Module Interface Specification for Bridging Gaps: AI for Diagram Accessibility

Team 22, Reading4All
Nawaal Fatima
Dhruv Sardana
Fiza Sehar
Moly Mikhail
Casey Francine Bulaclac

November 13, 2025

1 Revision History

Date	Version	Notes
November 13, 2025	1.0	Rev-1 Design Document

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at SRS (2025)

Contents

3 Introduction

The following document details the Module Interface Specifications for the Bridging Gaps: AI for Diagram Accessibility system, **Reading4All**. Reading4All is an artificial intelligence (AI)/machine learning (ML) tool that provides detailed and contextually aware alternative text for complicated technical graphics, notably those found in postsecondary Science, Technology, Engineering, and Mathematics (STEM) course materials.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/4G06-CAPSTONE-2025/Reading4All/.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Bridging Gaps: AI for Diagram Accessibility.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
boolean	$\mathbb{B}$	True, False

The specification of Bridging Gaps: AI for Diagram Accessibility uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Bridging Gaps: AI for Diagram Accessibility uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	N/A
Behaviour-Hiding Module	DataPreprocess Module WCAGCompliance Module ModelOutput Module AuthenticationService Module SessionManagement Module BackendController Module Logging Module ImageValidation Module UserInterfaceInteractions Module MainScreen Module ShowHistoryScreen Module LogInScreen Module UserInterfaceAccessibility Module
Software Decision Module	AIModelTraining Module CaptionGeneration Module InternalMetricCompliance FeedbackMetrics Module FeedbackLoop Module Feedback Module

Table 1: Module Hierarchy

6 MIS of DataPreprocess Module

6.1 Module

`DataPP`

6.2 Uses

None.

6.3 Syntax

6.3.1 Exported Constants

None.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
filterInput	imageFilePath:String	PreprocessedImagePath:String	InvalidInputException

6.4 Semantics

6.4.1 State Variables

None.

6.4.2 Environment Variables

None.

6.4.3 Assumptions

None.

6.4.4 Access Routine Semantics

filterInput():

- transition: None.
- output: Returns preprocessed image data suitable for both model training and caption generation.
- exception: Raises `InvalidInputException` if the input image format or type is unsupported.

6.4.5 Local Functions

None.

7 MIS of WCAGCompliance Module

7.1 Module

`WCAGComp`

7.2 Uses

Takes generated captions from the `CaptionGeneration` module and validates them against WCAG 2.1 AA criteria (W3C (2018)).

7.3 Syntax

7.3.1 Exported Constants

None.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
checkCompliance	CaptionText:String	ComplianceReport:List	ValidationException

7.4 Semantics

7.4.1 State Variables

None.

7.4.2 Environment Variables

None.

7.4.3 Assumptions

- Assumes the input caption text is available and properly formatted for validation.

7.4.4 Access Routine Semantics

checkCompliance():

- transition: None.
- output: Returns a `ComplianceReport` summarizing accessibility validation results.
- exception: Raises `ValidationException` if validation cannot be completed or the input is invalid.

7.4.5 Local Functions

None.

8 MIS of ModelOutput Module

8.1 Module

`ModOut`

8.2 Uses

Receives validated captions from the `WCAGCompliance` module or directly from the `CaptionGeneration` module when compliance checks are bypassed. Sends the final caption output to the user-facing interface or backend display component.

8.3 Syntax

8.3.1 Exported Constants

None.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
generateOutput	FinalCaption:String	DisplayConfirmation:$\mathbb{B}$	OutputException

8.4 Semantics

8.4.1 State Variables

None.

8.4.2 Environment Variables

None.

8.4.3 Assumptions

- Assumes the output channel (UI or backend) is active and ready to receive data.

8.4.4 Access Routine Semantics

generateOutput(FinalCaption):

- transition: Sends the final caption text to the appropriate user-facing component for display.
- output: Returns `DisplayConfirmation` upon successful output delivery.
- exception: Raises `OutputException` if communication with the display component fails.

8.4.5 Local Functions

None.

9 MIS of AIModelTraining Module

9.1 Module

`AITrain`

9.2 Uses

Takes preprocessed data from the `DataPreprocess` module and trains a model that will later be used by the `CaptionGeneration` module for alt-text generation.

