

AI Mock Interview Tool

Project Deliverable #4:

Team 3:

Team member	Deliverables
Rohit	Database Selection and Integration for storing user data
Dilpreet kaur	Filler words detection
Chanthuru Thavarajah	Full app Integration into one cohesive system.
Arunkumar Anugu	Real-time video background optimization
Vasu Bejugam	Live response dialogue generation & Real-time video background check
Arshdeep Kaur	Analyzes the user's spoken English for grammatical errors
Dhvani Bhavani	Live video Face expressions
Varunan Gurushev	External audio background check
Jaskaran Singh	Eye contact analysis

Rohit:

I am working on database integration for the AI Mock Interview Tool to efficiently store and manage interview questions and user data. I chose SQLAlchemy with SQLite for simplicity, ease of use, and no additional cost.

A screenshot of a Windows desktop environment. At the top, there's a taskbar with several open applications: 'AI Mock Interview Tool', '(9) WhatsApp', 'Project Deliverable #4 - Google Doc', 'Home', and 'Database'. Below the taskbar is a browser window showing the URL 'localhost:8890/notebooks/OneDrive/Documents/Rohit%20Mock%20Interview%20Project/Database.ipynb?'. The main content area is a Jupyter Notebook interface. A code cell in the notebook contains the following Python code:

```
[5]: from sqlalchemy import create_engine, inspect
from prettytable import PrettyTable

# Replace 'sqlite:///your_database.db' with your actual database URI
engine = create_engine('sqlite:///interview_tool.db')
inspector = inspect(engine)

# Function to create a PrettyTable for a given table
def display_table_structure(table_name):
    table = PrettyTable()
    table.field_names = ["Column Name", "Type", "Nullable"]

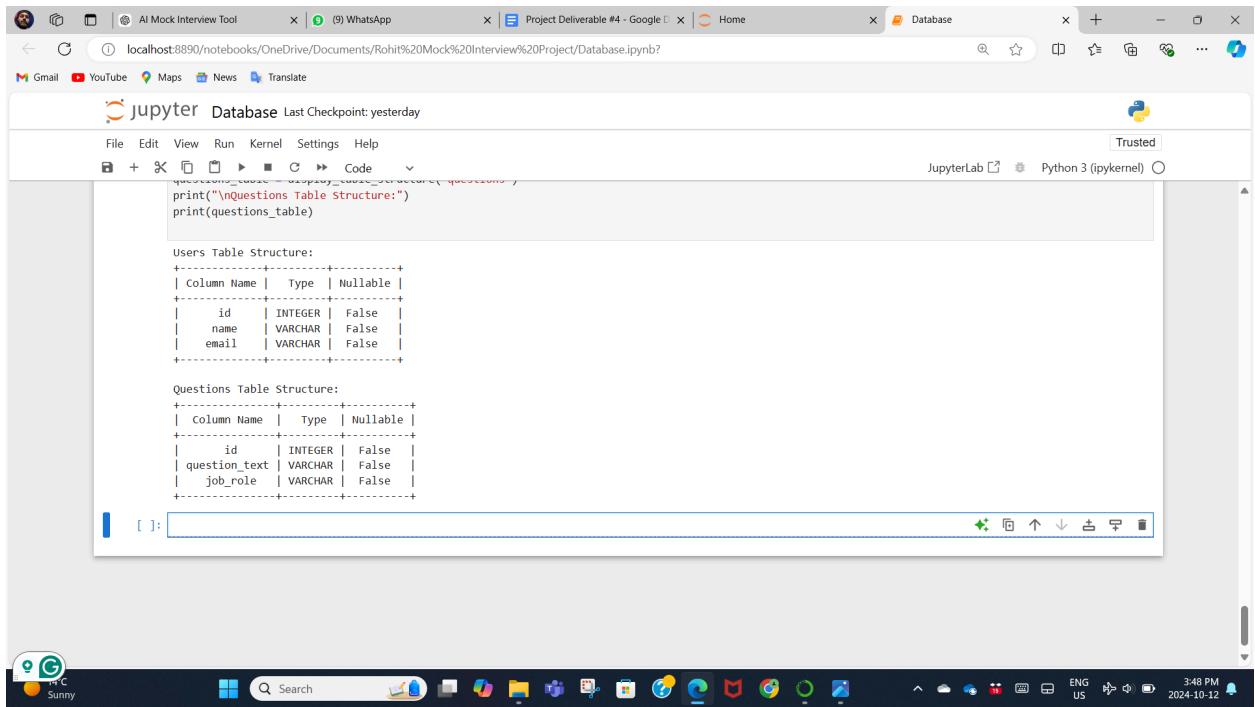
    # Fetch columns for the specified table
    columns = inspector.get_columns(table_name)
    for column in columns:
        table.add_row([column['name'], column['type'], column['nullable']])

    return table

# Display the structure for the 'users' table
users_table = display_table_structure('users')
print("Users Table Structure:")
print(users_table)

# Display the structure for the 'questions' table
questions_table = display_table_structure('questions')
print("\nQuestions Table Structure:")
print(questions_table)
```

The system tray at the bottom of the screen displays a weather icon (Sunny), the date '2024-10-12', and the time '3:48 PM'.



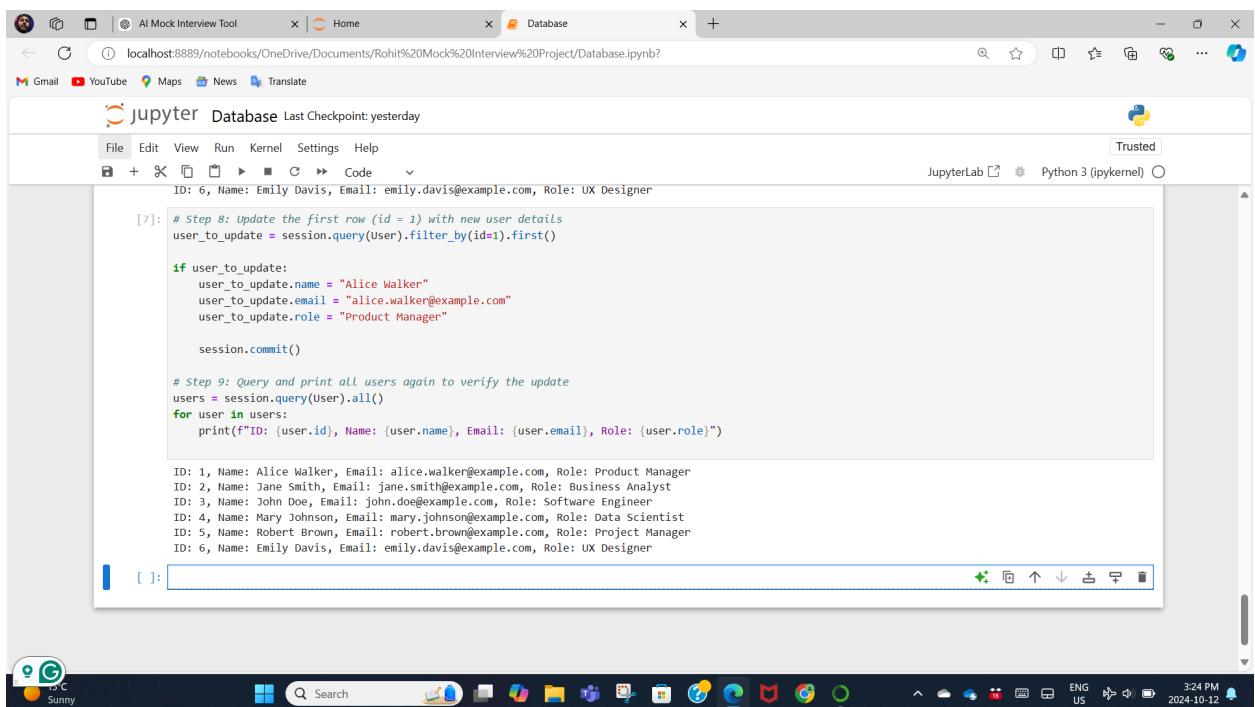
```
print("\nUsers Table Structure:")
print(users_table)

Users Table Structure:
+-----+-----+-----+
| column_name | type | nullable |
+-----+-----+-----+
| id | INTEGER | False |
| name | VARCHAR | False |
| email | VARCHAR | False |
+-----+-----+-----+

print("\nQuestions Table Structure:")
print(questions_table)

Questions Table Structure:
+-----+-----+-----+
| column_name | type | nullable |
+-----+-----+-----+
| id | INTEGER | False |
| question_text | VARCHAR | False |
| job_role | VARCHAR | False |
+-----+-----+-----+
```

I created a User class to manage user information, including name, email, and role. This class facilitates seamless integration and storage of user data within the database.



```
ID: 6, Name: Emily Davis, Email: emily.davis@example.com, Role: UX Designer

[7]: # Step 8: Update the first row (id = 1) with new user details
user_to_update = session.query(User).filter_by(id=1).first()

if user_to_update:
    user_to_update.name = "Alice Walker"
    user_to_update.email = "alice.walker@example.com"
    user_to_update.role = "Product Manager"

session.commit()

# Step 9: Query and print all users again to verify the update
users = session.query(User).all()
for user in users:
    print(f"ID: {user.id}, Name: {user.name}, Email: {user.email}, Role: {user.role}")

ID: 1, Name: Alice Walker, Email: alice.walker@example.com, Role: Product Manager
ID: 2, Name: Jane Smith, Email: jane.smith@example.com, Role: Business Analyst
ID: 3, Name: John Doe, Email: john.doe@example.com, Role: Software Engineer
ID: 4, Name: Mary Johnson, Email: mary.johnson@example.com, Role: Data Scientist
ID: 5, Name: Robert Brown, Email: robert.brown@example.com, Role: Project Manager
ID: 6, Name: Emily Davis, Email: emily.davis@example.com, Role: UX Designer
```

I created a question class that connects to the CSV file to fetch interview questions based on different job roles.

The screenshot shows two stacked Jupyter Notebook windows. Both windows have the title 'Untitled' and were last checked at 'yesterday'. The top window displays the initial steps of the code, including importing pandas and sqlalchemy, setting up a SQLite database, defining a Question model, creating database tables, and setting up a session. The bottom window continues the code, adding steps to load questions from a CSV file ('business.csv'), retrieve questions based on selected job roles, and print example usage. The output pane of the bottom window shows 15 questions related to business, such as 'What is a business model?' and 'What do you know about the term "market research?"'. The system tray at the bottom indicates it's 3:10 PM on October 12, 2024, with a sunny weather icon and 15°C.

```
[17]: # Step 1: Import the necessary libraries
import pandas as pd
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker, declarative_base # Updated import for declarative_base
from sqlalchemy.exc import IntegrityError

# Step 2: Set up the SQLite database and SQLAlchemy
DATABASE_URL = "sqlite:///mock_interview_tool.db" # Database URL for SQLite
engine = create_engine(DATABASE_URL)
Base = declarative_base() # No change needed here, as it is imported correctly

# Step 3: Define the Question model
class Question(Base):
    __tablename__ = 'questions'

    id = Column(Integer, primary_key=True)
    question_text = Column(String, nullable=False)
    job_role = Column(String, nullable=False)

# Step 4: Create the database tables
Base.metadata.create_all(engine)

# Step 5: Set up a session
Session = sessionmaker(bind=engine)
session = Session()

# Step 6: Function to load questions from CSV into the database
def load_questions_to_db(csv_file):
    df = pd.read_csv(csv_file)

# Step 7: Load questions from the CSV file
load_questions_to_db('data/business.csv') # Ensure this path is correct

# Step 8: Function to retrieve questions based on selected job role
def load_questions(selected_role):
    return session.query(Question).filter(Question.job_role == selected_role).all()

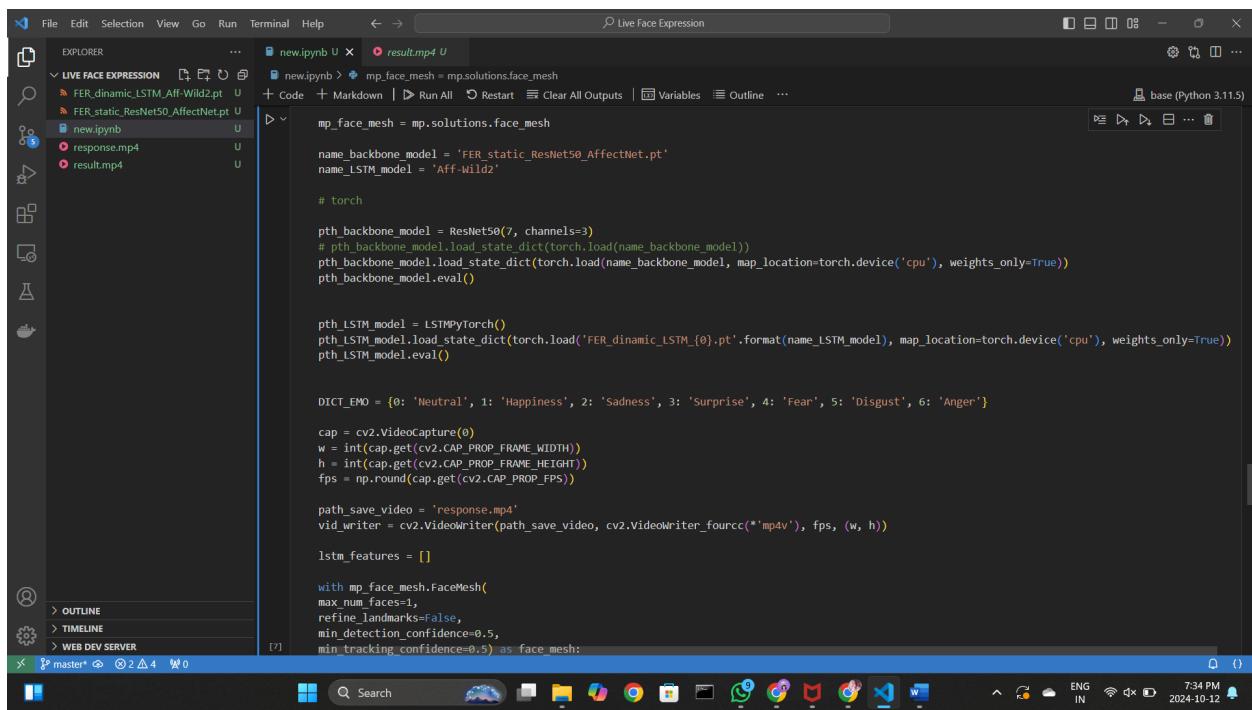
# Example usage
questions = load_questions('Business') # Replace 'Business' with the desired job role
for question in questions:
    print(f'ID: {question.id}, Question: {question.question_text}, Role: {question.job_role}')

# Step 9: Close the session
session.close()
```

Dhvani Bhavani: Live video Face expressions

Emo-AffectNet model is designed for robust and real-time facial emotion recognition, whether from single images or live videos, making it versatile for various AI-driven emotional analysis applications.

