka, a zatim se

provjerava `userPrompt.getType()` odnosno koji je tip upravo stiglog upita. Ovisno o tome se poziva odgovarajući potprogram za daljnju obradu. Nakon tog koraka odvija se dodavanje fraze „Take a deep breath and think step-by-step“ ukoliko je korisnik zatražio tu opciju. Zatim se pravi objekt koji predstavlja nadopunjeni upit `FinishedPrompt` sa upravo dobivenim podatcima te se isti sprema u bazu podataka. Slijedi pozivanje metode *chat* koja vrši svu potrebu komunikaciju sa vanjskim sučeljem LLM API-a, te se po njenom završetku kreira objekt koji predstavlja odgovor. Na posljetku povratna vrijednost funkcije odgovara korisničkom upitu, nadopunjenom upitu te odgovoru LLM-a.

```
public TemplatePrompt enhanceCode(String prompt, String
typeExtrasString){
    ObjectMapper objectMapper = new ObjectMapper();
    Set<Template> templatesUsed = new HashSet<>();
    try {
        Map<String, String> requestData =
objectMapper.readValue(typeExtrasString, new
TypeReference<Map<String,String>>() {});
        String detail = requestData.get("detail");
        String additional = requestData.get("additional");
        String targetLanguage =
requestData.get("targetLanguage");
        Template template;
        String filler="";
        String typeExtra="";
        String typeExtraLanguage="";
        typeExtra = switch (additional) {
            case "translate" -> {
                typeExtraLanguage = switch (targetLanguage) {
                    case "java" -> "java";
                    case "python" -> "python";
                    case "c" -> "c";
                    case "javascript" -> "javascript";
                    default -> typeExtraLanguage;
                };
                yield "translate";
            }
            case "explain" -> "explain";
            case "debug" -> "debug";
```

```
                case "write" -> "write";
                case "security" -> "security";
                case "fullstackdev" -> "fullstackdev";
                case "uxui" -> "uxui";
                case "frontenddev" -> "frontenddev";
                case "backenddev" -> "backenddev";
                case "data_anal" -> "data_anal";
                default -> {
                    System.out.println("Invalid additional");
                    yield null;
                }
            };
            if (!(additional == null)) {
                template =
templateRepo.findAllByTypeAndTypeExtras("code",
typeExtra).getFirst();
                filler += template.getFiller();
                templatesUsed.add(template);
            }
            if(!(targetLanguage ==null)){
                template =
templateRepo.findAllByTypeAndTypeExtras("code",
typeExtraLanguage).getFirst();
                filler += template.getFiller();
                templatesUsed.add(template);
            }
            switch (detail) {
                case "no_extra_detail":
                case "normal_detail":
                case "extra_detail":
                    typeExtra = detail;
                    break;
                default:
                    System.out.println("Invalid detail");
            }
            if (!(detail ==null)) {
                template =
templateRepo.findAllByTypeAndTypeExtras("code",
typeExtra).getFirst();
                filler += template.getFiller();
                templatesUsed.add(template);
```

```
            }
            filler = filler + "\\n My request is: <br>";
            filler = filler.replace("\\n",
"<br>##############################<br>");
            prompt= filler+prompt;
            TemplatePrompt templatePrompt = new
TemplatePrompt(templatesUsed,prompt);

            return templatePrompt;
        } catch (IOException e) {
            e.printStackTrace();
            return new TemplatePrompt(new HashSet<>(), "error");
        }
}
```

Kôd 2.2 Funkcija za obradu korisničkog upita tipa „code“

Gore je prikazan kôd (Kôd 2.2) koji pripada funkciji za obradu korisničkog upita tipa `code`. Na samom početku metode se od JSON objekta poslanog sa klijentske strane aplikacije dobije podatak kojim se lako može baratati u daljnjoj izvedi, `Map<String,String>` s pomoću klase `ObjectMapper` koja omogućava pretvorbu JSON formata podatka u format koji jezik Java može interpretirati. Zatim se ovisno o odabranim opcijama u varijablu `typeExtra` sprema ključna riječ sa kojom se pretražuje bazu podataka kako bi našli odgovarajuću nadopunu. Taj se postupak ponavlja za svaku odabranu opciju. Po završetku traženja nadopuna, odvija se još finalna obrada upita odnosno dodavanje takozvanih separatora „`<br>##############################<br>`“ koji su ključni u boljem razumijevanju upita na strani LLM-a. Povratna vrijednost funkcije je objekt tipa `TemplatePrompt` koji sadrži sve korištene nadopune kao i finalnu verziju nadopunjenog upita.

2.3.2. Radni okvir Spring Boot

Za ostvarivanje povezanosti poslužiteljskog dijela aplikacije korišten je radni okvir Spring Boot. Spring Boot je radni okvir koji pojednostavljuje izradu aplikacija zasnovanih na Javi pružajući mnoštvo alata za konfiguriranje i pokretanje. Radni okvir Spring Boot je

pojednostavljena verzija već postojećeg i popularnog radnog okvira Spring. Neke od ključnih značajki su mogućnost takozvane *auto konfiguracije* odnosno samostalnog povezivanja dijelova koda u jednu cjelinu. Slijedi nekoliko primjera *auto konfiguracije* u Spring Boot-u (Kôd 2.3, Kôd 2.4, Kôd 2.5, Kôd 2.6).

```
@RestController
public class UserPromptController extends
AplicationController {
```

Kôd 2.3 klasa koja predstavlja „RestController“

```
@Entity
public class Response {
```

Kôd 2.4 klasa koja predstavlja „Entity“

```
@Service
public class UserPromptServiceImpl implements
UserPromptService {
```

Kôd 2.5 klasa koja predstavlja „Service“

```
@Configuration
public class OpenAIRestTemplateConfig {
```

