A

PROJECT REPORT

ON

“JARVIS”

FOR

NeoSoft Services

SUBMITTED BY - **Manish Jangid**
Seat no. 10791

UNDER THE GUIDANCE OF

Ms. Kalyani Sahastrabuddhe

SAVITRIBAI PHULE PUNE UNIVERSITY (SPPU)
MASTER OF COMPUTER APPLICATIONS

DR. D. Y. PATIL UNITECH SOCIETY’S
DR. D. Y. PATIL INSTITUTE OF MANAGEMENT AND RESEARCH, PIMPRI,
PUNE-18

2024-2025

Dr. D. Y. Patil Unitech Society's
Dr. D.Y. PATIL INSTITUTE OF MANAGEMENT & RESEARCH
Sant Tukaram Nagar, Pimpri, Pune-411018
Recognized by the Savitribai Phule Pune University (SPPU-IMMP014220) AISHE CODE C-42109

Recipient of the "Best College Award" of Savitribai Phule Pune University | Accredited by NAAC with "A++ Grade" (CGPA 3.52) | MBA & MCA Programme Accredited by NBA

CERTIFICATE

This is to certify that **Manish Tulcharam Jangid** has successfully completed the project on**''JARVIS''**as a partial fulfilment of his **Master of Computer Applications (MCA Sem-IV)** under the curriculum of **Savitribai Phule Pune University, Pune** for the academic year 2024-25

Mrs. Kalyani S.	Dr. Shikha Dubey	Dr. Vishal Wadajkar
Project Guide	**H.O.D. MCA**	**Director**

Signature
Name
Internal Examiner
Date :

Signature
Name
External Examiner
Date :

NEOSOFT SERVICES

INTERNSHIP COMPLETION CERTIFICATE

Dear Mr. **Manish Jangid** **Date: 24th Apr 2025**

This is to certify that **Mr. Manish Tulcharam Jangid** pursuing MCA from Dr. D.Y. Patil Institute of Management and Research, Pune has completed internship with **Neosoft Services**. He has been working on the project Title : "JARVIS" during this period and also completed the tasks assigned to him as "Software Developer Intern".

The Internship started on **7th January 2025**. The intern has completed the assigned tasks within the mentioned period. The technologies used in the project are as listed below:

Frontend: Next.js (React-based framework)

Database: PostgreSQL (via Drizzle ORM)

API Models: Gemini-2.0-Flash API

We wish him all the Best.

Regards,

For Neosoft Services, Solapur

Director

ACKNOWLEDGEMENT

The successful completion of my final-year project, **"JARVIS – Job AI Ready Virtual Interview System"**, marks a significant milestone in my academic journey, and I would like to express my heartfelt gratitude to all those who supported and guided me throughout this endeavor.

First and foremost, I am profoundly thankful to **Dr. Vishal Wadajkar (Director)**, and **Dr. Shikha Dubey (Head of Department)** for providing an environment that fosters innovation and research. Their leadership and vision encouraged me to aim higher and push the boundaries of what I could achieve.

I extend my deepest appreciation to my project guide, **Ms. Kalyani sahastrabuddhe**, for her continuous support, insightful feedback, and expert guidance. Her valuable suggestions and mentorship were instrumental in shaping the project and helping me navigate technical challenges with clarity and confidence.

I would also like to thank all the faculty members for their encouragement and for sharing their knowledge, which has been a source of constant learning. A special mention goes to the laboratory staff for their timely help and cooperation during the practical implementation of the system.

A sincere thank you to my friends and fellow students who stood by me during every phase of this journey. Their honest feedback and camaraderie made this process enjoyable and enriching.

Finally, I owe everything to my family, whose unwavering support, patience, and belief in me have been my greatest strength. Their motivation kept me going even in the toughest moments. This project would not have been possible without the collective support of each of these individuals, and I am genuinely grateful to have had them by my side.

Yours Sincerely,
Manish Jangid

DECLARATION

This declaration is part of the project work entitled **"JARVIS"** is submitted as part of academic requirement for 4th semester of MCA Management.

I Manish Tulcharam Jangid solely declare that

1. I have not used any unfair means to complete the project.
2. I have followed the discipline and the rules of the organization where I was doing the project.
3. I have not been part of any act which may impact the college reputation adversely.

The information I have given is true, complete and accurate. I understand that failure to give truthful, incomplete and inaccurate information may result in cancellation of my project work.

Yours Sincerely,
Manish Jangid

TABLE OF CONTENT

Chapter		Title	Page No
1		Introduction	1-5
	1.1	Abstract	1
	1.2	Company Profile	1
	1.3	Existing System and Need for System	2
	1.4	Scope of System	2-3
	1.5	Operating Environment - Hardware and Software	3
	1.6	Brief Description of Technology Used	3-5
2		Proposed System	6-8
	2.1	Proposed System	6
	2.2	Feasibility Study	6-7
	2.3	Objectives of Proposed System	7
	2.4	Modules of Proposed System	8
3		Analysis and Design	9-22
	3.1	System Requirements	9-10
	3.2	Entity Relationship Diagram (ERD)	11
	3.3	Table Structure	12-13
	3.4	Use Case Diagram	14
	3.5	Class Diagram	15
	3.6	Activity Diagram	16
	3.7	Sequence Diagram	17
	3.8	Deployment Diagram	18
	3.9	Module Hierarchy Diagram	18
	3.10	Sample Input and Output Screens	19-22
4		Coding	23-36
	4.1	Code snippets	23-36
5		Testing	37-38
	5.1	Test Strategy	37
	5.2	Test Cases	38
6		Limitations of Proposed System	39-40
7		Proposed Enhancements	41-43
8		Conclusion	44-45
9		Bibliography	46-47

Chapter 1: Introduction

1.1 Abstract

In today's competitive job market, performing well in interviews is a key factor for success. However, not everyone has access to regular mock interviews, professional mentors, or structured practice environments. JARVIS (Job AI Ready Virtual Interview System) is a web- based, AI-powered mock interview platform that addresses this gap by offering a smart, automated, and personalized interview simulation system.

This project uses modern technologies like **Next.js**, **PostgreSQL**, and **Drizzle ORM**, combined with AI tools such as **GPT APIs**, to deliver domain-specific questions, evaluate responses, and provide detailed feedback. The system supports role-based question sets, user authentication, performance tracking, and a dashboard-driven experience. It is designed to help students, job seekers, and institutions conduct interview preparation efficiently and effectively.

JARVIS empowers candidates to rehearse interviews, reduce anxiety, and improve their skills through consistent and intelligent feedback, making it a valuable solution for placement readiness and career growth.

1.2 Company Profile

This project was developed under the guidance and mentorship of Neosoft Services, a recognized software solutions provider established in January 2011. Neosoft Services specializes in Web, Desktop, and Hybrid Mobile Application Development, focusing on innovation, timely delivery, and customer satisfaction.

With a presence in the ever-evolving global tech landscape, Neosoft Services stands out by delivering highly distinct digital identities for its clients. The company prides itself on its innovative vision and standard of excellence, which drives every project — whether for local enterprises or global brands.

The organization aims to make technology a true asset for its clients by thoroughly evaluating business needs and offering robust, scalable, and high-quality software solutions. Their customer-centric approach ensures that every product not only meets but often exceeds expectations in terms of usability, performance, and visual impact.

By integrating modern technologies and following industry best practices, Neosoft Services has built a reputation for delivering innovative solutions.

Website: https://neosoftservices.co.in/

1.3 Existing System and Need for System

Traditional mock interview systems are typically:

- **Manual and Time-Consuming**: Depend heavily on human evaluators.
- **Subjective**: Feedback is not always consistent or data-driven.
- **Generic**: Often lack role-based customization.
- **Limited in Analytics**: Do not provide measurable insights.

These limitations make it difficult for candidates to identify weak areas or track progress.

JARVIS solves these problems by automating the entire process — from dynamic question generation to feedback delivery. It offers scalability, personalization, and real-time evaluation using AI models, thereby making interview preparation more accessible and effective.

1.4 Scope of System

JARVIS is built for:

- **Students** preparing for campus placements.
- **Job seekers** who want regular mock practice.
- **Institutions** that want to offer interview training at scale. It

provides a digital platform where users can:

- Attempt role-specific mock interviews.
- Record and store responses.
- Receive intelligent, structured feedback.

Out of Scope:

- Real-time human interviewer interactions.
- Advanced biometric features (e.g., emotion detection, eye tracking).
- Resume based Questions (planned as a future enhancement).

1.5 Operating Environment – Hardware and Software

JARVIS is a web-based platform that can be used across multiple devices and platforms. It uses the following hardware and software configuration:

- **Operating Systems**: Windows 10/11, Linux (Ubuntu), macOS Ventura+
- **Browser Compatibility**: Chrome, Firefox, Edge, Safari
- **Database**: PostgreSQL (relational database), managed using Drizzle ORM
- **Backend Stack**: JavaScript (Next.js API routes), Drizzle for ORM
- **Frontend Stack**: React (via Next.js), Tailwind CSS for styling

No special hardware is required beyond standard computing devices with internet access and a webcam/microphone for future features.

1.6 Brief Description of Technology Used

The JARVIS mock interview platform is built using a full-stack web development architecture that combines a modern frontend, a serverless backend, AI-powered APIs, and a cloud-based database. Below is a detailed breakdown of the actual technologies used:

1. Next.js (Frontend and API Layer)

JARVIS is built using **Next.js**, a React-based framework that powers both the **frontend interface** and **API routes**. It enables server-side rendering, dynamic routing, and fast performance. Pages like Sign In, Dashboard, Interview Session, and Feedback are all developed using this framework.

2. React.js (UI Components)

React components are used to create dynamic and interactive UI elements such as:

- Interview question cards
- Collapsible feedback per question
- Responsive dashboards

3. Tailwind CSS (Styling)

Tailwind CSS is used for consistent and responsive styling across the platform. It helps create a clean and professional UI for all components including buttons, cards, modals, and feedback sections.

4. React Speech Recognition (Speech-to-Text Hook)

JARVIS integrates react-speech-recognition, a library that uses the browser's Web Speech API to convert spoken responses into text. This allows users to answer interview questions via **voice input**, improving realism and accessibility.

5. Gemini Flash 2.0 API (AI Integration)

The system uses **Gemini Flash 2.0**, a powerful large language model from Google, integrated via API. It performs two key functions:

- **Question Generation**: It dynamically generates interview questions based on selected roles like "Frontend Developer" or "HR".
- **Answer Analysis**: It analyzes each answer submitted by the user and returns intelligent, structured feedback focusing on **clarity, fluency, tone, and relevance**.

This AI integration removes the need for manually curated question banks and provides personalized feedback at scale.

6. Drizzle ORM (Database Layer)

The backend of JARVIS uses **Drizzle ORM**, a modern TypeScript-based ORM to handle interaction with the database. It provides type-safe schema definitions, smooth query building, and easy migration support.

7. PostgreSQL (Database)

Data such as users, interview sessions, questions, answers, and feedback are stored in a **Neon serverless PostgreSQL database**, making the system scalable and reliable. It allows efficient retrieval and storage of session data and user activity.

This tech stack ensures JARVIS is:

- **AI-powered**
- **Speech-enabled**
- **Fully responsive**
- **Serverless and scalable**

It creates a realistic, engaging, and professional mock interview experience for users preparing for job interviews.

Chapter 2: Proposed System

2.1 Proposed System

The proposed system, **JARVIS (Job AI Ready Virtual Interview System)**, is an AI-powered mock interview platform designed to simulate real interview environments using modern web technologies. The platform addresses the limitations of traditional mock interview methods by offering a fully digital, voice-enabled, and intelligent solution for self-practice.

Users can sign up, attempt role-specific interviews, answer questions by speaking, and receive structured AI feedback. Unlike manual systems, JARVIS offers personalization, instant analysis, and performance tracking, all within an intuitive web interface.

The system is suitable for students preparing for placements, professionals preparing for interviews, and institutions wanting to offer mock interview solutions to large batches.

2.2 Feasibility Study

Before development, a detailed feasibility study was conducted across three main aspects:

a. Technical Feasibility

- The system uses **Next.js** for building scalable UI and APIs.
- It integrates with **Gemini Flash 2.0 API** for AI-powered question generation and feedback analysis.
- Uses **React Speech Recognition** for speech-to-text functionality.
- Backend is powered by **Drizzle ORM + PostgreSQL (Neon serverless DB)** for reliable data storage.

