SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 872

**Uporaba alata GPT-3 u obrazovanju**Ivan Terzić

Zagreb, lipanj 2023.

Zahvaljujem obitelji i prijateljima koji su mi puno pomogli i olakšali studij te mentoru izv. prof. dr. sc. Ivici Botičkom.

Contents

[Uvod 1](#_Toc135295146)

[*1.* *Teorijska pozadina* 2](#_Toc135295147)

[1.1. Umjetna inteligencija 2](#_Toc135295148)

[*1.2.* *Chatbot* 2](#_Toc135295149)

[1.3. Obrada prirodnog jezika 3](#_Toc135295150)

[1.4. Web aplikacija **Error! Bookmark not defined.**](#_Toc135295151)

[1.5. Aplikacijsko programsko sučelje 4](#_Toc135295152)

[2. Korištene tehnologije 5](#_Toc135295153)

[2.1. JavaScript 5](#_Toc135295154)

[2.1.1. Iznimke 5](#_Toc135295155)

[2.1.2. Asinkronost 6](#_Toc135295156)

[*2.1.3.* Ključne riječi *async* i *await* 7](#_Toc135295157)

[2.2. React 9](#_Toc135295158)

[2.3. Node.js 9](#_Toc135295159)

[2.4. Ostale web tehnologije 10](#_Toc135295160)

[2.4.1. Bootstrap 11](#_Toc135295161)

[2.4.2. Axios proxy 12](#_Toc135295162)

[2.5. PostgreSQL 12](#_Toc135295163)

[2.6. Aplikacijska programska sučelja **Error! Bookmark not defined.**](#_Toc135295164)

[3. Arhitektura sustava 14](#_Toc135295165)

[*Zaključak* 20](#_Toc135295166)

[*Literatura* 21](#_Toc135295167)

[*Sažetak* 23](#_Toc135295168)

[*Summary* 24](#_Toc135295169)

[*Skraćenice* **Error! Bookmark not defined.**](#_Toc135295170)

[*Privitak* **Error! Bookmark not defined.**](#_Toc135295171)

# Uvod

U današnjem smo vremenu svjedoci ubrzanog razvoja umjetne inteligencije, jezičnih modela i, posljedično, raznih generativnih alata kao što su generatori slika, zvuka, govora ili teksta. Kao daleko najpopularniji alat ističe se ChatGPT, jezični model tvrtke OpenAI, koji je dobio veliku medijsku pozornost i popularnost među širom javnosti zbog svojih mogućnosti i stila odgovora koji izrazito nalikuje ljudskom.

Kao i sa svakom novom tehnologijom, tako i s ChatGPT-em, ljudi nastoje pronaći primjenu koja će im olakšati svakodnevne poslove ili ih potpuno automatizirati. OpenAI je, baš za tu svrhu, ChatGPT, odnosno, preciznije, GPT-3.5 model koji se nalazi iza ChatGPT-a, stavio na raspolaganje programerima preko posebnog aplikacijskog programskog sučelja (engl. application programming interface, API) [1] [2].

No, osim olakšavanja nekih poslova, ova nam tehnologija omogućuje puno više, pogotovo u području računarstva gdje umjetna inteligencija, a tako i jezični modeli OpenAI-a, imaju širok raspon primjena i načina na koje se mogu koristiti i ugraditi u sustave. Jedno od važnijih područja primjene te tehnologije je obrazovanje. Tehnologija se u obrazovanju može iskoristiti na razne načine, npr. zadatak pronalaska pojedinih informacija, no potencijalno najbolja primjena bi bila u izradi chatbota koji bi učenicima omogućavao interaktivno učenje kroz razgovor s osobama relevantnim za pojedino područje.

Cilj je ovog rada bio izrada aplikacije koja bi omogućila upravo to: čavrljanje (engl. chat) s pojedinim osobama. Aplikacija podržava i interaktivnu komunikaciju u smislu sinteze govora u tekst (engl. text to speech) i teksta u govor (engl. speech to text) i funkcionalnost čavrljanja dvije osobe čije poruke generira isključivo GPT-3.5 model.

# Teorijska pozadina

## Umjetna inteligencija

Umjetna se inteligencija definira kao dio računarske znanosti koja se bavi razvojem i izučavanjem sustava koji pokazuju neki oblik inteligencije te se ti sustavi koriste za efikasnije obavljanje poslova u domeni računarstva i izvan nje [3]. Dok definicije i kriteriji same računalne inteligencije među ljudima i područjima nisu ujednačeni, generalno je u javnosti prihvaćena činjenica da se računalni sustav smatra inteligentnim kad pokazuje neka svojstva koja inače pokazuju ljudi, npr. prepoznavanje govora, fotografija ili sinteza teksta [3]. Postoji puno različitih sustava umjetne inteligencije koji se razlikuju po svojim funkcijama, a neki od njih su:

* ekspertni sustavi
* sustavi za obradu prirodnog jezika
* sustavi za prepoznavanje govora
* sustavi za računalni vid
* *chatboti.*

## Chatbot

*Chatbot* je računalni sustav ili program koji koristi umjetnu inteligenciju i obradu prirodnog jezika kako bi razumio upite korisnika i kreirao odgovor koji u najvećoj mogućoj mjeri odgovara na upit i sliči ljudskom prirodnom jeziku [4]. Osim same mogućnosti obrade prirodnog jezika, *chatboti* moraju imati i bazu znanja na temelju koje će uzimati elemente jezika (riječi, rečenice) iz kojih će kreirati odgovore. Najpoznatiji primjeri chatbota su:

* ChatGPT
* Siri
* Cortana
* LaMDA.

Rapidnim razvojem umjetne inteligencije *chatboti* postaju sve bolji i sintetiziraju tekst sve sličniji ljudskom govoru. Povećanjem računalnih resursa koji stoje iza tih sustava se u model sustava može staviti sve više i više podataka pa tako dobiti i tekst koji, osim izuzetne sličnosti ljudskom govoru, posjeduje i iznimnu točnost podataka koje koristi u generiranom tekstu. Nadalje, kako bi konverzacija između sustava i korisnika bila što realnija, *chatboti* imaju sposobnost i nastavljanja razgovora ili postavljanja pitanja korisniku te predlaganja potencijalnih pitanja koja korisnik može postaviti sustavu. Sve te značajke čine ovaj tip sustava umjetne inteligencije izuzetno moćnim alatom, a pogotovo za programere koji ih korištenjem programskih sučelja mogu implementirati u svoje sustave.

## Obrada prirodnog jezika

Obrada prirodnog jezika (engl. *natural language processing*, NLP) je sposobnost pojedinog sustava ili programa da razumije ljudski jezik u pisanom ili izgovorenom obliku. Obrada prirodnog jezika se koristi u raznim područjima računarske znanosti i sustavima [5]. Obrada prirodnog jezika svoje korijene vuče iz područja lingvistike, a u samoj implementaciji u računalnim sustavima koristi se umjetna inteligencija. Neke od primjena sustava obrade prirodnog jezika su:

* Email filteri
* Tražilice (engl. *search engines*)
* Predviđanje teksta
* Analiza teksta.