Emotions can be '**Neutral**', '**Happiness**', '**Sadness**', '**Surprise**', '**Fear**', '**Disgust**', '**Anger**'



The screenshot shows a Jupyter Notebook interface titled "Live Face Expression". The left sidebar displays the file structure of the "LIVE FACE EXPRESSION" folder, which includes "new.ipynb", "result.mp4", and several model files like "FER_dynamic_LSTM_Aff-Wild2.pt", "FER_static_ResNet50_AffectNet.pt", and "response.mp4". The main notebook cell contains the following Python code:

```
mp_face_mesh = mp.solutions.face_mesh

name_backbone_model = 'FER_static_ResNet50_AffectNet.pt'
name_LSTM_model = 'Aff-Wild2'

# torch

pth_backbone_model = ResNet50(7, channels=3)
# pth_backbone_model.load_state_dict(torch.load(name_backbone_model))
pth_backbone_model.load_state_dict(torch.load(name_backbone_model, map_location=torch.device('cpu'), weights_only=True))
pth_backbone_model.eval()

pth_LSTM_model = LSTMPTorch()
pth_LSTM_model.load_state_dict(torch.load('FER_dynamic_LSTM_{0}.pt'.format(name_LSTM_model), map_location=torch.device('cpu'), weights_only=True))
pth_LSTM_model.eval()

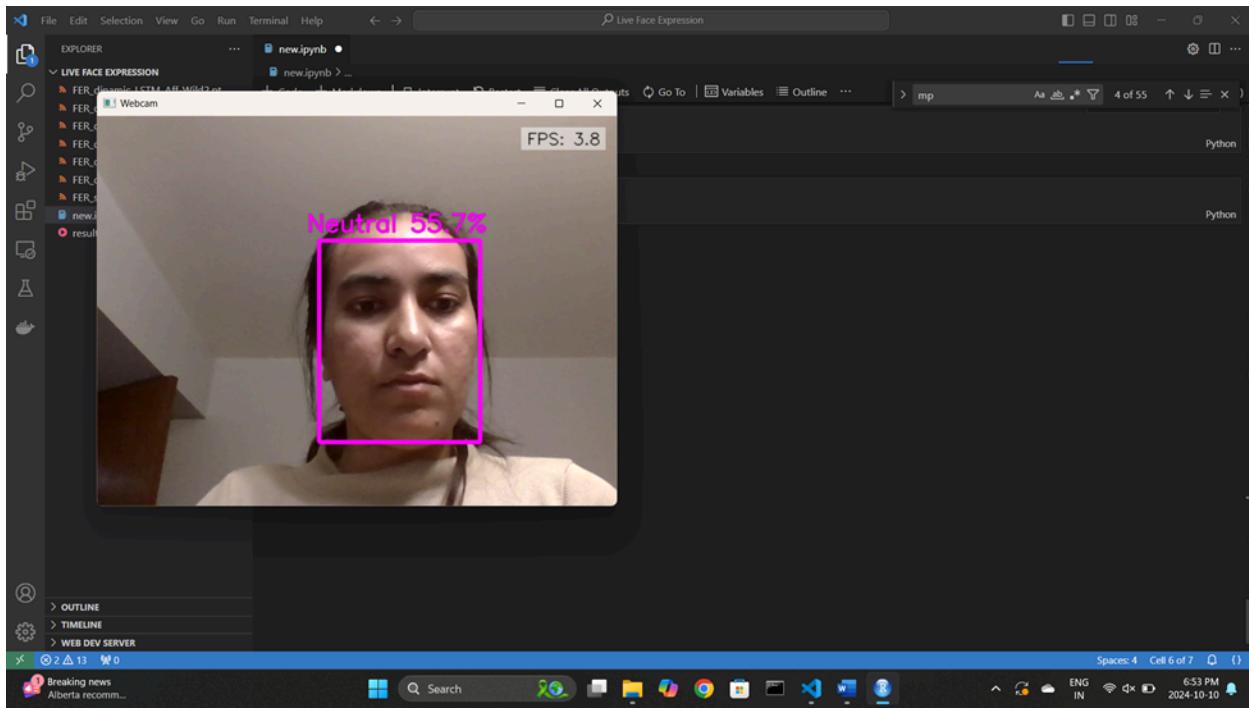
DICT_EMO = {0: 'Neutral', 1: 'Happiness', 2: 'Sadness', 3: 'Surprise', 4: 'Fear', 5: 'Disgust', 6: 'Anger'}

cap = cv2.VideoCapture(0)
w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = np.round(cap.get(cv2.CAP_PROP_FPS))

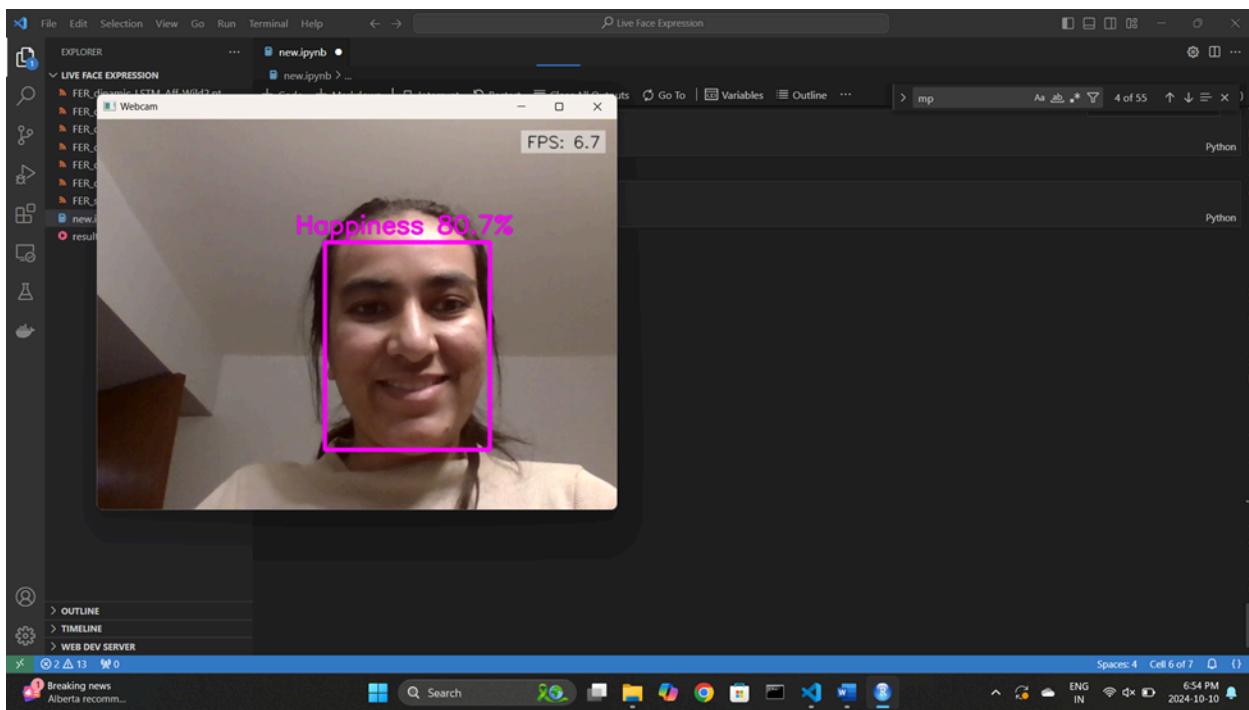
path_save_video = 'response.mp4'
vid_writer = cv2.VideoWriter(path_save_video, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))

lstm_features = []

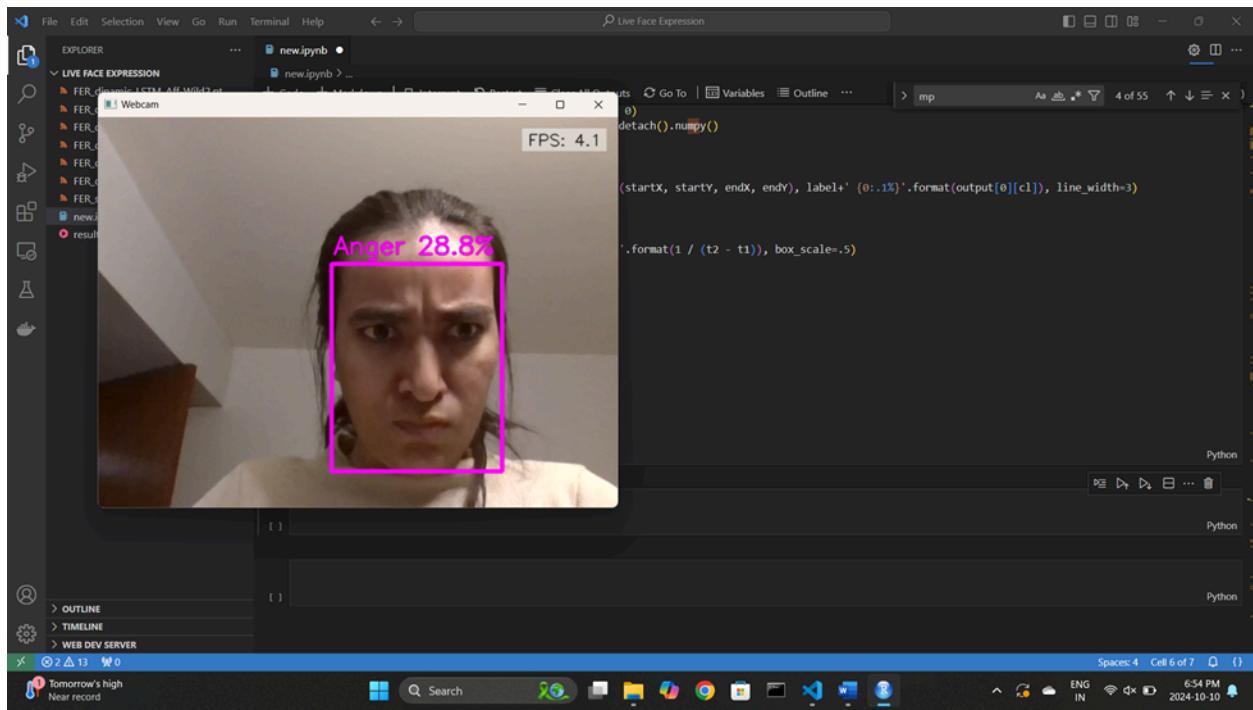
with mp_face_mesh.FaceMesh(
    max_num_faces=1,
    refine_landmarks=False,
    min_detection_confidence=0.5) as face_mesh:
```



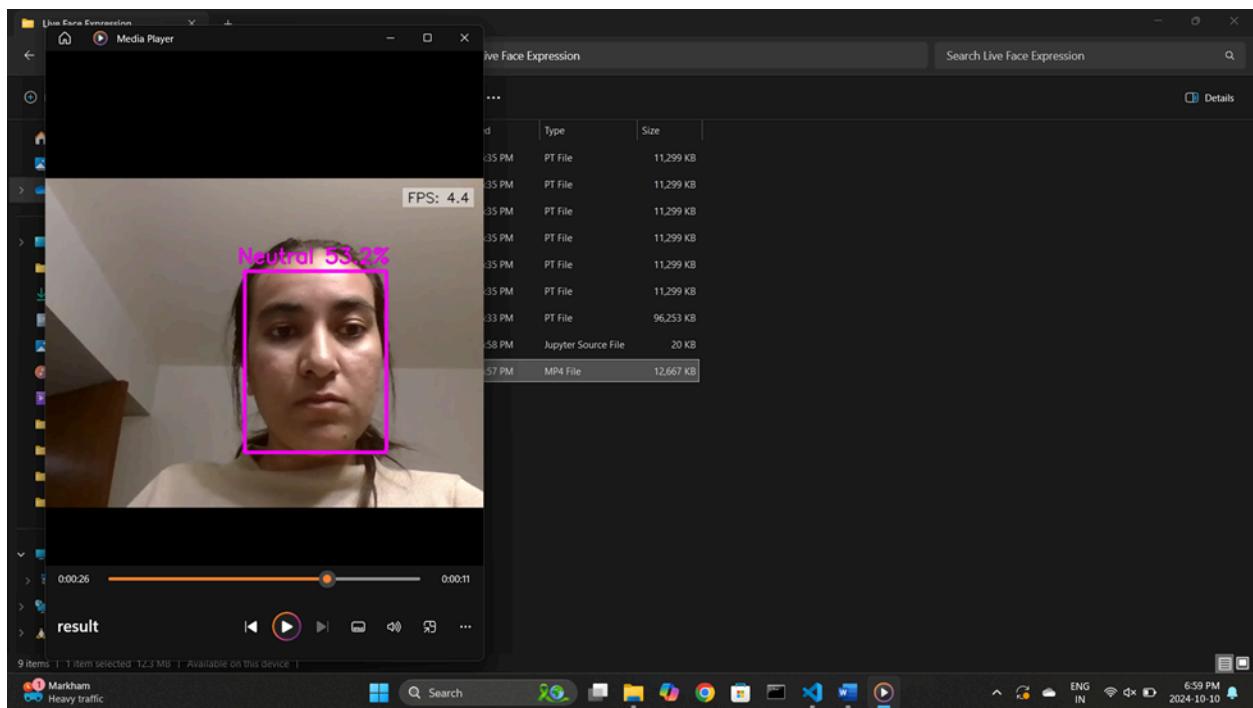
Happy:



Anger:

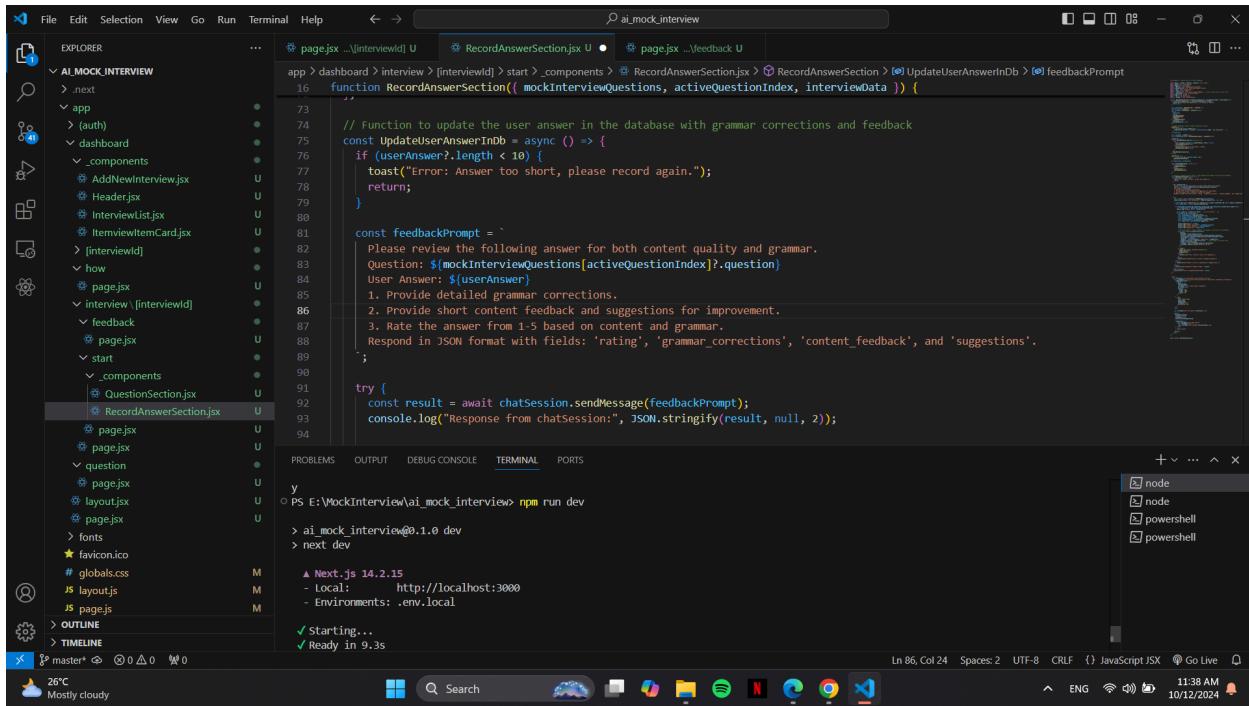


Downloaded/Saved video for response analysis.



Chanthuru Thavarajah:

Combining all the small models we built together



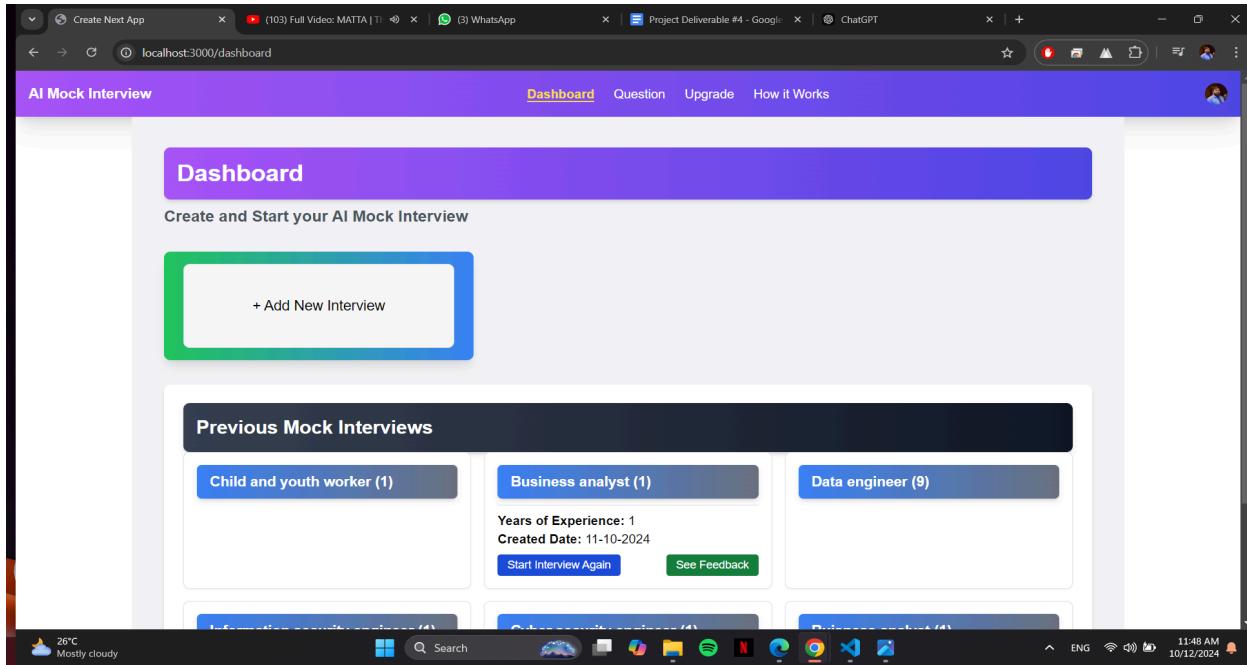
```
function RecordAnswerSection({ mockInterviewQuestions, activeQuestionIndex, interviewData }) {  
  const UpdateUserAnswerInDb = async () => {  
    if (userAnswer?.length < 10) {  
      toast("Error: Answer too short, please record again.");  
      return;  
    }  
  
    const feedbackPrompt = `  
      Please review the following answer for both content quality and grammar.  
      Question: ${mockInterviewQuestions[activeQuestionIndex].question}  
      User Answer: ${userAnswer}  
      1. Provide detailed grammar corrections.  
      2. Provide short content feedback and suggestions for improvement.  
      3. Rate the answer from 1-5 based on content and grammar.  
      Respond in JSON format with fields: 'rating', 'grammar_corrections', 'content_feedback', and 'suggestions'.  
    `;  
  
    try {  
      const result = await chatSession.sendMessage(feedbackPrompt);  
      console.log("Response from chatSession:", JSON.stringify(result, null, 2));  
    } catch (error) {  
      console.error(error);  
    }  
  };  
  
  return {  
    UpdateUserAnswerInDb,  
  };  
}  
  
export default RecordAnswerSection;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

y
PS E:\MockInterview\ai_mock_interview> npm run dev
> ai_mock_interview@0.1.0 dev
> next dev
▲ Next.js 14.2.15
- Local: http://localhost:3000
- Environments: .env.local

✓ Starting...
✓ Ready in 9.3s

I developed a user-friendly dashboard that allows users to initiate a new interview and review their past interview history. The dashboard provides the option to either retake a previous interview or view detailed feedback from completed interviews. All relevant data is efficiently retrieved and displayed from our database, ensuring a seamless experience for users managing their interview activities.



