ASPIRA AN AI INTERVIEWER

MINI PROJECT REPORT

Submitted by

ARNAV JAGADEESH(SNG22CS045)

AKSHAY PRAKASH (SNG22CS026)

ANANDHU SURESH(SNG22CS034)

AJAY KRISHNA PR(SNG22CS022)

to

APJ Abdul Kalam Technological University

in partial fulfillment of the requirements for the award of the Degree of

Bachelor of Technology

In

Computer Science and Engineering

Department of Computer Science and Engineering

Sree Narayana Gurukulam College of Engineering, Kadayiruppu,682311

APRIL 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, SREE NARAYANA GURUKULAM COLLEGE OF ENGINEERING, KADAYIRUPPU, 682311

(Affiliated to APJ Abdul Kalam Technological University & Approved by A.I.C.T.E)

CERTIFICATE

This is to certify that the mini project report, "**ASPIRA**" submitted by **ARNAV JAGADEESH, AKSHAY PRAKASH,AJAY KRISHNA PR,ANANDHU SURESH** to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering is a bona fide record of the project work carried out by him under our guidance and supervision.

This report in any form has not been submitted to any other University or Institute for any purpose.

Dr Smitha Suresh	**Mr.Saini Jacob Soman**	**Mrs.Beeta Narayan**
(Prof. CSE Dept)	(Asst. Prof . CSE Dept)	(Prof. CSE Dept)
HEAD OF THE DEPARTMENT	**COORDINATOR**	**PROJECT GUIDE**

Submitted for the University Evaluation on :

University Register Number :

Internal Examiner External Examiner

DECLARATION

I undersigned hereby declare that the project report "**Aspira"**, submitted for partial fulfillment of the requirements for the award of degree of Bachelor of Technology of the APJ Abdul Kalam Technological University, Kerala is a bona fide work done by me under supervision of Prof. Dr. Smitha Suresh. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place: Kadayiruppu

Date :

Arnav Jagadeesh

Akshay Prakash

Ajay krishna P.R

Anandhu suresh

COURSE OUTCOME AND PROGRAM OUTCOMES

COURSE OUTCOMES: After the completion of the course the student will be able to

CO1	Identify technically and economically feasible problems (Cognitive Knowledge Level: Apply)
CO2	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes (Cognitive Knowledge Level: Apply)
CO3	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions of minimal complexity by using modern tools &advanced programming techniques (Cognitive Knowledge Level: Apply)
CO4	Prepare technical report and deliver presentation (Cognitive Knowledge Level: Apply)
CO5	Apply engineering and management principles to achieve the goal of the project (Cognitive Knowledge Level: Apply)

Program outcomes
Engineering Graduates will be able to:
PO1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

C0, PO & PSO MAPPING

	PO1 (Engineering Knowledge)	PO2 (Problem analysis)	PO3 (Design/development of solutions)	PO4 (Conduct investigations of complex problems)	PO5 (Modern tool usage)	PO6 (The engineer and society)	PO7 (Environment and sustainability)	PO8 (Ethics)	PO9 (Individual and teamwork)	PO10 (Communication)	PO11 (Project management and finance)	PO12. (Lifelong learning)	PSO1 (Finding Innovative)	PSO2 (Software Development)
CO1	3	3	2	2		2	2	3	2	3	3	2	3	2
CO2	2	3	3	2	3	2		2	2	2	3	3	2	3
CO3	2	3	1	3	3	3	2	2	2	3	2	3	3	3
CO4	3	2	3	2	3			2	2	2	2	2	3	3
CO5	2	3	2	3	2	3	2	2	2		2	2	3	3
Avg.	2.4	2.8	2.2	2.4	2.75	2.5	2.33	2.2	2.0	2.5	2.4	2.4	2.8	2.8

PO PSO ATTAINMENT AND JUSTIFICATION

PO	Attained point (0/1/2/3)	Justification
PO1	2	We have used the application of mathematical concepts, image processing algorithms, and knowledge of machine learning techniques to detect fake currency. The use of Python libraries demonstrates the application of engineering knowledge.
PO2	3	The project involves analyzing complex engineering problems related to fake currency detection, considering factors such as image features, pattern recognition, and classification. Extensive research literature review is conducted to understand the problem and formulate effective solutions.
PO3	3	The project involves designing and developing a solution for fake currency detection by creating a system that utilizes image processing techniques and machine learning algorithms. Consideration is given to public safety, cultural sensitivities, and the use of appropriate tools and technologies.
PO4	3	The project requires conducting investigations to evaluate the effectiveness of various image processing techniques and machine learning models for detecting fake currency. Experiments are performed, data is analyzed, and valid conclusions are drawn based on the results.
PO5	3	The project extensively uses Python libraries such as OpenCV for image processing, scikit-learn for machine learning, and GUI libraries for creating a user-friendly interface. The appropriate selection and application of these modern tools demonstrate proficiency in their usage.

PO6	2	The project considers the societal impact of fake currency and the responsibility of engineers in combating this issue. Ethical considerations such as maintaining public trust, safeguarding financial systems, and preventing fraudulent activities are addressed
PO7	0	The project does not focus on environmental aspects.
PO8	2	The project adheres to ethical principles by addressing the legal and moral implications of fake currency detection. It emphasizes the responsibility of the engineer to maintain integrity, accuracy, and privacy during the detection process
PO9	3	Demonstrates the ability to work both individually (e.g., programming, algorithm implementation) and in a team (e.g., collaborating on design decisions, sharing responsibilities). Effective communication and coordination among team members are essential for project success.
PO10	2	The project requires effective communication within the engineering community and with society at large (e.g., presenting the project, documenting the system). Reports, documentation, and presentations are prepared to convey information clearly.
PO11	2	It involves managing resources, setting goals, and planning tasks to ensure the timely completion of deliverables. Knowledge of project management principles, such as task scheduling and resource allocation, contributes to successful project execution.
PO12	3	The project necessitates continuous learning to stay updated with the latest advancements in image processing, machine learning, and fraud detection techniques. The project also encourages self-directed learning and prepares the individual for lifelong learning in the evolving field of technology.

PSO1	3	Utilized industry-standard software and libraries commonly used in the field. Proper software architecture design was followed, incorporating relevant diagrams such as use case diagrams and ER diagrams. This demonstrates adherence to industry best practices.
PSO2	3	Emphasized thinking innovatively and seeking solutions beyond the scope of the academic curriculum. By referring to blog posts, research papers, and tutorial videos, the team developed the ability to think creatively and apply innovative approaches to address challenges encountered during the project

TABLE OF CONTENTS

CONTENTS **PAGE NO**

ACKNOWLEDGEMENT

I extend my sincere gratitude to all those who have contributed to the successful completion of this project. First and foremost, I express my heartfelt appreciation to **Mrs.Beeta Narayan**(*Asst. Prof. CSE Dept*), my project guide, whose invaluable guidance, unwavering support, and insightful feedback have been instrumental throughout the entire duration of this project. Her expertise, patience, and encouragement have significantly enriched my learning experience and contributed immensely to the quality of this work.

I am also deeply thankful to **Mr.Saini Jacob Soman**(*Asst. Prof. CSE Dept*), the assigned faculty member, for her valuable input, constructive criticism, and continuous support, which have greatly enhanced the depth and scope of this project. Her expertise and commitment to academic excellence have been a source of inspiration throughout this endeavor.

Furthermore, I would like to acknowledge the Department of Computer Science and Engineering, Sree Narayana Gurukulam College of Engineering, Kadayiruppu for providing the necessary resources, facilities, and a conducive environment for the successful execution of this project. Last but not least, I am grateful to my family and friends for their unwavering encouragement, understanding, and patience throughout this journey. This project would not have been possible without the collective efforts and support of all those mentioned above, and for that, I am truly grateful.

ABSTRACT

The AI Interviewer project aims to develop an intelligent system for conducting mock interviews on a web-based platform. Leveraging advanced natural language processing (NLP) and machine learning algorithms, including generative models like GPT-based architectures, the system delivers dynamic and realistic interview experiences. The platform incorporates AI avatar technology, text-to-speech (TTS), speech-to-text (STT), and virtual characters to create life-like interactions that closely mimic real interviews. The system facilitates real-time interviews by tailoring questions based on candidates' responses and generates contextually relevant, adaptive follow-ups. Designed to simulate realistic interview scenarios, it evaluates users' skills, personality traits, and communication abilities. The intuitive interface ensures ease of use, while its integration capabilities make it accessible to individuals, academic institutions, and organizations. With its intuitive interface and integration features, the AI Interviewer enhances and updates users' interview skills, boosts confidence, and ensures readiness for real-world opportunities.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

ASPIRA is an AI-driven web-based platform designed for mock interviews, providing realistic interview simulations. It utilizes advanced natural language processing (NLP) and machine learning algorithms to evaluate users' skills, personality, and communication abilities. The platform integrates AI avatars, text-to-speech (TTS), speech-to-text (STT), and virtual characters to create lifelike interview experiences. ASPIRA is beneficial for individuals, academic institutions, and organizations looking to enhance interview readiness.

1.1 OVERVIEW

ASPIRA adapts dynamically to candidates' responses, generating contextually relevant follow-up questions in real-time. The platform ensures a personalized and immersive interview experience using AI technologies. By simulating real-world interviews, ASPIRA helps candidates gain confidence, refine their skills, and prepare effectively. It serves as a tool for students, job seekers, recruiters, and HR professionals to assess and improve interview performance. Additionally, ASPIRA aims to promote fair and unbiased candidate evaluations.

Key Features:

Dynamic Adaptation: AI generates contextually relevant questions based on user responses.

Realistic Interaction: AI avatars and virtual characters enhance the interview experience.

Multiple Applications: Beneficial for individuals, educational institutions, and organizations.

Fair & Unbiased Evaluation: Ensures equal opportunities for all candidates.

-

1.2 OBJECTIVES

Providing a Realistic Simulation of Job Interviews

ASPIRA creates an AI-driven virtual interview environment that closely resembles real-world job interviews, making the experience dynamic and interactive.

Helping Individuals Build Confidence and Improve Communication

The platform enables job seekers to practice interviews, enhance their speaking skills, and reduce nervousness through repeated exposure.

Offering Personalized Feedback for Skill Improvement

ASPIRA analyses user responses, body language, and speech patterns to provide detailed feedback, helping candidates refine their performance.

Assisting Academic Institutions in Career Development

Universities and training centres can use ASPIRA as a tool to prepare students for job interviews, enhancing their employability and professional skills.

Helping Recruiters and HR Professionals in Candidate Evaluation

The platform automates initial screening processes, allowing recruiters to assess candidates more efficiently while reducing workload.

Promoting Fair and Unbiased Assessments

By eliminating human biases, ASPIRA ensures an equal and objective evaluation process, giving all candidates a fair opportunity.

1.3 SCOPE

The scope of ASPIRA extends across individuals, academic institutions, and organizations by providing AI-driven mock interviews to enhance interview skills, streamline hiring processes, and support career development. Additionally, it has the potential to expand into areas such as online education, mental health assessments, and behavioural analysis, making it a versatile tool for various industries.

For Individuals

ASPIRA helps job seekers practice interviews, refine their skills, and gain confidence through AI-driven mock interviews.

For Academic Institutions

Universities and training centres can integrate ASPIRA into their career development programs, allowing students to prepare effectively for job placements.

For Organizations and Recruiters

HR professionals can use ASPIRA to streamline the hiring process by automating initial interview screenings and evaluating candidates more efficiently.

For Future Enhancements

ASPIRA can expand beyond job interviews to areas such as online education, mental health assessments, customer service training, and behavioural studies.

CHAPTER 2

LITERATURE REVIEW

Tensor Flow-Based Automatic Personality Recognition Used in Asynchronous video interview,2021. (Authors : Hung-Yue Suen , Kuo-En Hung , Chien-Liang Lin)
The paper presents an AI-based Automatic Personality Recognition (APR) system that leverages TensorFlow and Convolutional Neural Networks (CNNs) for deep learning-based feature extraction and personality prediction. The system analyses data collected from asynchronous video interviews (AVI) of 120 real job applicants. Facial expressions are assessed using OpenCV, D lib, and a semi-supervised learning approach, achieving a high accuracy rate of 90.9% to 97.4% in predicting personality traits. The approach eliminates biases inherent in human assessments, is cost-effective, and scalable compared to manual personality evaluations, and integrates well with cloud-based platforms. However, there are some limitations, such as the small dataset of only 120 participants, which potentially reduces generalizability. Additionally, the system is focused on a specific professional group, limiting its broader applicability, and its predictions may be biased due to the non-diverse datasets. The results are also highly dependent on image quality and facial visibility. Looking ahead, the paper suggests incorporating multimodal features, such as audio and visual inputs, to enhance accuracy. Expanding the datasets for greater diversity and scalability, improving the model's robustness against varying interview environments (e.g., lighting and background noise), and applying the technology in areas beyond employment, such as education, mental health, and personalized marketing, are all potential future directions for the research.

Gaze and Head Rotation Analysis in a Triadic VR Job Interview Simulation ,2023. (Authors: Saygin Artiran, Poorva S. Bedmutha, Aaron Li, Pamela Cosman)

The paper presents a framework for a Virtual Reality (VR)-based simulation for job interviews, utilizing the HTC Vive Pro Eye VR headset integrated with Unity and Vive for data capturing. The system employs eye tracking, head rotation monitoring, and voice activity detection using the SRanipal SDK. The study analyses the social modulation of gaze and head orientation in triadic interactions during job interviews. The approach

enables realistic and immersive job interview simulations, providing an inexpensive, solo practice option without the need for a human coach. It helps increase self-confidence and reduces anxiety through repeatable practice. However, there are some limitations, such as calibration challenges in eye-tracking technologies, which can affect accuracy. The study also points to potential biases, as the participant sample is small and predominantly male, and the dependency on VR hardware, which might not be comfortable for all users. Looking ahead, the paper suggests expanding the system to diverse user groups to prevent behavioural biases, developing varied question sets for broader training scenarios, and adding emotional and conversational dynamics between virtual interviewers to enhance the simulation's realism and effectiveness.