## Organizacija web sustava

Web aplikacija je interaktivan program kojem korisnik pristupa preko internetske veze. Aplikacija se nalazi na nekom jedinstvenom URL-u (*Uniform Resource Locator*). Web aplikacije imaju izuzetno široko područje primjena, sve od aplikacija za čitanje i slanje e-mailova, internih stranica neke tvrtke ili ustanove pa do zabavih sadržaja ili igara. Web aplikacije se često sastoje od nekoliko različitih dijelova, a to su tipično:

* Klijentski dio (engl. *clientside, frontend*)
* Poslužiteljski dio (engl. *server, serverside, backend*)
* Baza podataka (engl. *database*)

Danas su web aplikacije široko raširene i vrlo popularne zbog činjenice da im se lako može pristupiti i da ima iznenađujuće velik broj programskih jezika i njihovih radnih okvira (eng. *workframe*) u pomoću kojih se aplikacija može izraditi. Zbog široke dostupnosti dokumentacije i vodiča za pojedine tehnologije, moguće je raditi s puno različitih tehnologija.

### Aplikacijska programska sučelja

Aplikacijsko programsko sučelje je skup definiranih rutina koje omogućavaju korisnicima pristup funkcionalnostima koje određeno sučelje nudi. Ovakva sučelja omogućavaju nezavisnost i odvajanje dijelova sustava, npr. klijentskog i poslužiteljskog dijela web aplikacije. Ta nam nezavisnost omogućuje da dijelove sustava pišemo u različitim programskim jezicima ili koristeći druge tehnologije i radne okvire, a da oni svejedno mogu komunicirati i razmjenjivati podatke.

Kao u navedenom primjeru, aplikacijska programska sučelja mogu se koristiti za povezivanje različitih dijelova sustava, ali i za korištenje nekih vanjskih usluga, kao što je u ovom radu korišteno sučelje za pristup uslugama OpenAI-evih jezičnih modela i sučelje za sintezu govora iz teksta.

# Korištene tehnologije

## JavaScript

JavaScript[[1]](#footnote-1) je skriptni programski jezik koji je najzastupljeniji u domeni web programiranja i izrade web aplikacija. On se parsira i odmah izvodi. To je proceduralan jezik koji podržava objektno orijentiranu paradigmu i najčešće se izvodi u samom pregledniku kod pokretanja web aplikacija. Danas je JavaScript jedan od najpopularnijih i najraširenijih programskih jezika koji se koristi i na klijentskoj i poslužiteljskoj strani razvoja web aplikacija. Zajednica programera koja koristi JavaScript je velika, stoga postoji i puno različitih radnih okvira za izradu aplikacija.

JavaScript kompleksan jezik s puno specifičnih aspekata koji su bitni programerima web aplikacija, ali njegove specifičnosti i karakteristike nisu fokus ovog rada, stoga će biti opisani detalji koji su bili bitni za realizaciju izrađene web aplikacije.

### Iznimke

Iznimke (engl. *Exception*) su događaji koji se pojavljuju kod izvođenja kôda koji ometaju izvođenje programa. Iznimke mogu nastati na mnogo načina, npr. greškom programera, neslaganjem tipova podataka pa čak i nedostatkom memorije.

U JavaScriptu se iznimkama upravlja, slično kao i u drugim programskim jezicima, posebnim blokovima koji imaju nazive *try, catch* i *finally*.

U *try* blok se stavlja kôd za koji postoji mogućnost da generira iznimku. U trenutku generiranja iznimke, ovaj blok prestaje s izvođenjem.

U *catch* blok se piše kôd koji želimo da se izvrši ako dođe do iznimke. Ovaj blok prima i argument (kao funkcija) objekta za grešku tipa *Error* koji sadrži korisne informacije za obradu i analizu greške, npr. vrsta i poruka greške. Takvi se objekti mogu i stvarati u kôdu.

*Finally* blok sadrži kôd koji se izvršava u svakom slučaju, dogodila se iznimka ili ne.

async function fillData(){

try {

let response = await backend.get("/getvoice/voices");

setVoices(response.data.map(voice => ({

value : voice.name, language : voice.language, sample :voice.sample, gender : voice.gender

})));

let categories = await backend.get("/categories");

setCategories(categories.data.map(c => ({value : c.id, name : c.name})))

} catch (e) {

alert(e);

} finally {

console.log("Data fetch done.")

}

}

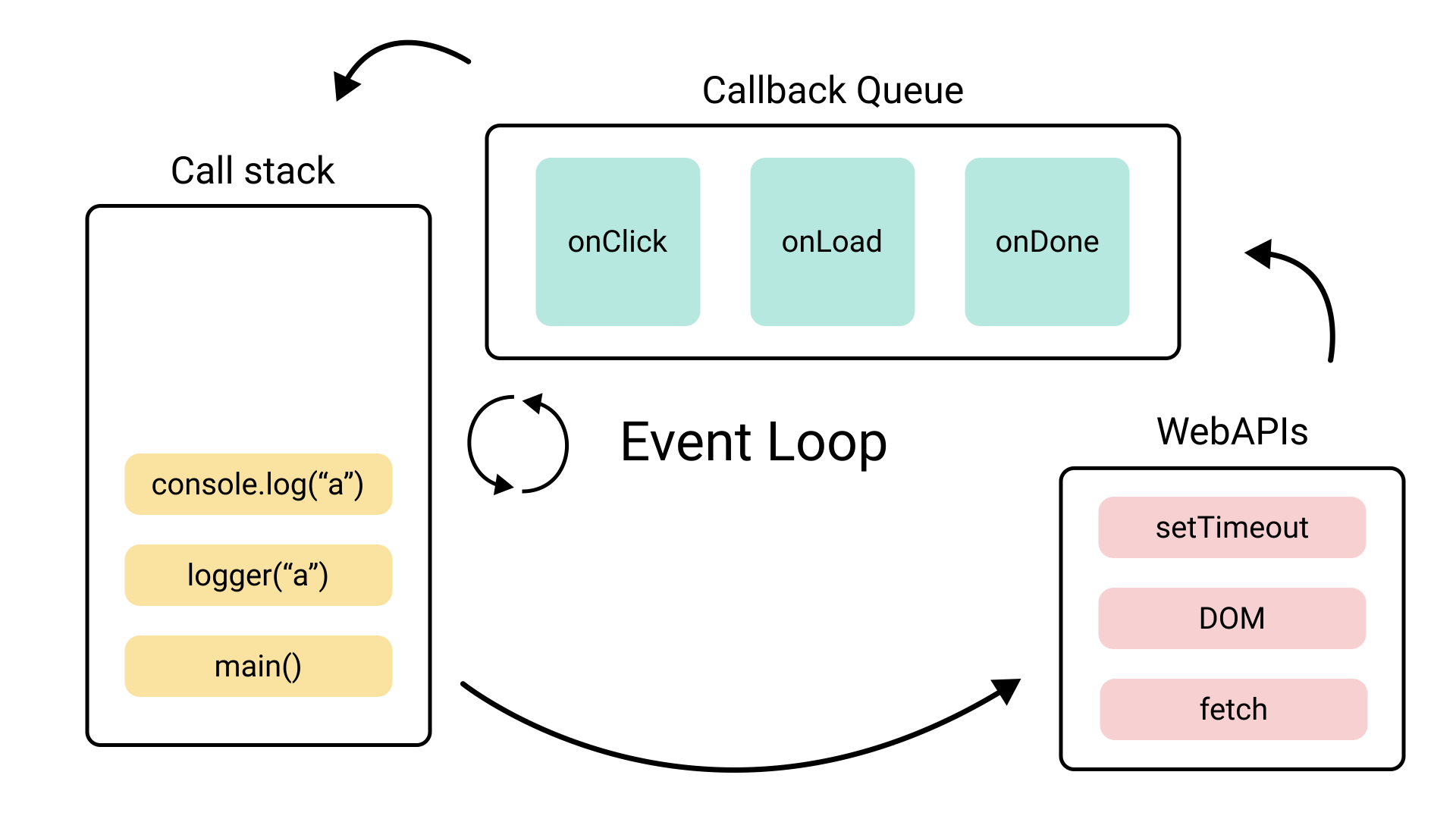
Kôd 2.1 Primjer dohvaćanja podataka s try, catch i finally blokovima

