Univerzitet u Zenici

Politehnički fakultet

Odsjek: Softversko inženjerstvo

Predmet: Multimedijalni sistemi i aplikacije

# SPEECH TO TEXT APP – Projektna DokumentaCija

Profesori: v. prof. dr. Alen Begović,

Doc.dr. Nermin Goran

Studenti: Ajna Fetić, Dženan Bejdić

Br. indexa: II-64, II-68

# O PROJEKTU

Ideja projekta speech-to-text app predstavlja softvrerski produkt čija je osnovna funkcionalnost chat applikacija sa opcijom pretvaranja govora u text i obrnuto. Komunikacija između korisnika bila bi omogućena preko web socketa, te TCP i UDP protokola. Osnovni fokus projekta je Wireshark analiza paketa prilikom pretvorbe govora u text i obrnuto. U narednih nekoliko poglavlja će biti opisane glavne značajke te osnovne funkcionalnosti aplikacije.

# SPEECH TO TEXT

Prepoznavanje govora je podoblast računarstva i računarske lingvistike koja razvija metodologije i tehnologije koje omogućavaju prepoznavanje i prevođenje govornog jezika u tekst od strane računara uz glavnu prednost mogućnosti pretraživanja. Poznato je i kao automatsko prepoznavanje govora (ASR), kompjutersko prepoznavanje govora ili govor u tekst (STT). Uključuje znanja i istraživanja u oblasti računarstva, lingvistike i računarskog inženjerstva. Obrnuti proces je sinteza govora. Neki sistemi za prepoznavanje govora zahtijevaju "obuku" (takođe zvanu "upis") gdje pojedinačni govornik čita tekst ili izolovani vokabular u sistem. Sistem analizira specifičan glas osobe i koristi ga za fino podešavanje prepoznavanja govora te osobe, što rezultira povećanom preciznošću. Sistemi koji ne koriste obuku nazivaju se "speaker-independent" sistemi. Sistemi koji koriste obuku nazivaju se "zavisni od govornika". Aplikacije za prepoznavanje govora uključuju glasovna korisnička sučelja kao što su glasovno biranje (npr. "pozovi kući"), usmjeravanje poziva (npr. "Želim izvršiti poziv"), kontrolu uređaja, pretraživanje ključnih riječi (npr. pronaći podcast gdje određene riječi bili izgovoreni), jednostavan unos podataka (npr. unos broja kreditne kartice), priprema strukturiranih dokumenata (npr. radiološki izvještaj), određivanje karakteristika govornika, obrada govora u tekst (npr. programi za obradu teksta ili e-poruke), i avion (obično se naziva direktni glasovni unos). Termin prepoznavanje glasa ili identifikacija govornika odnosi se na identifikaciju govornika, a ne na ono što govore. Prepoznavanje govornika može pojednostaviti zadatak prevođenja govora u sistemima koji su obučeni za glas određene osobe ili se može koristiti za autentifikaciju ili provjeru identiteta govornika kao dio sigurnosnog procesa. Iz perspektive tehnologije, prepoznavanje govora ima dugu istoriju sa nekoliko talasa velikih inovacija. Nedavno je ovo polje imalo koristi od napretka dubokog učenja i velikih podataka. O napretku svjedoči ne samo nalet akademskih radova objavljenih u ovoj oblasti, već što je još važnije usvajanje različitih metoda dubokog učenja u svjetskoj industriji u dizajniranju i primjeni sistema za prepoznavanje govora. U ovomo projektu korišten je Google Speech Recognition API kao alat za prepoznavanje govora i pretaranja u text, o čemu će više biti govora u narednim poglavljima.

# TEXT TO SPEECH

Text to speech se još naziva i sinteza govora. Sinteza govora je umjetna proizvodnja ljudskog govora. Računarski sistem koji se koristi za ovu svrhu naziva se sintisajzer govora i može se implementirati u softverske ili hardverske proizvode. Sistem za pretvaranje teksta u govor (TTS) pretvara tekst normalnog jezika u govor; drugi sistemi prikazuju simboličke lingvističke reprezentacije poput fonetskih transkripcija u govor. Obrnuti proces je prepoznavanje govora. Sintetizirani govor se može kreirati spajanjem dijelova snimljenog govora koji su pohranjeni u bazi podataka. Sistemi se razlikuju po veličini pohranjenih govornih jedinica; sistem koji skladišti telefone ili difone pruža najveći izlazni opseg, ali može mu nedostajati jasnoća. Za specifične domene upotrebe, skladištenje čitavih reči ili rečenica omogućava visokokvalitetan izlaz. Alternativno, sintisajzer može inkorporirati model vokalnog trakta i drugih karakteristika ljudskog glasa kako bi stvorio potpuno "sintetički" glasovni izlaz. Sistem za pretvaranje teksta u govor (ili "motor") se sastoji od dva dijela: front-end i back-end. Front-end ima dva glavna zadatka. Prvo, pretvara sirovi tekst koji sadrži simbole poput brojeva i skraćenica u ekvivalent ispisanih riječi. Ovaj proces se često naziva normalizacija teksta, prethodna obrada ili tokenizacija. Front-end zatim svakoj riječi dodjeljuje fonetske transkripcije, te dijeli i obilježava tekst u prozodijske jedinice, kao što su fraze, rečenice i rečenice. Proces dodjeljivanja fonetskih transkripcija riječima naziva se konverzija teksta u fonem ili grafem u fonem. Fonetske transkripcije i informacije o prozodiji zajedno čine simboličku lingvističku reprezentaciju koju daje front-end. Pozadina — koja se često naziva sintisajzerom — zatim pretvara simboličku lingvističku reprezentaciju u zvuk. U određenim sistemima, ovaj dio uključuje izračunavanje ciljne prozodije (kontura tona, trajanje fonema), koja se zatim nameće izlaznom govoru.

