Univerzitet u Zenici

Politehnički fakultet

Softversko inženjerstvo

**Sistemska dokumentacija za**

**klijent - server aplikaciju koja pretvara govor u text (Speech to Text)**

Multimedijalni sistemi i aplikacije

Studenti: Profesor:

Ajna Fetić (II-64), Dženan Bejdić (II-68) Doc. Dr. Nermin Goran,

**Sadržaj**

[1. DOKUMENTACIJA ZAHTJEVA 3](#_Toc124545815)

[1.1. ZAHTJEVI NA STRANI KLIJENTA 4](#_Toc124545816)

[1.2. ZAHTJEVI NA STRANI SERVERA 4](#_Toc124545817)

[1.3. ZAHTJEVI ZA METRIKU PROJEKTA 5](#_Toc124545818)

[2. ARHITEKTURA I DIZAJN APLIKACIJE 6](#_Toc124545819)

[2.1. ARHITEKTURA APLIKACIJE 6](#_Toc124545820)

[2.1.1 HTTP PROTOKOL 6](#_Toc124545821)

[2.1.1.1 OSNOVNI ELEMENTI HTTP PROTOKOLA 7](#_Toc124545822)

[2.1.2 TCP PROTOKOL 8](#_Toc124545823)

[2.1.3 UDP PROTOKOL 9](#_Toc124545824)

[2.1.4 PYTHON 10](#_Toc124545825)

[2.1.5 FLASK 10](#_Toc124545826)

[2.1.6 REACT 10](#_Toc124545827)

[2.1.7 GOOGLE SPEECH TO TEXT 11](#_Toc124545828)

[2.1.8 PROCES PRETVARANJA GOVORA U TEKST 13](#_Toc124545829)

[2.1.9 GOOGLE TEXT TO SPEECH 14](#_Toc124545830)

[2.1.10 PROCES PRETVARANJA TEKSTA U GOVOR (SINTEZA) 15](#_Toc124545831)

[2.2 DIZAJN APLIKACIJE 16](#_Toc124545832)

[2.2.1 USE CASE DIJAGRAM 16](#_Toc124545833)

[2.2.2 DEPLOYMENT DIJAGRAM 17](#_Toc124545834)

[3. APLIKACIJA 18](#_Toc124545835)

[3.1 IZVORNI KOD 18](#_Toc124545836)

[3.1.1 BACKEND 18](#_Toc124545837)

[3.1.2 FRONTEND 20](#_Toc124545838)

[3.1.3 KORISNIČKI INTERFEJS 25](#_Toc124545839)

[4. WIRESHARK ANALIZA 27](#_Toc124545840)

[5. POKRETANJE APLIKACIJE 30](#_Toc124545841)

# DOKUMENTACIJA ZAHTJEVA

Ideja projekta Speech-to-Text app predstavlja softvrerski produkt čija je osnovna funkcionalnost chat aplikacija sa opcijom pretvaranja govora u tekst i obrnuto. Kreirana je web aplikacija u kojoj je omogućena chat funkcionalnost sa mogućnošću unosa teksta poruke putem mikrofona, te naknadno preslušavanje poruke. Frontend dio aplikacije je kreiran u JavaScript frameworku-u React, dok je backend dio aplikacije kreiran u Python framework-u Flask-u. Komunikacija između korisnika bila bi omogućena preko web socketa, te TCP i UDP protokola. U nastavku su navedene karakteristike projekta.

**Naziv proizvoda**: Speech-To-Text App

**Vizija** : Klijent – server aplikacija koja pretvara govor u tekst i obrnuto

**Opis**: Aplikacija radi pomoću HTTP, TCP i UDP protokola, te je u svakom trenutku moguće poslati tekstualnu poruku ostalim prisutnim korisnicima u chat sobi, moguće je putem mikrofona izgovorit poruku koja će na server strani biti pretvorena u tekst, te će isti biti unesen u input poruke. Također je moguće preslušati bilo koju od ranije poslanih poruka, svoju ili od drugih korisnika, klikom na dugme play pored poruke, koje okida zahtjev za pretvaranje govora u tekst te preslušavanje dobijenog audio-zapisa.

**Slučajevi upotrebe**: Priključivanje na server, ulazak u chat sobu, slanje tekstualne poruke, unos poruke putem mikrofona, pretvaranje poruke u tekst, preslušavanje poruke, pauziranje audio zapisa, te izlazak iz chat sobe.

**Pretpostavke:** Krajnji korisnik ima PC računar na kojem je instaliran jedan od popularnih operativnih sistema Windows, Linux ili MacOS, sa pristupom Internetu. Klijent pristupa aplikaciji putem linka, ulazi u chat sobu te nastavlja sa korištenjem aplikacije.

**Arhitektura i komponente :** Arhitektura aplikacije bazirana je na Python i JavaScript programskim jezcima, te njhovim frameworcima Flask i React respektivno. Korištena su dva mrežna protokola, TCP i UDP.

# ZAHTJEVI NA STRANI KLIJENTA

Klijent-server aplikacija Speech-to-text predstavlja web aplikaciju sa osnovnom funkcionalnošću chata odnosno razmjenjivanja poruka. Klijent mora moći u bilo kom trenutku biti u mogućnosti da se priključi na server, pristupi chat sobi, šalje ili prima tekstualne poruke. Također mora biti u mogućnosti da u bilo kojem trenutuku unese poruku putem mikrofona ili da glasovno presluša već postojeće poslane poruke.

Na strani klijenta potrebno je:

* Kreirati dugme za pristup chat sobi
* Kreirati korisnički interfejs za chat sobu
* Kreirat input za unos poruke
* Kreirati dugme za unos govora (ikona mikrofona)
* Kreirati dugme za preslušavanje poruke
* Kreirati funkcionalnosti koje se pokreću klikovima na ovu dugmad
* Omogućiti pristup socketima
* Povezati klijent dio sa serverom, omogućiti socket komuikaciju

# ZAHTJEVI NA STRANI SERVERA

Na serverskoj strani ove aplikacije potrebno je omogućiti socket konekciju sa klijentom, te

omogućiti slanje paketa podataka u obliku toka (stream-a) kako bi klijent u bilo kom trenutku mogao da pošalje poruku, unese poruku putem mikrofona te presluša neku od postojećih poruka. Kako bi se ovo omogućilo potrebno je:

* Implementirati funckije za konekciju sa klijentom
* Napraviti endpointe koji će omogućiti pretvaranje teksta u govor i obrnuto
* Kreirati sockete
* omogućiti paketiziranje podataka koji će se slati u header-u i playload-u odgovora klijentu
* kreirati funkcije koje će klijentu vraćati odgovarajući response.