```

app > dashboard > interview > [interviewId] > start > page.jsx > ...
  1 "use client"; // Ensure this is a Client Component
  2
  3 import { useState, useEffect } from "react";
  4 import { db } from "@/utils/db"; // Ensure this path is correct
  5 import { MockInterview } from "@/utils/schema";
  6 import { eq } from "drizzle-orm";
  7 import QuestionSection from "@/components/QuestionSection";
  8 import RecordAnswerSection from "@/components/RecordAnswerSection";
  9 import { Button } from "@/components/ui/button";
 10 import Link from "next/link";
 11
 12 function StartInterview({ params }) {
 13   const { interviewId } = params; // Destructure the interviewId from URL params
 14   const [interviewData, setInterviewData] = useState(null); // State to hold interview data
 15   const [mockInterviewQuestions, setMockInterviewQuestions] = useState([]);
 16   const [error, setError] = useState(null); // State for error handling
 17   const [loading, setLoading] = useState(true); // State for loading status
 18   const [activeQuestionIndex, setActiveQuestionIndex] = useState(0);
 19
 20   useEffect(() => {
 21     GetInterviewDetails();
 22   }, []);
 23
 24   const GetInterviewDetails = async () => {
 25     try {
 26       const res = await db.query(MockInterview).findOne({
 27         where: eq(MockInterview.interviewId, interviewId),
 28       });
 29
 30       if (res) {
 31         setInterviewData(res);
 32         setLoading(false);
 33       } else {
 34         setError("Interview not found");
 35       }
 36     } catch (err) {
 37       setError("Error fetching interview details");
 38     }
 39   };
 40
 41   return (
 42     <div>
 43       <h1>Welcome to the Interview!</h1>
 44       <p>Job Title: {interviewData?.jobTitle}</p>
 45       <p>Experience: {interviewData?.experience}</p>
 46       <button onClick={handleStart}>Start Interview</button>
 47     </div>
 48   );
 49 }
 50
 51 export default StartInterview;
  
```

I have developed the **New Interview Setup** functionality in our AI-powered mock interview tool, which allows users to create a personalized interview experience.

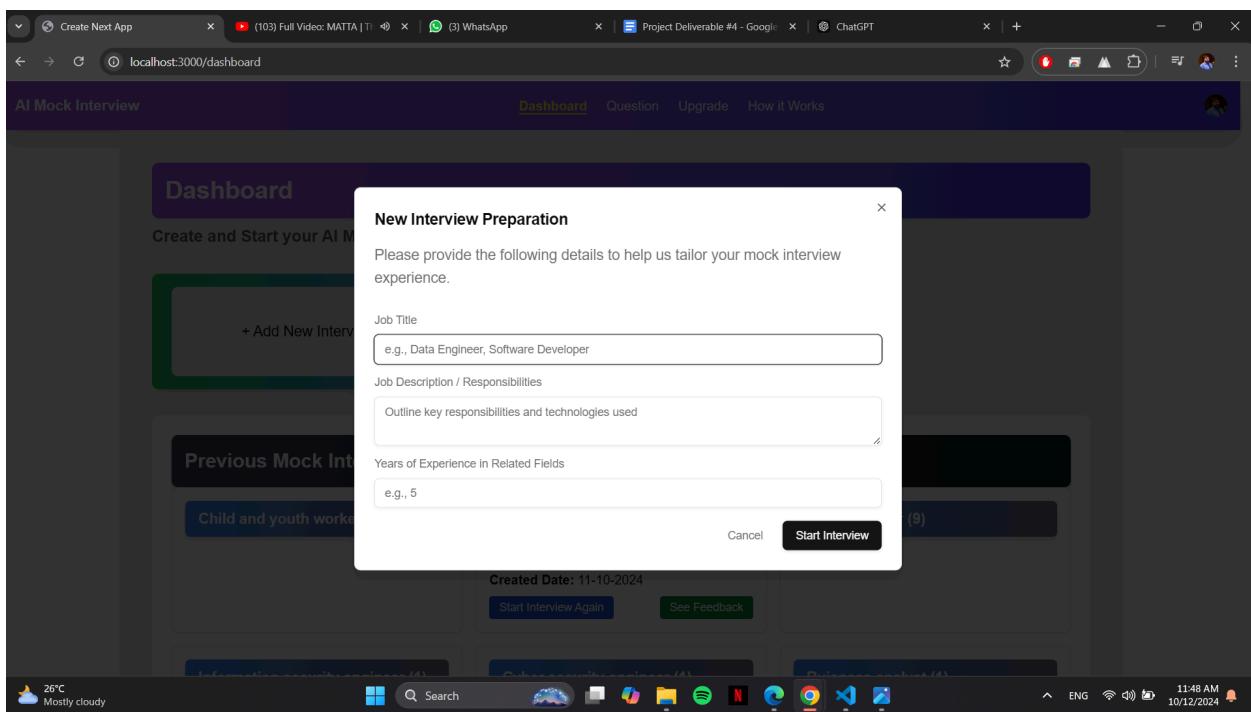
When creating a new interview, users are prompted to fill in the following fields:

- **Job Title:** Users enter a specific job title, such as "Data Engineer" or "Software Developer."

- **Job Description / Key Responsibilities:** This section allows users to provide a brief overview of the role and its key responsibilities, along with any relevant technologies they may work with.
- **Years of Experience:** Users specify how many years of experience they have in related fields.

Once the user completes these fields, they can proceed to create a tailored mock interview based on the provided information. I'll attach a screenshot of the setup interface for further clarification.

This functionality ensures that the mock interviews are highly relevant to the user's target job role and experience level.



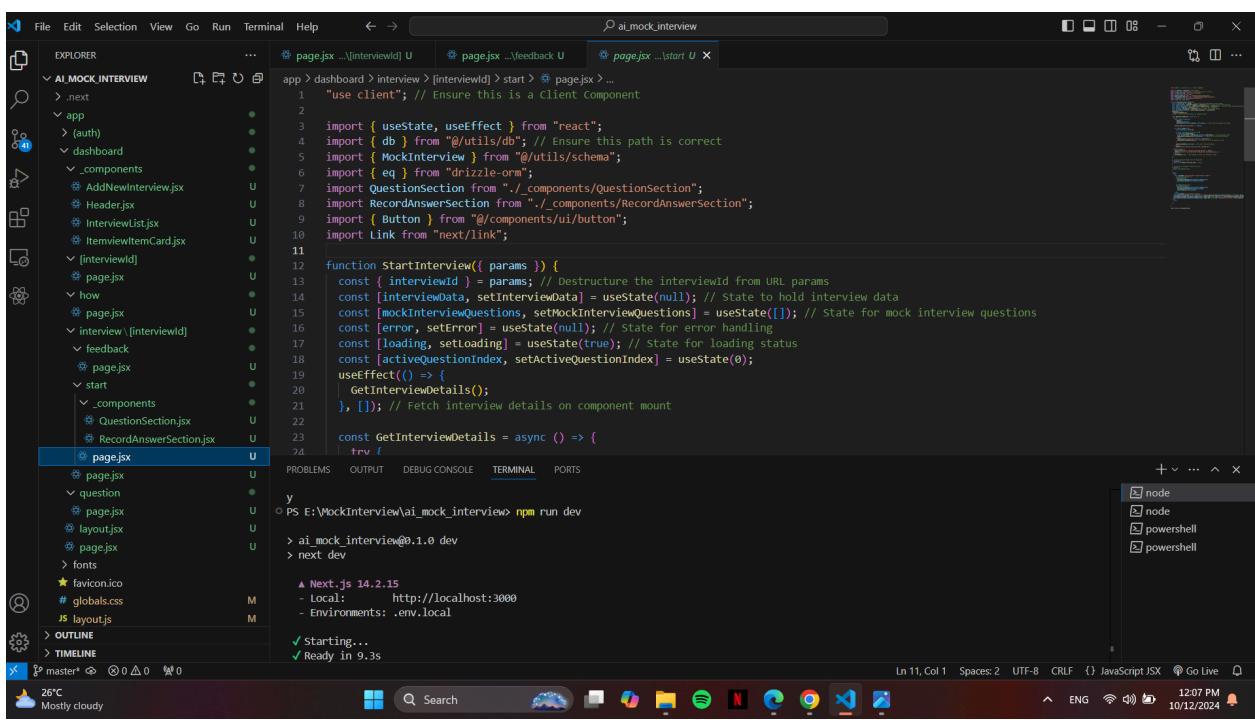
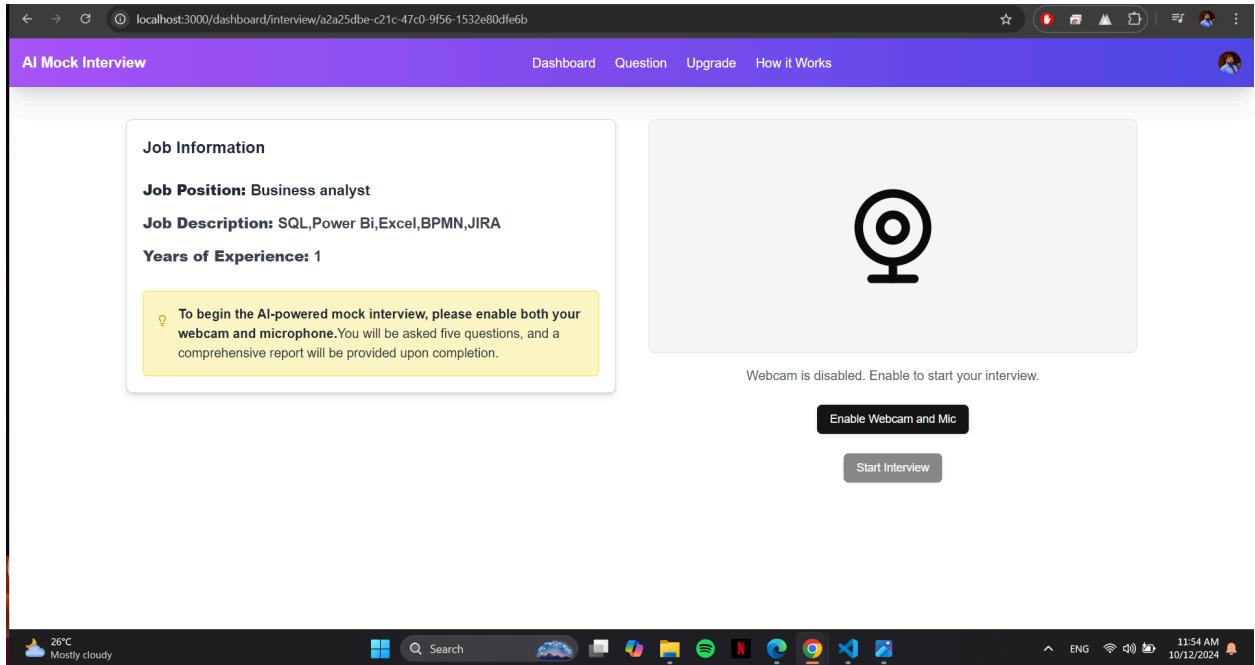
The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure for "AI MOCK INTERVIEW". The current file is "AddNewInterview.jsx" in the "app/dashboard/_components" folder.
- Code Editor:** Displays the "AddNewInterview.jsx" file. The code defines a function "AddNewInterview" that handles job position input and sends it to an AI session. It also logs the raw response from the AI.
- Terminal:** Shows the command "npm run dev" being run, resulting in the application starting at "http://localhost:3000".
- Output:** Shows the application's log output: "Starting..." and "Ready in 9.3s".
- Search Bar:** Contains the text "ai_mock_interview" and a dropdown menu with suggestions like "page.jsx", "Feedback", and "AddNewInterviewjsx".
- Bottom Status Bar:** Includes icons for weather (26°C), battery, signal, and date/time (10/12/2024).

next step in the process after the user provides the job information.

As seen in the attached screenshot, once the user has entered the job position, job description, and years of experience, the page displays a confirmation of the provided job details under "Job Information."

To proceed with the AI-powered mock interview, the user is prompted to enable both the webcam and microphone. After enabling them, they can click "Start Interview" to begin the interview process, during which they will be asked five questions. A comprehensive report based on the interview will be generated once it's completed. This ensures the user is fully prepared for the mock interview based on their specific job role.



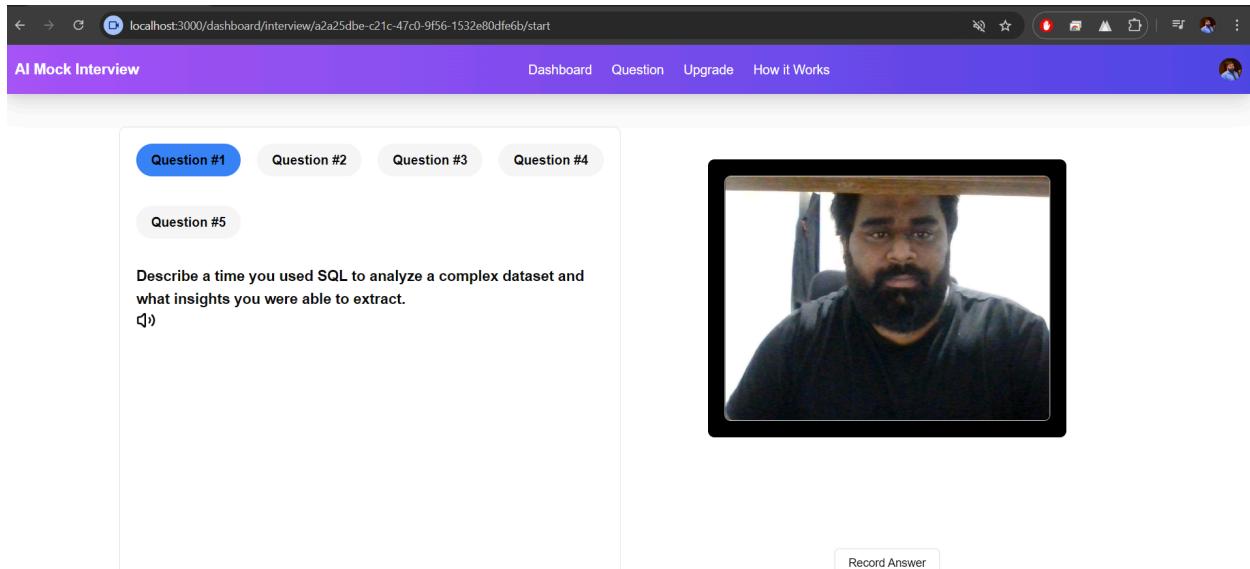
the next screen of the mock interview interface.

Once the user starts the interview, the system presents the first of five preloaded questions. In this example, the user is prompted with the question:

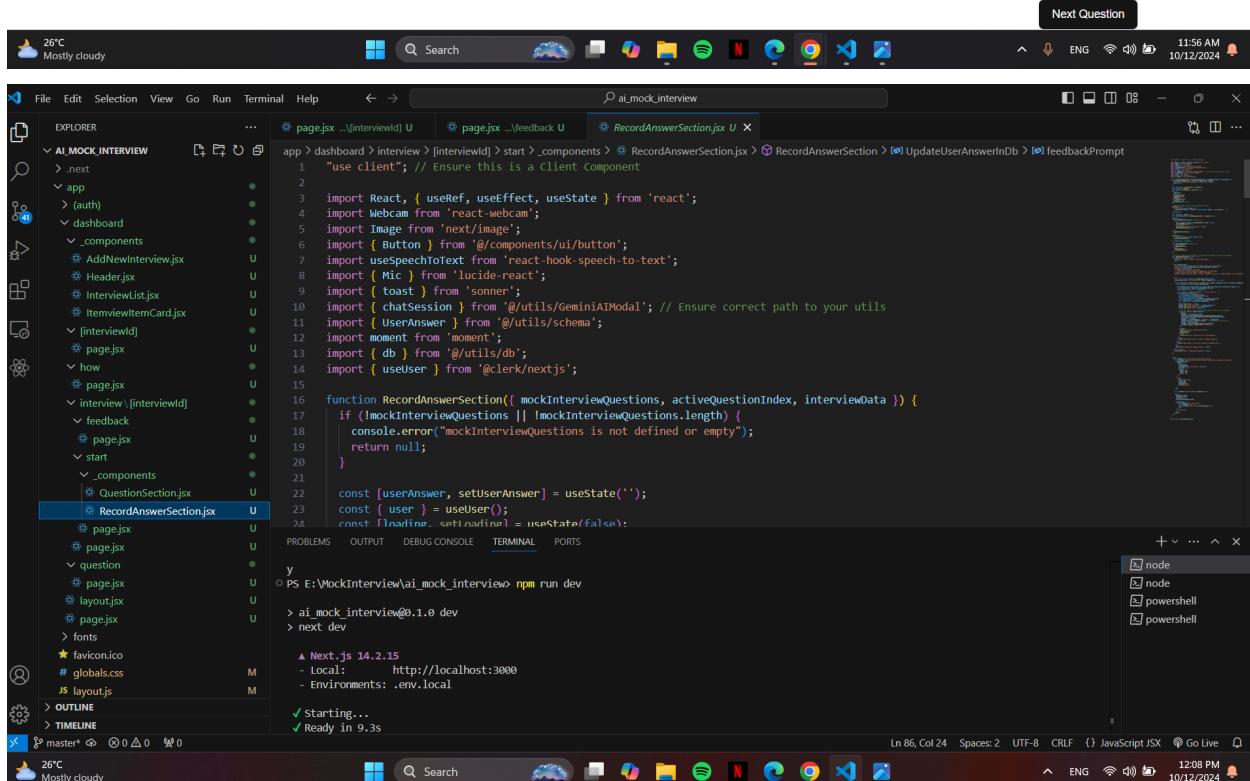
"Describe a time you used SQL to analyze a complex dataset and what insights you were able to extract."

The webcam is active, capturing the user's video response. The user has the option to record their answer by clicking on the **Record Answer** button, followed by the **Next Question** button to proceed to the subsequent question.

This interface ensures that the interview process mimics a real-life scenario, allowing users to respond to each question while being recorded, enabling feedback based on their answers and video interaction.



The screenshot shows the AI Mock Interview application interface. At the top, there is a purple header bar with the title "AI Mock Interview" and navigation links for "Dashboard", "Question", "Upgrade", and "How it Works". On the right side of the header is a user profile icon. Below the header, there is a navigation bar with tabs: "Question #1" (highlighted in blue), "Question #2", "Question #3", "Question #4", and "Question #5". The main content area contains a question: "Describe a time you used SQL to analyze a complex dataset and what insights you were able to extract." To the right of the question is a video feed window showing a person with a beard. At the bottom right of the main content area is a "Record Answer" button. The status bar at the bottom of the browser window shows the temperature as 26°C and the date/time as 10/12/2024.



The screenshot shows the VS Code code editor with the file "RecordAnswerSection.js" open. The code is a component for recording user answers. It imports React, useState, useEffect, and useRef. It uses a hook to ensure it's a client component. It imports Webcam, Image, Button, Mic, toast, and Moment. It also imports from utils/GeminiAIModal and utils/schema. The component function handles mock interview questions and updates the database with user answers. The code editor shows syntax highlighting for JavaScript and JSX. The status bar at the bottom shows the file path as "E:\MockInterview\ai_mock_interview> RecordAnswerSection.js", the terminal command as "PS E:\MockInterview\ai_mock_interview> npm run dev", and the output "Starting... Ready in 9.3s". The bottom status bar also shows the date/time as 10/12/2024.

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The code editor displays a file named `QuestionSection.jsx` with the following content:

```

1 import { Volume2 } from 'lucide-react';
2 import React, { useEffect } from 'react';
3
4 function QuestionSection({ mockInterviewQuestions, activeQuestionIndex }) {
5   useEffect(() => {
6     console.log(mockInterviewQuestions); // check if the array is received and correctly formatted
7   }, [mockInterviewQuestions]);
8   const textToSpeech=(text)>{
9     if('speechSynthesis' in window){
10       const speech=new SpeechSynthesisUtterance(text);
11       window.speechSynthesis.speak(speech)
12     }
13   else{
14     alert('Sorry your browser does not support text to speech')
15   }
16 }
17 return (
18   mockInterviewQuestions && (
19     <div className='p-5 border rounded-lg my-10'>
20       <div className='grid grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-5'>
21         {mockInterviewQuestions.length > 0 ? (
22           mockInterviewQuestions.map((questionObj, index) => (
23             <div key={index} className="mh-4">
24
25
26
27
28
29
2

```

The terminal below shows the command `npm run dev` being executed, indicating the application is running locally.

After the user completes all interview questions, the system provides feedback on their performance. In the screenshot, the overall interview rating is **1.4/5**, indicating areas for improvement.

The feedback section displays each question from the mock interview along with the user's recorded answers and specific feedback for improvement, though the answers are currently not shown in the image. This allows users to review their responses to each question, such as:

- **Question 1:** "Describe a time you used SQL to analyze a complex dataset and what insights you were able to extract."
- **Question 2:** "Explain the concept of BPMN and how you would use it to model a business process."
- **Question 3:** "Describe your experience using Jira and how you have used it to manage projects and track progress."
- **Question 4:** "You are given a large Excel spreadsheet with data from multiple departments. How would you clean and prepare this data for analysis?"

This structured feedback helps users understand their strengths and areas for improvement in specific topics related to the job they are interviewing for, enhancing their preparation for real interviews.

Arunkumar Anugu :

The code uses OpenCV and MediaPipe to access the webcam and capture live video. MediaPipe's SelfieSegmentation creates a mask to separate the user from the background, which is refined for smooth edges. The background is blurred using a Gaussian filter, and the foreground (user) is combined with the blurred background for optimization. The video feed displays the result in real-time until 'q' is pressed, and resources are released afterward.

In [2]:

```
pip install protobuf==3.20.3
```

Collecting protobuf==3.20.3
Using cached protobuf-3.20.3-cp39-cp39-win_amd64.whl (904 kB)
Installing collected packages: protobuf
Attempting uninstall: protobuf
Found existing installation: protobuf 4.25.5
Uninstalling protobuf-4.25.5:
Successfully uninstalled protobuf-4.25.5
Successfully installed protobuf-3.20.3
Note: you may need to restart the kernel to use updated packages.

ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
tensorflow-intel 2.11.0 requires protobuf<3.20,>=3.9.2, but you have protobuf 3.20.3 which is incompatible.
mediapipe 0.10.14 requires protobuf<5,>=4.25.3, but you have protobuf 3.20.3 which is incompatible.

