# VoiceBridge

# Module Interface Specification for Software Engineering

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

February 16, 2026

# 1 Revision History

| Date | Version | |
|---|---|---|
| Nov 12, 2025 | 1.0 | Added |
| Jan 11, 2026 | 1.1 | Upda |
| Jan 21, 2026 | 2.0 | Merged modules i |
| Jan 28, 2026 | 2.1 | Added reflection for peer modu |
| Feb 16, 2026 | 2.2 | Updated module specifications to resemble re |

# 2 Symbols, Abbreviations and Acronyms

| Acronym | Meaning |
| --- | --- |
| DOM | Document Object Model |
| ARIA | Accessible Rich Internet Applications |

For previously defined acronyms, please refer to the SRS documentation at SRS. The acronyms listed above are newly introduced in this document.

# Contents

# 3    Introduction

This document presents the Module Interface Specifications (MIS) for VoiceBridge, a modular platform designed to facilitate real-time voice interaction and transcription across diverse applications. VoiceBridge integrates advanced speech-to-text processing, natural language understanding, and command execution to enable intuitive voice-driven workflows.

The system supports accessibility standards, personalized user settings, and feedback mechanisms, allowing both developers and end-users to interact with software efficiently and securely. Core capabilities include continuous audio streaming, intent interpretation, contextual command mapping, and encrypted storage of transcripts and user profiles.

The MIS provides detailed specifications for each module, outlining their interfaces, expected inputs and outputs, state management, and interactions with other components. This document complements the System Requirements Specification (SRS) and Module Guide (MG), offering a reference for implementation, testing, and integration.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/speech-buddies/VoiceBridge/blob/main/docs

# 4    Notation

The structure of the MIS for modules follows **?**, with template adaptations from **?**. The mathematical notation used throughout this document is consistent with the formal conventions presented in Chapter 3 of **?**. For example, the symbol := denotes multiple assignment, and conditional rules appear in the form $(c_1 \Rightarrow r_1 \mid c_2 \Rightarrow r_2 \mid \cdots \mid c_n \Rightarrow r_n)$.

This section summarizes the primitive and derived data types used by the VoiceBridge system.

| Data Type | Notation | |
|---|---|---|
| character | char | A single UT |
| integer | $\mathbb{Z}$ | Integer values used |
| natural number | $\mathbb{N}$ | Posi |
| real | $\mathbb{R}$ | Numerical values used for confi |
| boolean | bool | Logical v |

In addition to these primitive types, VoiceBridge uses several derived types relevant to speech processing, browser automation, and interaction workflows:

- **Sequence** — an ordered list of elements of the same type. Used for sequences of audio frames, transcripts, UI messages, or system logs.

- **String** — a sequence of characters (UTF–8). Used for transcripts, command text, error messages, ARIA labels, and browser actions.

- **Tuple** — fixed-length heterogeneous group of values. Used for configuration records, model parameters, recognized intents, and command mappings.

- **Map / Dictionary** — key–value associations. Commonly used for storing UI elements, active sessions, VAD states, feedback items, command registries, and structured metadata.

- **UUID** — universally unique identifier. Used for session IDs, command IDs, feedback items, error events, and audit log entries.

- **AudioFrame** — a fixed-duration slice of PCM audio sampled at the engine's operating rate (typically 16 kHz). Used by MicrophoneManager, VADNoiseFilter, and SpeechToTextEngine.

- **Transcript** — structured object containing recognized text and confidence metadata. Produced by the Speech-to-Text Engine and consumed by the Intent Interpreter.

- **Intent** — structured semantic representation of a user command. Contains interpreted intent name, slots, and confidence score.

- **Command** — validated, executable instruction produced by CommandMapping and consumed by the Execution Layer.

- **BackendReq / BackendResp** — typed request/response objects exchanged with the Browser Orchestrator and automation bridge.

Functions used by VoiceBridge are typed by input and output domains. Local functions are documented in each module by listing their type signatures followed by their specification. Where relevant, imported functions (e.g., VAD filters, ASR decoders, encryption routines, browser automation calls) are abstracted using their formal types rather than implementation details.

# 5 Module Decomposition

The high-level module decomposition of the system is summarized in Table 1 of the MG document. For a comprehensive and detailed breakdown, please refer to the Module Guide (MG) document available at MG document.

# 6 (M1) MIS of User Interface Module

**Module:** UserInterface

## 6.1 Uses:

- Accessibility Layer (M2) to ensure UI semantics, ARIA roles, and announcements.

- Feedback Display Module (M3) to present messages, prompts, and recovery options.

- Command Orchestrator (M5) for forwarding validated user events.

- Browser Orchestrator (M7) / rendering engine to perform DOM updates and capture input events.

## 6.2 Syntax:

### 6.2.1 Exported Constants:

- None.

### 6.2.2 Exported Access Programs:

| Name | In | Out | Exceptions |
|------|-----|-----|------------|
| UserInterface | config: UiConfig | self: UserInterface | InitializationError |
| receiveInput | event: UiEvent | - | InputError |
| render | state: UiState | - | RenderError |
| showFeedback | msg: FeedbackItem | - | - |
| setFocus | elem_id: string | - | - |

## 6.3 Semantics

### 6.3.1 State Variables:

- `currentState`: `UiState` — current layout, visible components, and active feedback.