# ZAHTJEVI ZA METRIKU PROJEKTA

Prilikom pokretanja projekta potrebno je pratiti i analizirati poslane pakete podataka. Korišten je alat Wireshark koji je potrebno pokrenuti i pratiti slanje paketa između klijenta i servera.

Posebno je potrebno obratiti pažnju na:

* Korišteni port (TCP ili UDP)
* Veličinu paketa
* Redslijed isporuke
* Redoslijed slanja paketa te njihov protokol

# ARHITEKTURA I DIZAJN APLIKACIJE

Kao što je ranije spomenuto, backend dio aplikacije je odrađen u Pythonovom framewroku-u Flask-u, dok je frontend dio aplikacije kreiran u JavaScript framework-u React-u. Komunikacija između backenda i frontenda, odnosno klijenta i servera omogućena je putem HTTP, TCP i UDP protokola. U sljedećim poglavljima će ove komponente aplikacije biti detaljnije opisane.

# ARHITEKTURA APLIKACIJE

# HTTP PROTOKOL

Protokol za prijenos hiperteksta (HTTP) je temelj World Wide Weba i koristi se za učitavanje web stranica pomoću hipertekstualnih veza. HTTP je protokol sloja aplikacije dizajniran za prijenos informacija između umreženih uređaja i radi na vrhu drugih slojeva steka mrežnih protokola. Tipičan tok preko HTTP-a uključuje klijentsku mašinu koja šalje zahtjev serveru, koji zatim šalje poruku odgovora.

HTTP funkcionira kao protokol zahtjev-odgovor u modelu klijent-server. Web pretraživač, na primjer, može biti klijent, dok proces, nazvan web server, koji radi na računaru na kojem se nalazi jedna ili više web lokacija, može biti server. Klijent šalje serveru poruku HTTP zahtjeva. Server, koji obezbjeđuje resurse kao što su HTML datoteke i drugi sadržaj ili obavlja druge funkcije u ime klijenta, vraća poruku odgovora klijentu. Odgovor sadrži informacije o statusu dovršenosti zahtjeva, a također može sadržavati traženi sadržaj u svom tijelu poruke.

Web pretraživač je primjer korisničkog agenta (UA). Druge vrste korisničkih agenata uključuju softver za indeksiranje koji koriste pružaoci usluga pretraživanja (web indekseri), glasovni pretraživači, mobilne aplikacije i drugi softver koji pristupa, konzumira ili prikazuje web sadržaj.

HTTP je dizajniran da omogući posrednim mrežnim elementima da poboljšaju ili omoguće komunikaciju između klijenata i servera. Web lokacije s velikim prometom često imaju koristi od web keš servera koji isporučuju sadržaj u ime uzvodnih servera kako bi poboljšali vrijeme odgovora. Web pretraživači keširaju prethodno pristupane web resurse i ponovo ih koriste, kad god je to moguće, kako bi smanjili mrežni promet. HTTP proxy serveri na granicama privatne mreže mogu olakšati komunikaciju za klijente bez globalno rutabilne adrese, prenošenjem poruka s vanjskim serverima.

Diagram

Description automatically generated

Slika 1 Način funkcionisanja HTTP protokola

# OSNOVNI ELEMENTI HTTP PROTOKOLA

Neki od osnovnih elemenata vezanih za http protocol su:

**HTTP request (zahtjev) :**

HTTP zahtjev je način na koji internet komunikacijske platforme kao što su web pretraživači traže informacije koje su im potrebne za učitavanje web stranice i web apliakcija.

Svaki HTTP zahtjev napravljen preko Interneta nosi sa sobom niz kodiranih podataka koji nose različite vrste informacija. Tipičan HTTP zahtjev sadrži:

* http verziju
* URL
* HTTP metodu
* Zaglavlje HTTP zahtjeva
* Opcionalno tijelo zahtjeva

**HTTP metoda**

HTTP metoda , ukazuje na akciju koju HTTP zahtjev očekuje od upitanog servera. Na primjer, dvije najčešće HTTP metode su 'GET' i 'POST'; 'GET' zahtjev zauzvrat očekuje informacije (obično u obliku web stranice), dok zahtjev 'POST' obično ukazuje da klijent šalje informacije web serveru (kao što su informacije obrasca, npr. poslano korisničko ime i lozinka ).

**Zaglavlje HTTP zahtjeva**

HTTP zaglavlja sadrže tekstualne informacije pohranjene u parovima ključ/vrijednost i uključena su u svaki HTTP zahtjev. Ova zaglavlja komuniciraju osnovne informacije, kao što je pretraživač koji klijent koristi i koji se podaci traže.

**Tijelo HTTP zahtjeva**

Tijelo zahtjeva je dio koji sadrži 'tijelo' informacija koje zahtjev prenosi. Tijelo HTTP zahtjeva sadrži sve informacije koje se šalju na web server, kao što su korisničko ime i lozinka, ili bilo koji drugi podatak unet u obrazac.

**HTTP response (odgovor)**

HTTP odgovor je ono što veb klijenti (često pretraživači) dobijaju od internet servera kao odgovor na HTTP zahtjev. Ovi odgovori saopštavaju vrijedne informacije na osnovu onoga što je traženo u HTTP zahtjevu. HTTP odgovor se uglavnom sastoji od:

* HTPP status kod
* Zaglajvlje HTTP odgovora
* Opcionalno HTTP tijelo.

# TCP PROTOKOL

Osim HTTP protokola, klijent server komunikacija se u ovom projktu također ostvaruje putem TCP i UDP protokola.