Diagram

Description automatically generated

Slika 1 Princip TTS Sistema

U sklopu ovog projekta kao alat za pretvaranje text-a u govor korišten je Google text-speech api o čemu će biti detaljnijeg govora u narednim poglavljima.

# TCP PROTOKOL

Osnovni fokus projekta jeste, kao što je već rečeno na analizi paketa i analizi komunikacije koja se 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.

Table

Description automatically generated

Slika 2 TCP protokol u slojevima

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.

# 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.

A picture containing timeline

Description automatically generated

Slika 3 Razlika između TCP i UDP protokola

# OSNOVNE FUNKCIONALNOSTI I NAČIN FUNKCIONIRANJA APLIKACIJE

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 TCP i UDP protokola. User Interface ovog segemnta aplikacije su prikazani na sljedećim slikama:

A picture containing text

Description automatically generated

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

Description automatically generated

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 sljedećim slikama.

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

Description automatically generated

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

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

Description automatically generated

# KORIŠTENI ALATI I NAČIN IMPLEMENTACIJE

# BACKEND

* 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.

U nastavku se nalaze najbitniji dijelovi programkog koda backend dijela aplikacije:

from flask import Flask, request,jsonify

from flask\_socketio import SocketIO,emit

from flask\_cors import CORS

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

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

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

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

@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)

@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"})

@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)

@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)

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

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

U prvom dijelu progamskog koda inicijalizirana je flask aplikacija na portu 5000. Za povezivanje socketa korištena je socket.io biblioteka. Implementirane su funkicje http-call za inicijalizaciju protokola, connected koja se izvršava nakon spajanja socketa, handle\_message za obradu pristiglih i poslanih poruka, te diconnected koja se izvršava nakon izlaska iz chata i deaktivacije socketa.

* GOOGLE TEXT-TO-SPEECH

Google Cloud Text-to-Speech omogućava programerima da sintetiziraju govor prirodnog zvučanja sa 100+ glasova, dostupnih na više jezika i varijanti. Primjenjuje revolucionarno istraživanje DeepMind-a u WaveNet-u i Google-ovim moćnim neuronskim mrežama kako bi pružio najveću moguću vjernost. Kao API jednostavan za korištenje, moguće je stvoriti realistične interakcije sa svojim korisnicima, na mnogim aplikacijama i uređajima.Amazon Polly je certificiran za korištenje s reguliranim radnim opterećenjima za HIPAA (Zakon o prenosivosti i odgovornosti zdravstvenog osiguranja iz 1996.) i Standard sigurnosti podataka industrije platnih kartica (PCI DSS).

Neke od glavnih značajki text-to-sppech API-ja su:

WaveNet glasovi, Neural2 glasovi, custom glasovi, modifikovanje glasova, text i SSML podrška.

Datoteke audio podataka koje se kreiraju pomoću pretvaranja teksta u govor mogu se koristiti za pokretanje vaših aplikacija ili povećanje medija kao što su video ili audio snimci (u skladu s Uslovima korištenja usluge Google Cloud Platform, uključujući usklađenost sa svim primjenjivim zakonima).

Text-to-Speech pretvara tekst ili unos jezika za označavanje sinteze govora (SSML) u audio podatke kao što su MP3 ili LINEAR16 (kodiranje koje se koristi u WAV datotekama). U nastavku se nalazi flask endpoint za pretvorbu text-a u govor:

@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')

* FRONTEND

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.

U nastavku se nalazi najbitniji dijelovo programkog koda frontend dijela aplikacije kreiran u React.js.

