NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY

ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS

(CT-361)

CCP PROJECT REPORT

GROUP MEMBERS:

- **AREEBA BATOOL (CT-22059)**
- **SYEDA SHINZA WASIF (CT-22063)**
- **ASIFA SIRAJ (CT-22070)**

SUBMITTED TO:

- **DR. WASEEMULLAH**
- **SIR MUHAMMAD AHMED ZAKI**
- **SIR MUHAMMAD ABDULLAH SIDDIQUI**

PROJECT REPORT:

AI-Based Chatbot for Hospital Recommendation

1. Project Title:

AI-Based Smart Chatbot for Hospital Recommendation

2. Project Description:

The proposed project aims to develop an AI-driven chatbot that assists users in finding the most suitable hospital based on their specific requirements. The chatbot will take inputs such as consultation fees, disease type, location, and availability of doctors and hospital timings. Utilizing machine learning algorithms and natural language processing (NLP), the chatbot will analyze user queries and provide relevant hospital recommendations. This system aims to enhance healthcare accessibility, reduce search time, and provide reliable suggestions tailored to the user's needs.

3. Project Methodology:

- **Dataset:** The project will utilize healthcare datasets containing information about hospitals, their specializations, consultation fees, operational hours, and geographic locations. Publicly available datasets from healthcare organizations and web-scraped data from hospital websites will be used.

- **Tools & Technology:** Python, TensorFlow/PyTorch, Natural Language Processing (NLP) libraries (spaCy, NLTK), Flask/Django for backend, React.js for frontend, Firebase/MySQL for database storage.

- **Algorithms:** NLP techniques for query processing, Decision Trees/Random Forests for hospital ranking, Geospatial analysis for location-based recommendations, and Chatbot frameworks like Dialogflow or Rasa for interaction.

- **Timeline:**
    - Week 1-2: Requirement analysis & dataset collection
    - Week 3-4: Data preprocessing & exploratory data analysis (EDA)
    - Week 5-6: Model selection and implementation
    - Week 7-8: Chatbot development and integration
    - Week 9-10: Testing & optimization
    - Week 11-12: Deployment & final project report

- **Objectives:**

- Develop an interactive chatbot that understands and processes natural language inputs.
- Implement AI-based decision-making for hospital recommendations.
- Ensure real-time and accurate responses for user queries.
- Provide location-based hospital recommendations.

- **Outcomes:**

- A fully functional chatbot capable of suggesting hospitals based on user preferences.
- An AI-powered system that improves healthcare accessibility.
- A scalable solution that can be expanded with more features.

- **Goals:**

- Enhance patient experience by reducing search time for hospitals.
- Provide accurate and reliable recommendations using AI.
- Improve healthcare decision-making through data-driven insights.

4. Code Explanation

- **Backend Code Explanation:**

❖ **App.py:**

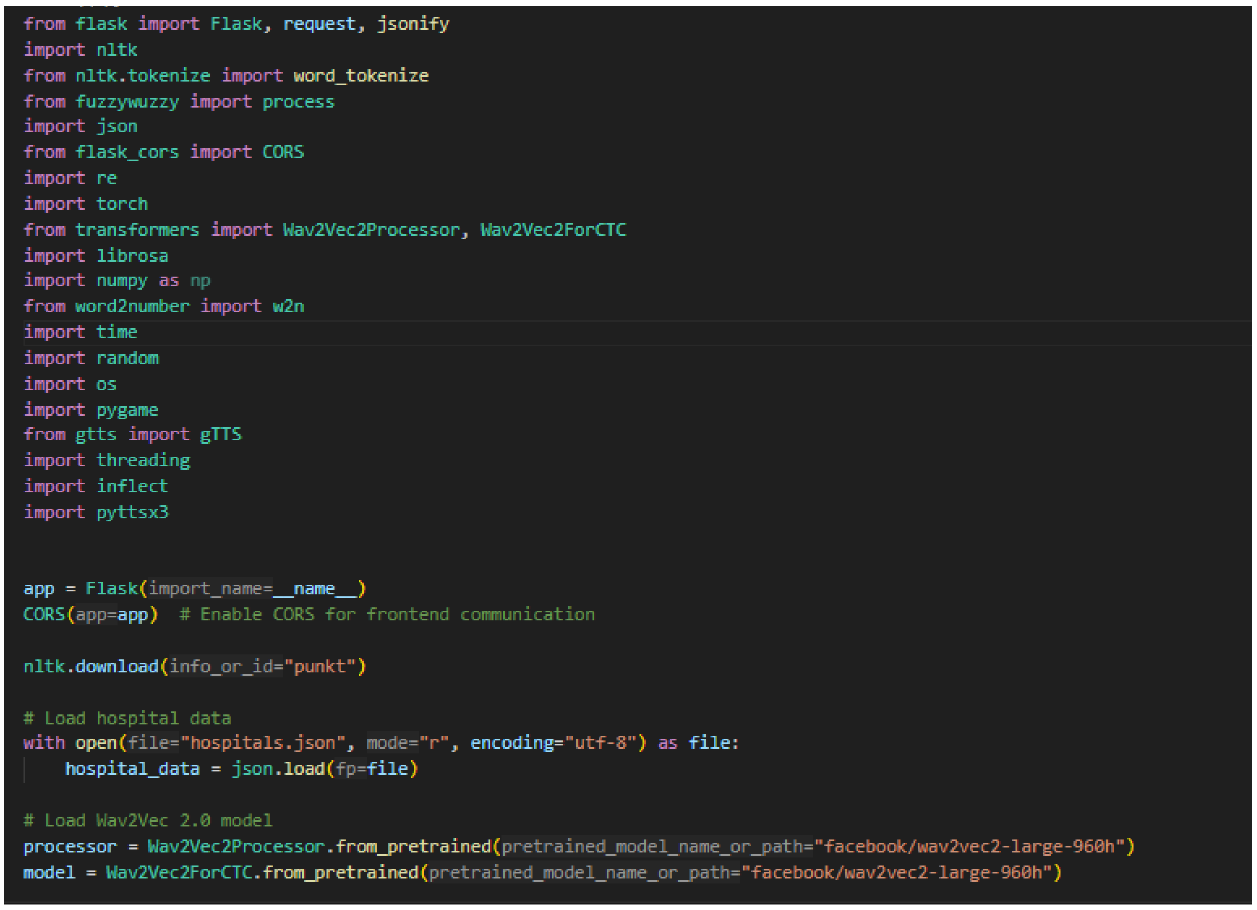
Purpose

- A smart voice-enabled chatbot that helps users find hospitals based on specialization, city, and fee range. It supports both text and audio input using advanced speech recognition with **Wav2Vec2** and provides results through voice feedback using **pyttsx3** or **gTTS**.

