Izrada konverzacijskog agenta koristeći Angular

Žiroš, Domagoj

Sveučilište Josipa Jurja Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike

Studij: Sveučilišni prijediplomski studij Matematika i računarstvo

Izrada konverzacijskog agenta koristeći Angular

Završni rad

Mentor:
doc.dr.sc. Domagoj Ševerdija

Student:
Domagoj Žiroš

Osijek, 2024

Sadržaj

1 | Uvod

1.1 Motivacija i ciljevi rada

S razvojem tehnologije i sveprisutnošću interneta, konverzacijski agenti su postali ključni dio moderne komunikacije i interakcije s korisnicima. Od jednostavnih automatskih odgovora do sofisticiranih sustava temeljenih na umjetnoj inteligenciji, razvili su se u alat koji poboljšava korisničko iskustvo, optimizira poslovne procese i omogućava brži pristup informacijama. S obzirom na njihovu sve važniju ulogu u različitim industrijama, razvoj naprednih agenata za razgovor predstavlja značajan izazov i priliku.

Ovaj rad fokusira se na izradu konverzacijskog agenta koristeći Angular, moderni okvir za razvoj korisničkog sučelja. Cilj je pružiti uvid u proces izrade funkcionalnog i interaktivnog konverzacijskog agenta koji koristi Angular i TypeScript kako bi omogućio dinamičko i intuitivno korisničko iskustvo. Ovim radom nastojimo demonstrirati kako se tehnologije i alati mogu integrirati kako bi se razvila kvalitetna aplikacija.

1.2 Pregled teme

U ovom radu, prvo ćemo se upoznati s osnovnim konceptima Angulara i TypeScripta, koji su ključni za razvoj naše aplikacije. Angular omogućava izgradnju složenih aplikacija putem komponenti, modula i servisa, dok TypeScript dodaje dodatnu vrijednost kroz statičku tipizaciju i modularnost. Razumijevanje ovih tehnologija omogućuje nam bolje planiranje i implementaciju aplikacije.

Nakon toga, detaljno ćemo opisati postupak postavljanja projekta, uključujući instalaciju potrebnih alata i konfiguraciju radnog okruženja. Slijedit će opis ključnih komponenti i modula aplikacije, s naglaskom na strukturu i organizaciju koda. Također ćemo predstaviti implementaciju korisničkog sučelja, od HTML i CSS stilizacije do funkcionalnosti aplikacije, uključujući način na koji konverzacijski agent obrađuje i odgovara na korisničke poruke. Također ćemo posvetiti pažnju integraciji sa OpenAI API-jem, koji omogućava generiranje odgovora.

2 | Tehnologije i alati

2.1 Angular

Angular je popularan okvir za razvoj korisničkih sučelja, razvijen od strane Googlea, koji se koristi za izradu dinamičkih web aplikacija. Prvobitno je objavljen 2010. godine pod imenom AngularJS, ali je kasnije potpuno prepisan i redizajniran, što je rezultiralo pojavom Angulara (poznatog i kao Angular 2+) 2016. godine.

Glavne značajke Angulara uključuju:

- **Dvostruko povezivanje podataka (Data Binding)**: Angular omogućava sinkronizaciju između modela podataka i prikaza (view) u stvarnom vremenu, što olakšava razvoj interaktivnih aplikacija.
- **Komponente**: Angular koristi komponente kao osnovne građevne blokove aplikacija. Komponente omogućuju modularnost i ponovnu upotrebu koda.
- **Dependency Injection (DI)**: Angular koristi DI kako bi upravljao zavisnostima objekata, što pojednostavljuje testiranje i ponovno korištenje koda.
- **Direktive**: Angular omogućava proširenje HTML-a korištenjem direktiva, koje omogućuju prilagodbu ponašanja elemenata.
- **Rutiranje**: Angularov ugrađeni sustav za rutiranje omogućuje navigaciju između različitih pogleda unutar aplikacije bez ponovnog učitavanja stranice.

Angular je dobar izbor za razvoj velikih aplikacija zahvaljujući svojoj modularnoj strukturi, opsežnoj dokumentaciji i aktivnoj zajednici. Osim toga, pruža odličnu podršku za TypeScript, što doprinosi većoj pouzdanosti i održivosti koda. Više o Angularu i njegovim značajkama može se vidjeti na [3].

2.2 TypeScript

TypeScript je moderan programski jezik koji je razvijen od strane Microsofta i koji nadograđuje JavaScript dodavanjem niza korisnih značajki. Kao nadskup

JavaScript-a, TypeScript omogućava korištenje postojećeg JavaScript koda, dok istovremeno pruža dodatne mogućnosti koje olakšavaju razvoj velikih i složenih aplikacija.