- `config`: `UiConfig` — persisted UI preferences (theme, verbosity).

- `focus_target`: `string | null` — element currently targeted for keyboard/screen-reader focus.

### 6.3.2 Environment Variables:

- `UI_THEME` — runtime theme selection (light/dark).

- `LANG` — active locale.

### 6.3.3 Assumptions:

- Browser rendering engine and event APIs are available and conform to expected semantics.

- Downstream modules (M4–M6) accept events in the documented formats.

### 6.3.4 Access Routine Semantics

`UserInterface(config)`:

- **transition:** Initialize `currentState` and `config`; bind to Accessibility Layer (M2) and Feedback Display (M3).

- **output:** Initialized UI instance.

- **exception:** `InitializationError` if required resources are unavailable.

`receiveInput(event)`:

- **transition:** Validate `event`; update `currentState` or forward to Command Orchestrator (M5) where appropriate.

- **output:** -

- **exception:** `InputError` if `event` is malformed or unsupported.

`render(state)`:

- **transition:** Reconcile `currentState` with `state`; update DOM/renderer; notify Accessibility Layer (M2) of attribute changes.

- **output:** -

- **exception:** `RenderError` on failure.

`showFeedback(msg)`:

- **transition:** Delegate presentation to Feedback Display Module (M3); ensure accessible announcement via Accessibility Layer (M2).

- **output:** -

- **exception:** -

`setFocus(elem_id)`:

- **transition:** Set keyboard and screen-reader focus to element identified by `elem_id`; update `focus_target` state.

- **output:** -

- **exception:** `InputError` if `elem_id` does not exist.

4

### 6.3.5 Local Functions

- `apply_config(UiConfig cfg)`: `UiConfig` → `UiState`
  **Description:** `apply_config(cfg)` returns the UI state after setting theme, verbosity, and other options to match `cfg`.

- `normalize_event(RawUiEvent e)`: `RawUiEvent` → `UiEvent`
  **Description:** `normalize_event(e)` returns a standard `UiEvent` with type, target, and payload from raw event `e`.

# 7   (M2) MIS of Accessibility Layer

**Module:** AccessibilityLayer

## 7.1   Uses:

- User Interface (M1) to read and modify UI elements and attributes.

- WCAG guidance / accessibility utilities to verify contrast, labeling, and keyboard support.

- Feedback Display (M3) to coordinate announcements for user messages.

## 7.2   Syntax:

### 7.2.1   Exported Constants:

- None.

### 7.2.2   Exported Access Programs:

| Name | In | Out | Exceptions |
|---|---|---|---|
| AccessibilityLayer | parent:  UserInterface | self | - |
| applySettings | settings:  AccessConfig | bool | - |
| announce | msg:  string | string | - |
| validateElement | elem_id:  string | ValidationReport | - |

## 7.3   Semantics

### 7.3.1   State Variables:

- settings:  AccessConfig — active accessibility options (font scale, contrast overrides, ARIA mappings).

- live_region_id:  string — identifier for announcement region.

### 7.3.2   Environment Variables:

- WCAG_LEVEL — target conformance level (e.g., AA).

### 7.3.3   Assumptions:

- Parent UI (M1) exposes element identifiers and supports attribute updates.

- Localization resources exist for accessible labels when required.

### 7.3.4    Access Routine Semantics

`applySettings(settings)`:

- **transition:** Merge provided `settings` with defaults; apply text scaling, contrast adjustments, and ARIA attribute mappings via `parent`.

- **output:** Boolean success status indicating whether settings were applied successfully.

- **exception:** -.

`announce(msg)`:

- **transition:** Post `msg` to the live region and/or invoke screen-reader API for immediate announcement.

- **output:** Confirmation token or status indicating the announcement was successfully scheduled or posted.

- **exception:** -.

`validateElement(elem_id)`:

- **transition:** Inspect UI element attributes (role, label, states); compute a `ValidationReport` capturing compliance with accessibility standards.

- **output:** `ValidationReport` object detailing any missing roles, labels, or contrast issues.

- **exception:** -.

### 7.3.5    Local Functions

- `check_contrast(rgb fg, rgb bg)`: $\texttt{rgb} \times \texttt{rgb} \to \texttt{bool}$
  **Description:** $\texttt{check\_contrast(fg, bg)} \equiv \frac{L_1 + 0.05}{L_2 + 0.05} \geq \texttt{WCAG\_THRESHOLD}$ where $L_1, L_2$ are relative luminances of fg/bg (used by `validateElement`)

- `aria_set(string elemId, AccessMetadata metadata)`: $\texttt{string} \times \texttt{AccessMetadata} \to \texttt{bool}$
  **Description:** `aria_set(elemId, metadata)` returns `true` if ARIA attributes from `metadata` are applied to element `elemId` (used by `applySettings`)

# 8 (M3) MIS of Feedback Display Module

**Module:** FeedbackDisplay

## 8.1 Uses:

- User Interface (M1) to render feedback content.