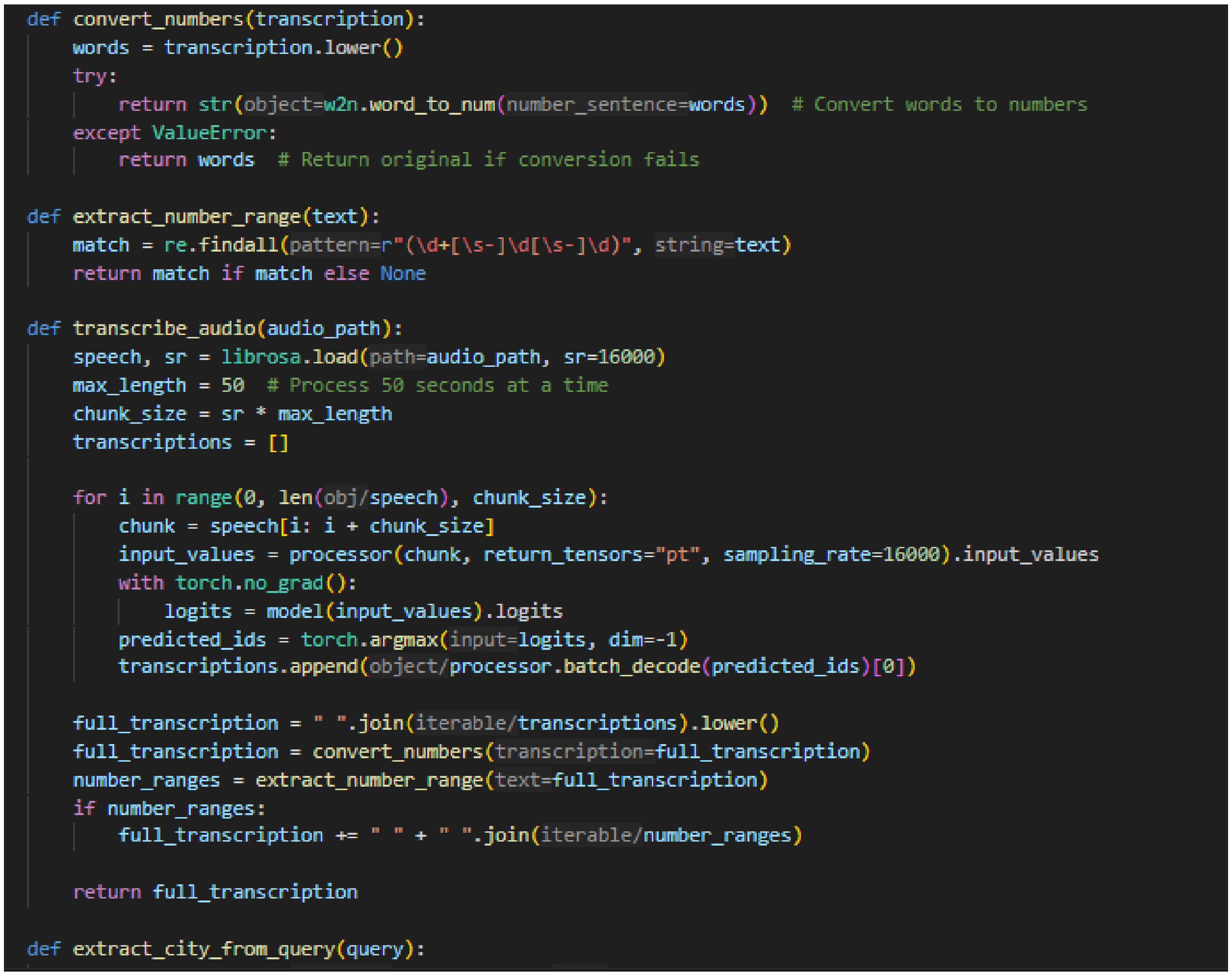
Process Flow

- **User Input**: Text or audio sent via POST request to `/chatbot`.
- **Audio Transcription**: If audio, it's transcribed using Wav2Vec2 model.
- **Query Parsing**:
  - City and specialization tokens are extracted.
  - Fee ranges are detected (e.g., “3000-5000” or “4000”).
- **Hospital Filtering**:
  - Exact and fuzzy matching based on specialization.
  - Filtered by city and fee constraints.
- **Response**:
  - JSON response with matching hospital details.
  - Background voice narration of the result.

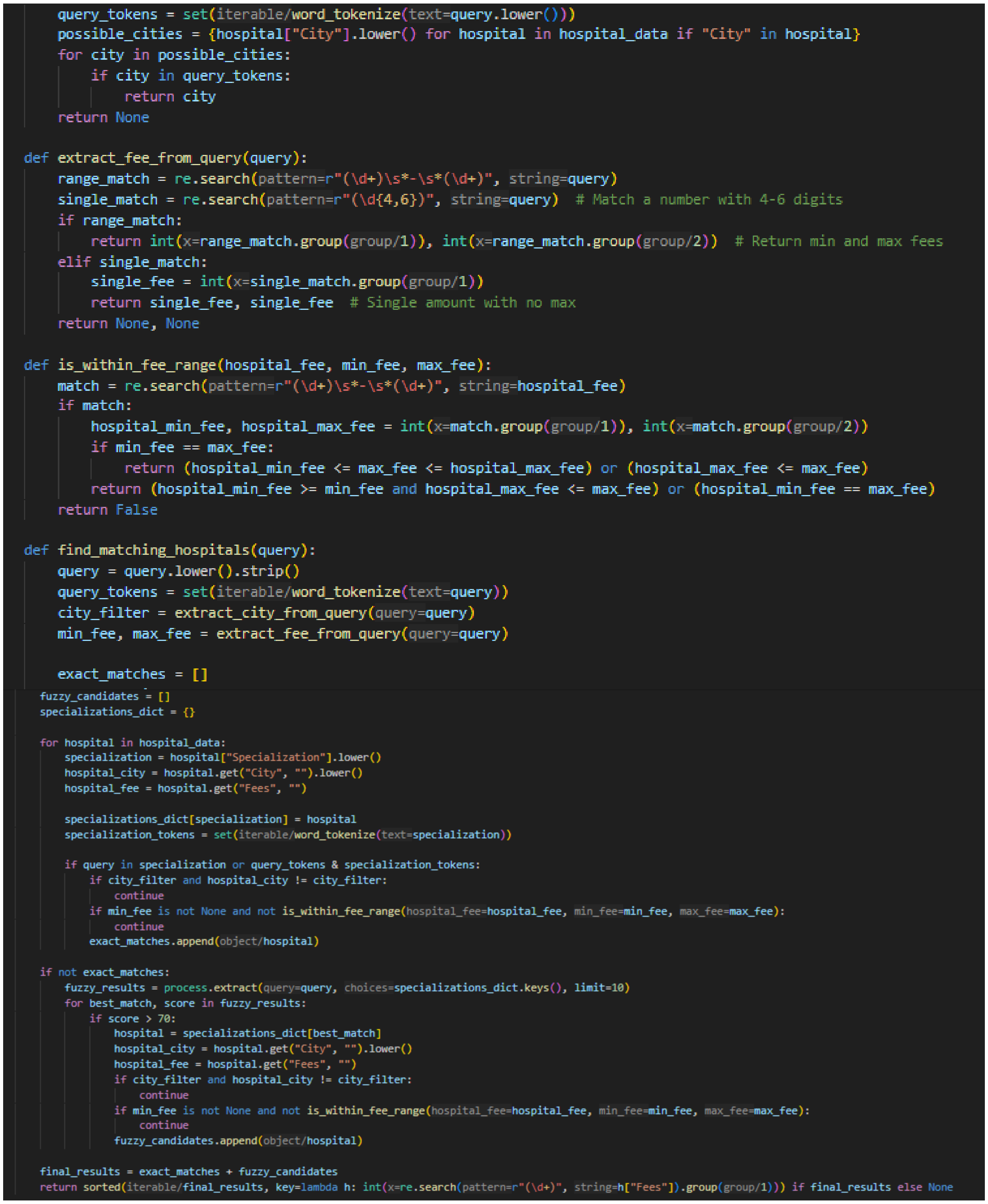
Source Code:



