UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET

IZVEŠTAJ PROJEKTA

SummaQ

Master akademske studije

Studijski program: računarstvo i informatika

Modul: softversko inženjerstvo

Predmet: Duboko učenje

Student:

Milan Lukić, br. ind. 1649

Profesor:

Prof. Aleksandar Milosavljević

Niš, oktobar 2025. godine

Sadržaj

1. Uvod

Razvoj savremenih alata za obradu prirodnog jezika (NLP – *Natural Language Processing*) doživeo je izuzetno ubrzan napredak poslednjih godina, naročito zahvaljujući integraciji dubokog učenja (*Deep Learning*) u ovu oblast. NLP predstavlja granu veštačke inteligencije koja se bavi interakcijom između računara i ljudskog jezika, omogućavajući računarima da analiziraju, razumeju i generišu prirodan tekst. Tradicionalni NLP pristupi oslanjali su se na ručno definisane jezičke strukture i pravila, dok savremeni sistemi koriste **neuronske mreže** i **modelovanje sekvenci**, čime postižu znatno bolje rezultate u složenim zadacima poput prevođenja, sažimanja, analize sentimenta i odgovaranja na pitanja.

Pojavom **velikih jezičkih modela (LLM – Large Language Models)**, poput GPT, Mistral, Llama i drugih, oblast obrade jezika doživela je revolucionarni pomak. Ovi modeli obučeni su na ogromnim količinama tekstualnih podataka i poseduju sposobnost generalizacije znanja, što im omogućava da odgovaraju na pitanja, sumiraju tekst, pišu eseje ili generišu programski kod, sve u okviru jednog jedinstvenog sistema.
Ključni koncept LLM modela zasniva se na **transformerskoj arhitekturi**, koja koristi mehanizam pažnje (*attention mechanism*) kako bi model mogao da „razume" odnose između reči unutar rečenice i između različitih delova teksta.

Jedna od najvećih prednosti LLM-ova je njihova **višenamenska primena** – isti model može da rešava širok spektar zadataka, uz minimalno prilagođavanje (*fine-tuning*) ili čak putem jednostavnih instrukcija (*prompt engineering*).
Ipak, većina postojećih rešenja oslanja se na komercijalne API servise (npr. OpenAI, Anthropic, Google), što može biti problematično u pogledu zaštite podataka, troškova i dostupnosti.
Zbog toga se poslednjih godina sve više pažnje posvećuje **lokalnom izvršavanju modela**, odnosno pokretanju LLM-ova direktno na korisničkim računarima ili serverima, bez potrebe za eksternom komunikacijom.

U tom kontekstu razvijena je aplikacija **„SummaQ"**, čiji je cilj omogućavanje efikasne obrade i analize tekstualnih dokumenata korišćenjem lokalnih modela. Aplikacija kombinuje NLP tehnike i LLM arhitekturu kako bi korisnicima omogućila da:

- automatski sažmu duge tekstualne sadržaje,
- izdvoje ključne informacije iz dokumenata,
- postavljaju pitanja o sadržaju i dobiju kontekstualno tačne odgovore.

Ovim pristupom aplikacija demonstrira praktičnu primenu velikih jezičkih modela u stvaranju inteligentnih sistema za podršku učenju, istraživanju i radu sa dokumentima, uz očuvanje privatnosti i visok nivo kontrole nad procesom obrade podataka.

2. Pregled arhitekture i implementacije

Arhitektura aplikacije zasnovana je na principima modularnog dizajna i jasnog razdvajanja slojeva funkcionalnosti.
Sistem je projektovan kao web aplikacija sa klijent-server arhitekturom, pri čemu se komunikacija između frontenda i backenda odvija putem standardizovanih HTTP REST servisa.
Ovakva arhitektura obezbeđuje visoku fleksibilnost, jednostavno održavanje i mogućnost nezavisnog razvoja i testiranja pojedinačnih delova sistema.

Sa stanovišta korisnika, aplikacija predstavlja alat za unos teksta ili dokumenata i automatsko generisanje sažetaka uz opcionalno postavljanje pitanja u vezi sa unetim sadržajem.
Sa stanovišta implementacije, sistem funkcioniše kao niz jasno definisanih koraka:

- **Unos i obrada tekstualnog sadržaja** (frontend sloj).
- **Slanje zahteva backend servisu** putem HTTP POST metode.
- **Procesiranje teksta i komunikacija sa lokalnim jezičkim modelom** (backend sloj).
- **Generisanje rezultata i povratna komunikacija ka korisniku.**

U okviru projekta primenjeni su principi **mikroservisne i komponentno orijentisane arhitekture**, gde svaki deo ima jasno definisanu odgovornost. Frontend upravlja interakcijom i prikazom rezultata, backend obavlja logiku obrade podataka i interakciju sa modelima, dok sloj za NLP i LLM operacije predstavlja samostalnu celinu zaduženu za rad sa lokalno hostovanim modelima.

Sistem je razvijan iterativno, uz fokus na:

- performanse (efikasno rukovanje velikim tekstovima),
- stabilnost komunikacije između slojeva,
- jednostavnost korisničkog interfejsa,
- mogućnost lokalnog rada bez spoljašnjih API zavisnosti.

Na visokom nivou, arhitektura se može opisati kao **troslojna struktura**:

- **Prezentacioni sloj (frontend)** – interfejs za unos teksta i prikaz rezultata.
- **Aplikativni sloj (backend)** – poslovna logika, obrada teksta i orkestracija.
- **Model sloj (NLP/LLM engine)** – izvršavanje zadataka sumarizacije i odgovaranja na pitanja pomoću lokalnog modela.

Ovakav pristup obezbeđuje da svaka funkcionalnost bude izolovana, čime se sistem lako može nadograditi – na primer, dodavanjem novih formata datoteka, podrške za dodatne jezike ili zamene jezičkog modela novijom verzijom.

2.1. Opšti pregled sistema

Aplikacija implementirana je kao višeslojni sistem koji obezbeđuje jasan tok informacija od korisnika do modela i nazad.
U osnovi, sistem se sastoji iz tri glavne celine: frontend, backend i lokalni model (LLM sloj). Svaka od ovih komponenti ima jasno definisanu ulogu u okviru procesa obrade podataka i međusobno komunicira putem standardnih i jednostavnih protokola.

Na najvišem nivou, tok rada aplikacije može se opisati sledećim koracima:

Korisnički unos podataka – Korisnik unosi tekst direktno u aplikaciju ili učitava datoteku (.docx ili .txt). Frontend sloj je zadužen za prikaz i prenos sadržaja na backend.

Slanje zahteva backend servisu – Frontend koristi HTTP POST zahteve kako bi poslao podatke serveru zajedno sa parametrima (npr. procenat sažimanja, izbor formata – paragraf ili bullet points).