```
import cv2
import mediapipe as mp
import numpy as np

# Initialize Mediapipe segmentation solution
mp_selfie_segmentation = mp.solutions.selfie_segmentation
selfie_segmentation = mp_selfie_segmentation.SelfieSegmentation(model_selection=1)

selfie_segmentation = mp_selfie_segmentation.SelfieSegmentation(model_selection=1)

# Initialize the webcam
cap = cv2.VideoCapture(0)

# Set the desired frame width and height
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Failed to read frame from webcam. Exiting...")
        break

    # Convert the image from BGR to RGB
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Get segmentation results
    results = selfie_segmentation.process(rgb_frame)

    # Create a mask from the segmentation results
    mask = results.segmentation_mask

    # Resize the mask to match the frame size
    mask = cv2.resize(mask, (frame.shape[1], frame.shape[0]))
```

Jupyter Real-time video background optimization Last Checkpoint: 12 minutes ago (autosaved)

```
results = selfie_segmentation.process(rgb_frame)

# Create a mask from the segmentation results
mask = results.segmentation_mask

# Resize the mask to match the frame size
mask = cv2.resize(mask, (frame.shape[1], frame.shape[0]))

# Convert the mask to binary format with a threshold
_, binary_mask = cv2.threshold(mask, 0.5, 1, cv2.THRESH_BINARY)
binary_mask = binary_mask.astype(np.uint8)

# Refine the mask using a dilation operation to smooth the edges
kernel = np.ones((15, 15), np.uint8)
dilated_mask = cv2.dilate(binary_mask, kernel, iterations=1)

# Invert the mask for background
inverse_mask = cv2.bitwise_not(dilated_mask)

# Apply a strong blur effect to the entire frame
blurred_background = cv2.GaussianBlur(frame, (45, 45), 0)

# Extract the foreground (user) from the original frame using the mask
foreground = cv2.bitwise_and(frame, frame, mask=dilated_mask)

# Extract the background from the blurred frame using the inverse mask
```

The screenshot shows a Jupyter Notebook interface running on a local host. The title bar indicates the notebook is titled "Real-time video background optimization.ipynb". The code cell contains the following Python script:

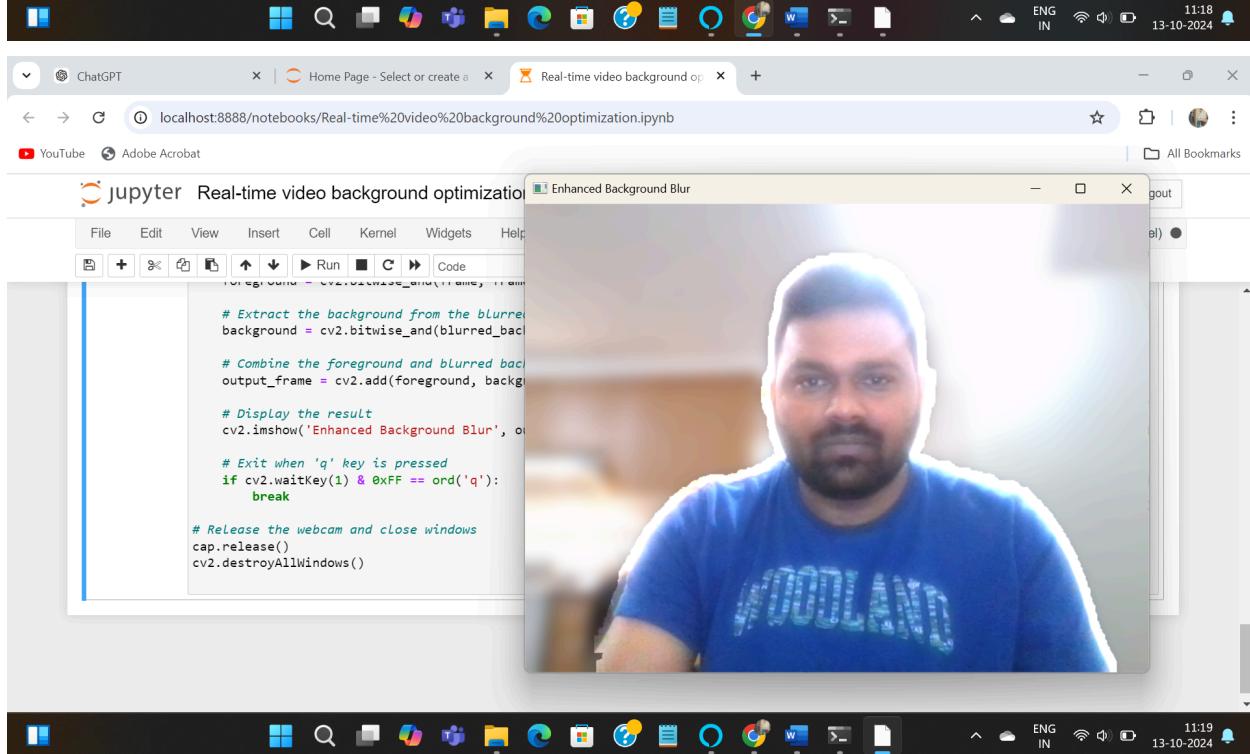
```
# Extract the background from the blurred frame using the inverse mask
background = cv2.bitwise_and(blurred_background, blurred_background, mask=inverse_mask)

# Combine the foreground and blurred background
output_frame = cv2.add(foreground, background)

# Display the result
cv2.imshow('Enhanced Background Blur', output_frame)

# Exit when 'q' key is pressed
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Release the webcam and close windows
cap.release()
cv2.destroyAllWindows()
```

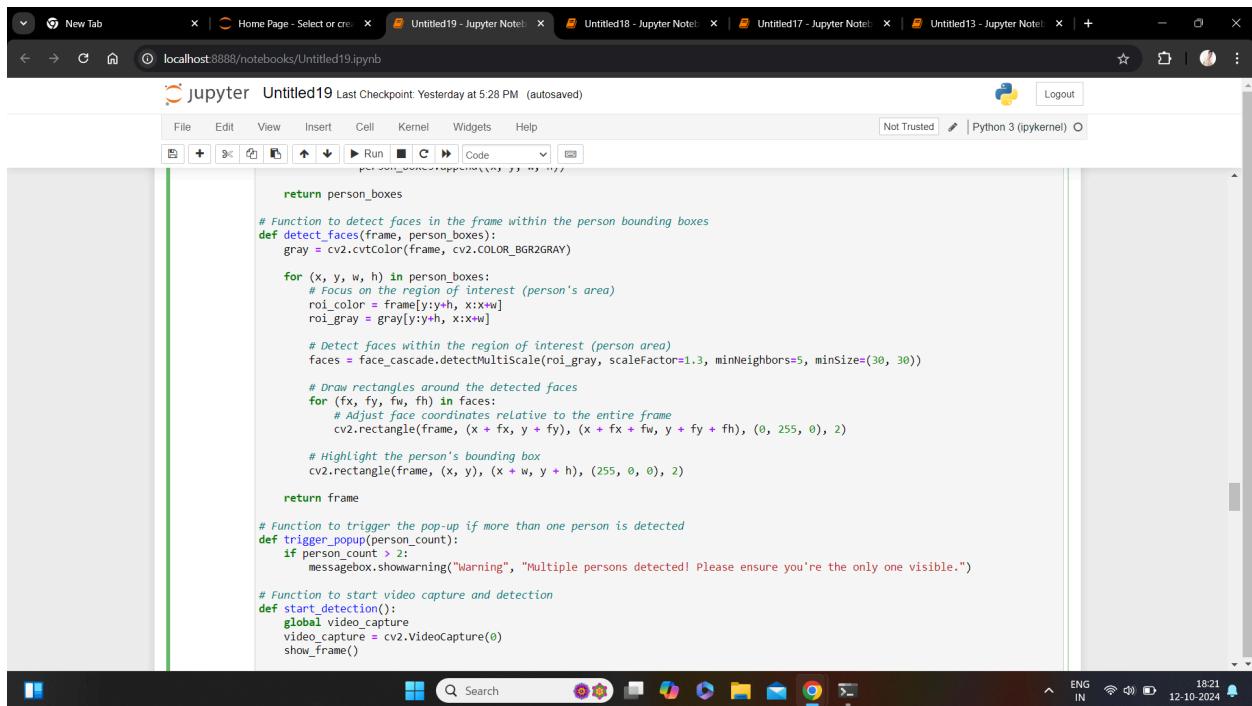


Vasu Bejugam :

The **video background feature** of our mock interview tool allows the system to:

- **Distinguish between the user and the background.**
- **Focus on the main subject** in the frame if multiple people appear.
- **Detect disturbances** in the background and prompt the user to find a space with less distraction if needed.

Here is the code to implement the user and the background detection for the mock interview tool



The screenshot shows a Jupyter Notebook interface running on a local host. The notebook has several tabs open, including 'Untitled19 - Jupyter Notebook' (the active tab), 'Untitled18 - Jupyter Notebook', 'Untitled17 - Jupyter Notebook', and 'Untitled13 - Jupyter Notebook'. The code in the notebook is as follows:

```
return person_boxes

# Function to detect faces in the frame within the person bounding boxes
def detect_faces(frame, person_boxes):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    for (x, y, w, h) in person_boxes:
        # Focus on the region of interest (person's area)
        roi_color = frame[y:y+h, x:x+w]
        roi_gray = gray[y:y+h, x:x+w]

        # Detect faces within the region of interest (person area)
        faces = face_cascade.detectMultiScale(roi_gray, scaleFactor=1.3, minNeighbors=5, minSize=(30, 30))

        # Draw rectangles around the detected faces
        for (fx, fy, fw, fh) in faces:
            # Adjust face coordinates relative to the entire frame
            cv2.rectangle(frame, (x + fx, y + fy), (x + fx + fw, y + fy + fh), (0, 255, 0), 2)

            # Highlight the person's bounding box
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)

    return frame

# Function to trigger the pop-up if more than one person is detected
def trigger_popup(person_count):
    if person_count > 2:
        messagebox.showwarning("Warning", "Multiple persons detected! Please ensure you're the only one visible.")