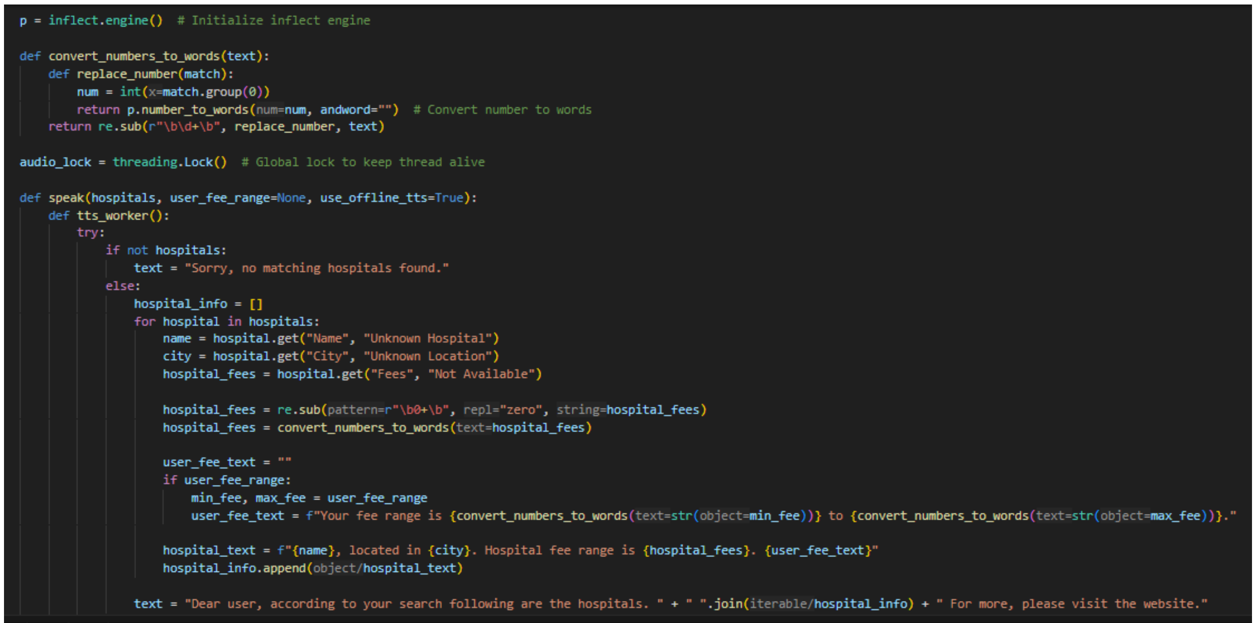
```python
from flask import Flask, request, jsonify
import nltk
from nltk.tokenize import word_tokenize
from fuzzywuzzy import process
import json
from flask_cors import CORS
import re
import torch
from transformers import Wav2Vec2Processor, Wav2Vec2ForCTC
import librosa
import numpy as np
from word2number import w2n
import time
import random
import os
import pygame
from gtts import gTTS
import threading
import inflect
import pyttsx3


app = Flask(import_name=__name__)
CORS(app=app)  # Enable CORS for frontend communication

nltk.download(info_or_id="punkt")

# Load hospital data
with open(file="hospitals.json", mode="r", encoding="utf-8") as file:
    hospital_data = json.load(fp=file)

# Load Wav2Vec 2.0 model
processor = Wav2Vec2Processor.from_pretrained(pretrained_model_name_or_path="facebook/wav2vec2-large-960h")
model = Wav2Vec2ForCTC.from_pretrained(pretrained_model_name_or_path="facebook/wav2vec2-large-960h")
```



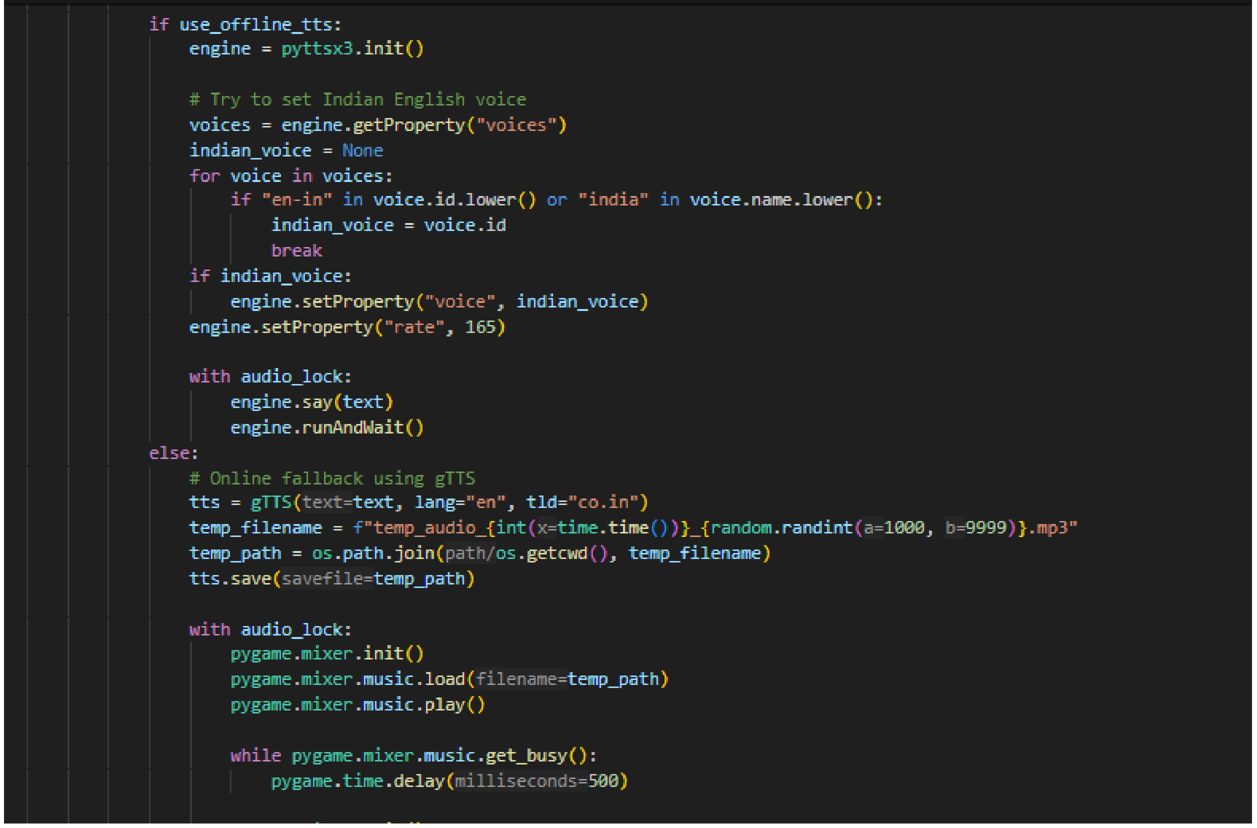
```
def convert_numbers(transcription):
    words = transcription.lower()
    try:
        return str(object=w2n.word_to_num(number_sentence=words))  # Convert words to numbers
    except ValueError:
        return words  # Return original if conversion fails

def extract_number_range(text):
    match = re.findall(pattern=r"(\d+[\s-]\d[\s-]\d)", string=text)
    return match if match else None

def transcribe_audio(audio_path):
    speech, sr = librosa.load(path=audio_path, sr=16000)
    max_length = 50  # Process 50 seconds at a time
    chunk_size = sr * max_length
    transcriptions = []

    for i in range(0, len(obj/speech), chunk_size):
        chunk = speech[i: i + chunk_size]
        input_values = processor(chunk, return_tensors="pt", sampling_rate=16000).input_values
        with torch.no_grad():
            logits = model(input_values).logits
        predicted_ids = torch.argmax(input=logits, dim=-1)
        transcriptions.append(object/processor.batch_decode(predicted_ids)[0])

    full_transcription = " ".join(iterable/transcriptions).lower()
    full_transcription = convert_numbers(transcription=full_transcription)
    number_ranges = extract_number_range(text=full_transcription)
    if number_ranges:
        full_transcription += " " + " ".join(iterable/number_ranges)

    return full_transcription

def extract_city_from_query(query):
```