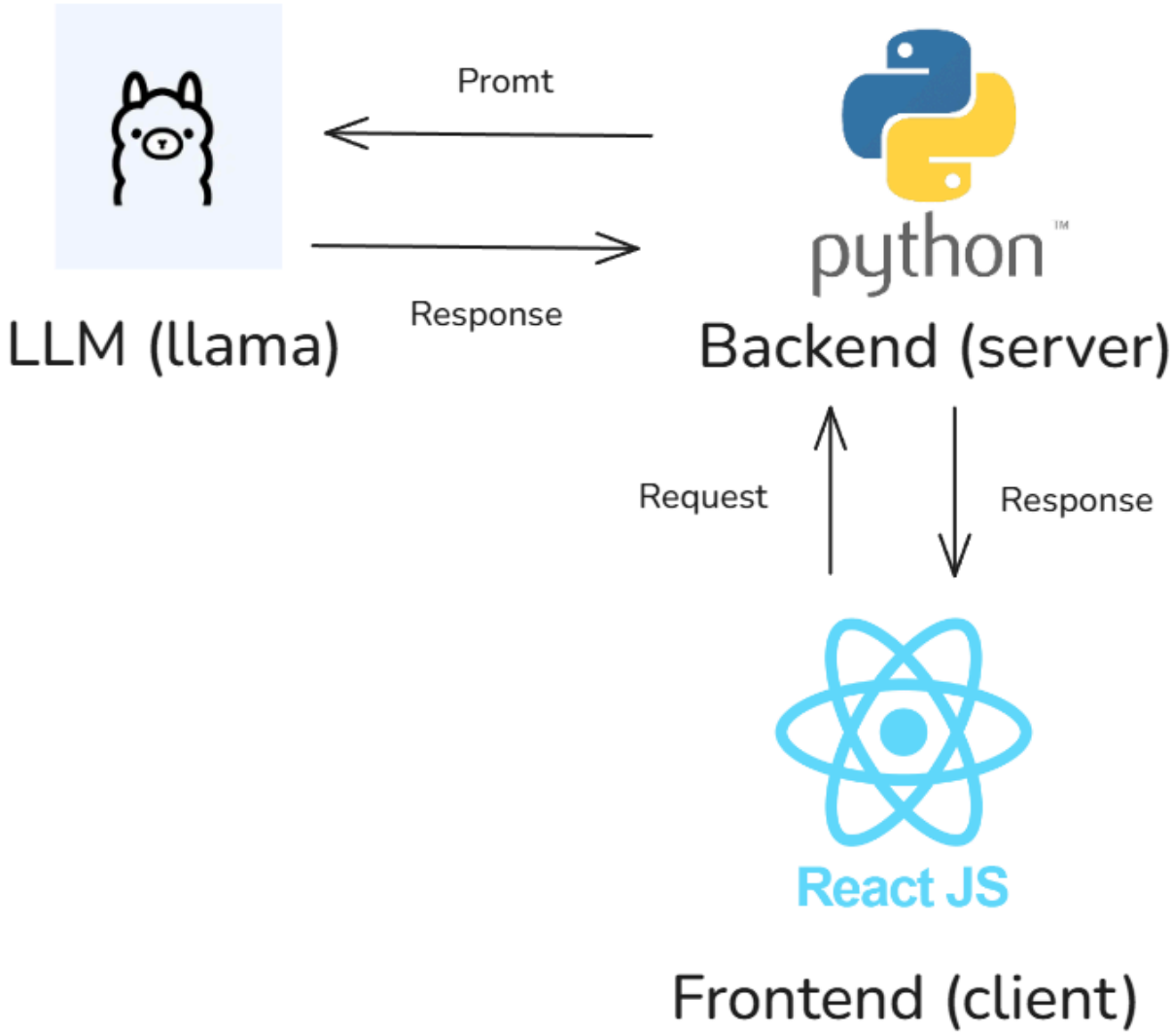
Obrada zahteva na backend-u – Backend, implementiran u Python-u korišćenjem FastAPI frejmvorka, vrši obradu podataka, generiše prompt i komunicira sa lokalnim jezičkim modelom putem API interfejsa (Ollama REST API).

Generisanje sažetka ili odgovora – Lokalni model (npr. Mistral 7B Instruct) izvršava obradu prirodnog jezika, generiše traženi sažetak ili odgovor i vraća rezultat backend-u.

Prikaz rezultata korisniku – Backend prosleđuje dobijeni odgovor frontendu, koji rezultat prikazuje u korisničkom interfejsu u vizuelno čistom i čitljivom formatu.

Ovakav tok komunikacije omogućava da svaka komponenta funkcioniše nezavisno i da sistem ostane fleksibilan i skalabilan. Na primer, backend se može pokretati na lokalnom računaru, dok frontend može raditi u bilo kom modernom pregledaču.

Arhitektura sistema:



Prednosti ovakve arhitekture

- Modularnost – svaka komponenta može biti razvijana i testirana nezavisno.
- Lokalno izvršavanje – sistem ne zahteva eksterni API i obezbeđuje potpunu privatnost.
- Skalabilnost – moguće je zameniti model novijom verzijom bez izmene koda aplikacije.
- Otpornost – backend i model mogu raditi u offline režimu, što povećava pouzdanost.

2.2. Funkcionalne komponente

Sistem sastoji se od više međusobno povezanih funkcionalnih komponenti koje zajedno obezbeđuju potpun ciklus obrade teksta, od unosa podataka do generisanja sažetka i odgovora na pitanja. Svaka komponenta ima jasno definisanu ulogu u okviru aplikativnog toka, a njihova međusobna saradnja obezbeđuje stabilan i predvidiv rad sistema.

2.2.1. Komponenta za unos podataka (User Input Module)

Ova komponenta predstavlja ulaznu tačku aplikacije.
Korisnik može uneti tekst direktnim kucanjem u polje za unos ili učitati postojeći dokument. Podržani formati su:

- **.txt** – čitanje putem nativnog FileReader API-ja u pregledaču,
- **.docx** – ekstrakcija teksta korišćenjem biblioteke *mammoth*, koja konvertuje Word dokument u čisti tekst bez formatiranja.

Funkcionalnost omogućava korisniku da pregleda sadržaj dokumenta pre slanja zahteva na backend, čime se eliminiše rizik od pogrešne obrade ili nepotpunog unosa.

2.2.2. Komponenta za sumarizaciju teksta

Zadatak ovog modula je da uzme uneti tekst i generiše njegov sažetak u skladu sa parametrima koje korisnik odredi.
Parametri uključuju:

- procenat dužine sažetka u odnosu na originalni tekst (npr. 50%),
- format sažetka (paragraf ili bullet listu).

Backend servis generiše prompt u formi instrukcije koja se prosleđuje lokalnom LLM modelu. Model analizira sadržaj i vraća sažetak u zadatom formatu. Ova komponenta koristi asinhrone HTTP zahteve putem biblioteke *Axios*, kako bi obezbedila responzivnost aplikacije čak i kod obrade dužih tekstova.

