VoiceBridge

Module Guide for Software Engineering

Team 13, Speech Buddies
Mazen Youssef
Rawan Mahdi
Luna Aljammal
Kelvin Yu

February 14, 2026

1 Revision History

Date	Version	Notes
Nov 11, 2025	1.0	Added modules and Traceability Matrix
Nov 12, 2025	1.1	Added Modules M6-M11
Nov 12, 2025	1.2	Added Modules M15-M19
Nov 26, 2025	1.3	Added Modules M1-M6
Jan 10, 2026	1.4	Implemented TA Feedback for M7-M11
Jan 10, 2026	1.4	Implemented TA and Peer Feedback for AC/UC distinctions
Jan 21, 2026	2.0	Merged modules in application layer and updated timeline
Jan 21, 2026	2.1	Added communication protocol details
Feb 10, 2026	2.2	Implemented advisor feedback on structure, clarity and terminology

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
UC	Unlikely Change
FR	Functional Requirement
LF	Look and Feel
PF	Performance
OER	Operational & Environmental
MS	Maintainability & Support
SEC	Security
CUL	Cultural
CPL	Compliance
TLS	Transport Layer Security

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (**?**). For VoiceBridge, we adopt this decomposition based fundamentally on Parnas's principle of information hiding (**?**). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. This approach supports design for change, a critical concern in assistive technologies where adaptation to evolving user needs and cutting-edge speech recognition improvements frequently occur, especially in early development phases.

Our design follows the rules layed out by **?**, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (**?**). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: Consumer-grade hardware platform and input devices.
VoiceBridge is expected to run on a range of consumer-grade devices (desktops, laptops, tablets) with standard microphones and evolving OS audio stacks. Variations in microphone quality, form factor, and host processing capabilities, as well as future improvements in consumer audio hardware, should be handled by the audio acquisition and preprocessing modules (e.g., Microphone Manager and VAD/NoiseFilter) without impacting higher layers.

AC2: Speech-to-text engine and language support.
The system might expand to support additional languages and dialects beyond English requiring adaptations in ASR models and intent interpretation logic. Continuous improvements in speech recognition accuracy and noise filtering will also necessitate regular updates.

AC3: User profile and personalization management.
Personalization features may evolve to accommodate changing user speech patterns and preferences, demanding updates in model fine-tuning, adaptive prompting based on confidence, and profile management strategies.

AC4: Command mapping and browser automation protocols.
Browser capabilities, permission models, and automation interfaces (such as extension manifests and scripting APIs) are expected to evolve over time. New commands, revised command schemas, and updated browser/OS integration will require changes localized to the Command Orchestration and Browser Orchestrator modules, which are designed to isolate such API-level changes from the rest of the system.

AC5: User interface and accessibility compliance.
Responsive adjustments to UI designs and accessibility layers will be required to stay compliant with evolving WCAG guidelines and to address user feedback for improved usability.

AC6: Error handling and recovery policies.
Classification schemes, messaging protocols, retry/backoff strategies, and compensation mechanisms may be enhanced to increase robustness and user experience quality.

AC7: Session lifecycle and interaction flow.
Changes may extend session duration limits, improve state persistence, and better handle asynchronous or interrupted user interactions.

AC8: Data privacy, audit logging, and security.
Adaptations will be necessary to comply with evolving data privacy laws and security best practices, affecting encryption, audit trail formats, and consent mechanisms.

AC9: Personalization and prompting enhancements.
Enhancements aiming at more context-aware and confidence-driven user assistance will continually refine prompting and instruction modules.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Switching to specialized or non-consumer deployment platforms.
VoiceBridge is designed to run on consumer-grade devices using standard microphones and common operating systems. Supporting specialized assistive hardware would require major changes to hardware assumptions, certification, and system integration, and is out of scope for this project.

UC2: Offloading speech processing or command execution to external or cloud systems.
To protect user privacy and maintain responsiveness, VoiceBridge performs all key processing locally. Changing this would require major architecture redesign and is not planned.