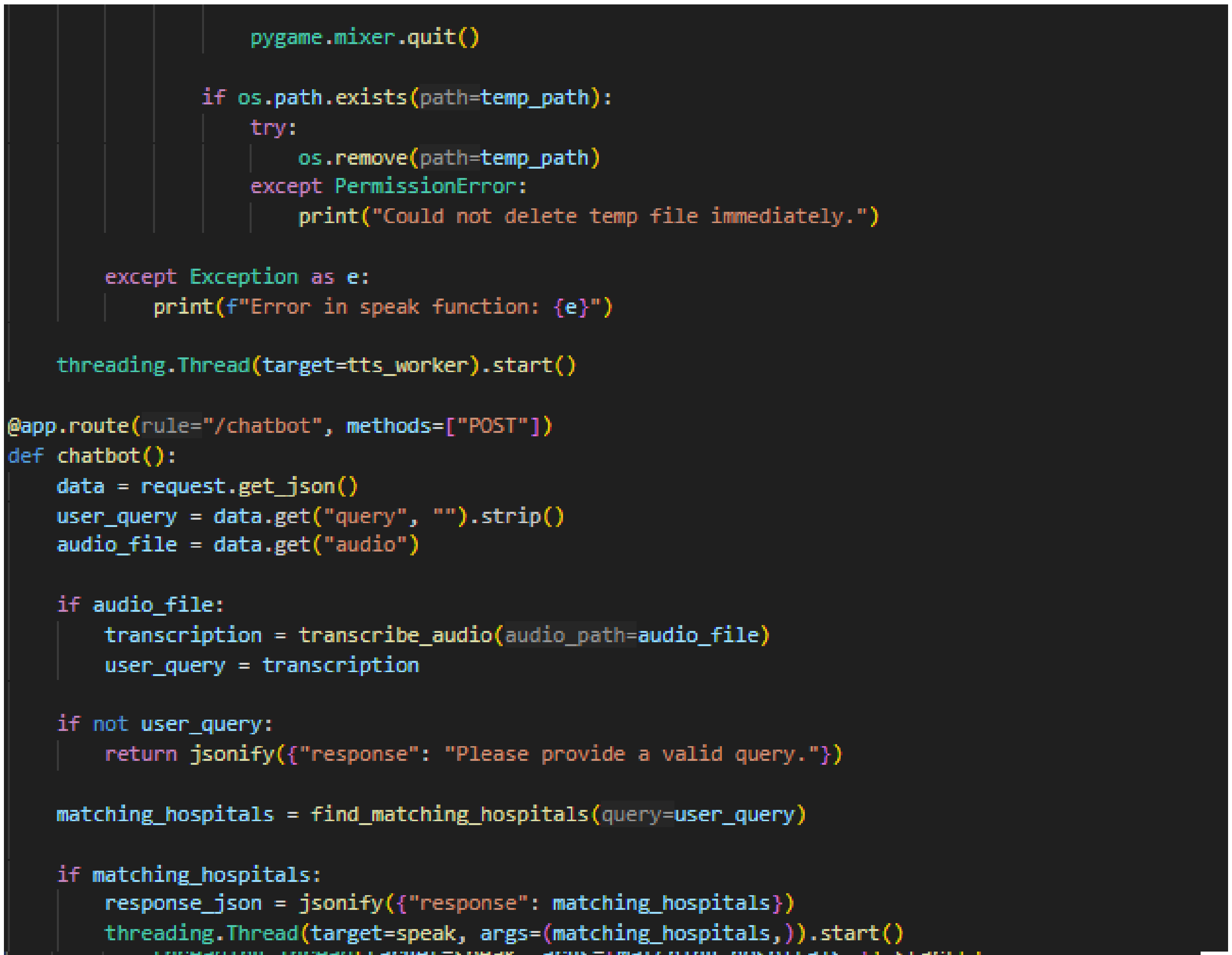
```python
    query_tokens = set(iterable/word_tokenize(text=query.lower()))
    possible_cities = {hospital["City"].lower() for hospital in hospital_data if "City" in hospital}
    for city in possible_cities:
        if city in query_tokens:
            return city
    return None

def extract_fee_from_query(query):
    range_match = re.search(pattern=r"(\d+)\s*-\s*(\d+)", string=query)
    single_match = re.search(pattern=r"(\d{4,6})", string=query)  # Match a number with 4-6 digits
    if range_match:
        return int(x=range_match.group(group/1)), int(x=range_match.group(group/2))  # Return min and max fees
    elif single_match:
        single_fee = int(x=single_match.group(group/1))
        return single_fee, single_fee  # Single amount with no max
    return None, None

def is_within_fee_range(hospital_fee, min_fee, max_fee):
    match = re.search(pattern=r"(\d+)\s*-\s*(\d+)", string=hospital_fee)
    if match:
        hospital_min_fee, hospital_max_fee = int(x=match.group(group/1)), int(x=match.group(group/2))
        if min_fee == max_fee:
            return (hospital_min_fee <= max_fee <= hospital_max_fee) or (hospital_max_fee <= max_fee)
        return (hospital_min_fee >= min_fee and hospital_max_fee <= max_fee) or (hospital_min_fee == max_fee)
    return False

def find_matching_hospitals(query):
    query = query.lower().strip()
    query_tokens = set(iterable/word_tokenize(text=query))
    city_filter = extract_city_from_query(query=query)
    min_fee, max_fee = extract_fee_from_query(query=query)

    exact_matches = []
    fuzzy_candidates = []
    specializations_dict = {}

    for hospital in hospital_data:
        specialization = hospital["Specialization"].lower()
        hospital_city = hospital.get("City", "").lower()
        hospital_fee = hospital.get("Fees", "")

        specializations_dict[specialization] = hospital
        specialization_tokens = set(iterable/word_tokenize(text=specialization))

        if query in specialization or query_tokens & specialization_tokens:
            if city_filter and hospital_city != city_filter:
                continue
            if min_fee is not None and not is_within_fee_range(hospital_fee=hospital_fee, min_fee=min_fee, max_fee=max_fee):
                continue
            exact_matches.append(object/hospital)

    if not exact_matches:
        fuzzy_results = process.extract(query=query, choices=specializations_dict.keys(), limit=10)
        for best_match, score in fuzzy_results:
            if score > 70:
                hospital = specializations_dict[best_match]
                hospital_city = hospital.get("City", "").lower()
                hospital_fee = hospital.get("Fees", "")
                if city_filter and hospital_city != city_filter:
                    continue
                if min_fee is not None and not is_within_fee_range(hospital_fee=hospital_fee, min_fee=min_fee, max_fee=max_fee):
                    continue
                fuzzy_candidates.append(object/hospital)

    final_results = exact_matches + fuzzy_candidates
    return sorted(iterable/final_results, key=lambda h: int(x=re.search(pattern=r"(\d+)", string=h["Fees"]).group(group/1))) if final_results else None
```