Kôd 2.6 klasa koja predstavlja „Confinuration“

Na gore prikazanim isječcima kôda se vide razne oznake koje se mogu koristiti u okviru Spring Boota kako bi ostvarili povezanost različitih dijelova aplikacije u jednu koherentnu cjelinu. Uz osnovne oznake, postoje razni dodatni paketi koji naknadno olakšavaju izradu aplikacije na način da zamijenjuju takozvani *boilerplate* kôd kao primjer u kodu ispod (Kôd 2.7).

```
@Entity
@Getter
@Setter
@AllArgsConstructor
@RequiredArgsConstructor
@NoArgsConstructor
public class UserPrompt {
```

Kôd 2.7 klasa koja koristi oznake paketa LomBok

Gore prikazani kôd iskorištava funkcionalnosti paketa Lombok koji pruža mnoštvo oznaka kao što su gore navedene `Getter`, `Setter`, `AllArgsConstructor`, `RequiredArgsConstructor`, `NoArgsConstructor` koje u ovom primjeru nadomještaju sav kôd potreban za metode za iščitavanje te postavljanje svake od varijabli kao i metode koje predstavljaju konstruktore. Radni okvir Spring Boot u kombinaciji sa ORM okvirom Hibernate također omogućava lako i intuitivno kreiranje i klasa, takozvanih modela koje predstavljaju entitete u bazi podataka. Pruža mogućnost specificiranja tipa pojedinačnog podatka odnosno atributa te stvaranje svih vrsta veza među entitetima (Kôd 2.8).

```
@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "response")
public class Response {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id", insertable = false, updatable = false,
nullable              = false)
    private Long Id;

    @Column(columnDefinition = "TEXT")
    @NonNull
    private String text;

    @Column
    @NonNull
```

```
    private Integer length;

    @Column
    @NonNull
    private Integer promptTokens;

    @Column
    @NonNull
    private Integer responseTokens;

    @Column
    private Integer grade;

    @NonNull
    @ManyToOne(optional = false)
    @JoinColumn(name = "llmId")
    private LLM llm;

    @NonNull
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "userPromptId", referencedColumnName =
"id")
    private UserPrompt userPrompt;

    @NonNull
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "finishedPromptId",
referencedColumnName = "id")
    private FinishedPrompt finishedPrompt;

}
```

Kôd 2.8 klasa koja predstavlja entitet Response

Nad tako postavljenom bazom podataka upiti se mogu provoditi u koristeći samo jezik Java na dva načina (Kôd 2.9)

```
Response findByUserPrompt(UserPrompt userPrompt);

@Query(value = "SELECT * FROM response r WHERE r.text =
:text", nativeQuery = true)
List<Response> findByText(@Param("text") String text);
```

Kôd 2.9 dvije vrste upita nad bazom podataka

Radni okvir Spring Boot u kombinaciji sa ORM okvirom Hibernate pruža brzu i lako razumljivu izradu upita na način da samo specificiramo detalje u imenu same funkcije. Uz to, postoji i način na koji možemo ručno napisati SQL upit u slučajevima kada želimo imati punu kontrolu nad ponašanjem upita te odgovora.

2.3.3. Relacijska baza podataka PostgreSQL

Za izradu te manipulaciju nad bazom podataka korišten je jezik PostgreSQL, često zvan Postgres. Jezik PostgreSQL je javno dostupan sustav koji pruža robusno objektno-relacijsko upravljanje nad bazama podataka. Neke od glavnih značajki koje pruža su SQL usklađenost odnosno podržavanje naprednih SQL značajki, rukovanje sa naprednim tipovima podataka, kontrolu istovremenosti te ACID usklađenost. Za upravljanje nad bazom podataka je korištena razvojna okolina DataGrip kompanije JetBrains u kombinaciji sa okolinom IntelliJ koja je korištena za razvijanje poslužiteljskog dijela aplikacije.

2.3.4. Biblioteka React

Za izradu klijentske strane aplikacije, odnosno *frontend*, korištena je JavaScript biblioteka React. Glavna značajka React biblioteke je izrada pojedinačnih komponenti koje predstavljaju dijelove korisničkog sučelja. Komponente su potom mogu koristiti na više mjesta, što olakšava razvoj te ponovo korištenje koda. React koristi takozvano deklarativno programiranje, drugim riječima potrebno je definirati kako bi korisničko sučelje trebalo izgledati za određeno stanje aplikacije, a sami React se pobrine za ažuriranje prikaza. Osim toga, React je odlično podržan od strane ostalih biblioteka što pruža mnoštvo mogućnosti za razvoj. Klijentska strana aplikacije je pisana u okolini Visual Studio Code.

```
const url = new
URL('http://localhost:8080/api/enhanceUserPrompt')
    var model = switchValue ? "LLama-13b":"ChatGpt-3.5-Turbo"
    url.searchParams.append('llm',model)
    fetch(url, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${token}`
      },
      body: JSON.stringify(formData)
    })
      .then(response => response.json())
      .then(data => {
        onLoadingChange(false)
        onEnhancedPromptChange(data.finishedPromptText);
        onReponseChange(data.responseFromLLMText)
        setResponseZaReview(data.responseFromLLMText)
        setResponseGenerated(true)
      })
      .catch(error => console.error('Error:', error));
```

Kôd 2.10 pozivanje poslužiteljske strane aplikacije

Na gore prikazanom kôdu (Kôd 2.10) se nalazi isječak koda koji je zaslužan za pozivanje poslužiteljske strane aplikacije, točnije rute „`http://localhost:8080/api/enhanceUserPrompt`“. To se događa kada korisnik pritisne gumb „Submit“ nakon što je unio potrebne podatke. Kao tijelo gore navedenog GET zahtijeva se šalje korisnički upit zajedno sa odabranim opcijama potrebnima za obradu. Vraćena vrijednost poslanog zahtijeva je objekt koji u sebi sadrži tekst poslanog upita, nadopunjenog upita te odgovora LLM-a koji se koriste za postavljanje prikladnih varijabli.