Jedna od glavnih prednosti TypeScripta je njegovo statičko tipiziranje. Dok JavaScript koristi dinamičko tipiziranje, što može dovesti do grešaka pri izvođenju, TypeScript omogućava programerima da definiraju tipove varijabli, funkcija i objekata unaprijed. Ova karakteristika pomaže u otkrivanju grešaka u fazi kompilacije, prije nego što aplikacija bude pokrenuta, čime se značajno povećava pouzdanost koda.

Uz to, TypeScript uvodi koncept klasa i sučelja, što doprinosi organiziranju i strukturiranju koda na način koji olakšava rad u velikim projektima. Korištenjem klasa, programeri mogu definirati objekte s jasno definiranim metodama i svojstvima, dok sučelja omogućavaju precizno definiranje oblika objekata i obaveza koje objekti moraju ispuniti. Ovi dodaci olakšavaju stvaranje čitljivog i održivog koda, što je posebno važno u kontekstu kompleksnih aplikacija.

Osim toga, TypeScript je poznat po svojoj izvrsnoj integraciji s alatima i okruženjima za razvoj. Njegova kompatibilnost s modernim razvojnim alatima i razvojnim okvirima, uključujući Angular, čini ga izuzetno popularnim među programerima. Kada se koristi u kombinaciji s Angularom, TypeScript doprinosi boljoj strukturi koda i većoj produktivnosti, zahvaljujući svojim značajkama kao što su autokompletiranje i refaktorizacija. Više o svojstvima Typescripta na [4].

Sve ove karakteristike čine TypeScript izuzetnim izborom za razvoj složenih web aplikacija, omogućujući programerima da pišu kod koji je lakše razumljiv, održiviji i manje sklon greškama.

3 | Postavljanje projekta

3.1 Instalacije

3.1.1 Instalacija Node.js-a

Prvi korak u postavljanju Angular projekta je instalacija Node.js-a, koji je potreban za upravljanje paketima i pokretanje Angular CLI-a. Node.js možemo preuzeti sa [službene web stranice](https://nodejs.org/). Instaliramo najnoviju LTS (Long Term Support) verziju koja je stabilna i preporučena za većinu korisnika.

3.1.2 Instalacija Angular CLI-a

Nakon što smo instalirali Node.js, trebamo instalirati Angular CLI (Command Line Interface), koji je alat za upravljanje Angular projektima. To možemo učiniti pomoću sljedeće naredbe u terminalu:

```
npm install -g @angular/cli
```

Ova naredba instalira Angular CLI globalno na našem računalu.

3.1.3 Kreiranje novog Angular projekta

Da bismo započeli novi Angular projekt, koristimo Angular CLI naredbu 'ng new'. Ova naredba će generirati osnovnu strukturu projekta. Na primjer, da bismo kreirali projekt s nazivom 'ChatApp', upišemo:

```
ng new ChatApp
```

Ova naredba će nas pitati za nekoliko opcija, uključujući želimo li uključiti Angular rutiranje i koji stil (CSS, SCSS, itd.) želimo koristiti.

3.1.4 Pokretanje razvojnog poslužitelja

Nakon što je projekt kreiran, možemo prijeći u direktorij projekta i pokrenuti razvojni poslužitelj pomoću sljedeće naredbe:

```
cd ChatApp
ng serve
```

Ova naredba pokreće razvojni poslužitelj i aplikaciju možemo otvoriti u našem web pregledniku na adresi 'http://localhost:4200'.

3.1.5 Kreiranje komponenti

Za dodavanje novih komponenti u Angular projekt, koristimo naredbu 'ng generate component'. Na primjer, za kreiranje komponente 'chat', upišemo:

```
ng generate component chat
```

Na taj način ćemo automatski generirati potrebne datoteke (chat.component.html, chat.component.css, chat.component.ts, chat.component.spec.ts) za novu komponentu unutar direktorija 'src/app/chat'.

3.1.6 Korištenje paketa

Ako trebamo instalirati dodatne pakete, poput knjižnica ili alata, koristimo 'npm' (Node Package Manager). Na primjer, za instalaciju PrimeNG-a, upišemo:

```
npm install primeng primeicons
```

Ovi koraci pružaju osnovne informacije za postavljanje i konfiguraciju Angular projekta. Detaljnije pogledati na [2]. Sljedeće sekcije će detaljnije obraditi implementaciju specifičnih funkcionalnosti i integracija unutar projekta.

