|  |  |  |  |
| --- | --- | --- | --- |
| **Name(s)** | Muhamad Ashar Javed | Ali Hassan | Aamir Hussain |
| **Registration Id(s)** | 223576 | 223583 | 223568 |
| **Session** | F-22 | | |
| **Course Name** | Full Stack Web Development | | |
| **Course Instructor** | Muhammad Rashaf Jamil | | |
| **Task Name** | Semester Project Submission | | |

Full Stack Web Development

**AI-Powered Mock Interview System**

**Table of Contents**

1. **Executive Summary**
2. **Project Objectives**
3. **Scope of Work**
4. **Target Users & Use-Cases**
5. **System Overview**
   * 5.1 High-Level Architecture
   * 5.2 Component Diagram
6. **Technology Stack**
7. **Detailed Feature List**
8. **Data Flow & State Management**
9. **User-Interface Design**
10. **Implementation Highlights**
11. **Testing Strategy & Quality Assurance**
12. **Security & Privacy**
13. **Performance & Scalability**
14. **Deployment Pipeline**
15. **Limitations**
16. **Future Enhancements**
17. **Conclusion and UI**
18. **Appendices** (file tree & key code snippets)

**1 Executive Summary**

The **AI Mock Interview Web Application** is a browser-based platform that lets job-seekers practise role-specific interviews, record their answers, and receive instant AI-generated feedback. Built with **React + Vite + TypeScript** on the front-end and **Firebase** services on the back-end, the app combines seamless user experience with reliable serverless infrastructure. Authentication is handled by **Clerk**, while answer evaluation leverages **GPT-4** via OpenAI or Google Gemini APIs.

**2 Project Objectives**

|  |  |  |
| --- | --- | --- |
| **#** | **Objective** | **Success Metric** |
| 1 | Provide realistic, time-boxed mock interviews | 5 role-specific questions delivered per session |
| 2 | Generate constructive, actionable feedback | ≥ 85 % of users rate feedback “Useful” in survey |
| 3 | Offer friction-free onboarding | OAuth sign-in < 10 s; drop-off rate < 5 % |
| 4 | Ensure data privacy & zero video storage | No webcam frames saved; only text stored |

**3 Scope of Work**

*In-scope*

* Web application (desktop & mobile responsive)
* Interview creation, execution, and results dashboard
* AI text evaluation and scoring
* Firebase Hosting, Auth, and (future) Functions

*Out-of-scope (v1)*

* Native mobile app
* Multi-language voice recognition
* HR analytics suite

**4 Target Users & Use-Cases**

1. **Fresh Graduates** – practise behavioural & tech rounds.
2. **Mid-Career Switchers** – review domain-specific questions.
3. **Career-Coaches / Bootcamps** – assign mock interviews to cohorts.

**5 System Overview**

**5.1 High-Level Architecture**

pgsql

CopyEdit

┌──────────┐ HTTPS ┌──────────────┐

│ Client │ ⇆ Browser ⇆│ Firebase │

│ (React) ├───────────────►│ Hosting │

└──────────┘ ├──────────────┤

▲ ▲ │ Auth (Clerk)│

│ │ ├──────────────┤

WebRTC │ Text │ Firestore │

│ │ └──────────────┘

│ │ ┌──────────────┐

│ └──────────────────►│ AI Scoring │

│ REST / gRPC │ (Cloud Fn) │

▼ └──────────────┘

User Camera /

Microphone

**5.2 Component Diagram**

* **AuthLayout** → wraps public / private routes
* **InterviewBuilder** → form wizard (role, description, stacks)
* **InterviewRunner**
  + QuestionPanel
  + WebcamRecorder (WebRTC)
  + SpeechToText (Browser API)
* **FeedbackPage** → Expected vs. User Answer, rating, tips
* **Dashboard** → card grid, CRUD actions

**6 Technology Stack**

|  |  |  |
| --- | --- | --- |
| **Layer** | **Technology** | **Why Chosen** |
| **UI** | React 18 + Vite | Fast HMR, modern build |
| **Language** | TypeScript | Type safety, refactor-friendly |
| **Styling** | Tailwind CSS | Utility-first, responsive out-of-box |
| **State** | Zustand | Lightweight global store |
| **Auth** | Clerk | Plug-and-play OAuth, JWT under the hood |
| **Back-end** | Firebase Hosting + Firestore | Serverless, easy CI/CD |
| **AI** | GPT-4 / Gemini via Cloud Functions | Natural-language grading |
| **Testing** | Jest + React Testing Library | Unit & component tests |
| **CI/CD** | GitHub Actions → Firebase Deploy | Automated build & release |