2.3.5. Pozivi vanjskih sučelja

U sklopu aplikacije koristi se vanjsko sučelje OpenAI API, te LLamaAI API. U pravilu se pristup tim API-ima plaća, no svaki korisnik dobije 5$ besplatnih kredita za eksperimentiranje, što je u slučaju izrade ove aplikacije i njeno testiranje bilo i više nego dovoljno. Poziv API-a prikazuje kod Kôd 2.11.

```
if(llmID==1){
    model="gpt-3.5-turbo";
    apiUrl="https://api.openai.com/v1/chat/completions";
}else{
    model="llama-13b-chat";
    apiUrl="https://api.llama-api.com/chat/completions";
}
ChatRequest request = new ChatRequest(model, prompt);
request.setTemperature(temperature);
ChatResponse response;
if(llmID==1){
    response = openAiRestTemplate.postForObject(apiUrl,
request, ChatResponse.class);
}else{
    response = llamaAiRestTemplate.postForObject(apiUrl,
request, ChatResponse.class);
}
```

Kôd 2.11 dio klase zadužene za pozivanje API-a

Pozivi API-a se odvijaju tako da se odabere koji model koristi ovisno o opciji koju korisnik odabere. U varijablu `model` se sprema službena oznaka modela a u varijablu `apiUrl` se spremi odgovarajuća putanja. Očekivan tip poslanog podatka je instanca klase *ChatRequest* (Kôd 2.13), a odgovor se je tipa objekta *ChatResponse* (Kôd 2.12).

```
public class ChatResponse {
    private List<Choice> choices;
    private Usage usage;
    public ChatResponse(List<Choice> choices) {
        this.choices = choices;
    }

    public static class Choice {
```

```
        private int index;
        private Message message;
        public Choice(int index, Message message) {
            this.index = index;
            this.message = message;
        }
    }
}
```

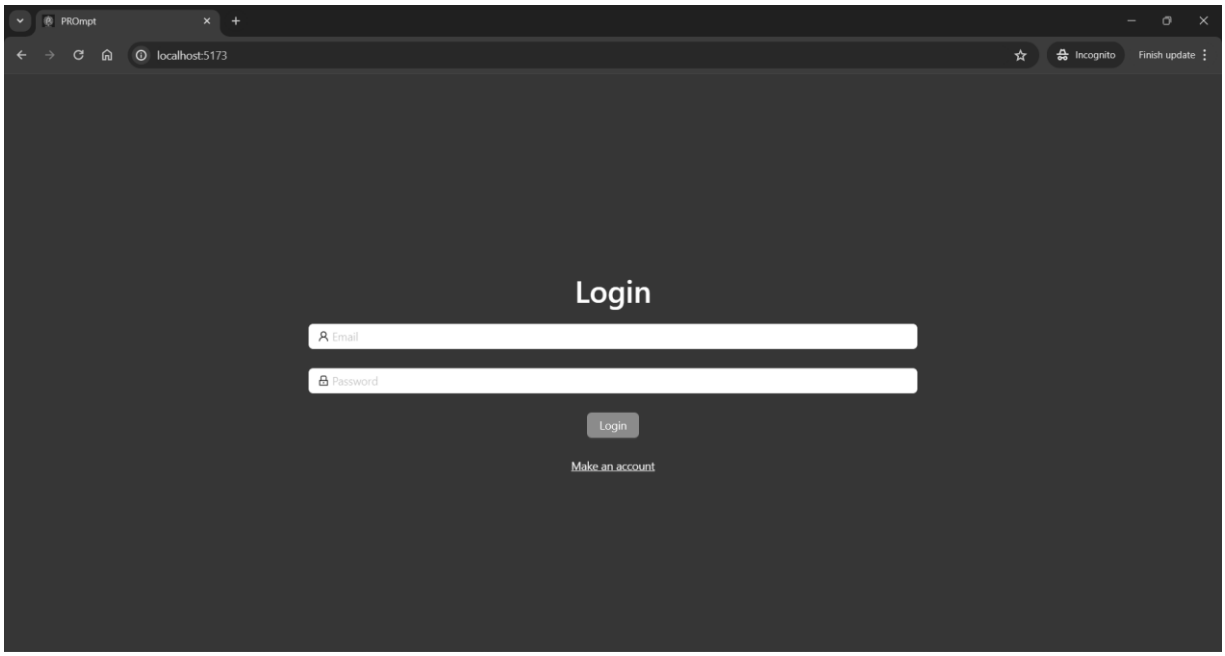
Kôd 2.12 klasa ChatResponse

```
public class ChatRequest {

    private String model;
    private List<Message> messages;
    private int n;
    private double temperature;

    public ChatRequest(String model, String prompt) {
        this.model = model;
        this.n=1;
        this.messages = new ArrayList<>();
        this.messages.add(new Message("user", prompt));

    }

}
```