# Function to start video capture and detection
def start_detection():
    global video_capture
    video_capture = cv2.VideoCapture(0)
    show_frame()

# Function to start video capture and detection
def start_detection():
    global video_capture
    video_capture = cv2.VideoCapture(0)
    show_frame()
```

```
# Function to stop video capture
def stop_detection():
    global video_capture
    video_capture.release()

# Function to display the video feed and perform detection
def show_frame():
    ret, frame = video_capture.read()

    if ret:
        # Detect persons in the frame
        person_boxes = detect_persons(frame)

        # Detect and highlight faces within the person bounding boxes
        frame_with_faces = detect_faces(frame, person_boxes)
        |
        # Trigger pop-up if more than one person is detected
        trigger_popup(len(person_boxes))

        # Convert the frame for displaying in Tkinter
        frame_rgb = cv2.cvtColor(frame_with_faces, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(frame_rgb)
        imgtk = ImageTk.PhotoImage(image=img)
        video_label.imgtk = imgtk
        video_label.configure(image=imgtk)

        # Refresh the video feed every 30ms
        video_label.after(30, show_frame)

    # Create the main window
    root = tk.Tk()
    root.title("Person and Face Detection")

    # Video display Label
    video_label = Label(root)
```

```
In [2]: import cv2
import numpy as np
import tkinter as tk
from tkinter import Label, Button, messagebox
from PIL import Image, ImageTk

# Load the pre-trained YOLO model and COCO class labels
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]

# Load the pre-trained face detection model (Haar Cascade)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Function to detect persons using YOLO
def detect_persons(frame):
    height, width, channels = frame.shape

    # Preprocess the image for YOLO
    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    detections = net.forward(output_layers)

    person_boxes = []

    for output in detections:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]

            # Only consider detections with high confidence of person (class_id == 0)
            if class_id == 0 and confidence > 0.75:
                center_x = int(detection[0] * width)
```

Here is the code for live response for dialogue generation using the GPT-2 model that helps us to generate follow up questions in the interview process.

localhost:8888/notebooks/Untitled13.ipynb

jupyter Untitled13 Last Checkpoint: Last Thursday at 11:14 AM (autosaved)

In [5]:

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer
# Load your fine-tuned model and tokenizer
model = GPT2LMHeadModel.from_pretrained("./fine_tuned_gpt2_interview")
tokenizer = GPT2Tokenizer.from_pretrained("./fine_tuned_gpt2_interview")

# Set the model's padding token
tokenizer.pad_token = tokenizer.eos_token # Use EOS token as padding
```

In [6]:

```
def generate_interview_response(input_text, context=""):
    # Combine the context (previous dialogue) with the user's input
    if context:
        input_text = context + "\n" + input_text

    # Tokenize the input text
    input_ids = tokenizer.encode(input_text, return_tensors="pt")

    # Generate response
    output = model.generate(
        input_ids,
        max_length=150,           # Max length of the output
        temperature=0.7,          # Adjust randomness
        top_p=0.95,               # Nucleus sampling for more variety
        top_k=50,                 # Limit sampling to top k tokens
        do_sample=True,            # Enable sampling for creative responses
        pad_token_id=tokenizer.eos_token_id
    )

    # Decode the generated output
    generated_text = tokenizer.decode(output[0], skip_special_tokens=True)

    return generated_text
```

localhost:8888/notebooks/Untitled13.ipynb

jupyter Untitled13 Last Checkpoint: Last Thursday at 11:14 AM (autosaved)

In [6]:

```
# Tokenize the input text
input_ids = tokenizer.encode(input_text, return_tensors="pt")

# Generate response
output = model.generate(
    input_ids,
    max_length=150,           # Max length of the output
    temperature=0.7,          # Adjust randomness
    top_p=0.95,               # Nucleus sampling for more variety
    top_k=50,                 # Limit sampling to top k tokens
    do_sample=True,            # Enable sampling for creative responses
    pad_token_id=tokenizer.eos_token_id
)

# Decode the generated output
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)

return generated_text

# Example usage
user_input = "I implemented a sorting algorithm to optimize our data processing pipeline."
context = "Interviewer: Can you explain a time when you optimized a system?\nCandidate:"
response = generate_interview_response(user_input, context)
print("Interviewer: " + response)
```

Interviewer: Interviewer: Can you explain a time when you optimized a system?
Candidate:
I implemented a sorting algorithm to optimize our data processing pipeline. The algorithm was designed to keep track of every item in our file and, in order to ensure that we could optimize each item in turn, we also used algorithms to iterate over each item's attributes in a row.
In the past, we had to allocate resources to each item in our file, but since we had already allocated a large amount of memory, this was not always possible.
Candidate: In an upcoming version of the algorithm, you will be able to quickly identify and store multiple items in a single query.

The screenshot shows a Jupyter Notebook running in a web browser. The title bar indicates the URL is `localhost:8888/notebooks/Untitled13.ipynb`. The notebook interface includes a toolbar with various icons for file operations, a menu bar with File, Edit, View, Insert, Cell, Kernel, Widgets, and Help, and a status bar showing "Trusted" and "Python 3 (ipykernel)". A message bar at the top right says "jupyter Untitled13 Last Checkpoint: Last Thursday at 11:14 AM (autosaved)". Below the toolbar, there is a code cell with the following Python script:

```
import speech_recognition as sr

# Initialize recognizer
recognizer = sr.Recognizer()

# Function to capture and process speech input
def capture_speech():
    with sr.Microphone() as source:
        print("Listening for the candidate's response...")
        audio = recognizer.listen(source)

    try:
        user_input = recognizer.recognize_google(audio) # Using Google's speech recognition
        print("Candidate: " + user_input)
        return user_input
    except sr.UnknownValueError:
        print("Sorry, I could not understand the audio.")
        return ""

In [ ]:
```

Varunan Gurushev :

- Each question now includes predefined keywords; missing keywords in transcriptions are highlighted for feedback.

The screenshot shows a code editor interface with the following details:

- Explorer View:** Shows the project structure for "AI-IN-MOCK-INTERVIEW". Files include app.py, login.html, role.html, result.html, signup.html, templates, choose_role.html, dashboard.html, index.html, manage_users.html, questions.html, result.html, role.html, select_role.html, signup.html, style.css, UI_file, and various audio files (interview.wav, interview.webm, response.txt, response.wav, response.webm, video.webm).
- Code Editor:** The main pane displays the Python file `app.py`. The code defines a Flask application for handling mock interviews. It includes functions for loading predefined questions and keywords from a JSON file, and a class `MockInterview` for managing the interview process. A terminal window at the bottom shows the output of a command related to audio processing.
- Terminal:** The terminal shows the following command and its output:

```
lout@lout-OptiPlex-5090:~/Desktop/Varun$ python app.py
```

```
minor_version : 1
compatible_brands: isomiso5hsif
ISFT : Lavf56.7.100
Stream #0:0(und): Audio: pcm_s16le ([1][0][0][0] / 0x0001), 44100 Hz, mono, s16, 705 kb/s (default)
Metadata:
    creation_time : 2024-10-13T23:34:10.000000Z
    handler_name : Core Media Audio
    vendor_id : {01}0[0]0[0]
    encoder : Lavc61.19.100 pcm_s16le
lout@lout-OptiPlex-5090:~/Desktop/Varun$
```

The terminal also shows a transcription of the audio file, which includes several highlighted words in red, indicating they were present in the original audio but missing from the transcription.

```

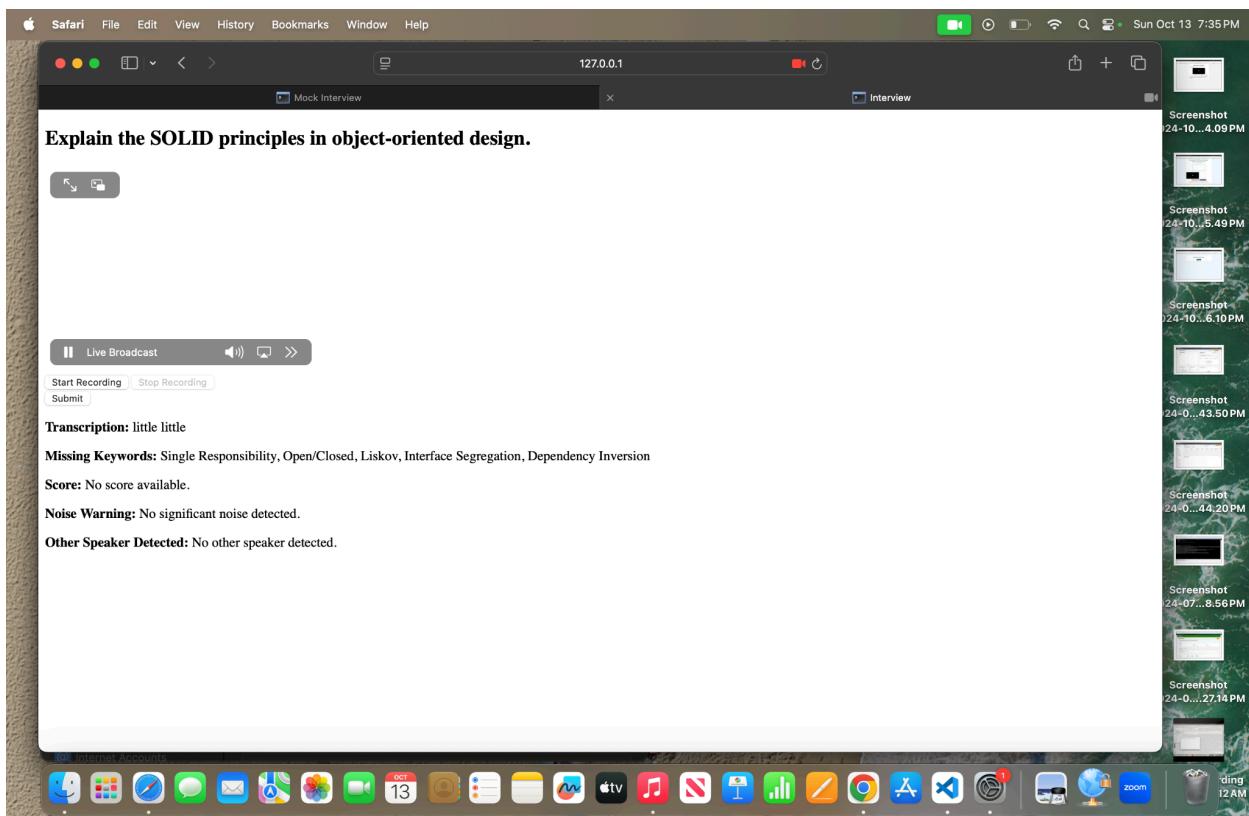
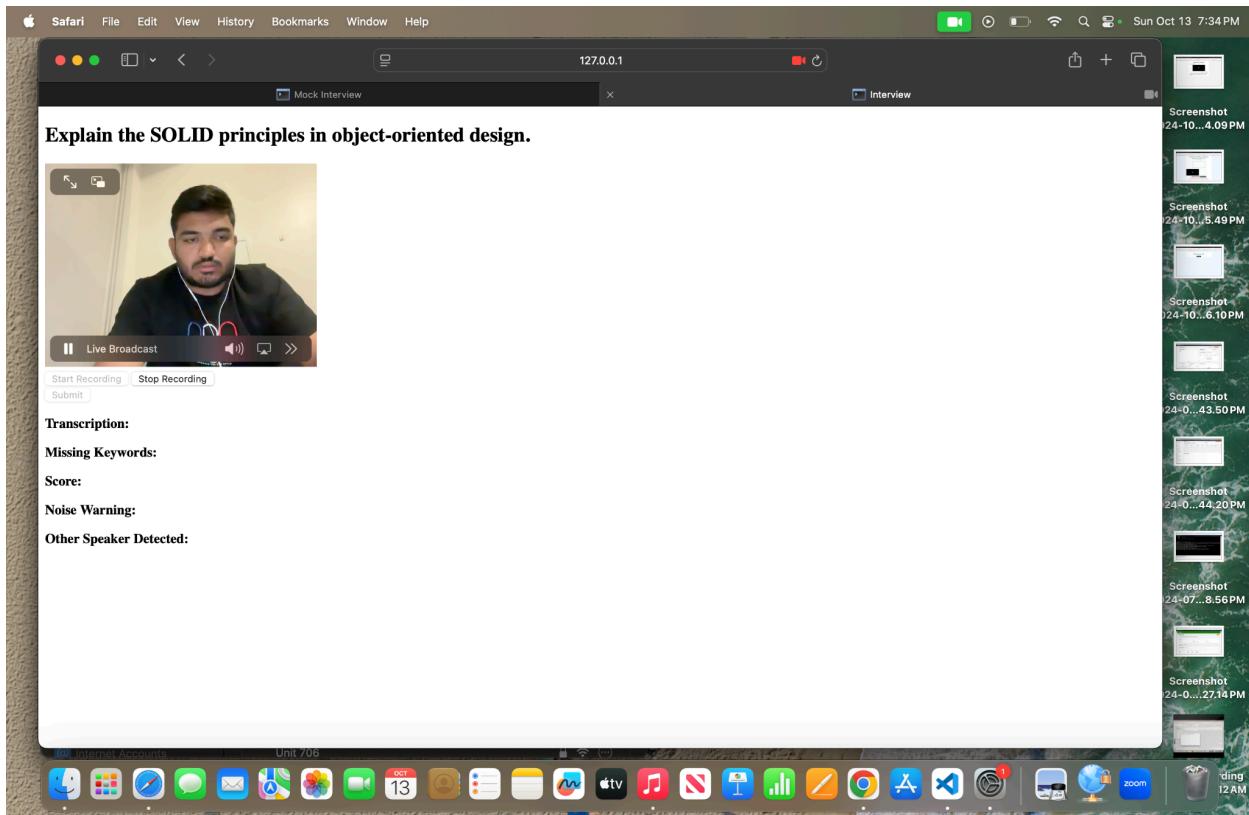
static > JS script.js ...
45  };
46
47 // Handle form submission and display results
48 uploadForm.onsubmit = async function(e) {
49   e.preventDefault();
50
51   let formData = new FormData(uploadForm);
52
53   let response = await fetch('/upload_video',
54     {
55       method: 'POST',
56       body: formData
57     });
58
59   let result = await response.json();
60
61   if (result.error) {
62     alert(result.error);
63   } else {
64     // Display transcription, noise warnings, and other analysis results
65     document.getElementById('transcribed-text').textContent = result.transcribed_text || "No transcription available.";
66     document.getElementById('missing-keywords').textContent = result.missing_keywords.join(', ') || "No missing keywords.";
67     document.getElementById('score').textContent = result.score || "No score available.";
68     document.getElementById('noise-warning').textContent = result.noise_warning ? "Too noisy" : "No significant noise detected.";
69     document.getElementById('other-speaker').textContent = result.other_speaker ? "Another speaker detected" : "No other speaker detected";
70   }
71

```

The code handles a file upload form. It reads the uploaded video file, extracts the audio stream, and performs transcription and noise analysis. The results are displayed on the page.