3.2 Komponente i moduli u Angularu

Angular aplikacija se sastoji od komponenti i modula. Komponente su osnovne građevne jedinice svake Angular aplikacije, dok moduli omogućuju organizaciju i strukturiranje aplikacije u logične cjeline.

3.2.1 Komponente

Komponente u Angularu predstavljaju zasebne dijelove korisničkog sučelja (UI). Svaka komponenta se sastoji od tri glavna dijela:

- **HTML datoteka** - Definira izgled i strukturu korisničkog sučelja komponente.
- **CSS datoteka** - Sadrži stilove specifične za tu komponentu.
- **TypeScript datoteka** - Sadrži logiku i ponašanje komponente.

Komponenta se kreira pomoću naredbe 'ng generate component', kao što je prethodno opisano.

3.2.2 Moduli

Moduli su klase u Angularu koje grupiraju povezane komponente, servise i druge funkcionalnosti. Glavni modul aplikacije je 'AppModule', koji se nalazi u datoteci 'app.module.ts'. Unutar ove datoteke registriraju se sve komponente, direktive, servisi i drugi moduli koji su potrebni aplikaciji.

Primjer 1. *Primjer strukture app.module.ts datoteke:*

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ChatComponent } from './chat/chat.component';
import { HttpClientModule } from '@angular/common/http';
import { AvatarModule } from 'primeng/avatar';
import { AvatarGroupModule } from 'primeng/avatargroup';

@NgModule({
  declarations: [
    AppComponent,
    ChatComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    AppRoutingModule,
    AvatarModule,
    AvatarGroupModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Ovaj kod prikazuje kako se konfigurira osnovni modul aplikacije u Angularu.

- **declarations**: ovdje se registriraju komponente koje će biti korištene u aplikaciji. U ovom slučaju, to su `AppComponent` i `ChatComponent`.
- **imports**: ovdje se dodaju svi moduli koji su potrebni aplikaciji, uključujući osnovne Angular module i dodatne module kao što su `HttpClientModule` i moduli iz `primeng`.
- **bootstrap**: ovdje se specificira početna komponenta `AppComponent`, koja se učitava kada se aplikacija pokrene.

Detaljnije o komponentama, modulima, uvozima(imports) i inicijalizaciji (bootstrappingu) na [1].

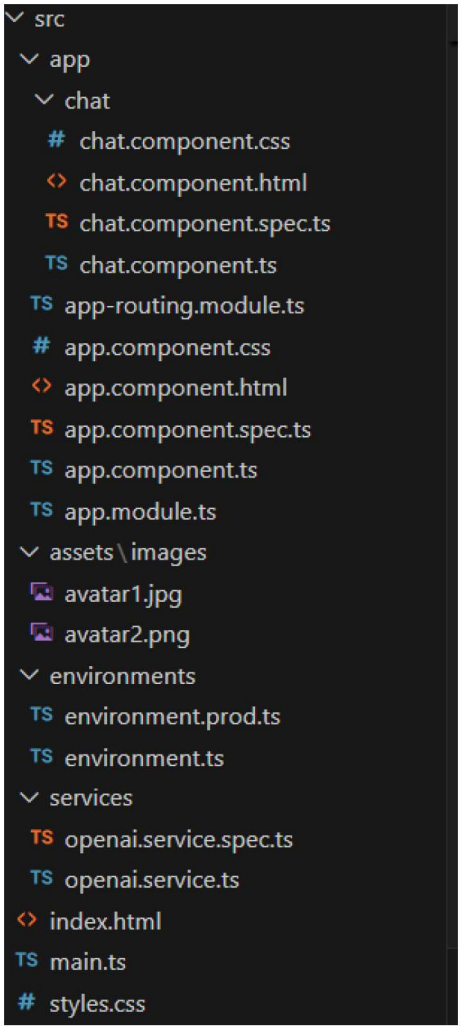
4 | Implementacija

4.1 Struktura projekta

Struktura projekta je ključna za organizaciju i održavanje koda, posebno kod složenijih aplikacija. U ovom projektu, glavne direktorije i datoteke organizirali smo unutar direktorija `src`, kako slijedi:

- `src/app/`: glavni direktorij za sve Angular komponente.
  - `chat/`: sadrži datoteke koje čine `ChatComponent`, uključujući HTML, CSS, TypeScript, (`chat.component.html`, `chat.component.css`, `chat.component.ts`, `chat.component.spec.ts`).
  - `app-routing.module.ts`: modul za definiranje ruta unutar aplikacije.
  - `app.module.ts`: glavni modul aplikacije, gdje se deklariraju sve komponente i servisi.
- `src/services/`: direktorij za Angular servise koji upravljaju logikom aplikacije i komunikacijom s vanjskim API-ima.
  - `openai.service.ts`: servis za komunikaciju s OpenAI API-jem, koji upravlja slanjem upita i primanjem odgovora od API-ja.
- `src/environments/`: ovaj direktorij sadrži konfiguracijske datoteke za različita okruženja (npr. razvojno i produkcijsko). U datotekama `environment.ts` i `environment.prod.ts` nalazi se API ključ koji se koristi za komunikaciju s OpenAI API-jem.
- `src/assets/images`: direktorij za slike.