### Asinkronost

JavaScript je izvorno osmišljen kao jednodretven jezik u smislu da ne postoji opcija da se kreira nova dretva u pojedinom procesu. Naredbe se izvršavaju slijedno, sinkrono [6]. Četo je na webu potrebno dohvaćati resurse, ažurirati DOM (*Document Object Model*) ili čekati neki događaj na DOM-u, npr. klik gumba, što bi blokiralo izvođenje danjih naredbi i tako učinilo kôd sporim.

Kao rješenje, JavaScript koristi petlju događaja (engl. *event loop*), red čekanja poruka (engl. *message queue*) i *web APIs*. Sve se naredbe spremne za izvršavanje stavljaju na sistemski stog. Ako se naredba treba izvršiti asinkrono, ona se preusmjerava na *web APIs.* On preuzima naredbu i izvršava ju. Nakon njenog izvršavanja se njena povratna funkcija (engl. *callback*) stavlja u red čekanja poruka. Povratna funkcija je funkcija koja sadrži kôd koji želimo da se izvrši nakon uspješnog izvršavanja asinkrone naredbe. Petlja događaja konstantno provjerava postoji li u redu čekanja povratna funkcija koju treba izvršiti te ju stavlja na sistemski stog s kojeg se ta povratna funkcija izvodi sinkrono [6]. Dakle, iako ne postoji višedretvenost kao takva, dio posla koji treba odraditi asinkrono može se prebaciti na *web APIs.*

Asinkrono izvođenje može se ostvariti korištenjem obećanja (engl. *Promises*) ili ključnih riječi *async* i *await.*



Slika 2.1 Prikaz asinkronog izvođenja [7]

### Ključne riječi *async* i *await*

*Async* i *await* su ključne riječi programskog jezika JavaScript koje koristimo kako bismo efektivno mogli izvoditi asinkroni kôd sinkrono, slijedno. U ovom su radu ove ključne riječi često korištene jer se dohvaća puno podataka i čeka na puno odgovora s raznih sučelja.

Označavanje funkcije ključnom riječi *async* znači da će se u funkciji izvoditi neki asinkroni kôd.

Ključna riječ *await* može se koristiti isključivo unutar funkcije označene s *async*. Njome se čeka izvršavanje pojedine asinkrone operacije, ta se operacija u kontekstu asinkrone funkcije izvršava sinkrono, dakle čeka se izvršavanje asinkrone naredbe kako bi se moglo dalje nastaviti s izvršavanjem naredbi funkcije.

Ove su ključne riječi ravnopravne obećanjima, ali njihovim korištenjem dobiva se čitljiviji i jednostavniji kôd.

async function getAudio(){

const request = {

"voice": personaObj.voice,

"content": [props.data],

};

let response

try {

response = await backend.post("/getvoice", request)

}

catch(e){

alert("An error has occured while fetching voices!")

return

}

isSpeaking = true

let a = new Audio(response.data)

a.onended = function(e){

clearInterval(interval);

isSpeaking = false

speakIdx = 0

setMouth(defaultMouth)

}

await a.play()

interval = setInterval(() => {

handleAnimation();

}, 60);

}

Kôd 2.2 Primjer async funkcije

„Kôd 2.2 Primjer async funkcije“ pokazuje kako se ključne riječi mogu koristiti za dohvat podataka i reprodukciju glasa.

## React

React[[2]](#footnote-2) je besplatna JavaScript knjižnica (engl. *library*) otvorenog kôda (engl. *open source*) koja se koristi za izradu klijentskog dijela web aplikacije. Danas je React jedna od najpopularnijih i najkorištenijih knjižnica za izradu web aplikacija. React se bazira na komponentama (engl. *components*) koje se mogu više puta koristiti kod izrade pa ubrzavaju vrijeme izrade aplikacije i čine kôd čitljivijim. Osim standardnih web aplikacija, React se često upotrebljava za izradu jednostraničnih web aplikacija (engl. *single-page web application*), ali i mobilnih aplikacija.

React koristi JSX (*JavaScript Syntax Extension*), ekstenziju JavaScript-a koja omogućava pisanje HTML-a (*HyperText Markup Language*) u Reactovim komponentama. Uz to, koristi i tzv. *Hooks* i stanja (engl. *states*) kako bi ostvario željene funkcionalnosti te mogao pratiti vrijednosti pojedinih varijablii na temelju njih ažurirati osnovni HTML stranice.

React koristi virtualni HTML dokument koji se mijenja, a onda se samo promjene prikazuju korisniku. Ne mora ažurirati prikaz cijele stranice, već samo onaj dio stranice koji se promijenio.

function ProfileAvatar(props) {

    return (

       <img src={props.image} alt={props.image} className = "avatar"/>

    );

}

export default ProfileAvatar;

Kôd 2.3 Jednostavan primjer React kôda

## Node.js

Node.js[[3]](#footnote-3) je poslužiteljsko okruženje otvorenog kôda za JavaScript koje se može pokrenuti na raznim operacijskim sustavima. U ovom se okuženju često izrađuju poslužiteljske strane web aplikacija.