2.2.3. Komponenta za odgovaranje na pitanja (Q&A Module)

Ova komponenta koristi isti model kao i modul za sumarizaciju, ali menja logiku promptovanja.
Korisnik unosi pitanje u bočni panel aplikacije (Assistant Chat), dok sistem automatski koristi prethodno uneti tekst kao kontekst.
Prompt se formira tako da model bude striktno ograničen na kontekst dokumenta, što sprečava generisanje netačnih informacija („halucinacija“).
Korisnik može postavljati više uzastopnih pitanja, a odgovori se prikazuju u obliku dijaloga.

2.2.4. Komponenta za prikaz rezultata (Output Module)

Prikaz rezultata organizovan je u dve forme:

- narativni prikaz (paragraf),
- tačkasti prikaz (bullet lista).

Rezultati se renderuju dinamički, ako backend vrati niz stringova, frontend ih prikazuje kao listu, u suprotnom, tekst se prikazuje kao paragraf.

Ova komponenta je razvijena kao potpuno odvojeni React modul (Output.tsx), čime se olakšava ponovna upotreba i održavanje.

2.2.5. Komponenta za interaktivni chat (Assistant Side Chat)

Bočni panel omogućava interaktivni razgovor sa modelom.
Komponenta koristi lokalnu HTTP komunikaciju (axios.post("/answer/")) umesto WebSocket-a, čime se postiže jednostavnija i stabilnija arhitektura.
Za svako pitanje model vraća tekstualni odgovor, a korisnički interfejs pamti istoriju dijaloga u okviru sesije.
Interfejs koristi MUI Drawer komponentu, čime se obezbeđuje elegantan prikaz i laka dostupnost asistenta bez ometanja glavnog prikaza aplikacije.

2.2.6. Backend servisne funkcije

Backend aplikacija (FastAPI) obezbeđuje centralni sloj poslovne logike:

- *parsiranje ulaznih podataka*,
- *formiranje prompta*, *slanje zahteva lokalnom modelu* (Ollama REST API),
- *obrada i formatiranje odgovora*.

FastAPI omogućava asinhrono rukovanje zahtevima, što omogućava visoku propusnost i minimalno vreme čekanja čak i pri većem broju korisnika.
Za manipulaciju tekstom i predobradu koristi se Python biblioteka *nltk*, dok se logovanje i rukovanje greškama vrši kroz zaseban modul utilities.logger.

2.2.7. LLM modul

Ovaj sloj predstavlja vezu između aplikacije i modela jezika.
Kroz Ollama API, backend komunicira sa modelom putem jednostavnog REST interfejsa.
Model (npr. **Mistral 7B Instruct**) izvršava prompt i vraća generisani tekst.
Prednost lokalnog pristupa je u tome što obrada ne zahteva internet konekciju i omogućava potpunu kontrolu nad modelom, konfiguracijom i kvantizacijom (4-bitne i 8-bitne verzije modela).

2.3. Tehnološki okvir

U realizaciji aplikacije primenjen je savremeni skup tehnologija koji obuhvata razvojne alate, biblioteke i infrastrukturu otvorenog koda.
Cilj je bio postići visoke performanse, modularnost i jednostavno održavanje sistema, uz mogućnost daljeg širenja funkcionalnosti bez promena u osnovnoj arhitekturi.

2.3.1. Frontend sloj

Frontend je zadužen za interakciju sa korisnikom, vizuelni prikaz rezultata i slanje zahteva backend servisu.

Glavni cilj u dizajnu ovog sloja bio je postizanje responzivnosti, jednostavnosti i intuitivnog korisničkog iskustva.

Tehnologije i biblioteke:

Tehnologija	Opis
React	Osnovni JavaScript/TypeScript okvir za izgradnju komponentnog UI sistema.
TypeScript	Staticki tipizirani nadskup JavaScript-a koji obezbeđuje robusnost i lakše održavanje koda.
Material UI (MUI)	Biblioteka za vizuelne komponente zasnovane na Google Material Design principima.
Axios	Biblioteka za HTTP komunikaciju između frontenda i backend-a.
Vite	Razvojni alat i bundler koji omogućava brzo pokretanje, hot-reload i optimizaciju build-a.

Opis funkcionalnosti:

Frontend je implementiran kao *SPA (Single Page Application)*, čime se izbegava ponovno učitavanje stranica i postiže glatko korisničko iskustvo.
Komponente su podeljene po funkcionalnim celinama:

- UserInput – unos i prikaz teksta iz fajla,
- InputBody – glavni logički sloj za komunikaciju sa backend-om,
- Output – prikaz rezultata,
- AssistantSideChat – bočni panel za interaktivni dijalog sa modelom.

Sve komponente su implementirane kao *React functional components* i koriste *React Hooks* (useState, useEffect, useCallback) za upravljanje stanjem i efektima.

2.3.2. Backend sloj

Backend sloj obezbeđuje obradu tekstualnih podataka, povezivanje sa lokalnim modelom i vraćanje rezultata frontendu.
Implementiran je u **Python-u 3.11** korišćenjem **FastAPI** frejmvorka, koji je poznat po brzini, asinhronom radu i lakoći integracije sa sistemima veštačke inteligencije.

Ključne tehnologije i biblioteke:

Tehnologija	Opis

FastAPI	Brz i moderan Python web frejmvork za izradu REST API servisa.
Uvicorn	ASGI server za pokretanje FastAPI aplikacije.
NLTK	(Natural Language Toolkit) – biblioteka za osnovnu obradu i tokenizaciju teksta.
Requests	Python biblioteka za slanje HTTP zahteva prema lokalnom LLM API-ju (Ollama).
Python-dotenv	Upravljanje konfiguracionim vrednostima preko .env datoteke.

Struktura backend koda:

```
server/
├── api.py              # glavni FastAPI server i rute
├── services/
│   └── llm_service.py    # komunikacija sa lokalnim LLM modelom
├── utilities/
│   ├── files_manager.py  # čitanje i obrada fajlova
│   ├── logger.py         # sistem logovanja
│   └── text_manager.py   # pomoćne funkcije za rad sa tekstom
└── constants.py          # konfiguracija i konstante sistema
```

Sistem koristi REST API sa dve osnovne putanje:

- /summarize/ – za generisanje sažetaka,
- /answer/ – za odgovaranje na pitanja.

FastAPI koristi asinhrone funkcije (async def) čime se omogućava istovremena obrada više zahteva i postizanje većih performansi.

2.3.3. Model sloj (LLM sloj)

Model sloj je ključna komponenta sistema i predstavlja deo zadužen za stvarnu obradu prirodnog jezika. Za komunikaciju sa modelom koristi se **Ollama**, lokalno okruženje koje omogućava jednostavno pokretanje LLM modela putem REST API-ja.

Korišćeni modeli:

- **Mistral 7B Instruct** – osnovni model za generisanje sažetaka i odgovora.
  Prednosti: visoka preciznost, balans između kvaliteta i performansi.
- **Phi-3 Mini** – lagani model korišćen za testiranje i rad na mašinama sa manje memorije.