Audio Background Detection:

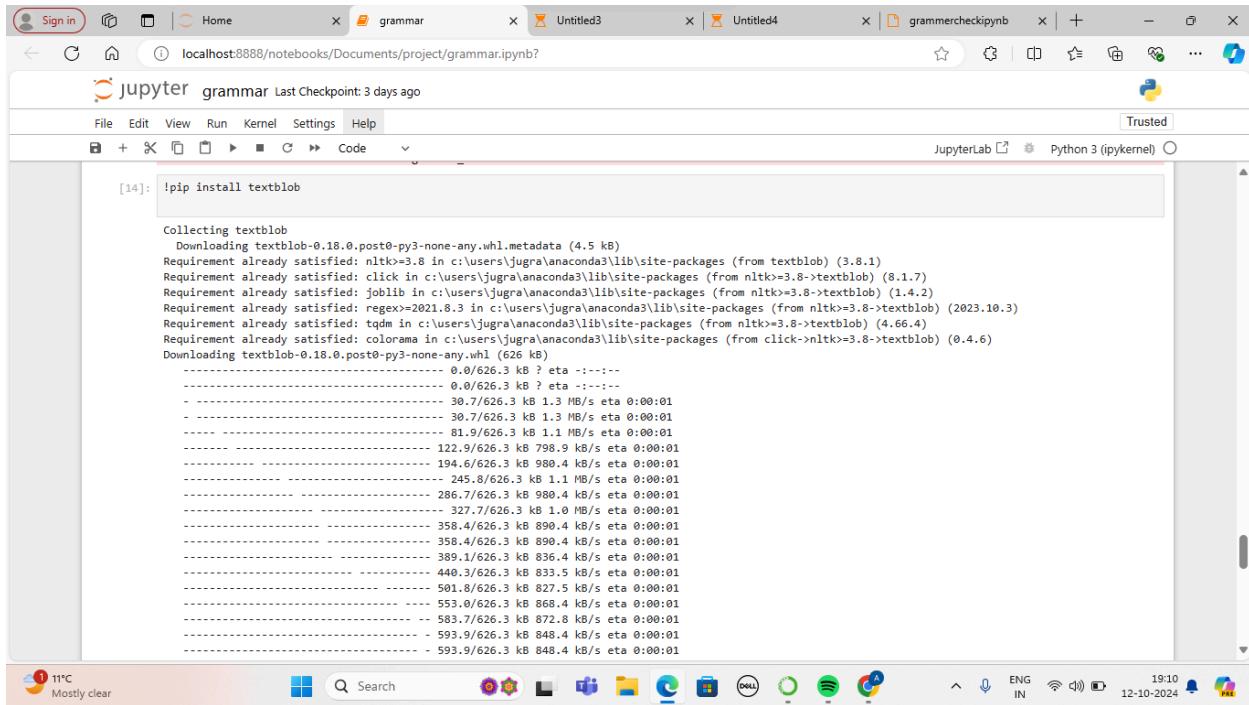
- Developed a model to detect the surrounding environment during audio input.
- The model also identifies if other individuals are present, ensuring focused and controlled input.



Arshdeep Kaur:

I am analyzing user's grammatical errors in an interview by converting audio to text ,after that checking the basic grammatical errors .

Analyzes the transcribed text for grammatical errors using a grammar-checking tool that is textblob



The screenshot shows a Jupyter Notebook interface with multiple tabs open. The active tab is titled 'grammar' and contains a code cell with the command `!pip install textblob`. The output of this command is displayed, showing the progress of the package download. The terminal output includes:

```
Collecting textblob
  Downloading textblob-0.18.0.post0-py3-none-any.whl.metadata (4.5 kB)
Requirement already satisfied: nltk>=3.8 in c:\users\jugra\anaconda3\lib\site-packages (from textblob) (3.8.1)
Requirement already satisfied: click in c:\users\jugra\anaconda3\lib\site-packages (from nltk>=3.8>textblob) (8.1.7)
Requirement already satisfied: joblib in c:\users\jugra\anaconda3\lib\site-packages (from nltk>=3.8>textblob) (1.4.2)
Requirement already satisfied: regex==2021.8.3 in c:\users\jugra\anaconda3\lib\site-packages (from nltk>=3.8>textblob) (2023.10.3)
Requirement already satisfied: tqdm in c:\users\jugra\anaconda3\lib\site-packages (from nltk>=3.8>textblob) (4.66.4)
Requirement already satisfied: colorama in c:\users\jugra\anaconda3\lib\site-packages (from click->nltk>=3.8>textblob) (0.4.6)
Downloading textblob-0.18.0.post0-py3-none-any.whl (626 kB)
----- 0.0/626.3 kB ? eta -----
```

The system tray at the bottom of the screen shows various icons, including a weather icon (11°C), a search icon, and a battery icon. The taskbar also displays several pinned application icons.

Converts audio into text using a speech recognition library and checking the error

The screenshot shows a Jupyter Notebook interface with multiple tabs open. The active tab is titled 'grammar' and shows Python code for speech recognition and grammar checking. The code imports necessary libraries (speech_recognition, moviepy.editor, AudioFileClip, TextBlob) and defines functions for converting MP3 files to WAV format, processing audio to text, and checking grammar. It also includes an example usage section. The notebook is running in a 'Python 3 (ipykernel)' kernel.

```
[1]: import speech_recognition as sr
from moviepy.editor import AudioFileClip
from textblob import TextBlob

# Initialize recognizer
recognizer = sr.Recognizer()

def mp3_to_wav(mp3_file):
    """Convert MP3 file to WAV format using moviepy."""
    wav_file = mp3_file.replace('.mp3', '.wav')
    audio_clip = AudioFileClip(mp3_file)
    audio_clip.write_audiofile(wav_file, codec='pcm_s16le')
    return wav_file

def process_audio(audio_file):
    """Convert audio to text and check grammar."""
    with sr.AudioFile(audio_file) as source:
        audio_data = recognizer.record(source)
        text = recognizer.recognize_google(audio_data)

    # Check grammar using TextBlob
    blob = TextBlob(text)
    corrected_text = str(blob.correct())

    return text, corrected_text

# Example usage
mp3_path = "Recording.mp3" # Replace with your MP3 file path
wav_path = mp3_to_wav(mp3_path)
transcribed_text, corrected_text = process_audio(wav_path)
```

This is the output

The screenshot shows the execution output of the code from the previous screenshot. The code prints the transcribed and corrected text. The output shows the original transcribed text followed by the corrected text. The notebook is running in a 'Python 3 (ipykernel)' kernel.

```
# Example usage
mp3_path = "question1.mp3" # Replace with your MP3 file path
wav_path = mp3_to_wav(mp3_path)
transcribed_text, corrected_text = process_audio(wav_path)

# Output results
print("Transcribed Text:")
print(transcribed_text)
print("\nCorrected Text:")
print(corrected_text)

MoviePy - Writing audio in question1.wav
MoviePy - Done.

Transcribed Text:
what are your greatest strengths

Corrected Text:
what are your greatest strength
```

JASKARAN SINGH-

Overview

This application uses your computer's webcam to detect where your eyes are looking. It can tell if you are looking straight at the camera or if your gaze is directed elsewhere. The app aims to help users focus on the camera during Mock Interview.

Key Features

- **Live Video Feed:** The app shows a live video feed from your webcam.
- **Eye Detection:** It detects your eyes and calculates their position.
- **Gaze Feedback:** It provides real-time feedback on whether you are looking at the camera.

How It Works

1. **Camera Access:** When you start the application, it will access your webcam to capture video.
2. **Eye Tracking:** The application identifies the position of your eyes in the video feed.
3. **Gaze Evaluation:** It checks if your eyes are looking straight ahead. If you look away, it will alert you.
4. **Visual Cues:** Circles are drawn around your pupils to indicate their positions, and helpful messages are displayed on the screen.

```
[1]: !pip install mediapipe --user

Requirement already satisfied: mediapipe in c:\users\jaska\appdata\roaming\python\python311\site-packages (0.10.14)
Requirement already satisfied: absl-py in c:\users\jaska\anaconda3\lib\site-packages (from mediapipe) (2.1.0)
Requirement already satisfied: attrs>=19.1.0 in c:\users\jaska\anaconda3\lib\site-packages (from mediapipe) (23.1.0)
Requirement already satisfied: flatbuffers>=2.0 in c:\users\jaska\anaconda3\lib\site-packages (from mediapipe) (24.3.25)
Requirement already satisfied: jax in c:\users\jaska\appdata\roaming\python\python311\site-packages (from mediapipe) (0.4.34)
Requirement already satisfied: jaxlib in c:\users\jaska\appdata\roaming\python\python311\site-packages (from mediapipe) (0.4.34)
Requirement already satisfied: matplotlib in c:\users\jaska\anaconda3\lib\site-packages (from mediapipe) (3.8.0)
Requirement already satisfied: numpy in c:\users\jaska\anaconda3\lib\site-packages (from mediapipe) (1.26.4)
Requirement already satisfied: opencv-contrib-python in c:\users\jaska\appdata\roaming\python\python311\site-packages (from mediapipe) (4.10.0.84)
Collecting protobuf<4.25.5,>=4.25.0
  Using cached protobuf-4.25.0-ab13-win_amd64.whl.metadata (541 bytes)
Requirement already satisfied: sounddevice>=0.4.4 in c:\users\jaska\appdata\roaming\python\python311\site-packages (from mediapipe) (0.5.1)
Requirement already satisfied: CFFI>=1.0 in c:\users\jaska\anaconda3\lib\site-packages (from sounddevice>0.4.4->mediapipe) (1.16.0)
Requirement already satisfied: ml-types>=0.2.0 in c:\users\jaska\anaconda3\lib\site-packages (from mediapipe) (0.3.2)
Requirement already satisfied: opt-einsim in c:\users\jaska\anaconda3\lib\site-packages (from jax->mediapipe) (3.3.0)
Requirement already satisfied: scipy>=1.10 in c:\users\jaska\anaconda3\lib\site-packages (from jax->mediapipe) (1.11.4)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\jaska\anaconda3\lib\site-packages (from matplotlib->mediapipe) (1.2.0)
Requirement already satisfied: cycler>=0.10.0 in c:\users\jaska\anaconda3\lib\site-packages (from matplotlib->mediapipe) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\jaska\anaconda3\lib\site-packages (from matplotlib->mediapipe) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\jaska\anaconda3\lib\site-packages (from matplotlib->mediapipe) (1.4.4)
Requirement already satisfied: packaging>=20.1 in c:\users\jaska\anaconda3\lib\site-packages (from matplotlib->mediapipe) (23.1)
Requirement already satisfied: pyparsing>=2.4.2 in c:\users\jaska\anaconda3\lib\site-packages (from matplotlib->mediapipe) (3.0.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\jaska\anaconda3\lib\site-packages (from matplotlib->mediapipe) (2.21)
Requirement already satisfied: six>=1.5 in c:\users\jaska\anaconda3\lib\site-packages (from CFFI>=1.0->sounddevice>0.4.4->mediapipe) (2.16.0)
Using cached protobuf-4.25.5-cp310-ab13-win_amd64.whl (413 kB)
Installing collected packages: protobuf
Successfully installed protobuf-4.25.5

[6]: import cv2
      import mediapipe as mp
```

```
[6]: View site information
      Eye Detection Last Checkpoint: 10 minutes ago
      File Edit View Run Kernel Settings Help
      Trusted
      JupyterLab Python 3 (ipykernel)
```

```
[6]: import cv2
import mediapipe as mp

# Initialize Mediapipe Face Mesh and drawing utilities
mp_face_mesh = mp.solutions.face_mesh
mp_drawing = mp.solutions.drawing_utils

# Set up Face Mesh with appropriate parameters
face_mesh = mp_face_mesh.FaceMesh(
    max_num_faces=1, refine_landmarks=True,
    min_detection_confidence=0.5, min_tracking_confidence=0.5
)

# Define indices for eye landmarks
LEFT_EYE_INDICES = [133, 144, 145, 163, 157, 158, 160, 161] # Left eye Landmarks
RIGHT_EYE_INDICES = [362, 373, 374, 392, 386, 387, 389, 390] # Right eye Landmarks

def main():
    # Start video capture
    cap = cv2.VideoCapture(0)

    while cap.isOpened():
        success, frame = cap.read()
        if not success:
            print("Failed to grab frame")
            break

        # Flip and convert the frame to RGB for Mediapipe processing
        frame = cv2.flip(frame, 1)
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Process the frame to get facial Landmarks
        results = face_mesh.process(rgb_frame)
```

```

# Flip and convert the frame to RGB for MediaPipe processing
frame = cv2.flip(frame, 1)
rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

# Process the frame to get facial Landmarks
results = face_mesh.process(rgb_frame)

# Assume the user is not looking at the camera initially
looking_straight = False

if results.multi_face_landmarks:
    # Get the landmarks of the first detected face
    landmarks = results.multi_face_landmarks[0]

    ih, iw, _ = frame.shape # Image dimensions

    # Calculate pupil positions by averaging the eye landmarks
    left_eye_x = sum(landmarks.landmark[i].x for i in LEFT_EYE_INDICES) / len(LEFT_EYE_INDICES)
    left_eye_y = sum(landmarks.landmark[i].y for i in LEFT_EYE_INDICES) / len(LEFT_EYE_INDICES)

    right_eye_x = sum(landmarks.landmark[i].x for i in RIGHT_EYE_INDICES) / len(RIGHT_EYE_INDICES)
    right_eye_y = sum(landmarks.landmark[i].y for i in RIGHT_EYE_INDICES) / len(RIGHT_EYE_INDICES)

    # Convert normalized landmarks to pixel coordinates
    left_pupil_pos = (int(left_eye_x * iw), int(left_eye_y * ih))
    right_pupil_pos = (int(right_eye_x * iw), int(right_eye_y * ih))

    # Draw circles around the pupils
    cv2.circle(frame, left_pupil_pos, 5, (0, 255, 0), -1)
    cv2.circle(frame, right_pupil_pos, 5, (0, 255, 0), -1)

    # Check if the user is Looking at the camera based on pupil positions
    # Define thresholds

```

Output

Looking in the Camera

When the eyes are detected looking in the camera, a message appears “**Looking Good!**”

```

[1]: !pip install mediapipe --user
Requirement already satisfied: mediapipe in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: absl-py in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: attrs>=19.1.0 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: flatbuffers>2.0 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: jax in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: jaxlib in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: matplotlib in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: numpy in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied:opencv-contrib-python in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Collecting protobuf<4.25.3 (from mediapipe)
  Using cached protobuf-4.25.5-cp310-ab32-win_amd64.whl.metadata
Requirement already satisfied: sounddevice>0.4.4 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: CFFI>=1.14.0 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: ml-dtypes>=0.2.0 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: opt-einsum in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: scipy>=1.10 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: colorama>0.4.1 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: cycler>=0.10 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: fonttools>=4.22.2 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: packaging>=20.0 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: pillow>=6.2.0 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: pyrsistent>=2.3.1 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: python-dateutil>=2.7 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: pycparser in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Requirement already satisfied: six>=1.5 in c:\users\jaskal\anaconda3\envs\eye\lib\site-packages
Using cached protobuf-4.25.5-cp310-ab32-win_amd64.whl (413 kB)
Installing collected packages: protobuf
Successfully installed protobuf-4.25.5