U Node.js uključen je i npm[[4]](#footnote-4) (*Node Packet Manager*). To je upravitelj paketa za Java Script koji omogućava preuzimanje paketa. Paketi su različite kompleksnosti, od jednostavnih paketa koji provjeravaju je li broj neparan pa sve do samog Reacta. U ovom su radu korišteni razni paketi i na klijentskoj i poslužiteljskoj strani.

Jedan od korištenih paketa je i *express*[[5]](#footnote-5)koji je korišten za izradu poslužiteljskog dijela web aplikacije. Fleksibilnost i robusnost te lakoća korištenja glavne su karakteristike ovog radnog okvira. U sklopu *servera* korišten je i *express-validator* [[6]](#footnote-6) koji predstavlja posrednički softver (engl. *middleware*) koji služi za validaciju i sanitizaciju podataka koji se primaju na poslužitelj.

app.use('/edit', editRouter)

app.use('/log', logsRouter)

const PORT = 5300;

app.listen(PORT, () => {

console.log(`Server listening on ${PORT}`);

});

Kôd 2.4 Isječak poslužiteljskog kôda u expressu

## Ostale web tehnologije

Osim navedenih web tehnologija koje čine glavninu sustava i koje su bitne za njegov rad, korištene su i druge tehnologije koje su pomogle u obavljanju poslova kao što su stilizacija stranice.

Standardno za izradu web stranice, korišteni su i HTML i CSS (*Cascading Style Sheets*), HTML u velikoj mjeri u sklopu JSX kôda, a CSS kod izrade nekih specifičnih stilova elemenata na sučelju.

### Bootstrap

Bootstrap[[7]](#footnote-7) je radni okvir za razvoj korisničkog sučelja sadržaja na webu koji sadrži predloške (engl. *templates*) za uređivanje kontejnera, gumba, obrazaca i ostalih HTML elemenata. Bootstrapovi su predlošci oblikovani kao klase te su responzivni i laki za čitanje, što znači da je lako iz samog naziva klase zaključiti kakve ona modifikacije primjenjuje na element koji je njom označen. Trenutno je u upotrebi verzija 5, koja je izašla 2021. godine.

<label className={(props.name === radio ? "btn-light" : "btn-outline dark") + " btn mx-3 radio-button"} htmlFor={props.name}>

<div className='container-fluid d-flex flex-column align-items-center justify-content-center flex-wrap'>

<h6>{props.name}</h6>

<div><img className='radio-img' src={props.image} alt={props.name}></img></div>

</div>

</label>

Kôd 2.5 Primjer JSX kôda s Bootstrap klasama

### Axios proxy

Axios[[8]](#footnote-8) je knjižnica koja se ponaša kao posrednik kod slanja HTTP (*Hypertext Transfer Protocol*) zahtjeva. Preuzima se i instalira kao npm paket. Korištenjem Axiosa se mogu definirati specifični parametri zahtjeva i mogu se ponovo koristiti.

const {API\_KEY} = require('../data/voice\_api\_key.js')

const {USER} = require('../data/voice\_user.js')

let voiceRequest = axios.create({

headers: {

"Authorization": API\_KEY,

"X-User-ID": USER,

'Content-Type': 'application/json'

}

}

);

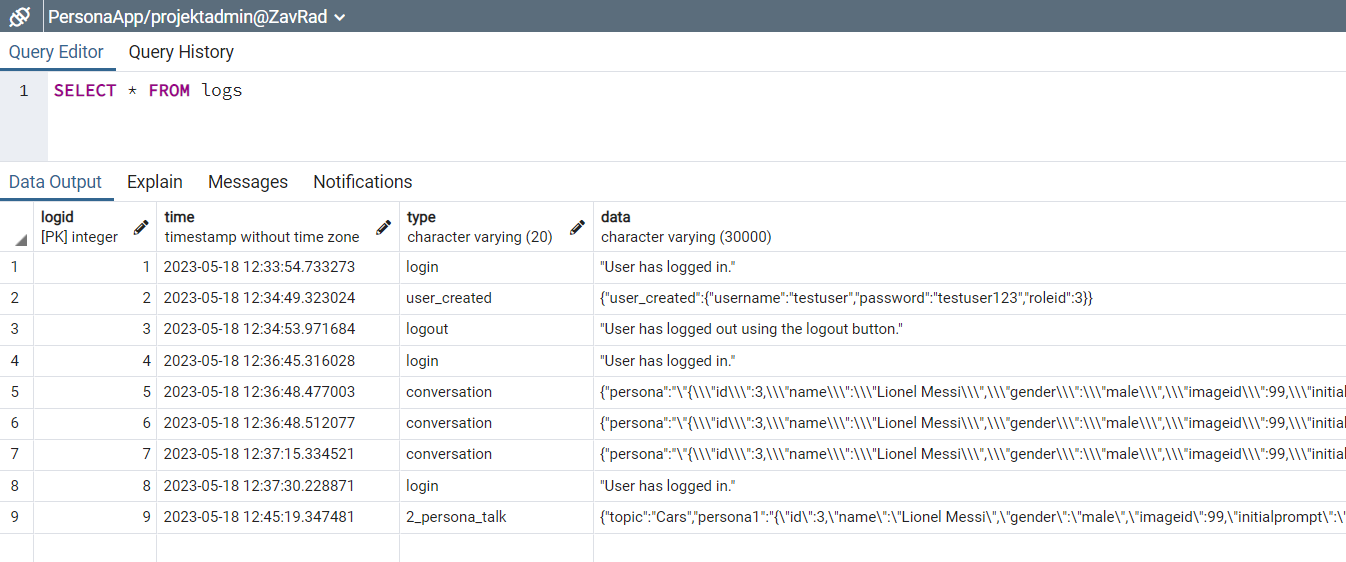
Kôd 2.6 Definiranje instance Axiosa

response = await voiceRequest.get("https://play.ht/api/v1/getVoices")

Kôd 2.7 Slanje zahtjeva preko definirane instance Axiosa

## PostgreSQL

PostgreSQL[[9]](#footnote-9) je sustav za upravljanje bazama podataka koji upravlja relacijskim bazama podataka i koristi jezik SQL. Sustav koristi transakcije sa svojstvima atomarnosti, konzistencije, izolacije i izdržljivosti.