9.3 Syntax

9.3.1 Exported Constants

None.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
initializeModel	configPath: String	-	ConfigFileNot-FoundException
trainModel	trainingData: Dataset	trainedModel: Model	TrainingException
saveCheckpoint	progress: Integer	-	WriteErrorException

9.4 Semantics

9.4.1 State Variables

`trainedModel`: Stores the trained model instance.

9.4.2 Environment Variables

`GPU/CPU` — used for training operations.
`Filesystem` — stores intermediate checkpoints and model outputs.

9.4.3 Assumptions

- The input data is valid and preprocessed.
- The system has sufficient computational resources.
- The configuration file exists and can be accessed.

9.4.4 Access Routine Semantics

initializeModel(configPath):

- transition: Loads or initializes the model structure and related parameters.
- output: None.
- exception: Raises `ConfigFileNotFoundException` if the configuration is missing or invalid.

trainModel(trainingData):

- transition: Trains the model on provided data and updates `trainedModel`.
- output: Returns the trained model.
- exception: Raises `TrainingException` if the process fails.

saveCheckpoint(progress):

- transition: Saves the model's current state for future use or recovery.
- output: None.
- exception: Raises `WriteErrorException` if the checkpoint cannot be saved.

9.4.5 Local Functions

- **splitData(NewFilePath:String) `->` TestPath:String, TrainPath:String, TrainPath:String**: Divides input data into training and validation portions.

10 MIS of CaptionGeneration Module

10.1 Module

`CapGen`

10.2 Uses

Uses the trained model from the `AIModelTraining` module to generate descriptive alt-text for input images.

10.3 Syntax

10.3.1 Exported Constants

None.

10.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
loadModel	modelPath: String	-	ModelLoadException
generateCaption	image: Image	caption: String	GenerationException

10.4 Semantics

10.4.1 State Variables

`loadedModel`: Stores the active trained model used for caption generation.

10.4.2 Environment Variables

`GPU/CPU` — executes model inference.

10.4.3 Assumptions

- A valid trained model is available and compatible with the input format.
- Input images are properly preprocessed.

10.4.4 Access Routine Semantics

loadModel(modelPath):

- transition: Loads the trained model into memory.
- output: None.
- exception: Raises `ModelLoadException` if the model file is missing or invalid.

generateCaption(image):

- transition: Uses the loaded model to produce a descriptive caption.
- output: Returns generated alt-text for the input image.
- exception: Raises `GenerationException` if caption generation fails.

10.4.5 Local Functions

None.

11 MIS of FeedbackMetrics Module

11.1 Module

`ModMetric`

11.2 Uses

Receives generated captions from the `WCAGCompliance` module, evaluates their overall quality and accessibility, and sends the resulting performance scores to the `FeedbackIntegration` module for further model improvement.

11.3 Syntax

11.3.1 Exported Constants

None.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
evaluateCaption	CaptionText:String	PerformanceScores:List	EvaluationException

11.4 Semantics

11.4.1 State Variables

None.

11.4.2 Environment Variables

None.

11.4.3 Assumptions

- Assumes generated captions are available and that predefined evaluation metrics are accessible.

11.4.4 Access Routine Semantics

evaluateCaption(CaptionText):

- transition: None.
- output: Returns performance scores summarizing caption quality and accessibility (see Appendix for Jing's evaluation metrics).
- exception: Raises `EvaluationException` if evaluation cannot be completed.

11.4.5 Local Functions

None.

12 MIS of FeedbackIntegration Module

12.1 Module

`ModInt`

12.2 Uses

Consumes performance scores from the `FeedbackMetrics` module and applies them to update model parameters in the `AIModelTraining` module, closing the feedback loop after inference.

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
updateModel	PerformanceScores:List	UpdateAck:$\mathbb{B}$	UpdateException

12.4 Semantics

12.4.1 State Variables

`modelState`: current model parameters selected for (re)training.

12.4.2 Environment Variables

None.

12.4.3 Assumptions

- Assumes valid performance scores are provided by the `FeedbackMetrics` module and that retraining can be invoked in `AIModelTraining`.

12.4.4 Access Routine Semantics

updateModel(PerformanceScores):

- transition: updates `modelState` and signals `AIModelTraining` to (re)train with revised parameters.
- output: returns `UpdateAck` confirming that the update request was accepted.
- exception: raises `UpdateException` if parameter update or retraining trigger fails.

12.4.5 Local Functions

None.