UC3: Fundamental changes to the speech recognition approach.
The system uses speech recognition models adapted for users with speech impairments. Major changes to the recognition method—such as replacing local processing with a fully cloud-based approach—would be disruptive and are not planned.

UC4: Complete redesign of command mapping and browser control modules.
The system uses a browser-bridge automation pattern, with dedicated modules (Command Orchestrator, Browser Orchestrator) that encapsulate browser APIs and OS integration. Updates to support new browser APIs or extensions are expected, but replacing the underlying automation architecture is not planned.

UC5: Abandoning modular design for a monolithic or radically different architecture.
Modularity was chosen for maintainability and scalability. Changing to a less modular approach would increase complexity and reduce flexibility.

5 Module Hierarchy

This section provides an overview of the module design. The system is organized into architectural layers, each containing modules that encapsulate specific responsibilities. Table 1 presents the hierarchy of modules by their layers, including their primary purposes and responsibilities. The modules listed below, which are leaves in the hierarchy tree, are the modules that will be implemented.

5.1 Architectural Terminology

To clarify the structural organization of this system, we distinguish between three key architectural concepts:

Layer: A layer is a horizontal grouping of modules that share a common level of abstraction and responsibility within the system architecture. Layers enforce separation of concerns by restricting dependencies—modules within a layer may only depend on modules in lower layers, never on those in higher layers. For example, the Presentation Layer contains all user-facing components (M1-M3), while the Data Management Layer (M9-M14) handles persistence and security. Layers represent what broad categories of functionality exist.

Module: A module is a self-contained software component that encapsulates a specific set of secrets (design decisions) and provides a well-defined set of services through its interface. Each module is the fundamental unit of implementation and change—modifications to a module's internal implementation should not require changes to other modules, provided its interface remains stable. Modules represent how specific responsibilities are implemented and hidden behind abstractions.

Orchestrator: An orchestrator is a specialized type of module whose primary responsibility is coordinating the interactions between other modules to accomplish complex workflows. Unlike standard modules that focus on a single domain-specific task, orchestrators manage sequencing, decision-making, and data flow across module boundaries. In this system, M5 (Command Orchestrator) exemplifies this pattern by synthesizing speech transcripts into browser commands through coordination with M4, M17 (Prompting Module), M7 (Browser Orchestrator), and M8 (Session Manager). Orchestrators represent when modules interact and why certain sequences occur.

5.2 Design Rationale

Separation of Concerns: The architecture strictly separates presentation (M1-M3), processing (M4-M6), control (M7-M8), data (M9-M14), input (M15-M16), and personalization (M17-M18). This ensures that changes in one layer (e.g., switching STT engines) do not cascade across unrelated modules.

Security by Design: Security-critical functions are isolated in dedicated modules (M11-M14) with restricted access patterns. Credentials, encryption keys, and audit logs are managed independently to support defense-in-depth.

Accessibility First: The Accessibility Layer (M2) is a first-class architectural component rather than an afterthought, ensuring WCAG compliance is enforced systematically across all UI components.

LLM-Driven Command Synthesis: Unlike traditional intent-based systems with fixed command registries, the Command Orchestrator (M5) uses prompt engineering and schema validation to synthesize commands dynamically, enabling flexible natural language understanding without hardcoded mappings.

Personalization Without Privacy Trade-offs: The Model Tuner (M18) and User Profile Manager (M10) work in tandem with Encryption (M13) to enable user-specific adaptations while maintaining data anonymization for model training.

5.3 Module Listing

The following 18 modules represent the leaf nodes in the architectural decomposition and will be implemented as distinct software components:

- User Interface Module
- Accessibility Layer
- Feedback Display Module
- Speech-to-Text Engine
- Command Orchestrator
- Error Feedback
- Browser Orchestrator
- Session Manager
- Data Management Layer
- User Profile Manager
- Audit Logger
- Credential Manager
- Encryption Manager
- Out-of-Scope Handler
- Microphone Manager
- VAD Noise Filter
- Prompting Module
- Model Tuner

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the (SRS). In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 1.