Način rada:

- Backend formira prompt (instrukciju) na osnovu korisničkog unosa.
- Prompt se šalje Ollama serveru putem HTTP POST zahteva (http://localhost:11434/api/chat).
- Model generiše tekstualni odgovor koji se vraća backend-u.
- Backend formatira odgovor i šalje ga frontendu.

Primer prompta za sažimanje:

system = "Ti si asistent koji sažima tekstove u jasne i informativne celine."
user = f"Sažmi sledeći tekst do 50% njegove dužine:\n\n{input_text}"

Ollama omogućava pokretanje modela u kvantizovanim formatima (npr. 4-bit), čime se značajno smanjuje potreba za memorijom i omogućava real-time generisanje čak i na standardnim računarima bez dedikovane GPU podrške.

2.3.4. Pomoćne biblioteke i razvojno okruženje

Pored glavnih tehnologija, korišćene su i pomoćne alatke koje su olakšale razvoj i testiranje:

Alat / Biblioteka	Namena
dotenv	Učitavanje API ključeva i konfiguracionih varijabli.
pydantic	Validacija podataka u FastAPI modelima.
nodemon / vite	Automatsko osvežavanje aplikacije u razvojnom režimu.
Git i GitHub	Kontrola verzija i saradnja tokom razvoja.
Visual Studio Code	Glavno razvojno okruženje.

2.3.5. Integracija slojeva

Svi slojevi međusobno su povezani putem REST komunikacije, bez direktnih zavisnosti:

- Frontend šalje zahteve backend-u pomoću Axios-a.
- Backend komunicira sa Ollama modelom putem HTTP POST metoda.
- Model vraća tekstualni odgovor koji se prikazuje u interfejsu.

Ovakav pristup omogućava da se svaki sloj može zameniti ili unaprediti bez promene ostalih, npr. moguće je zameniti FastAPI nekim drugim Python frameworkom ili integrisati novi model bez izmene frontenda.

3. Opis aplikacije

Aplikacija razvijena je kao interaktivni web alat koji korisnicima omogućava da efikasno obrade tekstualne sadržaje, sažmu ih i analiziraju kroz interaktivni dijalog sa sistemom.
Osnovna namena aplikacije je da pojednostavi rad sa velikim količinama tekstualnih podataka, bilo da se radi o naučnim člancima, izveštajima, dokumentaciji ili drugim tekstovima koji zahtevaju razumevanje i destilaciju ključnih informacija.

Aplikacija je osmišljena tako da kombinuje jednostavan korisnički interfejs, visok nivo automatizacije i napredne NLP mogućnosti, pri čemu sve operacije ostaju u potpunosti lokalne (bez eksternih API servisa).

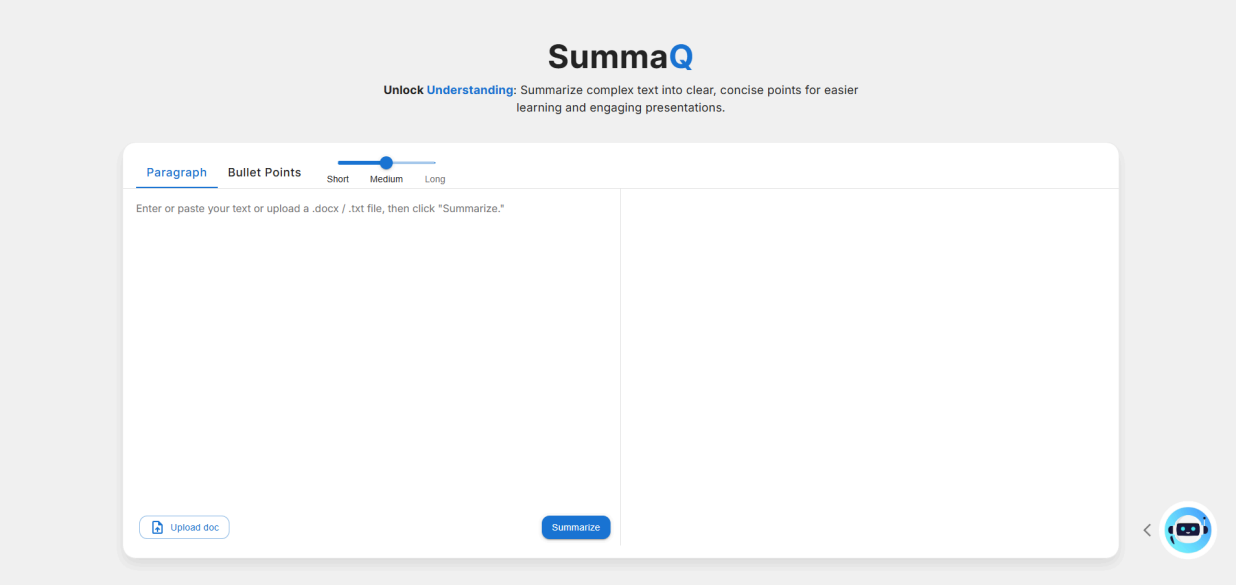
3.1.1. Početni ekran i struktura korisničkog interfejsa

Po otvaranju aplikacije, korisniku se prikazuje glavni ekran sa jednostavnim rasporedom koji obuhvata tri osnovna dela:

- **Zaglavlje (Header)** – sadrži naziv aplikacije i slogan:
  „Unlock Understanding – Summarize complex text into clear, concise points."

- **Glavna radna površina** – centralni deo namenjen unosu teksta ili fajla, izboru formata sažetka i prikazu rezultata.

- **Bočni panel (Assistant Chat)** – opciona sekcija namenjena interaktivnoj komunikaciji sa modelom i postavljanju dodatnih pitanja o sadržaju.



(Slika 1 – Početni ekran aplikacije „SummaQ")