You're Hired! Effect of Virtual Agents' Social Status and Social Attitudes on Stress Induction in Virtual Job Interviews ,2024.(Authors: Celia Kessassi, Cedric Dumas, Caroline G. L. Cao, Mathieu Chollet)

This study investigates the impact of virtual agents' social status and attitudes on stress induction during virtual job interviews. The virtual agents were controlled using the Wizard of Oz paradigm and equipped with meta-human models and pre-recorded voices. Participants' stress levels were measured using physiological sensors and subjective questionnaires before, during, and after the interviews. The advantages of this methodology include providing a controlled environment to study stress mechanisms in social evaluation scenarios, incorporating virtual reality for immersive simulations that increase ecological validity, and offering insights into how social status and feedback types influence anxiety, especially in individuals with social anxiety. However, there are some drawbacks, such as the limited ability to generalize findings due to reliance on virtual avatars and the lack of significant differentiation in stress levels based on virtual agents' status. The study also mentions that its findings may not fully replicate real-world job interview dynamics. Future research could focus on improving the realism and perceived status of virtual agents to better mimic real-world interactions, studying the impact of additional factors such as cultural differences and non-verbal communication, and investigating personalized feedback mechanisms to enhance training effectiveness for socially anxious individuals.

Intelligent Deception Detection Through Machine-Based Interviewing ,2020. (Authors: Jim O'Shea, Keeley Crockett, Wasiq Khan, Philippos Kindynis, Athos Antoniades, Georgios Boultadakis)

This paper presents a system that uses an artificial neural network (ANN)-based framework to detect deception by analysing non-verbal behaviour, such as facial micro-gestures. The system collects data from interviews conducted by an avatar in border control scenarios. Participants were stratified into truthful and deceptive categories, and video data was analysed. The Silent Talker, a patented deception detection system, identifies micro-gesture patterns to classify truthfulness or deception. The system's advantages include being effective in scenarios where large-scale interviews are necessary, reducing reliance on human observers, and minimizing fatigue and subjective bias. It has demonstrated classification accuracy between 74% and 87%. However, the system is limited to specific scenarios, such as border control applications, and could produce false positives due to varying emotional states unrelated to deception. Future directions for this research include enhancing avatar realism to improve human interaction, expanding applications beyond border control (e.g., fraud detection in financial institutions), and integrating advanced sensors for richer behavioral data analysis.

Fairness-Aware Multimodal Learning in Automatic Video Interview Assessment 2023.
(Authors: Changwoo Kim, Jinho Choi, Jongyeon Yoon, Daehun Yoo, Woojin Lee)

This study developed a fairness-aware deep learning model to predict interview scores using multimodal data (video, audio, text). The model incorporated regularization terms based on the Wasserstein distance to balance fairness and accuracy, and adversarial training was used to encode representations independent of sensitive attributes like gender. The system's advantages include maintaining fairness across groups by addressing sensitive attributes and demonstrating superior performance in both fairness and accuracy over existing methods. It also allows for flexible trade-off adjustment between fairness and accuracy. However, the model is computationally expensive for

large-scale multimodal datasets, and prioritizing fairness may reduce overall model accuracy. Future work could extend the fairness framework to include additional sensitive attributes, optimize the model to reduce computational overhead for real-world deployment, and explore broader applications in domains such as education.

AI-based Behavioural Analyser for Interviews/Viva ,2021.(Authors: Venuri Amalya, Raveen Dissanayaka, Lahiru Lakshan)

This system analyses interviewees' non-verbal behaviours using deep learning and machine learning models. It includes smile detection and genuineness evaluation using CNN models, eye gaze and blink detection with VGG16-based CNNs, and head movement tracking using a CNN regression model. The system achieves high accuracy (>85%) in detecting smiles, eye gaze, emotion, and head movements. It also effectively assesses Big Five personality traits with Random Forest (RF) models, ensuring unbiased evaluation. The system provides comparative analysis of individual vs. group performance and non-interview vs. interview contexts. The advantages include accurate and unbiased assessment of non-verbal behaviours and personality traits. However, the system faces challenges due to limited training data for certain features, which required augmentation, and its dependence on high-quality video inputs, which might be hindered by network issues in virtual interviews. Future research could focus on enhancing real-time capabilities for dynamic behaviour analysis, integrating question generation based on candidates' behavioural states, customizing personality assessments for specific job roles, and expanding the system to broader contexts like online education and general behavioural studies.

Virtual Speech Anxiety Training - Effects of Simulation Fidelity on User Experience ,2023.(Authors: Sandra Poeschl, Nicola Doering)

This study tested two levels of simulation fidelity (static vs. animated audience) in a virtual speech anxiety training session. The study found that an animated audience led to significantly higher anxiety effects during public speaking, which can be useful for realistic training in anxiety-inducing environments. It established that simulation fidelity has a measurable impact on anxiety levels, which is crucial for therapy applications. The

advantages include providing an immersive and controlled environment for speech anxiety training. However, there were no significant effects of simulation fidelity on virtual presence or perceived realism, and the alpha-version application lacked realism due to limited audience models and non-verbal behaviours. The study did not include real performance measures, limiting its scope. Future research could explore whether virtual presence is necessary for VR therapy and training, analyse the relationship between presence and performance, improve the application's realism with more diverse and realistic audience behaviours, and include performance metrics to better evaluate training effectiveness.

Automated Analysis and Prediction of Job Interview Performance ,2019. (Authors: Iftekhar Naim, M. Iftekhar Tanveer, Daniel Gildea, Mohammed (Ehsan) Hoque)

This research uses the MIT Interview Dataset, consisting of 138 audio-visual recordings of mock interviews, to analyse prosodic features (e.g., pitch, vocal intensity, pause duration), lexical features (e.g., speaking rates, LIWC categories), and learned topics. The study evaluates interview performance through correlation coefficients and area under the ROC curve (AUC). The advantages include providing quantitative insights into verbal and non-verbal behaviours in interviews, supporting real-time feedback via multimodal analysis, and predicting multiple traits with high accuracy. However, the dataset is limited to MIT students, which may not reflect diverse populations, and the ground truth ratings rely on Mechanical Turk workers, which may differ from expert evaluations. The study also lacks stress simulation in mock interviews and has limited modelling of rare but impactful moments. Future research could focus on expanding datasets to include more diverse participants and real-world stress scenarios, integrating temporal features to capture dynamic interactions, and exploring anomaly detection for rare but impactful behaviours.

CHAPTER 3

SYSTEM ANALYSIS

Here I am familiarizing with the requirements that the system needs. They are listed below by categories.

3.1 FUNCTIONAL REQUIREMENTS

1. **User Authentication**
    - Users can register and log in using **email**, **social media**, or **organizational credentials**.
2. **Interview Process**
    - The system must conduct **adaptive interviews** based on the responses of the user.
    - The **virtual avatar** should be able to display **realistic body language** and modulate its **voice** according to the conversation.
3. **Analytics and Feedback**
    - The system must provide feedback on the user's:
        - Communication skills
        - Confidence
        - Body language
4. **User Access**
    - The system should support access for **individual institutions** and **organizations**, allowing different user types to interact with it.

3.2 NON-FUNCTIONAL REQUIREMENTS

1. **Performance Requirements**
    - The system should be able to support **up to 50 concurrent users**.

- The **response time** for real-time interactions should not exceed **500 ms** to ensure smooth and timely communication.

2. **Security Requirements**
    - The system must ensure **data encryption** for **user information** and **recordings** to protect sensitive data.
3. **Usability Requirements**
    - The interface should be **intuitive** and **user-friendly** to ensure ease of use.
    - The system should include **accessibility features** like **screen readers** and **keyboard navigation** for users with disabilities.

3.3 SOFTWARE AND HARDWARE REQUIREMENTS

HARDWARE REQUIREMENTS

1. **Processor**:
    - **Quad-core** (minimum)
    - **2.5 GHz or higher** (e.g., Intel Xeon or AMD EPYC)
2. **RAM**:
    - Minimum of **16GB**, **32GB recommended** for large-scale use
3. **Storage**:
    - SSD with at least **500GB**, expandable for user data and logs
4. **GPU**:
    - **NVIDIA GPUs** with **CUDA support** (e.g., NVIDIA A100, RTX 3090) for AI model processing
5. **Network**:
    - High-speed internet (at least **1 Gbps**)

SOFTWARE REQUIREMENTS

Programming Languages

- **Node.js** (for backend server-side logic)
- **Python** (for AI/ML, NLP, etc.)

2. **Database**
    - **MongoDB** (for storing user data, logs, and system configurations)
3. **Natural Language Processing (NLP)**
    - **Transformers** (for language models)
    - **spaCy**
    - **NLTK**
    - **TextBlob**
4. **Speech Recognition**
    - **Speech-to-Text (STT)**
    - **Pytesseract** (for OCR tasks)
5. **Machine Learning & Deep Learning**
    - **PyTorch** (for training and running ML models)
6. **Visual Avatar and Graphics**
    - **OpenCV-Python** (for computer vision tasks)
    - **Blender Py**
    - **thon API** (for rendering and animation of the virtual avatar)

CHAPTER 4

DESIGN

4.1 ARCHITECTURE DIAGRAM

The Architectural diagram is the graphical representation of a set of concepts that are part of an architecture, including their principles, elements and components.

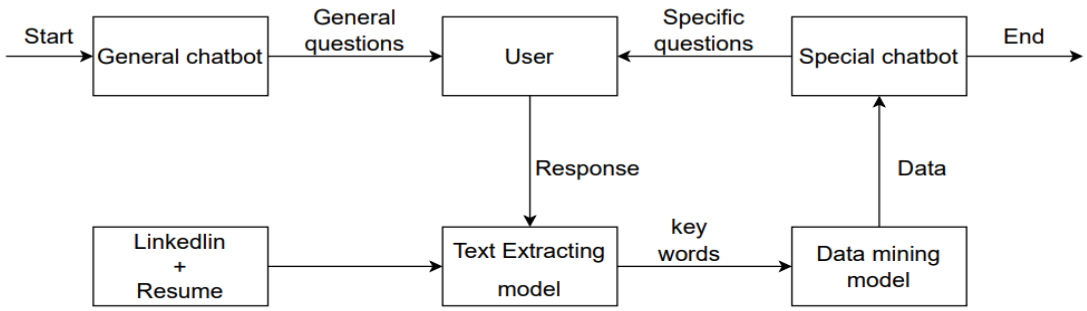

Figure 1: Architecture Diagram

4.2 SYSTEM DESIGN DIAGRAM

A system design diagram visually represents the architecture and components of a system, illustrating how different elements interact and function together to achieve the system's goals.

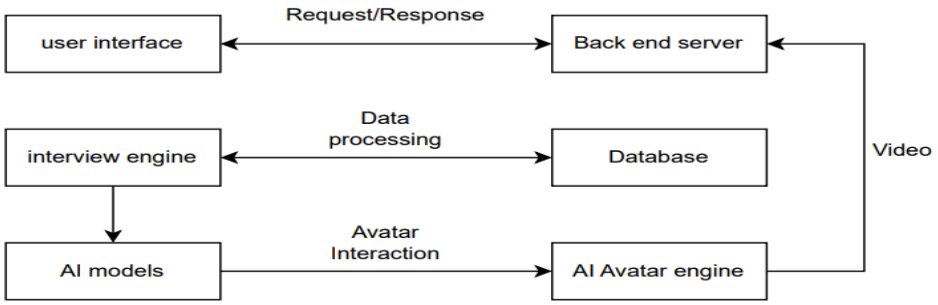

Figure 2: System Design Diagram

4.3 USECASE DIAGRAM

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well.