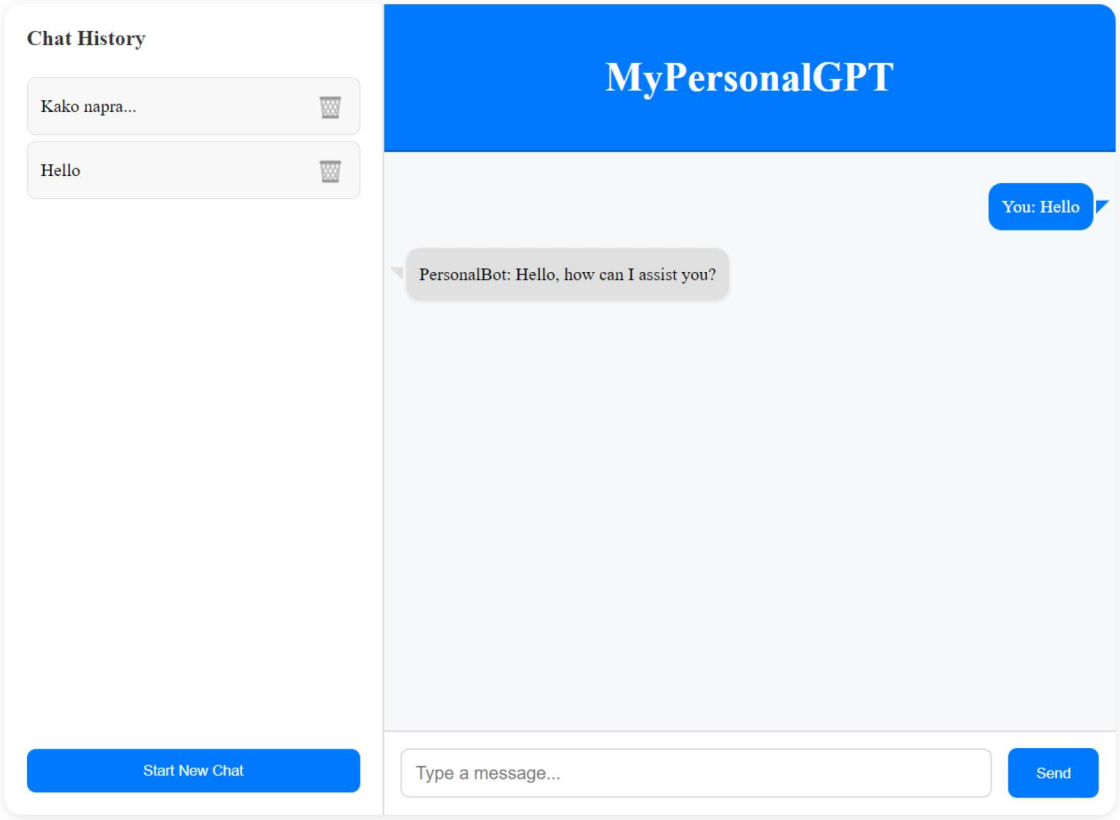
Ovakva struktura omogućuje jasnu organizaciju koda i olakšava rad na aplikaciji.



Slika 4.1: Struktura projekta u Visual Studio Code-u

4.2 Opis i izgled korisničkog sučelja

Sučelje se sastoji od nekoliko ključnih komponenti koje su implementirane pomoću HTML-a i CSS-a unutar Angular aplikacije. Korisničko sučelje naše aplikacije uključuje: glavni prozor za prikaz poruka, traku za unos poruka, bočnu traku za pregled prethodnih razgovora te gumb za stvaranje novog razgovora.



Slika 4.2: Prikaz korisničkog sučelja

4.3 Razvoj korisničkog sučelja

4.3.1 HTML struktura

HTML struktura naše aplikacije osmišljena je tako da omogući jasnu organizaciju elemenata i intuitivnu interakciju korisnika. Kod je podijeljen u nekoliko glavnih blokova, svaki s određenom funkcijom.

