Here is the comprehensive Software Requirements Specification (SRS) document for **RuralMedAI**. This document is structured to serve as a context file for an LLM to generate production-ready code.

Software Requirements Specification: RuralMedAI

1. Project Overview

RuralMedAI is a real-time AI medical scribe designed for doctors in rural India. It listens to doctor-patient conversations in real-time, extracts structured medical data (demographics, symptoms, diagnosis), and auto-fills a clinical intake form. It eliminates manual data entry, allowing doctors to focus on diagnosis.

1.1 Core Philosophy

- **Voice-First:** No typing required during consultation.
- **Human-in-the-Loop:** AI drafts, Doctor approves.
- **Low-Latency:** Updates the UI immediately as information is spoken (Streaming JSON).

---

2. System Architecture & Tech Stack

2.1 Technology Stack

- **LLM Engine:** Google Gemini 2.0 Flash (Live API via WebSockets).
- **Backend:** Python 3.11+, FastAPI, websockets, google-genai SDK.
- **Frontend:** Node.js, React (Next.js 14+), Tailwind CSS, Lucide React (Icons).
- **Audio Pipeline:** Browser AudioWorklet (Float32 to Int16 PCM) to WebSocket to Gemini.

2.2 High-Level Data Flow

1. **Input:** User speaks into the browser microphone.
2. **Processing (Client):** AudioWorklet downsamples audio to 16kHz Mono PCM (16-bit).
3. **Transport:** Frontend streams base64-encoded PCM chunks via WebSocket to Backend.
4. **Proxy:** Backend forwards audio streams to Gemini 2.0 Flash Live Session.
5. **Inference:** Gemini extracts entities (e.g., "Name is Raju") and yields partial JSON.
6. **Update:** Backend pushes JSON chunks to Frontend; UI updates form fields in real-time.

---

3. Development Phases

Phase 1: Proof of Concept (The Hackathon Deliverable)

- **Live Form Filling:** Real-time extraction of Name, Age, Gender, Symptoms, and Vitals.
- **Streaming UI:** Form fields highlight and populate as the patient speaks.
- **Doctor Review:** Editable fields with a "Commit to Record" button.
- **Basic Summary:** "Catch-me-up" text generation at the end of the session.

Phase 2: Enhanced Features (Post-Hackathon)

- **Ayushman Bharat Check:** Rule-based logic checking eligibility criteria (Income/Caste) extracted from conversation.
- **Timeline View:** Visual history of patient visits.
- **Source Grounding:** Hovering over a form field plays the specific audio clip where the info was stated.
- **PDF Export:** Generating an official OP (Outpatient) slip.

---

4. Directory Structure & File Specifications

Plaintext

```
rural-med-ai/
├── backend/
│   ├── app/
│   │   ├── __init__.py
│   │   ├── main.py            # Entry point, WebSocket router, CORS setup
│   │   ├── core/
│   │   │   ├── config.py      # Env vars (GEMINI_API_KEY), Log config
│   │   │   └── schemas.py     # Pydantic models (PatientForm, MedicalNote)
│   │   ├── services/
│   │   │   ├── gemini_service.py # Manages Gemini Live Session & Audio I/O
│   │   │   └── prompt_eng.py   # System instructions for the "Scribe" persona
│   │   └── api/
│   │       └── routes.py      # HTTP endpoints for export/summary (non-streaming)
│   ├── requirements.txt
│   └── .env
├── frontend/
│   ├── src/
│   │   ├── app/
│   │   │   ├── page.tsx       # Main Dashboard (Split view: Form | Live Transcript)
│   │   │   └── layout.tsx     # Global providers
│   │   ├── components/
│   │   │   ├── LiveForm.tsx    # The auto-filling inputs with animation state
│   │   │   ├── AudioVisualizer.tsx # Canvas based waveform renderer
│   │   │   └── ActionButtons.tsx # Start/Stop/Approve controls
│   │   ├── hooks/
│   │   │   ├── useAudioStream.ts # Captures Mic, converts to 16kHz PCM
│   │   │   └── useSocket.ts    # Manages WS connection & message parsing
│   │   └── lib/
│   │       └── audio-utils.ts  # Low-level PCM conversion logic (Float32->Int16)
│   ├── public/
│   │   └── worklet.js         # AudioWorklet processor for smooth recording
│   ├── package.json
│   └── tailwind.config.ts
```

---

5. Technical Implementation Details (File-by-File)

5.1 Backend Logic

backend/app/main.py

- **Goal:** Initialize FastAPI and the WebSocket endpoint /ws/live-consultation.
- **Logic:** Accepts a WebSocket connection. Instantiates GeminiService. Enters a loop: Receive Audio Chunk to Send to Gemini to Receive Text/JSON from Gemini to Send to Client.

backend/app/services/gemini_service.py

- **Goal:** Interface with google.genai.
- **Key Config:**
  - Model: gemini-2.0-flash-exp
  - Config: response_modalities=["TEXT"] (We send audio, receive text).
  - **Crucial:** Must handle the session initialization to keep context alive.