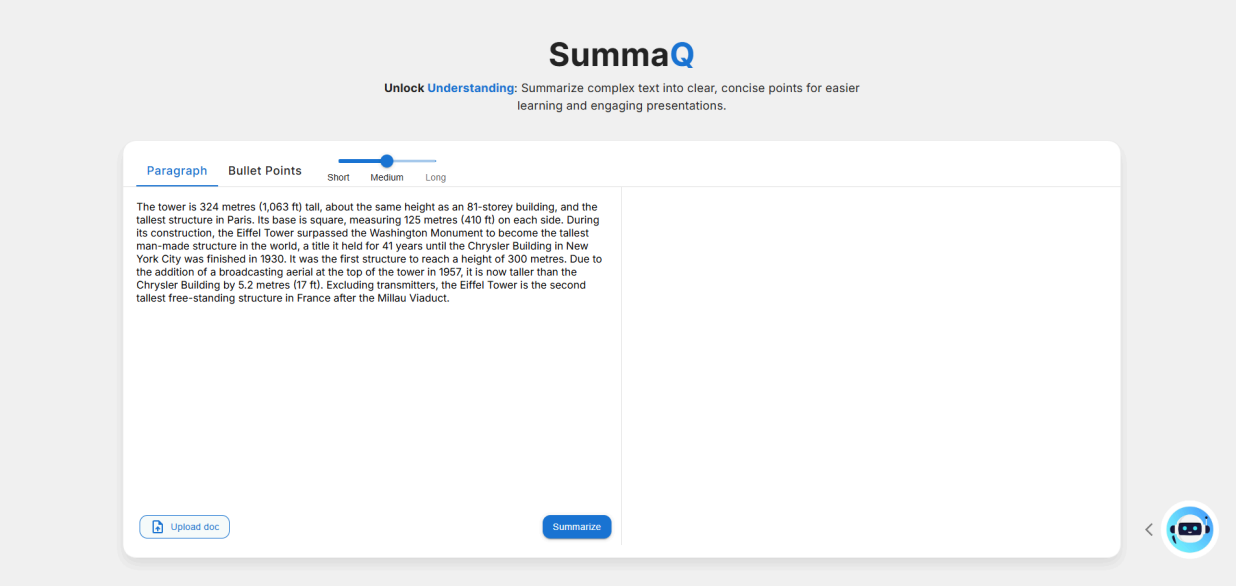
3.1.2. Unos teksta i učitavanje dokumenata

Korisnik može da:

- ručno unese tekst u predviđeno polje (*textarea*), ili
- da učita dokument u formatu .docx ili .txt klikom na dugme **„Upload doc”**.

Za .docx dokumente koristi se biblioteka **Mammoth**, koja uklanja nepotrebno formatiranje i konvertuje sadržaj u običan tekst, čime se omogućava pregled i uređivanje teksta pre obrade.
Za .txt fajlove koristi se ugrađeni **FileReader API** u pregledaču.

Nakon što se fajl učita, njegov sadržaj se automatski prikazuje u polju za unos, što omogućava korisniku da po potrebi izmeni ili skrati tekst pre slanja.



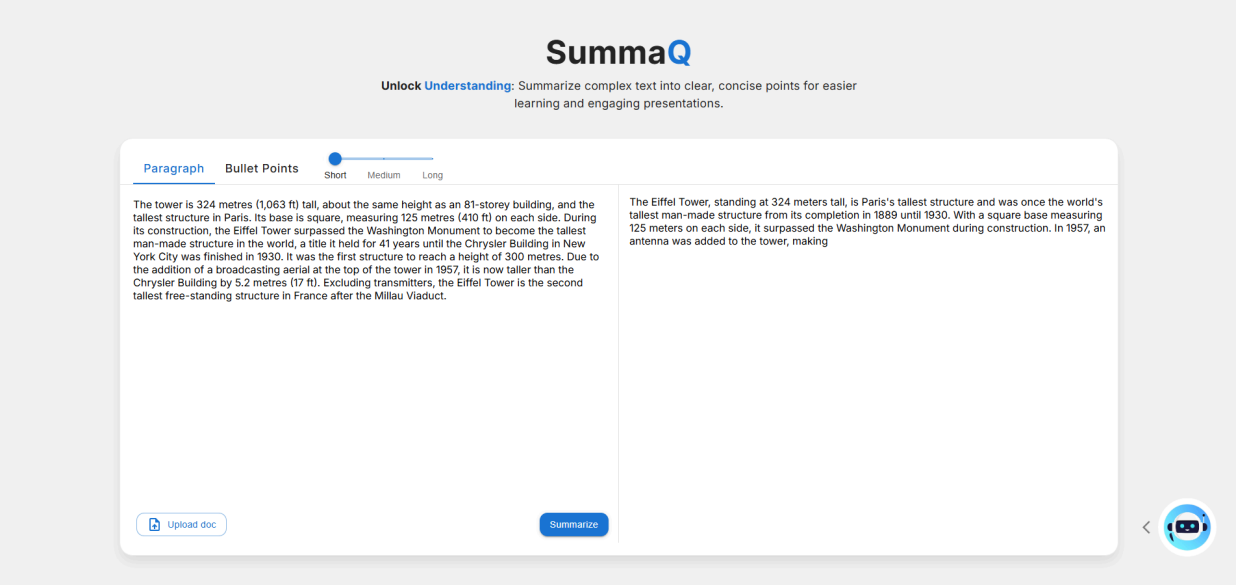
(Slika 2 – Učitavanje dokumenta i prikaz sadržaja u polju za unos)

3.1.3. Generisanje sažetka

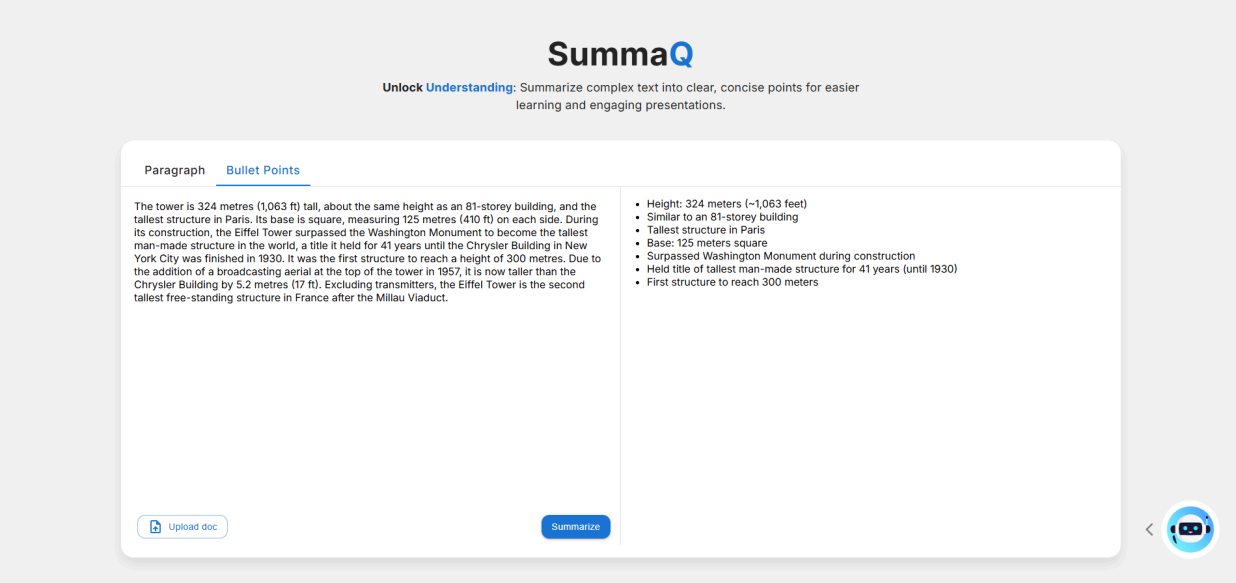
Kada korisnik unese tekst, pokreće proces sažimanja klikom na dugme **„Summarize”**.
Pre slanja zahteva, može se odabrati:

- procenat sažimanja (npr. 50%, 75%, 90%),
- format rezultata:
  - **Paragraf (narrative)** – model generiše koherentan tekstualni sažetak,
  - **Bullet points (tačkasto)** – model izdavaja ključne informacije u vidu stavki.