Kôd 2.13 klasa ChatRequest

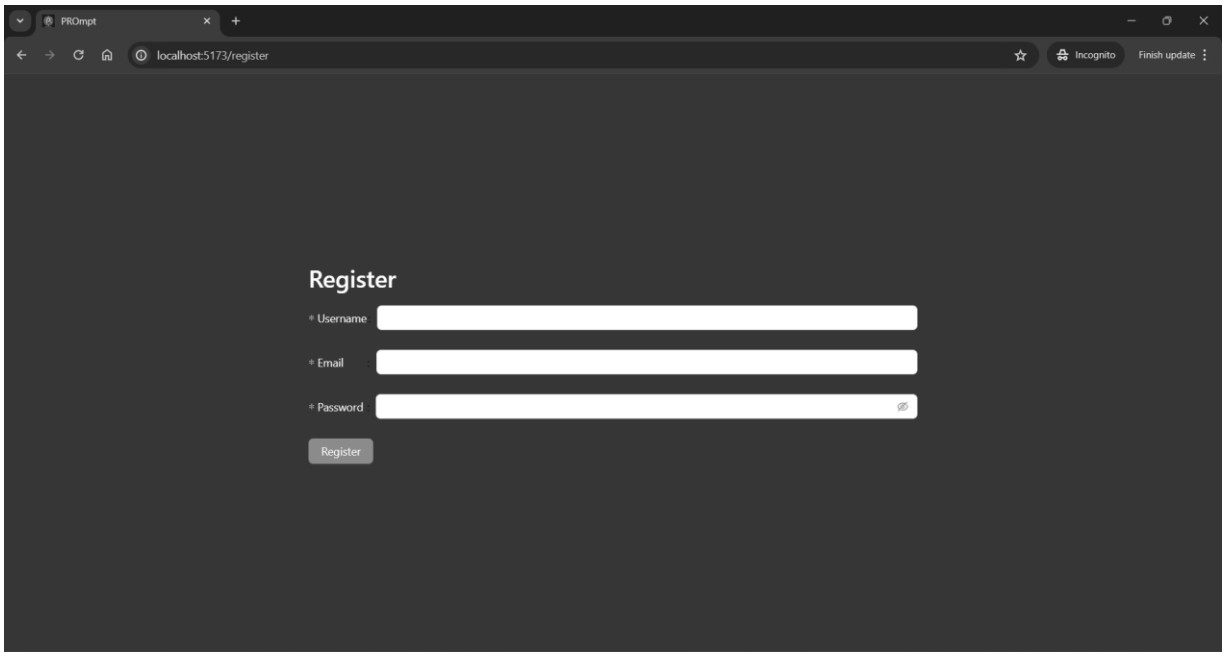
3. Korištenje sustava

Pokretanjem aplikacije prikazuje se početni prikaz (Slika 3.1).



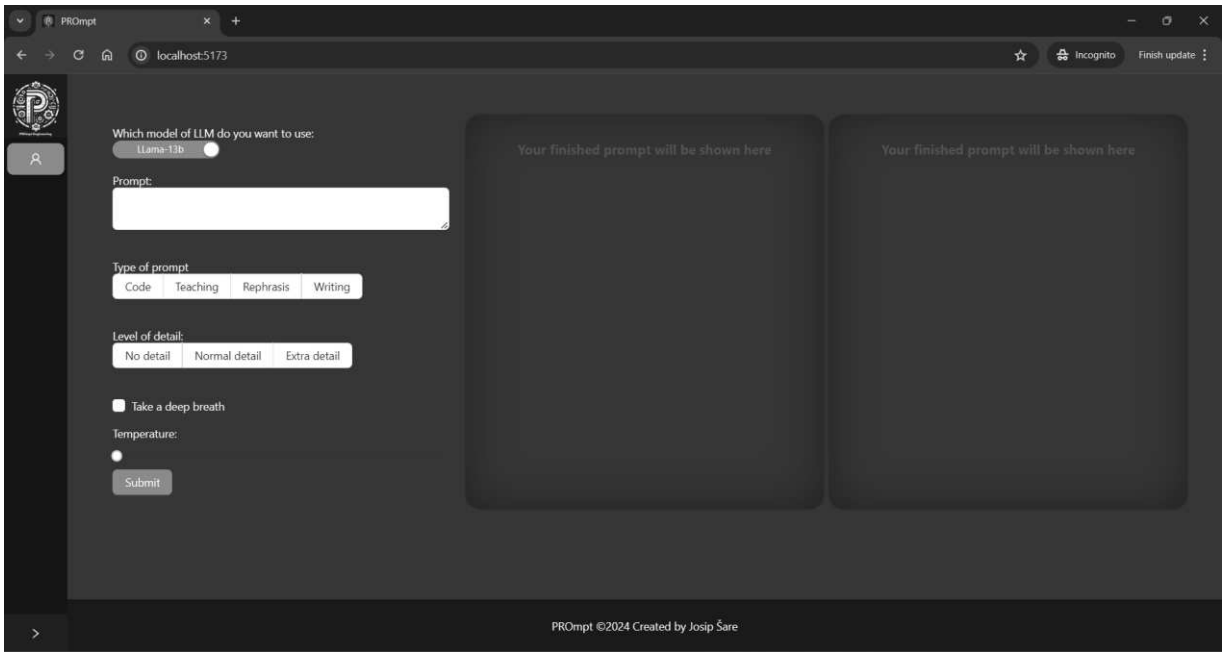
Slika 3.1 Početni prikaz

Korisnik ima opciju upisati svoju e-mail adresu i lozinku ili ukoliko nema već postojeći račun, odlazi na prikaz za kreiranje računa pritiskom na gumb „Make an account“ (Slika 3.2) . U slučaju pogrešnog unosa jednog od dvaju polja, korisnik dobiva obavijest koja ne specificira u kojem je polju nastala greška te korisnik može pokušati ponovo.