Architectural Layer	**Included Modules**	**Purpose / Responsibilities**
Presentation Layer	User Interface Module Accessibility Layer Feedback Display Module	Manages direct user interaction and feedback presentation. Displays transcribed text, confirmations, feedback messages, and status indicators. Ensures compliance with WCAG accessibility standards including contrast, font, and color.
Control (Orchestration) Layer	Browser Orchestrator Session Manager	Coordinates and manages flow between modules. Tracks session states such as capture, transcribe, confirm, and execute. Manages events, exceptions, retries, and cancellations to ensure smooth operation.
Application (Processing) Layer	Error Feedback Module Command Orchestrator Speech-to-Text Engine	Performs speech recognition and intent processing. Converts speech audio to text, interprets user intent, maps intent to browser or OS commands, executes commands, and handles error feedback.
Data Management Layer	Data Storage Manager User Profile Manager Audit Logger	Maintains persistent data and user personalization. Stores user speech samples, preferences, command mappings, transcripts, and logs. Supports diagnostics and evaluation.
Security Layer	Credential Manager Encryption Manager Out-of-Scope Handler	Manages authentication, encryption, and system boundaries. Ensures secure integration with external APIs and operating system interfaces. Handles login confirmations.
Input Processing Layer	Microphone Manager VAD and Noise Filter	Controls microphone hardware and audio capture. Filters noise to improve voice input quality.
Personalization Layer	Prompting Module Model Tuner	Provides user prompting, model fine-tuning, and instruction management. Supports optional personalization based on confidence scores and usage history.

Table 1: Module Hierarchy

7 Module Decomposition

Presentation Layer Modules

M1: User Interface Module

M1 **Secrets:**

- Representation of UI state, layout, and visible components
- Mapping of raw browser events into UiEvents
- Management of keyboard and screen-reader focus

Services:

- Initialize and render UI
- Receive and normalize input events into UiEvents
- Route feedback items
- Update keyboard and screen-reader focus

Implemented By: UserInterface module using the browser rendering engine and DOM/event APIs.

M2: Accessibility Layer

M2 **Secrets:**

- Encoding of accessibility configuration (AccessConfig)
- ARIA attribute mappings
- Validation rules for UI elements against WCAG standards

Services:

- Apply accessibility options
- Announce messages
- Check UI elements and report problems using ValidationReports

Implemented By: AccessibilityLayer module operating on UI elements exposed by User-Interface.

M3: Feedback Display Module

M3 **Secrets:**

- Internal representation of FeedbackItems keyed by UUID
- Policies for creation, clearance, and recovery options of feedback messages

Services:

- Attach to parent UI
- Show messages of given types
- Clear messages
- Create recovery options for feedback items

Implemented By: FeedbackDisplay module rendering via UserInterface and coordinating with AccessibilityLayer.

Application (Processing) Modules

M4: Speech-to-Text Engine

M4 **Secrets:**

- Configuration (AsrConfig) and internal processing details of the speech-to-text engine
- Methods for converting audio input into accurate text transcripts

Services:

- Initialize engine
- Process audio to produce transcripts with confidence metadata
- Reset internal state

Implemented By: SpeechToTextEngine module wrapping acoustic and language models.

M5: Command Orchestrator

M5

Secrets:

- Prompt engineering strategies and system instructions for direct command synthesis.
- Session-based conversation context and short-term memory management.
- Heuristics for mapping natural language to browser-specific schemas without a fixed registry.
- Validation logic for ensuring LLM outputs conform to executable security guardrails.

Services:

- Initialize the orchestrator with system prompts and API configurations.
- Synthesize raw speech transcripts directly into structured, validated browser commands.
- Assign confidence scores based on model reasoning.
- Reset session context and clear conversation history.

Implemented By: CommandOrchestrator module using an external LLM API, dynamic context injection, and a JSON schema validation layer.