**7 Detailed Feature List**

1. **Social & Email Login**
2. **“Create Interview” Wizard**
3. **Dynamic Question Generation**
4. **Voice or Text Answer Modes**
5. **Real-Time Transcription (Web Speech API)**
6. **AI Scoring (0-10) + Colour-coded Rating Bar**
7. **Personalised Feedback Block**
8. **Results Dashboard** (delete, duplicate, restart)
9. **Dark-Mode Ready UI**
10. **Accessibility** – full keyboard nav, ARIA labels

**8 Data Flow & State Management**

1. InterviewBuilder dispatches **CREATE\_INTERVIEW** → state store → Firestore doc.
2. InterviewRunner subscribes to live doc; pushes each answer.
3. On completion, Cloud Function triggers **scoreInterview()** → writes feedback back to Firestore.
4. Dashboard listens with onSnapshot for real-time update.

**9 User-Interface Design**

|  |  |
| --- | --- |
| **Screen** | **Key UX Decisions** |
| **Sign-In** | Minimal, third-party OAuth buttons first; dev-mode banner. |
| **Builder** | Single-column form, ghost labels, progress breadcrumb. |
| **Runner** | Webcam placeholder with clear mic toggle; timer chip. |
| **Feedback** | Accordion per question: expected - your answer - tips. |
| **Dashboard** | Card layout, soft shadows, inline actions. |

**10 Implementation Highlights**

* **Lazy Load AI SDK** to keep initial bundle under 200 KB.
* **Custom Hook useRecorder()** abstracts WebRTC, provides blob ⇒ text.
* **Debounced Firestore Writes** every 3 s for near-real-time save without quota burn.
* **Tailwind Plugin** for brand palette & motion-safe animations.
* **Role-based Question Bank** stored in questions.json; fallback to GPT prompt when bank empty.

**11 Testing Strategy**

|  |  |  |
| --- | --- | --- |
| **Level** | **Tool** | **Sample Tests** |
| Unit | Jest | scoreInterview outputs score ∈ [0,10] |
| Component | React Testing Library | Render InterviewBuilder -> fill form -> expect submit handler |
| E2E (to-do) | Playwright | Login → create → run interview on staging |

**12 Security & Privacy**

* **No video frames stored**; only text transcripts saved.
* **Firestore rules** check auth.uid === doc.owner.
* **HTTPS-only** (automatic on Firebase).
* **Environment secrets** stored in GitHub-Actions encrypted vars.

**13 Performance & Scalability**

* Vite-driven code-splitting; critical path < 100 KB.
* Cloud Functions are stateless & scale to zero.
* Firestore auto-indexes for ownerId, createdAt.

**14 Deployment Pipeline**

1. **PR Merged → GitHub Actions**
2. pnpm install && pnpm build
3. firebase deploy --only hosting (preview)
4. Manual approval for production channel

**15 Limitations (v1)**

* English-only voice-to-text accuracy varies with accent.
* AI scoring may produce edge-case bias; manual override missing.
* No offline mode.