Result: Technically feasible with all tools being open-source, documented, and actively supported

b. Operational Feasibility

- End users (students and job seekers) can easily access the platform through a web browser.
- The system flow (Login → Start Interview → Answer → Feedback) is intuitive and requires no technical training.
- Admin dashboard is not required for MVP, which reduces operational complexity.

Result: Operationally feasible with a simple onboarding flow.

c. Economic Feasibility

- Development done using free/open-source tools.
- Database hosting on **Neon** (serverless free tier).
- Gemini Flash 2.0 API used via **Google AI Studio** (free tier for development).
- No need for expensive licenses or infrastructure.

Result: Economically feasible for individual/academic deployment.

2.3 Objectives of Proposed System

The primary objectives of JARVIS are:

- To provide an **automated mock interview experience** for candidates.
- To generate **role-specific questions** using AI.
- To allow users to **speak answers** and transcribe them using speech-to-text.
- To **evaluate answers** using AI and provide real-time feedback.
- To **track user performance** across multiple sessions.
- To offer a system that is scalable, user-friendly, and intelligent.

2.4 Modules of Proposed System

The JARVIS platform consists of the following core modules:

1. **Authentication Module**
    a. Handles user signup, login, and session management.
2. **Dashboard Module**
    a. Shows existing interview history.
    b. Allows users to create new interview sessions.
3. **Interview Module**
    a. Generates questions based on the selected role.
    b. Captures answers via text or speech.
    c. Stores responses linked to session ID.
4. **AI Feedback Module**
    a. Sends responses to Gemini Flash 2.0 model.
    b. Receives feedback and scores for each answer.
    c. Stores and displays results for the user.
5. **Feedback Review Module**
    a. Presents feedback in a collapsible question-wise format.
    b. Includes detailed performance insights for each response.
6. **Speech Input Integration**
    a. Uses react-speech-recognition to convert speech to text.
    b. Allows voice-based answering during interviews.
7. **Database Interaction Module**
    a. Uses Drizzle ORM to manage schema and perform CRUD operations on PostgreSQL.

Chapter 3: Analysis and Design

3.1 System Requirements

3.1.1 Functional Requirements

These define the core functionalities that the JARVIS platform must support in order to deliver a seamless mock interview experience:

- **User Registration s Login:**
  The system must provide a secure authentication system, allowing users to register and log in. Users will have their unique profiles where interview history and progress are stored.

- **Interview Creation:**
  Users can create new interview sessions by selecting a predefined role (e.g., Software Developer, HR). Based on the selected role, the system generates role- specific questions dynamically.

- **Dynamic Question Generation**:
  The Gemini API is used to generate interview questions dynamically based on the user's selected role. This feature eliminates the need for static question banks and allows for personalized mock interviews.

- **Answering Questions:**
  Users can answer questions via using speech-to-text (enabled through React Speech Recognition). This allows for a more interactive and realistic interview simulation.

- **Data Storage:**
  Each question, answer, and session metadata (user ID, timestamps, role) must be stored securely in the database for future access. This data can be used for performance tracking and historical review.

- **Feedback Generation:**
  After each interview session, the system should send user answers to the Gemini API for processing and receive feedback, which is then displayed to the user. The feedback will be question-wise, allowing users to review specific responses.

- **Performance Tracking:**
  The system will track key performance metrics such as accuracy, response time, and clarity of answers. Users should be able to review their performance in previous sessions, making it easy to track progress over time.

3.1.2 Non-Functional Requirements

These requirements describe the general properties of the system that ensure it meets quality standards:

- **Performance:**
  The system must load pages and respond to user actions in a timely manner. This includes quick page loads, fast question rendering, and smooth transitions between different interview phases. APIs must respond in under 3 seconds.

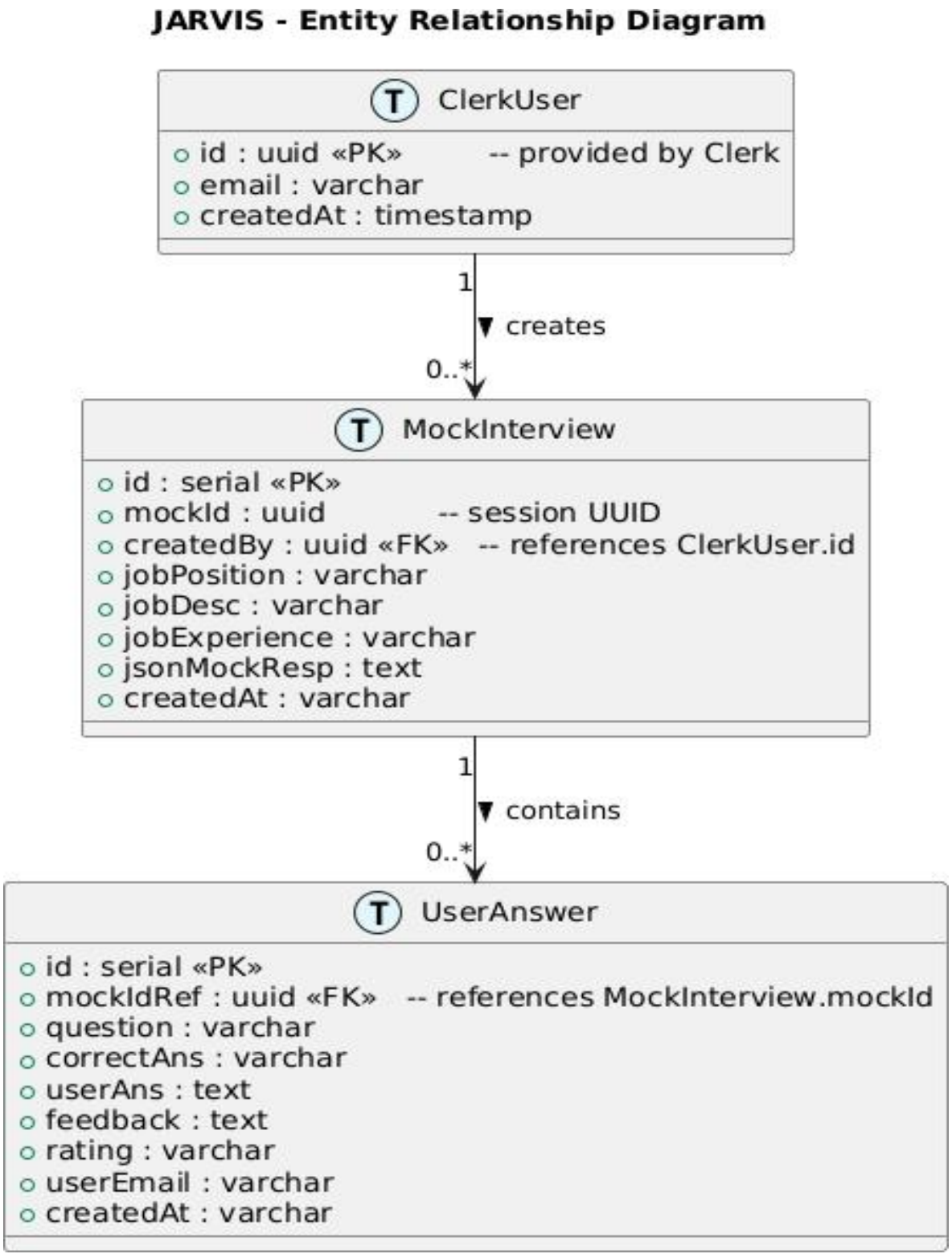
- **Usability:**
  The platform should have a clean and user-friendly interface. Tailwind CSS is used to ensure a consistent, responsive design that works well across devices (mobile, tablet, and desktop). The design should allow users to easily navigate between interviews, feedback, and their performance history.

- **Scalability:**
  The system is designed to handle an increasing number of users and interviews. It is built for serverless deployment using Neon DB for cloud-based storage, making it easy to scale as the user base grows. The backend API is designed for high scalability and availability.

- **Maintainability:**
  The codebase follows best practices for modular development. React components are reusable, and the Drizzle ORM provides a streamlined database interaction.
  This ensures the system can be easily maintained and enhanced with minimal effort as new features are added.

- **Accessibility:**
  The system is browser-compatible, ensuring it works well across all major web browsers (Chrome, Firefox, Safari, Edge). The platform is also designed with accessibility in mind, supporting features like speech-to-text for users with disabilities. A mobile-friendly interface ensures it can be accessed from various devices, providing a seamless experience for all users

3.2 Entity Relationship Diagram (ERD)



3.3 Table Structure

Table 1: MockInterview

Table: mockInterview	Type	Key	Description
id	SERIAL	PK	Internal surrogate primary key
mockId	UUID		Public session identifier (external key)
createdBy	VARCHAR	FK	Clerk user's email (references user)
jobPosition	VARCHAR		Role/position for the mock interview
jobDesc	VARCHAR		Short job description / tech stack
jobExperience	VARCHAR		Years of experience required
jsonMockResp	TEXT		JSON blob of AI-generated QCA
createdAt	VARCHAR		Timestamp (YYYY-MM-DD HH:MM:SS)