Slika 2.2 Prikaz PostreSQL u aplikaciji pgAdmin

# Arhitektura sustava

## Baza podataka

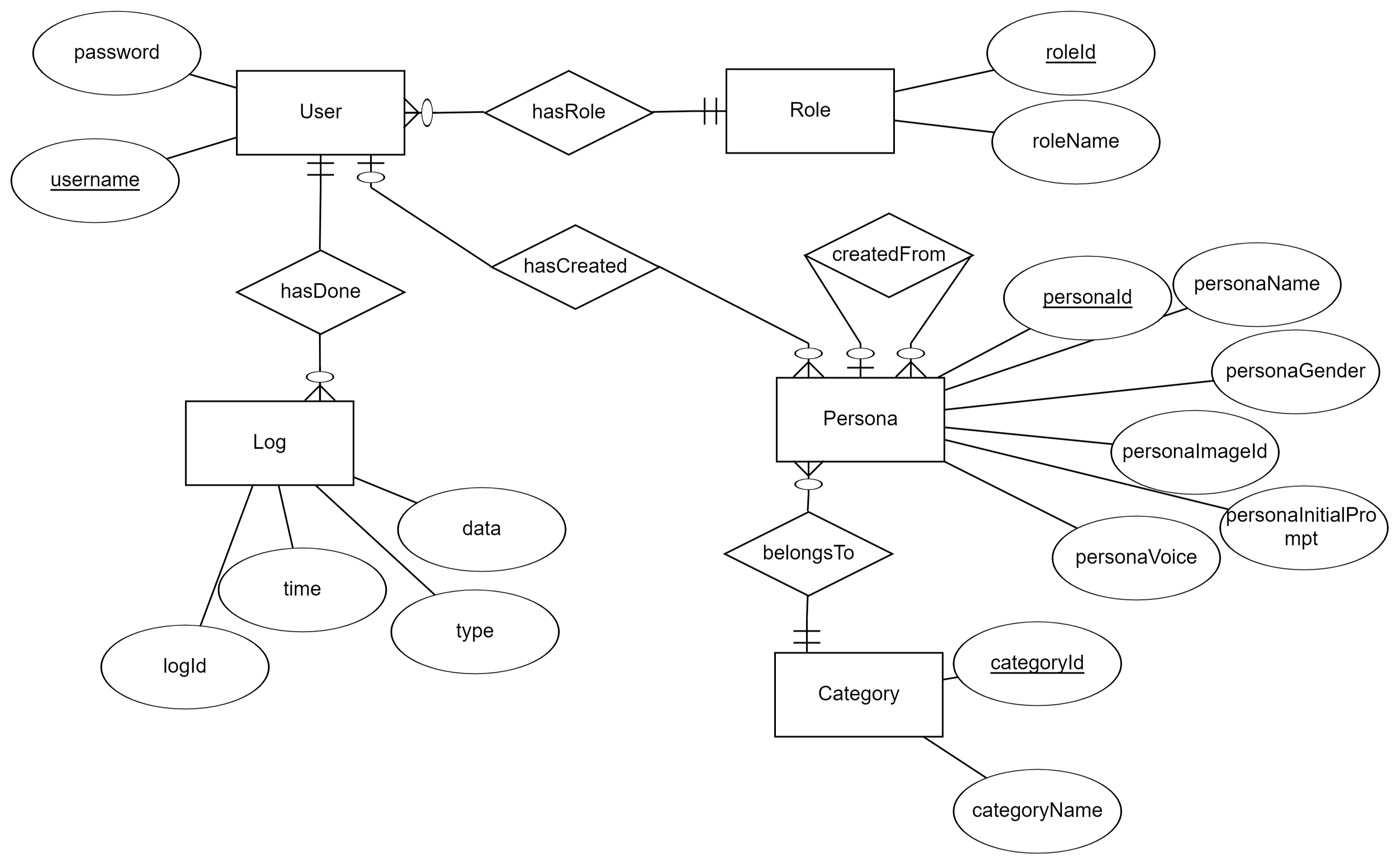
Baza podataka je za ovaj sustav relativno jednostavna. Njena temeljna svrha je čuvanje informacija i podataka o osobama (u kôdu i dijagramima označeno engl. *persona*) koje sustav treba imitirati. Korištena je relacijska baza gdje svaka relacija predstavlja neki entitet iz stvarnog života. Entiteti su međusobno povezani vezama različitih spojnosti.

### ER model baze podataka

Baza podataka sadrži sljedeće entitete:

* *User*
* *Role*
* *Log*
* *Persona*
* *Category.*

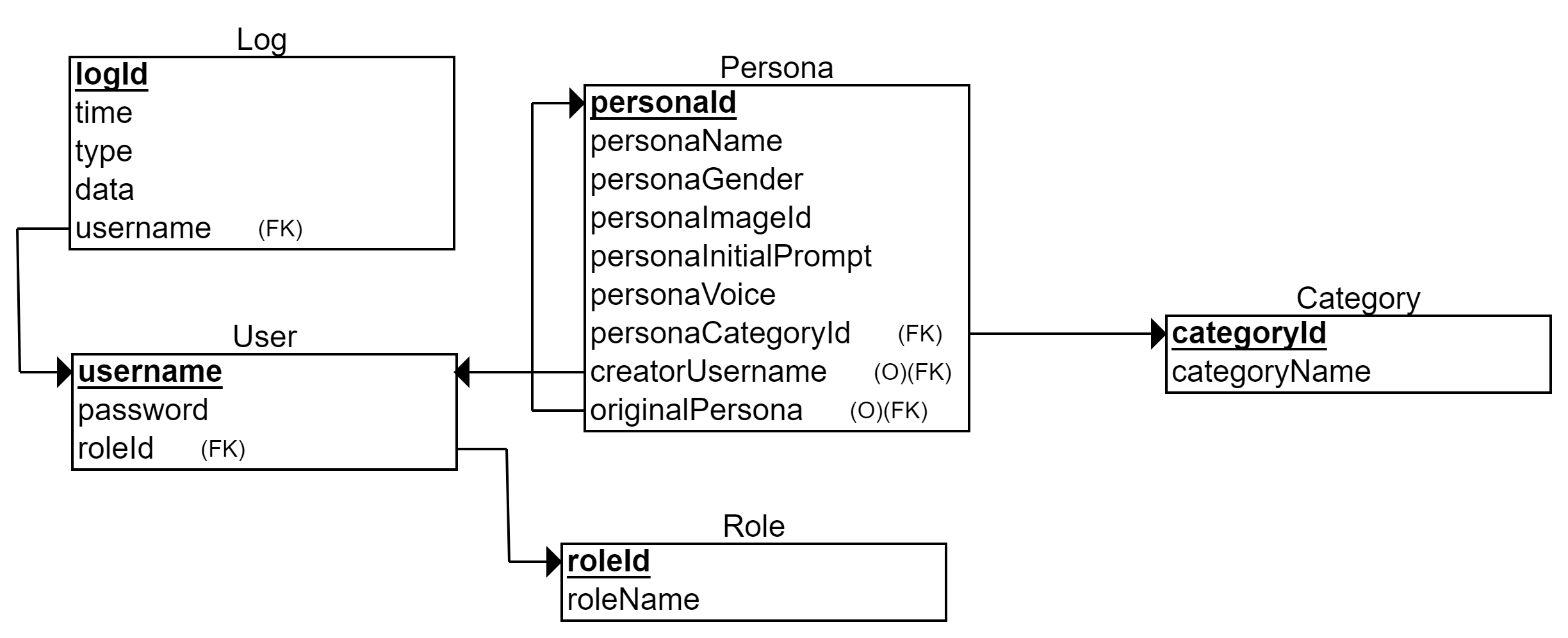
Entitet *User* predstavlja korisnika te je povezan s entitetima *Role* (koji označava jednu od uloga koje pojedini korisnik može imati), *Log* (koji modelira neku radnju koju je korisnik obavio) te *Persona* (veza se temelji na tome da korisnici mogu kreirati nove osobe te se u zapisu te osobe nalazi informacija tko ju je stvorio)*. Persona* je središnji entitet baze podataka ovog sustava i modelira pojedinu osobu koju sustav može oponašati. Taj je entitet refleksivno vezan sam za sebe s obzirom na to da jedna osoba može biti napravljena na temelju druge. *Persona* pripada pojedinoj kategoriji koja je modelirana zasebnim entitetom *Category* i stoga je povezana s tim entitetom.