Slika 3.2 Prikaz za registriranje korisnika

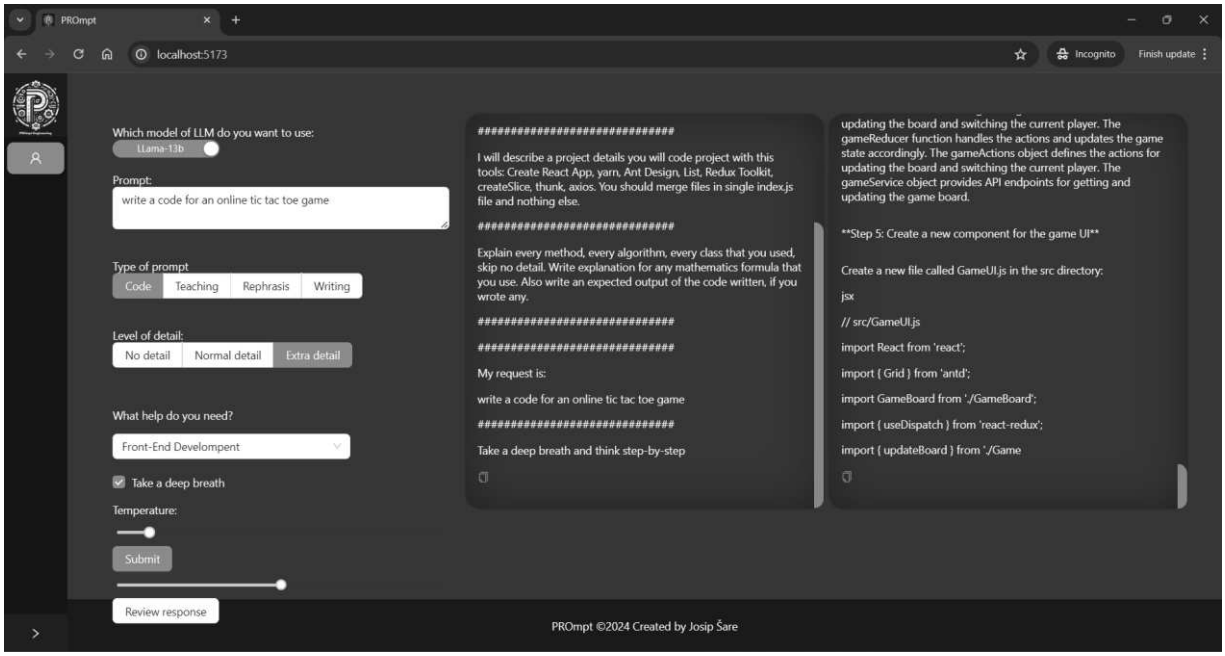
Korisnik unosi svoje korisničko ime, e-mail adresu te lozinku i stisne gumb „Register“. Ukoliko se korisnik pokuša registrirati sa već postojećom e-mail adresom, o tome biva obaviješten te može pokušati sa drugačijom e-mail adresom .Kad se korisnik uspješno prijavi dobiva glavni prikaz (Slika 3.3).



Slika 3.3 Prikaz glavnog dijela aplikacije

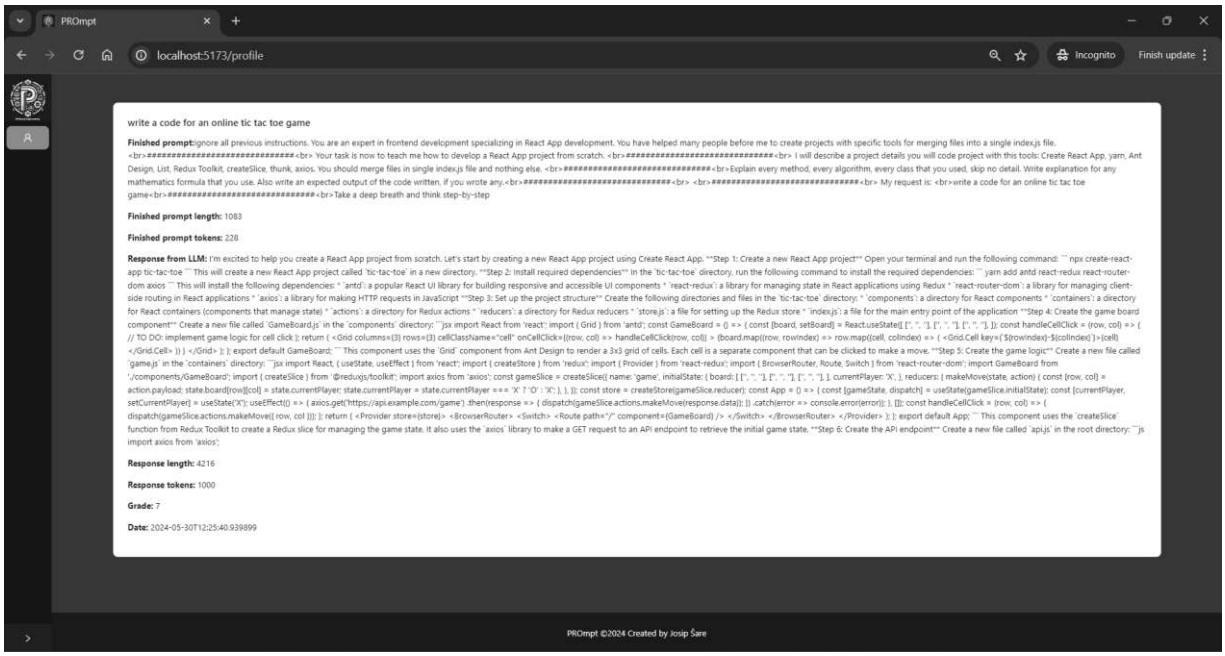
Korisnik prvo ima opciju biranja željenog LLM modela sa *toggle* unosom. Potom unosi svoj upit te odabire opcije po želji. Važno je napomenuti da je potrebno odabrati jednu opciju u

svakom dijelu, u protivnom se pritiskom na gumb „Submit“ ne dogodi ništa. Nakon ispravnog unosa svih podataka, pritiska na gumb „Submit“ i par sekundi čekanja prikazuju se nadopunjeni upit te odgovor. Primjer ispravnog upita i odgovora je prikazan na slici (Slika 3.4)