- Accessibility Layer (M2) to ensure feedback is announced to assistive technologies.

- Localization/configuration store for templated messages and user preferences.

## 8.2 Syntax:

### 8.2.1 Exported Constants:

- None.

### 8.2.2 Exported Access Programs:

| Name | In | Out | Exceptions |
|------|-----|-----|------------|
| FeedbackDisplay | parent: UserInterface | self | - |
| showMessage | msg: string, type: MsgType | feedbackId: UUID | - |
| clear | - | - | - |
| makeRecovery | feedbackId: UUID | RecoveryOptions | - |

## 8.3 Semantics

### 8.3.1 State Variables:

- `messages: map[UUID] to FeedbackItem` — active feedback items keyed by id.

- `parent: UserInterface` — reference to UI for rendering.

### 8.3.2 Assumptions:

- Parent UI is capable of rendering message templates and interactive recovery prompts.

### 8.3.3 Access Routine Semantics:

`FeedbackDisplay(parent)`:

- **transition:** attach to `parent`; initialize `messages`.

- **output:** initialized feedback display instance.

- **exception:** -.

```
showMessage(msg, type):
```

- **transition:** create `FeedbackItem`, store in `messages`, render via `parent`, and trigger Accessibility (M2) announcement if type requires.

- **output:** `feedbackId` for later reference.

- **exception:** -.

```
clear():
```

- **transition:** remove all entries from `messages` and update UI.

- **output:** -.

- **exception:** -.

```
makeRecovery(feedbackId):
```

- **transition:** build interactive recovery options (buttons, suggested actions) for the feedback item.

- **output:** `RecoveryOptions`.

- **exception:** -.

### 8.3.4   Local Functions:

- `format_message(string msg, MsgType type)`: string $\times$ MsgType $\rightarrow$ FeedbackItem
  **Description:** `format_message(msg, type)` returns a `FeedbackItem` with style, icon, and metadata matching `type` for text `msg`.

- `schedule_dismiss(UUID feedbackId, N ttl_s)`: UUID $\times$ N $\rightarrow$ bool
  **Description:** `schedule_dismiss(feedbackId, ttl_s)` returns `true` if timer is set to remove feedback `feedbackId` after `ttl_s` seconds.

# 9 (M4) MIS of Speech-to-Text Engine

**Module:** SpeechToTextEngine

## 9.1 Uses:

- Audio capture interface to receive microphone streams.

- Noise filtering and Voice Activity Detection (VAD) for preprocessing.

- Error Feedback (M6) for reporting processing failures.

- AuditLogger (M11) for logging processing events and errors.

- Command Orchestrator for integration with downstream modules.

- Loading ASR model for speech-to-text processing.

## 9.2 Syntax:

### 9.2.1 Exported Constants:

- `EXPECTED_SAMPLE_RATE: int = 16000` — Expected audio sample rate in Hz.

- `EXPECTED_FORMAT: str = "PCM_16K_MONO"` — Expected audio format.

### 9.2.2 Exported Access Programs

| Name | |
|------|---|
| SpeechToTextEngine | config: AsrConfig, model_endpoint: str, api_key: str, device: str, vad: VadInte |
| processAudio | |
| reset | |
| validateAudioFormat | |
| validateModelReady | |

## 9.3 Semantics

### 9.3.1 State Variables:

- `config:  AsrConfig` — engine parameters (sample rate, frame size, thresholds).

- `model:  AsrModel` — loaded acoustic/language models and decoder state.

- `audio_buffer:  AudioBuffer` — buffered input audio awaiting processing.

### 9.3.2 Environment Variables:

- `ASR_CLOUD_ENDPOINT` — URL of the cloud-hosted ASR model.

- `ASR_API_KEY` — API key for authenticating with the cloud-hosted ASR model.

### 9.3.3 Assumptions:

- Input audio meets expected format and sample rate.

- The ASR model is accessible via the provided endpoint and API key.

### 9.3.4 Access Routine Semantics:

`SpeechToTextEngine(config, model_endpoint, api_key, device, vad, noise_filter):`

- **transition:** Initialize engine internals, allocate `audio_buffer`, and set `model_endpoint` and `api_key`.

- **output:** Initialized engine instance.

- **exception:** `InitializationError` if the endpoint or API key is invalid or inaccessible.

`processAudio(audioData):`

- **transition:** Validate audio format, preprocess (VAD, noise suppression), send audio to the cloud-based ASR model via `model_endpoint` with `api_key`, and decode the response.

- **output:** `Transcript` containing recognized text and confidence metadata.

- **exception:** `ProcessingError` if format invalid, preprocessing fails, or the cloud model is unreachable.

`reset():`

- **transition:** Clear `audio_buffer`.

- **output:** -.

- **exception:** -.

`validateAudioFormat(audio):`

- **transition:** None.