Slika 3.1 ER model baze podataka

### Relacijska shema baze podataka

Relacijska se shema baze podataka temelji na ER modelu. Entiteti se pretvaraju u relacije, a veze se modeliraju dodavanjem atributa u relacije. Logika i razlozi za povezanost pojedinih relacija ostali su isti kao i za veze između entiteta u ER modelu.



Slika 3.2 Relacijska shema baze podataka

### Opis tablica baze podataka

Primarni će ključevi biti označeni svjetložutom pozadinskom bojom imena atributa, a atributi koji se referenciraju na drugi entitet, odnosno strani ključevi svijetloplavom pozadinskom bojom naziva atributa.

#### User

Entitet *User* sadrži sve potrebne informacije o korisnicima sustava. Sadrži atribute *username*, *password* i *roleId.* Ovaj je entitet u vezi *One-to-many* s entitetom *Role* preko *roleId*, u vezi *Many-to-one* s entitetom *Log* preko *username* te u vezi *Many-to-one* s entitetom *Persona* preko *creatorUsername.*

Tablica 3.1 Entitet *User*

|  |  |  |
| --- | --- | --- |
| **User** | | |
| username | VARCHAR | jedinstveno korisničko ime korisnika |
| password | VARCHAR | lozinka korisnika |
| roleId | INT | razina prava korisnika, njegova uloga (*Role.roleId*) |

#### Role

Entitet *Role* sadrži informacije o ulogama korisnika sustava. Sadrži atribute *roleId* i *roleName.* Ovaj je entitet u vezi *Many-to-one* s entitetom *User* preko *roleId*.

Tablica 3.2 Entitet *Role*

|  |  |  |
| --- | --- | --- |
| **Role** | | |
| roleId | INT | jedinstven identifikator uloge |
| roleName | VARCHAR | Ime pojedine uloge |

#### Log

Entitet *Log* sadrži informacije o aktivnostima u sustavu. Postavljanje upita osobi, prijava, odjava i slične aktivnosti bit će zapisane u ovoj tablici. Sadrži atribute *logId*, *time, type, data* i *username* *password.* Ovaj je entitet u vezi *One-to-many* s entitetom *User* preko *username.*

Tablica 3.3 Entitet *Log*

|  |  |  |
| --- | --- | --- |
| **Log** | | |
| logId | INT | jedinstven identifikator *loga* |
| time | TIMESTAMP | trenutak u kojem je *log* napravljen ili zadnje izmijenjen |
| type | VARCHAR | tip *loga* (prijava, odjava, razgovor…) |
| data | VARCHAR | podatci o aktivnosti koja se izvršila (JSON razgovora, kreiran osoba,…) |
| username | VARCHAR | Korisnik koji je izvršio radnju zapisanu u *log* (*User.username*) |

#### Persona

Entitet *Persona* sadrži karakteristike osoba koje sustav oponaša. Sve su potrebne informacije za postavljanje karakteristika osobe te njene slike i glasa u ovoj tablici. Sadrži atribute *personaId*, *personaName, personaGender, personaImageId, personaInitialPrompt, personaVoice, personaCategoryId, creatorUsername* i *originalPersona.* Ovaj je entitet u vezi *One-to-many* s entitetom *Persona* preko *creatorUsername,* u vezi *One-to-many* s entitetom *Category* preko atributa *personaCategoryId* te u *Many-to-one* vezi sa samim sobom preko atributa *originalPersona* koji referencira *personaId* neke postojeće osobe. *creatorUsername* i *originalPersona* su opcionalni atributi.

Tablica 3.4 Entitet *Persona*

|  |  |  |
| --- | --- | --- |
| **Persona** | | |
| personaId | INT | jedinstven identifikator osobe |
| personaName | VARCHAR | ime ili naziv osobe |
| personaGender | VARCHAR | spol osobe, može biti „*male*“ ili „*female*“ |
| personaImageId | INT | identifikator avatara osobe |
| personaInitialPrompt | VARCHAR | inicijalan upit kojim se dobiva željeno ponašanje modela, postavljaju se karakteristike željene osobe |
| personaVoice | VARCHAR | naziv glasa koji se koristi u interaktivnom načinu rada |
| personaCategoryId | INT | idenfitikator kategorije kojoj osoba pripada |
| creatorUsername | VARCHAR | *username* korisnika koji je kreirao osobu ili prazno polje ako je osoba dodana u bazu od strane admina |
| username | VARCHAR | Korisnik koji je izvršio radnju zapisanu u *log* (*User.username*) |

#### Category

Entitet *Category* sadrži informacije o kategorijama osoba. Sadrži atribute *categoryId* i *categoryName.* Ovaj je entitet u vezi *Many-to-one* s entitetom *Persona* preko *categoryId*.

Tablica 3.5 Entitet *Category*

|  |  |  |
| --- | --- | --- |
| **Category** | | |
| categoryId | INT | jedinstven identifikator kategorije |
| categoryName | VARCHAR | Ime pojedine kategorije |

## Dijagram komponenti

## Aplikacijska programska sučelja vanjskih usluga

Aplikacijska programska sučelja vanjskih usluga korištena su pri izradi ovog sustava sa svrhom realnih odgovora pojedinih osoba i sa svrhom omogućavanja interaktivnosti kreacijom zvučnih zapisa za pojedine poruke.

Oba u nastavku opisana sučelja koriste tzv. *API token* za autentifikaciju korisnika.

### OpenAI API

OpenAI API [8] se u kontekstu ovog sustava koristi za generiranje odgovora pojedinih osoba u pojedinim kontekstima. Ima nekoliko korisnih funkcija kao što su generiranje teksta, generiranje slika i sinteza teksta iz govora. Ovo aplikacijsko programsko sučelje sadrži nekoliko modela kojima se može pristupiti, uključujući GPT-4, GPT-3.5, DALL-E i Whisper.[1]. Konkretno, ovdje se koristi model GPT 3.5, model iza popularnog alata ChatGPT. Također, postoji i *openai* npm paket koji se u kodu koristi za olakšani pristup funkcijama sučelja kreiranjem instance API-ja OpenAIApi i postavljanjem zadane konfiguracije.