M6: Error Feedback

M6

Secrets:

- Expected error types
- Message templates and localization hooks
- Mapping of internal/domain errors to user-facing message keys and recovery options
- Redaction rules for logs and UI

Services:

- Display error notifications
- Display recovery prompts with options
- Dismiss items
- Log structured error events
- Normalize and map errors to UI messages and recovery actions

Implemented By: UI Notification Layer and Shared Error Handling Module (M11).

Control (Orchestration) Modules

M7: Browser Orchestrator

M7 **Secrets:**

- Construction of the `browser_use` agent (`Agent`) and LLM wrapper (`ChatBrowserUse`)
- How the natural-language `command` is passed to the agent (`task=command`)
- Interpretation/formatting of the returned execution `history`

Services:

- Execute a natural-language browser command using an active `Browser` instance

Implemented By: `src/control/browser_orchestrator.py`

M8: Session Manager

M8 **Secrets:**

- Management of a single shared browser instance (`_browser`) for the whole application
- Browser start/stop lifecycle and cleanup logic
- Keep-alive configuration for reusing the same browser across multiple commands (`keep_alive=True`)

Services:

- Start the shared browser session
- Stop the shared browser session
- Provide access to the active `Browser` instance (or `None` if not running)

Implemented By: `src/control/session_manager.py`

Data Management Modules

M9: Storage Management Module

M9 **Secrets:**

- Database connection credentials
- Cloud API tokens
- Encryption keys

Services:

- DataStorageService: Stores and retrieves transcripts, user data, and configuration files.
- BackupService: Performs scheduled or manual backups to secure cloud storage.
- DataRetrievalService: Provides indexed access to stored files for authorized modules.
- DataRetentionService: Enforces deletion and retention rules based on time or policy.

Implemented By: DataStorageManager.

M10: User Profile Management Module

M10 **Secrets:**

- User tokens and refresh keys
- Profile encryption keys
- Hashed user identifiers

Services:

- ProfileCreationService: Initializes new user profiles with unique IDs.
- PreferenceService: Saves and retrieves user preferences and personalization data.
- ConsentManagementService: Records and verifies consent for data collection and personalization.

Implemented By: UserProfileManagerImpl.

M11: Audit Logging Module

M11 **Secrets:**

- Log signing key
- Audit database token
- Log encryption key

Services:

- ActivityLogService: Logs system and user activities for traceability.
- LogQueryService: Provides authorized access to query and review logs.
- AnomalyDetectionService: Detects suspicious activity and security anomalies.

Implemented By: AuditLoggerImpl.

M12: Credential Management Module

M12 **Secrets:**

- Vault root key
- Token signing keypair
- OAuth client secrets

Services:

- AuthenticationService: Validates user credentials during login.
- SessionTokenService: Issues and validates secure access tokens (JWT/OAuth).
- PasswordVaultService: Stores and retrieves hashed passwords securely.

Implemented By: CredentialManagerImpl.

M13: Encryption Management Module

M13 **Secrets:**

- Private encryption keypair
- TLS certificates for secure communication
- Key rotation schedule metadata

Services:

- DataEncryptionService: Encrypts data at rest and in transit.
- KeyManagementService: Generates, rotates, and revokes cryptographic keys.
- AnonymizationService: Removes identifiable data prior to model training or storage.
- IntegrityVerificationService: Ensures stored or transmitted data has not been altered.

Implemented By: EncryptionManagerImpl.

M14: Out-of-Scope Handling Module

M14 **Secrets:**

- Command allowlist/denylist configuration
- Safety policy file checksum
- Secure incident reporting token

Services:

- CommandValidationService: Validates command scope and prevents unsafe actions.
- UserSafetyService: Provides safe fallback messages and cancel options.
- RecoveryService: Rolls back partially executed or invalid operations.
- IncidentReportingService: Reports anomalies and unsafe commands to the AuditLogger.

Implemented By: OutOfScopeHandlerImpl.

Input Processing Layer Modules

M15: Microphone Manager

M15 **Secrets:**

- Device-level audio capture configuration and sampling parameters
- Buffering strategy and fallback device selection
- Normalization of raw audio before downstream processing

Services:

- Start and stop microphone stream
- Read audio frames
- Provide normalized PCM buffer
- Expose current device status and surface hardware errors

Implemented By: Browser audio APIs or OS audio interface.