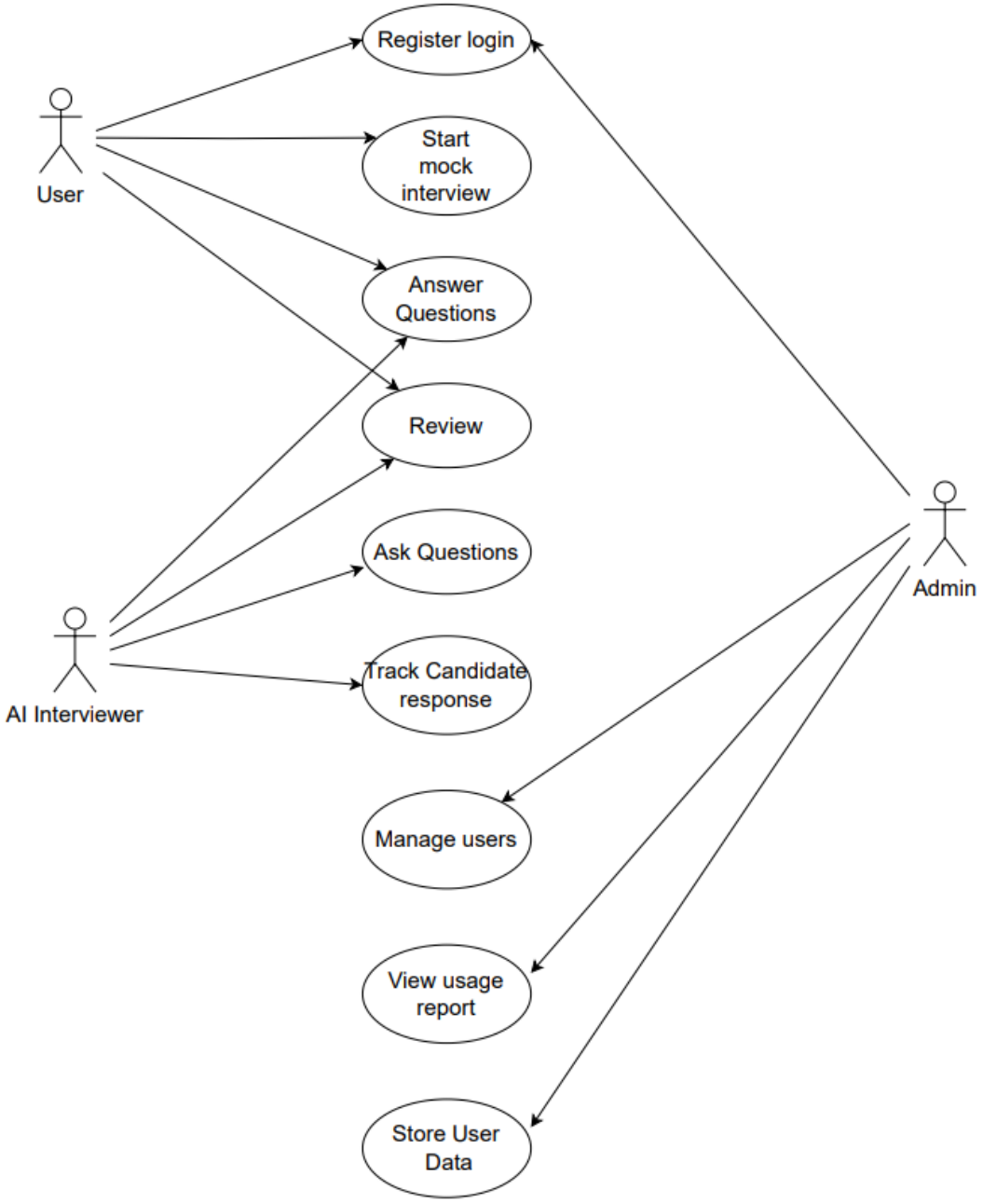

Figure 3: Use Case Diagram

CHAPTER 5

IMPLEMENTATION

5.1 PROGRAMMING LANGUAGES USED

This project primarily utilizes Python and JavaScript. Python's versatility, extensive libraries, and simplicity enable efficient development across various components, while JavaScript enhances interactivity and functionality, particularly in front-end development. By leveraging both languages, the project maintains a balance between backend efficiency and dynamic user experiences.

5.1.1 PYTHON

Python serves as the backbone of this project, handling core logic, data processing, and backend functionalities. Known for its simplicity and readability, Python allows for rapid development while maintaining code efficiency. Key reasons for choosing Python include:

- **Versatility** – Python supports a wide range of applications, from web development and automation to machine learning and data analysis.
- **Extensive Libraries** – Libraries such as NumPy, Pandas, Flask/Django, and TensorFlow simplify complex tasks, reducing development time.
- **Cross-Platform Compatibility** – Python runs seamlessly across different operating systems, ensuring flexibility.
- **Strong Community Support** – A large and active developer community provides continuous updates, documentation, and troubleshooting support.

In this project, Python is primarily used for:

- Backend development (handling server-side logic and API interactions)
- Data processing and analysis

- Automation and scripting
- Integration with databases and external services

By focusing on Python, the project ensures maintainability, scalability, and efficient collaboration within a unified programming environment.

5.1.2 JavaScript

JavaScript is used to enhance the frontend and interactivity of the web-based platform. With Node.js on the server-side and frontend frameworks (likely React or vanilla JS), JavaScript ensures a **seamless user experience**, enabling real-time updates, dynamic UI components, and integration with AI avatars for realistic interview simulations.

In this project, JavaScrpit is primarily used for:

- Frontend Development (User Interface & Experience)
- WebSockets for Real-Time Features
- AI Avatar & Interview Simulation
- Backend & Server-Side Integration

By leveraging JavaScript alongside Python, ASPIRA ensures a smooth, responsive, and AI-powered mock interview experience.

5.2 LIBRARIES

5.2.1 OPEN CV

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library, designed to facilitate the development of applications in the field of computer vision. It offers a comprehensive suite of functions and algorithms for image and video processing, enabling developers to perform a wide range of tasks, from simple image manipulation to complex object detection and recognition.

5.2.2 PyTorch

PyTorch, developed by Meta AI, is a flexible deep learning framework known for its dynamic computation graph and efficient GPU acceleration. It includes modules for neural

networks, optimization, and data handling, with extensions like torchvision for images and torchaudio for speech. Widely used in AI fields like computer vision and NLP, PyTorch offers seamless interoperability, easy debugging, and strong community support, making it ideal for research and industry applications

5.2.3 Transformes (from huggingface)

The Transformers library from Hugging Face plays a pivotal role in the ASPIRA project by enabling advanced Natural Language Processing (NLP) capabilities. It helps generate dynamic and context-aware interview questions, analyze candidates' responses, and assess sentiment, confidence, and personality traits. Models like GPT-based architectures facilitate real-time follow-up question generation, while BERT and RoBERTa assist in semantic analysis. Additionally, sentiment analysis models help gauge candidates' emotions and communication effectiveness. The integration of speech-to-text (STT) and text-to-speech (TTS) further enhances ASPIRA's interactive experience. By leveraging these models, the platform ensures adaptive and personalized mock interviews, improving users' readiness for real-world job opportunities

5.2.4 DuckDuckGo_Search

DuckDuckGo_Search is a Python library that enables private and efficient searches using DuckDuckGo. It supports web, image, and news searches, along with autocomplete suggestions, without requiring an API key. The library ensures user privacy by not tracking activity and provides structured search results for easy integration. While limited to DuckDuckGo's database, it offers fast and anonymous search capabilities, making it ideal for developers prioritizing security and simplicity.

5.2.5 Beutifull Soup

Beautiful Soup is a Python library for web scraping, used to extract data from HTML and XML documents. It simplifies parsing, navigating, and modifying web content using different parsers like html.parser and lxml. With an intuitive syntax, it efficiently extracts elements, attributes, and text, making it ideal for data collection and automation. When used with requests, it enables seamless web page retrieval and processing. However, ethical considerations and website terms of service should always be followed when scraping data.

5.2.6 Pyaudio

PyAudio is a Python library that enables real-time audio recording and playback using the PortAudio API. It is commonly used for speech recognition, text-to-speech, and voice analysis. In the ASPIRA project, PyAudio helps capture user responses, process speech-to-text (STT), and play AI-generated interviewer voices using text-to-speech (TTS), ensuring a seamless and interactive interview experience.

5.2.7 Librosa

Librosa is essential for ASPIRA's AI-driven mock interviews, enabling speech analysis through feature extraction like MFCCs, pitch, and intensity. It aids in preprocessing audio by normalizing, resampling, and reducing noise, ensuring cleaner speech input for STT and evaluation. By analyzing voice patterns, it helps assess communication skills, confidence, and stress, enhancing AI-driven personality and emotional analysis. Integrating Librosa improves ASPIRA's accuracy in evaluating candidates' spoken responses.

5.2.8 Batch Face

BatchFace is a high-performance Python library for batch face recognition, built on InsightFace and face_recognition. It supports fast face detection, alignment, and embedding generation using models like ArcFace. Optimized for GPU acceleration and multi-GPU support, it enables efficient face matching and verification. With RetinaFace and MTCNN for detection, it excels in security, biometrics, and automated photo management. Easy to install and scalable, BatchFace offers reliable face recognition for real-world applications.

5.3 PLATFORM USED

5.3.1 VISUAL STUDIO CODE

Visual Studio Code (VS Code) is an advanced source code editor developed by Microsoft, offering a comprehensive set of tools and features tailored to meet the demands of modern software development. It provides a unified platform for writing, editing, and debugging code across various programming languages and frameworks, seamlessly bridging the gap between different operating systems with its cross-platform

compatibility. One of the key strengths of VS Code lies in its extensive language support, boasting built-in functionality for popular languages like JavaScript, Python, Java, and many others. Its versatility is further enhanced by a vibrant ecosystem of extensions, allowing developers to customize their coding environment and integrate additional language support and tools as needed. At the core of VS Code's appeal is its intuitive user interface and lightweight design, which offers a seamless and responsive experience even when handling large codebases. Its integrated development environment (IDE) features, such as syntax highlighting, code completion, and integrated Git support, streamline the coding workflow and boost productivity.

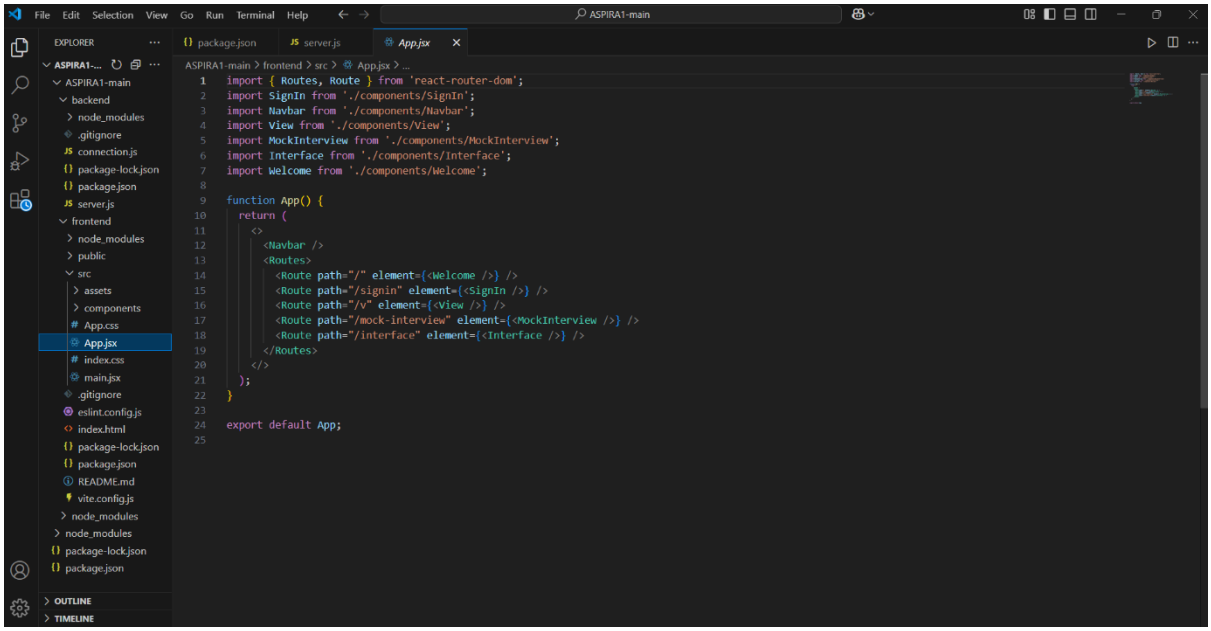

Figure 4: Visual Studio Code

5.4 FILE STRUCTURE

The file structure of the project has been designed to better organize the code files and resources. The structure is as follows:

```
.
├── Parser.py
├── README.md
├── Search_Engine.py
├── Summaraizer.py
├── Wav2Lip
├── __pycache__
├── aspira.py
├── chatbot.py
├── content
├── env_to_req.py
├── environment.yml
├── file
├── keyword_generator.py
├── recorder.py
├── requirements.txt
├── ret.txt
├── speaker.py
└── video.py
file
├── 1.txt
├── 2.txt
├── corpora
├── image.png
├── jobs.txt
├── mona_clip.mp4
├── qa.csv
├── summary.txt
├── temp_audio.mp3
├── test.wav
├── text_to_speech.mp3
└── tokenizers

8 directories, 24 files
```

5.5 SOURCE CODE

5.5.1 BACK END

aspira.py

```
from recorder import speech

from keyword_generator import extract

from Search_Engine import search

from Parser import Parse

from chatbot import chatbot

from Summaraizer import split_text_into_chunks,summrize

from speaker import convert

from sentence_transformers import SentenceTransformer

from sklearn.metrics.pairwise import cosine_similarity

import numpy as np

# Initialize the model

import time

from flask import Flask, jsonify

import os

import csv

# Load the data back into a dictionary

# loaded_data = {}

# with open('keywords.csv', 'r') as file:

#     reader = csv.reader(file)

#     for row in reader:

#         key = row[0]
```

```
#         # If the value consists of more than one element, convert it to a tuple
#         if len(row) > 2:
#             value = tuple(row[1:])
#         else:
#             value = row[1]
#         loaded_data[key] = value
# # Print the loaded dictionary
# print(loaded_data)
def read_last_row(file_path):
# Check if the file exists and is not empty
    if os.path.exists(file_path) and os.path.getsize(file_path) > 0:
        with open(file_path, newline='') as csvfile:
            reader = csv.reader(csvfile)
            # Read all rows and store them in a list
            rows = list(reader)
            # Get the last row
            last_row = rows[-1]
            return last_row
    return None
# app = Flask(__name__)
# start_time=time.perf_counter()
# model = SentenceTransformer('all-MiniLM-L6-v2')
# def compute_similarity(sentence1, sentence2):
#     embeddings1 = model.encode([sentence1])
```

```
#     embeddings2 = model.encode([sentence2])
#     similarity_score = cosine_similarity(embeddings1, embeddings2)[0][0]
#     return similarity_score
def aspira():
   #
   #while(timer!=0):
      # timer-=1
      q={
      kw={}
         text = '''Machine learning teaches computers to recognize patterns and make decisions automatically using data and algorithms.
      It can be broadly categorized into three types:
     Supervised Learning: Trains models on labeled data to predict or classify new, unseen data.
     Unsupervised Learning: Finds patterns or groups in unlabeled data, like clustering or dimensionality reduction.
      Reinforcement Learning: Learns through trial and error to maximize rewards, ideal for decision-making tasks.
       In addition these categories, there are also Semi-Supervised Learning and Self-Supervised Learning.
     Semi-Supervised Learning uses a mix of labeled and unlabeled data, making it helpful when labeling data is costly or time-consuming.
      Self-Supervised Learning creates its own labels from raw data, allowing it to learn patterns without needing labeled examples.
      Machine Learning Pipeline
```

