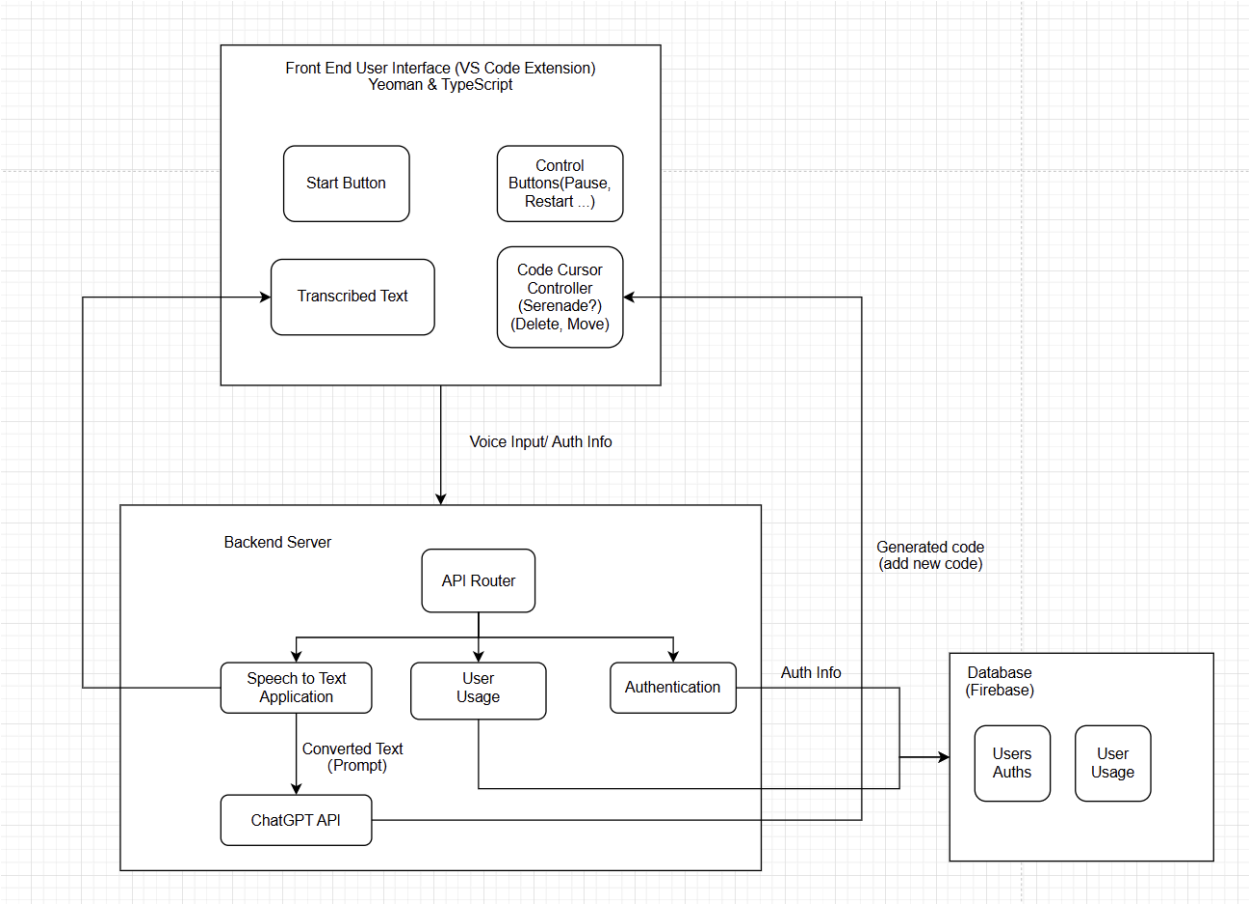
Software Architecture Document

1. Introduction

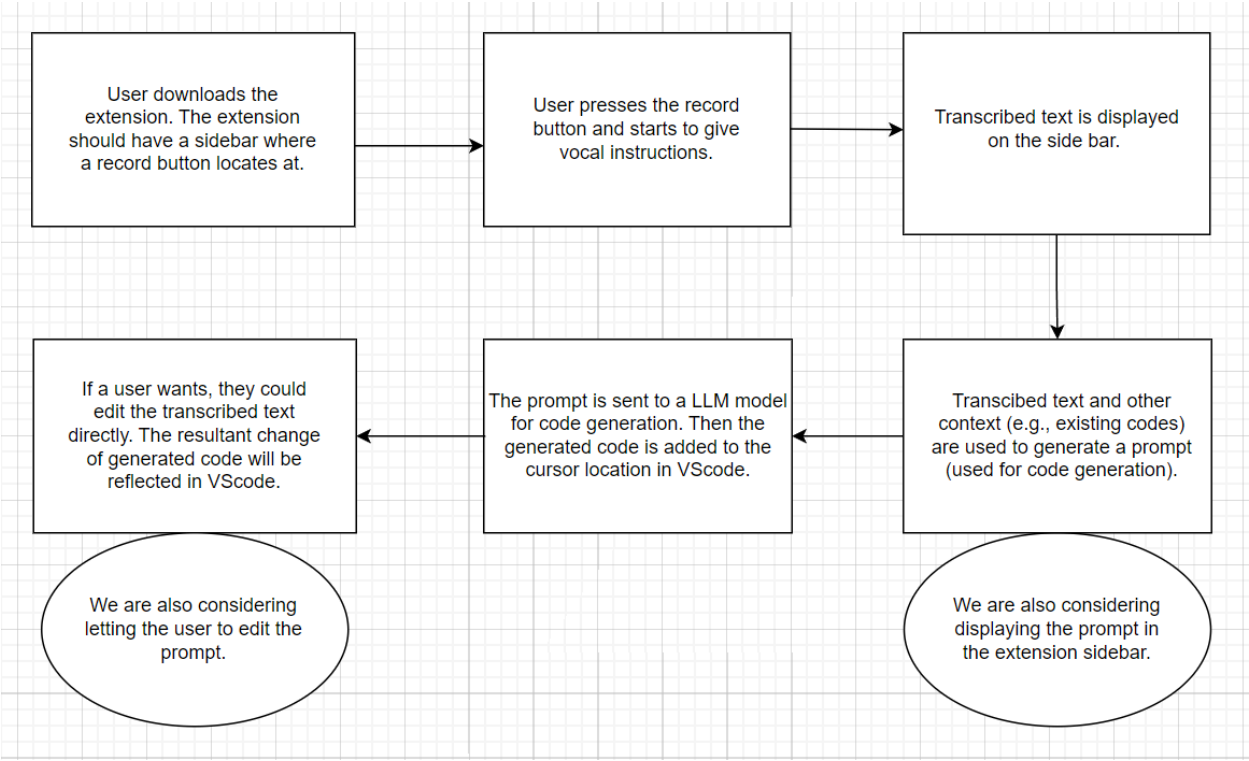
This document explains the architecture for a VSCode extension with the functionality of turning users' voices to code. The architecture involves multiple layers, including the frontend VSCode extension, backend server handling speech-to-text conversion and LLM APIs interactions.

The graph below shows the rough layout of current software architecture:



Architecture Diagram

The diagram below shows the general workflow for this project.



Workflow diagram

2. Layer 1: Frontend

The frontend component is responsible for gathering user input (voice command), managing interactions with the backend, and displaying the generated code within VSCode.

2.1 Technology Stack:

1) Yeoman, TypeScript, VSCode Extensions API

2.2 Features:

1) **Collect User Input and Context:** Gather user voice input, current code context, cursor position, and any open tabs.
2) **User Authentication:** If needed, send user authentication data to the backend.
3) **Communicate with Proxy Server**: Send user input and context to the backend server for code generation.
4) **Display and Insert Code:** Receive the generated code from the backend, display it to the user, and insert it where the cursor is located.
5) **Real-Time Feedback:** Show transcribed text before it's converted into code for user confirmation.

2.3 Frontend Workflow

1) User presses the start button to begin recording.
2) Voice input, along with the current code context, is sent to the backend for processing.
3) Transcribed text is returned to the frontend and displayed in the extension for review.
4) Generated code is returned to the frontend and inserted into the editor.

5) Users can control the session via control buttons (pause, restart, etc.).