- **output:** true iff `audio.sampleRate = EXPECTED_SAMPLE_RATE` $\wedge$ `audio.format = EXPECTED_FORMAT`.

- **exception:** -.

```
validateModelReady():
```

- **transition:** None.

- **output:** `true` iff the cloud-based ASR model is accessible via `model_endpoint` and the `api_key` is valid.

- **exception:** `ModelValidationError` if the endpoint is unreachable or the API key is invalid.

### 9.3.5 Local Functions:

- `extract_features(audio: AudioStream) -> FeatureVector`: Returns feature vector (e.g., spectrogram) computed from input audio.

- `send_to_cloud(features: FeatureVector) -> Transcript`: Sends the feature vector to the cloud-based ASR model using the `model_endpoint` and `api_key`, and returns the decoded transcript.

# 10    (M5) MIS of Command Orchestrator

**Module:** CommandOrchestrator

## 10.1    Uses:

- Google Gemini API for natural language understanding and command clarification.

- User Profile Manager (M8) for personalized context and preferences.

- Error Feedback Module (M6) for reporting API or schema validation failures.

- Security Layer (M9) for managing API credentials and encryption.

## 10.2    Syntax:

### 10.2.1    Exported Constants:

- None — Configuration values are passed via constructor parameters or loaded from system prompt.

### 10.2.2    Exported Access Programs:

| Name | In | Out | Exceptions |
|------|-----|-----|------------|
| CommandOrchestrator | - | self | InitializationError |
| process | user_input: str <br> conversation_context: <br> Optional[List[dict]] | OrchestratorResponse | InferenceError |
| reset | - | - | - |
| apply_guardrails | command: String | (bool, String) | - |

## 10.3    Semantics

### 10.3.1    State Variables:

- apiKey: stores API key for LLM service authentication

- client: LLM API client instance for command clarification

- modelId: identifier for selected LLM model

- systemPrompt: current system prompt instructions for LLM

- conversationHistory: list of user/assistant turns for multi-turn context

- currentGoal: most recent clarified user intent (if any)

### 10.3.2   Environment Variables:

- GEMINI_API_KEY — API key for Google Gemini service (used if api_key parameter not provided to constructor)

### 10.3.3   Assumptions:

- LLM returns valid JSON in format {"needs_clarification":  bool, "question": String} or {"needs_clarification":  bool, "clarified_command":  String}

- System prompt effectively guides the LLM to ask for clarification only when truly necessary

- Browser controller can handle natural language commands

### 10.3.4   Access Routine Semantics

CommandOrchestrator(api_key, model_id, prompt_path):

- **transition:** Load environment variables; initialize client, set model_id (default "gemini-2.5-flash"); load system_prompt from prompt_path if provided, otherwise use default prompt; initialize empty conversation_history; set current_goal to None.

- **output:** Initialized orchestrator instance.

- **exception:** InitializationError if API key is missing or config is malformed.

process(user_input, conversation_context):

- **transition:** Build message list: add messages from conversation_context if provided, then append user_input as new user message; call _call_llm(messages) to send to Gemini with system_prompt; parse JSON response using _parse_response(); construct and return OrchestratorResponse.

- **output:** OrchestratorResponse object with fields: needs_clarification (bool), user_prompt (optional question for user), clarified_command (optional natural language command), reasoning (optional debug info), metadata (optional dict with "original_input" and other context).

- **exception:** InferenceError if LLM API call fails, times out, or returns unparsable response.

reset():

- **transition:** Clear conversation_history to empty list; set current_goal to None.

- **output:** None.

- **exception:** None.

`apply_guardrails(command)` [static method]:

- **transition:** None (pure function).

- **output:** Tuple (`allowed: bool, message: String`) where `allowed` is `False` if command contains blocked keywords AND is phrased as an imperative action; `message` contains "Sorry, this command cannot be executed because it violates safety policy." if blocked, None otherwise.

- **exception:** None.

### 10.3.5 Local Functions:

- `_get_default_system_prompt(): void → String`
  **Description:** Returns default system prompt containing instructions for command clarification, JSON output format specification, examples of when to ask clarification vs. when to proceed, and guidance to ignore filler words while extracting user intent.

- `_call_llm(messages: List[Dict[String, String]]): List[Dict] → String`
  **Description:** Constructs full message list with `system_prompt` as first user message, then appends all messages from conversation history (converting "user"/"assistant" roles to "user"/"model" for Gemini API); returns response text.

- `_parse_response(response_text: String, original_input: String): String × String → OrchestratorResponse`
  **Description:** Strips markdown code fences from response_text; parses JSON; if `needs_clarificatio` returns OrchestratorResponse with `user_prompt` set to question from JSON; if `needs_clarification` returns OrchestratorResponse with `clarified_command` from JSON; if parsing fails, returns fallback OrchestratorResponse requesting clarification with error details in `reasoning` and `metadata`.

# 11 (M6) MIS of Error Feedback

**Module:** ErrorFeedback

## 11.1 Uses:

- User Interface (M1) and Feedback Display Module (M3) to render notifications and recovery prompts.