backend/app/services/prompt_eng.py

- **Goal:** The "System Instruction".
- **Content:**"You are an expert medical scribe. Listen to the consultation. Your output must be strictly JSON. When you hear a medical entity, output: {'type': 'update', 'field': 'symptoms', 'value': 'fever and cough'}. If the conversation is casual, ignore it. Do not hallucinate values."

backend/app/core/schemas.py

- **Goal:** Define the data structure.
- **Python:**

```python
class PatientData(BaseModel):
    name: str | None
    age: int | None
    gender: str | None
    symptoms: List[str]
    diagnosis: str | None
    medications: List[str]
```

5.2 Frontend Logic

frontend/src/hooks/useAudioStream.ts

- **Goal:** Capture high-quality audio.
- **Logic:** Uses navigator.mediaDevices.getUserMedia. Adds a AudioWorkletModule.
- **Processing:** The worklet downsamples the browser's 44.1kHz/48kHz input to **16kHz**. It converts the 32-bit float array to a **16-bit Integer** (Int16) array. This is strictly required for Gemini.

frontend/src/components/LiveForm.tsx

- **Goal:** The "Magic" UI.
- **Logic:**
  - State: formData object.
  - Effect: Listens to useSocket messages. When { field: "age", value: 45 } arrives:
    1. Update formData.age.
    2. Trigger a CSS animation (glow green) on the Age input.

frontend/src/lib/audio-utils.ts

- **Goal:** Binary manipulation.
- **Function:** floatTo16BitPCM(float32Array)
  - Clamps values between -1.0 and 1.0.
  - Multiplies by 32767.
  - Returns an Int16Array buffer to be sent over the socket.

---

6. Execution Flow (The "Runbook")

Step 1: Backend Startup

Bash

```
cd backend
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
uvicorn app.main:app --reload --port 8000
```

- Server listens on ws://localhost:8000/ws/live-consultation.

Step 2: Frontend Startup

Bash

```
cd frontend
npm install
npm run dev
```

- Client opens on http://localhost:3000.

Step 3: Usage Scenario

1. Doctor clicks **"Start Consultation"**.
2. Frontend requests Mic permission to Connects WS to Backend.
3. Doctor asks: *"What is your name?"* Patient: *"My name is Ramesh."*
4. Audio flows to Gemini. Gemini detects name="Ramesh".
5. Gemini sends JSON {"field": "name", "value": "Ramesh"}.
6. The "Name" field on the UI populates instantly with "Ramesh".
7. Doctor clicks **"Stop"**. The socket closes.
8. Backend triggers a final summarization call (non-streaming) to generate the "Clinical Note".

---

7. Key Constraints & Edge Cases

1. **Audio Format Mismatch:** If the audio is not 16kHz Int16, Gemini will return garbage or error out. The audio-utils.ts is the single point of failure here.
2. **JSON Hallucination:** Gemini might output markdown text instead of JSON. The prompt_eng.py must enforce strictly structured output or the backend must use a regex cleaner before parsing.
3. **Network Drop:** If WS disconnects, the frontend should attempt one reconnection or save the audio locally to upload as a file (fallback).

8. Multilingual & Code-Switching Constraints

1. Doctor and patient may speak different languages or switch mid-sentence; the system must assume mixed-language conversations by default.
2. Supported languages must include Hindi, Tamil, Telugu, and English without manual selection.
3. Extracted medical entities must be normalized to English while preserving original spoken phrases for reference.

---

9. Voice Activity Detection (VAD) Constraints

1. Client-side VAD is mandatory to prevent streaming silence to the LLM.
2. Silence streaming increases token usage, latency, and hallucination risk.
3. Speech start/stop detection must control when audio chunks are transmitted.

---

10. Noise Handling Constraints

1. The system must assume noisy rural clinic environments as the default condition.
2. Client audio pipeline must apply a high-pass filter and noise gate before streaming.
3. Over-aggressive filtering must be avoided to prevent loss of clinically relevant speech.

---

11. Structured Output Reliability Constraints

1. Prompt-based JSON enforcement alone is unreliable and insufficient.
2. Gemini native structured output or function calling must be used to guarantee schema compliance.
3. Non-conforming or malformed outputs must be discarded rather than repaired heuristically.

---

12. Incomplete Clinical Data Schema Constraints

1. The intake schema must include vitals: blood pressure, temperature, pulse, and SpO_2.
2. Core clinical fields must include chief complaint, medical history, and allergies.
3. No clinical data may be inferred if it is not explicitly stated during the consultation.

---

13. Session Persistence Constraints

1. Live form data must be continuously mirrored to browser localStorage.
2. Browser crashes or reloads must restore the last known session state.
3. Audio capture must require explicit manual restart after recovery.

---

14. Security & PHI Constraints

1. All captured data must be treated as Protected Health Information (PHI).
2. WebSocket communication must use secure transport (WSS) only.
3. Audio and extracted medical data must never be logged in plaintext.

---

15. Consent & Data Retention Constraints

1. Recording must begin only after explicit doctor action with a visible recording indicator.
2. No background or passive audio capture is permitted.
3. By default, no long-term storage is allowed; data export is an explicit doctor-controlled action.