13 MIS of BackendController Module

13.1 Module

`BackendController`

13.2 Uses

Serves as the coordinator between the frontend, backend and machine learning modules.

- Receives authentication and session status from the AuthenticationService Module (see Section 14) and SessionManagement module (see Section 15), respectively.
- Sends uploaded images to the ImageValidation module (see Section 16) and receives validation results.
- Sends validated inputs to the DataPreprocess module (see Section 6)
- Sends errors and log events to the Logging module (see Section 17)
- Returns the generated alt text or error message to the MainScreen Module (see Section 19)

13.3 Syntax

13.3.1 Exported Constants

`SESSION_TIMEOUT`: $\mathbb{N}$ - Specifies the amount of time in minutes a users logged-in session can remain active for before requiring reauthentication.

`API_RESPONSE_TIMEOUT`: $\mathbb{N}$ - Specifies the amount of time in seconds that the controller waits for a response from other modules before timing out.

13.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
validateUser	userToken:String	userStatus:$\mathbb{B}$	InvalidTokenException
validateImage	imagePath:String, userToken:	imagePath:$\mathbb{B}$	InvalidImageException
sendToModel	imagePath:String, userToken: String	-	AltTextGenerationError
getAltText	imagePath:String, userToken: String	altText:String	AltTextNotFoundException
returnToFrontend	altText or errorMsg: String, userID: String	-	UIUnreachableException
updateAltText	altText:String, userID:String	-	SessionStateSaveException

13.4 Semantics

13.4.1 State Variables

None.

13.4.2 Environment Variables

`Filesystem` - Used to retrieve images using the file path provided by users.

13.4.3 Assumptions

- MainScreen, AuthenticationService, ImageValidation and DataPreprocess modules are correctly operating and reachable.
- Network connection is available for API communication.

13.4.4 Access Routine Semantics

validateUser(userToken):

- transition: Sends the `userToken` to the User Authentication Module for Verification.
- output: Returns `True` if the user is valid, otherwise `False` is returned.
- exception: `InvalidTokenException` is raised if token is expired or cannot be validated.

validateImage(image,userToken):

- transition: Sends the uploaded image and associated user token to the Image Validation Module for verification of file type and size.
- output: Returns `True` if the image was successfully validated, otherwise `False` is returned.
- exception: `InvalidImageException` is raised if the image fails the validation process and does not meet system requirements.

sendToModel(image,userToken):

- transition: Sends the validated image and associated user token to the Machine Learning Module to initiate alt-text generation.
- output: None.
- exception: `AltTextGenerationError` is raised if the Machine Learning Module fails to process image or generate alt text.

returnToFrontend(alt Text or errorMsg, userToken):

- transition: Sends the generated alt text or corresponding error message to the Frontend Module.
- output: None.

- exception: `FrontendUnreachableException` is raised if the Frontend Module cannot be reached.

updateAltText(altText, userID):

- transition: Updates the user's existing alt-text generation to include their changes.
- output: None
- exception: `SessionStorageException` is raised if the system is unable to save the modified alt text to the users session history.

14 MIS of AuthenticationService Module

14.1 Module

`AuthenticationService`

14.2 Uses

This module is responsible for managing and validating user authentication across the Reading4All system. It ensures that only verified users can upload, edit, or retrieve images and generated alt text.

- LoginScreen Module: Receives user login and logout requests, sends authentication tokens.
- Session Management Module: Manages active sessions and stores validated tokens.
- Backend Controller Module: Requests user validation before allowing access to backend operations.

14.3 Syntax

14.3.1 Exported Constants

`TOKEN_EXPIRY_TIME`: $\mathbb{N}$ - Duration in minutes after which an authentication token expires.

`MAX_LOGIN_ATTEMPTS`: $\mathbb{N}$ - Maximum number of failed login attempts allowed before temporary account lockout.

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
validateToken	userToken:String	Success:$\mathbb{B}$	InvalidTokenException
signInUser	credentials:String	userToken:String	AuthenticationFailed-Exception
signOutUser	userToken:String	Success:$\mathbb{B}$	SessionNotFoundException
getCurrentUser	userToken:String	userID: String	UserNotAuthenticated-Exception
refreshToken	userToken:String	newToken:String	TokenRefreshException

14.4 Semantics

14.4.1 State Variables

- `currentUser`: Stores information of the currently authenticated user (userID, role, status).
- `activeTokens`: List of active tokens issued during authenticated sessions.

