Module Interface Specification for Software Engineering

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

January 12, 2026

1 Revision History

Date	Version	Notes
Nov 12, 2025	1.0	Added Modules M1-M22
Jan 11, 2026	1.1	Updated local functions

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

8

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	Description
character	char	A single UTF–8 encoded character. Used for transcript tokens, UI labels, and encoded metadata fields.
integer	$\mathbb{Z}$	Integer values used for counters, timestamps, retry counts, audio frame lengths, and configuration parameters.
natural number	$\mathbb{N}$	Positive integer values used for unique identifiers, session IDs, buffer sizes, and timeout values.
real	$\mathbb{R}$	Numerical values used for confidence scores, normalized audio energy, contrast ratios, and timing measurements (seconds).
boolean	bool	Logical value in {true, false}. Used frequently across validation, VAD detections, policy checks, etc.

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 BrowserController 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 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.

- Intent Interpreter (M5) and Command Mapping (M6) for forwarding validated user events.

- BrowserController / 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	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 Intent Interpreter (M5) / Command Mapping (M6) 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.

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 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): rgb × rgb → 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): string × AccessMetadata → bool`
  Description: `aria_set(elemId, metadata)` returns `true` if ARIA attributes from `metadata` are applied to element `elemId` (used by `applySettings`)

8   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 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.

- Personalization/Profile store for per-user adaptation.

- Error Feedback (M8) and AuditLogger (M14) for reporting processing failures.

9.2 Syntax:

9.2.1 Exported Constants:

- None.

9.2.2 Exported Access Programs:

Name	In	Out	Exceptions
`SpeechToTextEngine`	`config:  AsrConfig`	self	InitializationError
`processAudio`	`audioData:  AudioStream`	`Transcript`	ProcessingError
`reset`	`-`	`-`	`-`
`validateAudioFormat`	`audioData:  AudioStream`	bool	`-`
`validateModelReady`	`-`	bool	`ModelValidationError`

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:

- None.

9.3.3 Assumptions:

- None.

9.3.4 Access Routine Semantics:

`SpeechToTextEngine(config)`:

- **transition:** initialize engine internals, allocate `audio_buffer`, and load `model`.

- **output:** initialized engine instance.

- **exception:** `InitializationError` if models or resources are missing.

`processAudio(audioData)`:

- **transition:** validateAudioFormat(audioData) $\wedge$ preprocess (VAD, noise suppression), extract features, decode with models, and apply personalization.

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

- **exception:** `ProcessingError` if format invalid or processing fails.

`reset()`:

- **transition:** clear `audio_buffer` and reset decoder/model state.

- **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 model is loaded $\wedge$ model is validated

- **exception:** `exc := ModelValidationError` if model files missing or corrupted

9.3.5 Local Functions:

- extract_features(AudioStream audio): AudioStream $\rightarrow$ FeatureVector
  Description: extract_features(audio) returns feature vector (spectrogram, MFCCs) computed from input audio.

- decode(FeatureVector features): FeatureVector $\rightarrow$ Transcript
  Description: decode(features) returns recognized text and confidence scores from model decoding.

- apply_personalization(Transcript transcript, UserProfile profile): Transcript $\times$ UserProfile $\rightarrow$ Transcript
  Description: apply_personalization(transcript, profile) returns transcript adjusted using user profile adaptation.

10 MIS of Intent Interpreter

Module: IntentInterpreter

10.1 Uses:

- Speech-to-Text Engine (M4) for input transcripts.

- Intent Schema Registry for parsing and intent extraction.

- Error Feedback (M8) for reporting ambiguous or failed interpretations.

10.2 Syntax:

10.2.1 Exported Constants:

- `MIN_CONFIDENCE = 0.5`

- `EXPECTED_TRANSCRIPT_FORMAT = ``text + confidence''`

10.2.2 Exported Access Programs:

Name	In	Out	Exceptions
`IntentInterpreter`	-	self	InitializationError
`interpret`	`transcript: Transcript`	Intent	InterpretationError
`reset`	-	-	-
`validateTranscript`	`transcript: Transcript`	bool	-

10.3 Semantics

10.3.1 State Variables:

- `intent_schemas: IntentSchemaRegistry` — canonical intent definitions and slot schemas.