TCP (Transmission Control Protocol) je jedan od glavnih internetskih protokola. Nalazi se između slojeva aplikacije i mreže koji se koriste za pružanje pouzdanih usluga isporuke. To je protokol za komunikaciju orijentiran na vezu koji pomaže u razmjeni poruka između različitih uređaja preko mreže. Internet protokol (IP), koji uspostavlja tehniku za slanje paketa podataka između računara, radi sa TCP.

Da bi se osiguralo da svaka poruka stigne na svoju ciljnu lokaciju netaknuta, TCP/IP model razlaže podatke u male snopove i nakon toga ponovo sastavlja pakete u originalnu poruku na suprotnom kraju. Slanje informacija u malim snopovima informacija čini jednostavnijim održavanje efikasnosti za razliku od slanja svega odjednom.

Nakon što se određena poruka razbije u pakete, ovi paketi mogu putovati duž više ruta ako je jedna ruta zaglavljena, ali odredište ostaje isto.

Diagram

Description automatically generated

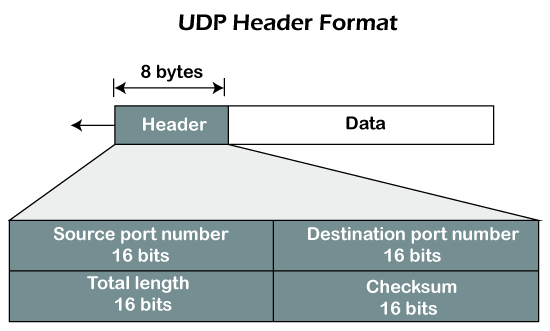
Slika 2 Protok podataka u TCP protokolu

# UDP PROTOKOL

UDP je protokol transportnog sloja. UDP je dio paketa Internet protokola, koji se naziva UDP/IP paket. Za razliku od TCP-a, to je nepouzdan protokol bez povezivanja. Dakle, nema potrebe za uspostavljanjem veze prije prijenosa podataka. UDP pomaže da se uspostave veze sa malim kašnjenjem i tolerisanjem gubitaka preko mreže. UDP omogućava procesu da obradi komunikaciju.

Iako je protokol kontrole prijenosa (TCP) dominantan protokol transportnog sloja koji se koristi sa većinom internetskih usluga; pruža osiguranu isporuku, pouzdanost i još mnogo toga, ali sve ove usluge nas koštaju dodatnih troškova i kašnjenja. Ovdje se pojavljuje UDP. Za usluge u stvarnom vremenu kao što su kompjuterske igre, glasovna ili video komunikacija, konferencije uživo; potreban nam je UDP. Pošto su potrebne visoke performanse, UDP dozvoljava ispuštanje paketa umjesto obrade odloženih paketa. U UDP-u nema provjere grešaka, tako da štedi i propusni opseg.

Protokol korisničkih datagrama (UDP) je efikasniji u pogledu kašnjenja i propusnog opsega.



Slika 3 Zagljavlje UDP protokola

# PYTHON

Python je interpretirani, interaktivni, objektno orijentisani programski jezik visokog nivoa, koji poseduje dinamičku sematiku. Odlikuje ga visokokvalitetna struktura podataka, koja ga, u kombinaciji sa dinamičkim pisanjem i vezivanjem, čini veoma atraktivnim za brz razvojr azličitih aplikacija.

Njegova fleksibilnost dozvoljava postizanje mnogih rezultata, bilo da su oni mali ili veliki. Python se tako može koristiti za pisanje jednostavnih programa, ali poseduje i snagu potrebnu za kreiranje složenih operacija koje koriste globalne, multinacionalne kompanije. Također,

Python je pogodan za upotrebu „scripting” i „glue” programskih jezika, koji se koriste zavezivanje postojećih komponenti u cjelinu.

# FLASK

Za izdradu backend dijela aplikaicje korišten je python te njegov framework Flask. U backend dijelu aplikacije kreiran je API koji omogućava socket konekciju priliskom ulaska u chat te je na backend dijelu omogućen pristup Google text to speehc-u koji pretvara tekst u govor.

Flask je mikro web framework napisan u Pythonu. Klasifikovan je kao mikrookvir jer ne zahteva posebne alate ili biblioteke. Nema sloj apstrakcije baze podataka, provjeru valjanosti obrasca ili bilo koje druge komponente gdje već postojeće biblioteke treće strane pružaju zajedničke funkcije. Međutim, Flask podržava ekstenzije koje mogu dodati funkcije aplikacije kao da su implementirane u samom Flasku. Postoje proširenja za objektno-relacijske mapere, provjeru valjanosti obrazaca, rukovanje upload-om, različite tehnologije otvorene autentifikacije i nekoliko uobičajenih alata vezanih za framework.

# REACT

Za izradu frontend dijela aplikacije korišten je JavaScript i React framework. Na frontend dijelu aplikacije izrađen je User Interface chat aplikacije, omogućeno je korisničko povezivanje n sockete te je implementiran Google Speech Recognition za pretvaranje govora u tekst.

React (također poznat kao React.js ili ReactJS) je JavaScript biblioteka otvorenog koda za izgradnju korisničkih interfejsa zasnovanih na komponentama korisničkog interfejsa. Održavaju ga Meta (bivši Facebook) i zajednica pojedinačnih programera i kompanija. React se može koristiti kao osnova u razvoju jednostraničkih, mobilnih ili serverskih aplikacija sa okvirima kao što je Next.js. Međutim, React se bavi samo upravljanjem stanjem i prikazivanjem tog stanja u DOM, tako da kreiranje React aplikacija obično zahtijeva korištenje dodatnih biblioteka za rutiranje, kao i određene funkcionalnosti na strani klijenta.