3. Layer 2: Proxy Server

The backend processes voice input, generates code, and manages user data. It utilizes RESTful APIs to communicate with the frontend and connects to a third party model for speech-to-text and code generation.

3.1 Technology Stack:

1) Node.js

3.2 Features:

- **Speech-to-Text:** Converts voice input into text.
- **Prompt Engineering:** Does the prompt engineering logic (combining speech and other context (e.g., existing code) into a prompt to send to the model). Sends prompt to model and then (processes) and sends output to frontend
- **Interchangeable Model Interface:** Has an interchangeable model interface (factory pattern) to allow us to easily change which model we send the prompt to.
- **User Data Tracking:** Collects user usage data and sends it to database

4. Layer 3: Database

The database stores user data, authentication details, and usage metrics. This data can be used for future improvements, feature analysis, or debugging during development.

4.1 Technology Stack:

1) Firebase

4.2 Features:

1) **User Authentication:** Stores user authentication information.
2) **User Behaviour Tracking:** Collects usage data, such as session length, command frequency.
3) **Development Debugging:** Stores generated code and prompts temporarily during the development phase for debugging purposes.

5. Key Architectural Decisions

5.1 Handling Cursor Logic

Instead of rewriting the whole file, the system will modify only the code around the cursor. It will focus on the current function or block, generating code just for that part.

5.2 Deletion Logic

Using voice commands like "delete method" or "delete line" to perform deletion operations. The system will identify and remove the specific method or line based on the cursor position.