- `context: DialogContext` — context for multi-turn dialog and disambiguation.

10.3.2 Environment Variables:

- None.

10.3.3 Assumptions:

- None.

10.3.4 Access Routine Semantics

`IntentInterpreter()`:

- **Transition:** Load `intent_schemas` and initialize `context` for dialog state.

- **Output:** Initialized interpreter instance ready to process transcripts.

- **Exception:** `InitializationError` if required schema resources are missing or invalid.

`interpret(transcript)`:

- **transition:** validateTranscript(transcript) $\wedge$ parse input $\wedge$ match against intent schemas $\wedge$ extract slots

- **output:** `Intent` object containing intent name, slots, and confidence score

- **exception:** `InterpretationError` if transcript invalid or no confident intent found

`reset()`:

- **transition:** Clear `context` and reset transient parsing caches.

- **output:** Boolean flag indicating success of reset operation.

- **exception:** -

`validateTranscript(transcript)`:

- **transition:** None

- **output:** true iff `transcript.text` $\neq \emptyset$ $\wedge$ `transcript.confidence` $>$ `MIN_CONFIDENCE`

- **exception:** None

10.3.5 Local Functions:

- `match_intent(Transcript text)`: `Transcript` $\rightarrow$ `Intent`
  Description: `match_intent(text)` returns the highest-scoring intent that matches the input transcript.

- `extract_slots(Transcript text, Intent intent)`: `Transcript` $\times$ `Intent` $\rightarrow$ `Slots`
  Description: `extract_slots(text, intent)` returns parameter values (slots) extracted from transcript for the given intent.

11 MIS of Command Mapping Module

Module: CommandMapping

11.1 Uses:

- Intent Interpreter (M5) for structured intents.
- Command Registry for available executable actions and metadata.
- User Preferences / Policy store to apply customization and guardrails.
- Error Feedback (M8) for reporting mapping or policy failures.

11.2 Syntax:

11.2.1 Exported Constants:

- None.

11.2.2 Exported Access Programs:

Name	In	Out	Exceptions
`CommandMapping`	-	self	InitializationError
`mapIntent`	`intent: Intent`	`Command`	MappingError
`validate`	`command: Command`	`bool`	-
`reset`	-	-	-

11.3 Semantics

11.3.1 State Variables:

- `command_registry: CommandRegistry` — available commands, parameter schemas, and metadata.
- `preferences: UserPreferences` — mapping rules and user-specific overrides.
- `policy: Policy` — mapping guardrails and authorization rules.

11.3.2 Environment Variables:

- None.

11.3.3 Assumptions:

– Intent objects conform to the registered schema; command registry is current and authoritative.

11.3.4 Access Routine Semantics

`CommandMapping()`:

– **transition:** Load `command_registry`, `preferences`, and `policy` configurations.
– **output:** Initialized mapping instance ready to process intents.
– **exception:** `InitializationError` if registry or policy resources are missing or invalid.

`mapIntent(intent)`:

– **transition:** Select candidate commands matching the `intent`; apply `preferences` and enforce `policy` constraints; produce a validated `Command`.
– **output:** Executable `Command` object representing the mapped device or browser action.
– **exception:** `MappingError` if no suitable mapping exists or policy prevents execution.

`validate(command)`:

– **transition:** Execute guardrail checks (authorization, confirmation) against `command`.
– **output:** `true` if `command` passes all validation criteria; otherwise `false`.
– **exception:** -.

`reset()`:

– **transition:** Clear internal caches and reload default configuration.
– **output:** Boolean indicating success of the reset operation.
– **exception:** -.