14.4.2 Environment Variables

- LoginAPI to authentication the information of users attempting to sign in.

14.4.3 Assumptions

- The database or session store used for token verification is operational.
- Network connection to backend and session services is stable.
- All tokens follow the system-defined structure and encryption standard.

14.4.4 Access Routine Semantics

validateToken(userToken):

- transition: Compares provided token against valid session records in `SessionManagement`.
- output: Returns `True` if token is valid and active, otherwise `False`.
- exception: `InvalidTokenException` is raised if the token has expired or does not exist.

signInUser(credentials):

- transition: Verifies credentials (e.g., username and password) and issues a unique token for session creation.
- output: Returns a valid `userToken` on successful authentication.

- exception: `AuthenticationFailedException` is raised if credentials are invalid or user does not exist.

signOutUser(userToken):

- transition: Removes the associated token from the active token list and ends the user's session.
- output: Returns `True` on successful logout.
- exception: `SessionNotFoundException` is raised if the token is invalid or session not found.

getCurrentUser(userToken):

- transition: Queries active sessions to identify the user associated with the token.
- output: Returns `userID` if token corresponds to an active session.
- exception: `UserNotAuthenticatedException` is raised if no active session exists for the token.

refreshToken(userToken):

- transition: Reissues a new token with an updated expiry time after verifying the old token's validity.
- output: Returns `newToken`.
- exception: `TokenRefreshException` is raised if old token is expired or tampered.

14.4.5 Local Functions

- `generateToken(userID)` — Creates a secure, encrypted token for the authenticated user.
- `encryptData(data)` — Encrypts sensitive authentication information before storage.
- `validateCredentials(credentials)` — Confirms user credentials against secure storage.

15 MIS of SessionManagement Module

15.1 Module

`SessionManagement`

15.2 Uses

The Session Management module is responsible for creating, storing, and maintaining active user sessions within the Reading4All system. It also tracks the session-specific history of user interactions, referencing corresponding entries in the Reading4All data. Other modules, such as the Authentication Service and Backend Controller, rely on it for validating active tokens and retrieving user activity logs within the current session.

- **Authentication Service:** Initiates creation, validation, and deletion of sessions.
- **Backend Controller:** Updates session history after each major user action (e.g., validateImage, sendToModel, getAltText).
- **Reading4All Data:** Serves as the persistent storage layer for user images and generated alt-text records, referenced by session data IDs.

15.3 Syntax

15.3.1 Exported Constants

`MAX_SESSION_DURATION`: $\mathbb{N}$ - Specifies the maximum duration, in minutes, that a session remains active before expiration.

`SESSION_REFRESH_INTERVAL`: $\mathbb{N}$ - The interval, in minutes, at which session validity is checked or refreshed.

`MAX_SESSION_HISTORY`: $\mathbb{N}$ - Maximum number of interaction entries stored per session to prevent unbounded growth.

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
createSession	userID:String	sessionToken:String	SessionCreation-Exception
storeSession	userID:String, sessionToken:String	success:$\mathbb{B}$	SessionStorageException
validateSession	userToken:String	success:$\mathbb{B}$	InvalidSessionException
deleteSession	userToken:String	success:$\mathbb{B}$	SessionNotFound-Exception
getSession	userToken:String	sessionData:String	SessionNotFound-Exception
appendInteraction	userToken:String, interactionEntry:String	success:$\mathbb{B}$	SessionNotFound-Exception
getSessionHistory	userToken:String, filter:String	interactionList:List	SessionNotFound-Exception
linkDataRecord	userToken:String, dataRecordID:String	success:$\mathbb{B}$	SessionNotFound-Exception
getRecentRecords	userToken:String, limit: $\mathbb{N}$	dataRecordRefs:List	SessionNotFound-Exception

15.4 Semantics

15.4.1 State Variables

- `activeSessions`: Mapping of `userID` $\rightarrow$ `sessionToken` representing currently active sessions.
- `sessionExpiry`: Mapping of `sessionToken` $\rightarrow$ expiry time.
- `sessionHistory`: Mapping of `sessionToken` $\rightarrow$ ordered list of interaction entries for that session.
- `sessionDataRefs`: Mapping of `sessionToken` $\rightarrow$ set of dataRecordIDs referencing Reading4All data entries.

