Software Design Specification

<Voice Coder>

Project Code: VC-WA01

Internal Advisor: Mr. Mudassar Ali Zaidi

External Advisor:

Project Manager: Dr. Muhammad Ilyas

Project Team: Alina Ahmed (S044)
Maryum Javed(S009)
Laiba Humayun(S031)

Submission Date: December 15, 2025

_____

Project Manager's Signature

Document Information

Category	Information
Customer	UOS – Department of Computer Science
Project	<Voice coder>
Document	Software Design Specification
Document Version	1.0
Identifier	<PGBH01-2025-DS>
Status	Draft
Author(s)	Alina Ahmed Maryum Javed Laiba Humayun
Approver(s)	PM
Issue Date	Sept. 15, 2025
Document Location	
Distribution	1. Advisor 2. PM 3. Project Office

Definition of Terms, Acronyms and Abbreviations

This section should provide the definitions of all terms, acronyms, and abbreviations required to interpret the terms used in the document properly.

Term	Description
ASP	Active Server Pages
DD	Design Specification
API	Application programming Interface
ERD	Entity relationship diagram
NLP	Natural Language processing
CORS	Cross-origin resource sharing
UI	User Interface
UML	Unified Modeling Language

Table of Contents

1. Introduction

1.1 Purpose of Document

*This Design Document provides a comprehensive architectural blueprint and technical specification for the **VoiceCoder** web application. The document serves as a foundational reference for developers, stakeholders, and academic evaluators to understand the system's architecture, design decisions, and implementation strategy. It outlines the complete technical vision, ensuring consistent development across all team members and providing clear guidelines for future enhancements.*

1.2 Project Overview

VoiceCoder is an innovative web-based application designed to enable programming through voice commands, specifically targeting the C++ programming language. The system converts spoken natural language into syntactically correct C++ code, providing an accessible programming environment for individuals with physical disabilities, repetitive strain injuries, or those seeking hands-free coding capabilities. The application operates entirely within modern web browsers, requiring no local installation, and leverages cloud-based services for speech recognition and code compilation.

Core Objectives:

- *Provide voice-controlled C++ coding capability*
- *Create an inclusive programming environment*
- *Implement real-time speech-to-code conversion*
- *Offer seamless code compilation and execution*
- *Maintain cross-browser compatibility and accessibility*

1.3 Scope

In-Scope Features:

- *Voice-to-text conversion for C++ programming commands*
- *Real-time code generation and editing interface*

- *Cloud-based C++ code compilation and execution*
- *Basic C++ language constructs support (variables, loops, conditionals, functions)*
- *User authentication and code saving functionality*
- *Cross-platform web browser compatibility*
- *Responsive user interface design*
- *Error handling and user feedback mechanisms*

Out-of-Scope Features:

- *Support for programming languages other than C++*
- *Advanced IDE features (debugging, version control, profiling)*
- *Offline functionality without internet connectivity*
- *Complex C++ features (templates, advanced OOP, metaprogramming)*
- *Natural language understanding beyond pre-defined command patterns*
- *Mobile application development*
- *Real-time collaborative coding features*

2. DESIGN CONSIDERATIONS

2.1 Assumptions and Dependencies

This system assumes users will run VoiceCoder on modern browsers supporting JavaScript and Web Speech API (Google, 2024) [1]. Users are expected to speak clear English commands since the system's current speech model only supports English. A stable internet connection is required as the compiler relies on cloud-based services (Paiza.io, 2024) [2]. It is further assumed that the user has a working microphone and basic knowledge of C++ syntax.

Dependencies include:

• Web Speech API for real-time speech-to-text (MDN, 2024) [4]
• Online compiler APIs such as Paiza.io or JDoodle (JDoodle, 2024) [3]
• A functioning code editor component
• Network availability for both recognition and compilation tasks

2.2 Risks and Volatile Areas

The application is at risk if browsers change or deprecate the Web Speech API (Google, 2024) [1]. Environmental noise can distort voice recognition accuracy. Compiler APIs may introduce rate limits or service downtimes (Paiza.io, 2024) [2].

Volatile areas include API endpoint updates, voice command expansion, and UI modifications depending on future user testing.

3. System Architecture

3.1 System Level Architecture

Overall System View:
The VoiceCoder system follows a ***three-tier web application architecture*** *with clear separation between presentation, business logic, and data processing layers. The architecture is designed for scalability, maintainability, and optimal user experience.*