Machine learning is fundamentally built upon data, which serves as the foundation for training and testing models. Data consists of inputs (features) and outputs (labels). A model learns patterns during training and is tested on unseen data to evaluate its performance and generalization. In order to make predictions, there are essential steps through which data passes in order to produce a machine learning model that can make predictions.'''

```
question ="what is machine learning"
#or question= "what subjects do you like?"
#convert(question)
# text='i very much like machine learning'
last_row = read_last_row('file/qa.csv')
if last_row:
   value1, value2 = last_row  # Unpack the two values into separate variables
   print(value1, value2)
else:
   print("The file is empty or does not exist.")
#text='Not found'
while(text == "Not found"):
   text=speech()
   if text =="stop":
      break
keywords=extract(text)
#if len(list(keywords.values)) < 2 and all(x < 8 for x in list(keywords.values)):
#   convert('ok')
for  key , score in keywords.items():
```

```
        sm=compute_similarity('machine learning',key)
        kw[key] = (score,sm)
        #value = my_dict.pop('b')
    first_elements = [v[0] for v in kw.values()]
    second_elements = [v[1] for v in kw.values()]
     first_threshold = sorted(first_elements)[len(first_elements) // 2]  # Middle of first elements
    second_threshold = sorted(second_elements)[len(second_elements) // 2]  # Middle of second elements
    # Print the thresholds for clarity
    print(f"First threshold: {first_threshold}")
    print(f"Second threshold: {second_threshold}")
    sorted_scores = dict(sorted(kw.items(), key=lambda x: (
    not (x[1][0] <= first_threshold and x[1][1] <= second_threshold),
        x[1][1],  #the first element
        x[1][0]  #second element
    ), reverse=True))
    for  key , score in sorted_scores.items():
        f_score = (f"{score[0]:.2f}", f"{score[1]:.2f}")
        print(f"{f_score}:{key}")
    kw=sorted_scores
    count1=3
    for  key , score in sorted_scores.items() :
        if score[0] >= 4 and len(key.split(' ')) < 4   :
            count1 -=1
```

```
        if (count1==0):
            break
        links =search(key,no=2)
        for  no , link in enumerate(links,1):
            text =Parse(link)
            if text == 'skip' or text is None:
                continue
            # with open(f'file/{no}.txt','w') as file:
            #     file.writelines(text)
            chunks = split_text_into_chunks(text, max_tokens=200)
            final_summary = "{}{}{}".join(chunks)
            # with open('file/summary.txt','w') as file:
            #     file.writelines(final_summary)
            print(len(chunks))
            count=0
            for i in range(min(len(chunks),5)):
                l=chatbot(chunks[i])
                while(count!=3):
                    #convert(l)
                    count+=1
                score = compute_similarity(question, l)
                q[l]=score
    sorted_qa = dict(sorted(q.items(), key=lambda x: x[1], reverse=True))
    values = list(sorted_qa.values())
```

```
        # mean_value = sum(values) / len(values)
        centre_value=np.mean(values)
        closest_key = min(sorted_qa, key=lambda k: abs(sorted_qa[k] - centre_value))
        print('Selected Question')
        print('-'*50)
        print(f"{sorted_qa[closest_key]:.2f}:{closest_key}")
        question=closest_key
        print(time.perf_counter()-start_time)
        convert(question)
        print(time.perf_counter()-start_time)
        with open('file/qa.csv', 'a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow([question, text])
        return 'stop'
# @app.route('/run-function', methods=['GET'])
# def run_function():
#     result = aspira()
#     return jsonify({'result': result})
# if __name__ == '__main__':
#     # app.run(debug=True)
aspira()
```

chatbot.py

```
#from transformers import GPT2LMHeadModel, GPT2Tokenizer
import torch
```

```
# tokenizer = BartTokenizer.from_pretrained("facebook/bart-large")
# model = BartForConditionalGeneration.from_pretrained("facebook/bart-large" )
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
import time
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
tokenizer = AutoTokenizer.from_pretrained("p208p2002/bart-squad-qg-hl")
model = AutoModelForSeq2SeqLM.from_pretrained("p208p2002/bart-squad-qg-hl")
# tokenizer = AutoTokenizer.from_pretrained("p208p2002/t5-squad-qg-hl")
# model = AutoModelForSeq2SeqLM.from_pretrained("p208p2002/t5-squad-qg-hl")
# model=torch.jit.script(model).to_device(device)
model.eval()
def chatbot(input_text):
   start_time=time.time()
   input_ids = tokenizer.encode(input_text, return_tensors="pt").to(device)
   with torch.no_grad():
      output_ids = model.generate(input_ids, max_length=100, num_return_sequences=1,
no_repeat_ngram_size=2, top_p=0.95, top_k=60, temperature=0.7,do_sample=True)
   generated_text = tokenizer.decode(output_ids[0], skip_special_tokens=True)
   print(generated_text)
   print(time.time()-start_time)
   return generated_text
```

env_to_req.py

```
import ruamel.yaml
yaml = ruamel.yaml.YAML()
```

```
data = yaml.load(open('environment.yml'))
requirements = []
for dep in data['dependencies']:
    if isinstance(dep, str):
        package, package_version, python_version = dep.split('=')
        if python_version == '0':
            continue
        requirements.append(package + '==' + package_version)
    elif isinstance(dep, dict):
        for preq in dep.get('pip', []):
            requirements.append(preq)
with open('requirements.txt', 'w') as fp:
    for requirement in requirements:
       print(requirement, file=fp)
```

keyword_generator.py

```
import nltk
from rake_nltk import Rake
from transformers import pipeline
import time
import fitz  # PyMuPDF
from keybert import KeyBERT
nltk.download('stopwords',download_dir='file')
nltk.download('punkt',download_dir='file')
# nltk.download('punkt_tab',download_dir='~/Developer/Ai_interviewer')
```

```
nltk.download('punkt_tab')
r = Rake()
a={}
with open("file/jobs.txt", "r") as file:
    job_list = {line.strip().lower() for line in file if line.strip()}  # Store jobs in a set
def is_job(text):
    text_lower = text.lower()
    return any(job in text_lower for job in job_list)
# from textblob import TextBlob
# def get_sentiment_score(text):
#     # Create a TextBlob object
#     blob = TextBlob(text)
#     # Get the sentiment polarity score (ranges from -1 to 1)
#     sentiment_score = blob.sentiment.polarity
# Proceed with extracting keywords from the uploaded PDF
def extract_text_from_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    text = ""
    for page_num in range(doc.page_count):
        page = doc.load_page(page_num)
        text += page.get_text("text")
    return text
def extract_keywords_from_pdf(pdf_path, num_keywords=10):
    resume_text = extract_text_from_pdf(pdf_path)
```

```
    kw_model = KeyBERT()
                keywords_with_scores = kw_model.extract_keywords(resume_text, keyphrase_ngram_range=(1, 2), top_n=num_keywords)
    return keywords_with_scores
def extract(text):
    start_time=time.time()
    r.extract_keywords_from_text(text)    # Extract keywords from the text
    keywords = r.get_ranked_phrases_with_scores()    # Get the ranked keywords
    print('\n')
    print('Keywords')
    for score, kw in keywords:
        if is_job(kw) :
            continue
        a[kw] =a.get(kw,0) + score
    #print("\n".join(f'{score:.2f}: {kw}' for kw, score in a.items()))
#     classifier = pipeline("zero-shot-classification")
#     output=classifier(
#     "machine learning",
#     candidate_labels=list(a.keys()),
# )
    # print(output)
     sorted_scores = dict(sorted(a.items(), key=lambda x: x[1], reverse=True))# Sort by value in descending order
    print(time.time()-start_time)
    return sorted_scores
```

parser.py

```
from bs4 import BeautifulSoup
import requests
import socket
import time
included_domain=['wikipedia','geekforgeek']
def Parse(url):
   start_time=time.time()
   if any(domain.lower() in url.lower() for domain in included_domain):
# Function to extract content from the page based on site structure
          headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'}
      response = requests.get(url, headers=headers)
      # Check if the request was successful
      if response.status_code == 200:
         # Parse the content of the webpage with BeautifulSoup
         soup = BeautifulSoup(response.text, 'html.parser')
         # Remove unwanted tags (like headers, script, style, etc.)
         for unwanted_tag in soup(['style', 'script', 'link', 'button', 'header', 'footer', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6']):
            unwanted_tag.decompose()
         # Initialize variable to store the content
         important_content = ""
          # Extract content for Wikipedia pages, typically found in the mw-parser-output class
```

```
        if "wikipedia" in url:
            content_div = soup.find('div', {'class': 'mw-parser-output'})
            if content_div:
                paragraphs = content_div.find_all('p')
                for p in paragraphs:
                    # Get the text of each paragraph, clean up, and join them into meaningful
content
                    paragraph_text = p.get_text(separator=" ", strip=True)
                    if paragraph_text:  # Only add non-empty paragraphs
                        important_content += paragraph_text + "\n\n"
        # If no content is found, fallback to extracting text from all <p> tags
        if not important_content:
            paragraphs = soup.find_all('p')
            for p in paragraphs:
                paragraph_text = p.get_text(separator=" ", strip=True)
                if paragraph_text:
                    important_content += paragraph_text + "\n\n"
        # Return the content with proper formatting
        return important_content.strip()
    else:
        return f"Failed to retrieve webpage. Status code: {response.status_code}"


try:
    response = requests.get(url)
```

```
        if response.status_code == 200:
            soup = BeautifulSoup(response.content, 'html.parser')
            # Remove header, footer, and other unwanted sections
            for element in soup(['header', 'footer', 'nav', 'aside']):
                element.decompose()  # This removes the element completely
            main_content = soup.get_text(separator=' ', strip=True)
            print(time.time()-start_time)
            return(main_content)
        else:
            print(f"Failed to retrieve the page. Status code: {response.status_code}")
    except socket.gaierror as e:
        print(f"Name resolution error occurred: {e}")
        return 'skipw'
```

recorder.py

```
import speech_recognition as sr
import time
def speech():
    r = sr.Recognizer()
    start_time = time.time()
    with sr.Microphone() as source:
        print("Talk")
        audio= r.listen(source,timeout=10,phrase_time_limit=50)
        print("Time over, thanks")
```

```
        #recognizer.energy_threshold = 4000  # Increase this value if there's too much background noise
        try:
            audio_text=r.recognize_google(audio)
            # using google speech recognition
            print(f"Text: {audio_text}")
            end_time = time.time()
            print(f"F-string time: {end_time - start_time} seconds")
            return audio_text
        except:
            print("Sorry, I did not get that")
            return("Not found")
# import pyaudio
# import wave
# import numpy as np
# import time
# def record_audio_until_silence(output_filename, silence_threshold=1000, silence_duration=3, rate=44100, channels=1, chunk_size=1024):
#     """
#     Records audio from the microphone and stops when silence is detected for a specified duration.
#     Parameters:
#     - output_filename (str): The name of the output file (e.g., 'output.wav').
#     - silence_threshold (int): The threshold below which sound is considered silence. Default is 1000.
```

```
#    - silence_duration (int): The duration (in seconds) of silence to detect before stopping the recording. Default is 3 seconds.
#    - rate (int): The sample rate (e.g., 44100 Hz).
#    - channels (int): Number of audio channels (1 for mono, 2 for stereo).
#    - chunk_size (int): The size of each chunk of audio data.
#    Returns:
#    None
#    """
#    # Initialize the PyAudio object
#    p = pyaudio.PyAudio()
#    # Open the stream for audio input
#    stream = p.open(format=pyaudio.paInt16,  # Audio format (16-bit PCM)
#                    channels=channels,       # Number of audio channels
#                    rate=rate,               # Sample rate
#                    input=True,              # Input stream (microphone)
#                    frames_per_buffer=chunk_size)  # Number of frames per buffer
#    print("Recording...")
#    frames = []
#    silent_chunks = 0  # Count consecutive silent chunks

#    while True:
#        # Read data from the stream
#        data = stream.read(chunk_size)
#        frames.append(data)
```

```
#       # Convert the data to numpy array for analysis
#       audio_data = np.frombuffer(data, dtype=np.int16)
#       # Calculate the amplitude (volume) of the audio signal
#           amplitude = np.linalg.norm(audio_data)  # Euclidean norm (L2 norm) of the signal
#          # If the amplitude is below the silence threshold, increment the silent_chunks counter
#       if amplitude < silence_threshold:
#           silent_chunks += 1
#       else:
#           silent_chunks = 0  # Reset if sound is detected
#       # If silence lasts for the specified duration, stop recording
#       if silent_chunks > silence_duration * (rate / chunk_size):
#           print("Silence detected, stopping recording.")
#           break
#    # Stop and close the stream
#    stream.stop_stream()
#    stream.close()
#    p.terminate()

#    # Save the recorded audio as a .wav file
#    with wave.open(output_filename, 'wb') as wf:
#       wf.setnchannels(channels)      # Set number of channels
#        wf.setsampwidth(p.get_sample_size(pyaudio.paInt16))  # Sample width (2 bytes for 16-bit)
```

```
#        wf.setframerate(rate)          # Set the sample rate (Hz)
#        wf.writeframes(b''.join(frames))  # Write audio data to the file
#    print(f"Audio saved to {output_filename}")
# # Example usage
# if __name__ == '__main__':
#                record_audio_until_silence("recorded_audio.wav",     silence_threshold=1000,
silence_duration=3)  # Stops after 3 seconds of silence
```