- Accessibility Layer (M2) to ensure error messages and prompts are announced accessibly.

- AuditLogger (M16) to record error events and diagnostics.

## 11.2 Syntax

### 11.2.1 Exported Constants

None.

### 11.2.2 Exported Access Programs

| Name | In | Out | Exceptions |
|------|-----|-----|-----------|
| ErrorFeedback | notifier:  UiClient | self | - |
| show_error | code:  string, detail:  string | - | - |
| show_recovery | cmd_id:  UUID, options:  string list | - | - |
| dismiss | feedback_id:  UUID | bool | - |
| log | event:  ErrorEvent | - | - |

## 11.3 Semantics

### 11.3.1 State Variables

- `active:  map[UUID] to FeedbackItem` — currently visible items

- `notifier:  UiClient` — handle to UI notifications

### 11.3.2 Environment Variables

- `DEFAULT_LANG` — fallback locale

- `ERROR_COPY_PATH` — message templates location

### 11.3.3 Assumptions

UI client is available and permitted to display notifications.

### 11.3.4 Access Routine Semantics

`ErrorFeedback(notifier)`:

- transition: initialize `active`; bind `notifier`

- output: initialized instance

- exception: -

`show_error(code, detail)`:

- transition: create and register a feedback item in `active`; display via `notifier`

- output: -

16

- exception: -

show_recovery(cmd_id, options):

- transition: render recovery prompt with provided options

- output: -

- exception: -

dismiss(feedback_id):

- transition: remove from `active`; instruct UI to hide

- output: `true` if removed; otherwise `false`

- exception: -

log(event):

- transition: write event to audit log

- output: -

- exception: -

### 11.3.5  Local Functions

- format_error_message(string code, string detail): $string \times string \rightarrow string$
  **Description:** `format_error_message(code, detail)` returns concise user-facing error message combining code and detail.

- make_recovery(string list options): $string\ list \rightarrow RecoveryOptions$
  **Description:** `make_recovery(options)` returns recovery prompt with clickable recovery options list.

# 12  (M7) MIS of Browser Orchestrator

**Module:** BrowserOrchestrator

## 12.1  Uses

- Session Manager (M8) to obtain the active `Browser` instance used for command execution.

- `browser_use` library (`Agent`, `Browser`, `ChatBrowserUse`) to perform LLM-powered browser automation.

## 12.2 Syntax

### 12.2.1 Exported Constants

None.

### 12.2.2 Exported Access Programs

| Name | In | Out | Exceptions |
|------|----|----|-----------|
| run_command | command: str, browser: Browser | history: Any | Exception |

## 12.3 Semantics

### 12.3.1 State Variables

None.

### 12.3.2 Environment Variables

- BROWSER_USE_API_KEY — API key used by the browser_use provider (loaded from .env).

### 12.3.3 Assumptions

- A valid, started Browser instance is provided as input (typically from M8).

- The browser_use provider is configured correctly via environment variables (e.g., BROWSER_USE_API_KEY).

### 12.3.4 Access Routine Semantics

run_command(command, browser):

- transition:

  - instantiate llm := ChatBrowserUse()
  - instantiate agent := Agent(task=command, llm=llm, browser=browser)
  - execute history := await agent.run()

- output: history returned by agent.run() (may be empty/falsey)

- exception: propagates any raised Exception from browser_use or underlying runtime

### 12.3.5 Local Functions

None.

# 13 (M8) MIS of Session Manager

**Module:** SessionManager

## 13.1 Uses

- `browser_use` library (`Browser`) to create and manage a single shared browser instance.

## 13.2 Syntax

### 13.2.1 Exported Constants

None.

### 13.2.2 Exported Access Programs

| Name | In | Out | Exceptions |
|------|-----|-----|-----------|
| start_session | – | str | Exception |
| stop_session | – | str | Exception |
| get_browser | – | Browser \| None | – |

## 13.3 Semantics

### 13.3.1 State Variables

- `_browser: Browser | None` — single shared browser instance for the whole app (module-level global).

### 13.3.2 Environment Variables

None.

### 13.3.3 Assumptions

- The application requires at most one shared browser session at a time.

- When started, the browser is kept alive across commands using `keep_alive=True`.

### 13.3.4 Access Routine Semantics

start_session():

- transition:

    - if `_browser` is not `None`, no change

    - else set `_browser := Browser(keep_alive=True)` and `await _browser.start()`

- output:

    - "Browser already running" if _browser was already initialized
    - "Browser started" if a new browser was created and started

- exception: propagates any Exception from browser creation/start

stop_session():

- transition:

    - if _browser is None, no change
    - else await _browser.stop() and set _browser := None

- output:

    - "Browser already stopped" if no browser exists
    - "Browser stopped" if the browser was stopped and cleared

- exception: propagates any Exception from browser stop

get_browser():

- transition: none

- output: current value of _browser (a Browser if running, otherwise None)

- exception: –

### 13.3.5   Local Functions

None.

# 14   (M9) MIS of Data Management Module

**Module:** Data Management Module

## 14.1   Uses:

- Encryption Manager (M13) for encryption/decryption at rest and in transit (R16.3).

