System Requirements Specification (SRS): Valora

1. Introduction

Valora is a multimodal AI technical interviewer that helps students prepare for job interviews. It ingests a candidate's Resume (PDF) and Job Description to conduct a real-time vocal interview, analyzing technical accuracy, confidence, and body language.

1.1 Scope

- **User:** Students/Candidates.
- **Platform:** Web Application (optimized for Chrome/Edge).
- **Core Capabilities:** Resume Parsing, AI Question Generation, Voice Interaction, Visual Analysis, Progress Tracking.

---

2. System Architecture

The system follows a **Client-Server Architecture** but offloads heavy media processing (Speech-to-Text, Text-to-Speech) to the client (Browser) to reduce latency and cost.

2.1 High-Level Architecture Diagram

Code snippet

```
graph TD
  Client[Client (React Browser)]
  Server[Backend Server (Node/Express)]
  DB[(MongoDB)]
  AI[Google Gemini API]
  Auth[Clerk Auth]

  %% Data Flow
  Client -- "1. Upload Resume (PDF)" --> Server
  Server -- "2. Parse Text" --> Server
  Server -- "3. Prompt (Resume + JD)" --> AI
  AI -- "4. Question Text" --> Server
  Server -- "5. Question Text" --> Client
```

```
%% Client Side Processing
Client -- "6. Text-to-Speech (Voice)" --> User
User -- "7. Speech (Mic) + Video" --> Client
Client -- "8. Speech-to-Text" --> Client

%% Analysis Loop
Client -- "9. Submit Answer (Text + Image)" --> Server
Server -- "10. Analyze Answer" --> AI
AI -- "11. Feedback/Next Question" --> Server

%% Storage
Server -- "12. Save Results" --> DB
Client -- "13. Auth Token" --> Auth
```

---

3. Frontend Specifications

3.1 Tech Stack

- **Framework:** React.js (Vite build tool for speed).
- **Styling:** Tailwind CSS (fast UI) or Chakra UI (accessible components).
- **State Management:** React Context API (sufficient for hackathon complexity).
- **Media Libraries:**
    - react-webcam: For capturing visual snapshots.
    - jspdf: For generating reports client-side.
    - **Web Speech API:** window.SpeechRecognition & window.speechSynthesis.

3.2 Key Components & Pages

1. **Landing Page:** Hero section, Login (Clerk), "How it Works."
2. **Dashboard:**
    - ProgressChart: Recharts line graph.

    - HistoryList: List of past interview cards.
3. **Setup Room:**
    - ResumeUploader: Drag-and-drop zone.
    - JobDescriptionInput: Text area.
4. **Interview Room (The Core):**
    - AvatarCanvas: Visualizer that pulses when Valora speaks.
    - WebcamView: Picture-in-Picture user view.
    - TranscriptBox: Real-time text stream.
    - ControlBar: Mic toggle, "End Interview" button.
5. **Report View:** Displays the JSON result and "Download PDF" button.

---

4. Backend Specifications

4.1 Tech Stack

- **Runtime:** Node.js.
- **Framework:** Express.js.
- **Middlewares:** cors, multer (file handling), express.json.
- **Libraries:** pdf-parse, @google/generative-ai.

4.2 API Endpoints

Method	Endpoint	Description	Payload (Request)	Response
POST	/api/interview/init	Starts session, parses PDF.	file (PDF), jobDesc (String)	{ sessionId, firstQuestion }
POST	/api/interview/next	Submits answer, gets next Q.	sessionId, answerText, imageSnapshot (Base64)	{ nextQuestion, feedback }
POST	/api/interview/end	Generates final report.	sessionId, fullHistory	{ overallScore, metrics, pdfData }
GET	/api/user/history	Fetches past results.	userId (Header)	[{ date, score, summary } ...]

---

5. Database Design (MongoDB)

We need two primary collections.

5.1 Users Collection

Managed mainly by Clerk, but we reference the userId.

5.2 Interviews Collection

Stores the complete log of a session.

JSON

```
{
  "_id": "ObjectId",
  "userId": "String (Clerk ID)",
  "createdAt": "ISODate",
  "jobRole": "String",
  "resumeText": "String (Cached for reference)",

  "qa_log": [
    {
      "question": "Explain React Virtual DOM",
      "user_answer": "It is a copy of the real DOM...",
      "valora_feedback": "Correct, but mention diffing algorithm.",
      "emotion_detected": "Neutral"
    }
  ],

  "metrics": {
    "overall_score": 7,
    "technical_accuracy": 8,
    "confidence_score": 6
  }
```

}

---

6. Use Case Diagram

This diagram defines *who* does *what*.

Actors:

1. **Candidate (User)**
2. **Valora System (AI/Backend)**

Use Cases:

- **Candidate:**
    - Register/Login.
    - Upload Resume & Job Description.
    - **Start Interview:** Listen to AI, Speak Answer.
    - View Real-time Transcript.
    - **End Interview:** View Report, Download PDF.
    - View Dashboard/History.
- **Valora System:**
    - **Extract Context:** Parse Resume PDF.
    - **Generate Question:** Based on context + history.
    - **Analyze Input:** Process Text (Technical) + Image (Visual).
    - **Provide Feedback:** Generate scores and tips.
    - **Persist Data:** Save session to DB.

---

7. Process Flow Diagram (The Interview Loop)

This logic flow ensures you handle the conversation correctly.

1. **Start:**
    - User uploads PDF -> Backend Extracts Text -> Prompt Gemini -> **Return Question 1**.
2. **Turn Loop:**

    - **System:** Plays Audio (Question 1).
    - **User:** Listens -> Starts Speaking (Mic Active).
    - **Browser:** Converts Speech -> Text.
    - **Browser:** Captures Webcam Snapshot (Image).
    - **User:** Stops Speaking -> Sends { Text + Image } to Backend.
    - **Backend:** Sends { Resume + History + Answer + Image } to Gemini.
    - **Gemini:** Grades Answer -> Generates **Question 2**.
    - **Backend:** Returns Question 2 to Browser.
    - *(Repeat until 5 questions or User clicks End)*.
3. **End:**
    - Backend compiles all Q&A -> Gemini generates Final JSON Report -> Frontend renders PDF.

---

8. Non-Functional Requirements (Constraints)

- **Latency:** Audio response generation + Network trip must be under 3 seconds to feel "conversational."
- **Browser Compatibility:** Must be tested on Chrome/Edge (Web Speech API requirement).
- **Security:** Resumes are processed in memory (RAM) and not stored permanently on disk (privacy).
- **Scalability:** Client-side processing ensures server load is minimal (handles text only).