M16: VAD Noise Filter

M16 **Secrets:**

- Voice-activity-detection thresholds
- Noise-suppression heuristics and smoothing windows
- Adaptive gain rules

Services:

- Filter raw microphone frames
- Determine speech vs. silence
- Emit clean audio frames
- Send VAD events (start/stop speech)

Implemented By: DSP/VAD library.

Personalization Layer

M17: Prompting Module

M17 **Secrets:**

- Prompt templates and personalization rules
- Condition-based variation (history, context)
- Confidence-based prompt selection logic

Services:

- Generate user prompts
- Provide clarifying follow-ups when intent confidence is low
- Produce culturally inclusive and accessibility-enabled copy

Implemented By: Application prompt engine.

M18: Model Tuner

M18 **Secrets:**

- Adaptation strategy for user-specific speech patterns
- Tuning weights for model updates
- Online vs. offline update thresholds
- Embedding storage rules

Services:

- Update inference parameters based on user data
- Compute personalized embeddings
- Adjust thresholds for intent classification and STT confidence

Implemented By: ML runtime (Whisper fine-tuning layer or lightweight personalization pipeline).

8 Traceability Matrix

The traceability matrices show how each requirement is implemented by specific modules and how each anticipated change is isolated within a single module.

Req.	**Modules**
FR-1 10.1	M15, M16, M4, M8
FR-2 10.2	M4, M16, M18, M9
FR-3 10.3	M1, M2, M3, M6
FR-4 10.4	M5, M6
FR-5 10.5	M7, M7, M11, M6

Table 2: Trace Between Functional Requirements and Modules

Req.	**Modules**
LF-1 11.1	M1, M2, M3, M6
LF-2 11.2	M1, M2, M17, M3

Table 3: Trace Between Look & Feel Requirements and Modules

Req.	**Modules**
UH-1 12.1	M1, M2, M3, M6
UH-2 12.2	M10, M9, M5, M6
UH-3 12.3	M1, M2, M17, M8
UH-4 12.4	M6, M3, M2, M17
UH-5 12.5	M2, M1, M3, M6

Table 4: Trace Between Usability & Humanity Requirements and Modules

Req.	**Modules**
PF-1 13.1	M4, M5, M6, M7, M7
PF-2 13.2	M5, M6, M7, M6, M11
PF-3 13.3	M4, M16, M18, M5
PF-4 13.4	M11, M8, M6, M7
PF-5 13.5	M8, M9, M10
PF-6 13.6	M8, M9, M7
PF-7 13.7	M9, M10, M11, M12

Table 5: Trace Between Performance Requirements and Modules

Req.	**Modules**
OER-1 14.1	M15, M16, M4, M11
OER-2 14.2	M1, M2, M7, M8
OER-3 14.3	M5, M6, M7
OER-4 14.4	M9, M10, M11, M12, M8
OER-5 14.5	M8, M9, M11, M12

Table 6: Trace Between Operational & Environmental Requirements and Modules

Req.	**Modules**
MS-1 15.1	M8, M9, M10, M11, M12
MS-2 15.2	M1, M2, M3, M6, M17
MS-3 15.3	M1, M2, M7, M8, M11

Table 7: Trace Between Maintainability & Support Requirements and Modules

Req.	**Modules**
SEC-1 16.1	M12, M13, M10, M9, M8
SEC-2 16.2	M5, M6, M7, M6, M11
SEC-3 16.3	M9, M10, M11, M12, M13
SEC-4 16.4	M11, M9, M8, M12
SEC-5 16.5	M11, M6, M16, M7, M7

Table 8: Trace Between Security Requirements and Modules

Req.	**Modules**
CUL-1 17.1	M2, M3, M6, M17, M5

Table 9: Trace Between Cultural Requirements and Modules

Req.	**Modules**
CPL-1 18.1	M9, M10, M11, M12, M13
CPL-2 18.2	M1, M2, M3, M6

Table 10: Trace Between Compliance Requirements and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