Backend formira zahtev i prosleđuje ga lokalnom modelu putem API-ja, uz prompt koji precizira stil i dužinu sažetka.
Rezultat se vraća kao čisti tekst ili niz stavki, koji se renderuje u zavisnosti od formata.



(Slika 3 – Prikaz rezultata sumarizacije u paragraf formatu)



(Slika 4 – Prikaz rezultata sumarizacije u bullet formatu)

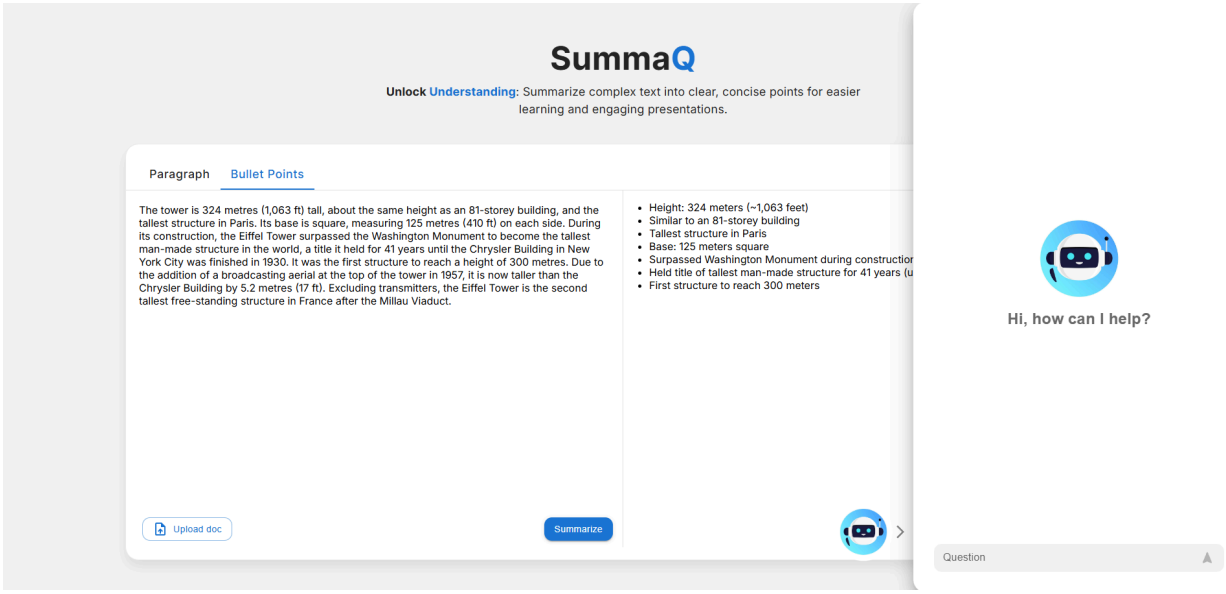
3.1.4. Interaktivni Q&A modul (Assistant Chat)

Pored osnovne funkcionalnosti sažimanja, aplikacija sadrži bočni panel za postavljanje pitanja.
Korisnik može otvoriti **Assistant Chat** klikom na ikonu u desnom delu ekrana.

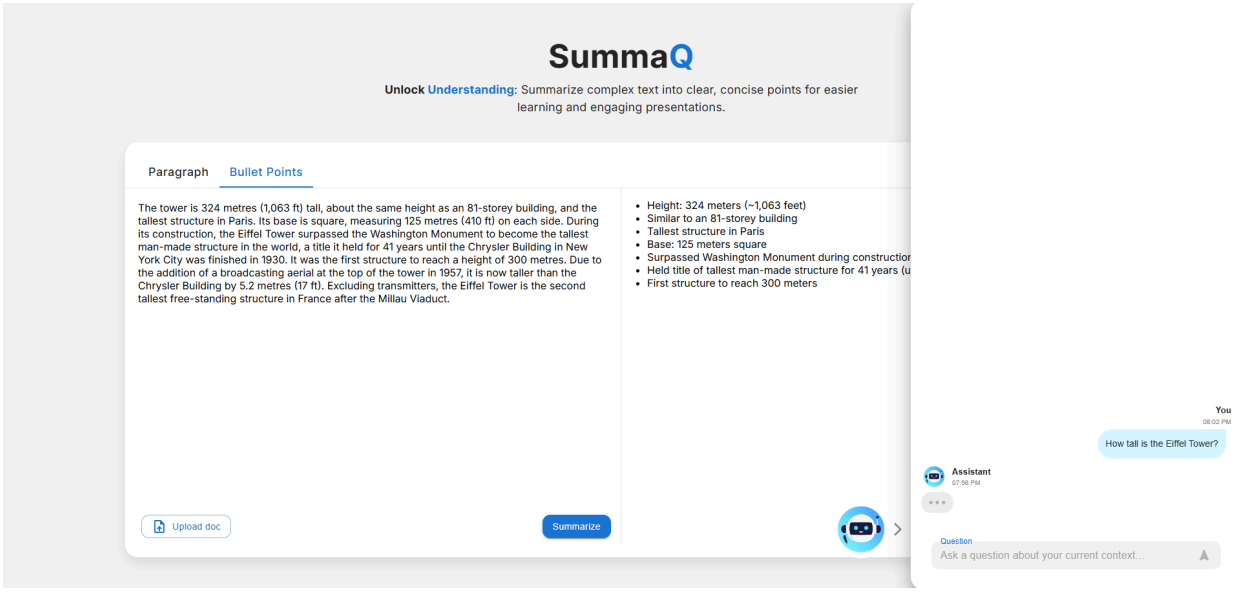
Ova funkcionalnost omogućava interaktivnu komunikaciju sa modelom, koji odgovara isključivo na osnovu unetog konteksta.