11.3.5 Local Functions:

– `find_command(Intent intent)`: Intent $\rightarrow$ Command
Description: `find_command(intent)` returns the best matching command from the registry for the given intent.

– `apply_preferences(Command cmd, UserPreferences prefs)`: Command × UserPreferences
→ Command
Description: `apply_preferences(cmd, prefs)` returns command with parameters adjusted by user preferences.

– `enforce_policy(Command cmd, Policy policy)`: Command × Policy → bool
Description: `enforce_policy(cmd, policy)` returns `true` if command passes all policy checks.

12 MIS of Command Execution Layer

Module: CommandExecutionLayer

12.1 Uses:

- BrowserController (M14) to send `BackendReq` requests and receive `BackendResp` results.
- Session Manager (M15) to associate `cmd_id` with an active session (when applicable).
- Error Handling & Recovery (M16) for standardized classification, retry/backoff, and compensation decisions.
- AuditLogger (M19) to record execution attempts, outcomes, and failures.
- Error Feedback (M13) to surface user-facing execution errors and recovery prompts.

12.2 Syntax

12.2.1 Exported Constants

None.

12.2.2 Exported Access Programs

Name	In	Out	Exceptions
CommandExecLayer	backend: Client,	self	-
execute	cmd: ExecCmd	Result	PermissionError, TimeoutError
cancel	cmd_id: UUID	bool	NotFoundError
status	cmd_id: UUID	ExecStatus	NotFoundError
validate	cmd: ExecCmd	bool	-
map_to_backend	cmd: ExecCmd	BackendReq	MappingError
rollback	cmd_id: UUID	bool	ExecError

12.3 Semantics

12.3.1 State Variables

- `pending_q: map[UUID]→ExecCtx` — commands in flight
- `default_timeout_s: int` — global timeout for executions

- `retries: int` — max retry attempts for transient failures
- `backend: Client` — handle to browser automation bridge
- `audit: Logger` — sink for execution logs

12.3.2 Environment Variables

- `BROWSER_BRIDGE_URL` — connection string for automation client
- `EXEC_HARD_LIMIT_S` — absolute upper bound on execution time
- `LOG_LEVEL` — audit verbosity

12.3.3 Assumptions

12.3.4 Access Routine Semantics

`CommandExecLayer(backend, timeout_s)`:

- transition: initialize `pending_q`, set `default_timeout_s`, bind `backend`, configure `audit`
- output: initialized instance
- exception: -

`execute(cmd)`:

- transition: add to `pending_q`; invoke `map_to_backend`; dispatch to `backend`; update status; remove on completion
- output: `Result` with success flag, message, and optional payload
- exception: `PermissionError` if disallowed; `TimeoutError` if exceeds limits; `ExecError` on backend failure

`cancel(cmd_id)`:

- transition: signal cancellation to backend; mark context as cancelled; remove from `pending_q`
- output: `true` if cancelled; otherwise `false`
- exception: `NotFoundError` if `cmd_id` not tracked

`status(cmd_id)`:

- transition: none
- output: `ExecStatus` in {`queued`, `running`, `succeeded`, `failed`, `cancelled`}

– exception: `NotFoundError` if unknown

`validate(cmd)`:

– transition: none
– output: `true` iff command matches allowed action set and guardrails (e.g., requires-confirmation flags met)
– exception: -

`map_to_backend(cmd)`:

– transition: none
– output: `BackendReq` (normalized request for the bridge)
– exception: `MappingError` if no mapping exists

`rollback(cmd_id)`:

– transition: attempt compensating operation (e.g., reopen tab, revert text entry)
– output: `true` on success; otherwise `false`
– exception: `ExecError` if rollback fails

12.3.5 Local Functions

– `start_timer(ExecCtx ctx): ExecCtx → Timer`
Description: `start_timer(ctx)` returns timer object that tracks timeout for execution context `ctx`.

– `complete(ExecCtx ctx, Result result): ExecCtx × Result → bool`
Description: `complete(ctx, result)` returns `true` if execution context is finalized and cleaned up.