```

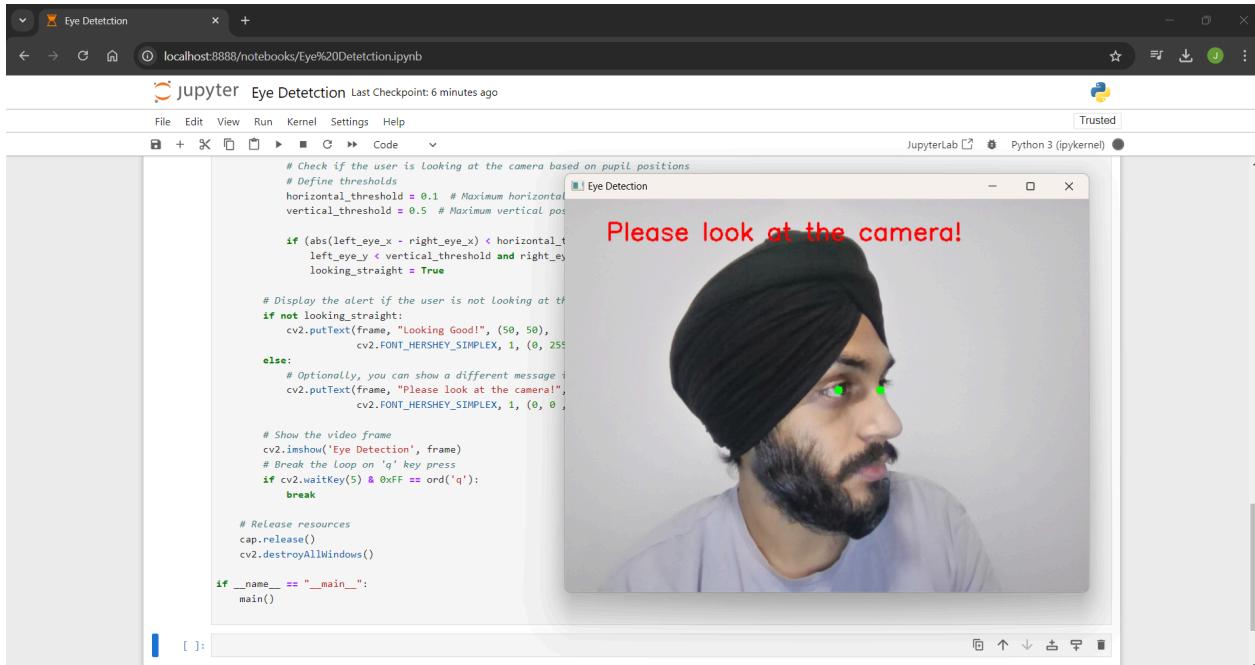
```

[1]: import cv2
      import mediapipe as mp

```

Looking Away from the Camera

When the eyes are detected looking away from the camera, a message appears “Please look at the camera!”



Dilpreet Kaur

I am working on Filler words Detection.

```
[1]: pip install SpeechRecognition pydub
      Defaulting to user installation because normal site-packages is not writeable
      Requirement already satisfied: SpeechRecognition in c:\users\dell\appdata\roaming\python311\site-packages (3.10.4)
      Requirement already satisfied: pydub in c:\users\dell\appdata\roaming\python311\site-packages (0.25.1)
      Requirement already satisfied: requests>=2.26.0 in c:\programdata\anaconda3\lib\site-packages (from SpeechRecognition) (2.31.0)
      Requirement already satisfied: typing-extensions in c:\programdata\anaconda3\lib\site-packages (from SpeechRecognition) (4.9.0)
      Requirement already satisfied: charset-normalizer<4,>=2 in c:\programdata\anaconda3\lib\site-packages (from requests>=2.26.0->SpeechRecognition) (2.0.4)
      Requirement already satisfied: idna<4,>=2.5 in c:\programdata\anaconda3\lib\site-packages (from requests>=2.26.0->SpeechRecognition) (3.4)
      Requirement already satisfied: urllib3<3,>=1.21.1 in c:\programdata\anaconda3\lib\site-packages (from requests>=2.26.0->SpeechRecognition) (2.0.7)
      Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anaconda3\lib\site-packages (from requests>=2.26.0->SpeechRecognition) (2024.2.2)
      Note: you may need to restart the kernel to use updated packages.

[10]: # Import necessary Libraries
      from pydub import AudioSegment
      import speech_recognition as sr

      # Convert MP3 to WAV
      audio_file = "Recording.mp3"
      audio = AudioSegment.from_mp3(audio_file)
      audio.export("Recording.wav", format="wav")

      # Initialize recognizer
      recognizer = sr.Recognizer()

      # Recognize speech from the WAV file
      with sr.AudioFile("Recording.wav") as source:
          audio_data = recognizer.record(source)

      # Perform speech recognition and filler word removal
      try:
          # Recognize speech using Google Speech Recognition
          text = recognizer.recognize_google(audio_data)
          print("Original Recognized Text:", text)

          # Define filler words
          filler_words = ["um", "uh", "like", "you know", "basically"]

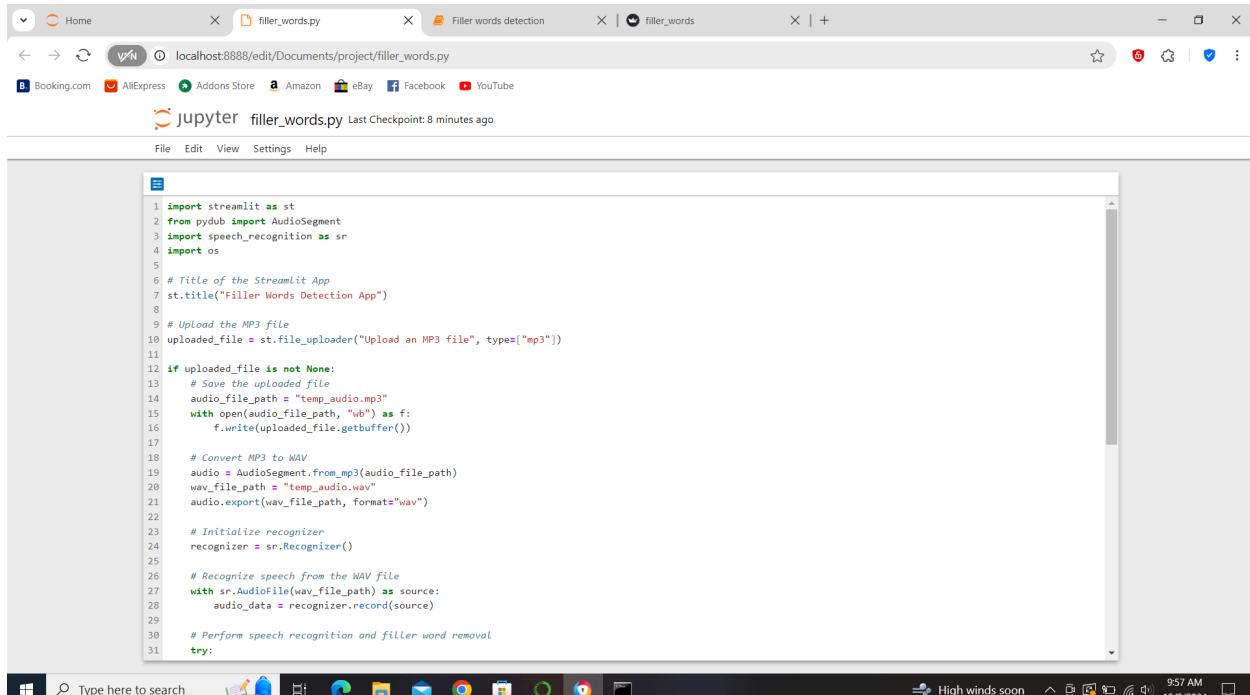
          # Split the text into words
          words = text.split()

          # Remove filler words from the recognized text
          cleaned_text = " ".join([word for word in words if word.lower() not in filler_words])

          # Output cleaned text
          print("Cleaned Text Without Filler Words:", cleaned_text)

      except sr.UnknownValueError:
          print("Speech recognition could not understand audio")
      except sr.RequestError as e:
          print(f"Could not request results from Google Speech Recognition service; {e}")

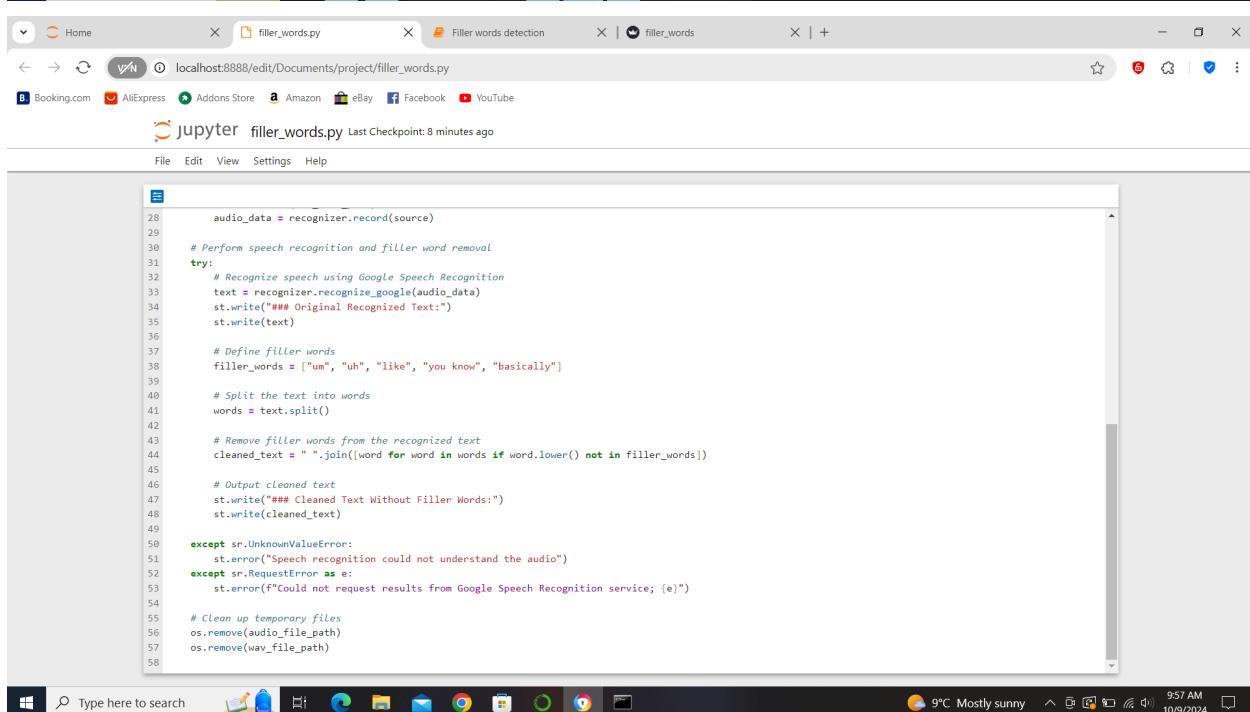
      Original Recognized Text: I was like you know going to the store but I forgot
      Cleaned Text Without Filler Words: I was you know going to the store but I forgot
```



```

1 import streamlit as st
2 from pydub import AudioSegment
3 import speech_recognition as sr
4 import os
5
6 # Title of the Streamlit App
7 st.title("Filler Words Detection App")
8
9 # Upload the MP3 file
10 uploaded_file = st.file_uploader("Upload an MP3 file", type=["mp3"])
11
12 if uploaded_file is not None:
13     # Save the uploaded file
14     audio_file_path = "temp_audio.mp3"
15     with open(audio_file_path, "wb") as f:
16         f.write(uploaded_file.getbuffer())
17
18     # Convert MP3 to WAV
19     audio = AudioSegment.from_mp3(audio_file_path)
20     wav_file_path = "temp_audio.wav"
21     audio.export(wav_file_path, format="wav")
22
23     # Initialize recognizer
24     recognizer = sr.Recognizer()
25
26     # Recognize speech from the WAV file
27     with sr.AudioFile(wav_file_path) as source:
28         audio_data = recognizer.record(source)
29
30     # Perform speech recognition and filler word removal
31     try:

```



```

28         audio_data = recognizer.record(source)
29
30     # Perform speech recognition and filler word removal
31     try:
32         # Recognize speech using Google Speech Recognition
33         text = recognizer.recognize_google(audio_data)
34         st.write("## Original Recognized Text:")
35         st.write(text)
36
37         # Define filler words
38         filler_words = ["um", "uh", "like", "you know", "basically"]
39
40         # Split the text into words
41         words = text.split()
42
43         # Remove filler words from the recognized text
44         cleaned_text = " ".join([word for word in words if word.lower() not in filler_words])
45
46         # Output cleaned text
47         st.write("## Cleaned Text Without Filler Words:")
48         st.write(cleaned_text)
49
50     except sr.UnknownValueError:
51         st.error("Speech recognition could not understand the audio")
52     except sr.RequestError as e:
53         st.error(f"Could not request results from Google Speech Recognition service; {e}")
54
55     # Clean up temporary files
56     os.remove(audio_file_path)
57     os.remove(wav_file_path)
58

```

Deployment

The screenshot shows a web browser window with multiple tabs open. The active tab is titled "Filler words detection" and displays the URL "localhost:8501". The page content is as follows:

Filler Words Detection App

Upload an MP3 file

Drag and drop file here
Limit 200MB per file • MP3

Recording.mp3 163.5KB x

Original Recognized Text:

I was like you know going to the store but I forget

Cleaned Text Without Filler Words:

I was you know going to the store but I forget

At the bottom of the browser window, the taskbar is visible, showing various pinned icons and system status.