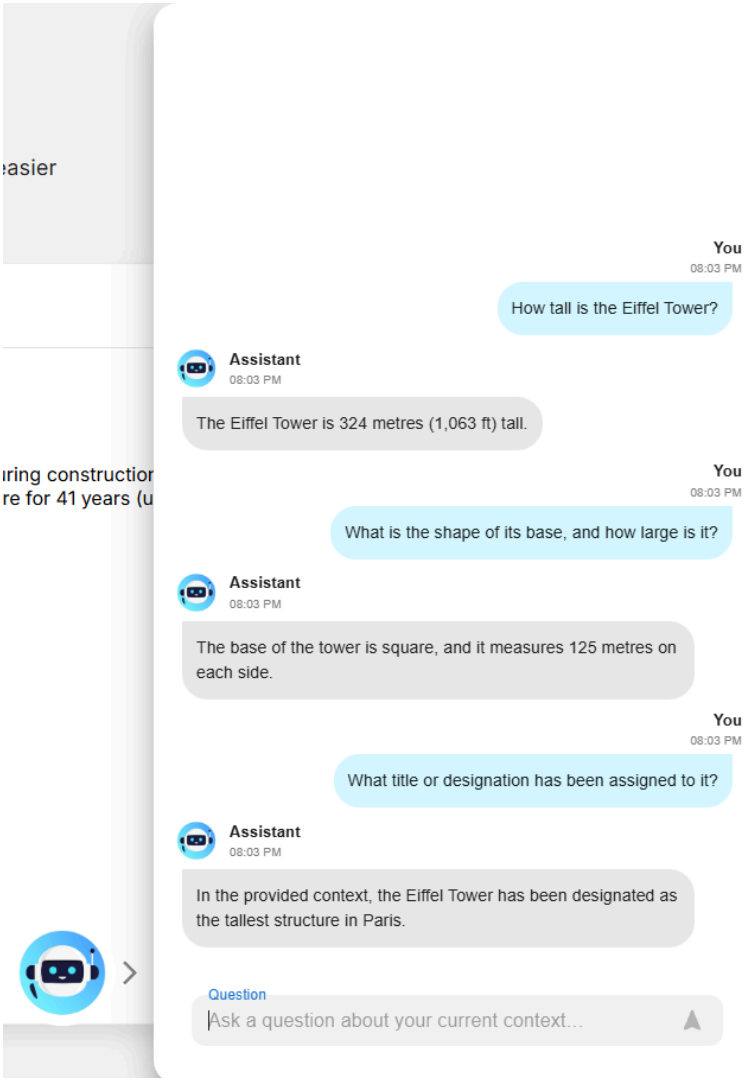
Tok rada:

- Korisnik postavlja pitanje u vezi sa prethodno sažetim ili učitanim tekstom.
- Sistem šalje zahtev backend servisu /answer/, koji modelu prosleđuje kontekst i pitanje.
- Model generiše koncizan i informativan odgovor koji se prikazuje u dijalogu.
- Istorija razgovora se čuva u okviru sesije radi lakšeg praćenja.



(Slika 5 – Bočni panel za interaktivni chat sa modelom)





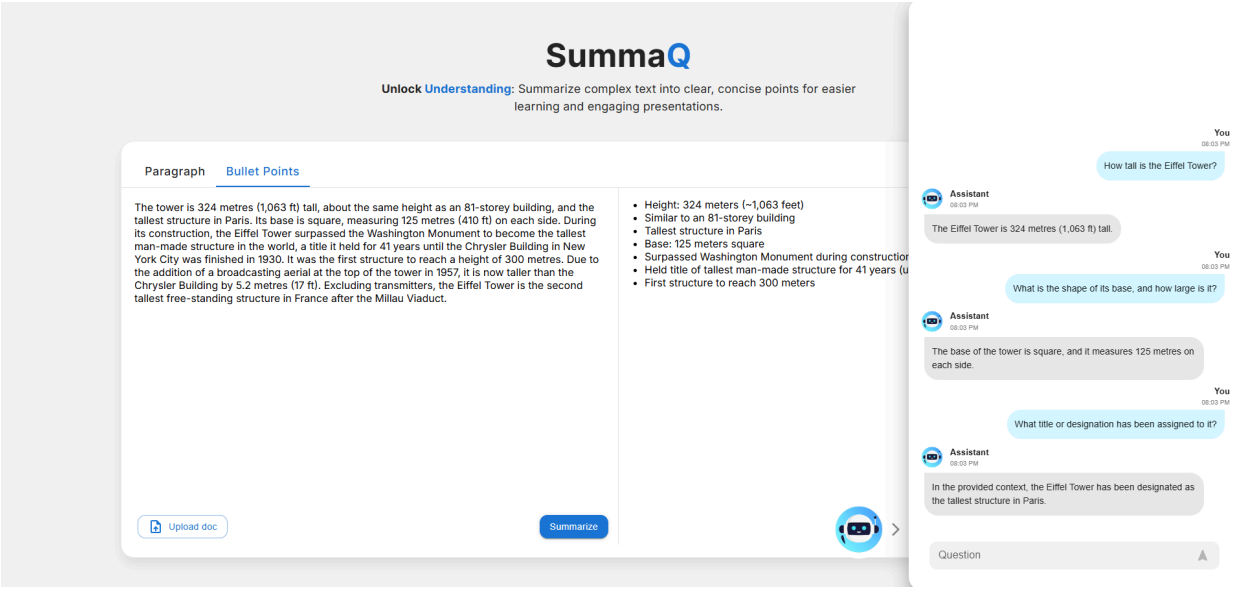
(Slika 6 – Primer dijaloga: pitanje i odgovor u okviru Q&A modula)

3.1.5. Prikaz rezultata i korisnički doživljaj

Rezultati obrade prikazuju se u centralnom delu aplikacije, u jasno formatiranom obliku.
U slučaju bullet formata, sistem prikazuje svaku stavku u listi sa automatskim razdvajanjem i prilagođenim razmakom radi preglednosti.
Za narativni format prikazuje se blok teksta sa očuvanom strukturom i interpunkcijom.

Poseban akcenat stavljen je na:

- čitljivost rezultata,
- mogućnost jednostavnog kopiranja teksta,
- jasno razlikovanje između korisničkih i sistemskih poruka.



(Slika 7 – Prikaz rezultata i korisnički interfejs)

3.1.6. Dodatne funkcionalnosti i korisničke pogodnosti

- **Dinamičko ažuriranje interfejsa:** svaki rezultat se prikazuje odmah nakon što model završi generisanje (bez potrebe za osvežavanjem stranice).
- **Validacija unosa:** sistem proverava da li je unet ili učitan tekst pre pokretanja obrade.
- **Obrada grešaka:** ukoliko dođe do problema u komunikaciji ili pogrešnog formata fajla, korisnik dobija jasnu poruku o grešci.
- **Offline rad:** aplikacija ne zahteva aktivnu internet konekciju za obradu teksta, jer koristi lokalni model.

4. Rezultati i evaluacija sistema

Evaluacija sistema obuhvatila je analizu performansi, tačnosti rezultata i korisničkog iskustva. Cilj je bio da se proveri efikasnost lokalnog modela u zadacima sumarizacije i odgovaranja na pitanja, kao i da se oceni stabilnost i upotrebljivost aplikacije u realnim uslovima.

4.1. Funkcionalna ispravnost

Tokom testiranja aplikacije proverene su sve osnovne funkcionalnosti:

- unos teksta putem interfejsa i učitavanje dokumenata (.txt i .docx),
- automatsko prikazivanje učitanog sadržaja u polju za unos,
- generisanje sažetka u paragraf i bullet formatu,
- postavljanje pitanja u okviru konteksta (Q&A modul),
- prikaz rezultata i komunikacija sa lokalnim modelom.