15.4.2 Environment Variables

- `SystemClock`: Provides current time for session expiry calculations.
- `StorageService`: Fast-access storage for session and interaction data.
- `Browser`: Storing session tokens for the valid duration of user activity.

15.4.3 Assumptions

- The system clock is synchronized to ensure accurate session expiry times.
- Tokens and user IDs are unique and securely generated.
- A fast-access storage service (e.g., in-memory cache or Redis) is available for storing session and interaction data.
- Reading4All data provides stable and retrievable `dataRecordID`s for cross-module referencing.

15.4.4 Access Routine Semantics

createSession(userID):

- transition: Generates a new session token, initializes empty history and data references, and sets the expiry time.
- output: Returns the created `sessionToken`.
- exception: Raises `SessionCreationException` if session creation fails.

validateSession(userToken):

- transition: None.
- output: Returns `True` if the session exists and is active, otherwise `False`.
- exception: Raises `InvalidSessionException` if the token is expired or invalid.

appendInteraction(userToken, interactionEntry):

- transition: Appends the provided `interactionEntry` (with timestamp and type) to `sessionHistory[userToken]`. If the maximum history size is exceeded, the oldest entry is removed.
- output: Returns `True` upon success.
- exception: Raises `SessionNotFoundException` if the token is not associated with an active session.

linkDataRecord(userToken, dataRecordID):

- transition: Associates the provided `dataRecordID` (from Reading4All data) with the user's current session for future retrieval.
- output: Returns `True` upon success.
- exception: Raises `SessionNotFoundException` if no active session exists.

getSessionHistory(userToken, filter):

- transition: None.
- output: Returns a filtered list of interaction entries (e.g., uploads, alt-text retrieval) for the specified session.
- exception: Raises `SessionNotFoundException` if the token is invalid or no session history exists.

getRecentRecords(userToken, limit):

- transition: None.
- output: Returns up to the specified number of recent `dataRecordID` references associated with the current session.
- exception: Raises `SessionNotFoundException` if no active session exists.

deleteSession(userToken):

- transition: Removes the token from active sessions and clears its associated history and data references.
- output: Returns `True` if deletion is successful.
- exception: Raises `SessionNotFoundException` if the token is invalid or not found.

15.4.5 Local Functions

- **generateSessionToken(userID)**: Creates a secure, unique token for the given user.
- **updateExpiry(sessionToken)**: Extends the expiry time of a session on continued activity.
- **cleanExpiredSessions()**: Removes all expired sessions and their histories from storage.
- **applyHistoryFilter(history, filter)**: Applies filtering parameters (e.g., type, time range, status) to session history queries.

16 MIS of ImageValidation Module

16.1 Module

`ImageValidation`

16.2 Uses

None.

16.3 Syntax

16.3.1 Exported Constants

`MAX_IMAGE_SIZE`: $\mathbb{N}$ - Specifies the maximum file size in (MB) allowed for uploaded images.
`MIN_IMAGE_SIZE`: $\mathbb{N}$ - Specifies the minimum file size in (MB) allowed for uploaded images.
`ACCEPTED_IMAGE_TYPES`: `List of Strings`- Specifies the accepted image file types.

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
validateImageFile	imagePath:String, userToken:String	imageStatus: $\mathbb{B}$	InvalidImageTypeException

16.4 Semantics

16.4.1 State Variables

None.

16.4.2 Environment Variables

`Filesystem` - Used to retrieve images using the file path provided by users.

16.4.3 Assumptions

- The `BackendController` module provides the ImageValidation module with an image file that is reachable for validation.
- The user has successfully been validated through the user authentication process.

16.4.4 Access Routine Semantics

validateImageFile(imagePath, userToken)

- transition: Verifies that the inputted image meets size and format requirements.
- output: Returns `True` if the image meets size and format requirements, otherwise `False` is returned.
- exception: `InvalidImageException` is raised if the image cannot be reached and therefore cannot be validated.

16.4.5 Local Functions

- **verifyFileType(imagePath)**: Checks that the images file type matches a type found in `ACCEPTED_IMAGE_TYPES`.
- **verifyFileSize(imagePath)**: Checks that the images size falls between `MIN_IMAGE_SIZE` and `MAX_IMAGE_SIZE`.
- **verifyFileReachable(imagePath)**: Checks that the image can be accessed and opened successfully.