# GOOGLE SPEECH TO TEXT

U ovom projektu, osnovna funkcionalnost jeste pretvaranje govora u tekst i obrnuto. Za pretvaranje govora u text korištena je Google Cloud Speech to Text tehnologija. Speech to text omogućuje integraciju Googleovih tehnologija za prepoznavanje govora u razvojne aplikacije. Nakon što se serveru pošalje audio, Google-ov API vraća odgovor u obliku teksta. Pretvaranje govora u tekst ima tri glavne metode za izvođenje prepoznavanja govora. Oni su navedeni u nastavku:

* **Sinhrono prepoznavanje** (REST i gRPC) šalje audiopodatke API-ju za pretvaranje govora u tekst, izvodi prepoznavanje tih podataka i vraća rezultate nakon što se sav audiozapis obradi. Zahtjevi za sinhrono prepoznavanje ograničeni su na audio podatke u trajanju od 1 minute ili manje.
* **Asinhrono prepoznavanje** (REST i gRPC) šalje audio podatke API-ju za pretvaranje govora u tekst i pokreće dugotrajnu operaciju. Pomoću ove operacije mogu se povremeno tražiti rezultate prepoznavanja. Asinhrono prepoznavanje se koristi za audiozapise u trajanju do 480 minuta.
* **Streaming prepoznavanje** (samo gRPC) koji se koristi u ovom projektu obavlja prepoznavanje audio podataka koji se nalaze unutar gRPC dvosmjernog toka. Zahtjevi za strujanje dizajnirani su za potrebe prepoznavanja u realnom vremenu, kao što je snimanje zvuka uživo s mikrofona. Streaming prepoznavanje daje privremene rezultate dok se zvuk snima, dopuštajući da se rezultat pojavi, na primjer, dok korisnik još govori.

Zahtjevi sadrže konfiguracijske parametre kao i audio podatke. U nastavku će detaljnije biti opisano Streaming Recognition prepoznavanje koje se koristi u projektu te će biti predstavlena arhitektura ovog prepoznavanja.

Streaming Speech-to-Text API poziv za prepoznavanje dizajniran je za snimanje i prepoznavanje zvuka u stvarnom vremenu unutar dvosmjernog toka. Aplikacija može slati audio u toku zahtjeva i primati privremene i konačne rezultate prepoznavanja u toku odgovora u stvarnom vremenu. Privremeni rezultati predstavljaju trenutni rezultat prepoznavanja za dio audiozapisa, dok konačni rezultat prepoznavanja predstavlja posljednju, najbolju pretpostavku za taj dio audiozapisa.

Za razliku od sinhronih i asinhronih poziva, u kojima se šalje konfiguraciju i zvuk unutar jednog zahtjeva, pozivanje API-ja za strujanje govora zahtijeva slanje više zahtjeva. Prvi StreamingRecognizeRequest mora sadržavati konfiguraciju tipa StreamingRecognitionConfig bez ikakvog popratnog zvuka. Naknadni StreamingRecognizeRequests poslani putem istog toka tada će se sastojati od uzastopnih okvira neobrađenih audio bajtova.

Jedan StreamingRecognitionConfig se sastoji od sljedećih polja:

* **Config** – (obavezno polje) – sadrži informacije vezane za audio.
* **Single\_utterance** – (neobavezno polje, defaultna vrijednost je false) – označava da li poslani zahtjev treba automatski da završi nakon što govor više nije detektiran. Ako je postavljeno, Govor u tekst će detektirati pauze, tišinu ili zvuk koji nije govor kako bi odredio kada prekinuti prepoznavanje. Ako nije postavljeno, stream će nastaviti slušati i obrađivati ​​zvuk sve dok se stream izravno ne zatvori ili dok se ne prekorači ograničenje dužine streama. Postavljanje single\_utterance na true korisno je za obradu glasovnih naredbi.
* interim\_results - (neobavezno, zadana vrijednost je false) označava da bi ovaj zahtjev za stream trebao vratiti privremene rezultate koji se mogu pročistiti kasnije (nakon obrade više zvuka). Privremeni rezultati bit će zabilježeni unutar odgovora postavljanjem is\_final na false.

Rezultati prepoznavanja govora vraćaju se unutar niza odgovora tipa StreamingRecognitionResponse. Takav odgovor sastoji se od sljedećih polja:

* **speakEventType** sadrži događaje tipa SpeechEventType. Vrijednost ovih događaja pokazat će kada se utvrdi da je pojedini iskaz dovršen. Govorni događaji služe kao oznake unutar odgovora toka.
* **Results** - sadrži popis rezultata, koji mogu biti privremeni ili konačni rezultati, tipa StreamingRecognitionResult. Popis rezultata sadrži sljedeća potpolja:
  + alternative - sadrži popis alternativnih transkripcija.
  + isFinal označava jesu li rezultati dobiveni unutar ovog unosa popisa privremeni ili konačni. Google može vratiti više isFinal=true rezultata kroz jedan tok, ali isFinal=true rezultat je zajamčen samo nakon zatvaranja pisanja (napola zatvaranja).
  + Stability- označava volatilnost do sada dobivenih rezultata, pri čemu 0,0 označava potpunu nestabilnost, dok 1,0 označava potpunu stabilnost. Imajte na umu da za razliku od pouzdanosti, koja procjenjuje je li transkripcija točna, stabilnost procjenjuje može li se dati djelomični rezultat promijeniti. Ako je isFinal postavljen na true, stabilnost neće biti postavljena.

Na sljedećoj slici prikazan je jedan od mnogih slučajeva upotreba google speech-to-text API-ja:

Graphical user interface, application

Description automatically generated

Slika 4 Use Case Google STT

# PROCES PRETVARANJA GOVORA U TEKST

Na sljedećoj slici je prikazan dijagram samog procesa pretvaranja govora u tekst:

Diagram

Description automatically generated

Slika 5 Pretvaranje govora u tekst

Sam process funkcioniše na principu umjetne inteligencije te se u širem pogledu sastoji od dvije faze: training i testing. Da bi se kreirao model koji pretvara tekst u govor potrebno je prikupiti uzorke audio snimaka te uraditi ekstrakciju mfcc obilježja. U obradi zvuka, kepstrum mel-frekvencije (MFCC) predstavlja prikaz kratkoročnog spektra snage zvuka, zasnovan na linearnoj kosinusnoj transformaciji logaritamskog spektra snage na nelinearnoj mel skali frekvencije. Nakon ekstrakcije ovih koeficijenata potrebno je odraditi trening modela. Nakon treniranja modela dolazi prikupljanja audio zapisa u realnom vremenu, audio se dijeli na segmente, računa se energija svakog od tih segmenata, ta se energija naknado analizira, da bi model mogao da prepozna kada nastupi tišina kako bi se segment razdvojio na odvojene riječi. Nakon razdvajanja riječi dolazi do ekstrakcije obilježja segmenata, te kalsifikacija tog audio uzorka na osnovu ranije procesiranih uzoroka. Nakon ovog procesiranja se dobija tekst u realnom vremenu. U ovom slučaju, kod Streaming prepoznavanja, cijeli ovaj process se odvija uporedo dok korisnik unosi govor kroz mikrofon na već istreniranom modelu, te kao odgovor putem HTTP, TCP i UDP protokolima korisnik dobija ispisan tekst na ekranu.

