AI Mock Interview Platform – Technical Summary

1. Problem Statement

Goal: Build an AI-powered interviewer application that can conduct mock interviews in real time, evaluate candidates' responses, and provide instant, personalized feedback to help them improve.

2. Approach and AI Components

2.1 Core AI Components

The system uses **Google Gemini (Generative AI)** for:

1. **Question Generation**
    - Role-specific prompts (e.g., Machine Learning Engineer, Data Analyst)
    - Context-aware follow-up questions based on previous answers
2. **Answer Evaluation**
    - AI analyses the user's answer and returns:
        - Communication Score
        - Technical Score
        - Confidence Score
        - Detailed feedback for each category
3. **Final Summary Generation**
    - After 10 questions, Gemini reviews full session history and generates:
        - Overall performance metrics
        - Strengths and weaknesses
        - Actionable improvement plan
        - Suggested learning materials

2.2 Speech-to-Text Integration

The frontend uses **Web Speech API** to allow voice answers during the interview.
Recognized text is shown in real time and submitted alongside typed answers.

3. Technical Architecture

3.1 Backend Architecture

- **FastAPI** handles REST endpoints:
    - /start-interview
    - /submit-answer
    - /final-feedback
- Stateless processing with a temporary sessions in-memory store
- Gemini AI wrapper (ai_client.py) handles all prompt communication
- CORS configured for deployed frontend
- Optional MongoDB schema: users, sessions, question-answer logs

3.2 Frontend Architecture

- **React + Vite** (SPA)
- Pages:
    - RoleSelection.jsx
    - Interview.jsx (typed + voice input)
    - Results.jsx (final summary)
- Global app state:
    - Session ID
    - Current question
    - Question count (max 10)
    - Last evaluation scores
    - Final report

4. Challenges and Mitigations

Challenge 1:LLM Cost & Latency

LLM calls must be optimized to avoid slow responses.

Mitigation:

Used Gemini Flash for fast inference; Kept prompts lightweight ; Cached session history instead of sending raw transcripts

Challenge 2 :Speech Recognition Browser Support

Web Speech API is not fully supported in all browsers.

Mitigation: Automatic fallback to text-only mode ; UI hides voice features where unsupported; Clear error handling when mic access fails

Challenge 3 :Maintaining Context Between Questions

AI requires history for meaningful follow-up questions.

Mitigation: Stored structured Q/A history (question + answer); Sent condensed history to Gemini for next-question generation

Challenge 4 :CORS and Deployment Issues

Frontend and backend run on different domains → CORS blocked requests.

Mitigation: Added explicit CORS middleware in FastAPI ; Allowed specific production origins (Render + Vercel)

Challenge 5 :Real-Time User Feedback

Immediate responsiveness is critical during interviews.

Mitigation: Non-blocking async calls ;Spinner/loader UI states ; Progressive question counter (1 to 10)

5. Roadmap to Final Build

Phase 1 (Completed for Prototype)

Backend core API
-AI question generation
-AI answer evaluation + scoring

-Frontend UI with navigation
-10-question interview limit
-Final feedback summary

Phase 2 (Planned for Final Round)

2.1 Backend Enhancements

- Integrate persistent **MongoDB** database
- Enable voice based answers through audio input
- Add user authentication
- Multi-round interview modes
- Resume parsing → personalized questions

2.2 Frontend Enhancements

- More polished, responsive UI
- Add interview timer, analytics dashboard
- History view: review all past answers

2.3 Real-Time Communication

- Add audio streaming
- Enable video interview mode (bonus)
- WebSocket integration for live updates

Phase 3 – Final Delivery

- Full deployment (Render + Vercel)
- Error logging + monitoring
- Performance improvements
- Final user testing and bug fixes
- Submission-ready presentation deck