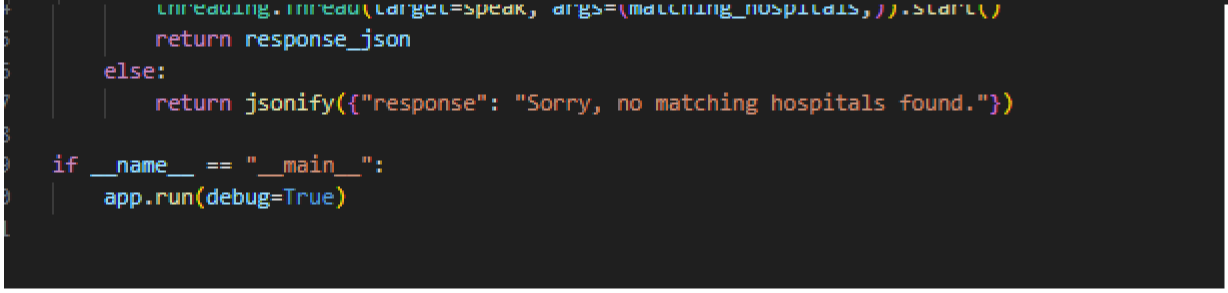
```python
p = inflect.engine()  # Initialize inflect engine

def convert_numbers_to_words(text):
    def replace_number(match):
        num = int(x=match.group(0))
        return p.number_to_words(num=num, andword="")  # Convert number to words
    return re.sub(r"\b\d+\b", replace_number, text)

audio_lock = threading.Lock()  # Global lock to keep thread alive

def speak(hospitals, user_fee_range=None, use_offline_tts=True):
    def tts_worker():
        try:
            if not hospitals:
                text = "Sorry, no matching hospitals found."
            else:
                hospital_info = []
                for hospital in hospitals:
                    name = hospital.get("Name", "Unknown Hospital")
                    city = hospital.get("City", "Unknown Location")
                    hospital_fees = hospital.get("Fees", "Not Available")

                    hospital_fees = re.sub(pattern=r"\b0+\b", repl="zero", string=hospital_fees)
                    hospital_fees = convert_numbers_to_words(text=hospital_fees)

                    user_fee_text = ""
                    if user_fee_range:
                        min_fee, max_fee = user_fee_range
                        user_fee_text = f"Your fee range is {convert_numbers_to_words(text=str(object=min_fee))} to {convert_numbers_to_words(text=str(object=max_fee))}."

                    hospital_text = f"{name}, located in {city}. Hospital fee range is {hospital_fees}. {user_fee_text}"
                    hospital_info.append(object/hospital_text)

                text = "Dear user, according to your search following are the hospitals. " + " ".join(iterable/hospital_info) + " For more, please visit the website."
```



```python
            if use_offline_tts:
                engine = pyttsx3.init()

                # Try to set Indian English voice
                voices = engine.getProperty("voices")
                indian_voice = None
                for voice in voices:
                    if "en-in" in voice.id.lower() or "india" in voice.name.lower():
                        indian_voice = voice.id
                        break
                if indian_voice:
                    engine.setProperty("voice", indian_voice)
                engine.setProperty("rate", 165)

                with audio_lock:
                    engine.say(text)
                    engine.runAndWait()
            else:
                # Online fallback using gTTS
                tts = gTTS(text=text, lang="en", tld="co.in")
                temp_filename = f"temp_audio_{int(x=time.time())}_{random.randint(a=1000, b=9999)}.mp3"
                temp_path = os.path.join(path/os.getcwd(), temp_filename)
                tts.save(savefile=temp_path)

                with audio_lock:
                    pygame.mixer.init()
                    pygame.mixer.music.load(filename=temp_path)
                    pygame.mixer.music.play()

                    while pygame.mixer.music.get_busy():
                        pygame.time.delay(milliseconds=500)
```



```python
                    pygame.mixer.quit()

                if os.path.exists(path=temp_path):
                    try:
                        os.remove(path=temp_path)
                    except PermissionError:
                        print("Could not delete temp file immediately.")

        except Exception as e:
            print(f"Error in speak function: {e}")

    threading.Thread(target=tts_worker).start()

@app.route(rule="/chatbot", methods=["POST"])
def chatbot():
    data = request.get_json()
    user_query = data.get("query", "").strip()
    audio_file = data.get("audio")

    if audio_file:
        transcription = transcribe_audio(audio_path=audio_file)
        user_query = transcription

    if not user_query:
        return jsonify({"response": "Please provide a valid query."})

    matching_hospitals = find_matching_hospitals(query=user_query)

    if matching_hospitals:
        response_json = jsonify({"response": matching_hospitals})
        threading.Thread(target=speak, args=(matching_hospitals,)).start()
```



```python
        return response_json
    else:
        return jsonify({"response": "Sorry, no matching hospitals found."})

if __name__ == "__main__":
    app.run(debug=True)
```

- **Frontend Code Explanation:**

❖ **Chatbot.jsx:**

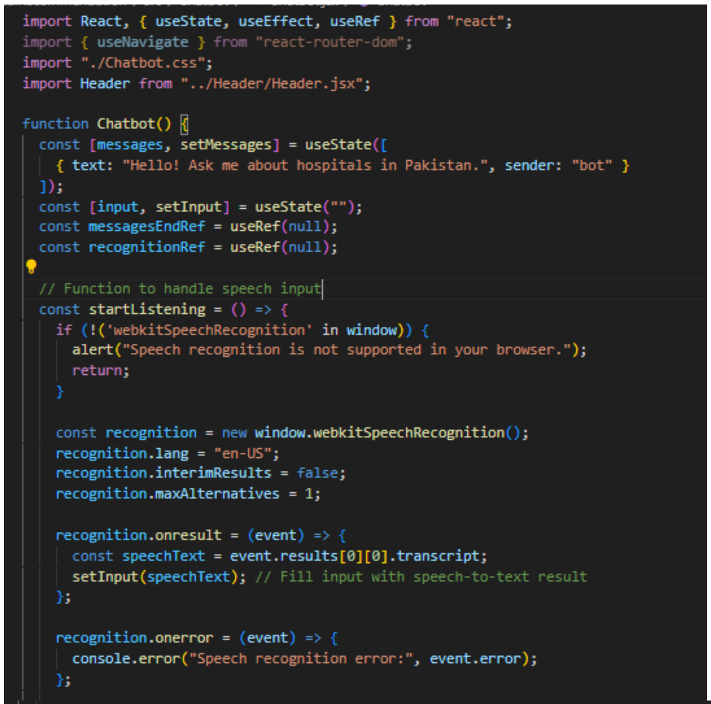
Purpose

- This React component provides a **user-friendly interface** for interacting with a voice-enabled chatbot that helps users find hospitals in Pakistan based on specialization, city, and fee range. It supports **text** and **speech input** using the browser's Web Speech API and displays detailed results with a clean UI.