Iako ovo aplikacijsko programsko sučelje sadrži mnoge URL-ove na koje se zahtjevi mogu slati, u implementaciji sustava se zahtjevi šalju samo na <https://api.openai.com/v1/chat/completions> [8]. Slanje zahtjeva na ovaj URL omogućava generiranje odgovora na poruke priložene u zahtjevu, što znači da generira odgovor na temelju konteksta trenutnog razgovora. Bez te funkcionalnosti ne bi bilo moguće definirati karakteristike pojedinih osoba i da one ostanu pohranjene, tj. da ih model uzme u obzir. Postoji i nedostatak: mora se svaki put slati cijeli razgovor kao kontekst, ali u praksi nije predstavljalo probleme s brzinom odgovora.

const { Configuration, OpenAIApi } = require("openai");

const model = "gpt-3.5-turbo"

const {OPENAI\_API\_KEY} = require('../data/openai\_api\_key')

const configuration = new Configuration({ apiKey: OPENAI\_API\_KEY, });

const openai = new OpenAIApi(configuration);

router.post("/", async (req, res) => {

chatData = req.body

console.log(chatData)

try {

const response = await openai.createChatCompletion({

model: model,

messages: chatData,

temperature : 0.8,

presence\_penalty : 1.3,

})

if(response.status == 429){

console.log(response)

res.status(429)

res.send(response)

}

res.send(response.data.choices[0].message);

}

catch (e) {

res.status(400)

console.log(e)

res.send()

}

});

Kôd 3.1 Primjer slanja zahtjeva na OpenAI API pomoću npm paketa

## Play.ht API

Play.ht[[10]](#footnote-10) je internetska usluga koja omogućava sintezu govora iz teksta na više različitih jezika što je bilo potrebno za implementaciju i omogućavanje interaktivne komunikacije s pojedinim osobama. Ovo aplikacijsko programsko sučelje je jednostavno za korištenje: šalje se zahtjev za generiranjem audio zapisa te se onda šalje zahtjev za dohvat tog audio zapisa. Nedostatak korištenja sučelja je vrijeme koje je potrebno da se zapis kreira za dulje odgovore osoba. Iz tog se razloga ne koriste vanjske usluge za drugi smjer, sintezu teksta iz govora, već npm paket react-speech-kit[[11]](#footnote-11) koji koristi sustav za prepoznavanje govora ugrađen u pregledniku koji se koristi.

router.get("/voices", async (req, res) => {

try {

response = await voiceRequest.get("https://play.ht/api/v1/getVoices")

let returnArray = response.data.voices.filter((v) =>

v.voiceType === "Standard"

)

res.send(returnArray)

}

catch {

res.status(400)

res.send()

}

})

Kôd 3.2 Primjer dohvata mogućih glasova tipa *Standard* s Play.ht API-ja

router.post("/", async (req, res) => {

try {

const sendData = {

"voice": req.body.voice,

"content": req.body.content,

}

console.log(sendData)

let response

try {

response = await voiceRequest.post("https://play.ht/api/v1/convert", sendData)

} catch (error) {

console.error(error);

}

let audio = {

"converted" : false

}

console.log(response)

console.log(response.data)

let transcriptionId = response.data.transcriptionId

do {

response = await voiceRequest.get("https://play.ht/api/v1/articleStatus?transcriptionId=" + transcriptionId )

audio = response.data

} while (!audio.converted)

res.send(response.data.audioUrl)

}

catch (e) {

res.status(429)

res.send()

}

});

Kôd 3.3 Primjer slanja zahtjeva za kreacijom audio zapisa i preuzimanja istog

U Kôdu 3.3 generira se i šalje zahtjev za generiranjem audio zapisa za pojedini tekst te se u petlji ispituje u *do-while* petlji je li audio zapis spreman za preuzimanje te ako je, preuzima se i šalje na klijentsku stranu gdje se taj zapis i reproducira.

# Primjeri korištenja

# Izazovi tijekom izrade sustava

Tijekom izrade rada pojavilo se nekoliko izazova čije je rješavanje zahtijevalo nove ideje i pristupe implementaciji te testiranje nekoliko različitih mogućih varijanti kako bi se moglo zaključiti koja najbolje rješava taj specifičan problem. Obzirom da je dobar dio tih izazova proizlazio iz odgovora aplikacijskog programskog sučelja jezičnog modela GPT-3.5 i njegove strukture te da su izazovi bili specifični za ovakav sustav, u nastavku su tablično popisani izazovi i rješenja koja su se ispostavila kao najbolja. Izazovi se dijele na izazove oblikovanja upita (engl. *prompt engineering*) i izazove oblikovanja sustava

## Izazovi oblikovanja upita

Tablica 5.1 Izazovi oblikovanja upita

|  |  |  |
| --- | --- | --- |
| Redni broj | Izazov | Rješenje |
| 1. | Definiranje osobe prije korisnikove prve poruke | Postavljanje inicijalnog upita (engl. *prompt*) koji definira osobu kao prvu poruku u razgovoru koji se šalje na API [referenca na API link]. |
| 2. | Definiranje osobina osobe prije korisnikove prve poruke | Dodavanje rečenica u inicijalan upit koje specificiraju osobine koje želimo, npr. „*Charles was built to be respectful, polite and inclusive*.“ |
| 3. | Eliminacija odgovora gdje sam model generira pitanja za osobu koju imitira | Dodavanje rečenice „*The questions will be provided by the user in the following messages.“* u inicijalan upit kako bismo naglasili da model treba odgovoriti na korisnikova pitanja, a ne ih zadati. |
| 4. | Eliminacija odgovora gdje model spominje da nije persona već da se radi o jezičnom modelu | Dodavanje rečenice „*Do not mention you are an AI model in any circumstance!“* u inicijalan upit, no ovaj izazov nije u potpunosti riješen jer u nekim slučajevima model i dalje naglašava da se radi o jezičnom modelu. |
| 5. | Eliminacija odgovora modela gdje se generira cijeli razgovor u razgovoru dvije osobe | Postavljanje *stop* polja u zahtjevu na API [referenca na API link] na ime druge persone u razgovoru iza koje neposredno slijedi znak „:“, parsiranje odgovora temeljeno na prethodno viđenim neželjenim odgovorima i dodavanje rečenice u inicijalan upit koja specifično govori da se generiraju poruke isključivo iz perspektive jedne osobe. |
| 6. | Postavljanje nasumičnosti odgovora i smanjivanje pojavljivanja istih izraza | Podešavanje polja *temperature* i *presence\_penalty* u zahtjevu na API [referenca na API link]. |