Search_Engine.py

```
import time
from duckduckgo_search import DDGS
from duckduckgo_search.exceptions import DuckDuckGoSearchException
a = []
exclude_domains = ['reddit', 'coursera']
include_domains = ['wikipedia']#'interview questions'
exclude_title = ['course', 'tutorial']
instance = DDGS()
def search(search_query='Machine Learning', no=2):
   start_time = time.time()
   try:
      results = instance.text(
         keywords=search_query,
         safesearch='off',
         timelimit='7d',
         max_results=no
```

```
)
for i in include_domains:
    try:
        name = search_query + ' ' + i
        I = instance.text(
            keywords=name,
            safesearch='off',
            timelimit='7d',
            max_results=1
        )
        # Add results to the main results list (make sure it's a list of dictionaries)
        if isinstance(I, list):
            results += I # Assuming `I` is a list of results
    except Exception as e:
        print(f"Error with domain {i}: {e}")
# Loop through the results and apply filters
for idx, item in enumerate(results, 1):
    if isinstance(item, dict) and 'href' in item:
        if any(domain in item['href'] for domain in exclude_domains):
            continue
        if any(domain.lower() in item['title'].lower() for domain in exclude_title):
            continue
        print(f"{idx}. {item['title']}")
        print(f"Link: {item['href']}")
```

```
            print('-' * 50)  # Separator between results
            a.append(item['href'])
    except DuckDuckGoSearchException as e:
        print(f"Error: {e}")
        if "Ratelimit" in str(e):
            print("Rate limit exceeded. Breaking the search process.")
        print("Retry...")
    print(time.time() - start_time)
    return a
if __name__ == "__main__":
    search_results = search('Machine Learning', no=2)
```

speaker.py

```
from gtts import gTTS
import os
from pydub import AudioSegment
import time
def convert(text):
    start_time=time.time()
    mp3_path = "file/temp_audio.mp3"
    output_wav_path="Wav2Lip/filelists/test.wav"
    tts = gTTS(text=text, lang='en', slow=False)  # slow=False for faster speech
    tts.save(mp3_path)
    # Convert mp3 to wav using pydub
    audio = AudioSegment.from_mp3(mp3_path)
```

```
    audio.export(output_wav_path, format="wav")
    print(time.time()-start_time)
        os.system('''cd Wav2Lip && python3 inference.py --checkpoint_path checkpoints/wav2lip_gan.pth --face "filelists/mona_clip.mp4" --audio "filelists/test.wav"''')
    os.system("ffplay Wav2Lip/results/result_voice.mp4")
```

Summaraizer.py

```
from transformers import pipeline
import math
def split_text_into_chunks(text, max_tokens=1024):
    # Approximate word tokenization: 1 token ≈ 3/4 of a word
    words = text.split()
    max_words_per_chunk = max_tokens * 3 // 4
        chunks = [words[i:i+max_words_per_chunk] for i in range(0, len(words), max_words_per_chunk)]
    return [" ".join(chunk) for chunk in chunks]
def summrize(text,max_tokens=200):
    # Initialize summarizer
    summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
    # Function to split text into chunks



    # # Sample long text
    # long_text = """
```

Artificial intelligence (AI) is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and animals. Leading AI textbooks define the field as the study of "intelligent agents": any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals. Colloquially, the term "artificial intelligence" is often used to describe machines (or computers) that mimic "cognitive" functions that humans associate with the human mind, such as "learning" and "problem solving." ...

""" # Make sure this text is long enough to exceed the token limit

Split the text into smaller chunks

chunks = split_text_into_chunks(text, max_tokens)

Summarize each chunk

summaries = [summarizer(chunk, max_length=len(chunk.split())//2, min_length=len(chunk.split())//4, do_sample=False)[0]['summary_text'] for chunk in chunks]

Combine the individual summaries into one

print('f')

return summaries

video.py

```
from IPython.display import HTML, Audio

# from google.colab.output import eval_js

from base64 import b64decode

import numpy as np

from scipy.io.wavfile import read as wav_read

import io

import ffmpeg

# from ghc.l_ghc_cf import l_ghc_cf
```

```
def get_audio():
  # display(HTML(AUDIO_HTML))
  data = eval_js("data")
  binary = b64decode(data.split(',')[1])
  process = (ffmpeg
    .input('pipe:0')
    .output('pipe:1', format='wav')
        .run_async(pipe_stdin=True, pipe_stdout=True, pipe_stderr=True, quiet=True, overwrite_output=True)
  )
  output, err = process.communicate(input=binary)
  riff_chunk_size = len(output) - 8
  # Break up the chunk size into four bytes, held in b.
  q = riff_chunk_size
  b = []
  for i in range(4):
     q, r = divmod(q, 256)
     b.append(r)
  # Replace bytes 4:8 in proc.stdout with the actual size of the RIFF chunk.
  riff = output[:4] + bytes(b) + output[8:]

  sr, audio = wav_read(io.BytesIO(riff))
  return audio, sr
from base64 import b64encode
```

```
# def showVideo(path):
#   mp4 = open(str(path),'rb').read()
#   data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
#   return HTML("""
#   <video width=700 controls>
#       <source src="%s" type="video/mp4">
#   </video>
#   """ % data_url)
print("All set and ready!")
import shutil
import os
# from google.colab import files
# from IPython.display import HTML, clear_output
# from base64 import b64encode
import moviepy.editor as mp
# def showVideo(file_path):
#    """Function to display video in Colab"""
#    mp4 = open(file_path,'rb').read()
#    data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
#    # display(HTML("""
#    # <video controls width=600>
#    #    <source src="%s" type="video/mp4">
#    # </video>
#    # """ % data_url))
```

```
def get_video_resolution(video_path):
    """Function to get the resolution of a video"""
    import cv2
    video = cv2.VideoCapture(video_path)
    width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
    return (width, height)
def resize_video(video_path, new_resolution):
    """Function to resize a video"""
    import cv2
    video = cv2.VideoCapture(video_path)
    fourcc = int(video.get(cv2.CAP_PROP_FOURCC))
    fps = video.get(cv2.CAP_PROP_FPS)
    width, height = new_resolution
    output_path = os.path.splitext(video_path)[0] + '_720p.mp4'
    writer = cv2.VideoWriter(output_path, fourcc, fps, (width, height))
    while True:
        success, frame = video.read()
        if not success:
            break
        resized_frame = cv2.resize(frame, new_resolution)
        writer.write(resized_frame)
    video.release()
    writer.release()
```

```
upload_method = "Custom Path"
# remove previous input video
if os.path.isfile('content/sample_data/input_vid.mp4'):
  os.remove('content/sample_data/input_vid.mp4')
if upload_method == "Upload":
  uploaded = files.upload()
  for filename in uploaded.keys():
    os.rename(filename, 'content/sample_data/input_vid.mp4')
  PATH_TO_YOUR_VIDEO = 'content/sample_data/input_vid.mp4'
elif upload_method == 'Custom Path':
  PATH_TO_YOUR_VIDEO = 'content/test.mp4' #@param {type:"string"}
  if not os.path.isfile(PATH_TO_YOUR_VIDEO):
    print("ERROR: File not found!")
    raise SystemExit(0)
PATH_TO_YOUR_VIDEO = 'content/test.mp4' #@param {type:"string"}
video_duration = mp.VideoFileClip(PATH_TO_YOUR_VIDEO).duration
if video_duration > 60:
  print("WARNING: Video duration exceeds 60 seconds. Please upload a shorter video.")
  raise SystemExit(0)
video_resolution = get_video_resolution(PATH_TO_YOUR_VIDEO)
print(f"Video resolution: {video_resolution}")
if video_resolution[0] >= 1920 or video_resolution[1] >= 1080:
  print("Resizing video to 720p...")
```

```
            os.system(f"ffmpeg -i {PATH_TO_YOUR_VIDEO} -vf scale=1280:720 content/sample_data/input_vid.mp4")
    PATH_TO_YOUR_VIDEO = "content/sample_data/input_vid.mp4"
    print("Video resized to 720p")
else:
    print("No resizing needed")
if os.path.isfile(PATH_TO_YOUR_VIDEO):
    # Check if the source and destination files are the same
    if PATH_TO_YOUR_VIDEO != "content/sample_data/input_vid.mp4":
        shutil.copyfile(PATH_TO_YOUR_VIDEO, "content/sample_data/input_vid.mp4")
        print("Video copied to destination.")
        print("Input Video")
#@title STEP3: Select Audio (Record, Upload from local drive or Gdrive)
import os
from IPython.display import Audio
from IPython.core.display import display
#remove previous input audio
if os.path.isfile('content/sample_data/input_audio.wav'):
    os.remove('content/sample_data/input_audio.wav'
def displayAudio():
  display(Audio('content/sample_data/input_audio.wav'))
# if upload_method == 'Record':
#   audio, sr = get_audio()
#   import scipy
```

```
#   scipy.io.wavfile.write('content/sample_data/input_audio.wav', sr, audio)
# elif upload_method == 'Upload':
#   from google.colab import files
#   uploaded = files.upload()
#   for fn in uploaded.keys():
#     print('User uploaded file "{name}" with length {length} bytes.'.format(
#         name=fn, length=len(uploaded[fn])))
#   # Consider only the first file
#   PATH_TO_YOUR_AUDIO = str(list(uploaded.keys())[0])
#   # Load audio with specified sampling rate
#   import librosa
#   audio, sr = librosa.load(PATH_TO_YOUR_AUDIO, sr=None)
#   # Save audio with specified sampling rate
#   import soundfile as sf
#   sf.write('content/sample_data/input_audio.wav', audio, sr, format='wav')
#   clear_output()
#   displayAudio()
# else: # Custom Path
#   from google.colab import drive
#   drive.mount('content/drive')
  #@markdown ``Add the full path to your audio on your Gdrive`` 👇
PATH_TO_YOUR_AUDIO = 'file/test.wav' #@param {type:"string"}


# Load audio with specified sampling rate
```

```
import librosa

audio, sr = librosa.load(PATH_TO_YOUR_AUDIO, sr=None)

# Save audio with specified sampling rate

import soundfile as sf

sf.write('content/sample_data/input_audio.wav', audio, sr, format='wav')

#@title STEP4: Start Crunching and Preview Output

#@markdown <b>Note: Only change these, if you have to</b>

# Set up paths and variables for the output file

output_file_path = 'content/Wav2Lip/results/result_voice.mp4'

# Delete existing output file before processing, if any

if os.path.exists(output_file_path):

   os.remove(output_file_path)

pad_top =  0#@param {type:"integer"}

pad_bottom =  10#@param {type:"integer"}

pad_left =  0#@param {type:"integer"}

pad_right =  0#@param {type:"integer"}

rescaleFactor =  1#@param {type:"integer"}

nosmooth = True #@param {type:"boolean"}

#@markdown ___

#@markdown Model selection:

use_hd_model = True #@param {type:"boolean"}

checkpoint_path = 'Wav2Lip/checkpoints/wav2lip.pth' if not use_hd_model else 'Wav2Lip/checkpoints/wav2lip_gan.pth'

if nosmooth == False:
```

```
 os.system(f'''python3 Wav2Lip/inference.py --checkpoint_path {checkpoint_path} --face "content/sample_data/input_vid.mp4"  --audio  "content/sample_data/input_audio.wav" --pads {pad_top} {pad_bottom} {pad_left} {pad_right} --resize_factor {rescaleFactor}''')

else:

 os.system(f'''python3 Wav2Lip/inference.py --checkpoint_path {checkpoint_path} --face "content/sample_data/input_vid.mp4"  --audio  "content/sample_data/input_audio.wav" --pads {pad_top} {pad_bottom} {pad_left} {pad_right} --resize_factor {rescaleFactor} --nosmooth''')

#Preview output video

if os.path.exists(output_file_path):

   print("Final Video Preview")

   print("Download this video from", output_file_path)

else:

   print("Processing failed. Output video not found.")
```