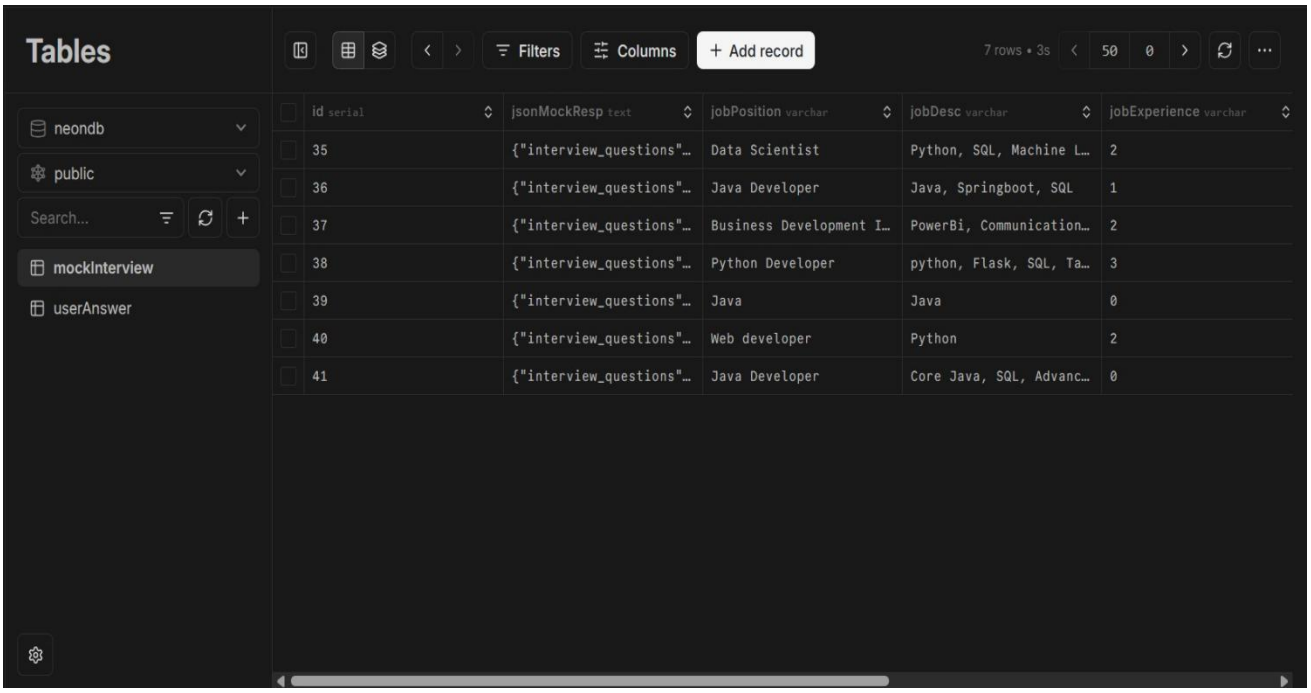
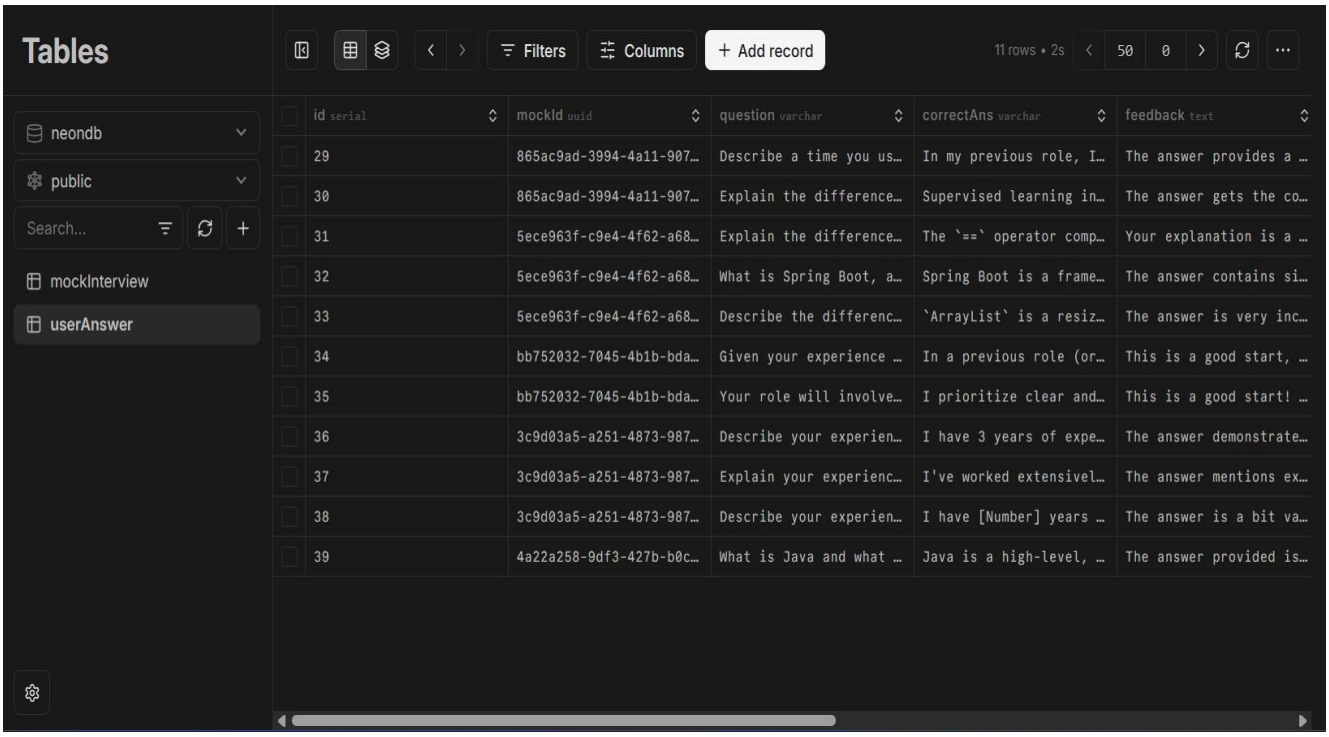
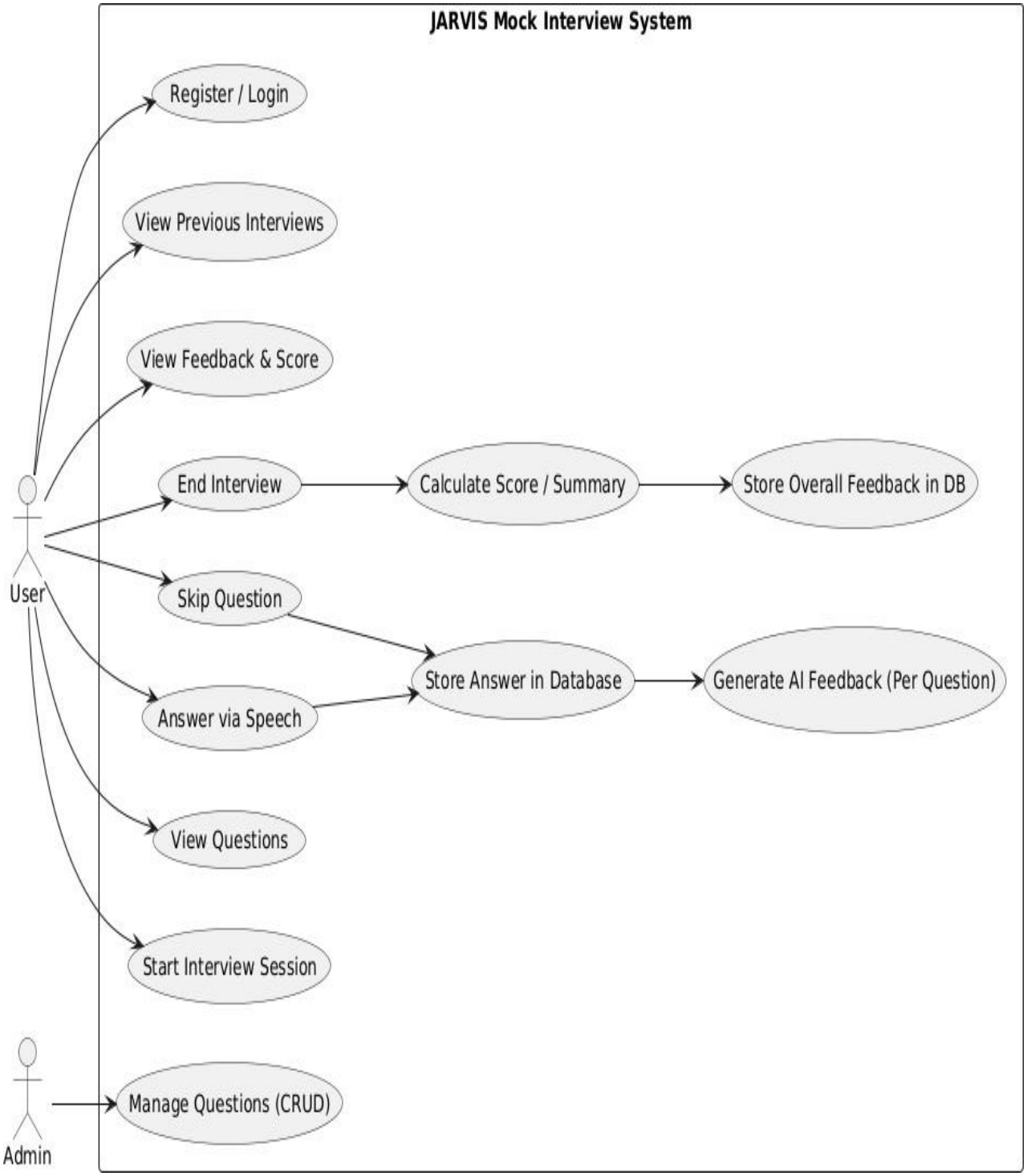


Table 2: UserAnswer

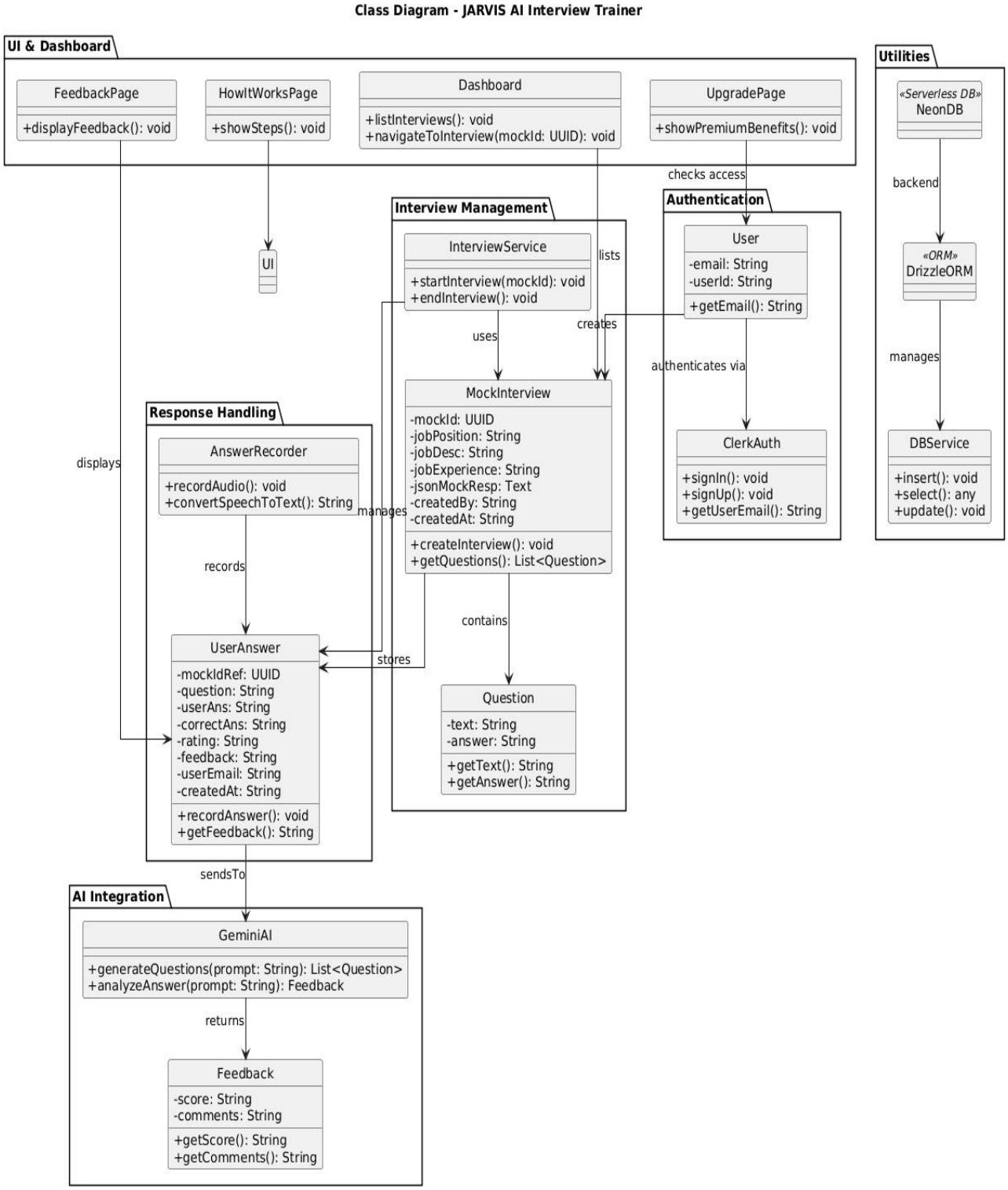
Table: userAnswer	Type	Key	Description
id	SERIAL	PK	Internal surrogate primary key
mockIdRef	UUID	FK	References mockInterview.mockId
question	VARCHAR		Interview question text
correctAns	VARCHAR		AI-provided correct answer
userAns	TEXT		User's response (speech-to-text or text)
feedback	TEXT		AI-generated feedback
rating	VARCHAR		AI-assigned rating (e.g. "8/10")
userEmail	VARCHAR		Clerk user's email who answered
createdAt	VARCHAR		Timestamp when answer was recorded