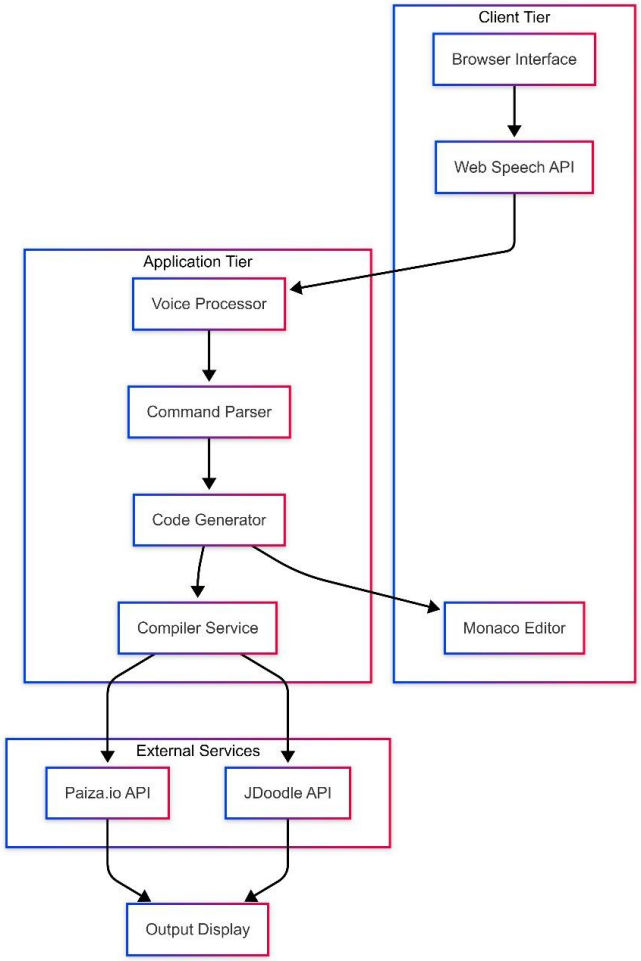


Figure 3.1: *VoiceCoder System Architecture Diagram showing the three-tier structure with client, application, and external service layers.*

Component Interactions:

1. ***User Interface Layer:*** *HTML5, CSS3, JavaScript with Monaco Editor integration*
2. ***Voice Processing Layer:*** *Web Speech API for speech recognition*
3. ***Business Logic Layer:*** *Command parser and code generator*
4. ***External Services Layer:*** *Cloud compilation APIs for code execution*
5. ***Data Management:*** *Local storage for user preferences and code snippets*

Communication Protocols:

- ***HTTP/HTTPS:*** *For all client-server communications*
- ***WebSocket:*** *For real-time voice streaming (future enhancement)*
- ***REST API:*** *For integration with external compilation services*
- ***Local Storage API:*** *For client-side data persistence*

Deployment Architecture:

- ***Frontend Deployment:*** *Static hosting on GitHub Pages or similar service*
- ***Backend Services:*** *Serverless functions (AWS Lambda / Google Cloud Functions)*
- ***External Dependencies:*** *Cloud compiler APIs with rate limiting considerations*
- ***CDN Integration:*** *For optimized asset delivery and reduced latency*

Scalability Considerations:

- *Stateless architecture for horizontal scaling*
- *Caching mechanisms for frequently used code templates*
- *Load balancing for compilation service requests*
- *Database sharding for user data management (future)*

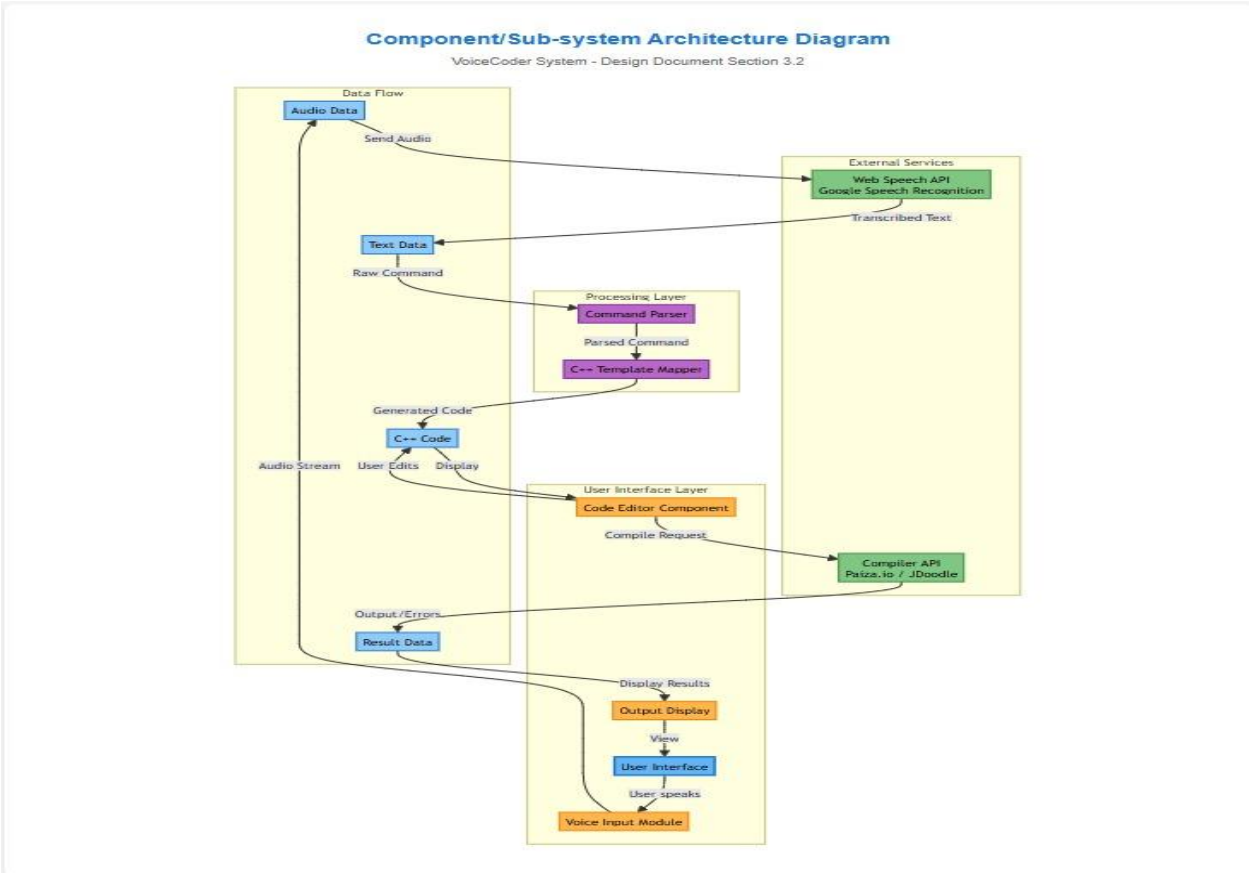
Security Architecture:

- *HTTPS enforcement for all communications*
- *Input validation and sanitization*
- *CORS policy implementation for API security*
- *Rate limiting to prevent abuse*
- *Secure storage of user preferences*

3.2 COMPONENT / SUB-SYSTEM ARCHITECTURE

VoiceCoder is composed of multiple interacting components. The Voice Input Module captures microphone audio, which is forwarded to the Speech Recognition Component powered by Web Speech API (Google, 2024) [1]. The Command Parser processes spoken text and maps it to structured C++ templates.
The Code Editor Component allows users to view, edit, and structure the generated code. The Compiler Integration Module sends the code to online compilers like Paiza.io or JDoodle for execution (JDoodle, 2024) [3].
Finally, the Output Display presents the compiled results or errors.

- *Component Architecture Diagram*:



3.3 Sub-Component / Sub-Module Level Architecture

The Voice Coding system is decomposed into granular sub-modules to ensure modularity, maintainability, and extensibility. Each sub-module performs a clearly defined responsibility, interacting through well-structured interfaces. This enables efficient development and easier future upgrades without impacting the entire system.

A. Speech Processing Sub-Modules

These modules together manage the voice input pipeline:

1. Audio Capture Module

- *Captures real-time audio through the browser microphone.*
- *Performs noise filtering and gains control where possible.*
- *Sends raw audio chunks to the Web Speech API for transcription.*

2. Speech Recognition Engine (Web Speech API Wrapper)

- *Converts audio into text streams.*
- *Handles recognition events (start, end, error, result).*
- *Returns recognized phrases for parsing.*

3. Command Pre-Processor

- *Cleans, normalizes, and tokenizes recognized text.*
- *Handles variations:*
  "for loop", "four loop", "create for loop", "make loop".

B. Parsing & Code Generation Sub-Modules

1. Parser Core

- *Maps processed text to C++ syntax templates.*
- *Uses rule-based command dictionaries drawn from SRS constructs.*
- *Ensures syntactic correctness by validating commands.*

2. Semantic Interpreter

- *Handles variable assignments, conditions, iterative ranges.*
- *Example:*
  "for i from 0 to 10" → detects variable = i, start = 0, end = 10.

3. Code Construction Engine

- *Builds formatted C++ blocks.*
- *Maintains indentation and scope levels.*
- *Appends generated code to the Monaco editor.*

C. Coding Interface Sub-Modules

1. Monaco Rendering Module

- *Displays generated C++ code.*
- *Manages syntax highlighting, indentation, cursor management.*
- *Supports user edits before compilation.*

2. Editor Interaction Layer

- *Bridges voice-generated code and editor operations.*
- *Handles undo/redo, line insertion, block removal.*

D. Compilation & Output Sub-Modules

1. Compiler API Handler

- *Sends C++ code to Paiza.io / JDoodle.*
- *Ensures secure HTTPS requests and error handling.*

2. Execution Manager

- *Waits for API response.*
- *Handles runtime errors, compiler errors, and infinite loops safely.*

3. Output Renderer

- *Displays program results in the output console.*
- *Formats warnings, errors, and runtime responses.*

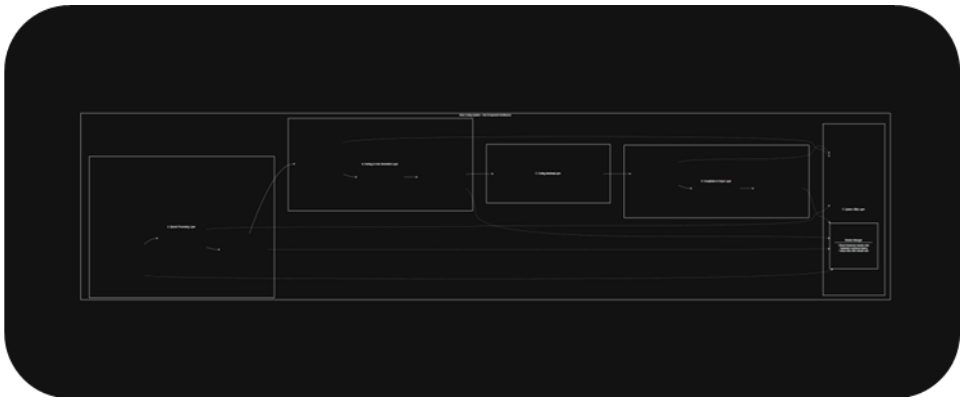
E. System Utility Sub-Modules

1. Error Logging & Handling Module

- *Captures parser errors, recognition issues, network failures.*
- *Displays user-friendly messages.*

2. Session Manager

- *Maintains temporary data (recognized commands, logs).*
- *Clears data after each session for security.*



4. Design Strategies

The design of Voice Coding follows strategic decisions to enhance system quality attributes such as performance, usability, scalability, and safety.

4.1 Strategy 1 : Modular, Component-Based Architecture

Reasoning:
The system integrates several external technologies (Web Speech API, Monaco Editor, compiler APIs) which may evolve over time. A modular design isolates each subsystem, making future replacement or updates easy.