5.5.2 FRONT END

Interface.jsx

```
import React, { useState, useEffect, useRef } from "react";

import { Button, Container, Typography, TextField, Card, Box } from "@mui/material";

import { useNavigate } from "react-router-dom";

import { styled } from "@mui/material/styles";


// ========== Question Banks ==========

const getBehavioralQuestions = () => [

  "Tell me about a time you faced a difficult challenge at work.",

  "Describe a situation where you had to work with a difficult team member.",
```

```
  "Give an example of how you handled a mistake you made at work.",
  "Why do you want to work with the company.",
  "What do you think is better- being perfect and delivering late or being good and delivering on time."
];

const getTechnicalQuestions = () => [
  "What programming languages are you most familiar with.",
  "Describe a time when you had to learn a new technology or programming language quickly.",
  "Explain the concept of a binary search algorithm and its time complexity.",
  "What are the different types of HTTP request methods.",
  "How would you implement a responsive web design.",
];

const getSituationalQuestions = () => [
  "What would you do if you disagreed with your manager's decision?",
  "How would you handle a tight deadline with multiple priorities?",
  "Describe how you would onboard a new team member.",
  "Have you been faced with a difficult decision without having much information? What did you do.",
  "Can you share a time you had to deal with a difficult customer."
];

// ========== AI Response Generators ==========
```

```
const getEncouragingResponses = () => [
  "That's an excellent point! Could you elaborate further?",
  "Great answer! What other factors did you consider?",
  "Interesting perspective! How did this experience shape you?"
];

const getFollowUpResponses = () => [
  "What was the most challenging part of that situation?",
  "How would you approach this differently today?",
  "What key lessons did you learn from this experience?"
];

// ========== Speech Functions ==========
const initializeSpeech = () => {
  const synth = window.speechSynthesis;
  let recognition = null;

  if ('SpeechRecognition' in window || 'webkitSpeechRecognition' in window) {
    recognition = new (window.SpeechRecognition || window.webkitSpeechRecognition)();
    recognition.continuous = false;
    recognition.interimResults = false;
  }
```

```
  return { synth, recognition };
};

const speakQuestion = (synth, question) => {
  if (synth.speaking) {
    synth.cancel();
  }

  const utterance = new SpeechSynthesisUtterance(question);
  utterance.rate = 1.0;
  utterance.pitch = 1.0;
  synth.speak(utterance);
};

// ========== Styled Components ==========
const Background = styled(Box)({
  position: "fixed",
  top: 0,
  left: 0,
  width: "100%",
  height: "100%",
  backgroundImage:
`url('https://careercenter.ucdavis.edu/sites/g/files/dgvnsk15461/files/styles/sf_landscape_16x9/public/media/images/CC%E2%80%93Horizontal-Marketing-Block-2.jpg?h=0419be36&itok=Q4eP0cAr')`,
```

```
  backgroundSize: "cover",
  backgroundPosition: "center",
  zIndex: -1,
  "&::before": {
    content: '""',
    position: "absolute",
    top: 0,
    left: 0,
    width: "100%",
    height: "100%",
    backgroundColor: "rgba(0, 0, 0, 0.5)",
    zIndex: 1,
  },
});

const ContentWrapper = styled(Box)({
  position: "relative",
  zIndex: 2,
  minHeight: "100vh",
  display: "flex",
  alignItems: "center",
  justifyContent: "center",
  padding: "20px",
  overflow: "auto",
```

```
});

const AIVideoContainer = styled(Box)({
  position: 'relative',
  width: '100%',
  height: '300px',
  borderRadius: '10px',
  overflow: 'hidden',
  marginBottom: '20px',
  boxShadow: '0 4px 20px rgba(0, 198, 255, 0.3)',
  border: '2px solid rgba(0, 198, 255, 0.5)',
  backgroundColor: '#000',
  '& video': {
    width: '100%',
    height: '100%',
    objectFit: 'cover',
  },
});

const AISpeakingIndicator = styled(Box)({
  position: 'absolute',
  bottom: '20px',
  left: '50%',
  transform: 'translateX(-50%)',
```

```
  display: 'flex',
  gap: '5px',
  '& span': {
    display: 'inline-block',
    width: '8px',
    height: '8px',
    borderRadius: '50%',
    background: '#00c6ff',
    animation: 'bounce 1.4s infinite ease-in-out',
  },
  '& span:nth-of-type(1)': {
    animationDelay: '0s',
  },
  '& span:nth-of-type(2)': {
    animationDelay: '0.2s',
  },
  '& span:nth-of-type(3)': {
    animationDelay: '0.4s',
  },
  '@keyframes bounce': {
    '0%, 80%, 100%': { transform: 'scale(0)' },
    '40%': { transform: 'scale(1)' },
  },
});
```

```
// ========== Main Component ==========
const Interface = () => {
  const navigate = useNavigate();
  const [questionIndex, setQuestionIndex] = useState(0);
  const [answer, setAnswer] = useState("");
  const [aiResponse, setAiResponse] = useState("");
  const [isListening, setIsListening] = useState(false);
  const [isInterviewStarted, setIsInterviewStarted] = useState(false);
  const [questionBank, setQuestionBank] = useState([]);
  const [isSpeaking, setIsSpeaking] = useState(false);
  const synthRef = useRef(null);
  const recognitionRef = useRef(null);
  const videoRef = useRef(null);

  // Initialize speech and question bank
  useEffect(() => {
    const { synth, recognition } = initializeSpeech();
    synthRef.current = synth;
    recognitionRef.current = recognition;

    if (recognition) {
      recognition.onresult = (event) => {
        const transcript = event.results[0][0].transcript;
```

```
    setAnswer(prev => prev + ' ' + transcript);
    setIsListening(false);
   };

   recognition.onerror = (event) => {
    console.error('Speech recognition error:', event.error);
    setIsListening(false);
   };

   recognition.onend = () => {
    setIsListening(false);
   };
  } else {
   console.warn("Speech recognition is not supported in this browser.");
  }

  // Default to behavioral questions
  setQuestionBank(getBehavioralQuestions());

  // Set up speech synthesis events
  synth.onvoiceschanged = () => {
   console.log("Voices loaded:", synth.getVoices());
  };
```

```
  return () => {
    if (synth.speaking) {
      synth.cancel();
    }
    if (recognition) {
      recognition.abort();
    }
  };
}, []);

// Speak question when it changes
useEffect(() => {
  if (isInterviewStarted && questionBank.length > 0) {
    const question = questionBank[questionIndex];
    const utterance = new SpeechSynthesisUtterance(question);
    utterance.rate = 0.9;
    utterance.pitch = 1.0;

    utterance.onstart = () => {
      setIsSpeaking(true);
      if (videoRef.current) {
        videoRef.current.play();
      }
    };
```

```
    utterance.onend = () => {
      setIsSpeaking(false);
      if (videoRef.current) {
        videoRef.current.pause();
        videoRef.current.currentTime = 0;
      }
    };

    synthRef.current.speak(utterance);
  }
}, [questionIndex, isInterviewStarted, questionBank]);

// Generate AI response
useEffect(() => {
  if (answer) {
    setAiResponse("");
    setTimeout(() => {
      const responses = [...getEncouragingResponses(), ...getFollowUpResponses()];
      setAiResponse(responses[Math.floor(Math.random() * responses.length)]);
    }, 1500);
  }
}, [answer]);
```

```
// ========== Question Navigation ==========
const handleNextQuestion = () => {
  if (questionIndex < questionBank.length - 1) {
    setQuestionIndex(prev => prev + 1);
    setAnswer("");
    setAiResponse("");
  } else {
    synthRef.current.cancel();
    alert("AI Interview Completed! Redirecting...");
    navigate("/");
  }
};

const handlePreviousQuestion = () => {
  if (questionIndex > 0) {
    setQuestionIndex(prev => prev - 1);
    setAnswer("");
    setAiResponse("");
  }
};

// ========== Question Bank Selection ==========
const selectQuestionBank = (type) => {
  switch(type) {
```

```
    case 'technical':
      setQuestionBank(getTechnicalQuestions());
      break;
    case 'situational':
      setQuestionBank(getSituationalQuestions());
      break;
    default:
      setQuestionBank(getBehavioralQuestions());
  }
  setQuestionIndex(0);
  setAnswer("");
  setAiResponse("");
};

// ========== Voice Control ==========
const toggleVoiceInput = () => {
  if (!recognitionRef.current) {
    alert("Voice input is not supported in your browser");
    return;
  }

  if (isListening) {
    recognitionRef.current.stop();
  } else {
```

```
    recognitionRef.current.start();
  }
  setIsListening(!isListening);
};

const replayQuestion = () => {
  speakQuestion(synthRef.current, questionBank[questionIndex]);
};

return (
  <>
    <Background />
    <ContentWrapper>
      <Container maxWidth="sm">
        <Card sx={{
          p: 4,
          borderRadius: 3,
          boxShadow: 3,
          textAlign: "center",
          backgroundColor: "rgba(255, 255, 255, 0.1)",
          backdropFilter: "blur(10px)",
          color: "white",
        }}>
          {!isInterviewStarted ? (
```

```
        <>
          <Box sx={{ mb: 3 }}>
            <img
              src="https://images.cnbctv18.com/uploads/2023/05/artificial-intelligence-
ai.jpeg"
              alt="AI Interview"
              style={{
                width: "100%",
                borderRadius: "10px",
                maxHeight: "200px",
                objectFit: "cover",
              }}
            />
          </Box>

          <Typography variant="h5" sx={{ mb: 3 }}>
            Select Question Type
          </Typography>

          <Box sx={{ display: 'flex', flexDirection: 'column', gap: 2, mb: 3 }}>
            <Button
              variant="outlined"
              onClick={() => selectQuestionBank('behavioral')}
              sx={{ color: 'white', borderColor: 'white' }}
```

```
>
  Behavioral Questions
</Button>
<Button
  variant="outlined"
  onClick={() => selectQuestionBank('technical')}
  sx={{ color: 'white', borderColor: 'white' }}
>
  Technical Questions
</Button>
<Button
  variant="outlined"
  onClick={() => selectQuestionBank('situational')}
  sx={{ color: 'white', borderColor: 'white' }}
>
  Situational Questions
</Button>
</Box>

<Button
 variant="contained"
 onClick={() => setIsInterviewStarted(true)}
 sx={{
  backgroundColor: "#00c6ff",
```

```
            "&:hover": { backgroundColor: "#0072ff" },
            fontSize: '1.1rem',
            padding: '10px 24px'
          }}
        >
          Start Interview
        </Button>
      </>
    ) : (
      <>
        <AIVideoContainer>
          <video
            ref={videoRef}
            width="100%"
            height="100%"
            loop
            muted
            playsInline
            poster="https://media.istockphoto.com/id/1334436084/photo/3d-rendering-of-ai-artificial-intelligence-robot-avatar-with-digital-data-network-background.jpg?s=612x612&w=0&k=20&c=QfXx5j5D5zG3XJmQJwVvq6o6p3Q5Q5Q5Q5Q5Q5Q5Q5Q5Q="
          >
            <source src="https://assets.mixkit.co/videos/preview/mixkit-robot-humanoid-with-led-face-12230-large.mp4" type="video/mp4" />
```

```
    Your browser does not support the video tag.
  </video>
  {isSpeaking && (
    <AISpeakingIndicator>
      <span></span>
      <span></span>
      <span></span>
    </AISpeakingIndicator>
  )}
</AIVideoContainer>
<Typography variant="h6" sx={{ mt: 1, fontWeight: "bold" }}>
  AI Interviewer
</Typography>

<Box sx={{ display: 'flex', alignItems: 'center', gap: 1, mb: 2 }}>
  <Typography variant="h5" sx={{ fontWeight: "bold" }}>
    AI: {questionBank[questionIndex]}
  </Typography>
  <Button
    variant="outlined"
    onClick={replayQuestion}
    sx={{
      color: '#00c6ff',
      borderColor: '#00c6ff',
```

```
    '&:hover': { backgroundColor: '#00c6ff20' }
   }}
  >
   Replay
  </Button>
 </Box>

 <TextField
  fullWidth
  multiline
  rows={3}
  value={answer}
  onChange={(e) => setAnswer(e.target.value)}
  placeholder="Your response..."
  sx={{
   mb: 2,
   input: { color: "white" },
   "& .MuiOutlinedInput-root": {
    "& fieldset": { borderColor: "white" },
    "&:hover fieldset": { borderColor: "#66a3ff" },
   },
  }}
 />
```

```
<Button
 variant="contained"
 onClick={toggleVoiceInput}
 sx={{
  mb: 2,
  backgroundColor: isListening ? "#ff3b2f" : "#ff6f61",
  '&:hover': { backgroundColor: isListening ? "#ff2b1f" : "#ff5f51" }
 }}
>
 {isListening ? "Stop Recording" : "Start Voice Input"}
</Button>

{aiResponse && (
 <Typography variant="body1" sx={{ mb: 2, fontWeight: "bold", color: "lightgreen" }}>
  AI: {aiResponse}
 </Typography>
)}

<Box sx={{ display: "flex", justifyContent: "space-between", mt: 2 }}>
 <Button
  variant="outlined"
  onClick={handlePreviousQuestion}
  disabled={questionIndex === 0}
```

```
                sx={{ borderColor: "#00c6ff", color: "#00c6ff", "&:hover": { backgroundColor:
"#00c6ff", color: "white" } }}
              >
               Back
              </Button>
              <Button
               variant="contained"
               onClick={handleNextQuestion}
               sx={{ backgroundColor: "#00c6ff", "&:hover": { backgroundColor: "#0072ff" }
}}
              >
               {questionIndex < questionBank.length - 1 ? "Next Question" : "Finish"}
              </Button>
             </Box>
            </>
           )}
         </Card>
        </Container>
      </ContentWrapper>
    </>
  );
};

export default Interface;
```