– `is_allowed(ExecCmd cmd): ExecCmd → bool`
Description: `is_allowed(cmd)` returns `true` if command passes guardrails and policy checks.

– `retryable(Error err): Error → bool`
Description: `retryable(err)` returns `true` if error type can be safely retried.

– `to_backend(ExecCmd cmd): ExecCmd → BackendReq`
Description: `to_backend(cmd)` returns normalized backend request for browser bridge.

13 MIS of Error Feedback

Module: ErrorFeedback

13.1 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.

- Error Handling & Recovery (M16) to standardize error classification and map internal errors to user-facing recovery options.

- AuditLogger (M19) to record error events and diagnostics.

13.2 Syntax

13.2.1 Exported Constants

None.

13.2.2 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	-	-

13.3 Semantics

13.3.1 State Variables

- active: map[UUID] to FeedbackItem — currently visible items
- notifier: UiClient — handle to UI notifications

13.3.2 Environment Variables

- DEFAULT_LANG — fallback locale
- ERROR_COPY_PATH — message templates location

13.3.3 Assumptions

UI client is available and permitted to display notifications.

13.3.4 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: -
- – 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: -

13.3.5 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.

14 MIS of BrowserController

Module: BrowserController

14.1 Uses

- Credential Manager (M20) to obtain/validate authentication material required to call the controller endpoint (when applicable).
- Error Handling & Recovery (M16) to standardize timeout/retry decisions and error classification for transport failures.
- AuditLogger (M19) to record request/response summaries, failures, and timeouts.

14.2 Syntax

14.2.1 Exported Constants

None.

14.2.2 Exported Access Programs

Name	In	Out	Exceptions
BrowserController	bridge: Client, timeout_s: N	self	–
send	req: BackendReq	BackendResp	TimeoutError, Trans...
get_status	cmd_id: UUID	ExecStatus	NotFoundError
cancel	cmd_id: UUID	bool	NotFoundError
open_session	user_id: UUID	SessionId	TransportError
close_session	session_id: SessionId	bool	TransportError

14.3 Semantics

14.3.1 State Variables

- bridge: Client — transport to the browser controller
- default_timeout_s: int — call timeout
- sessions: set[SessionId] — open sessions

14.3.2 Environment Variables

- BROWSER_BRIDGE_URL — endpoint for the controller

14.3.3 Assumptions

Controller endpoint is reachable and authenticated.

14.3.4 Access Routine Semantics

BrowserController(bridge, timeout_s):

- transition: set bridge, default_timeout_s, clear sessions
- output: initialized instance
- exception: -

send(req):

- transition: none
- output: BackendResp from controller
- exception: TimeoutError, TransportError on failures

get_status(cmd_id):

- transition: none
- output: current ExecStatus
- exception: NotFoundError if unknown

cancel(cmd_id):

- transition: signal cancellation upstream
- output: true if accepted
- exception: NotFoundError if unknown

open_session(user_id):

- transition: create session; add to sessions
- output: SessionId
- exception: TransportError if controller rejects

close_session(session_id):

- transition: close remotely; remove from sessions
- output: true on success; otherwise false
- exception: TransportError on failure

14.3.5 Local Functions

- with_timeout(ClientCall call): ClientCall → ClientCall
  Description: with_timeout(call) returns wrapped call that fails if it exceeds timeout limit.

- normalize(BackendResp resp): BackendResp → BackendResp
  Description: normalize(resp) returns controller response with standardized fields and error codes.

15 MIS of Session Manager

Module: SessionManager

15.1 Uses

15.2 Uses:

- Storage Management Module (M17) as the persistent store/cache for session state.
- AuditLogger (M19) to record session lifecycle events (start/stop/expiry) and state transitions.

15.2 Syntax

15.2.1 Exported Constants

None.