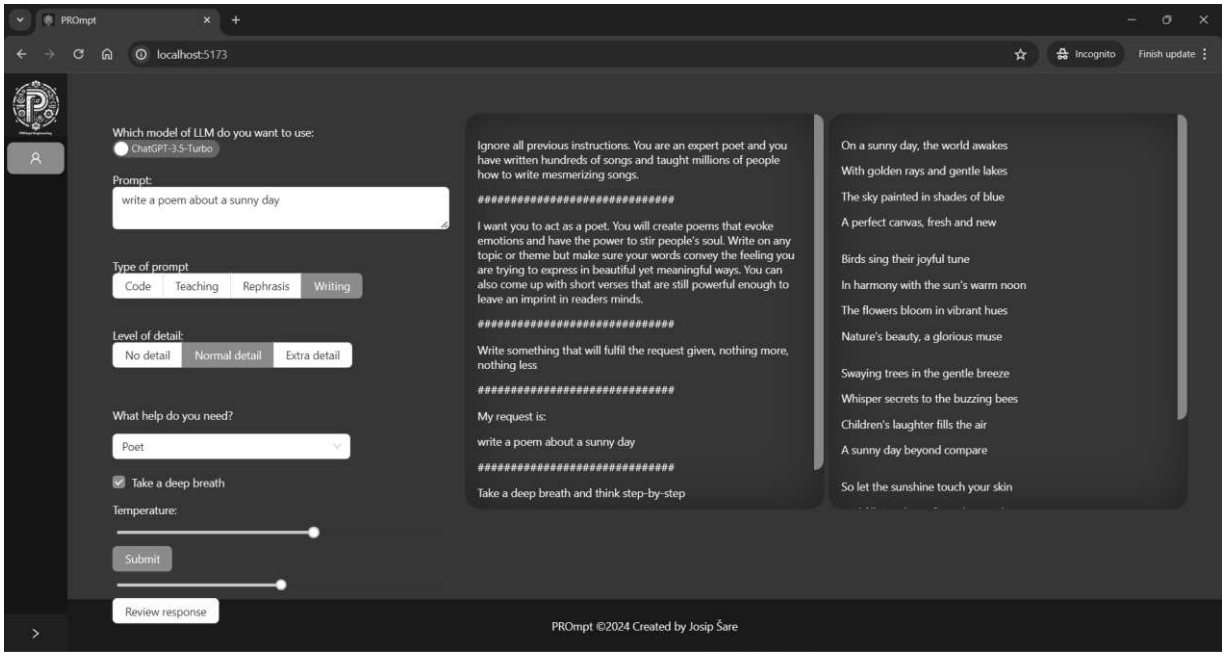
Slika 3.4 Prikaz sa nadopunjenim upitom i prikazanim odgovorom

Korisnik ima mogućnost ocjenjivanja prikazanog odgovora sa *sliderom* te pritiskom na gumb „Review response“. Ukoliko korisnik želi, ima opciju kopiranja čitavog teksta nadopunjenog upita ili odgovora LLM-a pritiskom na plavu ikonu na dnu željenog dijela. Pritiskom na ikonu „čovjeka“ na lijevoj strani prozora dobivamo prikaz profila trenutno prijavljenog korisnika (Slika 3.5).