Trade-offs:
Higher initial planning time, but long-term flexibility.

4.2 Strategy 2 : Event-Driven User Interface

Reasoning:
Speech recognition, parsing, and compilation are all asynchronous operations. An event-driven model ensures smooth handling of recognition events, API callbacks, and editor updates.

Benefits:

- *Zero UI freezing*
- *Real-time feedback*
- *Clean separation of UI and logic*

4.3 Strategy 3 : Rule-Based Command Parsing

Reasoning:

Instead of complex NLP models, rule-based parsing ensures predictable and deterministic code generation aligned with C++ syntax taught in academic settings.

Benefits:

- *High accuracy for programming-specific commands*
- *Reduced complexity*
- *Easy to expand with new rules*

4.4 Strategy 4 : Cloud-Sandboxed Execution

Reasoning:
User-generated C++ code can be unsafe. Running it in secure cloud sandboxes prevents system-level threats.

Benefits:

- *No local machine access*
- *Safe handling of malformed or malicious code*
- *Zero installation required*

4.5 Strategy 5 : Accessibility-First UI Design

Reasoning:
Target users include individuals with RSI and physical disabilities.

Design choices:

- *Large buttons (Start Recording, Stop, Compile)*
- *High-contrast editor theme*
- *Voice-driven actions with minimal manual input*

5. Detailed System Design

5.1 Class Diagram

Core Classes Structure:

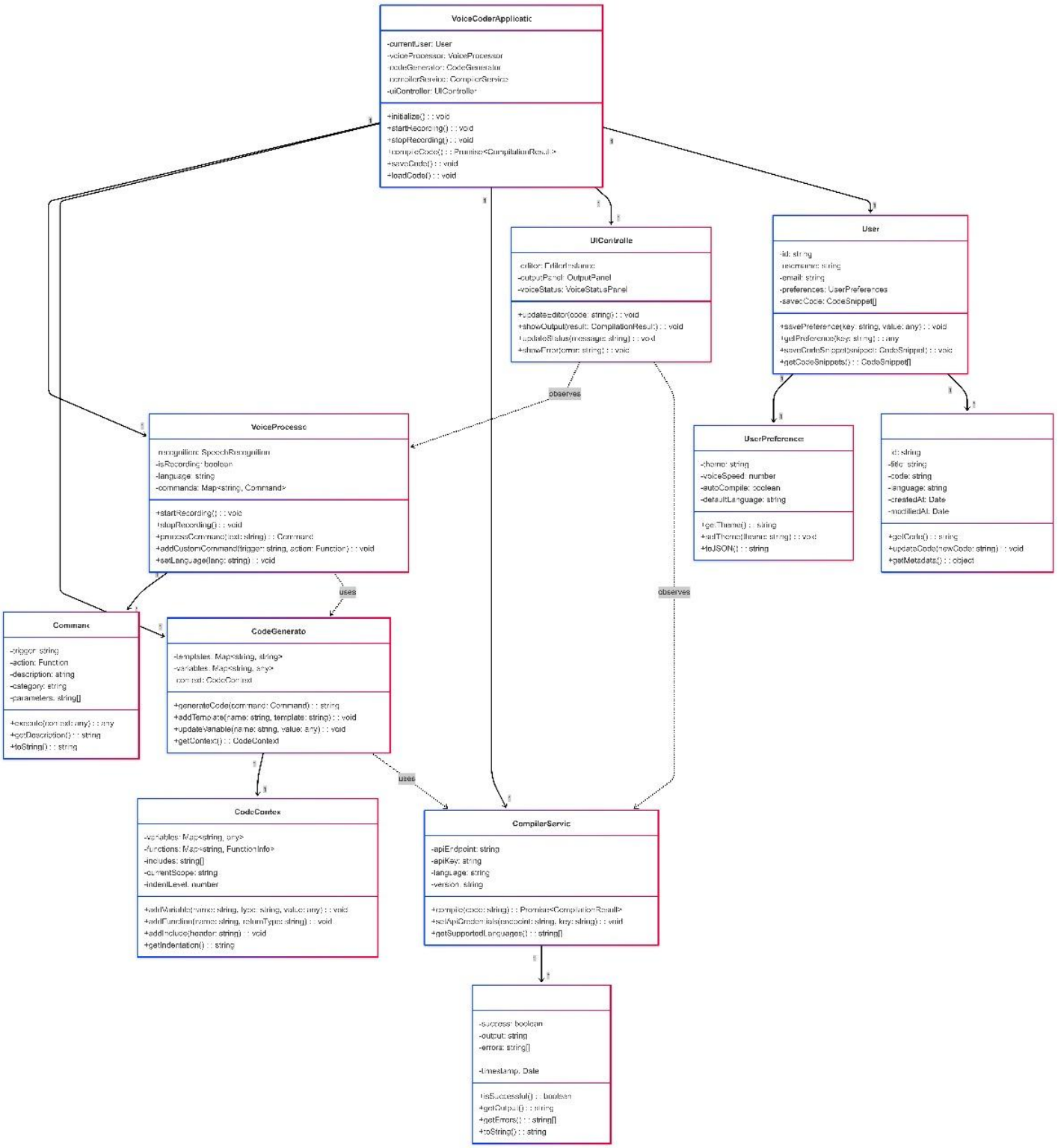

VoiceCoderApplicatic
-currentUser: User
-voiceProcessor: VoiceProcessor
-codeGenerator: CodeGenerator
-compilerService: CompilerService
-uiController: UIController
+initialize() : : void
+startRecording() : : void
+stopRecording() : : void
+compileCode() : : Promise<CompilationResult>
+saveCode() : : void
+loadCode() : : void
UIControlle
-editor: CodeMirrorInstance
-outputPanel: OutputPanel
-voiceStatus: VoiceStatusPanel
+updateEditor(code: string) : : void
+showOutput(result: CompilationResult) : : void
+updateStatus(message: string) : : void
+showError(error: string) : : void
User
-id: string
-username: string
-email: string
-preferences: UserPreferences
-savedCode: CodeSnippet[]
+savePreference(key: string, value: any) : : void
+getPreference(key: string) : : any
+saveCodeSnippet(snippet: CodeSnippet) : : void
+getCodeSnippets() : : CodeSnippet[]
observes
VoiceProcesso
-recognition: SpeechRecognition
-isRecording: boolean
-language: string
-commands: Map<string, Command>
+startRecording() : : void
+stopRecording() : : void
+processCommand(text: string) : : Command
+addCustomCommand(trigger: string, action: Function) : : void
+setLanguage(lang: string) : : void
UserPreference
-theme: string
-voiceSpeed: number
-autoCompile: boolean
-defaultLanguage: string
+getTheme() : : string
+setTheme(theme: string) : : void
+toJSON() : : string
id: string
-title: string
-code: string
-language: string
-createdAt: Date
-modifiedAt: Date
+getCode() : : string
+updateCode(newCode: string) : : void
+getMetadata() : : object
uses
observes
Command
-trigger: string
-action: Function
-description: string
-category: string
-parameters: string[]
+execute(context: any) : : any
+getDescription() : : string
+toString() : : string
CodeGenerato
-templates: Map<string, string>
-variables: Map<string, any>
-context: CodeContext
+generateCode(command: Command) : : string
+addTemplate(name: string, template: string) : : void
+updateVariable(name: string, value: any) : : void
+getContext() : : CodeContext
uses
CodeContex
-variables: Map<string, any>
-functions: Map<string, FunctionInfo>
-includes: string[]
-currentScope: string
-indentLevel: number
+addVariable(name: string, type: string, value: any) : : void
+addFunction(name: string, returnType: string) : : void
+addInclude(header: string) : : void
+getIndentation() : : string
CompilerServic
-apiEndpoint: string
-apiKey: string
-language: string
-version: string
+compile(code: string) : : Promise<CompilationResult>
+setApiCredentials(endpoint: string, key: string) : : void
+getSupportedLanguages() : : string[]
-success: boolean
-output: string
-errors: string[]
-timestamp: Date
+isSuccessful() : : boolean
+getOutput() : : string
+getErrors() : : string[]
+toString() : : string

Figure 5.1: *UML Class Diagram illustrating core classes, their attributes, methods, and relationships in the VoiceCoder application.*

Class Descriptions:

1. VoiceCoderApplication (Main Controller Class)

- ***Purpose:*** *Central coordinator managing all system components*
- ***Attributes:***
  - *currentUser: Currently logged-in user*
  - *voiceProcessor: Handles voice recognition*
  - *codeGenerator: Creates C++ code from commands*
  - *compilerService: Manages code compilation*
  - *uiController: Updates user interface*
- ***Key Methods:***
  - *initialize(): Sets up all system components*
  - *startRecording(): Begins voice capture*
  - *compileCode(): Compiles and executes code*
  - *saveCode(): Persists code snippets*

2. VoiceProcessor (Voice Recognition Handler)

- ***Purpose:*** *Processes voice input and converts to commands*
- ***Attributes:***
  - *recognition: Web Speech API instance*
  - *isRecording: Current recording status*
  - *commands: Map of voice commands to actions*
- ***Key Methods:***
  - *processCommand(): Converts speech to executable command*
  - *addCustomCommand(): Allows user-defined commands*

3. CodeGenerator (Code Creation Engine)

- ***Purpose:*** *Generates C++ code from parsed commands*
- ***Attributes:***

- *templates: Code templates for different constructs*
- *context: Current programming context (variables, scope)*

- ***Key Methods:***
  - *generateCode(): Creates C++ code from command*
  - *addTemplate(): Adds new code patterns*

4. CompilerService (Compilation Manager)

- ***Purpose:*** *Handles code compilation through external APIs*
- ***Attributes:***
  - *apiEndpoint: Cloud compiler API URL*
  - *apiKey: Authentication credentials*
- ***Key Methods:***
  - *compile(): Sends code to cloud compiler*
  - *setApiCredentials(): Configures external service*

5. User (User Management)

- ***Purpose:*** *Manages user data and preferences*
- ***Attributes:***
  - *preferences: User settings (theme, voice speed)*
  - *savedCode: Collection of code snippets*
- ***Key Methods:***
  - *savePreference(): Stores user settings*
  - *saveCodeSnippet(): Saves code for later use*

6. Supporting Classes:

- ***Command:*** *Represents a voice command with parameters*
- ***CodeContext:*** *Maintains programming context (variables, includes)*
- ***CompilationResult:*** *Stores compilation output and errors*
- ***CodeSnippet:*** *Represents saved code with metadata*
- ***UserPreferences:*** *User configuration settings*

Class Relationships:

- ***Aggregation:*** *VoiceCoderApplication contains VoiceProcessor, CodeGenerator, and CompilerService*
- ***Composition:*** *CodeGenerator owns CodeContext*
- ***Association:*** *VoiceProcessor uses Command objects*
- ***Dependency:*** *CompilerService depends on external API configuration*