15.2.2 Exported Access Programs

Name	In	Out	Exceptions
SessionManager	store: Store, ttl_s: int	self	-
start	user_id: UUID	SessionId	StoreError
stop	SessionId	bool	StoreError
get	SessionId	SessionState	NotFoundError
attach_command	SessionId, cmd_id: UUID	bool	NotFoundError
set_state	SessionId, state: SessionState	bool	StoreError

15.3 Semantics

15.3.1 State Variables

- store: Store — persistence for sessions
- ttl_s: int — idle expiry threshold
- index: map[SessionId] to SessionState — in-memory cache

15.3.2 Environment Variables

- SESSION_TTL_S — default idle timeout

15.3.3 Assumptions

Store operations are atomic per session key.

15.3.4 Access Routine Semantics

SessionManager(store, ttl_s):

- transition: set fields; warm cache from store if available
- output: initialized instance
- exception: -

start(user_id):

- transition: create session; write to store; cache in index
- output: SessionId
- exception: StoreError on failure

stop(session_id):

- transition: mark closed; evict from index; update store
- output: true on success; otherwise false
- exception: StoreError on failure

get(session_id):

- transition: none
- output: current SessionState
- exception: NotFoundError if unknown

`attach_command(session_id, cmd_id)`:

- transition: append command reference to session state
- output: `true` if attached
- exception: `NotFoundError` if session unknown

`set_state(session_id, state)`:

- transition: update state in cache and store
- output: `true` if updated
- exception: `StoreError` on write failure

15.3.5   Local Functions

- `now()`: → `timestamp`
  Description: `now()` returns the current system timestamp.
- `expired(SessionState state)`: `SessionState` → `bool`
  Description: `expired(state)` returns `true` if the session state has exceeded its idle expiry time.

16   MIS of Error Handling & Recovery Module

Module: ErrorHandler

16.1   Uses

- Command Execution Layer (M12) to schedule retries, trigger rollback/compensation, and query execution context.
- BrowserController (M14) to coordinate recovery actions involving the browser automation bridge.
- Session Manager (M15) to update or invalidate session state during recovery (when applicable).
- AuditLogger (M19) to persist error events, classifications, and recovery actions taken.

16.2   Syntax

16.2.1   Exported Constants

None.

16.2.2 Exported Access Programs

Name	In	Out	Exceptions
ErrorHandler	policy: Policy	self	-
handle	err: ExecError, ctx: ExecCtx	ActionResult	-
retry	cmd_id: UUID	bool	PolicyError
compensate	cmd_id: UUID	bool	ExecError
classify	err: ExecError	ErrorClass	-
record	err: ExecError, ctx: ExecCtx	UUID	-

16.3 Semantics

16.3.1 State Variables

- policy: Policy — retry and compensation rules
- history: map[UUID] to ErrorEvent — recent errors
- backoff: map[UUID] to int — current backoff in seconds

16.3.2 Environment Variables

- MAX_RETRIES — hard cap for attempts
- BASE_BACKOFF_S — initial delay

16.3.3 Assumptions

Compensation actions are idempotent or guarded.

16.3.4 Access Routine Semantics

ErrorHandler(policy):

- transition: set policy; clear history, backoff
- output: initialized instance
- exception: -

handle(err, ctx):

- transition: classify; decide action (retry, compensate, fail); update history, backoff
- output: ActionResult describing action taken

– exception: -

retry(cmd_id):

- transition: schedule retry with policy backoff
- output: true if scheduled
- exception: PolicyError if retries exhausted

compensate(cmd_id):

- transition: execute compensation; update state
- output: true on success; otherwise false
- exception: ExecError if compensation fails

classify(err):

- transition: none
- output: ErrorClass such as transient, permanent, or user
- exception: -

record(err, ctx):

- transition: persist error event; update history
- output: event identifier UUID
- exception: -

16.3.5 Local Functions

- calc_backoff(UUID cmd_id): UUID → N
  Description: calc_backoff(cmd_id) returns the next retry delay (seconds) for command cmd_id.