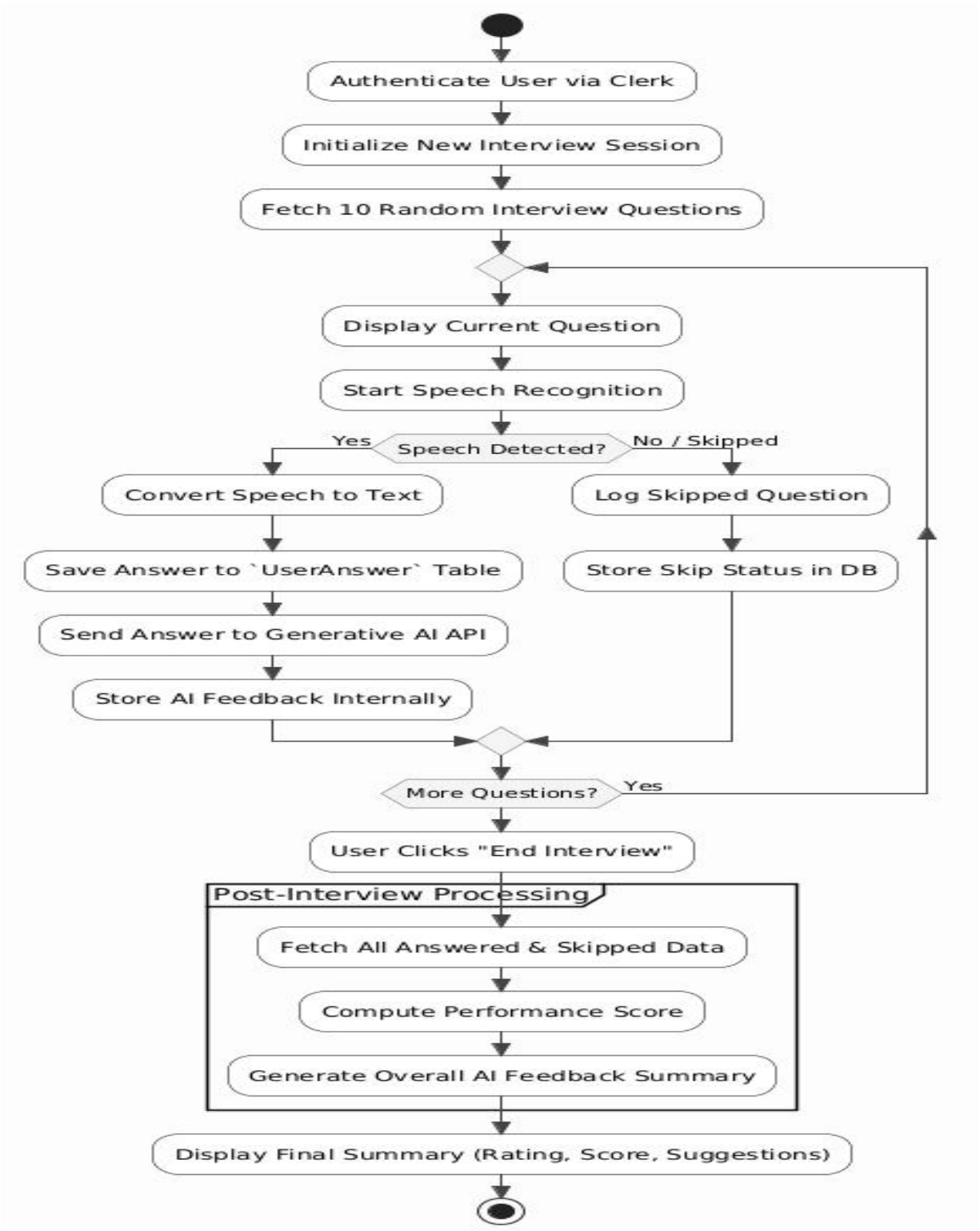
3.4 Use Case Diagram



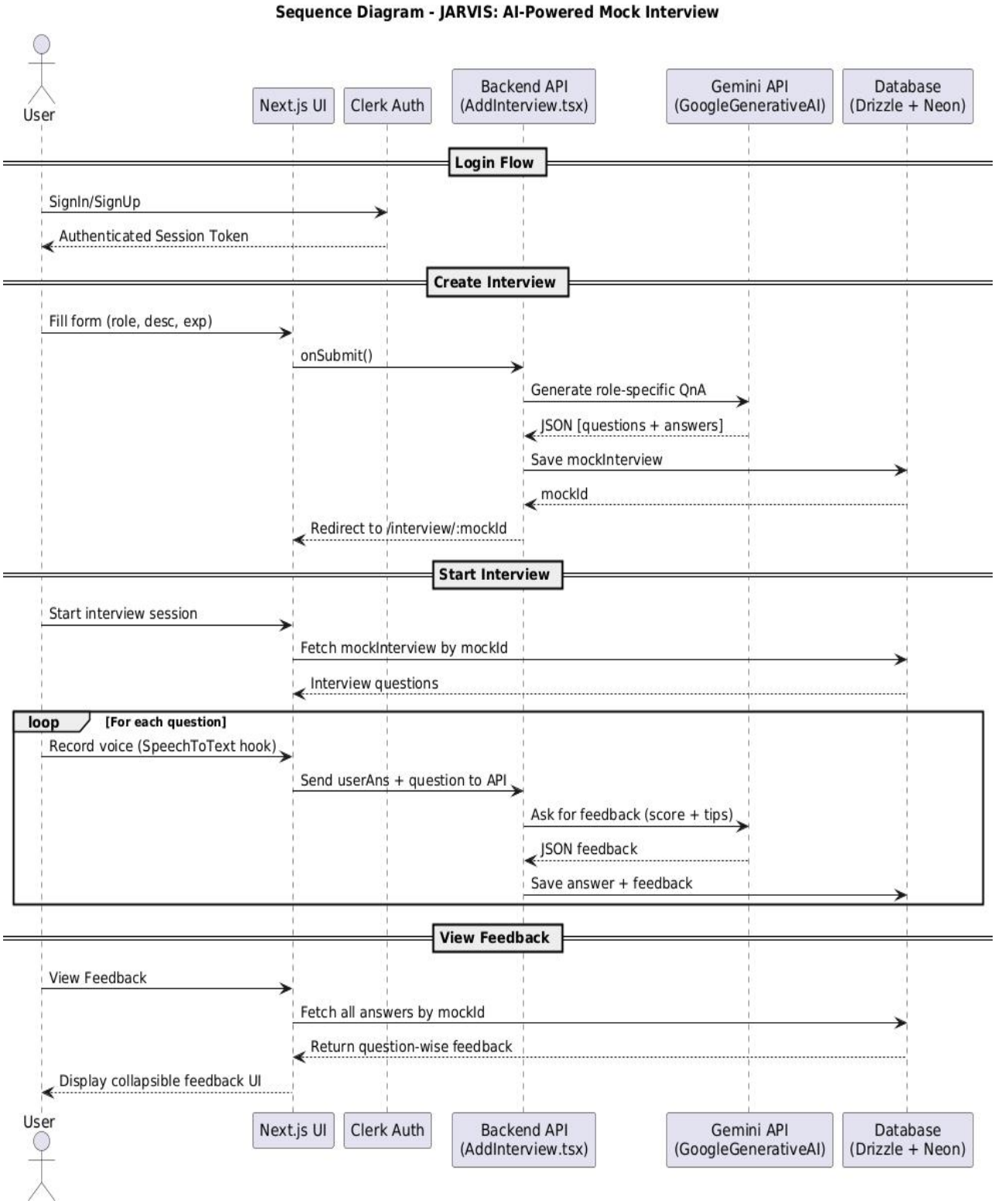
3.5 Class Diagram



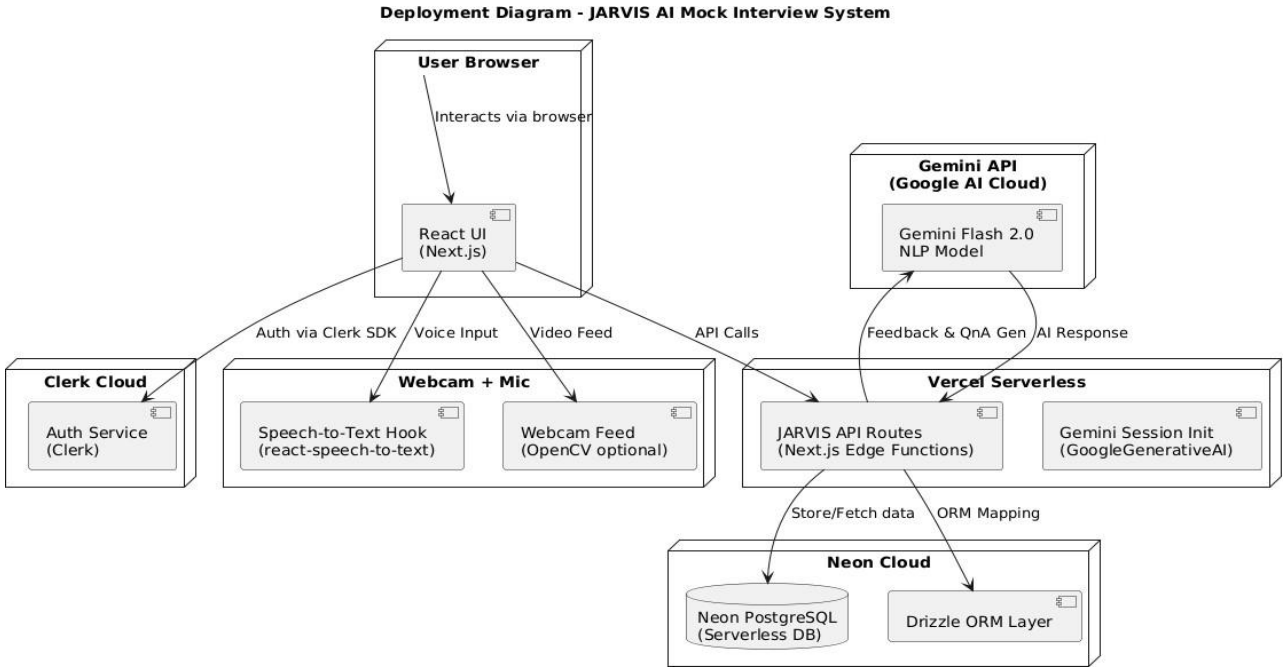
3.6 Activity Diagram



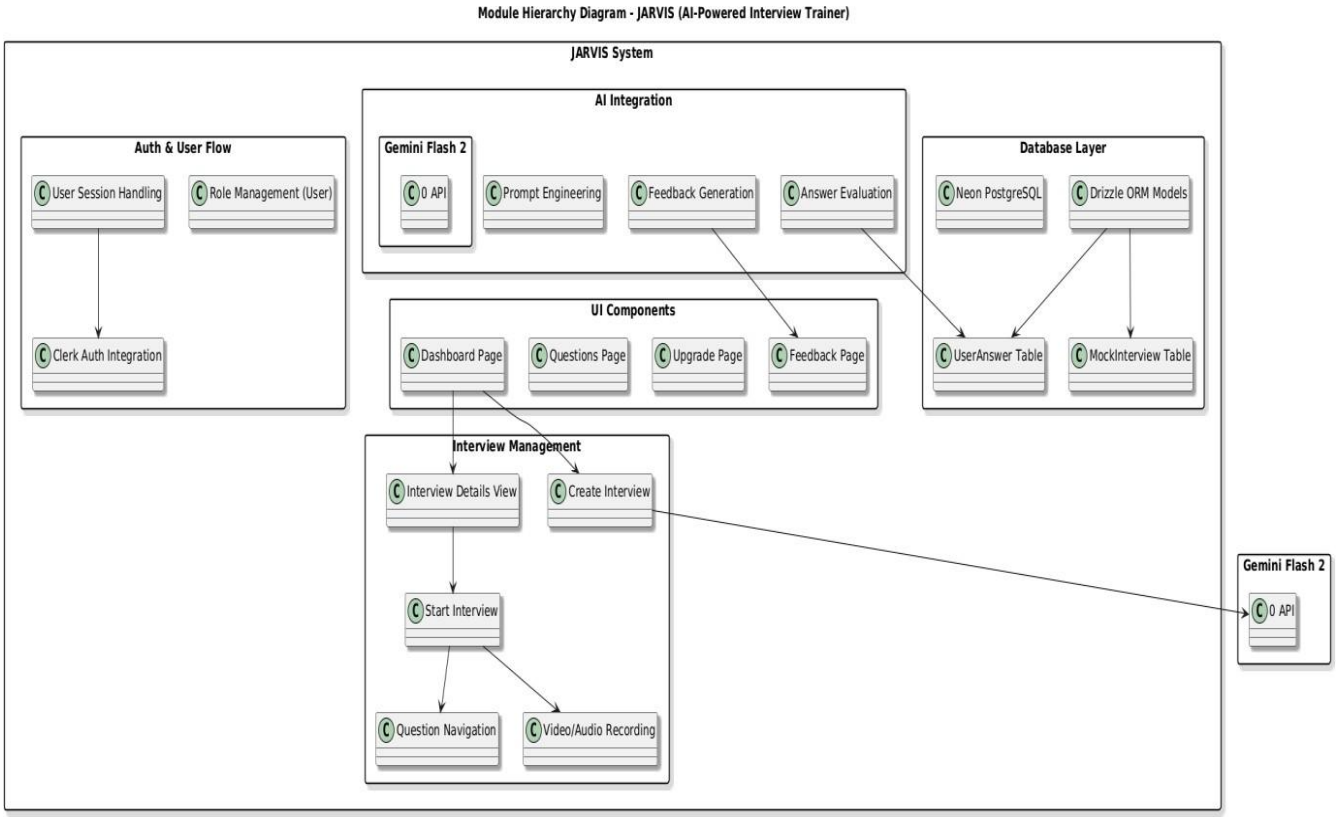
3.7 Sequence Diagram



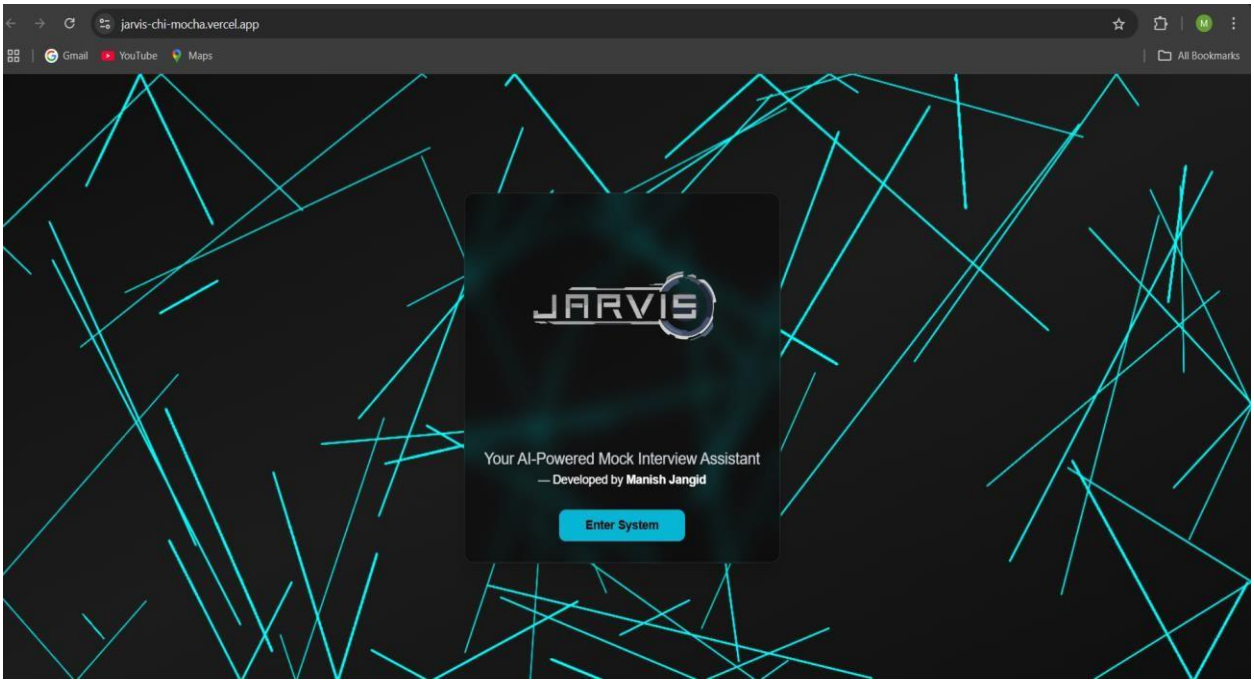
3.8 Deployment Diagram

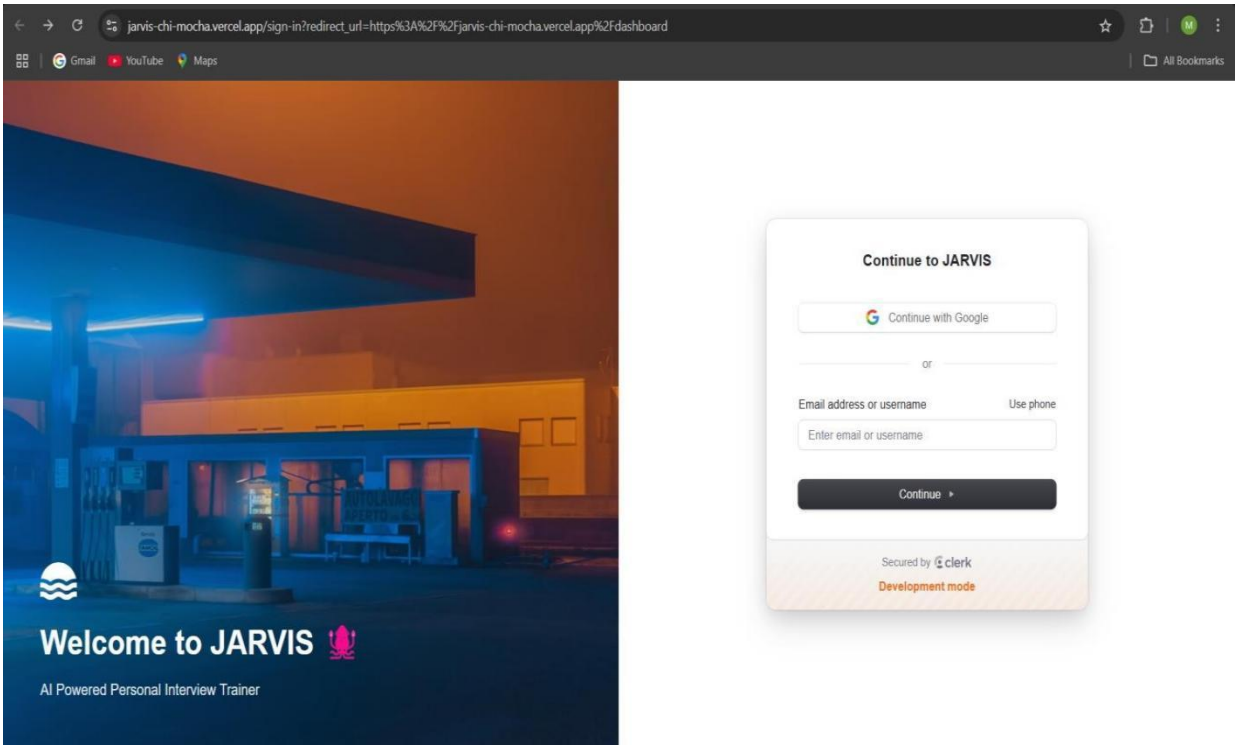


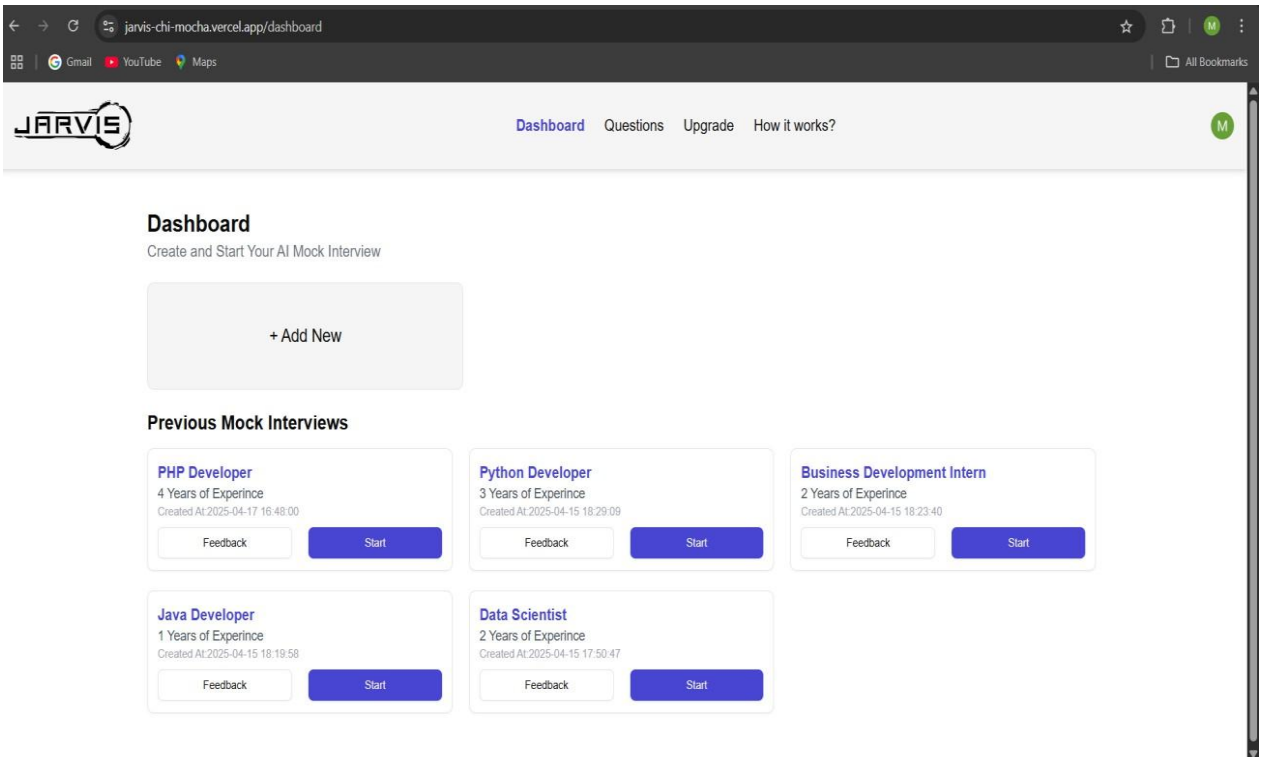
3.9 Module Hierarchy Diagram

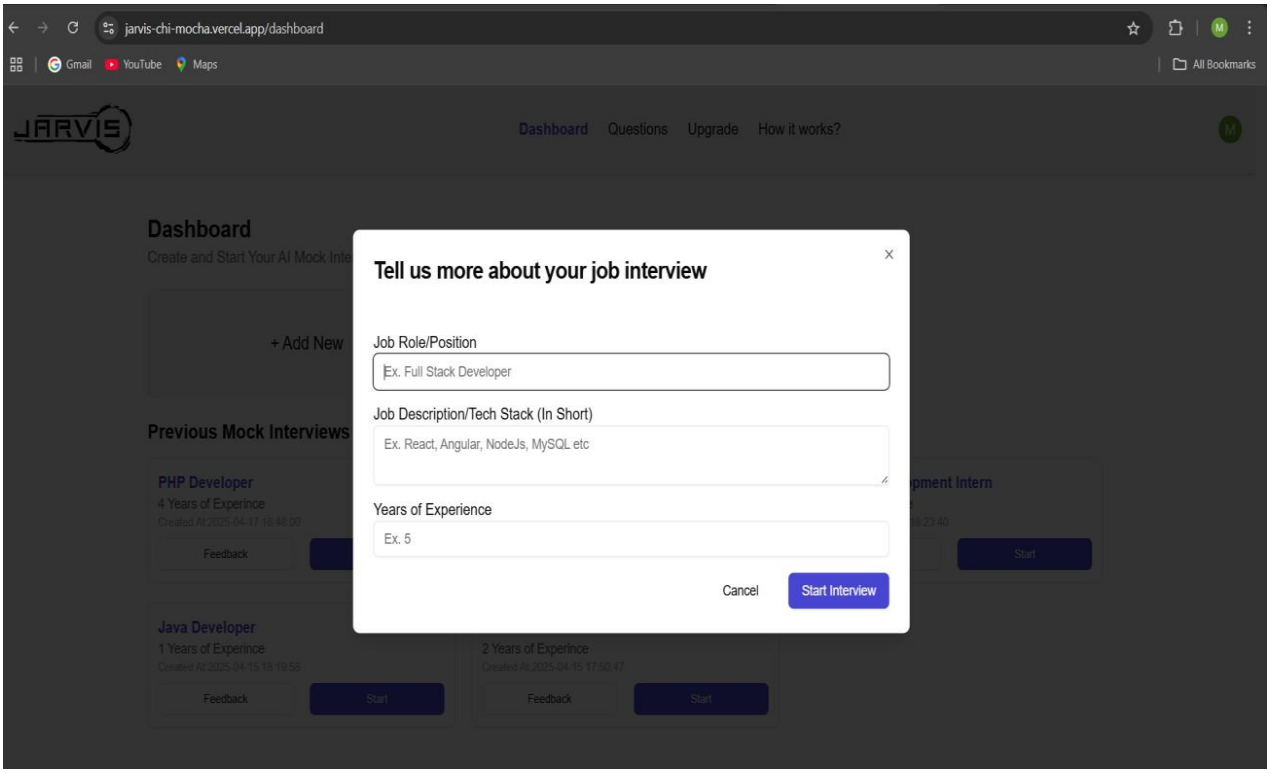


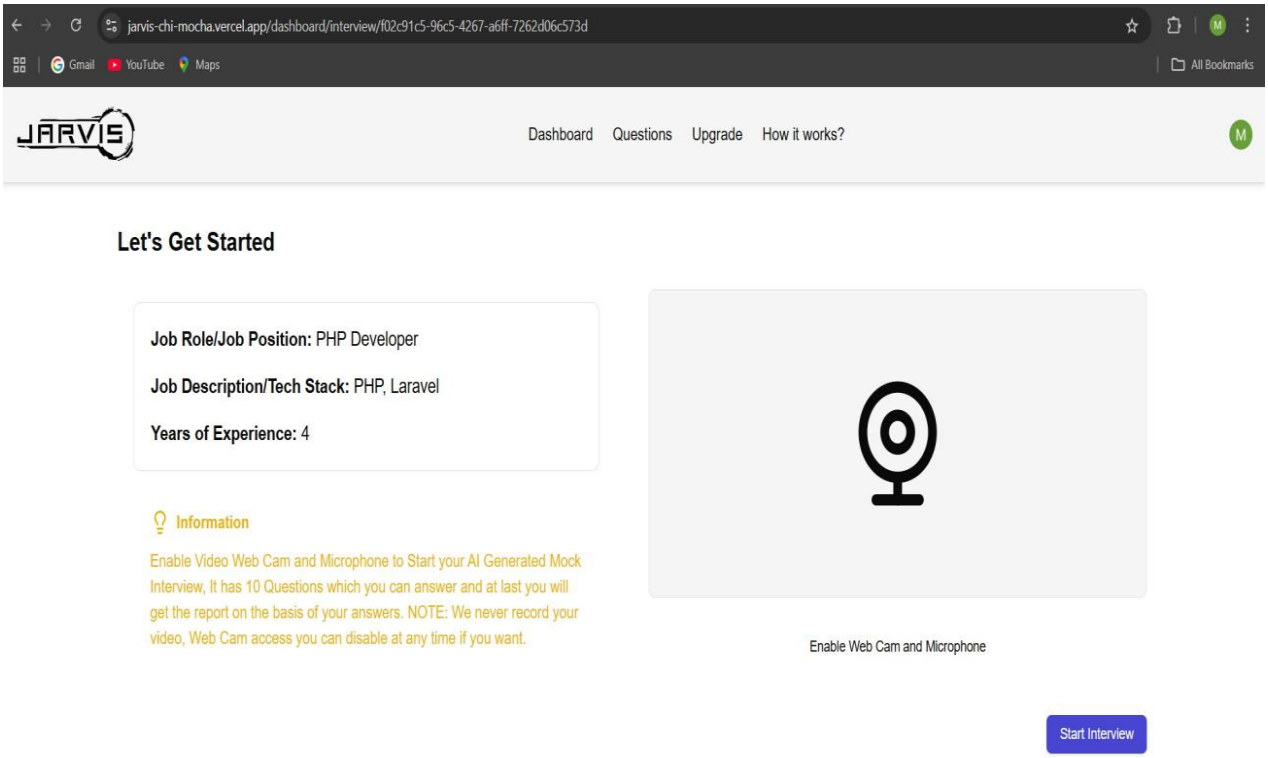
3.10 Sample Input and Output Screens

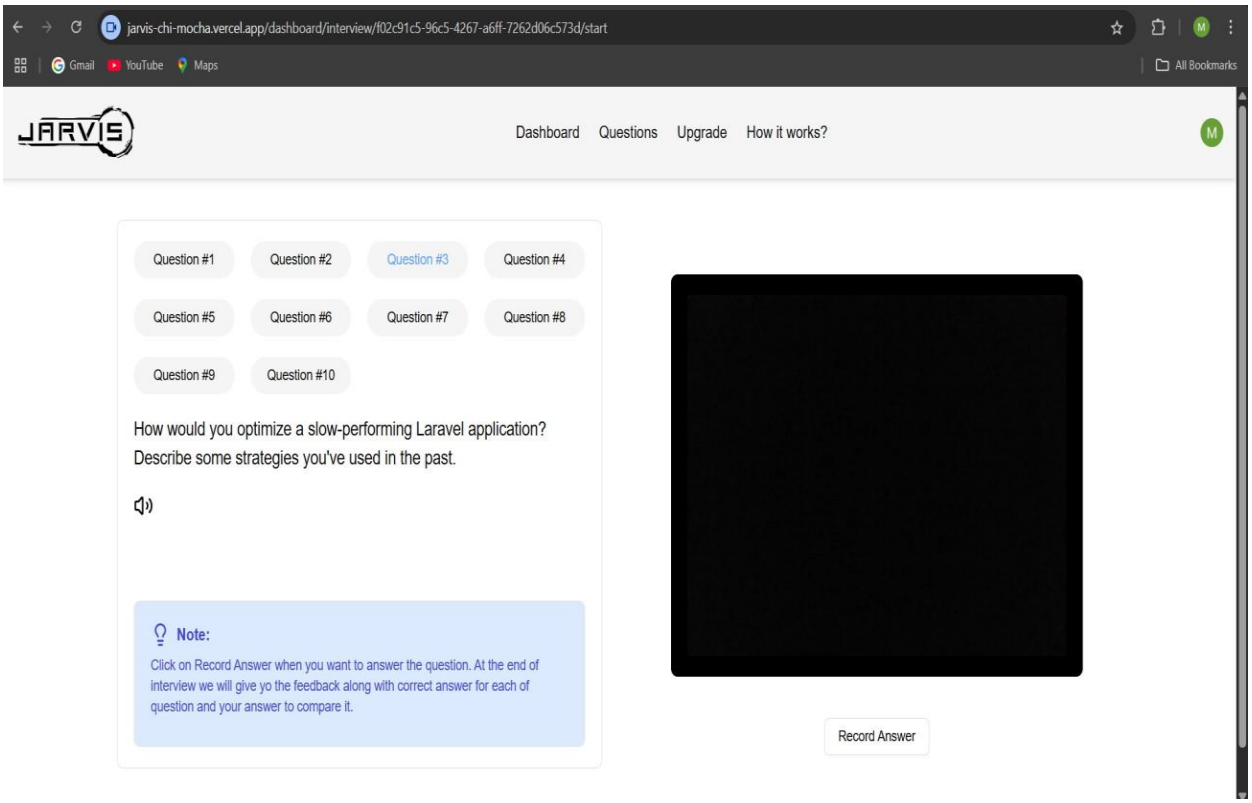


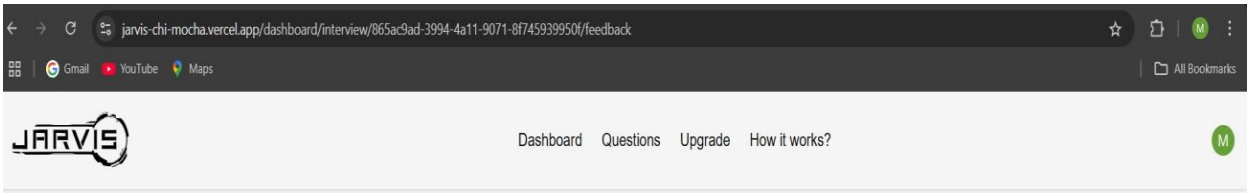



jarvis-chi-mocha.vercel.app/dashboard
Gmail
YouTube
Maps
All Bookmarks
JARVIS
Dashboard
Questions
Upgrade
How it works?
Dashboard
Create and Start Your AI Mock Interview
+ Add New
Previous Mock Interviews
PHP Developer
4 Years of Experince
Created At:2025-04-17 16:48:00
Feedback
Start
Python Developer
3 Years of Experince
Created At:2025-04-15 18:29:09
Feedback
Start
Business Development Intern
2 Years of Experince
Created At:2025-04-15 18:23:40
Feedback
Start
Java Developer
1 Years of Experince
Created At:2025-04-15 18:19:58
Feedback
Start
Data Scientist
2 Years of Experince