Design Patterns Implemented:

1. ***Factory Pattern:*** *Command creation based on voice input*
2. ***Strategy Pattern:*** *Different code generation strategies for various C++ constructs*
3. ***Observer Pattern:*** *UI updates based on voice recognition events*
4. ***Template Method Pattern:*** *Code generation with customizable templates*
5. ***Singleton Pattern:*** *Application configuration management*

Data Structures:

- ***HashMap:*** *For command lookup and template storage (O(1) average access)*
- ***Stack:*** *Managing code scope and indentation levels (LIFO operations)*
- ***Queue:*** *Processing voice command sequences (FIFO operations)*
- ***Tree:*** *Representing code structure hierarchy (nested scopes)*

Performance Considerations:

- ***Command Matching:*** *Trie data structure for O(n) lookup where n is command length*
- ***Template Caching:*** *LRU cache for frequently used code patterns*
- ***Memory Management:*** *Object pooling for Command and CodeContext objects*
- ***Async Operations:*** *Non-blocking I/O for compiler API calls*

Error Handling Strategy:

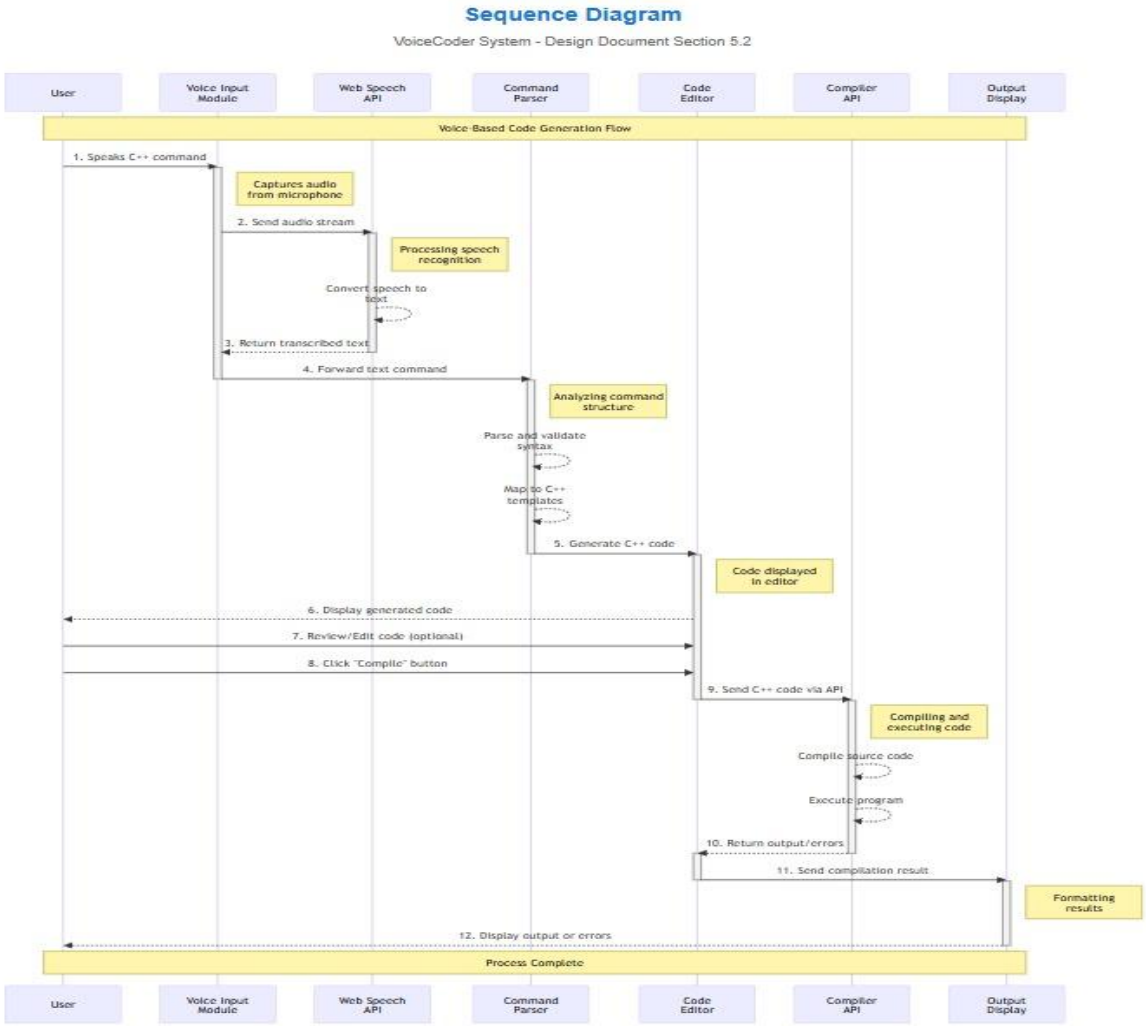
- ***Voice Recognition Errors:*** *Fallback to manual input with clear error messages*
- ***Compilation Failures:*** *Detailed error parsing with line number references*
- ***Network Issues:*** *Retry logic with exponential backoff*
- ***Invalid Commands:*** *Suggestions for correct command syntax*

5.2 SEQUENCE DIAGRAM

Below is the conceptual flow of how the system handles voice-driven code generation.

The user speaks a command, which is captured as audio. This audio is converted into text by the Web Speech API (MDN, 2024) [4].
The Command Parser analyzes this text and generates equivalent C++ structures.
When the user presses "Compile", the code is transmitted to an external compiler API (Paiza.io, 2024) [2], which returns the output.