- Credential Manager (M12) for authenticated access to user-specific data (R16.1).

- AuditLogger (M11) to record storage and retrieval events (R16.4).

- UserProfileManager (M10) for storing personalized ASR profiles and preferences.

## 14.2   Syntax:

### 14.2.1   Exported Constants:

- MAX_STORAGE_LIMIT = 5GB per user session (configurable).

- BACKUP_INTERVAL = 24 hours.

- RETENTION_PERIOD = 90 days (per R16.3 Privacy Requirements).

### 14.2.2   Exported Access Programs:

| Name | In | Out | Exceptions |
|---|---|---|---|
| storeTranscript | transcriptData, userID | confirmationID | StorageWriteException |
| retrieveTranscript | transcriptID, userID | transcriptData | DataNotFoundException |
| backupData | userID | backupStatus | BackupFailureException |
| enforceRetentionRules | policyConfig | summaryReport | RetentionException |

## 14.3   Semantics

### 14.3.1   State Variables:

- storageRegistry: maps user data identifiers to file metadata and encryption state.

- backupSchedule: list of active backups per user.

- retentionPolicies: retention rules derived from privacy configuration (R16.3).

### 14.3.2   Environment Variables:

- DATABASE_URL: location of secure storage database.

- CLOUD_STORAGE_API: API endpoint for encrypted cloud backups.

### 14.3.3   Assumptions:

- Encryption Manager (M13) is available for encrypting all stored data.

- User authentication is validated by Credential Manager (M12) before access.

### 14.3.4   Access Routine Semantics:

storeTranscript(transcriptData, userID):

- **transition:** Adds a new entry to storageRegistry, encrypts transcript data via M13, and logs the event via M11.

- **output:** Returns confirmation ID.

- **exception:** `StorageWriteException` if quota exceeded or encryption fails.

`retrieveTranscript(transcriptID, userID):`

- **output:** Returns decrypted transcript data.

- **exception:** `DataNotFoundException` if transcriptID not found or access denied.

`backupData(userID):`

- **transition:** Performs encrypted backup of stored data to cloud (per R16.3).

- **output:** Backup status (success/failure).

`enforceRetentionRules(policyConfig):`

- **transition:** Deletes expired or policy-violating entries from storage.

- **output:** Summary of removed or archived files.

### 14.3.5   Local Functions:

- `verifyBackupIntegrity(UUID backupID): UUID → bool`
  **Description:** `verifyBackupIntegrity(backupID)` returns `true` if backup `backupID` passes integrity checks.

- `applyRetentionRule(FileMeta fileMeta): FileMeta → bool`
  **Description:** `applyRetentionRule(fileMeta)` returns `true` if file metadata `fileMeta` should be retained by retention policy.

# 15   (M10) MIS of User Profile Manager

**Module:** UserProfileManager

## 15.1   Uses:

- Credential Manager (M12) for token validation and secure login (R16.1).

- Encryption Manager (M13) for encrypting stored profiles (R16.3).

- Storage Management Module (M9) for persistence.

## 15.2   Syntax:

### 15.2.1   Exported Access Programs:

| Name | In | Out | Exceptions |
|---|---|---|---|
| createProfile | userToken: string, initData: dict | profileID | ProfileCreationExcept |
| loadPreferences | userID | preferenceData | DataNotFoundExcept |
| saveConsent | userID, consentFlags | status | ConsentException |

## 15.3  Semantics

### 15.3.1  State Variables:

- `profiles`: map of userID $\rightarrow$ profile metadata.

- `preferences`: user personalization data (R12.2).

- `consentLog`: record of consent actions (R16.3).

### 15.3.2  Assumptions:

- Consent is required prior to storing personalization data (R16.3 Privacy).

### 15.3.3  Access Routine Semantics:

`createProfile(userToken, initData)`:

- **transition:** Creates encrypted user profile and saves to M9 storage.

- **exception:** `ProfileCreationException` on invalid token.

`saveConsent(userID, consentFlags)`:

- **transition:** Updates consentLog.

- **output:** Confirmation of saved consent (R16.3).

# 16  (M11) MIS of AuditLogger

**Module:** AuditLogger

## 16.1  Uses:

- Encryption Manager (M13) for log encryption.

- Credential Manager (M12) for secure log access.

## 16.2  Syntax:

### 16.2.1  Exported Access Programs:

| Name | In | Out | Exceptions |
|------|-----|-----|-----------|
| `logEvent` | eventData, severity | logID | LogWriteException |
| `queryLogs` | filterParams, userRole | logRecords | UnauthorizedAccessException |
| `detectAnomaly` | recentLogs | anomalyReport | DetectionException |

## 16.3 Semantics

### 16.3.1 Access Routine Semantics:

`logEvent(eventData, severity):`

- **transition:** Writes signed and encrypted log entry per R16.4.

`detectAnomaly(recentLogs):`

- **output:** Triggers OutOfScopeHandler (M14) on suspicious events.