# GOOGLE TEXT TO SPEECH

Druga osnovna funkcionalnost ove aplikaije, pored pretvaranja govora u tekst, spomenutog u ranijim poglavljima, jeste obrnuti proces, pretvaranje teksta u govor. Za pretvaranje teksta u govor korištena je još jedna Google-ova tehnologija - Google Cloud Text to Speech (gtts) tehnologija

Text-to-Speech pretvara tekst ili unos jezika za označavanje sinteze govora (SSML) u audio podatke prirodnog ljudskog govora. Text-to-Speech omogućava programerima da kreiraju sintetički ljudski govor prirodnog zvučanja kao zvuk koji se može reproducirati. Datoteke audio podataka koje kreirate pomoću pretvaranja teksta u govor mogu se koristiti za pokretanje aplikacija ili povećanje medija kao što su video ili audio.

Proces prevođenja unosa teksta u audio podatke naziva se sinteza, a izlaz sinteze se naziva sintetički govor. Text-to-Speech uzima dva tipa unosa: sirovi tekst ili SSML formatirani. Da bi se kreirala nova audio datoteka, poziva se krajnju tačku sinteze API-ja.

Proces sinteze govora generiše neobrađene audio podatke kao base64 kodiran niz. Potrebno je dekodirati base64 kodiran niz u audio datoteku prije nego što ga aplikacija može reproducirati. Većina platformi i operativnih sistema ima alate za dekodiranje base64 teksta u medijske datoteke koje se mogu reproducirati

Gtts stvara sirove audio podatke prirodnog, ljudskog govora. To jest, stvara zvuk koji zvuči kao da osoba govori. Kada se šalje zahtjev za sintezu u Text-to-Speech, potrebno je navesti glas koji 'govori' riječi.

Pretvaranje teksta u govor ima širok izbor prilagođenih glasova dostupnih za korištenje. Glasovi se razlikuju po jeziku, rodu i akcentu (za neke jezike). Zajedno sa drugim, tradicionalnim sintetičkim glasovima, Text-to-Speech takođe pruža glasove generisane WaveNet-om. Korisnici smatraju da su glasovi generisani Wavenetom topliji i sličniji ljudima od drugih sintetičkih glasova.

Ključna razlika u odnosu na WaveNet glas je WaveNet model koji se koristi za generiranje glasa. WaveNet modeli su obučeni koristeći neobrađene audio uzorke stvarnih ljudi koji govore. Kao rezultat toga, ovi modeli generiraju sintetički govor s naglaskom nalik ljudskim i infleksijom na slogovima, fonemima i riječima.

Na sljedećoj slici prikazan je jedan od mnogih slučajeva upotreba google text-to-speech API-ja:

Graphical user interface

Description automatically generated

Slika 6 Use case google TTS

# PROCES PRETVARANJA TEKSTA U GOVOR (SINTEZA)

Na sljedećoj slici je prikazan dijagram samog procesa pretvaranja teksta u govor:

Graphical user interface, text, application, chat or text message

Description automatically generated

Slika 7 Pretvaranje teksta u govor

Postoje tri faze u procesu sinteze govora; tekst u riječi, riječi u foneme i foneme u zvuk.

Početna faza sinteze govora, općenito se naziva preprocesiranje. ili normalizacija, radi se o smanjenju dvosmislenosti: radi se o sužavanju brojnih različitih načina na koje bi osoba mogla pročitati dio teksta na onaj koji je najprikladniji. preprocesiranju radi se o prolasku kroz tekst, a zatim o čišćenju kako bi model umjetne inteligencije napravio manje grešaka kada izgovara riječi. Elemente poput brojeva, datuma, vremena, skraćenica, akronima i posebnih znakova potrebno je pretvoriti u riječi. Na primjer; broj 1953 može se odnositi na nekoliko stavki, godinu ili; svaki od ovih pročitan će zvučati malo drugačije. Dok ljudi mogu shvatiti izgovor na osnovu načina na koji je tekst napisan, kompjuteri uglavnom nemaju tu sposobnost da to urade.

To je razlog zašto se koriste tehnike statističke vjerovatnoće ili neuronske mreže kako bi došli do najvjerovatnijeg izgovora.

Druga faza sinteze govora jeste pretvaranje riječi u glasove odnosne foneme:

Sintetizator govora mora generirati glasove govora koji čine riječi unesenog teksta. Svakom kompjuteru je potrebna ogromna abecedna lista riječi i detalji o tome kako izgovoriti svaku riječ. Za svaku riječ bi im bila potrebna lista fonema koji čine njen zvuk.

Treća posljednja faza predstavlja pretvaranje fonema u zvuk, što se radi pomoću unaprijed snimljenog ljudskog govora, uz pomoć treniranog modela.

U ovoj aplikaciji, nakon klika na play dugme koje se nalazi pored svake korisničke poruke, šalje se asinhroni zahtjev sa tekstom poruke u svom tijelu te kao odgovor se dobija audio zapis sintetitzranog govora.

# DIZAJN APLIKACIJE

Klijent-server aplikacija za pretvaranje govora u tekst ili teksta u govor, u prvi plan postavlja krajnjeg korisnika te su funkcionalnosti na strani klijenta od velikog značaja. Klijent pristupa aplikaciji, ulazi u chat sobu te je u mogućnosti da čita poruke drugih korisnika, šalje poruke drugim prijavljenim korisnicima, unese poruku putem mikrofona te glasovno presluša ranije polsane poruke, svoje ili drugih korisnika. Klijent u svakom trenutku može izaći iz aplikacije pri čemu se zaustavlja slanje HTTP. TCP i UDP paketa, te se gasi njegov socket.