Prvo, imamo bočnu traku koja služi kao navigacijski dio za pregled i upravljanje prethodnih razgovora. Ovaj blok koristi Angularovu direktivu `*ngFor` kako bi dinamički generirao popis razgovora, omogućujući korisnicima pregled i odabir pojedinačnih razgovora. Također, unutar bočne trake postoji gumb za pokretanje novog razgovora, koji aktivira odgovarajuću funkcionalnost unutar aplikacije.

Glavni dio sučelja je prozor za razgovor, gdje se prikazuju sve razmijenjene poruke korisnika i računala unutar odabranog razgovora. Unutar ovog bloka, poruke se stiliziraju i raspoređuju ovisno o tome tko ih je poslao (korisnik ili aplikacija), koristeći `ngClass` za dinamičko pridruživanje klasa.

Unos novih poruka omogućuje se putem tekstualnog polja koje koristi Angularovu `ngModel` direktivu za dvosmjerno vezivanje podataka. Ovo osigurava da se unosi korisnika odmah reflektiraju u aplikaciji, a slanje poruka omogućuje se pritiskom na tipku Enter ili putem gumba "Send".

Na kraju, aplikacija također uključuje skočni prozor za potvrdu brisanja pojedi-

nog razgovora, čime se osigurava sigurno uklanjanje razgovora. Ova funkcionalnost implementirana je pomoću Angularove `*ngIf` direktive [5] koja prikazuje ili skriva prozor ovisno o stanju aplikacije.

Ovakva struktura omogućuje aplikaciji da bude funkcionalna i jednostavna za korištenje, uz intuitivnu navigaciju i interakciju.

Primjer 2.

```
<div class="chat-input">
  <input type="text" [(ngModel)]="newMessage" (keyup.enter)="
    sendMessage()" placeholder="Type a message...">
  <button (click)="sendMessage()">Send</button>
</div>
```

Listing 4.1: HTML kod za unos i slanje poruke

4.3.2 CSS stilizacija

CSS stilizacija aplikacije osmišljena je kako bi pružila intuitivno i estetski ugodno korisničko iskustvo. Struktura CSS-a organizirana je u nekoliko ključnih dijelova koji odgovaraju glavnim komponentama korisničkog sučelja.

- **Opći stilovi**: Definiramo osnovne stilove za cijelu aplikaciju, uključujući font, boje pozadine, marginu i padding.
- **Kontejner aplikacije**: Glavni kontejner `chat-app` koristi `flexbox` za raspoređivanje elemenata u aplikaciji. Postavljanjem `max-width` i `margin: auto`, centriramo aplikaciju na stranici. `box-shadow` i `border-radius` dodaju dubinu i zaobljene kutove, čineći sučelje modernim i preglednim.
- **Bočna traka**: Bočna traka sadrži popis prethodnih razgovora. Koristimo `flex-direction: column` za vertikalno raspoređivanje elemenata. Stilizacija popisa uključuje prijelaze za glatke promjene boje pri hover efektu, što doprinosi boljoj interakciji korisnika sa sučeljem.
- **Glavni prozor za razgovor**: Glavni prozor uključuje zaglavlje, prostor za poruke i unos novih poruka. Poruke su stilizirane tako da se poruke korisnika i računala razlikuju u boji i poziciji, što omogućava lako razlikovanje tko je poslao koju poruku.
- **Potvrda brisanja**: Skočni prozor za potvrdu brisanja razgovora stiliziran je da se pojavi u centru ekrana, s jasnim vizualnim naglascima za potvrdu ili otkazivanje akcije.

Primjer 3. *Uređivanje elementa koji sadrži polje za unos poruke i gumb za slanje poruka.*

```
.chat-input {
  display: flex;
  padding: 15px;
  background-color: #ffffff;
  border-top: 2px solid #e0e0e0;
}

.chat-input input {
  flex: 1;
  padding: 12px;
  border: 2px solid #e0e0e0;
  border-radius: 8px;
  font-size: 1rem;
}

.chat-input button {
  margin-left: 15px;
  padding: 12px 25px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: background-color 0.3s, transform 0.2s;
}

.chat-input button:hover {
  background-color: #0056b3;
  transform: scale(1.02);
}
```

4.3.3 Implementacija funkcionalnosti aplikacije

Da bismo dodali funkcionalnost aplikaciji, potrebno je implementirati određene funkcije u TypeScriptu koje će omogućiti osnovne interakcije unutar aplikacije. Ako smo u HTML-u napravili odjeljak za slanje poruka, moramo kreirati i funkciju koja će se pobrinuti za slanje tih poruka, koja će se pozivati na pritisak gumba ili tipke *Enter*.