## Izazovi oblikovanja sustava

Tablica 5.2 Izazovi oblikovanja sustava

|  |  |  |
| --- | --- | --- |
| Redni broj | Izazov | Rješenje |
| 1. | Definiranje uloga u razgovoru koji se šalje kao kontekst na API | Korisnikove poruke označavaju se ulogom *user,* a poruke modela (osobe) ulogom *assistant* [referenca na API link]. |
| 2. | Sinkronizacija govora i animacije govora | Nakon puštanja zvuka govora, kreće se redom kroz tekst i za svaki znak na koji se naiđe na avatar persone postaviti prikladnu sliku usta za taj znak. Ideja je da sustav, što je više moguće, istovremeno prolazi kroz isti znak u zvučnom zapisu i u teksutalnom zapisu. Potreban je i vremenski razmak između promjene usta kako bi se kompenziralo vrijeme izgovora znaka u zvučnom zapisu, u suprotnom se usta prebrzo pomiču na animaciji. Interval je u sustavu postavljen na 60ms. |
| 3. | Izgovor brojeva | Obzirom da se brojevi gledaju kao jedan znak, treba ih parsirati u niz znakova (npr. 7 u *seven*) kako bi animacija bila napravljena za isti broj znakova koliko ima i izgovor u zvučnom zapisu. Korišten je npm paket num2words[[12]](#footnote-12). |
| 4. | Odgoda poruka radi usporavanja toka razgovora zbog lakše čitljivosti i držanja ispod *rate limita* na API-ju [referenca na API link] | Pisanje posebne funkcije *sleep* po uzoru na jezik C korištenjem JavaScriptove asinkrone funkcije *setTimeout.* |
| 5. | Simuliranje dvije osobe pristupom na jedan API | Korištenje dvije zasebne kopije razgovora za dvije osobe, instanca razgovora za pojedinu osobu drži poruke te osobe pod ulogom *assistant,* a poruke druge osobe pod ulogom *persona* te inicijalan upit te osobe kao prvu poruku. To osigurava da kod slanja razgovora kao konteksta na API on zna koju osobu treba imitirati kod odgovora. Za prikaz samog razgovora se koristi treća struktura podataka u koju se stavljaju samo informacije koje se trebaju prikazati na ekranu. |
| 6. | Promjena teme razgovora dvije osobe | Unosom željene teme se u poruke osobe čiji je red napisati poruku nadodaje poruka koja specificira o kojoj temi se želi razgovarati te se nastavljaju izmjenjivati poruke kako je opisano u rješenju 5. |

# Zaključak

Na kraju rada piše se kratak zaključak, duljine do najviše jedne stranice.

# Literatura

Popis literature dolazi na kraju rada, iza zaključka, a prije ostalih priloga.

Na naslov Literatura primijenite stil Heading 1, a zatim ručno maknite brojčanu oznaku (to je važno kako bi i naslov „Literatura“ ušao u sadržaj na početku rada, prije uvoda).

Pri kreiranju navoda u popisu literature koristite stil literatura.

Primjeri u nastavku ilustriraju navođenje raznih izvora u popisu literature: (1) knjige, (2) članka u časopisu, (3) članka u zborniku konferencije, (4) doktorskog, magistarskog ili diplomskog rada, (5) web-stranice.

1. Tanenbaum, A. S., Wetherall, D. J. Computer Networks. 5. izdanje. London: Pearson, 2013.
2. Brady, P.T. A Statistical Analysis of On-off Patterns in 16 Conversations, Bell System Technical Journal, 47,1 (1998), str. 55-62.
3. Brady, N. A Statistical Analysis of Use Case. Proceedings of the 7th International Conference on Telecommunications ConTEL, Zagreb, (2003), str. 45-52.
4. Ivić, M. Analiza ponašanja korisnika u digitalnim igrama namijenjenima učenju. Diplomski rad. Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 2016.
5. Epstein M., The best VR headset in 2019, PC Gamer, (2019, listopad). Poveznica: <https://www.pcgamer.com/best-vr-headset/>; pristupljeno 4. listopada 2019.

Uz svaki preuzeti sadržaj u svom radu – bilo da je riječ o tekstu (izravno citiranome ili „prepričanome“), slici ili grafičkom prikazu – treba navesti oznaku izvora (članak, knjiga, web-stranica ...) u popisu literature te se na nju „pozvati“, na primjer:

Međusobno povezivanje mreža zasniva se na primjeni komunikacijskih protokola (Tanenbaum i Wetheral, 2014).

Podaci o karakteristikama uređaja za virtualnu stvarnost preuzeti su s portala PC Gamer [5].

Početna verzija programa preuzeta je iz diplomskog rada [5].

U danim primjerima mogli ste uočiti dva načina referenciranja:

* (Tanenbaum i Wetheral, 2014),
* [1].

Kad izaberete jedan od njih svakako ga se držite konzistentno u cijelome radu.

DODATI SVE POD INSERT CITATIONS SOURCES

[1] <https://platform.openai.com/docs/models/overview>

[2] <https://platform.openai.com/docs/api-reference/introduction>

[3] <https://www.fer.unizg.hr/_download/repository/UI-1-Uvod.pdf>

[4] <https://www.ibm.com/topics/chatbots>

[5] <https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP>

[6] <https://www.fer.unizg.hr/_download/repository/08-JavaScript%203od3.pdf> pristupano 17.5.

[7] <https://www.webdevolution.com/blog/Javascript-Event-Loop-Explained>

[8] <https://platform.openai.com/docs/api-reference> pristupano 24.5.2023.

[9] <https://docs.play.ht/reference/api-getting-started> 24.5.

# Sažetak

Naslov, sažetak, ključne riječi (na hrvatskom jeziku)

Sažetak opisuje sadržaj rada, prepričan u stotinjak riječi.

# Summary

Title, summary, keywords (na engleskom jeziku)

1. <https://developer.mozilla.org/en-US/docs/Web/JavaScript> [↑](#footnote-ref-1)
2. <https://react.dev/> [↑](#footnote-ref-2)
3. <https://nodejs.org/en> [↑](#footnote-ref-3)
4. <https://www.npmjs.com/> [↑](#footnote-ref-4)
5. <https://expressjs.com/> [↑](#footnote-ref-5)
6. <https://www.npmjs.com/package/express-validator> [↑](#footnote-ref-6)
7. <https://getbootstrap.com/> [↑](#footnote-ref-7)
8. <https://axios-http.com/docs/intro> [↑](#footnote-ref-8)
9. <https://www.postgresql.org/> [↑](#footnote-ref-9)
10. <https://play.ht/> [↑](#footnote-ref-10)
11. <https://www.npmjs.com/package/react-speech-kit> [↑](#footnote-ref-11)
12. <https://www.npmjs.com/package/num2words> [↑](#footnote-ref-12)