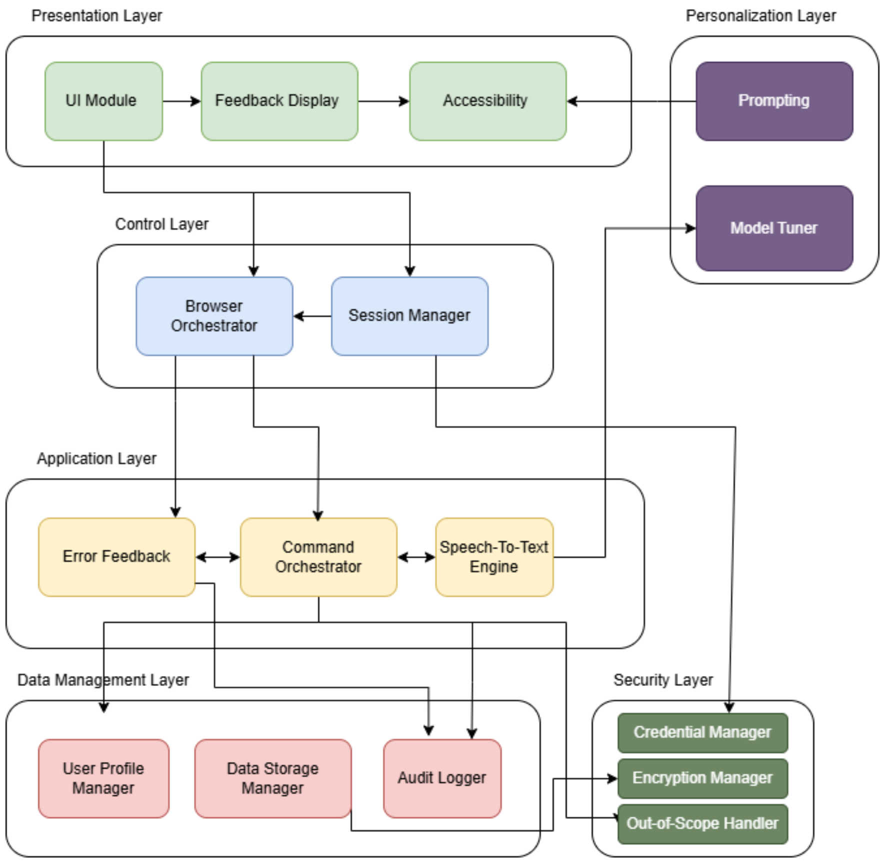


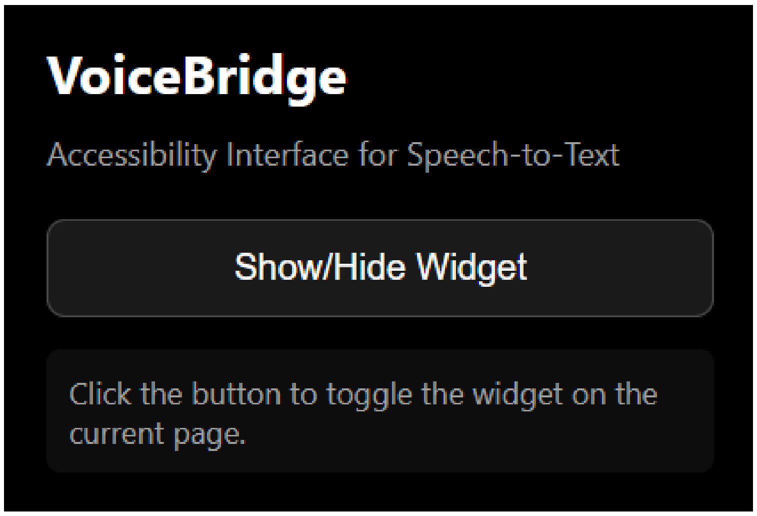
Figure 1: Use hierarchy among modules

10 User Interfaces

This section presents the primary user interface components of the VoiceBridge system. The interfaces are designed with accessibility, clarity, and low cognitive load in mind.