Također, svaki događaj unutar aplikacije, kao što je započinjanje novog razgovora ili odabir nekog drugog postojećeg, mora biti podržan odgovarajućim TypeScript funkcijama. Na primjer, potrebno je implementirati funkciju koja će se aktivirati kada korisnik pritisne gumb za pokretanje novog razgovora. Ta funkcija treba otvoriti novu karticu razgovora i pripremiti sučelje za novi set poruka.

- **Slanje poruka** (`sendMessage`)**:** Funkcija `sendMessage` omogućuje korisnicima slanje poruka unutar aplikacije. Prvo, funkcija provjerava je li unesena poruka prazna kako bi se izbjeglo slanje praznih poruka. Ako je poruka valjana, ona se dodaje u trenutni razgovor. Ako je ovo prva poruka u tom ra-

zovoru, radi lakšeg raspoznavanja, funkcija automatski postavlja ime razgovora koristeći prvih nekoliko znakova te poruke. Nakon što se poruka pošalje, polje za unos se ponovno pokreće, a korisničko sučelje osvježava kako bi prikazalo novu poruku. Na kraju, odgovarajuća funkcija simulira odgovor računala, čime se korisniku pruža osjećaj razgovora.

- **Započinjanje novog razgovora** (`startNewChat`)**:** Funkcija `startNewChat` omogućuje korisniku započinjanje potpuno novog razgovora. Prije nego što se pokrene novi razgovor, trenutni razgovor se sprema kako bi se osiguralo da su sve poruke sačuvane. Funkcija zatim kreira novi razgovor i postavlja ga kao aktivni, ponovno pokrećući prikaz poruka kako bi bio spreman za novi unos. Ova funkcionalnost je ključna za aplikacije koje podržavaju višestruke razgovore.

- **Brisanje razgovora** (`deleteChatSession`)**:** Kada korisnik želi obrisati određeni razgovor, funkcija `confirmDeleteChatSession` prikazuje potvrdni dijalog kako bi se izbjegla slučajna brisanja. Ako korisnik potvrdi brisanje, funkcija `deleteChatSession` uklanja odabrani razgovor, ažurira trenutni prikaz i osigurava da je korisnik prebačen na novu aktivni razgovor, ako je dostupan.

Ovo su neke od funkcija koje čine osnovu za izradu interaktivnog konverzacijskog agenta, omogućujući korisnicima da jednostavno šalju i primaju poruke, prebacuju se između različitih razgovora i upravljaju njima.

Primjer 4. *Funkcija `sendMessage` koja se brine za slanje upita.*

```
sendMessage() {
    if (this.newMessage.trim()) {
        if (this.messages.length === 0) {
            const sessionName = this.newMessage.length > 10 ? this.newMessage.substring(0, 10) + '...' : this.newMessage;
            this.chatSessions[this.currentSessionIndex].name = sessionName;
        }
        this.messages.push({ user: 'You', text: this.newMessage });
        const userMessage = this.newMessage;
        this.newMessage = '';
        this.simulateBotResponse(userMessage);
    }
}
```

4.3.4 Implementacija načina odgovaranja

Kako bi konverzacijski agent mogao odgovarati na korisničke poruke, potrebno je implementirati funkcionalnost koja omogućuje integraciju s API-jem OpenAI-a. Ovo omogućuje aplikaciji slanje korisničkih poruka na obradu te primanje generiranih odgovora koje se prikazuju korisniku.

API ključ: Za komunikaciju s OpenAI-jem koristi se API ključ koji se dodjeljuje prilikom registracije na OpenAI platformu. Ovaj ključ je jedinstven za svakog korisnika i mora se pažljivo čuvati jer omogućuje pristup resursima na OpenAI platformi. U Angular aplikaciji, API ključ se obično pohranjuje u konfiguracijsku datoteku ili kao varijabla okoliša, čime se osigurava njegova zaštita.

Pozivanje OpenAI API-ja: Implementacija započinje integracijom OpenAI servisa unutar aplikacije. Kreiramo zasebnu uslugu (service) unutar Angular aplikacije koja će se koristiti za slanje korisničkih poruka prema OpenAI API-ju i primanje odgovora. Kada korisnik pošalje poruku, funkcija `sendMessage` poziva servis koji šalje tu poruku prema OpenAI API-ju koristeći HTTP POST zahtjev.

Primanje odgovora: Nakon što OpenAI API obradi korisničku poruku, vraća generirani odgovor koji se zatim prikazuje korisniku u obliku poruke računala. Ovaj proces omogućuje dinamičku i interaktivnu komunikaciju unutar aplikacije.