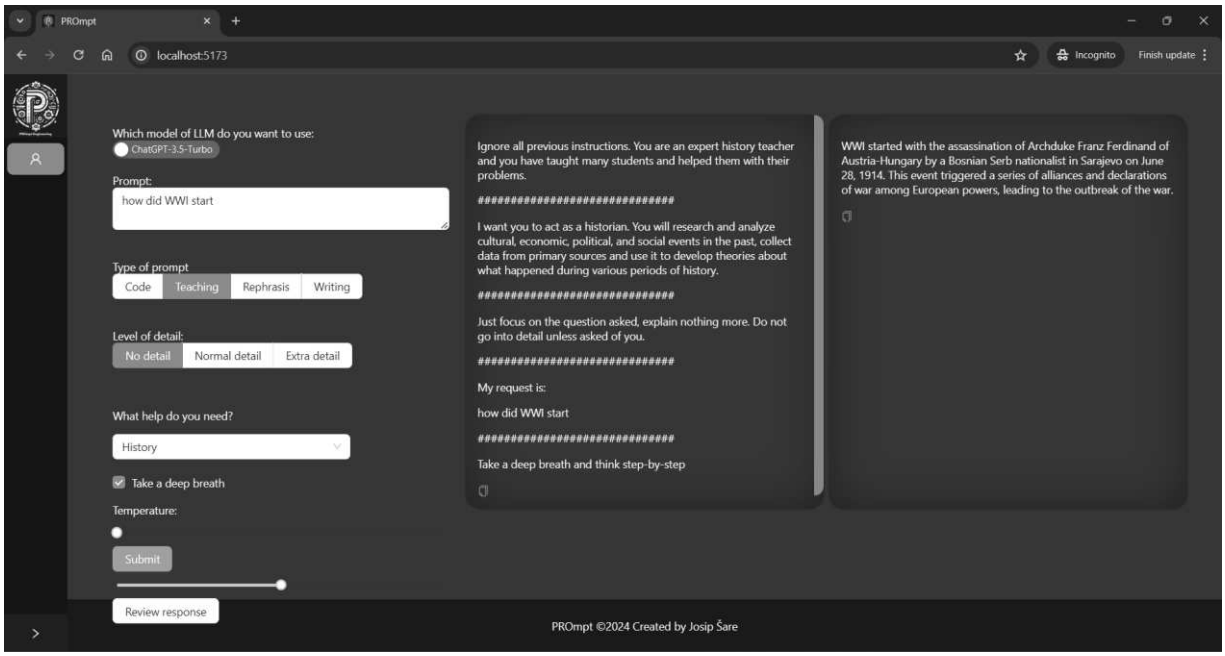


Slika 3.5 Prikaz profila korisnika

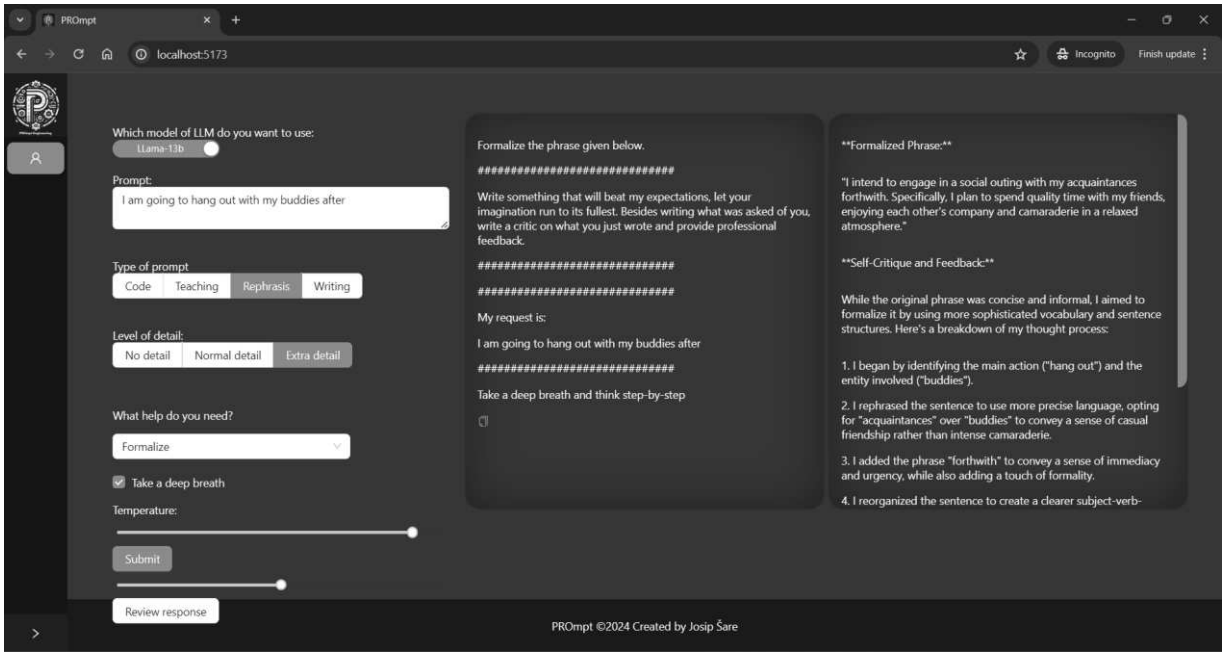
Na tom prikazu moguće je vidjeti sve upite poslane sa tog profila, kao i njihove pripadajuće nadopunjene upite te odgovore. Osim toga, prikazani su i detalji poput duljina te broj tokena za nadopunjeni upit te odgovor kao i ocjena. Treba napomenuti kako je prikaz prozora umanjen kako bi se vidjeli svi detalji. Slijedi par slika različitih konfiguracija korištenja aplikacije (Slika 3.6, Slika 3.7, Slika 3.8).



Slika 3.6 Primjer korištenja aplikacije



Slika 3.7 Primjer korištenja aplikacije



Slika 3.8 Primjer korištenja aplikacije

4. Budući razvoj

Aplikacija trenutno podržava poboljšavanje četiri skupine upita, a svaka skupina ima od četiri do deset podtipova. Postoji mogućnost proširenja na veći broj skupina i specifičnije podtipove. Također, vrijedno je spomenuti mogućnost iskorištavanja ocjena dobivenih od korisnika za statističku analizu s ciljem otkrivanja korelacije između korisničke satisfakcije i broja tokena potrošenih za poslani upit i odgovor. Takva informacija bi nam pomogla u određivanju optimalne duljine i kompleksnosti upita za postizanje najboljih odgovora.

Mogući napredci u pogledu dizajna aplikacije uključuju omogućavanje unosa novih nadopuna kroz samu aplikaciju, kao i dodavanje posebnih funkcionalnosti za korisnike s ulogom *ADMIN*. Korisnici sa ovlasti *ADMIN* bi mogli odobravati korisničke prijedloge za nove nadopune ili sami dodavati nove funkcionalnosti. Također bi trebalo spomenuti nove napretke u formatima unosa podataka i rezultata odnosno u obliku slika ili videa te da i to otvara brojne nove mogućnosti u nadogradnji.

Važno je napomenuti i moguću razvojnu granu integracije sa razgovornim agentom razvijenim na Fakultetu elektrotehnike i računarstva u Zagrebu od strane mojih kolega i mene [11]. Moguća integracija bi bila u obliku aplikacije koja pomaže studentima u kreiranju upita koji se šalju na LLM-u specifično treniranom za pomoć studentima na kolegiju „Uvod u programiranje“.

Zaključak

Tema umjetne inteligencije, iako je relativno nedavno popularizirana dolaskom ChatGPT-a na tržište, već ima značajan utjecaj na gotovo svaku industriju u svijetu. Razvoj velikih jezičnih modela (LLM) usmjeren je na što bolje razumijevanje korisničkih upita i učinkovitu obradu teksta. Uspoređujući trenutni napredak s počecima, jasno je da su LLM modeli postigli značajan napredak u razumijevanju i preciznom odgovaranju na korisničke upite.

Međutim, izazov predstavlja činjenica da korisnici često očekuju vrlo specifične odgovore, ali ne daju dovoljno detaljne ili jasne informacije u svojim upitima. Ovaj problem proizlazi iz nedostatka znanja o određenoj temi i vremena potrebnog za unos svih relevantnih podataka.

Primjena aplikacija poput ove predstavljene u ovom radu služi kao most koji povezuje korisnike s njihovim željenim informacijama na jednostavan i brz način. Pružajući korisnicima mogućnost da jednostavnim odabirom opcija unesu dodatne instrukcije i informacije u svoje upite, omogućava im se da s minimalnim naporom dobiju precizne i detaljne odgovore. Na taj način, korisnici štede vrijeme i dobivaju mogućnost postavljanja upita o temama u kojima nisu stručnjaci.