Figure 2: VoiceBridge browser toolbar icon

The browser toolbar icon displayed in Figure 2 serves as the primary entry point for VoiceBridge within the browser environment.



Click the button to toggle the widget on the current page.

Figure 3: Browser extension widget toggle interface

The browser extension control panel allows users to enable or disable the VoiceBridge widget on the current webpage using a single-action toggle.

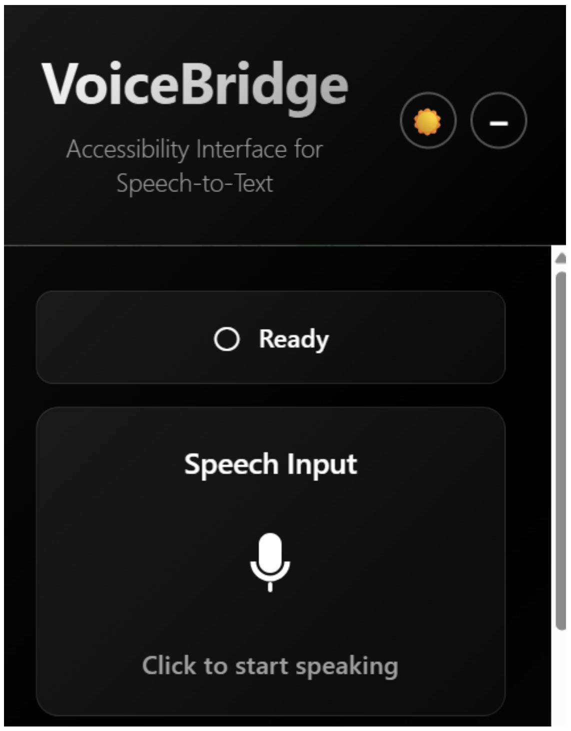


Figure 4: Main VoiceBridge application interface

The main application interface shown in Figure 4 integrates system status, speech input, and transcription components into a single cohesive layout.

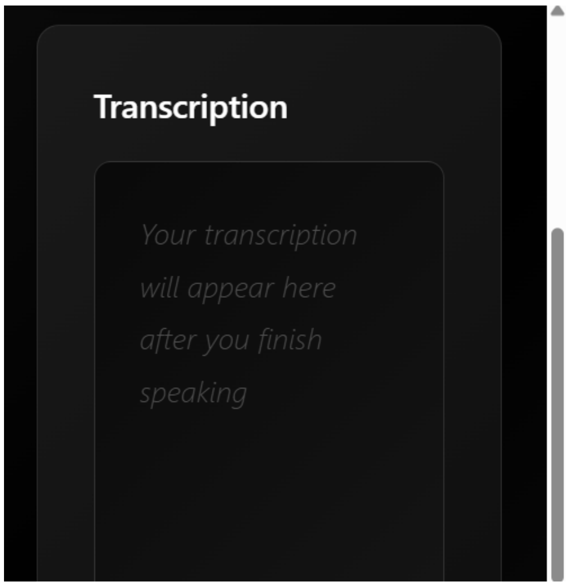

Transcription
Your transcription will appear here after you finish speaking

Figure 5: Live transcription output panel

Figure 5 presents the transcription panel where speech is displayed after the user finishes speaking. This component allows users to visually verify the system's interpretation of their speech before any further action is taken.

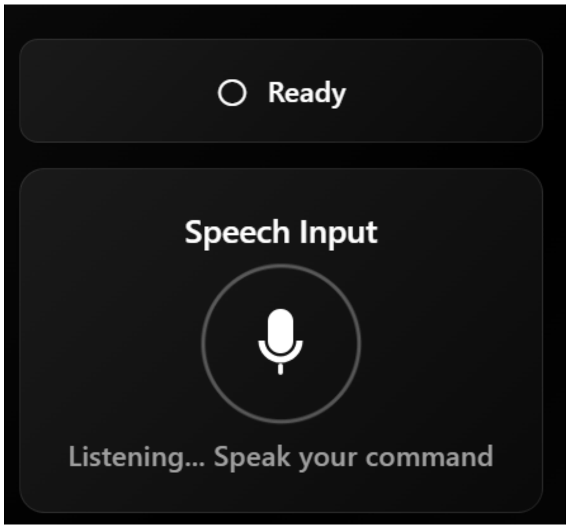


Figure 6: Speech input interface while listening

The speech input interface, shown in Figure 6, provides clear feedback on the system's listening state. A large microphone icon indicates when speech input is active, while supporting text guides the user to speak their command.