Error handling: Važno je napomenuti da u slučaju problema s mrežom ili API-jem, aplikacija treba biti sposobna obraditi te pogreške i obavijestiti korisnika o problemu. Ovo se najčešće postiže slanjem nekakve generičke poruke koja sugerira da je došlo do problema.

Primjer 5. *Funkcija `simulateBotResponse`*

```
simulateBotResponse(userMessage: string) {
    this.openaiService.sendMessage(userMessage).subscribe(response => {
        const botMessage = response.choices[0].message.content;
        this.messages.push({ user: 'PersonalBot', text: botMessage });
    }, error => {
        console.error('Error fetching response:', error);
        this.messages.push({ user: 'PersonalBot', text: 'Sorry, there was an error fetching the response.' });
    });
}
```

U ovom primjeru, funkcija `simulateBotResponse` koristi uslugu `openaiService` kako bi poslala korisničku poruku prema OpenAI API-ju. Nakon što se dobije odgovor, taj odgovor se dodaje u niz poruka i prikazuje u sučelju. U slučaju pogreške, korisniku se prikazuje poruka o grešci.

Ovakva implementacija omogućuje stvaranje interaktivnog konverzacijskog agenta koji može komunicirati s korisnikom na temelju upita, koristeći napredne modele generiranja teksta koje pruža OpenAI.

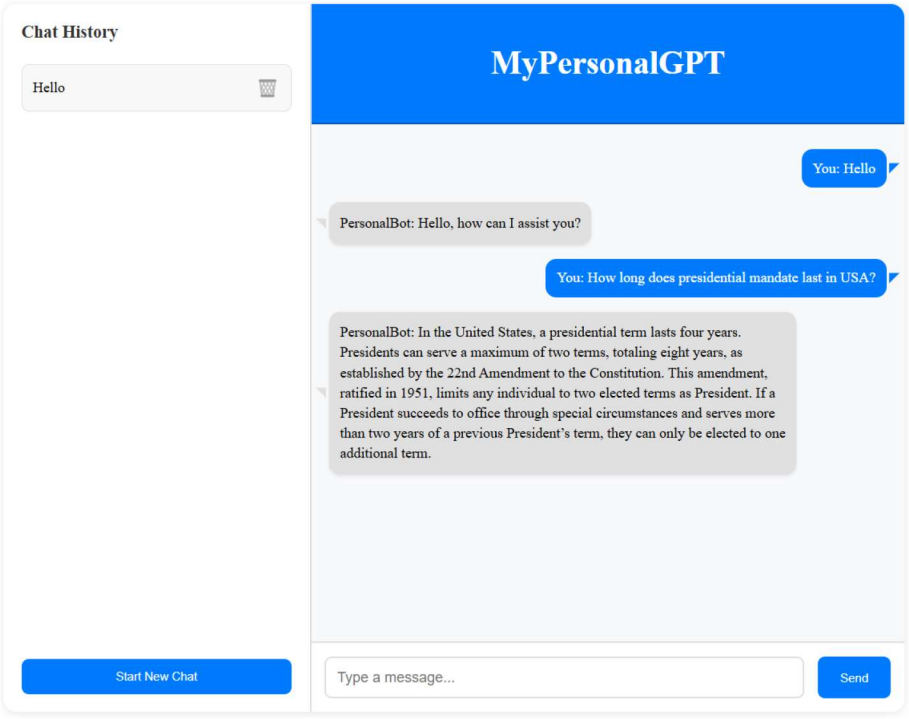
5 | Pregled funkcionalnosti aplikacije

U ovom poglavlju ćemo prikazati neke od funkcionalnosti naše aplikacije..

5.1 Generiranje odgovora

Jedna od osnovnih funkcionalnosti aplikacije je sposobnost generiranja odgovora na korisničke upite. U ovom dijelu prikazujemo primjer kako aplikacija odgovara na korisnički upit.