Created At:2025-04-15 17:50:47
Feedback
Start


jarvis-chi-mocha.vercel.app/dashboard
Gmail
YouTube
Maps
All Bookmarks
JARVIS
Dashboard
Questions
Upgrade
How it works?
Dashboard
+ Add New
Previous Mock Interviews
Tell us more about your job interview
Job Role/Position
Ex. Full Stack Developer
Job Description/Tech Stack (In Short)
Ex. React, Angular, NodeJs, MySQL etc
Years of Experience
Ex. 5
Cancel
Start Interview


jarvis-chi-mocha.vercel.app/dashboard/interview/f02c91c5-96c5-4267-a6ff-7262d06c573d
Gmail
YouTube
Maps
All Bookmarks
JARVIS
Dashboard
Questions
Upgrade
How it works?
M
Let's Get Started
Job Role/Job Position: PHP Developer
Job Description/Tech Stack: PHP, Laravel
Years of Experience: 4
Information
Enable Video Web Cam and Microphone to Start your AI Generated Mock Interview, It has 10 Questions which you can answer and at last you will get the report on the basis of your answers. NOTE: We never record your video, Web Cam access you can disable at any time if you want.
Enable Web Cam and Microphone
Start Interview


jarvis-chi-mocha.vercel.app/dashboard/interview/f02c91c5-96c5-4267-a6ff-7262d06c573d/start
Gmail
YouTube
Maps
All Bookmarks
JARVIS
Dashboard
Questions
Upgrade
How it works?
M
Question #1
Question #2
Question #3