# 17 (M12) MIS of Credential Manager

**Module:** Credential Manager (CredentialManagerImpl)

## 17.1 Uses:

- Encryption Manager (M13) for secure key storage.

- AuditLogger (M11) to record authentication attempts (R16.4).

## 17.2 Syntax:

### 17.2.1 Exported Access Programs:

| Name | In | Out | Exceptions |
|------|-----|-----|-----------|
| authenticateUser | username: string, password: string | sessionToken | AuthException |
| validateToken | sessionToken: string | validity: bool | TokenException |
| rotateKeys | scheduleID | status: string | KeyRotationExc |

## 17.3 Semantics

### 17.3.1 Access Routine Semantics:

`authenticateUser(username, password):`

- **transition:** Validates credentials via password vault; issues signed token (R16.1).

`rotateKeys(scheduleID):`

- **transition:** Calls M13 to rotate key pairs per R16.3.

# 18 (M13) MIS of Encryption Manager

**Module:** Encryption Manager (EncryptionManagerImpl)

## 18.1 Uses:

- None — foundational security service.

## 18.2 Syntax:

### 18.2.1 Exported Access Programs:

| Name | In | Out | Exceptions |
|------|-----|-----|------------|
| `encryptData` | `plainData: string, keyID: UUID` | cipherData | EncryptionException |
| `decryptData` | cipherData, keyID | plainData | DecryptionException |
| `rotateKeys` | rotationPolicy | `result: bool` | RotationFailureException |
| `verifyIntegrity` | dataBlob, signature | validFlag | IntegrityException |

## 18.3 Semantics

Implements R16.3 (Privacy Requirements): ensures all data is encrypted at rest/in transit.
Implements R16.2 (Integrity): validates message hashes before use.

# 19 (M14) MIS of OutOfScopeHandler

**Module:** OutOfScopeHandler

## 19.1 Uses:

- AuditLogger (M11) for incident reporting (R16.4).

- Command Orchestrator (M5) for validation of user commands.

- Error Handling & Recovery (M11) for rollback actions.

## 19.2 Syntax:

### 19.2.1 Exported Access Programs:

| Name | In | Out | Exceptions |
|------|-----|-----|------------|
| `validateCommand` | `commandText: string, context: dict` | validationResult | InvalidCommandEx |
| `reportIncident` | `incidentData: dict` | reportID | ReportFailureExcep |
| `recoverState` | sessionID | `status: bool` | RecoveryException |

## 19.3 Semantics

Implements R16.5 (Immunity Requirements): ensures robustness against unsafe operations.
Integrates with M11 to log anomaly-triggered safety events.

### 19.3.1 Access Routine Semantics:

`validateCommand(commandText):`

- **transition:** Compares command against whitelist (policy from config).

`recoverState(sessionID):`

- **transition:** Restores safe prior system state through M11.

# 20 (M15) MIS of MicrophoneManager

**Module:** MicrophoneManager

## 20.1 Uses:

- VAD Noise Filter (M16) for downstream speech/noise classification.
- Session Manager (M8) for tracking active input sessions.
- AuditLogger (M11) for logging permission events or device failures.

## 20.2 Syntax:

### 20.2.1 Exported Access Programs:

| Name | In | Out | Exceptions |
|------|-----|-----|------------|
| listDevices | - | deviceList | DeviceQueryException |
| startCapture | deviceID | status: bool | PermissionException |
| readFrame | - | audioFrame | CaptureInactiveException |

## 20.3 Semantics

Implements audio acquisition requirements (FR1). Provides consistent, normalized microphone input to subsequent modules. Integrates with M8 to maintain session continuity.

### 20.3.1 Access Routine Semantics:

`startCapture(deviceID):`

- **transition:** Initializes device stream and marks microphone as active.

`readFrame():`

- **transition:** Retrieves the latest audio frame for processing.

# 21  (M16) MIS of VADNoiseFilter

**Module:** VADNoiseFilter

## 21.1  Uses:

- MicrophoneManager (M15) for raw audio frames.

- Speech-to-Text Engine (M4) for improved transcription accuracy.

- AuditLogger (M11) for tracking confidence drops or noise anomalies.

## 21.2  Syntax:

### 21.2.1  Exported Access Programs:

| Name | In | Out | Exceptions |
|------|----|----|-----------|
| filterFrame | audioFrame: AudioFrame | cleanedFrame: AudioFrame | - |
| detectSpeech | audioFrame: AudioFrame | isSpeech: bool | - |
| resetState | - | status: bool | - |

## 21.3  Semantics

Supports Integrity Requirements (IR2) by reducing background noise prior to transcription. Provides binary VAD detections for downstream timing and segmentation.

### 21.3.1  Access Routine Semantics:

filterFrame(frame):

- **transition:** Updates internal noise profile and returns processed audio.

detectSpeech(frame):

- **transition:** Computes speech probability using configured thresholds.

# 22  (M17) MIS of PromptingModule

**Module:** PromptingModule

## 22.1  Uses:

- Command Orchestrator (M5) for phrasing confirmations.

- Error Feedback (M6) for generating user-facing explanations.