Sequence Diagram:

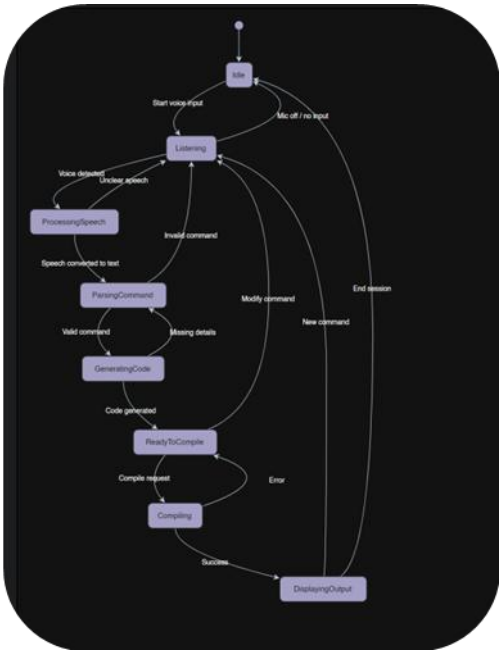


5.3 State Transition Diagram

Primary States

1. Idle
2. Listening
3. Processing Speech
4. Parsing Command
5. Generating Code
6. Ready to Compile
7. Compiling
8. Displaying Output

Transitions occur based on voice events, parsing success, or API responses.



5.4 Logical Data Model (ERD)

Entities & Attributes:

UserSession

- *sessionId*
- *startTime*
- *microphoneStatus*

VoiceCommand

- *commandId*
- *commandText*
- *timestamp*
- *parsedStructure*

GeneratedCode

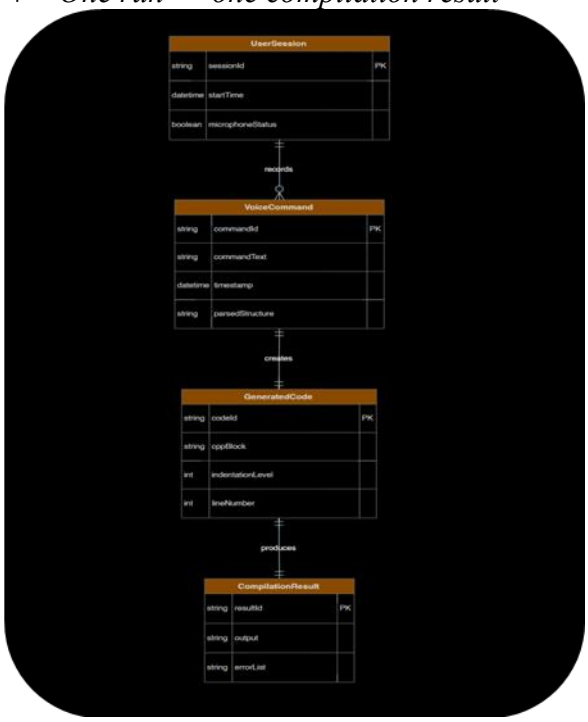
- *codeId*
- *cppBlock*
- *indentationLevel*
- *lineNumber*

CompilationResult

- *resultId*
- *output*
- *errorList*

Relationships:

✧ *One session → many commands*
✧ *One command → one code block*
✧ *One run → one compilation result*

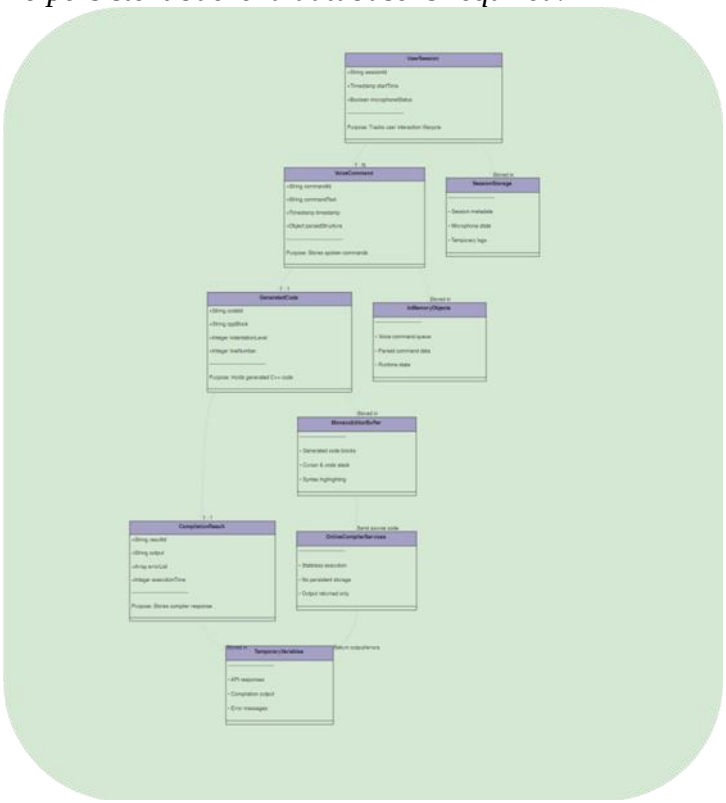


5.5 Physical Data Model

Since the system is browser-based and stateless, data is stored temporarily:

- ***Local Storage / Session Storage*** *for temporary logs*
- ***In-memory Objects*** *for commands, templates, parsing results*
- ***External API Storage*** *not used; compiler APIs return output only*

No persistent backend database is required.