17 MIS of Logger Module

17.1 Module

`Logger`

17.2 Uses

The module handles the recording of key system events and any errors encountered. It logs actions such as image uploads, alt-text generation and user validations, as well as system issues like timeouts. This logging will enables developers to debug for effectively and ensure system reliability and security during failures. The logging module does not use any other modules.

17.3 Syntax

17.3.1 Exported Constants

`LOG_FILE_PATH`: String - Specifies the location of where the logs should be stored.
`LOG_TYPE`: List of Strings - Specifies the possible log types such as an `Event` and `Error`.

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
logEvent	eventMssg:String, logType:String	-	LogWriteException

17.4 Semantics

17.4.1 State Variables

None.

17.4.2 Environment Variables

`Filesystem` - Used to save the log files to the specified path.

17.4.3 Assumptions

- The file system is available and logs can be saved to it.
- The LOG_FILE_PATH location can be accessed.
- Log messages are provided in string format by the Backend Controller Module.

17.4.4 Access Routine Semantics

logEvent(eventMssg, logType):

- transition: Creates an log entry containing the time of event occur, type and message.
- output: An updated log file with new event or error entry added.
- exception: `LogWriteException` is raised if the log file cannot be accessed or written to.

17.4.5 Local Functions

None.

18 MIS of UserInterfaceInteractions Module

18.1 Module

`InteractionsUI`

18.2 Uses

This module uses `BackendController` to authenticate the users that are signing in, and to obtain the generated alternative text. It also uses the display modules (`MainScreen`, `LogInScreen`, `ShowHistoryScreen`) to render the different screens and interfaces to the users.

18.3 Syntax

18.3.1 Exported Constants

```
SUPPORTED_IMG_TYPES = {"PNG", "JPEG"}
OUTPUT_FILE_TYPE = {".txt"}
MAX_IMG_SIZE = 10MB
```

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
uploadImage	imageFilePath:String	Success:$\mathbb{B}$	FileNotFound, FileTypeNot-Supported, FileTooLarge
downloadText	altText:String	altTextFile:String	NetworkError
copyText	altText:String	Success:$\mathbb{B}$	ClipboardError
editText	altText:String, edited-Text:String	Success:$\mathbb{B}$	NetworkError
signIn	token:String	Success:$\mathbb{B}$	AuthError
signOut	-	Success:$\mathbb{B}$	AuthError
showHistory	-	Success:$\mathbb{B}$	NetworkError

18.4 Semantics

18.4.1 State Variables

None.

18.4.2 Environment Variables

- `Clipboard` - system's clipboard interface for copy function.
- `HTTPClient` - used to call backend services.
- `Keyboard` - receives key presses for keyboard navigation.
- `FileSystem` - used to access file paths.

18.4.3 Assumptions

- `Clipboard` access is enabled by the user for `copyText`.
- User allows system to access files for uploading images and downloading alt text.
- `uploadImage` is called before any other access program.

18.4.4 Access Routine Semantics

uploadImage(imageFilePath):

- transition: `HTTPClient` sends the image bytes into the backend service which stores the image for that session, and communicates with the AI model for alt text generation.

- output: returns `true` iff the backend server acknowledges and is able to process the image, returns `false` otherwise.
- exception: `FileNotFound`, `FileTypeNotSupported` (not in `SUPPORT_IMG_TYPES`), and `FileTooLarge` (not within `MAX_IMG_SIZE`)

downloadText(altText):

- transition: None.
- output: a `.txt` payload that contains the generated alt text.
- exception: `NetworkError`.

copyText(altText):

- transition: `Clipboard` := altText
- output: returns `true` iff clipboard write succeeds, returns `false` otherwise.
- exception: `ClipboardError`.

editText(altText,editedText):

- transition: `HTTPClient` sends update to backend and stored altText for uploaded image is replaced with editedText.
- output: return `true` iff server acknowledges the update, returns `false` otherwise.
- exception: `NetworkError`.

signIn(token):

- transition: if `token` is valid according to BackendController then user session becomes active. Otherwise no state change.
- output: return `true` iff user session is active, returns `false` otherwise.
- exception: `AuthenticationError`.

signOut():

- transition: terminates current active user session if any exists. Otherwise no state change.
- output: return `true` iff user session is was terminated, returns `false` otherwise.
- exception: `AuthenticationError`.

showHistory():

- transition: requests the ShowHistoryScreen module to render the History Screen using the current stored alt text session history.
- output: return `true` iff history was successfully shown, returns `false` otherwise.
- exception: `AuthenticationError`.