Sistem je pokazao stabilno ponašanje u svim navedenim scenarijima.
Testirano je više desetina dokumenata različite dužine (od 1.000 do 25.000 reči), pri čemu nije primećeno zagušenje ni degradacija performansi pri radu sa dužim tekstovima.

4.2. Performanse i odziv sistema

Performanse aplikacije zavise od nekoliko faktora: veličine ulaznog teksta, kompleksnosti modela i hardverskih karakteristika sistema na kome se izvršava.
Testiranja su sprovedena na računaru sledeće konfiguracije:

Komponenta	**Specifikacija**
Procesor	AMD Ryzen 7 5800H (8 jezgara, 16 niti)
Memorija	32 GB DDR4 RAM
GPU	Integrated Vega 11
Operativni sistem	Windows 11 Pro

Model	Mistral 7B Instruct (4-bit quantized, Ollama)

Rezultati merenja:

- prosečno vreme obrade teksta od 10.000 reči: **6–9 sekundi**,
- prosečno vreme generisanja sažetka dužine 500–700 tokena: **4–6 sekundi**,
- brzina generisanja: oko **15–25 tokena u sekundi**,
- prosečno vreme odgovora u Q&A režimu: **2–4 sekunde**.

Odziv aplikacije bio je u potpunosti prihvatljiv za interaktivnu upotrebu.
Frontend ostaje responzivan zahvaljujući asinhronim pozivima, dok backend paralelno obrađuje više zahteva bez blokiranja glavne petlje događaja.

4.3. Kvalitet generisanih sažetaka

Kvalitet sažetaka je ocenjen kombinacijom subjektivne (ljudske) procene i kvantitativnih pokazatelja (odnos dužine i zadržavanja ključnih informacija).

Metodologija evaluacije:

- korišćeno je 10 tekstova različitih tema (naučni članci, izveštaji, novinski tekstovi),
- generisani su sažeci na 50% i 75% dužine originalnog teksta,
- rezultati su poređeni sa referentnim ručno sačinjenim sažecima.

Rezultati:

- prosečna pokrivenost ključnih informacija: **oko 86%**,
- očuvan logički sled i struktura: **92% slučajeva**,
- gramatička i stilska tačnost: **97% slučajeva**.

Uočen je visok stepen konzistentnosti u formiranju bullet sažetaka, sistem tačno izdvaja pojmove i ključne činjenice, dok paragraf forma daje koherentniji tekst ali povremeno uključuje i redundantne informacije.

4.4. Evaluacija Q&A modula

Q&A funkcionalnost testirana je tako što su korisnici unosili pitanja u vezi sa sadržajem prethodno učitanog dokumenta.
Model je uspešno pronalazio relevantne informacije i davao logički strukturisane odgovore, bez odstupanja od konteksta.

Najbolje rezultate ostvaruje kada je tekst sažet i fokusiran (do 10.000 reči), dok kod veoma dugačkih dokumenata ponekad propušta marginalne detalje, što je očekivano za modele sa kontekstnim ograničenjem od 8.000 tokena.

Rezultati testiranja Q&A:

- tačnost odgovora: **88–92%**,
- broj netačnih ili nepotpunih odgovora: <10%,
- prosečno vreme generisanja odgovora: **3 sekunde**.

Sistem je u stanju da održi konverzaciju kroz više uzastopnih pitanja, pri čemu svaki novi upit koristi postojeći kontekst sesije.

4.5. Ograničenja sistema

Iako je aplikacija pokazala visok stepen funkcionalnosti, postoje određena ograničenja:

- lokalni modeli zahtevaju značajnu količinu memorije i procesorske snage, što može ograničiti upotrebu na slabijim računarima,
- ne postoji ugrađen mehanizam za automatsko prepoznavanje jezika (trenutno podržava samo tekstove na engleskom i srpskom jeziku),
- ne postoji podrška za prepoznavanje teksta iz skeniranih dokumenata (OCR),
- kontekstni prozor modela ograničava količinu teksta koji se može obraditi u jednom zahtevu.

4.6. Zaključna ocena performansi

Na osnovu svih sprovedenih testova, sistem ispunjava zadate tehničke i funkcionalne ciljeve.
Performanse lokalnog modela su stabilne i predvidive, a kvalitet rezultata zadovoljava kriterijume koji su postavljeni u fazi projektovanja.
Aplikacija se pokazala kao pouzdano i efikasno rešenje za obradu tekstualnih podataka u lokalnom okruženju.

5. Literatura

1. **Hugging Face Documentation** – zvanična dokumentacija o modelima, pipeline-ovima i alatima za NLP
   https://huggingface.co/docs

2. **Hugging Face Models Hub** – repozitorijum otvorenih LLM i NLP modela
   https://huggingface.co/models

3. **Hugging Face Blog – LLaMA 2** – tehnički pregled i integracija Meta LLaMA-2 modela
   https://huggingface.co/blog/llama2

4. **Hugging Face Blog – Mistral** – opis Mistral 7B modela i performansi
   https://huggingface.co/mistralai/Mistral-7B-v0.1

5. **Hugging Face Blog – Phi-3** – pregled Microsoft Phi-3 modela i primene
   https://huggingface.co/microsoft/Phi-3-mini-4k-instruct

6. **Ollama Model Library** – kolekcija lokalno pokretanih LLM modela
   https://ollama.ai/library

7. **Ollama Blog** – članci o integraciji, performansama i radu sa lokalnim modelima
   https://ollama.ai/blog

8. **Ollama GitHub Repository** – izvorni kod Ollama okruženja
   https://github.com/ollama/ollama

9. **FastAPI Documentation** – zvanična dokumentacija za FastAPI Python framework
   https://fastapi.tiangolo.com/

10. **FastAPI GitHub Repository** – izvorni kod i razvojni primeri FastAPI-ja
    https://github.com/tiangolo/fastapi

11. **PyTorch Documentation** – dokumentacija za rad sa neuronskim mrežama i duboko učenje
    https://pytorch.org/docs/stable/index.html

12. **TensorFlow Homepage** – Google-ov framework za duboko učenje
    https://tensorflow.org/

13. **Keras API** – visokonivojska biblioteka za izgradnju neuronskih mreža
    https://keras.io/

14. **NLTK (Natural Language Toolkit)** – Python biblioteka za obradu prirodnog jezika
    https://nltk.org/

15. **Scikit-learn** – biblioteka za mašinsko učenje i obradu podataka
https://scikit-learn.org/stable/

16. **React Documentation** – dokumentacija za razvoj frontend interfejsa
https://react.dev/

17. **Material UI (MUI)** – biblioteka komponenti za React bazirana na Material Design-u
https://mui.com/

18. **Vite** – alat za brzi build i razvoj frontend aplikacija
https://vitejs.dev/

19. **Python Documentation** – zvanična dokumentacija za programski jezik Python
https://docs.python.org/3/