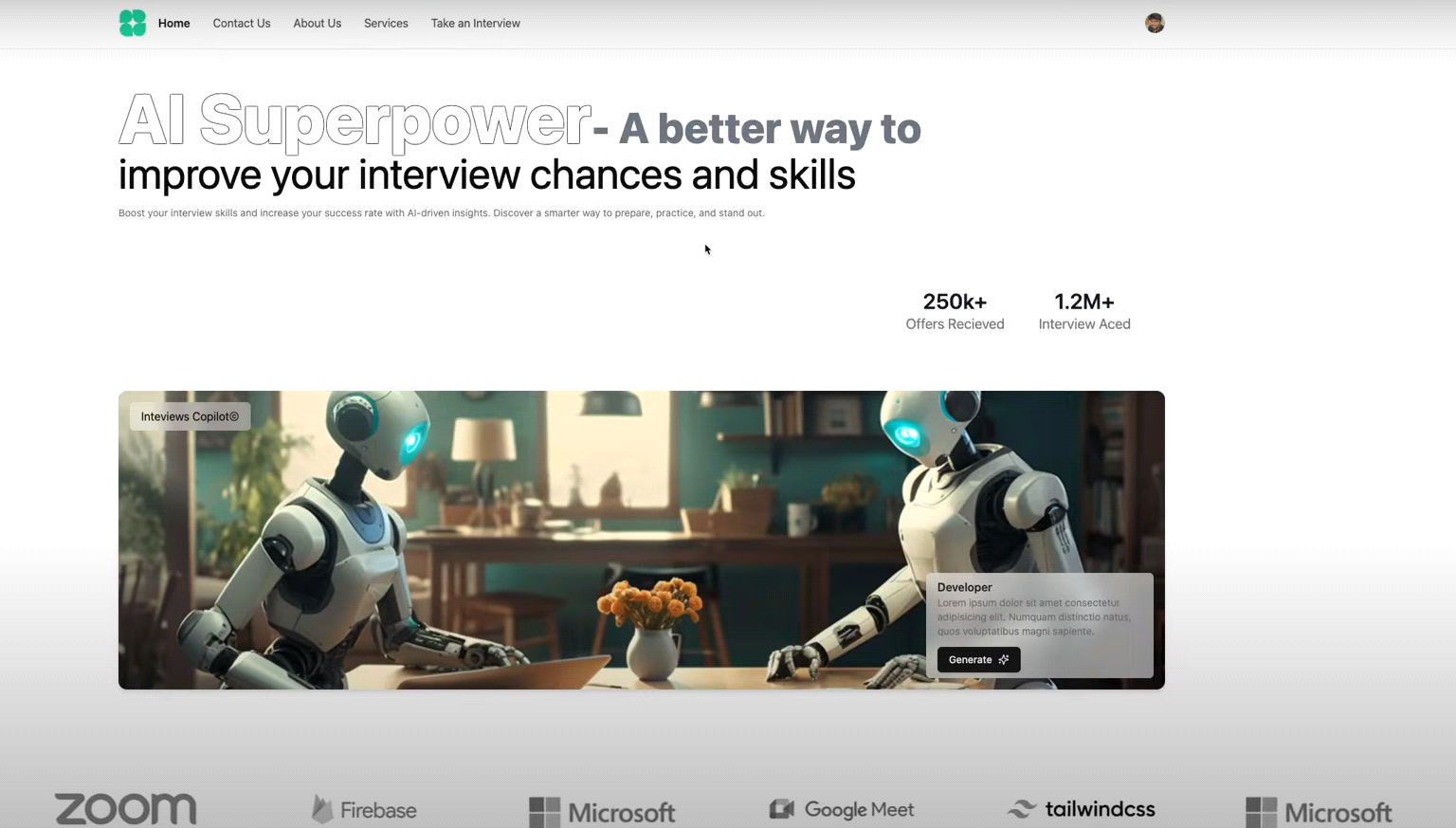
**16 Future Enhancements**

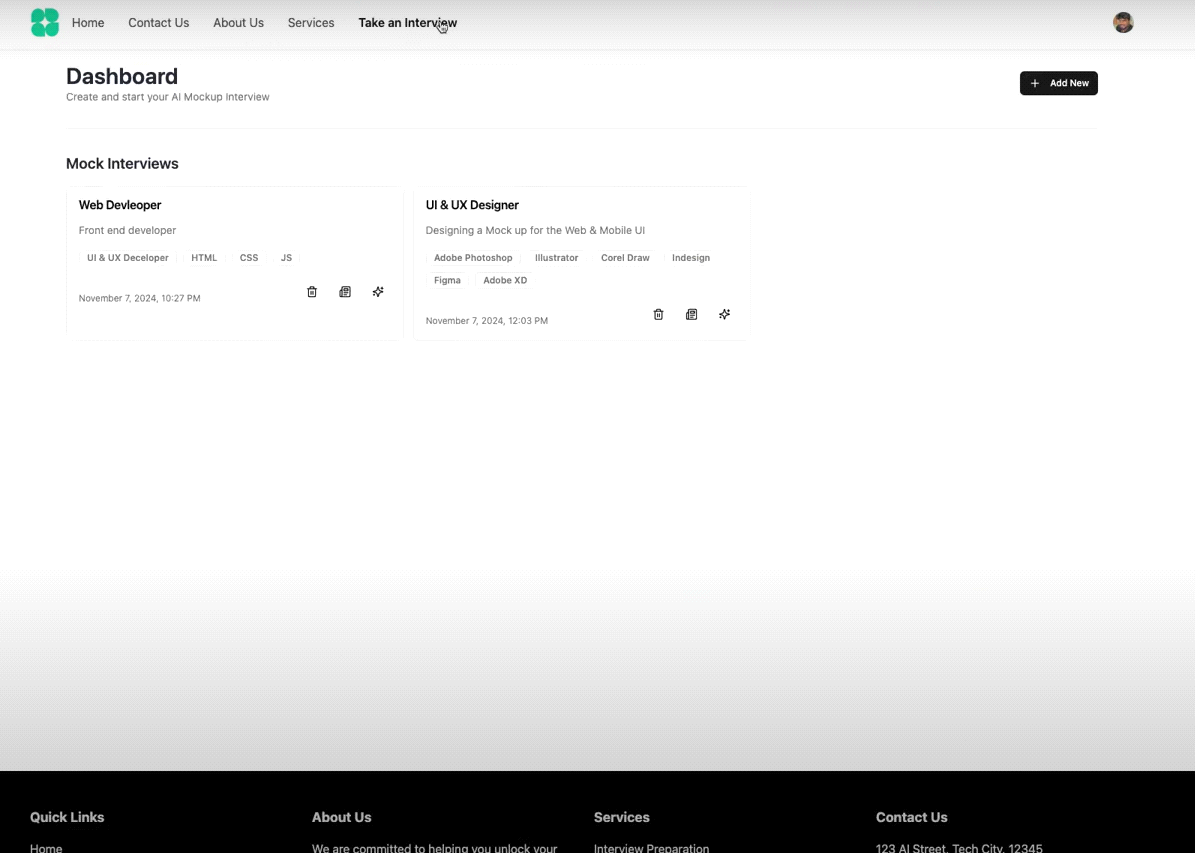
1. **Admin Portal** to curate custom question banks.
2. **Export PDF/Word Report** per interview (email attachment).
3. **Gamification** – streaks, leaderboards, shareable badges.
4. **Multilingual Support** – integrate Whisper or Azure STT.
5. **Mobile App** with React Native + Expo.