Question #4
Question #5
Question #6
Question #7
Question #8
Question #9
Question #10
How would you optimize a slow-performing Laravel application? Describe some strategies you've used in the past.
Note:
Click on Record Answer when you want to answer the question. At the end of interview we will give yo the feedback along with correct answer for each of question and your answer to compare it.
Record Answer



Congratulations!

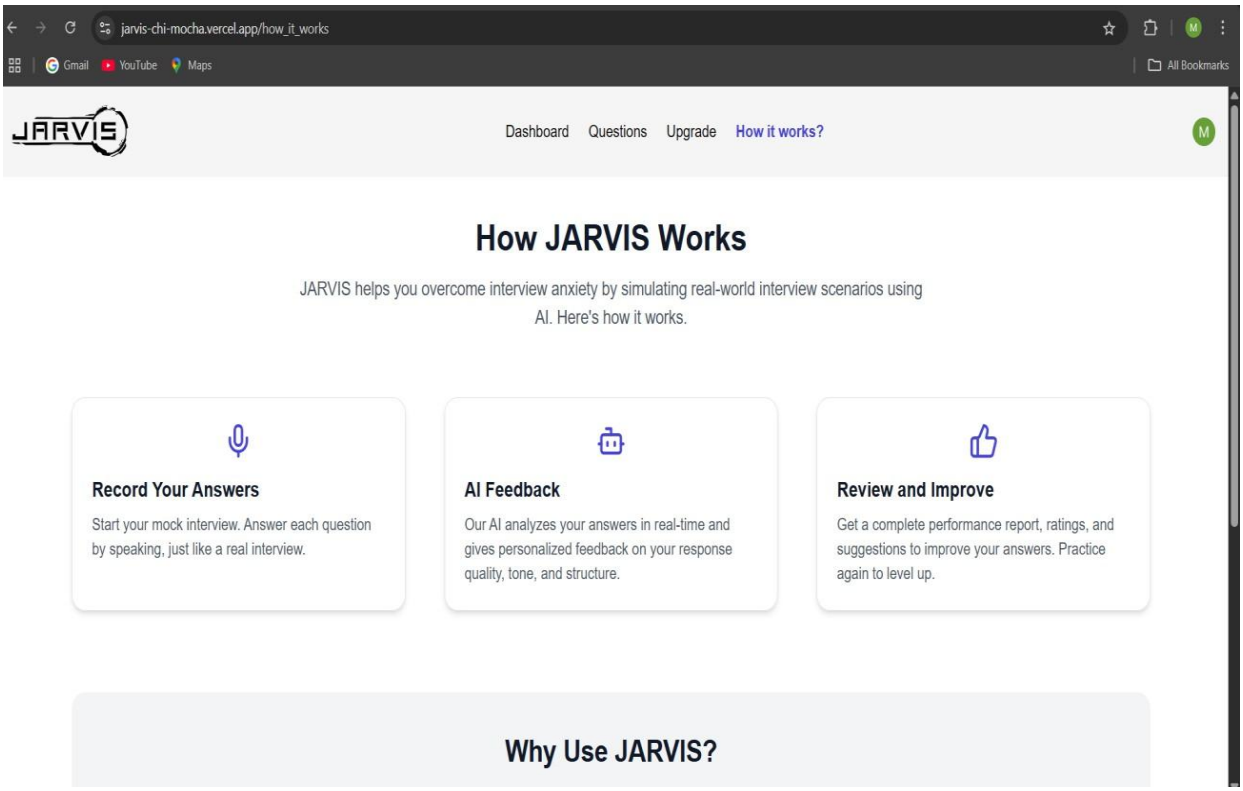
Here is your interview feedback

Below are the interview questions with your answers, correct answers, and tips for improvement.

Q1: Describe a time you used Python and SQL together to solve a data-related problem. What were the specific challenges you faced, and how did you overcome them?

Q2: Explain the difference between supervised and unsupervised learning. Provide examples of algorithms for each and when you might choose one over the other.

Go to Dashboard



Chapter 4: Coding

4.1 Code Snippets

This section provides an overview of the core code implementations used to bring the JARVIS mock interview system to life. The code is written using modern technologies such as React.js, Next.js, Drizzle ORM, Neon DB, Gemini AI API, and Clerk for authentication.

