# Business Document

## Voice Assistant Application

**1. Executive Summary**

**Purpose**:  
This document describes the technical and functional architecture of a voice assistant application designed to convert spoken language into text, process queries using AI, and deliver intelligent responses.

**Key Features**:

* Voice recording via browser
* Audio processing (resampling, WAV conversion)
* Integration with Google Speech-to-Text and Gemini AI
* Error handling and security
* Chat-style response history

**2. Technical Architecture**

**Frontend (Angular)**

* **Role**: Handles user interactions, audio recording, and UI.
* **Key Components**:
  + **AudioWorklet**: Records audio at 44.1kHz and sends chunks to the backend.
  + **Resampler.js**: Converts audio to 16kHz for Google Speech-to-Text compatibility.
  + **Chat Interface**: Displays conversation history with auto-scroll.

**Backend (Spring Boot)**

* **Role**: Processes audio, integrates with Google Cloud APIs, and manages security.
* **Key Components**:
  + **/transcribe Endpoint**: Accepts WAV files, returns AI-generated responses.
  + **Google Speech-to-Text**: Converts audio to text.
  + **Gemini API**: Generates responses using AI.

**Third-Party Services**

* **Google Cloud Platform (GCP)**:
  + **Speech-to-Text API**: For audio transcription.
  + **Vertex AI**: For Gemini model integration.

**3. Component Breakdown**

**A. Frontend Components**

1. **Recording Module**
   * Uses the Web Audio API to capture microphone input.
   * **Worklet Processor**: Processes audio in real-time to avoid main thread blocking.
   * **Resampling**: Reduces audio from 44.1kHz to 16kHz.
2. **Audio Conversion**
   * Converts raw audio data to WAV format with proper headers.
   * **Example**:

javascript

**const** wavBuffer = **this**.encodeWAV(audioData); *// Creates WAV-compliant bytes*

1. **HTTP Service**
   * Sends WAV blobs to the backend via POST requests.
   * Handles errors (e.g., network failures, invalid audio).
2. **Chat Interface**
   * Displays messages in a scrollable container.
   * New messages appear at the bottom automatically.

**B. Backend Components**

1. **SpeechController**
   * **Endpoint**: POST /transcribe
   * **Process**:
     1. Receives WAV file.
     2. Sends to Google Speech-to-Text.
     3. Forwards text to Gemini AI.
     4. Returns response to frontend.
2. **GeminiService**
   * **Dependency**: VertexAI client (configured with project ID).
   * **Method**:

java

**public** String processQuery(String userInput) {

**return** Gemini.generateResponse(userInput);

}

1. **Security Configuration**
   * **Secrets Management**: Google credentials stored in environment variables (not in code).
   * **CORS**: Allows requests only from http://localhost:4200.

**4. Data Flow**

1. **User Speaks** → Angular records audio (44.1kHz).
2. **Resampling** → Audio converted to 16kHz.
3. **WAV Conversion** → Frontend sends blob to backend.
4. **Backend Processing**:
   * Google Speech-to-Text → Text.
   * Gemini AI → Response.
5. **Response** → Displayed in Angular chat.

**5. Error Handling**

| **Error Type** | **Resolution** |
| --- | --- |
| IOException (Spring Boot) | Removed unnecessary try-catch blocks in GeminiConfig.java. |
| PERMISSION\_DENIED (GCP) | Ensure Google Cloud APIs are enabled and credentials are valid. |
| HTTP 500 (Backend) | Check server logs for Google API errors or invalid audio formats. |

**6. Security Measures**

1. **Credentials**:
   * Never commit \*.json keys to Git (use .gitignore).
   * Configure via environment variables:

bash

export GOOGLE\_APPLICATION\_CREDENTIALS="/path/to/key.json"

1. **HTTPS**: Enforced in production to encrypt data.
2. **Input Validation**: Rejects non-WAV files.

**7. Future Enhancements**

| **Feature** | **Description** |
| --- | --- |
| Multi-language Support | Add language dropdown to support Spanish, French, etc. |
| Voice Synthesis | Convert text responses to speech using Google Text-to-Speech. |
| User Authentication | Add OAuth2/login to track user-specific history. |

**8. Glossary**

* **AudioWorklet**: Browser API for background audio processing.
* **Resampling**: Changing audio sample rate (e.g., 44.1kHz → 16kHz).
* **WAV Header**: Metadata describing audio format (sample rate, channels).