5.6 GUI Designs (Detailed)

1. Home Dashboard

Title banner "Voice Coding"
Large microphone button (Start Recording)
Live waveform animation during speech capture

2. Code Editor Screen

Monaco Editor with:

- *Dark theme*
- *Syntax colors*
- *Line numbers*
- *Auto indentation*

Buttons:

- *Run Code*
- *Stop Recording*
- *Clear Code*
- *Open Command Guide*

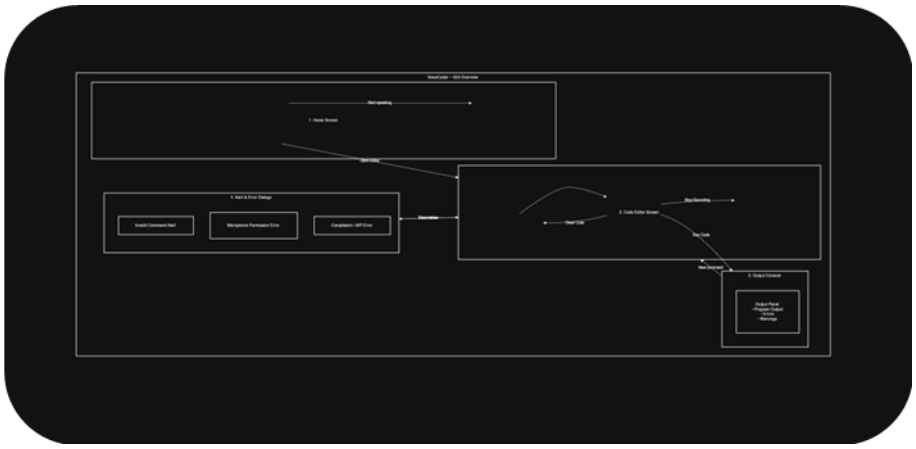
3. Output Console

Panel below editor

- *Results*
- *Warnings*
- *Compilation errors*
- *Auto-scroll enabled*

4. Error Popup Modals

Invalid command notifications
API timeout messages
Microphone access alerts



6.References

[1] I. Sommerville, *Software Engineering*, 10th ed. Boston, MA, USA: Pearson Education, 2016.

[2] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed. New York, NY, USA: McGraw-Hill, 2015.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley, 1994.

[4] Mozilla Developer Network (MDN), "Web Speech API," Mozilla Foundation, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API

[5] Microsoft, "Monaco Editor Documentation," Microsoft Docs, 2024. [Online]. Available: https://microsoft.github.io/monaco-editor/

[6] Paiza, Inc., "Paiza.IO Online Compiler API Documentation," Paiza, 2024. [Online]. Available: https://paiza.io/en/api

[7] JDoodle, "JDoodle Compiler API Documentation," JDoodle, 2024. [Online]. Available: https://www.jdoodle.com/compiler-api

[8] Object Management Group (OMG), *Unified Modeling Language (UML) Specification*, Version 2.5.1, 2017.

[9] A. Dennis, B. Wixom, and D. Tegarden, *Systems Analysis and Design: An Object-Oriented Approach with UML*, 5th ed. Hoboken, NJ, USA: Wiley, 2015.

[10] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. Upper Saddle River, NJ, USA: Pearson, 2023.

[11] W3C, "Web Storage API Specification," World Wide Web Consortium, 2011. [Online]. Available: https://www.w3.org/TR/webstorage/

[12] N. Leveson, *Safeware: System Safety and Computers*. Reading, MA, USA: Addison-Wesley, 1995.

[13] ISO/IEC, *ISO/IEC 25010:2011 – Systems and Software Quality Models*, International Organization for Standardization, 2011.

[14] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Hoboken, NJ, USA: Wiley, 2018.

[15] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann, 1994.

[16] W3C, *Web Content Accessibility Guidelines (WCAG) 2.1*, W3C Recommendation, 2018. [Online]. Available: https://www.w3.org/TR/WCAG21/

[17] Apple Inc., "Voice User Interface Design Guidelines," Apple Human Interface Guidelines, 2024. [Online]. Available: https://developer.apple.com/design/human-interface-guidelines/voice

[18] Google Developers, "Web Speech API Implementation Guide," Google Chrome Developers, 2024. [Online]. Available: https://developer.chrome.com/docs/web-platform/web-speech/

[19] A. Osmani, *Learning JavaScript Design Patterns*. Sebastopol, CA, USA: O'Reilly Media, 2017.

[20] IEEE, *IEEE Standard for Information Technology—Systems Design—Software Design Descriptions*, IEEE Std 1016-2009, 2009.

[21] B. Stroustrup, *The C++ Programming Language*, 4th ed. Boston, MA, USA: Addison-Wesley, 2013.

[22] ISO/IEC, *Programming languages — C++*, ISO/IEC 14882:2020, International Organization for Standardization, 2020.

[23] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley, 2002.

[24] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Boston, MA, USA: Addison-Wesley, 1999.

[25] ECMA International, *ECMAScript 2023 Language Specification*, ECMA-262 14th Edition, 2023. [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-262/

[26] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308-320, Dec. 1976.

[27] Microsoft Corporation, "N-tier architecture style," Microsoft Azure Architecture Guide, 2023. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier

[28] World Health Organization, *Global report on assistive technology*, WHO, 2022. [Online]. Available: https://www.who.int/publications/i/item/9789240049451