# USE CASE DIJAGRAM

Ranije navedene funkcionalnosti su u nastavku vizuelno predstavljene. Dijagram slučaja upotrebe (use case diagram) je grafički prikaz mogućih interakcija korisnika sa sistemom. U nastavku je prikazan Use Case dijagram klijenta ove aplikacije:

Diagram

Description automatically generated

Slika 8 Use Case dijagram aplikacije

# DEPLOYMENT DIJAGRAM

Za bolje razumijevanje arhitekture samog sistema kreiran je deployment dijagram. Dijagram implementacije je tip UML dijagrama koji prikazuje arhitekturu izvršavanja sistema, uključujući čvorove kao što su hardverska ili softverska izvršna okruženja, i međusfotver koji ih povezuje. U nastavku je prikazan dijagram implementacije (Deployment dijagram) ove aplikacije:

Graphical user interface, diagram

Description automatically generated

Slika 9 Deployment dijagram aplikacije

Deployment dijagram se sastoji od tri osnovne kompoennte: korisnikov uređaj (PC ili Mobile), na kojem se pokreće frontend aplikacije izrađen u React.js, sever koji komunicra sa klijentom putem http protokola na kojem je pokrenut backend aplikacije izrađen u Flask-u, te treća komponenta koja predstavlja Google Cloud servise sa izdvojena dva API-ja : Text to Speech API te Speech to text API. Backend komunicira sa Google Cloud servisima također putem http protokola.

# APLIKACIJA

Na osnovu predstavljenih dijagrama (use case I deployment) kreirana je aplikacija u React-u I Flask-u. U nastavku će biti predstavljen način implementiranja osnovnih funkcionalnosti te sam izvorni kod.

# IZVORNI KOD

# BACKEND

U app.py fajlu su implementirane sve relevantne funkcije vezane za backend dio aplikacije. U zaglavlju su importovane biblioteke flask za flask framework, flask\_socketio biblioteka za kreiranje i upravljanje socketima, te gtts biblioteka za pretvaranje teksta u govor koja je implemetirana na backendu.

from flask import Flask, request, jsonify, send\_file

from flask\_socketio import SocketIO,emit

from flask\_cors import CORS

from gtts import gTTS

Na samom početku fajla je inicijalizirana Flask aplikacija te je inicijalizirana SocketIO instanca:

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'secret!'

CORS(app,resources={r"/\*":{"origins":"\*"}})

socketio = SocketIO(app,cors\_allowed\_origins="\*")

Ispod SocketIO instance implemetirane su funckije: http\_call(), connected(), handle\_message(), disconnected(), te t2s() od kojih svaka predstavlja specifičan endpoint za različite operacije.

http\_call():

@app.route("/http-call")

def http\_call():

    """return JSON with string data as the value"""

    data = {'data':'This text was fetched using an HTTP call to server on render'}

    return jsonify(data)

U http\_call() funckiji je omogućena komunikacija HTTP protokolom.

connected():

Connected() funkcija se okida nakon što se korisnik spoji na socket, gdje se vrši samo emitovanje konekcije te se socket dodijeljuje jedinstven id (varijbla sid).

@socketio.on("connect")

@socketio.on("connect")

def connected():

    """event listener when client connects to the server"""

    print(request.sid)

    print("client has connected")

    emit("connect",{"data":f"id: {request.sid} is connected"})

handle\_message():

Handle\_Message funkcija se okida kada se bilo koji podatak vezan za socket promijeni što se u ovom slučaju dešava kada korisnik pošalje poruku. Poruci se dodjeljuje objekat data u kojem je smješten tekst poruke kao i vrijeme slanja poruke.

@socketio.on('data')

def handle\_message(data):

    """event listener when client types a message"""

    print("data from the front end: ",str(data))

    emit("data",{'data':data,'id':request.sid},broadcast=True)

disconnected():

Disconnected funckija se okida kada korisnik napusti chat sobu te se socket konekcija prekida.

@socketio.on("disconnect")

def disconnected():

    """event listener when client disconnects to the server"""

    print("user disconnected")

    emit("disconnect",f"user {request.sid} disconnected",broadcast=True)

t2s():

T2s() je posljednji endpoint koji omogućava komunikaciju sa Google Cloud servisom koji implementira POST I GET metode čije tijelo se sastoji od teksta poruke. Kao odgovor ovaj endpoint vraća audio file sa izgovorenim tekstom poruke.

@app.route("/tts", methods=['POST', 'GET'])

def t2s():

    text = request.get\_json()

    obj = gTTS(text = text, slow = False, lang = 'en')

    obj.save('audio.mp3')

    return send\_file('audio.mp3')

Na samom kraju je omogućeno pokretanje aplikacije i socketa na portu 5000.

if \_\_name\_\_ == '\_\_main\_\_':

    socketio.run(app, debug=True,port=5000)

# FRONTEND

U ovom poglavlju su opisani najbiniji dijelovi izvornog koda Frontend dijela alikacije. Frontend dio aplikacije se sastoji od App.js I WebSocket.jsx komponent u kojma je implementirana većina korisničkog interfejsa. U prvom dijelu App.js fajla su definisani potrebni state-ovi (stanja) koja manipulišu korinsičkim interfejsom.

 const [socketInstance, setSocketInstance] = useState("");

  const [loading, setLoading] = useState(true);

  const [buttonStatus, setButtonStatus] = useState(false);

socketInstance – definiše stanje socket instance

loading - stanje učitavanja

buttonState – stanje dugmeta (Enter / Exit Chat)

Sljedeća funckija je funkcija handleClick() koja se okida na klik dugmeta za ulazak ili izlazak iz chat sobe.

 const handleClick = () => {

    if (buttonStatus === false) {

      setButtonStatus(true);

    } else {

      setButtonStatus(false);

    }

  };

U nastavku je definisan useEffectHook koji se okida na svaku promjenu buttonState stanja. Unutar ovog useEffecta je implemetirano spajanje na socket i diskonektovanje sa socketa odnosno ulazak ili izlazak iz chat sobe.

 useEffect(() => {

    if (buttonStatus === true) {

      const socket = io("http://localhost:5000/", {

        transports: ["polling"],

        cors: {

          origin: "http://localhost:3000/",

        },

      });

      setSocketInstance(socket);

      socket.on("connect", (data) => {

        console.log(data);