4.1.1 Database Configuration using Drizzle ORM

```
// db.js

import { neon } from '@neondatabase/serverless'; import

{ drizzle } from 'drizzle-orm/neon-http'; import * as

schema from './schema';

const sql = neon(process.env.NEXT_PUBLIC_DRIZZLE_DB_URL);

export const db = drizzle(sql, {schema} );
```

4.1.2 User Answer and Mock Interview Table Schema

```
// schema.js

import { pgTable, serial, text, varchar, uuid, timestamp } from 'drizzle-orm/pg-core';

export const MockInterview = pgTable('mockInterview', {
   id: serial('id').primaryKey(),
   mockId: uuid('mockId').defaultRandom(),
   jsonMockResp: text('jsonMockResp').notNull(),
   jobPosition: varchar('jobPosition').notNull(),
   jobDesc: varchar('jobDesc').notNull(),
   jobExperience: varchar('jobExperience').notNull(),
   createdBy: varchar('createdBy').notNull(),
   createdAt:varchar('createdAt')
});

export const UserAnswer=pgTable('userAnswer',{
   id:serial('id').primaryKey(),
   mockIdRef:uuid('mockId').notNull(),
   question:varchar('question').notNull(),
   correctAns:varchar('correctAns'),
   userAns:text('UserAns'),
   feedback:text('feedback'),
   rating:varchar('rating'),
   userEmail:varchar('userEmail'),
   createdAt:varchar('createdAt')
});
```

4.1.3 Gemini AI Integration

```
// GeminiAIModel.js

import { GoogleGenerativeAI } from "@google/generative-ai"; export

function initializeChatSession() {

  const apiKey = process.env.NEXT_PUBLIC_GEMINI_API_KEY;

  if (!apiKey) {

    console.error(" API Key is missing. Make sure NEXT_PUBLIC_GEMINI_API_KEY is set
in .env.local");

    return null;}

  const genAI = new GoogleGenerativeAI(apiKey);

  const model = genAI.getGenerativeModel({ model: "gemini-2.0-flash" });


  return model.startChat({ generationConfig: {

      temperature: 1,

      topP: 0.95,

      topK: 40,

      maxOutputTokens: 8192, responseMimeType:

      "text/plain",

    },

  });

}
```

4.1.4 Clerk Authentication in Root Layout

```
// layout.js

import { Geist, Geist_Mono } from "next/font/google";

import { ClerkProvider } from "@clerk/nextjs";

import "./globals.css";
import dynamic from 'next/dynamic';
import { Toaster } from "@/components/ui/sonner";

const geistSans = Geist({ variable: "-

 -font-geist-sans", subsets: ["latin"],

});
const geistMono =

 Geist_Mono({ variable: "--font-

 geist-mono", subsets: ["latin"],

});
export const metadata = { title:

 "Create Next App",

 description: "Generated by create next app",

};
export default function RootLayout({ children }) { return (

  <ClerkProvider>

  <html lang="en">

   <body

    className={`${geistSans.variable} ${geistMono.variable} antialiased`}

   >

    <Toaster />

    {children}

      </body>

  </html>

  </ClerkProvider>

 )}
```

4.1.5 Start Interview: AI Question Generation s JSON Parsing

```
//Start Interview.jsx

"use client";

import React, { use, useEffect, useState } from 'react'

import { useParams } from "next/navigation";

import { MockInterview } from "@/utils/schema";

import { db } from "@/utils/db";

import { eq } from "drizzle-orm";

import RecordAnswerSection from './_components/RecordAnswerSection';

import QuestionsSection from './_components/QuestionsSection';

import { Button } from '@/components/ui/button';

import Link from "next/link";



function StartInterview() { const

params = useParams();

const interviewId = params?.interviewId || "";

const [interviewData, setInterviewData] = useState();

const [mockInterviewQuestion,setMockInterviewQuestion] = useState([]); const

[activeQuestionIndex,setActiveQuestionIndex] = useState(0);



useEffect(() => {

  if (!interviewId || typeof interviewId !== "string")

    { console.warn("Invalid interviewId:", interviewId);

    return;

  }

  console.log("Fetching interview for ID:", interviewId);

  GetInterviewData(interviewId);
```

```
}, [interviewId]);

const GetInterviewData = async (interviewId) => { try {
    const result = await db.select().from(MockInterview).where(eq(MockInterview.mockId, interviewId));

    if (result.length === 0) {
      console.warn("    No interview found for this ID.");
      return;
    }
    console.log(" Fetched Interview Details:", result[0]);

    let jsonMockResp;
    try {
      jsonMockResp = result[0]?.jsonMockResp ? JSON.parse(result[0].jsonMockResp) :
null;
    } catch (error) {
      console.error(" Error parsing JSON:", error); return;
    }
    if (!jsonMockResp || Object.keys(jsonMockResp).length === 0)
      { console.error(" jsonMockResp is empty or invalid:", jsonMockResp);
      return;
    }
    if (!Array.isArray(jsonMockResp.interview_questions)) {
      console.error(" interview_questions is missing or not an array:", jsonMockResp); return;
    }
    console.log("Valid interview questions found:", jsonMockResp.interview_questions);
    setMockInterviewQuestion(jsonMockResp.interview_questions); setInterviewData(result[0]);
```

```
    } catch (error) {
      console.error(" Error fetching interview:", error);
    }
};

  return (
    <div>
      <div className='grid grid-cols-1 md:grid-cols-2 gap-10'>
        {/* Questions */}
        <QuestionsSection
        mockInterviewQuestion={mockInterviewQuestion}
        activeQuestionIndex={activeQuestionIndex}
        />

        {/* Video/ Audio Recording */}
        <RecordAnswerSection
        mockInterviewQuestion={mockInterviewQuestion}
        activeQuestionIndex={activeQuestionIndex}
        interviewData={interviewData}
        />
      </div>

      <div className='flex justify-end gap-6'>
        {activeQuestionIndex>0CC <Button
onClick={()=>setActiveQuestionIndex(activeQuestionIndex-1)}>Previous
Question</Button>}
        {activeQuestionIndex!=mockInterviewQuestion?.length-1CC <Button
onClick={()=>setActiveQuestionIndex(activeQuestionIndex+1)}>Next    Question</Button>}
        {activeQuestionIndex === mockInterviewQuestion?.length - 1 CC interviewData?.mockId CC
(
```

```
  <Link href={`/dashboard/interview/${interviewData.mockId}/feedback`}>
    <Button>End Interview</Button>
  </Link>
)}
      </div>
    </div>
  )
}

export default StartInterview
```

4.1.6 Recording Answer with Speech-to-Text

//RecordAnswerSection.jsx

```
"use client";
import React, { useEffect, useState } from "react";
import Image from "next/image";
import WebCam from "react-webcam";
import { Button } from "@/components/ui/button";
import useSpeechToText from "react-hook-speech-to-text";
import { Mic } from "lucide-react";
import { toast } from "sonner";
import { useUser } from "@clerk/nextjs";
import { db } from "@/utils/db";
import moment from "moment";
import { initializeChatSession } from "@/utils/GeminiAIModel";
import { UserAnswer } from "@/utils/schema";
import { MockInterview } from "@/utils/schema";

function RecordAnswerSection({ mockInterviewQuestion, activeQuestionIndex, interviewData }) {
  const [userAnswer, setUserAnswer] = useState("");
  const { user } = useUser();
  const [loading, setLoading] = useState(false);

  const {
    error,
    interimResult,
    isRecording,
    results,
    startSpeechToText,
    stopSpeechToText,
    setResults,
  } = useSpeechToText({
    continuous: true,
    useLegacyResults: false,
  });

  useEffect(() => {
    if (results.length > 0) {
      setUserAnswer(results.map((r) => r.transcript).join(" "));
    }
  }, [results]);

  useEffect(() => {
    if (!isRecording && userAnswer?.length > 10) {
      UpdateUserAnswer();
    }
  }, [userAnswer]);

  const StartStopRecording = async () => {
    if (isRecording) {
```

```
    stopSpeechToText();
  } else {
    startSpeechToText();
  }
};

const UpdateUserAnswer = async () => {
  console.log("    User Answer:", userAnswer);
  setLoading(true);

  const feedbackPrompt = `Question: ${mockInterviewQuestion[activeQuestionIndex]?.question},
    User Answer: ${userAnswer},
    Provide a JSON response with "rating"(out of 10) and "feedback" fields.`;

  try {
    const chatSession = initializeChatSession();
    if (!chatSession) throw new Error("AI Chat session initialization failed");

    const result = await chatSession.sendMessage(feedbackPrompt);
    let mockJsonResp = result.response.text();

    console.log("Raw AI Response:", mockJsonResp);

    // Clean the response by removing unwanted characters and ensuring proper JSON format
    mockJsonResp = mockJsonResp.replace(/```json|```/g, '').trim();

    try {
      // Attempt to parse the cleaned JSON
      const JsonFeedbackResp = JSON.parse(mockJsonResp);

      //  Insert into database using Drizzle ORM
      await db.insert(UserAnswer).values({
        mockIdRef: interviewData?.mockId,
        question: mockInterviewQuestion[activeQuestionIndex]?.question,
        correctAns: mockInterviewQuestion[activeQuestionIndex]?.answer,
        userAns: userAnswer,
        feedback: JsonFeedbackResp?.feedback,
        rating: JsonFeedbackResp?.rating,
        userEmail: user?.primaryEmailAddress?.emailAddress,
        createdAt: moment().format("YYYY-MM-DD HH:mm:ss"),
      });

      toast.success(" Answer Recorded Successfully!");
      setUserAnswer("");
      setResults([]);
    } catch (jsonParseError) {
      console.error(" Invalid JSON format:", jsonParseError);
      toast.error("Failed to parse AI response. Please try again.");
    }
  } catch (error) {
    console.error(" Error Updating Answer:", error);
```

```
      toast.error("Something went wrong!");
    }

    setLoading(false);
  };

  return (
    <div className="flex items-center justify-center flex-col">
      <div className="flex flex-col mt-20 justify-center items-center bg-black rounded-lg p-5">
        <Image src={"/WebCam.png"} width={300} height={300} className="absolute" alt="WebCam"
/>
        <WebCam
          mirrored={true}
          style={{
            height: 350,
            width: "100%",
            zIndex: 10,
          }}
        />
      </div>
      <Button disabled={loading} variant="outline" className="my-10" onClick={StartStopRecording}>
        {isRecording ? (
          <h2 className="text-red-600 flex gap-2">
            <Mic /> Stop Recording
          </h2>
        ) : (
          "Record Answer"
        )}
      </Button>
    </div>
  );
}

export default RecordAnswerSection;
```