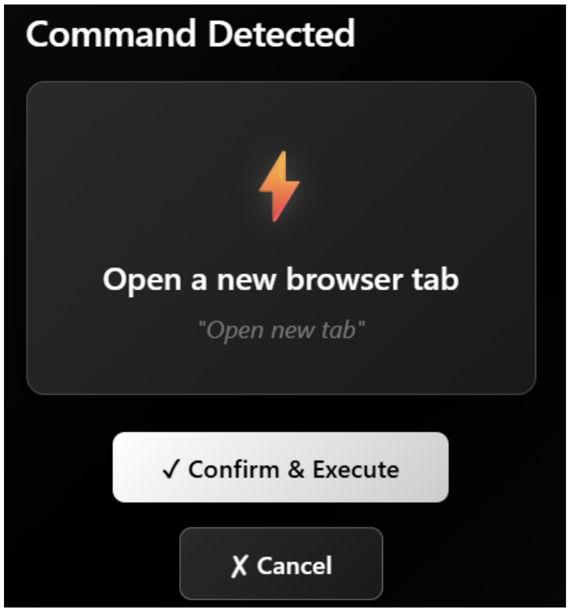


Figure 7: Detected voice command confirmation screen

Figure 7 shows the command detection interface that appears after a voice command has been successfully recognized. The detected intent (e.g., opening a new browser tab) is displayed, along with the interpreted spoken phrase.

11 Design of Communication Protocols

VoiceBridge uses typed procedure calls between modules following the MIS details. All communication is synchronous with structured data types and exception handling.

- **Typed Access Programs**: Module interfaces are defined in the MIS, utilizing request-response patterns for inter-module communication.
- **Structured Messages**: Data types such as `UiEvent`, `Transcript`, `Command`, and `FeedbackItem` are used to standardize communication.
- **Exception Handling**: Typed errors (e.g., `TimeoutError`, `StoreError`) are propagated and converted into user feedback where applicable.
- **Session Tokens**: UUID-based authentication is managed via the M12 Credential Manager to ensure secure communication.
- **Data Encryption**: The M13 Encryption Manager handles encryption at storage and communication boundaries to maintain data security.

12 Timeline

Core functionality such as speech-to-text and command orchestrator was developed collectively. Each team member took ownership of specific modules to ensure accountability and progress. The schedule in Table 11 reflects the planned completion dates for each module, along with the responsible team member.

Module Name	**Completion Date**	**Responsible Member**
User Interface Module	January 15, 2026	Kelvin
Speech-to-Text Engine	January 15, 2026	Luna
Command Orchestrator	January 15, 2026	Rawan
browser orchestrator	January 15, 2026	Mazen
Accessibility Layer	January 18, 2026	Mazen
Feedback Display Module	January 18, 2026	Rawan
Error Feedback	January 18, 2026	Kelvin
Session Manager	January 21, 2026	Luna
Prompting Module	January 21, 2026	Rawan
User Profile Manager	January 21, 2026	Kelvin
Data Management Layer	January 24, 2026	Rawan
Audit Logger	January 24, 2026	Mazen
Credential Manager	January 24, 2026	Luna
Encryption Manager	January 27, 2026	Rawan
Out-of-Scope Handler	January 27, 2026	Kelvin
Microphone Manager	January 27, 2026	Mazen
VAD Noise Filter	January 27, 2026	Luna
Model Tuner	January 30, 2026	Kelvin

Table 11: Module Completion Schedule