MockInterview.jsx

```
import React, { useState, useEffect } from "react";
import { Button, Container, Typography, TextField, Card, Box } from "@mui/material";
import { useNavigate } from "react-router-dom";
import { styled } from "@mui/material/styles";

// Styled Components
const Background = styled(Box)({
  position: "fixed",
  top: 0,
  left: 0,
  width: "100%",
  height: "100%",
  backgroundImage:
`url('https://careercenter.ucdavis.edu/sites/g/files/dgvnsk15461/files/styles/sf_landscape_16x9/public/media/images/CC%E2%80%93Horizontal-Marketing-Block-2.jpg?h=0419be36&itok=Q4eP0cAr')`,
  backgroundSize: "cover",
  backgroundPosition: "center",
  backgroundRepeat: "no-repeat",
  zIndex: -1,
  "&::before": {
    content: '""',
    position: "absolute",
    top: 0,
```

```
    left: 0,
    width: "100%",
    height: "100%",
    backgroundColor: "rgba(0, 0, 0, 0.5)",
    zIndex: 1,
  },
});

const ContentWrapper = styled(Box)({
  position: "relative",
  zIndex: 2,
  minHeight: "100vh",
  display: "flex",
  alignItems: "center",
  justifyContent: "center",
  padding: "20px",
});

// Sample Interview Questions
const questions = [
  "Tell me about yourself.",
  "What are your strengths and weaknesses?",
  "Why do you want this job?",
  "Where do you see yourself in 5 years?",
```

```
  "Do you have any questions for us?",
];

const MockInterview = () => {
  const navigate = useNavigate();
  const [questionIndex, setQuestionIndex] = useState(0);
  const [answers, setAnswers] = useState(Array(questions.length).fill(""));
  const [timeLeft, setTimeLeft] = useState(180);

  // Authentication Check
  useEffect(() => {
    if (localStorage.getItem("isLoggedIn") !== "true") {
      navigate("/signin");
    }
  }, [navigate]);

  // Timer Logic
  useEffect(() => {
    if (timeLeft === 0) {
      alert("Time's up! Moving to the next question.");
      handleNextQuestion();
      return;
    }
```

```
  const timer = setInterval(() => {
    setTimeLeft((prev) => prev - 1);
  }, 1000);

  return () => clearInterval(timer);
}, [timeLeft, questionIndex]);

// Reset Timer
useEffect(() => {
  setTimeLeft(180);
}, [questionIndex]);

// Question Navigation
const handleNextQuestion = () => {
  if (questionIndex < questions.length - 1) {
    setQuestionIndex(questionIndex + 1);
  } else {
    alert("Interview Completed! Redirecting to AI Interface...");
    navigate("/interface");
  }
};

const handlePreviousQuestion = () => {
  if (questionIndex > 0) {
```

```
    setQuestionIndex(questionIndex - 1);
  }
};

// Answer Handling
const handleAnswerChange = (e) => {
  const updatedAnswers = [...answers];
  updatedAnswers[questionIndex] = e.target.value;
  setAnswers(updatedAnswers);
};

return (
  <>
    <Background />
    <ContentWrapper>
      <Container maxWidth="sm">
        <Card
          sx={{
            p: 4,
            backgroundColor: "rgba(255, 255, 255, 0.1)",
            borderRadius: 3,
            boxShadow: 3,
            backdropFilter: "blur(10px)",
            color: "white",
```

```
  textAlign: "center",
 }}
>
 {/* Progress and Timer */}
 <Typography variant="subtitle1" sx={{ mb: 2 }}>
  Question {questionIndex + 1} of {questions.length}
 </Typography>
 <Typography variant="subtitle1" sx={{ mb: 2 }}>
  Time Left: {Math.floor(timeLeft / 60)}:{String(timeLeft % 60).padStart(2, "0")}
 </Typography>

 {/* Question Display */}
 <Typography variant="h4" gutterBottom sx={{ fontWeight: "bold" }}>
  Mock Interview
 </Typography>
 <Typography variant="h6" sx={{ mb: 2 }}>
  {questions[questionIndex]}
 </Typography>

 {/* Answer Input */}
 <TextField
  fullWidth
  multiline
  rows={3}
```

```
  value={answers[questionIndex]}
  onChange={handleAnswerChange}
  placeholder="Type your response here..."
  sx={{
    mb: 2,
    "& .MuiInputBase-input": {
      color: "white",
      "&::placeholder": {
        color: "rgba(255, 255, 255, 0.7)"
      }
    },
    "& .MuiOutlinedInput-root": {
      "& fieldset": { borderColor: "white" },
      "&:hover fieldset": { borderColor: "#66a3ff" },
    },
  }}
/>

{/* Navigation Buttons */}
<Box sx={{ display: "flex", justifyContent: "space-between", mt: 2 }}>
  <Button
    variant="outlined"
    onClick={handlePreviousQuestion}
    disabled={questionIndex === 0}
```

```
            sx={{
              borderColor: "#00c6ff",
              color: "#00c6ff",
              "&:hover": { backgroundColor: "#00c6ff", color: "white" },
            }}
          >
            Back
          </Button>
          <Button
            variant="contained"
            onClick={handleNextQuestion}
            sx={{
              backgroundColor: "#00c6ff",
              "&:hover": { backgroundColor: "#0072ff" },
            }}
          >
            {questionIndex < questions.length - 1 ? "Next" : "Finish"}
          </Button>
        </Box>
      </Card>
    </Container>
  </ContentWrapper>
 </>
);
```

```
};


export default MockInterview;
```

Navbar.jsx

```
import React, { useState } from "react";
import {
  AppBar,
  Toolbar,
  Typography,
  Button,
  Box,
  IconButton,
  Menu,
  MenuItem,
  useMediaQuery,
  useTheme,
} from "@mui/material";
import { useNavigate, useLocation } from "react-router-dom";
import { EmojiEvents, Home, ExitToApp, Menu as MenuIcon, Share } from "@mui/icons-material";


const Navbar = ({ isAuthenticated, setIsAuthenticated }) => {
  const theme = useTheme();
  const isMobile = useMediaQuery(theme.breakpoints.down("md"));
```

```
const navigate = useNavigate();
const location = useLocation(); // Get the current route
const [anchorEl, setAnchorEl] = useState(null);

// Hide Navbar on these pages
const hideNavbarRoutes = ["/", "/signin"];
if (hideNavbarRoutes.includes(location.pathname)) {
  return null;
}

const handleMenuOpen = (event) => {
  setAnchorEl(event.currentTarget);
};

const handleMenuClose = () => {
  setAnchorEl(null);
};

const handleNavigation = (path) => {
  navigate(path);
  handleMenuClose();
};

const handleLogout = () => {
```

```
  localStorage.removeItem("isLoggedIn");
  localStorage.removeItem("username");
  setIsAuthenticated(false);
  navigate("/signin");
  handleMenuClose();
};

const handleShare = () => {
  if (navigator.share) {
    navigator
      .share({
        title: "ASPIRA",
        text: "Check out this amazing platform!",
        url: window.location.href,
      })
      .then(() => console.log("Successful share"))
      .catch((error) => console.log("Error sharing", error));
  } else {
    navigator.clipboard
      .writeText(window.location.href)
      .then(() => alert("Link copied to clipboard!"))
      .catch(() => alert("Failed to copy link to clipboard."));
  }
};
```

```
return (
 <AppBar
  position="fixed"
  sx={{
   background: "linear-gradient(45deg, #1a237e 30%, #0d47a1 90%)",
   boxShadow: "0 3px 5px 2px rgba(0, 0, 0, .3)",
  }}
 >
  <Toolbar>
   <Typography
    variant="h6"
    sx={{
     flexGrow: 1,
     fontWeight: "bold",
     background: "linear-gradient(45deg, #00c6ff 30%, #0072ff 90%)",
     WebkitBackgroundClip: "text",
     WebkitTextFillColor: "transparent",
    }}
   >
    ASPIRA
   </Typography>

   {isAuthenticated && (
```

```
        <Box sx={{ display: "flex", alignItems: "center" }}>
         <IconButton color="inherit" onClick={handleShare} aria-label="share" sx={{ mx: 1
}}>
          <Share />
         </IconButton>

         {isMobile ? (
          <>
           <IconButton color="inherit" onClick={handleMenuOpen} aria-label="navigation
menu">
            <MenuIcon />
           </IconButton>
           <Menu anchorEl={anchorEl} open={Boolean(anchorEl)}
onClose={handleMenuClose}>
            <MenuItem onClick={() => handleNavigation("/home")}>
             <Home sx={{ mr: 1 }} /> Home
            </MenuItem>
            <MenuItem onClick={() => handleNavigation("/rank")}>
             <EmojiEvents sx={{ mr: 1 }} /> Rankings
            </MenuItem>
            <MenuItem onClick={handleLogout}>
             <ExitToApp sx={{ mr: 1 }} /> Logout
            </MenuItem>
           </Menu>
          </>
```

```
) : (
  <>
    <Button
      color="inherit"
      onClick={() => handleNavigation("/home")}
      startIcon={<Home />}
      sx={{
        mx: 1,
        "&:hover": {
          background: "rgba(255,255,255,0.1)",
        },
      }}
    >
      Home
    </Button>
    <Button
      color="inherit"
      onClick={() => handleNavigation("/rank")}
      startIcon={<EmojiEvents />}
      sx={{
        mx: 1,
        "&:hover": {
          background: "rgba(255,255,255,0.1)",
        },
```

```
            }}
          >
            Rankings
          </Button>
          <Button
            color="inherit"
            onClick={handleLogout}
            startIcon={<ExitToApp />}
            sx={{
              mx: 1,
              "&:hover": {
                background: "rgba(255,255,255,0.1)",
              },
            }}
          >
            Logout
          </Button>
        </>
      )}
    </Box>
  )}
 </Toolbar>
</AppBar>
);
```

```
};

export default Navbar;
```

CHAPTER 6

OUTPUT

6.1 START PAGE

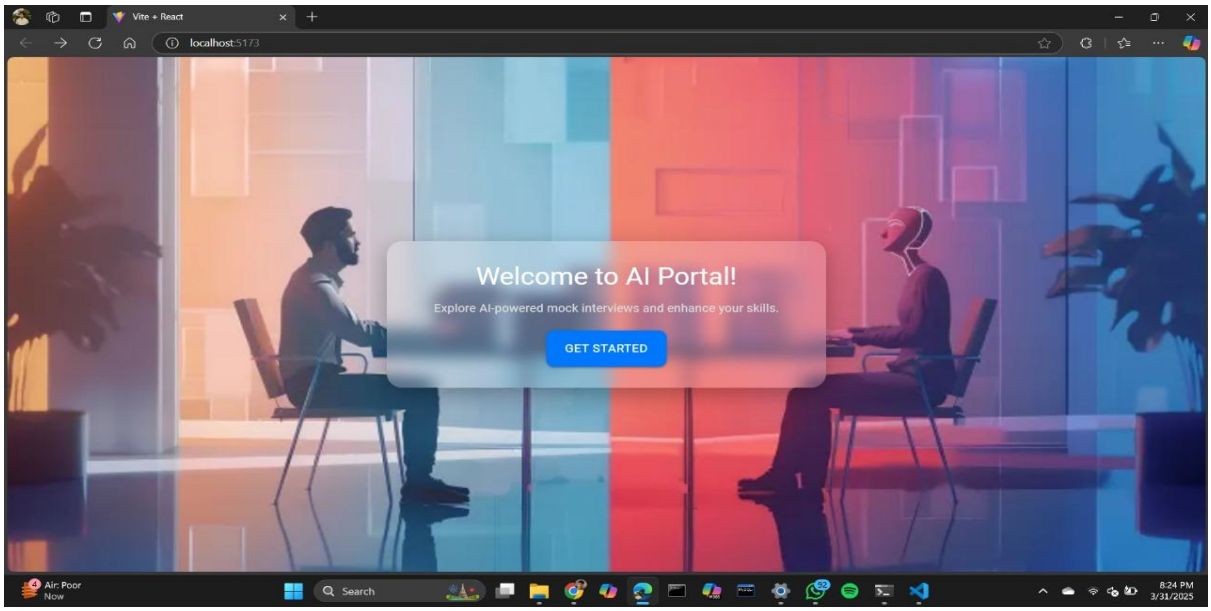

Figure 5:Sample image-1.

6.2 SECOND PAGE

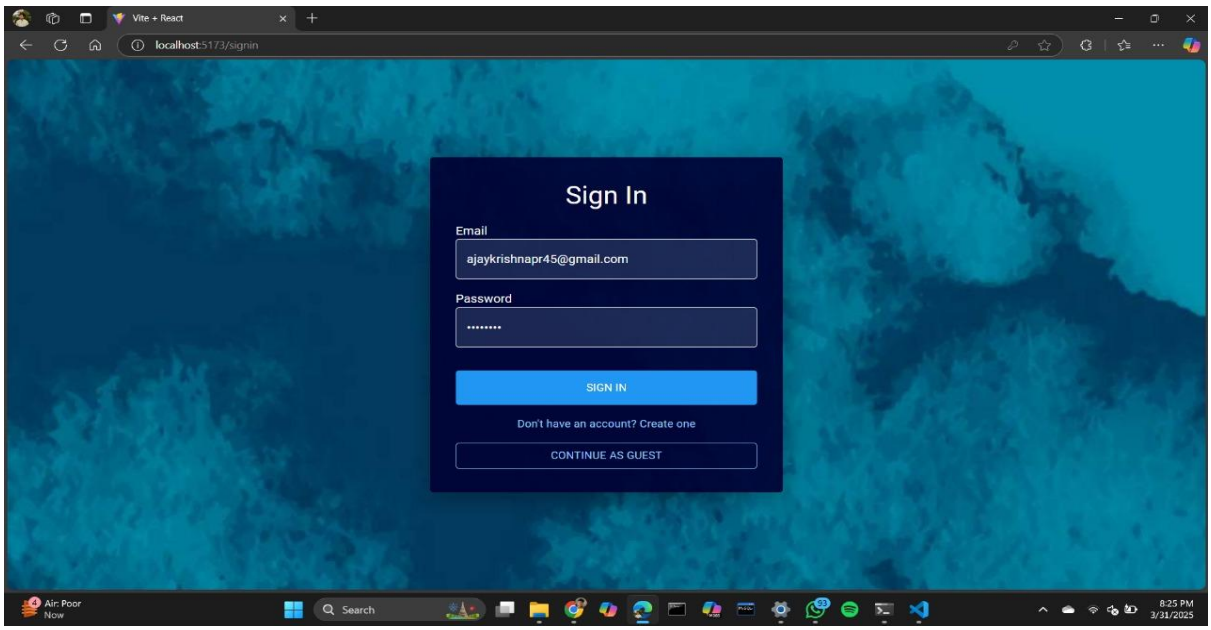

Figure 6:Sample image-2.