4.1.7 Feedback Design s Display

//Feedback.jsx

```
"use client";
import React, { useEffect, useState } from 'react';
import { db } from '@/utils/db';
import { UserAnswer } from '@/utils/schema';
import { eq } from 'drizzle-orm';
import { useParams, useRouter } from 'next/navigation';
import {
  Collapsible,
  CollapsibleContent,
  CollapsibleTrigger,
} from "@/components/ui/collapsible";
import { ChevronsUpDown } from 'lucide-react';
import { Button } from '@/components/ui/button';
import clsx from 'clsx';

function Feedback() {
  const { interviewId } = useParams();
  const [feedbackList, setFeedbackList] = useState([]);
  const router = useRouter();

  useEffect(() => {
    if (!interviewId) return;
    GetFeedback();
  }, [interviewId]);

  const GetFeedback = async () => {
    try {
      const result = await db.select()
        .from(UserAnswer)
        .where(eq(UserAnswer.mockIdRef, interviewId))
        .orderBy(UserAnswer.id);
      setFeedbackList(result);
    } catch (error) {
      console.error(" Error fetching feedback:", error);
    }
  };

  return (
    <div className="max-w-screen-xl mx-auto px-6 sm:px-12 py-10">
      {feedbackList?.length === 0 ? (
        <div className="text-center text-gray-500 text-xl font-medium">
          No Interview Feedback Record Found
        </div>
      ) : (
```

```
      <>
        <header className="text-center mb-12">
          <h1 className="text-4xl font-bold text-green-600 mb-2">    Congratulations!</h1>
          <h2 className="text-2xl font-semibold text-gray-800">Here is your interview feedback</h2>
          <p className="text-gray-500 mt-2 text-sm">
            Below are the interview questions with your answers, correct answers, and tips for improvement.
          </p>
        </header>

        <section className="space-y-6">
          {feedbackList.map((item, index) => (
            <Collapsible key={item.id} className="bg-white border rounded-2xl shadow-sm transition
hover:shadow-md">
              <CollapsibleTrigger className="w-full flex justify-between items-center px-6 py-4 text-left
text-lg font-medium bg-muted/30 rounded-t-2xl hover:bg-muted/50 transition-colors">
                <span className="text-base md:text-lg text-gray-800">
                  <strong>Q{index + 1}:</strong> {item.question}
                </span>
                <ChevronsUpDown className="h-5 w-5 text-gray-600" />
              </CollapsibleTrigger>

              <CollapsibleContent className="px-6 py-5 bg-white rounded-b-2xl space-y-4">
                <div className="flex items-center gap-2 text-sm">
                  <span className="inline-block px-3 py-1 rounded-full bg-gray-100 text-red-600 font-
semibold">
                    Rating: {item.rating}/5
                  </span>
                </div>

                <AnswerBlock label="Your Answer" content={item.userAns} color="red" />
                <AnswerBlock label="Correct Answer" content={item.correctAns} color="green" />
                <AnswerBlock label="Feedback" content={item.feedback} color="blue" />
              </CollapsibleContent>
            </Collapsible>
          ))}
        </section>
      </>
    )}

    <div className="mt-12 text-center">
      <Button
        onClick={() => router.replace('/dashboard')}
        className="px-6 py-3 text-base font-medium rounded-xl"
      >
        ⬅ Go to Dashboard
      </Button>
    </div>
  </div>
);
```

```
}

export default Feedback;

//  Reusable Answer Block Component
function AnswerBlock({ label, content, color }) {
  const colorClasses = {
    red: 'bg-red-50 text-red-900 border-red-200',
    green: 'bg-green-50 text-green-900 border-green-200',
    blue: 'bg-blue-50 text-blue-900 border-blue-200',
  };

  return (
    <div
      className={clsx(
        "p-4 border rounded-lg text-sm whitespace-pre-line",
        colorClasses[color]
      )}
    >
      <strong className="block mb-1">{label}:</strong>
      {content}
    </div>
  );
}
```

Chapter 5: Testing

5.1 Test Strategy

The testing phase is crucial in ensuring the stability, accuracy, and performance of the JARVIS platform. It focuses on validating both functional and non-functional aspects of the system.

5.1.1 Objectives

- Ensure all modules perform as expected under different scenarios.
- Identify and fix bugs in individual components before integration.
- Validate data integrity and AI response accuracy.

5.1.2 Testing Levels

1. **Unit Testing**
    a. Conducted on individual modules like InterviewForm, RecordAnswerSection, and FeedbackList.
    b. Ensures isolated logic (e.g., API calls, form validations, speech recognition) works correctly.
2. **Integration Testing**
    a. Checks flow across components: Sign In → Create Interview → Answer → Feedback.
    b. Ensures smooth interaction between Clerk authentication, Gemini API, Drizzle DB, and Frontend.
3. **System Testing**
    a. End-to-end tests simulate a complete interview flow.
    b. Includes checking audio/video inputs, dynamic question generation, and feedback accuracy.
4. **User Acceptance Testing (UAT)**
    a. Performed with sample users (students, mentors).
    b. Ensures that UI is intuitive and output is meaningful and actionable

5.2 Test Cases

Test Case ID	Test Scenario	Input	Expected Output	Status
TC_001	User login via Clerk	Valid email/password	Redirect to Dashboard	Pass
TC_002	Add new interview session	Job role: Full Stack, 2+ years exp.	AI questions generated, interview created	Pass
TC_003	Start interview with speech-to-text	Voice input	Answer saved, feedback generated	Pass
TC_004	AI feedback generation	Answer to technical question	Feedback JSON with rating and suggestions	Pass
TC_005	Display feedback UI	Collapse/expand interaction	Questions shown with correct answer and rating fields	Pass
TC_006	Review past sessions	Go to Interview List	Previous interviews displayed with feedback/start option	Pass
TC_007	Use text-to-Speech for question	Question click on	System speaks question	Pass
TC_008	Session data persistence	Refresh interview screen	Question state and responses retained	Pass
TC_009	Invalid input handling	Submit blank job role	Validation error shown	Pass
TC_010	Gemini API fail scenario	API key missing	Error logged, user alerted	Pass

Chapter 6: Limitations of Proposed System

Despite offering a robust AI-driven mock interview experience, the current version of JARVIS has certain limitations that can be addressed in future iterations:

6.1 AI Feedback Dependence on Gemini API

- The system heavily depends on Gemini 2.0 Flash API for generating questions and evaluating user responses.
- If the API service is interrupted or deprecated, the application loses its core NLP functionality.
- Rate limiting or API key misuse can cause failures in real-time feedback generation.

6.2 Limited Real-Time Biometric Monitoring

- While webcam support is available, the system **does not yet perform live facial expression analysis or emotion detection** using tools like OpenCV.
- Non-verbal cue analysis (eye contact, facial stress) is not fully implemented, limiting behavioral insight.

6.3 No Strict Timer Mechanism During Interview

- JARVIS currently does **not enforce time limits** per question or session.
- While this allows flexible practice, it does not replicate strict time-bound environments common in real interviews.

6.4 Static Feedback Visualization

- Feedback is shown using collapsible sections per question, but lacks interactive analytics.

- There are **no progress charts**, **topic-wise strengths/weakness breakdown**, or **voice-tone emotion graphs** yet.

6.5 Limited Question Personalization

- Although questions are generated based on role, experience, and job description, **they do not yet account for user history or skill gaps**.
- No adaptive difficulty scaling or personalized learning paths are available at this stage.

Chapter 7: Proposed Enhancements

To evolve JARVIS into a more powerful, realistic, and insightful mock-interview platform, the following enhancements are proposed. Each enhancement addresses a current limitation (see Chapter 6) and adds significant value for end users.

7.1 Advanced Biometric and Behavioral Analytics

Objective: Capture non-verbal cues to mirror in-person interviews.

- **Facial Expression Analysis**
    - Integrate **OpenCV** or MediaPipe to track eye contact, smile frequency, and micro-expressions.
    - Provide metrics on "engagement score" and "confidence level."
- **Posture s Gesture Monitoring**
    - Use pose-estimation libraries to detect slouching, fidgeting, or confident stance.
    - Offer actionable tips (e.g. "sit upright," "maintain steady gaze").
- **Voice Modulation Analysis**
    - Analyze tone, pitch variation, and speech rate via Web Audio API.
    - Generate "vocal impact" score and suggestions (e.g. "slow down," "add emphasis here").

7.2 Timed Interview Mode

Objective: Simulate strict time-bound interview rounds.

- **Per-Question Timer**
    - Allow configuration of time limits (e.g. 2 minutes per question).
    - Display countdown and record "time taken" KPI.
- **Overall Session Timer**
    - Enforce total session duration to mimic real coding or case-study interviews.
    - Track "pace consistency" across questions.

7.3 Adaptive Question Difficulty

Objective: Personalize difficulty based on user performance.

- **Skill Profiling**
    - Maintain a history of user scores by topic (e.g. Data Structures, System Design).
    - Identify weak areas via analytics.
- **Dynamic Prompt Adjustment**
    - Adjust AI prompt to generate easier or harder follow-up questions.
    - Implement "level-up" mechanism: when user consistently scores ≥8/10, increase complexity.

7.4 Enhanced Data Visualization s Reporting

Objective: Provide actionable insights through interactive dashboards.

- **Performance Dashboards**
    - Use **Chart.js** or **Power BI** to plot trends: overall score, time per question, engagement metrics.
    - Offer filterable views (by date, role, topic).
- **Comparison Reports**
    - Benchmark a user's performance against peer averages (anonymized).
    - Highlight percentile rank and improvement trajectory.

7.5 Mobile-Friendly s Offline Support

Objective: Extend accessibility to mobile devices and low-connectivity scenarios.

- **Responsive PWA**
    - Convert JARVIS into a Progressive Web App for offline caching of question sets.
    - Enable users to practice without continuous Internet access.
- **Local Storage Sync**
    - Store intermediate answers in IndexedDB when offline; sync to server when reconnected.

7.6 Multi-Role and Panel Interview Simulation

Objective: Emulate real panel interviews with multiple AI "interviewers."

- **Multi-Actor Questioning**
    - Create multiple AI agents (e.g. HR, Technical Lead) with distinct prompt personas.
    - Rotate questioning styles: behavioral, technical, situational.
- **Panel Feedback Aggregation**
    - Collect and display feedback from each AI persona separately.
    - Provide consolidated "panel summary" and individual "panelist scores."

7.7 Integration with Real-World Platforms

Objective: Bridge JARVIS with industry tools and learning resources.

- **LMS Integration**
    - Connect with Learning Management Systems (Moodle, Canvas) to track student progress.
- **Job Portal APIs**
    - Fetch real job descriptions from LinkedIn or Indeed to generate up-to-date, role-specific questions.

Chapter 8: Conclusion

The JARVIS project has proven to be an innovative and useful AI-powered mock interview platform that assists users in overcoming interview anxiety and improving their performance. By simulating real-world interview scenarios, JARVIS empowers students, job seekers, and professionals to practice their interview skills in a controlled, intelligent environment.

8.1 Achievements of the JARVIS Project

- **AI-Powered Question Generation:** The integration of the Gemini API enables dynamic, role-specific questions, providing users with tailored interview preparation.
- **Speech Recognition s Feedback:** The inclusion of speech-to-text technology allows for a realistic interaction with the system, and the AI provides detailed feedback based on fluency, relevance, and accuracy.
- **Real-Time Feedback Mechanism:** Users receive immediate feedback on their performance, making it easier to track their improvement and identify areas for development.
- **Personalized Learning Experience:** The system's ability to generate questions based on job roles, job descriptions, and years of experience ensures that each user receives a relevant and unique experience.

8.2 Future Scope of JARVIS

While the current version of JARVIS provides significant value, there is always room for improvement and future expansion:

- **Enhanced Analytics s Reporting:** Future iterations will include deeper analytics on users' performance, helping them to better understand their strengths and weaknesses.
- **Behavioral and Non-Verbal Cues Analysis:** Integrating facial expression and gesture analysis will provide more comprehensive feedback, simulating a real interview environment.
- **Time-Bound Interviews:** The addition of strict time limits per question or session will enhance the simulation of real-world pressure faced during job interviews.
- **Mobile Support s Offline Access:** Expanding the platform to mobile devices and supporting offline modes will increase accessibility for users across diverse locations and situations.

8.3 Conclusion Summary

JARVIS represents a significant step forward in transforming the traditional interview preparation process. By leveraging AI, speech recognition, and dynamic question generation, the system offers a personalized, intelligent, and realistic mock interview experience. As the project evolves, incorporating additional features such as biometric analysis and advanced data visualizations will enhance the platform's ability to meet the needs of its users. The future enhancements outlined in this report aim to make JARVIS not just a mock interview tool, but a comprehensive interview coaching platform that can guide users through every stage of their preparation, offering tailored feedback and performance tracking.

Chapter 9: Bibliography

❖ Documentation and Online Resources

1. **Gemini API Documentation**, Google Cloud
    a. Guided the integration of Gemini AI for question generation and feedback.
2. **React Documentation**
    a. Essential for component-based development and state management.
3. **Drizzle ORM Documentation**
    a. Assisted in managing the database schema and interactions with Neon DB.
4. **Clerk Authentication Documentation**
    a. Implemented secure authentication for users in JARVIS.
5. **react-hook-speech-to-text**
    a. Enabled speech recognition for answering interview questions.
6. **Tailwind CSS Documentation**
    a. Used for responsive UI design in the frontend.
7. **OpenCV Documentation**
    a. Helped integrate facial recognition and image processing (future enhancement).
8. **GitHub Repositories for AI Models and Tools**
    a. Referenced for model integration and API setup for AI functionalities.

❖ Online Articles and Blogs

1. **"How to Build a Machine Learning Model with OpenCV and Python"**
    a. Practical approach to OpenCV integration for future facial expression analysis.

2. **"Getting Started with Serverless Databases: Neon"**
    a. Guide to Neon serverless database setup, used in JARVIS backend.
3. **"Building a Scalable Web Application with React and Node.js"**
    a. Insights on integrating React frontend with Node.js backend.

❖ Miscellaneous Sources

1. **YouTube Tutorials on Speech Recognition**
    a. Provided practical examples for integrating speech-to-text functionality.
2. **Stack Overflow Discussions**
    a. Community-driven solutions for React, Node.js, and AI model integration issues.

These resources were crucial for the development of JARVIS, providing foundational knowledge and practical implementation guidance throughout the project.