- is_idempotent(UUID cmd_id): UUID → bool
  Description: is_idempotent(cmd_id) returns true if command cmd_id is safe for compensation/rollback.

17 MIS of Storage Management Module

Module: Storage Management Module

17.1 Uses:

– Encryption Manager (M16) for encryption/decryption at rest and in transit (R16.3).
– Credential Manager (M15) for authenticated access to user-specific data (R16.1).
– AuditLogger (M14) to record storage and retrieval events (R16.4).
– UserProfileManager (M13) for storing personalized ASR profiles and preferences.

17.2 Syntax:

17.2.1 Exported Constants:

– `MAX_STORAGE_LIMIT` = 5GB per user session (configurable).
– `BACKUP_INTERVAL` = 24 hours.
– `RETENTION_PERIOD` = 90 days (per R16.3 Privacy Requirements).

17.2.2 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

17.3 Semantics

17.3.1 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).

17.3.2 Environment Variables:

– `DATABASE_URL`: location of secure storage database.
– `CLOUD_STORAGE_API`: API endpoint for encrypted cloud backups.

17.3.3 Assumptions:

– Encryption Manager (M16) is available for encrypting all stored data.
– User authentication is validated by Credential Manager (M15) before access.

17.3.4 Access Routine Semantics:

`storeTranscript(transcriptData, userID):`

– **transition:** Adds a new entry to `storageRegistry`, encrypts transcript data via M16, and logs the event via M14.
– **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.

17.3.5 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.

18 MIS of User Profile Manager

Module: UserProfileManager

18.1   Uses:

– Credential Manager (M15) for token validation and secure login (R16.1).
– Encryption Manager (M16) for encrypting stored profiles (R16.3).
– Storage Management Module (M12) for persistence.

18.2   Syntax:

18.2.1   Exported Access Programs:

Name	In	Out	Exceptions
createProfile	userToken: string, initData: dict	profileID	ProfileCreationE
loadPreferences	userID	preferenceData	DataNotFoundE
saveConsent	userID, consentFlags	status	ConsentExceptio

18.3   Semantics

18.3.1   State Variables:

– `profiles`: map of userID $\rightarrow$ profile metadata.
– `preferences`: user personalization data (R12.2).
– `consentLog`: record of consent actions (R16.3).

18.3.2   Assumptions:

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

18.3.3   Access Routine Semantics:

createProfile(userToken, initData):

– **transition:** Creates encrypted user profile and saves to M12 storage.
– **exception:** `ProfileCreationException` on invalid token.

saveConsent(userID, consentFlags):

– **transition:** Updates consentLog.
– **output:** Confirmation of saved consent (R16.3).

19 MIS of AuditLogger

Module: AuditLogger

19.1 Uses:

– Encryption Manager (M16) for log encryption.
– Credential Manager (M15) for secure log access.

19.2 Syntax:

19.2.1 Exported Access Programs:

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

19.3 Semantics

19.3.1 Access Routine Semantics:

logEvent(eventData, severity):

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

detectAnomaly(recentLogs):

– **output:** Triggers OutOfScopeHandler (M17) on suspicious events.

20 MIS of Credential Manager

Module: Credential Manager (CredentialManagerImpl)

20.1 Uses:

– Encryption Manager (M16) for secure key storage.
– AuditLogger (M14) to record authentication attempts (R16.4).

20.2 Syntax:

20.2.1 Exported Access Programs:

Name	In	Out	Exception
`authenticateUser`	`username: string, password: string`	sessionToken	AuthExcep
`validateToken`	`sessionToken: string`	`validity: bool`	TokenExce
`rotateKeys`	scheduleID	`status: string`	KeyRotatio

20.3 Semantics

20.3.1 Access Routine Semantics:

`authenticateUser(username, password):`

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

`rotateKeys(scheduleID):`

  – **transition:** Calls M16 to rotate key pairs per R16.3.

