Software Requirement Specification (SRS)

Project Title: Multilingual Voice Assistant with Avatar & Streaming

1. Introduction

1.1 Purpose

The purpose of this project is to build an AI-powered **Voice Assistant** that can:

1.1 Listen to the user's voice,
2.1 Convert it into text (STT),
3.1 Translate into **English, Japanese, and Hindi**,
4.1 Respond in the selected language,
5.1 Provide responses via both **speech (TTS)** and a **2D/3D avatar** interface.

1.2 Scope

This system will support:

1.1 **Multi-language support (EN, JP, HI)**
2.1 **Real-time STT and TTS**
3.1 **Avatar-based responses** (2D Anime via Live2D or 3D VRM avatar with three.js)
4.1 **Streaming communication** using WebRTC/WebSockets
5.1 **Scalable backend** with FastAPI + Docker/Kubernetes
6.1 **Persistent session and storage** (Redis, Postgres, S3/MinIO)

2. Overall Description

2.1 System Perspective

The system consists of the following modules:

1.1 **Frontend (Web/PC/Mobile)**: Displays avatar, streams audio.
2.1 **Backend (FastAPI)**: Handles STT, TTS, translation, orchestration (LangChain).
3.1 **AI Models**: Whisper/Vosk/coqui for STT, Edge/Coqui TTS for TTS.
4.1 **Storage**: Postgres for structured data, S3/MinIO for audio files.
5.1 **Deployment**: Dockerized, scalable with Kubernetes.

2.2 User Classes and Characteristics

1.1 **End Users**: Interact with the voice assistant.
2.1 **Developers/Admins**: Configure backend, manage sessions and models.

2.3 Operating Environment

1.1 Backend: Python (FastAPI)
2.1 Frontend: Browser (WebRTC, three.js, Live2D SDK)
3.1 Deployment: Cloud-native (Docker, Kubernetes)

3. Functional Requirements

1.1 **Voice Capture & STT**

a. The system must record user voice in real-time.

b. Convert speech to text using Whisper/whisperx/Vosk/Coqui STT.

2.1 **Language Translation**

a. System must translate text into English, Japanese, and Hindi.

3.1 **Assistant Response Generation**

a. Generate responses via LangChain + LLM.

4.1 **TTS (Text to Speech)**

a. Convert response text into speech using Coqui TTS / Edge TTS / Azure / Google.

5.1 **Avatar Rendering**

a. Option A: 2D avatar via Live2D Cubism SDK.

b. Option B: 3D avatar via three.js + three-vrm.

6.1 **Streaming**

a. Use WebRTC/WebSocket for real-time audio streaming.

7.1 **Storage**

a. Store conversation logs in Postgres.

b. Store audio/video files in S3/MinIO.

8.1 **Session Management**

a. Use Redis for managing session states and streaming buffers.

4. Non-Functional Requirements

I. **Performance**: Response latency < 1.5s.

II. **Scalability**: Support concurrent users via Kubernetes scaling.

III. **Security**: Encrypted communication (TLS), authentication for users.

IV. **Reliability**: Must handle reconnection in case of dropped sessions.

5. System Architecture (High-Level)

Workflow:

1. User speaks → Captured by WebRTC → Sent to Backend.
2. Backend STT → Translates → Generates response (LangChain).
3. Response → TTS → Audio + Avatar animation.
4. Avatar displays response with lip-sync.
5. Storage of logs & audio in Postgres/S3.

6. Appendix - Suggested Libraries & Tools (quick list)

1.1 **LangChain** — Orchestration for prompt + tool calling.

2.1 **FastAPI** — Python backend.

3.1 **Whisper / whisperx / Vosk / coqui STT** — options for STT.

4.1 **Coqui TTS / Edge TTS / Azure / Google** — options for TTS.

5.1 **Live2D Cubism SDK** — 2D anime avatar.

6.1 **three.js + three-vrm** — 3D avatar.

7.1 **WebRTC / WebSocket** — audio streaming.

8.1 **OpenCV** — optional: audio-video processing, avatar preprocessing.

9.1 **Redis** — session state & streaming buffers.

1.1 **Postgres / S3 / MinIO** — storage.

2.1 **Docker / Kubernetes** — deployment.