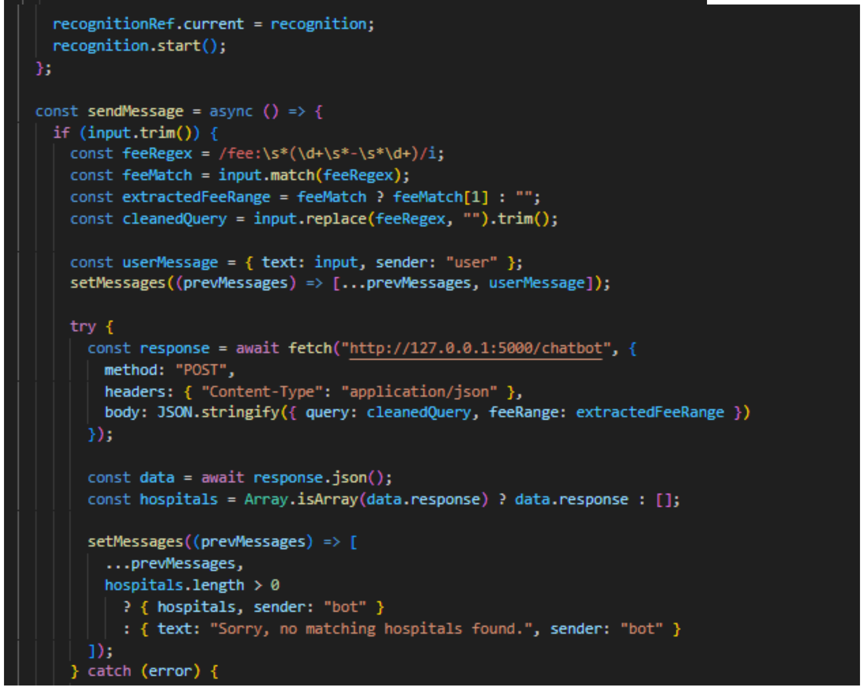
Process Flow

- **Initial State**:
  - Loads with a greeting from the bot: *"Hello! Ask me about hospitals in Pakistan."*
  - **User Interaction**:
  - User can type a query or click the **mic button** to speak.
  - Speech is transcribed via `webkitSpeechRecognition` and filled into the input box.
  - **Message Handling**:
  - On **send** (button press or Enter key), the app:
    - Extracts any fee range (e.g., `fee: 3000-5000`) from the input.
    - Sends a POST request to the Flask backend with:
      - `query`: text minus fee info
      - `feeRange`: extracted range
  - Displays user message immediately.
  - **Response Handling**:
  - Awaits backend JSON response.
  - If hospitals are returned:
    - Renders each hospital in a **card format** showing:
      - Name, City, Province, Specialization
      - Phone, Contact Person, Address, Website
      - Fee range
  - If no results found:
    - Displays: *"Sorry, no matching hospitals found."*
  - **UI Enhancements**:
  - Scrolls to the latest message.
  - Responsive buttons for sending messages or activating the mic.
  - Integrates a consistent **header** component.

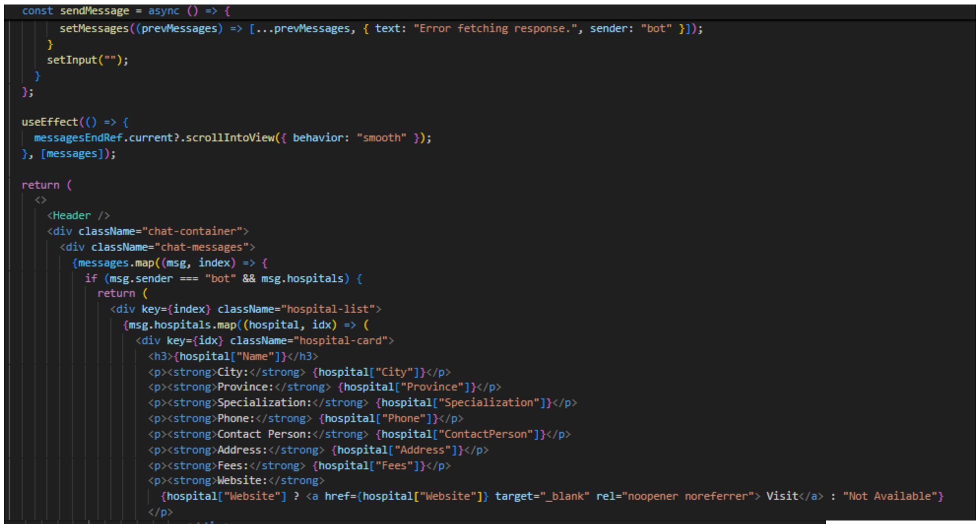
Source Code:



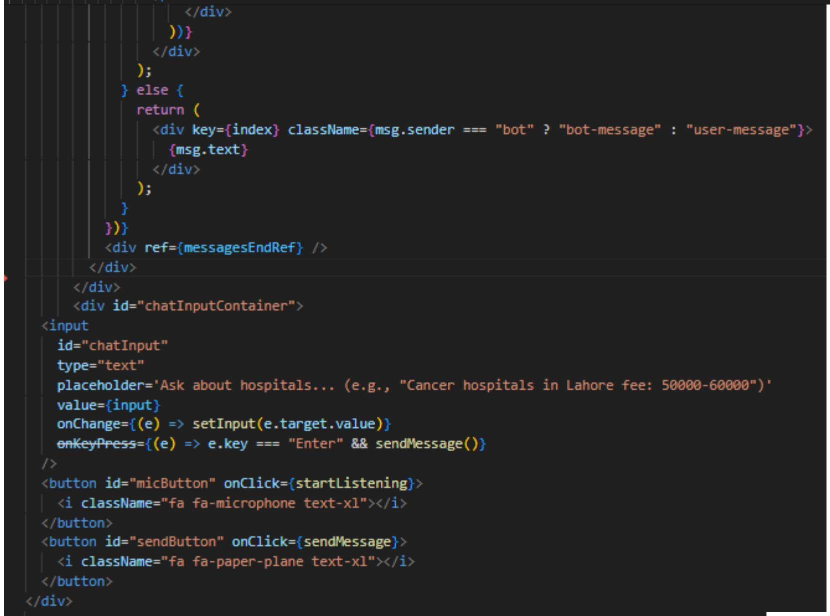
```
import React, { useState, useEffect, useRef } from "react";
import { useNavigate } from "react-router-dom";
import "./Chatbot.css";
import Header from "../Header/Header.jsx";

function Chatbot() {
  const [messages, setMessages] = useState([
    { text: "Hello! Ask me about hospitals in Pakistan.", sender: "bot" }
  ]);
  const [input, setInput] = useState("");
  const messagesEndRef = useRef(null);
  const recognitionRef = useRef(null);

  // Function to handle speech input
  const startListening = () => {
    if (!('webkitSpeechRecognition' in window)) {
      alert("Speech recognition is not supported in your browser.");
      return;
    }

    const recognition = new window.webkitSpeechRecognition();
    recognition.lang = "en-US";
    recognition.interimResults = false;
    recognition.maxAlternatives = 1;

    recognition.onresult = (event) => {
      const speechText = event.results[0][0].transcript;
      setInput(speechText); // Fill input with speech-to-text result
    };

    recognition.onerror = (event) => {
      console.error("Speech recognition error:", event.error);
    };
```