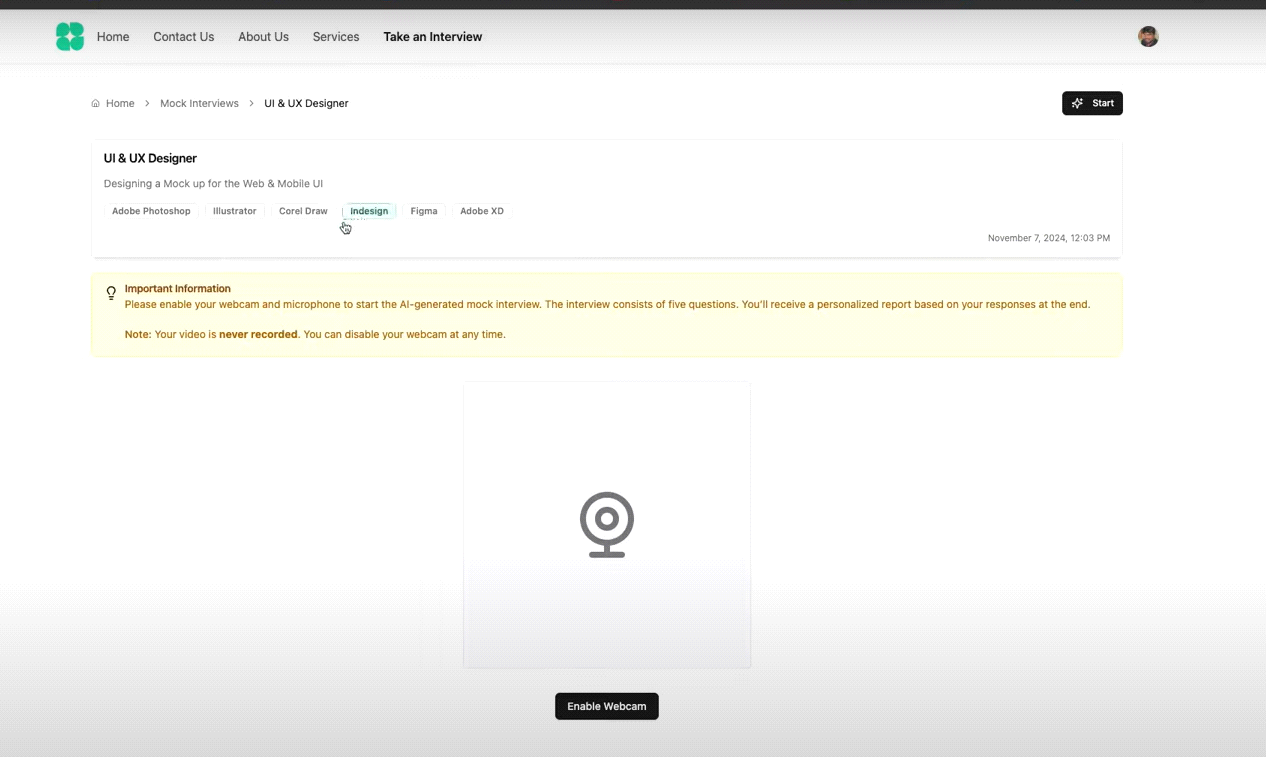
**17 Conclusion and UI**

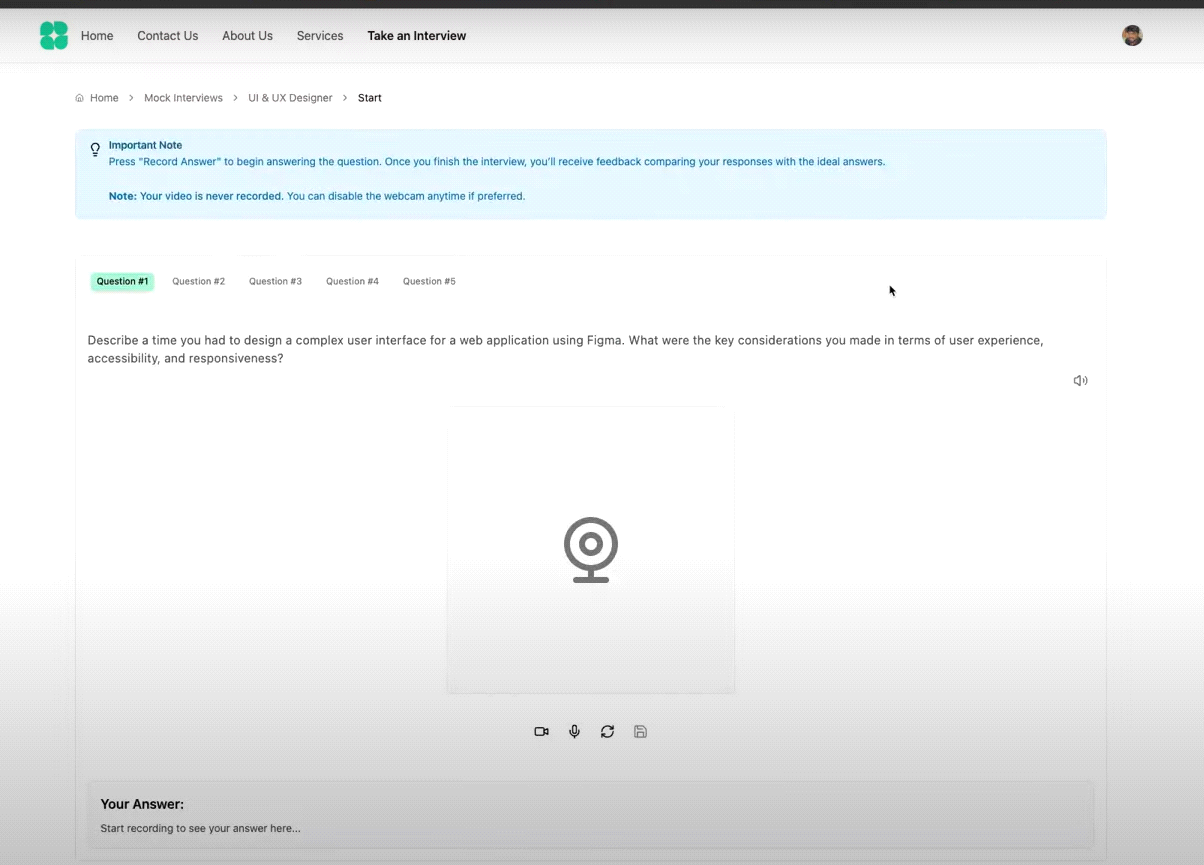
The AI Mock Interview application delivers a **streamlined, personalised, and privacy-conscious** interview practice experience. Its modern tech stack keeps the codebase maintainable while Firebase provides effortless scalability. With incremental enhancements, the platform can evolve into a comprehensive career-readiness suite.