18.4.5 Local Functions

- **isSupportedImage(filePath:String)** `->` $\mathbb{B}$: returns `true` if the file type $\in$ `SUPPORTED_IMG_TYPES`.
- **checkImgSize(filePath:String)** `->` $\mathbb{B}$: returns `true` if the size of the image $\leq$ `MAX_IMG_SIZE`.
- **saveAsTxt(altText: String)** `->` **filePath:String**: wraps generated alt text into a `.txt` file for `downloadText()`.
- **buildUploadPayload(filePath: String)** `->` **Binary**: reads the file and constructs the binary payload for the `uploadImage` function to send to the BackendController module.

19 MIS of MainScreen Module

19.1 Module

`MainScreen`

19.2 Uses

This module uses `AccessibilityUI` to ensure that the content within the interface is compatible with screen readers and tabbable by a keyboard.

19.3 Syntax

19.3.1 Exported Constants

None.

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderMain	-	-	RenderError, AccessibilityError

19.4 Semantics

19.4.1 State Variables

None.

19.4.2 Environment Variables

`DOM` - active document element into which UI is rendered.

19.4.3 Assumptions

- Accessibility services are provided by the `AccessibilityUI` module for screen reader announcements, landmark management, initial focus, and keyboard navigation helpers. This module will provide notifications to the `AccessibilityUI` module.

19.4.4 Access Routine Semantics

renderMain():

- transition: update DOM to reflect main screen containing the upload page, then call on `AccessibilityUI` to handle compatibility with screen readers and keyboard navigation.
- output: None.
- exception: `RenderError`, `AccessibilityError`.

19.4.5 Local Functions

- **ensureMainLandmark()** `->` $\mathbb{B}$: calls `AccessibilityUI` to mark/set the main landmark.
- **setInitialFocus()**`->` $\mathbb{B}$: calls `AccessibilityUI` to move focus to the main region.
- **announceMainLoaded()** `->` $\mathbb{B}$: calls `AccessibilityUI` to announce screen load.

20 MIS of ShowHistoryScreen Module

20.1 Module

`ShowHistoryScreen`

20.2 Uses

This module uses `AccessibilityUI` to ensure that the content within the interface is compatible with screen readers and tabbable by a keyboard.

20.3 Syntax

20.3.1 Exported Constants

None.

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
showHistory	-	-	RenderError, AccessibilityError

20.4 Semantics

20.4.1 State Variables

None.

20.4.2 Environment Variables

`DOM` - active document element into which UI is rendered.

20.4.3 Assumptions

- If history is shown, a valid and authenticated session already exists.
- `AccessibilityUI` is used for announcements and focus.
- History content to display is already prepared by the `InteractionsUI` module through the `BackendController` module.

20.4.4 Access Routine Semantics

showHistory():

- transition: update DOM to reflect user's current session history, then call on `AccessibilityUI` to handle compatibility with screen readers and keyboard navigation.
- output: None.
- exception: `RenderError`, `AccessibilityError`.

20.4.5 Local Functions

- **ensureMainLandmark()** `->` $\mathbb{B}$: calls `AccessibilityUI` to mark/set the main landmark.
- **setInitialFocus()**`->` $\mathbb{B}$: calls `AccessibilityUI` to move focus to the main region.
- **announceHistoryLoaded()** `->` $\mathbb{B}$: calls `AccessibilityUI` to announce history screen load.

21 MIS of LogInScreen Module

21.1 Module

`LogInScreen`

21.2 Uses

This module uses `AccessibilityUI` to ensure that the content within the interface is compatible with screen readers and tabbable by a keyboard.

21.3 Syntax

21.3.1 Exported Constants

None.

21.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderLogIn	-	-	RenderError, AccessibilityError

21.4 Semantics

21.4.1 State Variables

None.

21.4.2 Environment Variables

`DOM` - active document element into which UI is rendered.

21.4.3 Assumptions

- `AccessibilityUI` is used for announcements and focus.

21.4.4 Access Routine Semantics

renderLogIn():

- transition: update `DOM` to reflect log in screen with username and password fields, then call on `AccessibilityUI` to handle compatibility with screen readers and keyboard navigation.
- output: None.
- exception: `RenderError`, `AuthenticationError`.