      });

      setLoading(false);

      socket.on("disconnect", (data) => {

        console.log(data)

      });

      return function cleanup() {

        socket.disconnect();

      };

    }

  }, [buttonStatus]);

Na samom kraju App.js fajla se nalasi JSX kod odnosno layout App komponente u kojoj se nalazi WebSocket komponenta o kojoj će biti govora nastavku.

 return (

    <div className="App">

      {!buttonStatus ? (

        <button className="chat-button " onClick={handleClick}>Enter Chat</button>

      ) : (

        <div style={{display:"flex", flexDirection:"column", alignItems:"center", justifyContent:"center"}}>

          <button className="chat-button exit"  onClick={handleClick}>Exit Chat</button>

          <div className="line">

            {!loading && <WebSocket socket={socketInstance} />}

          </div>

        </div>

      )}

    </div>

  );

U WebSocket.jsx komponetni je implementiran grafički interfejs chat sobe kao i upravljanje slanja I primanja poruk kao i prisup Google Cloud servisima. Na početku su definisana stanja a jednu poruku i za cijeli niz poruka.

  const [message, setMessage] = useState('');

  const [messages, setMessages] = useState([]);

Nakon toga je definisan useSpeechRecognition() objekat koji služi za pristup Google STT servisu. Sastoji se od varijabli transcript – u koju se smješta dobijeni tekst kao odgovor I listening koja označava da li je unos govora još uvijek aktivan.

  const { transcript, listening } = useSpeechRecognition();

U nastavku je definisan useEffect u kojem je implementirana funkcionalnost slanja poruke.

  useEffect(() => {

    socket.on('data', (data) => {

      let message = { ...data, time: new Date() };

      setMessages([...messages, message]);

    });

    return () => {

      socket.off('data', () => {

        console.log('data event was removed');

      });

    };

  }, [socket, messages]);

Na samom kraju je implementiran korisnički interfejs WebSocket komponente u kojem su definisani API pozivi I za TTS I STT service.

    <div>

      <section className="msger">

        <header className="msger-header">

          <div className="msger-header-title">

            <i className="fas fa-comment-alt"></i> Multimedijalni sistemi i aplikacije

          </div>

          <div className="msger-header-options">

            <span>

              <i className="fas fa-cog"></i>

            </span>

          </div>

        </header>

        <main className="msger-chat">

          {messages.length > 0 ? (

            messages.map((message, ind) => {

              return (

                <div

                  className={socket.id === message.id ? 'msg right-msg' : 'msg left-msg'}

                  key={ind}

                  ref={messagesEndRef}

                >

                  <div className="msg-img">

                    <FontAwesomeIcon icon={faUser} className="user-icon" />

                  </div>

                  <div className="msg-bubble">

                    <div className="msg-info">

                      <div className="msg-info-name">{message.id}</div>

                      <div className="msg-info-time">

                        {message.time.getHours() + ':' + message.time.getMinutes()}

                      </div>

                    </div>

                    <div className="msg-text">{message.data}</div>

                  </div>

                  <FontAwesomeIcon

                    className="play-tts"

                    style={{ cursor: 'pointer' }}

                    icon={play && clicked === ind ? faCirclePause : faCirclePlay}

                    onClick={() => {

                      if (!play) {

                        const requestOptions = {

                          method: 'POST',

                          headers: { 'Content-Type': 'application/json' },

                          body: JSON.stringify(message.data),

                        };

                        fetch('http://localhost:5000/tts', requestOptions)

                          .then((response) => response.blob())

                          .then((data) => {

                            const objectURL = URL.createObjectURL(data);

                            audio = new Audio(objectURL);

                            audio.play();

                          });

                      } else {

                        if (audio) {

                          audio.pause();

                        }

                      }

                      setPlay(!play);

                      setClicked(ind);

                    }}

                  />

                </div>

              );

            })

          ) : (

            <></>

          )}

        </main>

        <div className="msger-inputarea">

          <input

            type="text"

            className="msger-input"

            placeholder="Enter your message..."

            value={message}

            onChange={handleText}

            onKeyPress={handleKeyPress}

          />

          <button className="msger-send-btn" onClick={handleSubmit}>

            Send

          </button>

          <FontAwesomeIcon

            icon={listening ? faMicrophone : faMicrophoneSlash}

            onClick={() => {

              listening ? SpeechRecognition.stopListening() : SpeechRecognition.startListening();

            }}

          />

        </div>

      </section>

    </div>

# KORISNIČKI INTERFEJS

Osnovna funkcionalnost aplikacije jeste primanje i slanje tekstulnih poruka u okviru chat aplikacije sa mogućnošću njihovog preslušavanja po potrebi. Komunikacija je ostvarena pomoću web socketa i ranije pomenutih HTTP, TCP i UDP protokola. User Interface ovog segemnta aplikacije su prikazani na sljedećim slikama:

A picture containing text

Description automatically generated

Slika 10 Ulazak u chat sobu

Graphical user interface, text, application, chat or text message

Description automatically generated

Slika 11 Slanje poruke

Sljedeća funkcionalnost jeste korištenje mikrofona za unos govora koje se nakdnadno uz pomoc Google Speech Recognition API-a pretvara u text. Način funkcioniranja i UI je prikazan na sljedećoj slici.

Graphical user interface, text, application, chat or text message

Description automatically generated

Slika 12 Unos teksta poruke putem mikrofona

Sljedeća funkcionalnost jeste mogućnost glasovnog preslušavanja primljene ili poslane pourke. Način funkcioniranja i UI je prikazan na sljedećoj slici.

Graphical user interface, text, application, chat or text message

Description automatically generated

Slika 13 Glasovno preslušavanje poruke

# WIRESHARK ANALIZA

U ovom poglavlju će detaljnije biti govora o wireshark-u i wireshark analizi paketa prilikom pretavaranja teksta u govor i obrnuto, što predstavlja jedan od najbitnijih dijelova u sklopu ovog projekta.