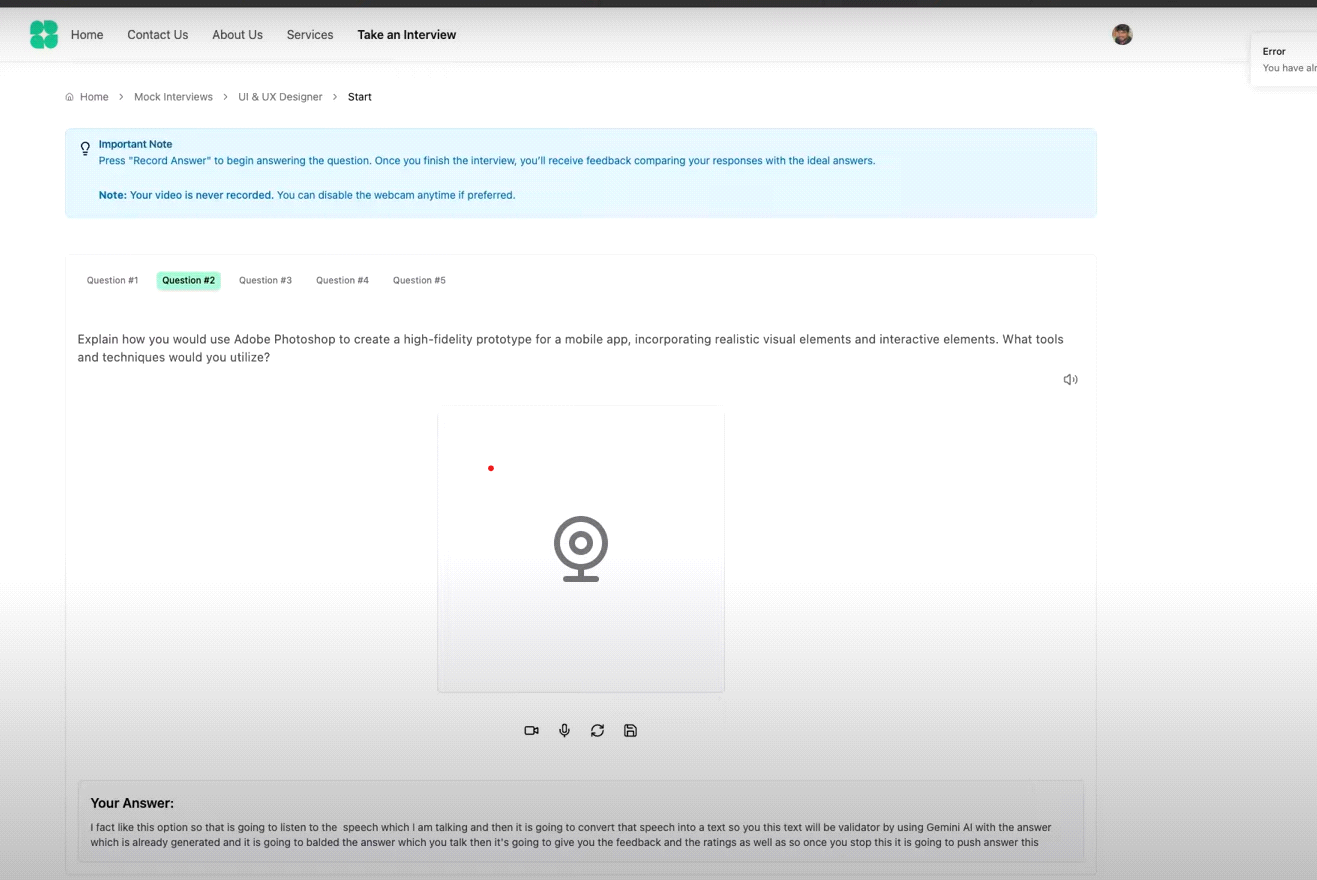
**User Interface:**

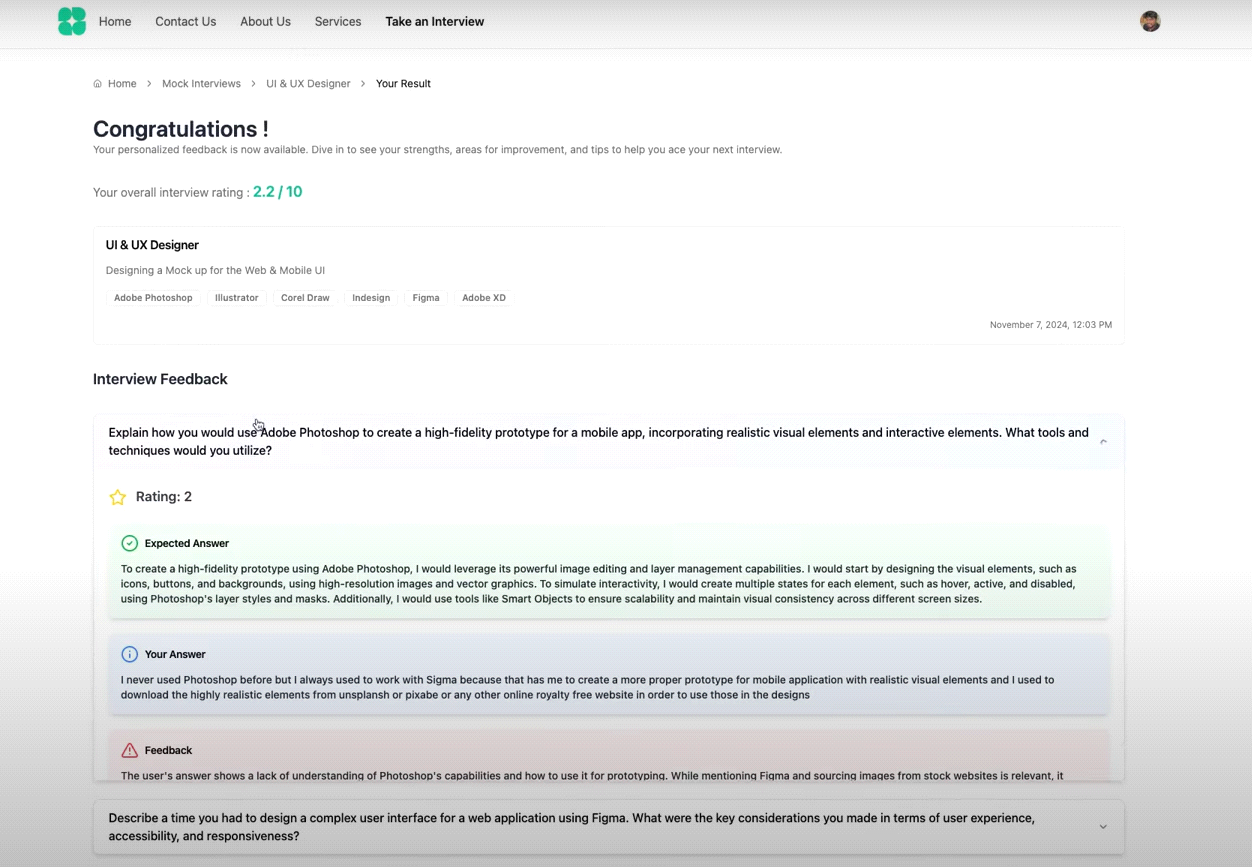


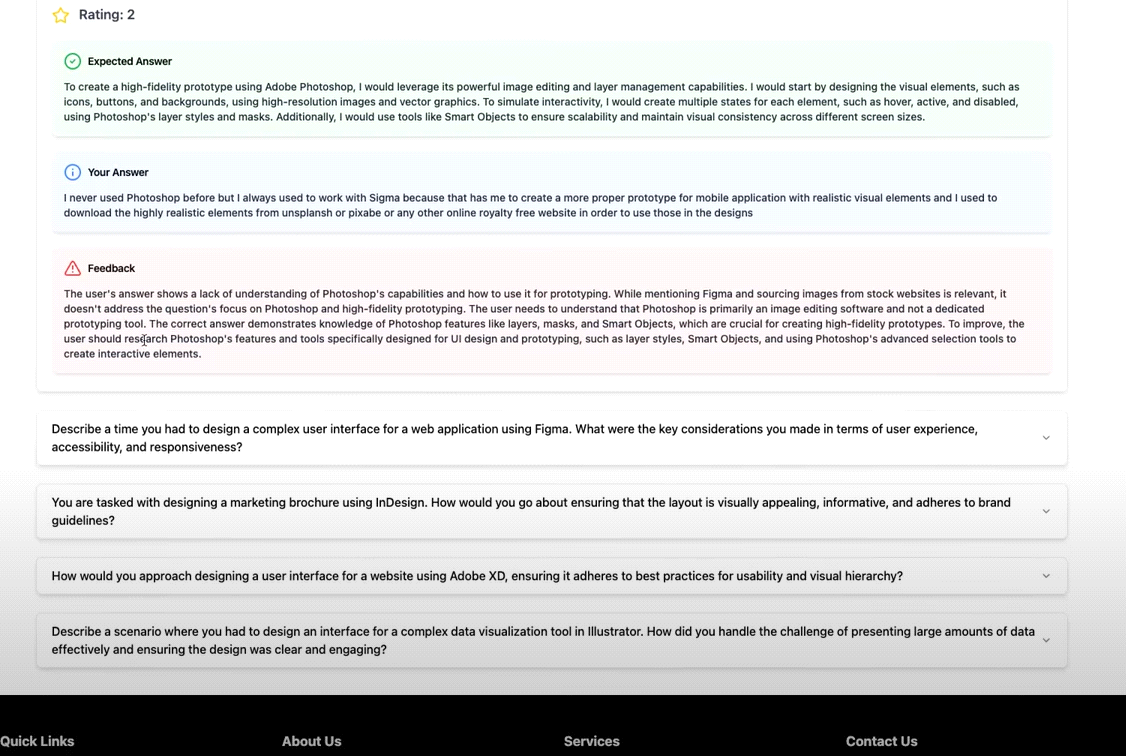


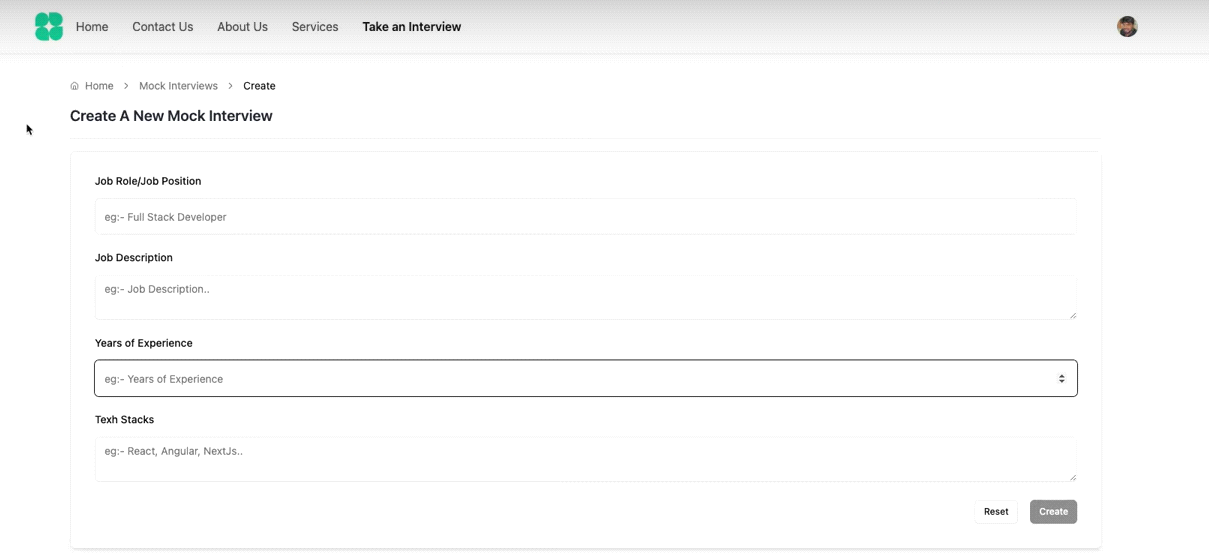


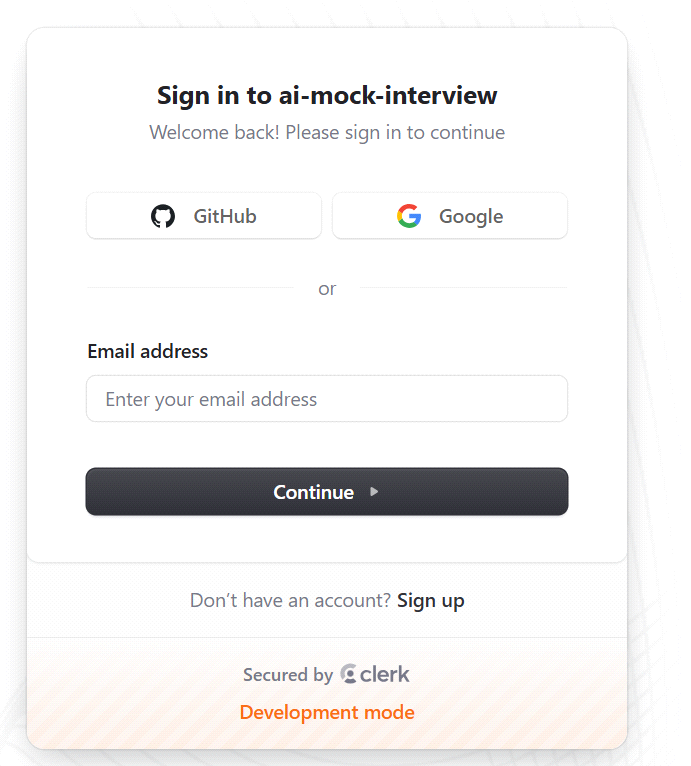












**18 Appendices**

**18.1 Simplified File Tree**

arduino

CopyEdit

ai-mock-interview/

├─ public/

├─ src/

│ ├─ components/

│ ├─ hooks/

│ ├─ pages/

│ ├─ services/

│ ├─ App.tsx

│ └─ main.tsx

├─ firebase.json

├─ vite.config.ts

└─ tailwind.config.js

**18.2 Key Code Snippet – AI Scoring Cloud Function**

ts

CopyEdit

import { onCall } from "firebase-functions/v2/https";

import { Configuration, OpenAIApi } from "openai";

export const scoreInterview = onCall(async (req) => {

const { qna } = req.data; // { question, answer }[]

const prompt = buildPrompt(qna);

const openai = new OpenAIApi(new Configuration({

apiKey: process.env.OPENAI\_KEY,

}));

const res = await openai.chat.completions.create({

model: "gpt-4o-turbo",

messages: [{ role: "user", content: prompt }],

});

return JSON.parse(res.choices[0].message.content);

});