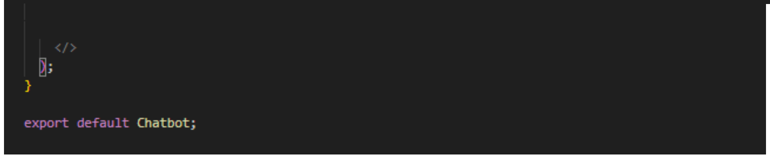
```
    recognitionRef.current = recognition;
    recognition.start();
  };

  const sendMessage = async () => {
    if (input.trim()) {
      const feeRegex = /fee:\s*(\d+\s*-\s*\d+)/i;
      const feeMatch = input.match(feeRegex);
      const extractedFeeRange = feeMatch ? feeMatch[1] : "";
      const cleanedQuery = input.replace(feeRegex, "").trim();

      const userMessage = { text: input, sender: "user" };
      setMessages((prevMessages) => [...prevMessages, userMessage]);

      try {
        const response = await fetch("http://127.0.0.1:5000/chatbot", {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ query: cleanedQuery, feeRange: extractedFeeRange })
        });

        const data = await response.json();
        const hospitals = Array.isArray(data.response) ? data.response : [];

        setMessages((prevMessages) => [
          ...prevMessages,
          hospitals.length > 0
            ? { hospitals, sender: "bot" }
            : { text: "Sorry, no matching hospitals found.", sender: "bot" }
        ]);
      } catch (error) {
```



```
  const sendMessage = async () => {
        setMessages((prevMessages) => [...prevMessages, { text: "Error fetching response.", sender: "bot" }]);
      }
      setInput("");
    }
  };

  useEffect(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: "smooth" });
  }, [messages]);

  return (
    <>
      <Header />
      <div className="chat-container">
        <div className="chat-messages">
          {messages.map((msg, index) => {
            if (msg.sender === "bot" && msg.hospitals) {
              return (
                <div key={index} className="hospital-list">
                  {msg.hospitals.map((hospital, idx) => (
                    <div key={idx} className="hospital-card">
                      <h3>{hospital["Name"]}</h3>
                      <p><strong>City:</strong> {hospital["City"]}</p>
                      <p><strong>Province:</strong> {hospital["Province"]}</p>
                      <p><strong>Specialization:</strong> {hospital["Specialization"]}</p>
                      <p><strong>Phone:</strong> {hospital["Phone"]}</p>
                      <p><strong>Contact Person:</strong> {hospital["ContactPerson"]}</p>
                      <p><strong>Address:</strong> {hospital["Address"]}</p>
                      <p><strong>Fees:</strong> {hospital["Fees"]}</p>
                      <p><strong>Website:</strong>
                        {hospital["Website"] ? <a href={hospital["Website"]} target="_blank" rel="noopener noreferrer"> Visit</a> : "Not Available"}
                      </p>
```



```
                    </div>
                  ))}
                </div>
              );
            } else {
              return (
                <div key={index} className={msg.sender === "bot" ? "bot-message" : "user-message"}>
                  {msg.text}
                </div>
              );
            }
          })}
          <div ref={messagesEndRef} />
        </div>
      </div>
      <div id="chatInputContainer">
  <input
    id="chatInput"
    type="text"
    placeholder='Ask about hospitals... (e.g., "Cancer hospitals in Lahore fee: 50000-60000")'
    value={input}
    onChange={(e) => setInput(e.target.value)}
    ~~onKeyPress~~={(e) => e.key === "Enter" && sendMessage()}
  />
  <button id="micButton" onClick={startListening}>
    <i className="fa fa-microphone text-xl"></i>
  </button>
  <button id="sendButton" onClick={sendMessage}>
    <i className="fa fa-paper-plane text-xl"></i>
  </button>
</div>
```



```


    </>
  );
}

export default Chatbot;
```

- **Web Scraping Code Explanation:**

❖ **BeautifulSoup.py:**

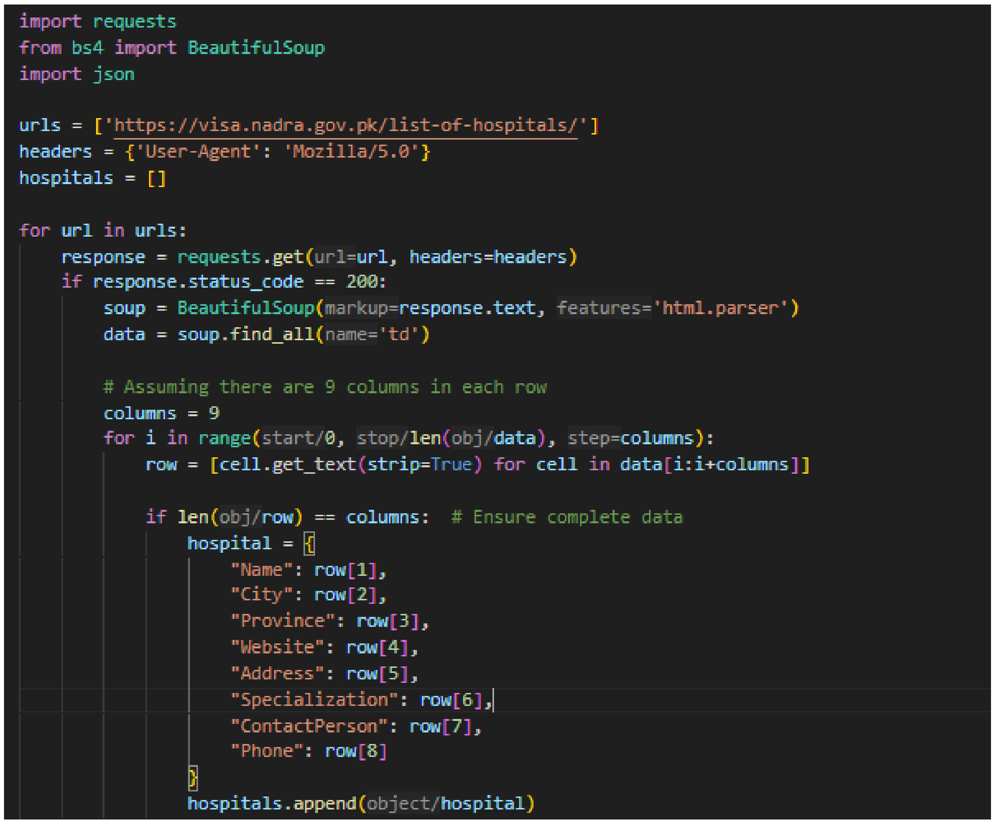
Purpose

- Extract hospital information from NADRA's public website and store it in a structured JSON file.