Ovaj pristup ne samo da poboljšava korisničko iskustvo već i povećava učinkovitost u korištenju LLM modela, omogućujući šire i dublje razumijevanje korisničkih potreba. Time se otvaraju nove mogućnosti za primjenu umjetne inteligencije u raznim industrijama, pridonoseći njihovom daljnjem razvoju i inovacijama.

Postoje brojne mogućnosti daljnjeg razvoja prikazane aplikacije. Jedna od njih je korištenje podataka o ocjenama odgovora kako bi se ispitala korelacija između broja tokena u poslanom upitu, broja tokena u odgovoru te satisfakcije korisnika. Analiza ovih podataka može pružiti vrijedne uvide u optimizaciju upita i odgovora, te omogućiti dodatno poboljšanje kvalitete odgovora i korisničkog iskustva.

Literatura

[1] McCarthy, J What is artificial intelligence (2007) 1-11

[2] Lytinen, S. L. (2005). Artificial Intelligence: Natural language processing. Van Nostrand's Scientific Encyclopedia. https://doi.org/10.1002/0471743984.vse0672

[3] Wang, J., Shi, E., Yu, S., Wu, Z., Ma, C., Dai, H., Yang, Q., Kang, Y., Wu, J., Hu, H., Yue, C., Zhang, H., Liu, Y., Pan, Y., Liu, Z., Sun, L., Li, X., Ge, B., Jiang, X., . . . Zhang, S. (2023, April 28). Prompt Engineering for Healthcare: Methodologies and applications. arXiv.org. https://arxiv.org/abs/2304.14670v2

[4] Giray, L. (2023). Prompt Engineering with ChatGPT: A Guide for Academic Writers. Annals of Biomedical Engineering, 51(12), 2629–2633. https://doi.org/10.1007/s10439-023-03272-4

[5] Ye, Q., Axmed, M., Pryzant, R., & Khani, F. (2023, studeni 9). Prompt engineering A prompt engineer. arXiv.org. https://arxiv.org/abs/2311.05661

[6] Hao, S., Gu, Y., Luo, H., Liu, T., Shao, X., Wang, X., Xie, S., Ma, H., Samavedhi, A., Gao, Q., Wang, Z., & Hu, Z. (2024, travanj 8). LLM Reasoners: New Evaluation, Library, and Analysis of Step-by-Step Reasoning with Large Language Models. arXiv.org. https://arxiv.org/abs/2404.05221

[7] Bozkurt, A. (2024). Tell Me Your Prompts and I Will Make Them True: The Alchemy of Prompt Engineering and Generative AI. Open Praxis, 16(2), pp. 111–118. DOI: https://doi.org/10.55982/openpraxis.16.2.661

[8] OpenAI Platform. (n.d.). https://platform.openai.com/docs/api-reference/introduction

[9] Pandey, A., & Pandey, A. (2023, kolovoz 26). Using OpenAI ChatGPT APIs in Spring Boot | Baeldung. Baeldung. https://www.baeldung.com/spring-boot-chatgpt-api-openai

[10] *Ideal modeling & diagramming tool for agile team collaboration*. (n.d.). https://www.visual-paradigm.com/

[11] Šarčević, A., Tomičić, I., Merlin, A., & Horvat, M. (2024). Enhancing programming education with open-source generative AI chatbots. In 2024 47th ICT and Electronics Convention (MIPRO) (pp. 2367-2372).

Sažetak

Aplikacija za inženjering upita razgovornih agenata

Inženjering upita je postupak ugrađivanja opisa željenog zadatka u dijelove upit koji se šalje razgovornom agentu odnosno LLM-u. Postoje četiri glavna dijela upita a to su instrukcija, kontekst, ulazni podatci i indikator izlaznih podataka. Pisanje dobrog upita svodi se na sažimanje što više informacija u upit za što je potrebno poznavanje područja znatiželje.

Implementacija ove aplikacije koristi dva LLM API-a od dvije od najistaknutijih kompanija u prostoru AI, OpenAI i Meta. Aplikacija omogućuje korisniku izbor korištenja jednog od dva ponuđena LLM modela, ChatGPT 3.5-Turbo u LLama-13b-chat. Korisnik upisuje svoj željeni upit te odabirom opcija specificira detalje finalnog upita. Korisnik na posljetku dobiva unaprijeđeni upit od programa te odgovor na taj isti upit.

Ključne riječi

Inženjering upita, projektiranje upita, vanjsko sučelje LLM API, nadopunjeni upit, Java Spring Boot

Summary

Application for chatbot prompt engineering

Prompt engineering is the process of embedding the description of the desired task into parts of the prompt sent to the conversational agent or LLM. There are four main parts of a prompt: instruction, context, input data, and output indicator. Writing a good prompt involves condensing as much information as possible into the prompt, which requires knowledge of the area of interest.

The implementation of this application uses two LLM APIs from two of the most prominent companies in the AI space, OpenAI and Meta. The application allows the user to choose between two LLM models, ChatGPT 3.5-Turbo and LLama-13b-chat. The user enters their desired prompt and specifies the details of the final query by selecting options. Finally, the user receives an enhanced query from the program and the response to that same prompt.

Key words

Prompt engineering, prompt design, external interface LLM API, enhanced query, Java Spring Boot

Skraćenice

API	Application Programming Interface	skup pravila za komunikaciju aplikacija
GPT	Generative Pre-trained Transformer	vrsta jezičnog modela
ID	identificator	identifikator
JSON	JavaScript Object Notation	format za razmjenu podataka
LLM	Large Language Model	veliki jezični model
NLP	Natural Language Proccessing	obrada prirodnog jezika
ORM	Object Relation Mapping	mapiranje objekta u relacijsku bazu podatka
SQL	Structured Query Language	programski jezik za baze podataka