21.4.5 Local Functions

- **ensureMainLandmark()** `->` $\mathbb{B}$: calls `AccessibilityUI` to mark/set the main landmark.
- **setInitialFocus(username:String)** `->` $\mathbb{B}$: calls `accessibilityUI` to move keyboard focus to the username input field after the log in screen is rendered.
- **announceLogInLoaded()** `->` $\mathbb{B}$: calls `AccessibilityUI` to announce log in screen load.

22 MIS of UserInterfaceAccessibility Module

22.1 Module

`AccessibilityUI`

22.2 Uses

This module uses the display modules (`MainScreen`, `LogInScreen`, `showHistorScreeny`) to receive notifications of user activity to announce changes through a screen reader and apply keyboard navigation to the content in those screens.

22.3 Syntax

22.3.1 Exported Constants

22.3.2 Exported Access Programs

Name	In	Out	Exceptions
ensureMainLandmark	rootSel:String	Success:$\mathbb{B}$	AccessibilityError
setInitialFocus	targetSel:String	Success:$\mathbb{B}$	AccessibilityError
announce	msg:String	Success:$\mathbb{B}$	AccessibilityError
applyKeyboardNav	scopeSel:String	Success:$\mathbb{B}$	AccessibilityError

22.4 Semantics

22.4.1 State Variables

None.

22.4.2 Environment Variables

`DOM` - the active HTML document containing interactive elements
`ScreenReaderAPI` - hidden live region nodes used to deliver announcements to screen readers

for compatibility
`Keyboard` - delivers keypress events to use for keyboard navigation

22.4.3 Assumptions

- The page allows insertion of valid ARIA (W3C (2024)) live region nodes
- `MainScreen`, `LogInScreen`, `ShowHistoryScreen` modules calls `AccessibilityUI` to announce changes on the user interface.

22.4.4 Access Routine Semantics

ensureMainLandmark(rootSel:String):

- transition: ensure that the element selected by `rootSel` is the page's main landmark (i.e. `<main>`).
- output: returns `true` if main landmark was present after the call, returns `false` otherwise.
- exception: `AccessibilityError`.

setInitialFocus(targetSel:String):

- transition: move keyboard focus to the element matching `targetSel` for keyboard navigation; if not focusable, make it focusable for the interaction.
- output: returns `true` if focus was moved, returns `false` otherwise.
- exception: `AccessibilityError`.

announce(msg:String):

- transition: update the ARIA (W3C (2024)) live region with `msg` so screen readers can announce the change.
- output: returns `true` if live region was updated, returns `false` otherwise.
- exception: `AccessibilityError`.

applyKeyboardNav(scopeSel:String):

- transition: attach handlers within `scopeSel` (container that gets arrow-key navigation behaviour) to enable roving-tabindex and arrow-key navigation (i.e., ArrowLeft/Right/Up/Down, Home/End, Enter/Space activation) without changing application state.
- output: returns `true` if handlers are attached successfully, returns `false` otherwise.
- exception: `AccessibilityError`.

22.4.5 Local Functions

- **getLiveRegion(type:String)** `->` **Node**: returns the hidden live region for announcements; calls on `createLiveRegion(type:  String)` if no live region to create one.
- **createLiveRegion(type:String)** `->` **Node**: creates and returns the hidden live region for announcements.
- **isFocusable(node:Node)** `->` $\mathbb{B}$: returns `true` if `node` is focusable.
- **applyRovingTabindex(container:Node)** `->` $\mathbb{B}$: ensures exactly one child is tabbable and others are reachable via arrow keys.
- **moveFocus(container:Node, dir:String)** `->` $\mathbb{B}$: shifts focus to the next or previous roving item given `dir` $\in$ `"next","prev","home","end"` .

References

Software requirements specification for bridging gaps: Ai for diagram accessibility, 2025. URL https://github.com/4G06-CAPSTONE-2025/Reading4All/blob/main/docs/SRS-Volere/SRS.pdf.

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.

W3C. Web content accessibility guidelines (wcag) 2.1. https://www.w3.org/TR/WCAG21/, 2018. Accessed: 2025-11-10.

W3C. Wai-aria authoring practices 1.2. https://www.w3.org/TR/html-aria/, 2024. Accessed: 2025-11-10.

23 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

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

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?
2. What pain points did you experience during this deliverable, and how did you resolve them?
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?
4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?
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)
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)