- Accessibility Layer (M2) for formatting prompts according to accessibility rules.

## 22.2   Syntax:

### 22.2.1   Exported Access Programs:

| Name | In | Out | Exceptions |
|------|-----|-----|------------|
| makePrompt | uiState: UiState | promptText: string | - |
| makeConfirm | intent: Intent | promptText: string | - |
| makeErrorPrompt | errorData: dict | promptText: string | - |

## 22.3   Semantics

Supports usability requirements (UH-1, UH-4) and cultural neutrality (CUL-1). Ensures consistent phrasing across confirmations, errors, and system messages.

### 22.3.1   Access Routine Semantics:

`makeConfirm(intent)`:

- **transition:** Constructs a confirmation string based on target action.

`makeErrorPrompt(errorData)`:

- **transition:** Builds a polite, accessible error message.

# 23   (M18) MIS of ModelTuner

**Module:** ModelTuner

## 23.1   Uses:

- Data Management Layer (M9) for retrieving stored audio samples.

- ASR Engine (M4) for updating recognition parameters.

- AuditLogger (M11) for recording tuning events or failures.

## 23.2   Syntax:

### 23.2.1   Exported Access Programs:

| Name | In | Out | Exceptions |
|------|-----|-----|------------|
| scheduleTune | userID, dataset | jobID | TunePolicyException |
| checkStatus | jobID | status | NotFoundException |
| applyModel | userID, jobID | success: bool | ModelLoadException |

## 23.3 Semantics

Supports Accuracy Requirements (PF-3) by enabling adaptive personalization for impaired speech. Updates the user's model profile upon successful tuning.

### 23.3.1 Access Routine Semantics:

`scheduleTune(userID, dataset):`

- **transition:** Registers a tuning job with the training backend.

`applyModel(userID, jobID):`

- **transition:** Loads and activates the newly tuned ASR parameters.

# References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering.* Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach.* International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

# 24    Appendix

Not applicable for this document.

# Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

1. **What went well while writing this deliverable?** Writing this deliverable progressed smoothly due to effective collaboration and clear communication among team members. The team successfully maintained consistency in formatting and terminology throughout the document, resulting in a cohesive and professional presentation. Peer reviews helped identify and resolve ambiguities early, which enhanced the overall quality and clarity of the deliverable.

2. **What pain points did you experience during this deliverable, and how did you resolve them?** Some challenges arose in maintaining consistent LaTeX formatting, especially with complex tables and detailed semantic descriptions. To resolve these issues, the team adopted established LaTeX best practices and created templates to ensure uniformity. Additionally, scheduled review meetings allowed the team to discuss and fix formatting inconsistencies collaboratively.

3. **Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g., your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?** Many design decisions, particularly those related to user interface and accessibility, were influenced by discussions with potential users and stakeholders. Their feedback emphasized the importance of compliance with accessibility standards and intuitive user workflows. Decisions not directly influenced by client input were grounded in recognized industry standards, prior project experience, and academic literature.

4. **While creating the design doc, what parts of your other documents (e.g., requirements, hazard analysis, etc), if any, needed to be changed, and why?** During the design process, updates were made to the requirements and hazard analysis documents to reflect clarified module responsibilities and interface definitions. For example, some interface details were refined to improve modularity and error handling based on design insights. These changes ensured alignment and consistency across all project documentation.

5. **What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)** The current solution has limitations regarding scalability and adaptability in diverse deployment scenarios. With unlimited resources, we would invest in extensive automated testing frameworks and incorporate advanced machine learning techniques for improved personalization and robustness. Additionally, comprehensive accessibility audits and user testing would be expanded to further enhance usability.

6. **Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)** Alternative designs considered included monolithic architectures and different modular breakdowns. Monolithic design offered simplicity but reduced maintainability and scalability. Various modular structures were explored; the chosen modular design balances separation of concerns, ease of integration, and parallel development capability. This approach was selected to optimize maintainability and accommodate future enhancements efficiently.

7. **What did you learn by implementing another team's module? Were all the details you needed in the documentation, or did you need to make assumptions, or ask the other team questions? If your team also had another team implement one of your modules, what was this experience like? Are there things in your documentation you could have changed to make the process go more smoothly for when an "outsider" completes some of the implementation?** Implementing the Mel Filter Module demonstrated how much a formal mathematical foundation simplifies cross-team collaboration. Because the documentation provided a specific matrix summation formula, we could translate signal processing requirements into C++ logic with very little ambiguity. We briefly struggled with the index mapping for the filterbank matrix $H$, but the provided semantic notation allowed us to resolve the structure ourselves. This experience showed that a rigorous mathematical contract reduces the need for an external implementer to make arbitrary assumptions or constantly ask the original designers for clarification. In contrast, having another team implement our User Profile module revealed that our specifications relied too heavily on implicit context. Since we didn't formalize the internal logic or state changes, the module's behavior wasn't immediately clear to an outsider. To make our documentation more self-contained, we should have included detailed logic in the future. This highlighted that documentation requires a much higher level of formal detail when the developer is not the original designer.