App.js file:

import "./App.css";

import { useEffect, useState } from "react";

import WebSocket from "./components/WebSocket";

import { io } from "socket.io-client";

function App() {

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

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

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

  const handleClick = () => {

    if (buttonStatus === false) {

      setButtonStatus(true);

    } else {

      setButtonStatus(false);

    }

  };

  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]);

  return (

    <div className="App">

      {!buttonStatus ? (

        <button onClick={handleClick}>Enter Chat</button>

      ) : (

        <>

          <button onClick={handleClick}>Exit Chat</button>

          <div className="line">

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

          </div>

        </>

      )}

    </div>

  );

}

export default App;

Komponenta WebSocket.jsx:

import SpeechRecognition, { useSpeechRecognition } from 'react-speech-recognition';

import { faMicrophone, faMicrophoneSlash } from '@fortawesome/free-solid-svg-icons';

import { useEffect, useState } from 'react';

import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';

export default function WebSocket({ socket }) {

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

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

    const {

    transcript,

    listening,

  } = useSpeechRecognition();

  useEffect(()=>{

   setMessage(transcript)

  }, [transcript])

  const handleText = (e) => {

    const inputMessage = e.target.value;

    setMessage(inputMessage);

  };

  const handleSubmit = () => {

    if (!message) {

      return;

    }

    socket.emit('data', message);

    setMessage('');

  };

  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]);

  return (

    <div>

      <section className="msger">

        <header className="msger-header">

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

            <i className="fas fa-comment-alt"></i> SimpleChat

          </div>

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

            <span>

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

            </span>

          </div>

        </header>

        <main className="msger-chat">

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

            return (

              <div

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

                key={ind}

              >

                <div className="msg-img"></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>

              </div>

            );

          })}

        </main>

        <div className="msger-inputarea">

          <input

            type="text"

            className="msger-input"

            placeholder="Enter your message..."

            value={message}

            onChange={handleText}

          />

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

            Send

          </button>

          <FontAwesomeIcon

          icon={

           listening ? faMicrophone : faMicrophoneSlash

          }

          onClick={()=>{listening ? SpeechRecognition.stopListening() : SpeechRecognition.startListening()}}

        />

        </div>

      </section>

    </div>

  );

}

Najbitnije segment frontend dijela aplikacije jeste Google Speech Recognition API. Zahtjev za sinhrono prepoznavanje API-ja govora u tekst je najjednostavniji metod za izvođenje prepoznavanja govornih audio podataka. Speech-to-Text može obraditi do 1 minute govornih audio podataka poslatih u sinkronom zahtjevu. Nakon što Govor u tekst obradi i prepozna sav zvuk, vraća odgovor.

Sinhroni zahtjev je blokiran, što znači da govor u tekst mora vratiti odgovor prije obrade sljedećeg zahtjeva. Govor u tekst obično obrađuje zvuk brže od realnog vremena, obrađujući 30 sekundi zvuka u prosjeku za 15 sekundi. U slučajevima lošeg kvaliteta zvuka, zahtjev za prepoznavanje može potrajati znatno duže.

* WIRESHARK ANALIZA PAKETA

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 TCP i UDP protokola. U nastavku će biti izdvojeni paketi ranije navedeni sa detaljnim opisom.

* Spajanje na socket:

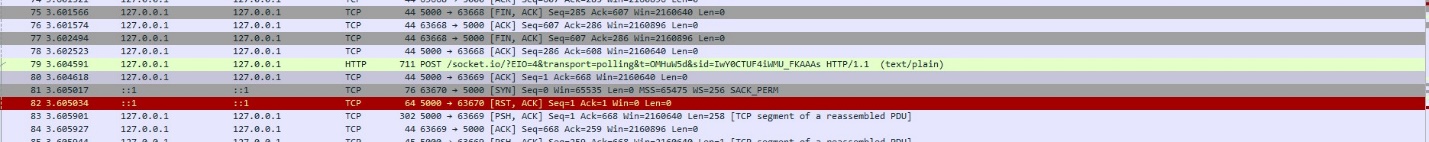
Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

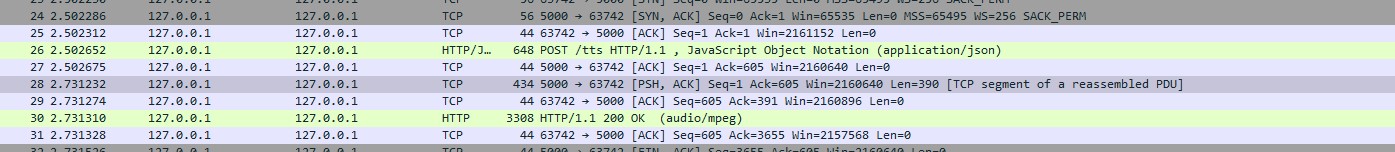
* Slanje poruke:



Text

Description automatically generated with medium confidence

* Pretvaranje govora u text:



* Pretvaranje texta 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 applikacije.