Wireshark je besplatni analizator paketa otvorenog koda. Koristi se za rješavanje problema s mrežom, analizu, razvoj softvera i komunikacijskih protokola i edukaciju. Prvobitno nazvan Ethereal, projekat je preimenovan u Wireshark u maju 2006. zbog problema sa zaštitnim znakom.

Wireshark je višeplatformski, koristi Qt widget alat u trenutnim izdanjima za implementaciju svog korisničkog interfejsa i koristi pcap za hvatanje paketa; radi na Linuxu, macOS-u, BSD-u, Solarisu, nekim drugim operativnim sistemima sličnim Unixu i Microsoft Windows-u. Postoji i verzija bazirana na terminalu (ne-GUI) pod nazivom TShark. Wireshark i drugi programi koji se distribuiraju s njim, kao što je TShark, su besplatni softver, objavljen pod uslovima GNU General Public License verzije 2 ili bilo koje novije verzije.

Wireshark je vrlo sličan tcpdumpu, ali ima grafički front-end i integrirane opcije sortiranja i filtriranja.

Wireshark dozvoljava korisniku da stavi kontrolere mrežnog interfejsa u promiskuitetni režim (ako ga podržava kontroler mrežnog interfejsa), tako da može da vidi sav saobraćaj vidljiv na tom interfejsu, uključujući jednostruki saobraćaj koji nije poslat na MAC adresu tog kontrolera mrežnog interfejsa. Međutim, kada se vrši hvatanje analizatorom paketa u promiskuitetnom modu na portu na mrežnom komutatoru, ne mora se sav promet kroz komutirati na port na kojem je hvatanje obavljeno, tako da hvatanje u promiskuitetnom načinu rada nije nužno dovoljno da se vidi sva mreža saobraćaja. Preslikavanje portova ili različiti mrežni dodiri proširuju snimanje na bilo koju tačku na mreži. Jednostavne pasivne slavine su izuzetno otporne na neovlašteno neovlašteno korištenje.

Na Linuxu, BSD-u i macOS-u, s libpcap 1.0.0 ili novijim, Wireshark 1.4 i noviji također mogu staviti kontrolere bežičnog mrežnog sučelja u nadzorni mod.

Ako udaljena mašina uhvati pakete i pošalje uhvaćene pakete mašini na kojoj radi Wireshark koristeći TZSP protokol ili protokol koji koristi OmniPeek, Wireshark secira te pakete, tako da može analizirati pakete uhvaćene na udaljenoj mašini u trenutku kada su uhvaćeni.

U nastavku je odrađena Wireshark analiza paketa prilikom pozivanja speech-to-text i text-to-speech API-a, te prilikom slanja chat poruka. Nakon pokretanja Wiresharka, potrebno je odabrati konekciju na kojoj želimo da analiziramo pakete.

Shape

Description automatically generated with low confidence Za demonstrativne potrebe odabran je WI-Fi interfejs te su na sljedećoj slici prikazani svi uhvaćeni paket ina Wi-Fi konekciji.

Text

Description automatically generated with medium confidence

S obzirom da je aplikacija pokrenuta lokalno, u našem slučaju potrebno je odabrati interfejs „Adapter for loopback traffic capture“, kako bi se mogli dohvatiti lokalni paketi. U nastavku će biti prikazani detalji sljedećih paketa:spajanje na socket, slanje poruke, pretvorba govora u tekst te pretvorba teksta u govor.

Za početak će biti izlistani svi paketi pokrenute aplikacije:

Graphical user interface, text

Description automatically generated Analizom možemo utvrditi da se cijeli saobraćaj odvija putem HTTP, TCP i UDP protokola. U nastavku će biti izdvojeni paketi ranije navedeni sa detaljnim opisom.

**Spajanje na socket:**

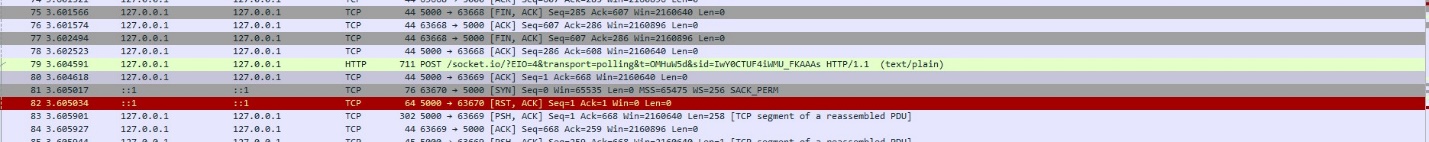
Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

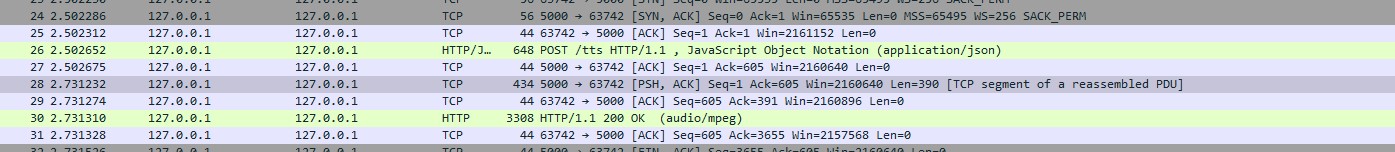
**Slanje poruke:**



Text

Description automatically generated with medium confidence

**Pretvaranje govora u tekst**



**Pretvaranje teksta u govor**

Graphical user interface, text, application

Description automatically generated

Iz wireshark analize možemo zakljjučiti da pretvorba text-a u govor zahtijeva najviše kapaciteta te da je potreban najveći broj paketa kako bi se ovaj proces realizovao.

# POKRETANJE APLIKACIJE

1. Nakon kloniranja projekta, unutar foldera se mogu naći Backend i Frontend folderi.
2. Cijeli folder je potrebno otvoriti u Visual Studio Code.
3. U integrated terminalu je potrebno otvoriti Backend folder te nakon toga unijeti:

env\Scripts\activate.bat

1. Nakon toga je poterbno unijeti komandu flask run te je backend dio time pokrenut
2. Sljedeći korak je otvaranje Frontend foldera u integrated terminalu te kucanje komande cd chatapp
3. Posljednji korak je komanda npm start čime je pokrenut i Frontend dio aplikacije.