6.3 THIRD PAGE

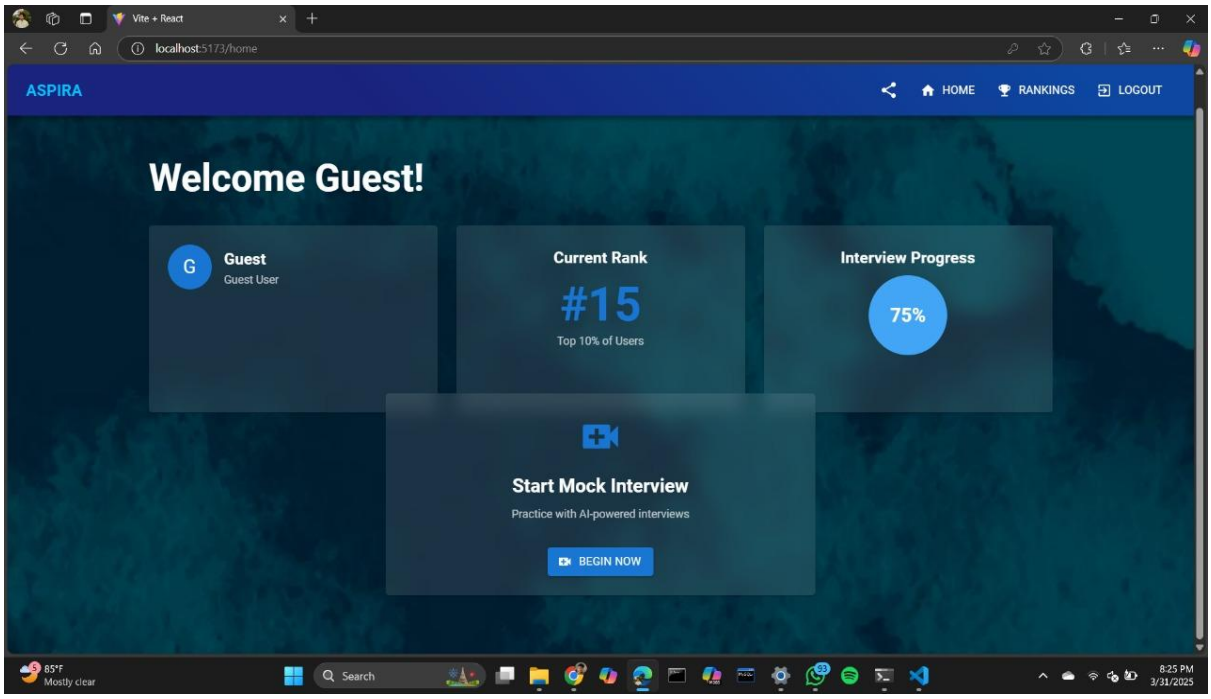

Figure 7:Sample image-3.

6.4 FOURTH PAGE

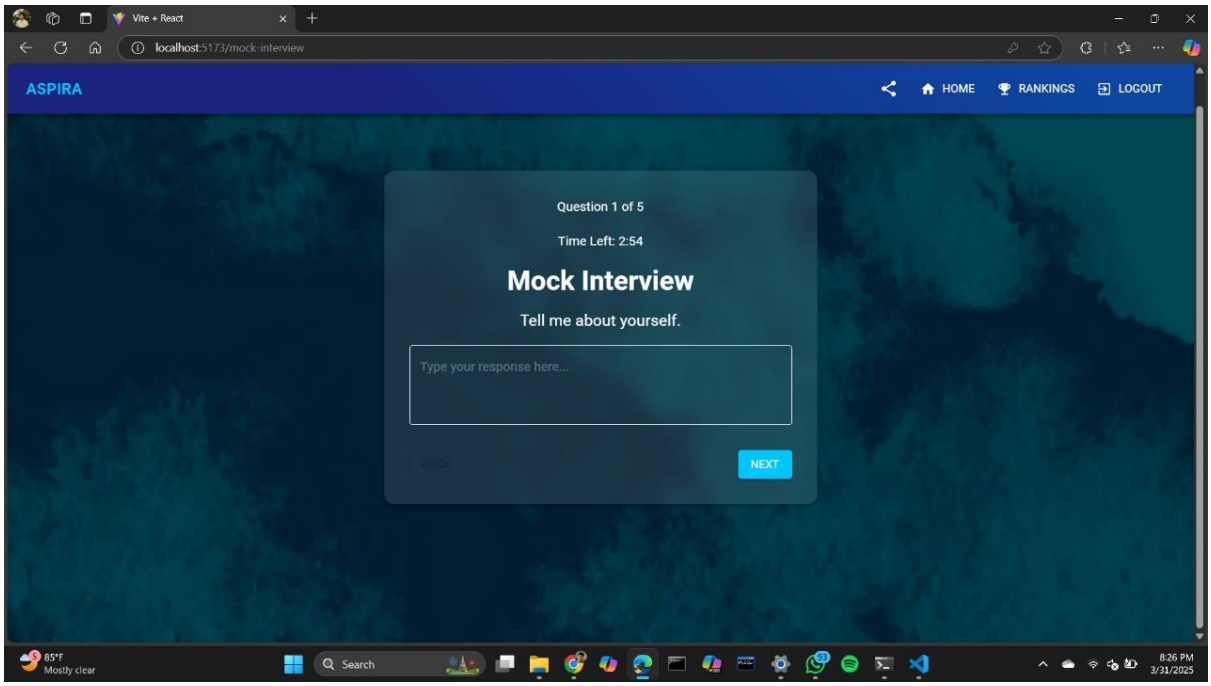

Figure 8:Sample image-4.

6.5 FIFTH PAGE

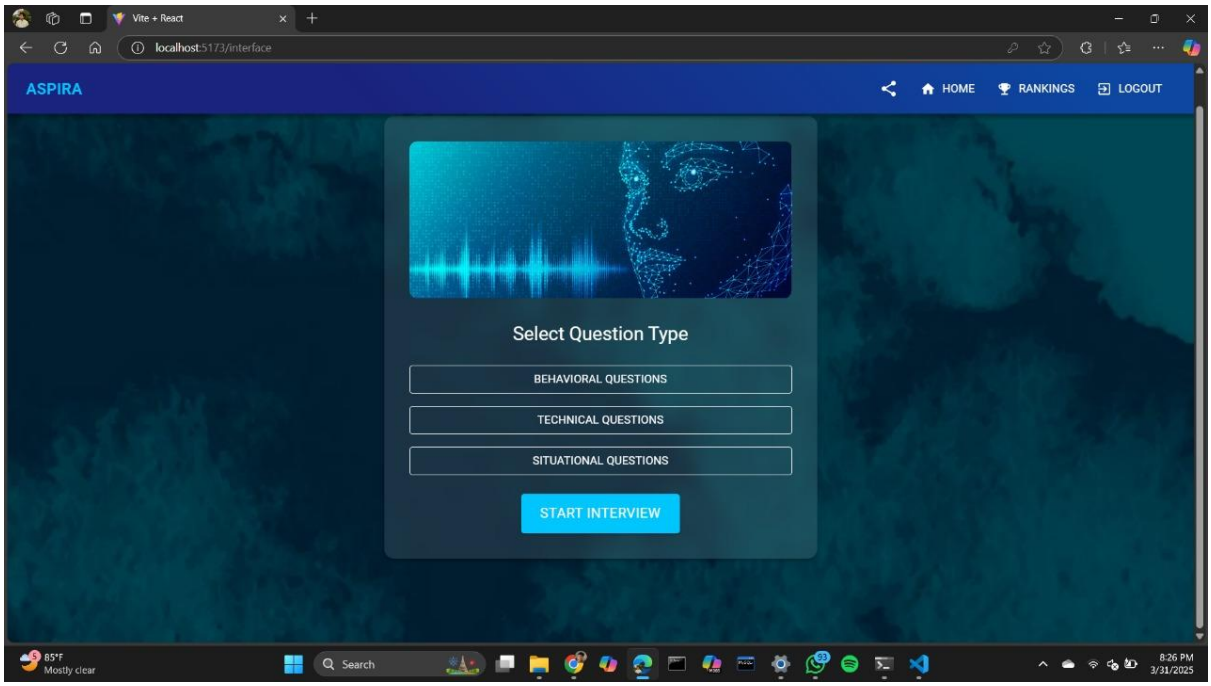

Figure 9:Sample image-5.

6.6 SIXTH PAGE

6.7 SEVENTH PAGE

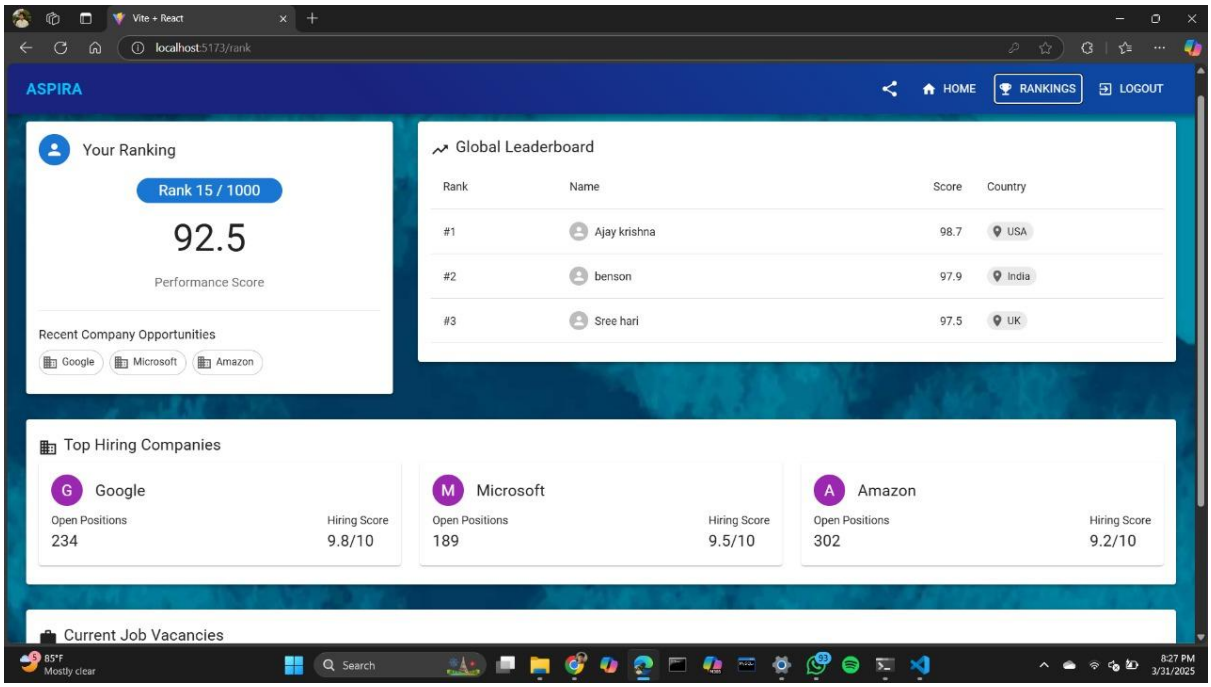

Figure 10:Sample image-7.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

ASPIRA is an AI-driven web-based platform designed to simulate real-world interview scenarios. By leveraging technologies like NLP, machine learning, and AI avatars, ASPIRA provides dynamic and adaptive interviews that assess candidates' skills, personality traits, and communication abilities. The system enhances users' confidence and interview preparedness by offering real-time feedback and analytics. Its application extends to academic institutions and training centres, allowing students and professionals to practice and refine their interview skills in a realistic and supportive environment. The integration of AI ensures an engaging and effective mock interview experience, helping users build confidence and improve their performance for real-world job opportunities.

7.2 FUTURE SCOPE

The future scope of interview training and assessment platforms extends beyond just the traditional tech and computer science fields, offering a broad potential for real-time stress interviews and mock interviews in a variety of professional domains. By integrating such systems into the education sector, these platforms can be introduced to students across different colleges and institutions, ensuring that they are better prepared for the real-world challenges they will face. These platforms can also be expanded to assess professionals by incorporating more complex models that simulate high-pressure scenarios, helping them to hone their skills even after they've entered the workforce. Additionally, the inclusion of behavioural analysis can play a crucial role in understanding a candidate’s mental state during interviews. By evaluating factors like anxiety, stress levels, and personal habits, these systems could provide valuable insights and personalized suggestions to help individuals manage and improve their performance. This holistic approach would not only focus on technical skills but also enhance emotional intelligence and overall wellbeing, creating more well-rounded professionals.

CHAPTER 8

REFERENCE

1.Tensor Flow-Based Automatic Personality Recognition Used in Asynchronous video interview,2021. (Authors : Hung-Yue Suen , Kuo-En Hung , Chien-Liang Lin).

2.Gaze and Head Rotation Analysis in a Triadic VR Job Interview Simulation ,2023. (Authors: Saygin Artiran, Poorva S. Bedmutha, Aaron Li, Pamela Cosman)

3.You're Hired! Effect of Virtual Agents' Social Status and Social Attitudes on Stress Induction in Virtual Job Interviews ,2024.(Authors: Celia Kessassi, Cedric Dumas, Caroline G. L. Cao, Mathieu Chollet)

4.Intelligent Deception Detection Through Machine-Based Interviewing ,2020. (Authors: Jim O'Shea, Keeley Crockett, Wasiq Khan, Philippos Kindynis, Athos Antoniades, Georgios Boultadakis)

5.Fairness-Aware Multimodal Learning in Automatic Video Interview Assessment 2023. (Authors: Changwoo Kim, Jinho Choi, Jongyeon Yoon, Daehun Yoo, Woojin Lee)

6.AI-based Behavioural Analyser for Interviews/Viva ,2021.(Authors: Venuri Amalya, Raveen Dissanayaka, Lahiru Lakshan)

7.Virtual Speech Anxiety Training - Effects of Simulation Fidelity on User Experience ,2023.(Authors: Sandra Poeschl, Nicola Doering)

8.Automated Analysis and Prediction of Job Interview Performance ,2019. (Authors: Iftekhar Naim, M. Iftekhar Tanveer, Daniel Gildea, Mohammed (Ehsan) Hoque)