Primjer 6. *Generiranje odgovora za postavljeni upit.*



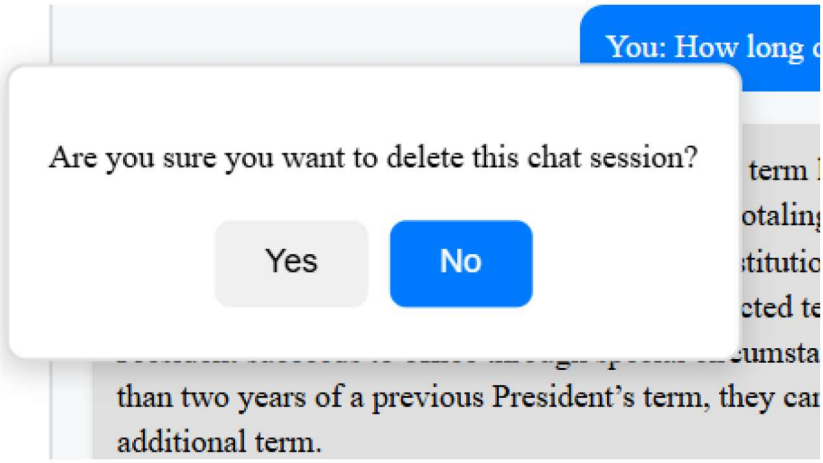
Slika 5.1: Primjer odgovora aplikacije

Na slici iznad prikazan je dio korisničkog sučelja gdje se vidi kako aplikacija reagira na unesene poruke. Kao što je prikazano, aplikacija generira odgovore u

stvarnom vremenu, omogućujući korisnicima interakciju i dobivanje informacija brzo i efikasno.

5.2 Skočni prozor za brisanje razgovora

Kako bismo omogućili korisnicima upravljanje razgovorima, implementirali smo dijalog za potvrdu brisanja razgovora. Ova funkcionalnost omogućava korisnicima da obrišu nepoželjne ili zastarjele razgovore uz potvrdu kako bi se spriječilo slučajno brisanje važnih podataka.



Slika 5.2: Skočni prozor za potvrdu brisanja razgovora

Na slici iznad prikazana je funkcionalnost dijaloga koji se pojavljuje kada korisnik želi obrisati razgovor. Kao što vidimo, samim skočnim prozorom te različitim bojama za odgovore 'Yes' i 'No', stvaramo sigurniji pristup brisanju.

6 | Zaključak

U ovom radu smo istražili tehnologije i pristupe koji su osnova modernog razvoja web aplikacija, s posebnim fokusom na Angular i TypeScript. Kroz analizu ovih tehnologija, postalo je jasno da one značajno unapređuju proces razvoja, pružajući odlične alate za izgradnju složenih aplikacija.

Angular, kao jedan od najpopularnijih razvojnih okvira, nudi brojne prednosti, uključujući dvostruko povezivanje podataka, komponente, dependency injection i direktive. Ove značajke omogućavaju programerima da kreiraju dinamičke i interaktivne web aplikacije uz visoku modularnost i održivost. Angularov sustav za rutiranje i njegova podrška za TypeScript čine ga odličnim izborom za razvoj velikih aplikacija.

S druge strane, TypeScript se pokazuje kao izvrstan dodatak JavaScript-u, pružajući statičko tipiziranje i druge napredne karakteristike koje poboljšavaju kvalitetu i održivost koda. U kombinaciji s Angularom, TypeScript omogućava pisanje čitljivijeg, strukturiranijeg i pouzdanijeg koda, što doprinosi uspješnom razvoju kompleksnih aplikacija. Statističko tipiziranje i podrška za klase i sučelja su ključni za rad u većim projektima.

Kao što je pokazano kroz ovaj rad, jasno je da kombinacija Angulara i TypeScripta pruža snažnu platformu za razvoj web aplikacija.

Literatura

[1] A. Hussain, *Angular from Theory to Practice*.

[2] *How to create an Angular project from scratch*, dostupno na
`https://www.geeksforgeeks.org/how-to-create-an-angular-project-from-scratch/`

[3] *Angular documentation*, dostupno na
`https://v17.angular.io/docs`

[4] *Typescript Documentation*, dostupno na
`https://www.typescriptlang.org/docs/`

[5] *Angular ngIf*, dostupno na
`https://blog.angular-university.io/angular-ngif/`

Sažetak

U ovom radu istražujemo upotrebu Angular frameworka i TypeScript jezika u razvoju chat aplikacije. Ovaj rad analizira njihove značajke, pruža uvid u njihove prednosti i izazove, te demonstrira praktičnu primjenu. Rezultati pokazuju da kombinacija Angulara i TypeScripta značajno unapređuje razvojne procese i kvalitetu aplikacija.

Ključne riječi

Angular, TypeScript, konverzacijski agent, HTML, CSS, web aplikacija

Development of a chatbot application using Angular

Summary

In this paper, we explore the use of the Angular framework and the TypeScript language in the development of a chat application. This work analyzes their features, provides insights into their advantages and challenges, and demonstrates their practical application. The results show that the combination of Angular and TypeScript significantly enhances development processes and the quality of applications.

Keywords

Angular, Typescript, chatbot, HTML, CSS, web application