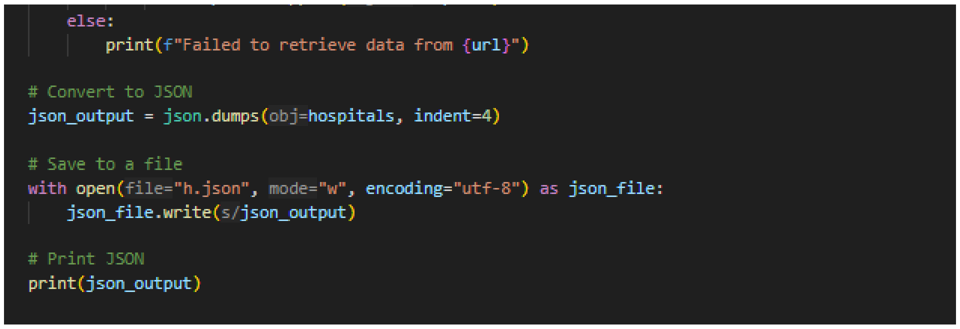
Process Flow

- Define URL and headers: Prevents being blocked by using a browser-like user agent.
- Send Request & Parse HTML: Scrapes all <td> elements (table data) from the webpage.
- Chunk Table Data: Breaks the flat list of table data into meaningful rows.
- Build Hospital Dictionary: Assigns each cell value to a specific key.
- Save as JSON File: Saves the entire list of hospitals in readable JSON format.

Source Code:

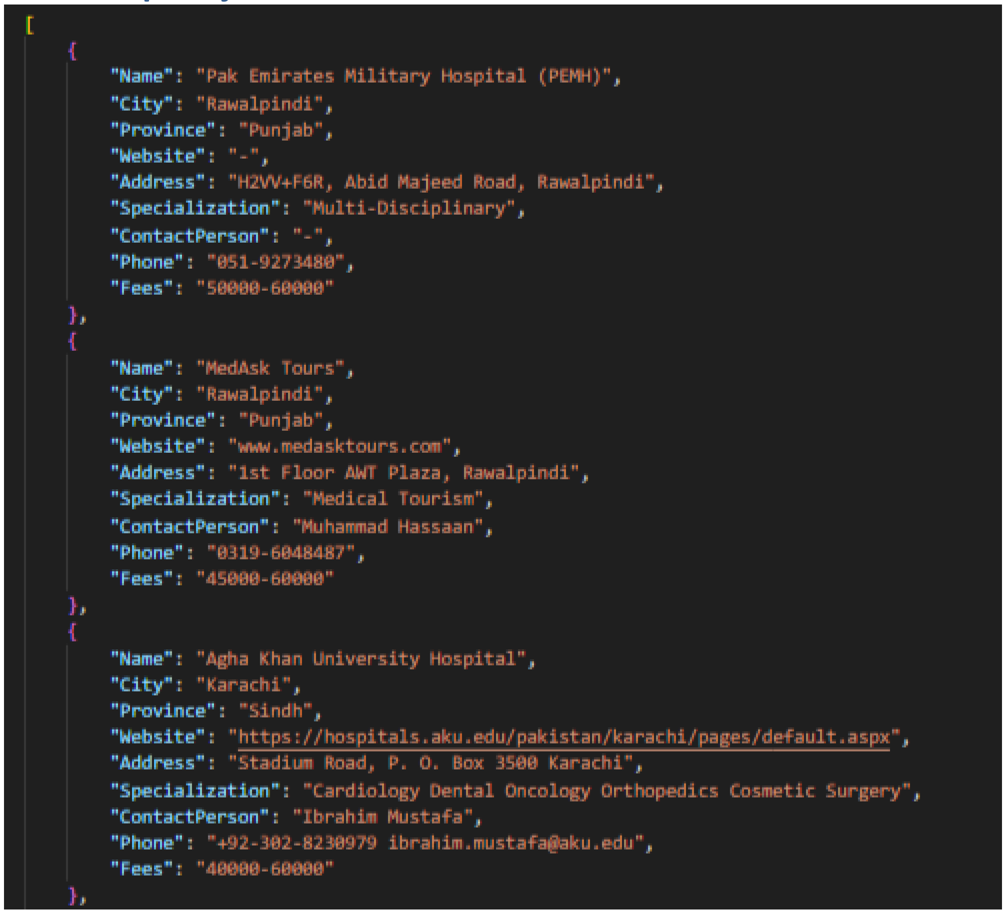


```
import requests
from bs4 import BeautifulSoup
import json

urls = ['https://visa.nadra.gov.pk/list-of-hospitals/']
headers = {'User-Agent': 'Mozilla/5.0'}
hospitals = []

for url in urls:
    response = requests.get(url=url, headers=headers)
    if response.status_code == 200:
        soup = BeautifulSoup(markup=response.text, features='html.parser')
        data = soup.find_all(name='td')

        # Assuming there are 9 columns in each row
        columns = 9
        for i in range(start/0, stop/len(obj/data), step=columns):
            row = [cell.get_text(strip=True) for cell in data[i:i+columns]]

            if len(obj/row) == columns:  # Ensure complete data
                hospital = {
                    "Name": row[1],
                    "City": row[2],
                    "Province": row[3],
                    "Website": row[4],
                    "Address": row[5],
                    "Specialization": row[6],
                    "ContactPerson": row[7],
                    "Phone": row[8]
                }
                hospitals.append(object/hospital)
```



```
    else:
        print(f"Failed to retrieve data from {url}")

# Convert to JSON
json_output = json.dumps(obj=hospitals, indent=4)

# Save to a file
with open(file="h.json", mode="w", encoding="utf-8") as json_file:
    json_file.write(s/json_output)

# Print JSON
print(json_output)
```

Hospital.json File:



```
[
    {
        "Name": "Pak Emirates Military Hospital (PEMH)",
        "City": "Rawalpindi",
        "Province": "Punjab",
        "Website": "-",
        "Address": "H2VV+F6R, Abid Majeed Road, Rawalpindi",
        "Specialization": "Multi-Disciplinary",
        "ContactPerson": "-",
        "Phone": "051-9273480",
        "Fees": "50000-60000"
    },
    {
        "Name": "MedAsk Tours",
        "City": "Rawalpindi",
        "Province": "Punjab",
        "Website": "www.medasktours.com",
        "Address": "1st Floor AWT Plaza, Rawalpindi",
        "Specialization": "Medical Tourism",
        "ContactPerson": "Muhammad Hassaan",
        "Phone": "0319-6048487",
        "Fees": "45000-60000"
    },
    {
        "Name": "Agha Khan University Hospital",
        "City": "Karachi",
        "Province": "Sindh",
        "Website": "https://hospitals.aku.edu/pakistan/karachi/pages/default.aspx",
        "Address": "Stadium Road, P. O. Box 3500 Karachi",
        "Specialization": "Cardiology Dental Oncology Orthopedics Cosmetic Surgery",
        "ContactPerson": "Ibrahim Mustafa",
        "Phone": "+92-302-8230979 ibrahim.mustafa@aku.edu",
        "Fees": "40000-60000"
    },
```

5. **Justification** - Why it is a Complex Computing Problem:

The complexity of this problem arises due to multiple interdependent factors:
Natural Language Processing (NLP): Understanding user queries with varying sentence structures and medical terminologies.

- **Real-time Decision Making**: Processing multiple attributes such as fees, location, disease specialization, and timings simultaneously.
- **Data Integration**: Combining structured hospital databases with real-time query processing.
- **Scalability:** Handling large amounts of hospital data and dynamic recommendations based on user inputs.
- **Geospatial Analysis:** Computing distance-based recommendations and mapping hospitals efficiently.

6. **Industrialization Aspect:**

The project has strong industrial potential and can be transformed into a commercial product. It can be integrated into hospital management systems, healthcare applications, and government health portals. Future enhancements may include AI-powered voice assistance, integration with hospital appointment booking systems, and mobile app development, making it a viable solution for large-scale deployment in the healthcare industry.