21 MIS of Encryption Manager

Module: Encryption Manager (EncryptionManagerImpl)

21.1 Uses:

  – None — foundational security service.

21.2 Syntax:

21.2.1 Exported Access Programs:

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

21.3 Semantics

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

22 MIS of OutOfScopeHandler

Module: OutOfScopeHandler

22.1 Uses:

- AuditLogger (M14) for incident reporting (R16.4).
- Command Mapping Module (M6) for validation of user commands.
- Error Handling & Recovery (M11) for rollback actions.

22.2 Syntax:

22.2.1 Exported Access Programs:

Name	In	Out	Exceptions
validateCommand	commandText: string, context: dict	validationResult	InvalidComma
reportIncident	incidentData: dict	reportID	ReportFailure
recoverState	sessionID	status: bool	RecoveryExce

22.3 Semantics

Implements R16.5 (Immunity Requirements): ensures robustness against unsafe operations. Integrates with M14 to log anomaly-triggered safety events.

22.3.1 Access Routine Semantics:

validateCommand(commandText):

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

recoverState(sessionID):

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

23   MIS of MicrophoneManager

Module: MicrophoneManager

23.1   Uses:

– VAD Noise Filter (M19) for downstream speech/noise classification.
– Session Manager (M10) for tracking active input sessions.
– AuditLogger (M14) for logging permission events or device failures.

23.2   Syntax:

23.2.1   Exported Access Programs:

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

23.3   Semantics

Implements audio acquisition requirements (FR1). Provides consistent, normalized microphone input to subsequent modules. Integrates with M10 to maintain session continuity.

23.3.1   Access Routine Semantics:

startCapture(deviceID):

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

readFrame():

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

24   MIS of VADNoiseFilter

Module: VADNoiseFilter

24.1 Uses:

- MicrophoneManager (M18) for raw audio frames.
- Speech-to-Text Engine (M4) for improved transcription accuracy.
- AuditLogger (M14) for tracking confidence drops or noise anomalies.

24.2 Syntax:

24.2.1 Exported Access Programs:

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

24.3 Semantics

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

24.3.1 Access Routine Semantics:

filterFrame(frame):

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

detectSpeech(frame):

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

25 MIS of PromptingModule

Module: PromptingModule

25.1 Uses:

- Intent Interpreter (M5) for phrasing confirmations.
- Error Feedback (M8) for generating user-facing explanations.
- Accessibility Layer (M2) for formatting prompts according to accessibility rules.

25.2 Syntax:

25.2.1 Exported Access Programs:

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

25.3 Semantics

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

25.3.1 Access Routine Semantics:

`makeConfirm(intent):`

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

`makeErrorPrompt(errorData):`

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

26 MIS of ModelTuner

Module: ModelTuner

26.1 Uses:

– Data Management Layer (M12) for retrieving stored audio samples.
– ASR Engine (M4) for updating recognition parameters.
– AuditLogger (M14) for recording tuning events or failures.

26.2 Syntax:

26.2.1 Exported Access Programs:

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

26.3   Semantics

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

26.3.1   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.

27   MIS of InstructionRegistry

Module: InstructionRegistry

27.1   Uses:

  – Command Mapping Module (M6) for resolving intent-to-command mappings.
  – Data Management Layer (M12) for persistent storage of registered instructions.
  – AuditLogger (M14) for tracking updates to command schemas.

27.2   Syntax:

27.2.1   Exported Access Programs:

Name	In	Out	Exceptions
lookup	intent: Intent	instruction: Command	NotFoundException
registerInstr	instruction: Command	status: bool	StoreException
updateInstr	id, instruction	status	StoreException

27.3   Semantics

Supports command mapping requirements (FR4) and safety guardrails (SEC-2). Maintains the authoritative index of allowed command definitions.

27.3.1 Access Routine Semantics:

`lookup(intent):`

– **transition:** Retrieves matching instruction from registry.

`updateInstr(id, instruction):`

– **transition:** Overwrites stored schema with the updated definition.

28   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.