Module Interface Specification for Bridging Gaps: AI for Diagram Accessibility

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

January 28, 2026

1 Revision History

Date	Version	Notes
November 13, 2025	1.0	Rev-1 Design Document
January 11, 2025	1.1	Peer Review `#239` Applied
January 21 2026	2.0	Rev-0 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 FeedbackIntegration Module FeedbackLoop 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

SUPPORTED_IMAGE_TYPES: List[String] – Accepted input formats (e.g., {"PNG","JPEG"}).
TARGET_WIDTH: $\mathbb{N}$ – Desired output width in pixels (may be hardcoded in early versions).
TARGET_HEIGHT: $\mathbb{N}$ – Desired output height in pixels (may be hardcoded in early versions).
NORMALIZE_MODE: String – Normalization scheme (e.g., {"none","imagenet"}); may be hardcoded.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
initialize	configPath:String	-	ConfigFileNotFoundException
filterInput	imageFilePath:String	PreprocessedImagePath:String	InvalidInputException

6.4 Semantics

6.4.1 State Variables

lastInputPath: String – (Optional) caches most recently processed input path for debugging/traceability.
lastOutputPath: String – (Optional) caches most recently produced output path for debugging/traceability.

6.4.2 Environment Variables

InputImage: Image – Image loaded from imageFilePath via the filesystem.
OutputImage: Image – Processed image produced by this module and made available to downstream modules in the system.
Filesystem – Used to read input images and write preprocessed outputs.
ImageCodec – System/library support for decoding and encoding image formats.

6.4.3 Assumptions

- `initialize` is called before `filterInput`.
- Constants may be hardcoded in the current implementation and moved to configuration later.
- State variables are optional; omitting them does not affect functional correctness.

6.4.4 Access Routine Semantics

initialize(configPath):

- transition:
  - Load preprocessing configuration parameters (image size, normalization mode)
  - Initialize filesystem and image codec dependencies
- output: None.
- exception: Raises `ConfigFileNotFoundException` if configuration cannot be loaded.

filterInput(imageFilePath):

- transition:
  - (Optional) `lastInputPath` := imageFilePath
  - Read `InputImage` from `Filesystem` using `ImageCodec`
  - Verify that `InputImage` is readable and its format is in `SUPPORTED_IMAGE_TYPES`
  - Transform `InputImage` by:
    - resizing it to (`TARGET_WIDTH`, `TARGET_HEIGHT`)
    - applying pixel-value normalization according to `NORMALIZE_MODE`
  - Assign the transformed image to `OutputImage`
  - Make `OutputImage` available to downstream modules
  - (Optional) update internal state to reflect successful preprocessing
- output: Returns `OutputImage`.
- exception: Raises `InvalidInputException` if the input image is unsupported, unreadable, or corrupt.

6.4.5 Local Functions

None.

7 MIS of WCAGCompliance Module

7.1 Module

`WCAGComp`

7.2 Uses

None.

7.3 Syntax

7.3.1 Exported Constants

`WCAG_LEVEL`: String – Target conformance level (e.g., {"AA"}); may be hardcoded.
`MIN_CAPTION_LEN`: $\mathbb{N}$ – Minimum caption length (characters), may be hardcoded.
`MAX_CAPTION_LEN`: $\mathbb{N}$ – Maximum caption length (characters), may be hardcoded.

7.3.2 Exported Access Programs

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

7.4 Semantics

7.4.1 State Variables

`lastCheckedCaption`: String – (Optional) caches the most recently validated caption for debugging/traceability.
`lastReport`: List – (Optional) caches the most recently produced compliance report.

7.4.2 Environment Variables

None.

7.4.3 Assumptions

- `initialize` is called before `checkCompliance`.
- Constants may be hardcoded in the current implementation and moved to configuration later.
- State variables are optional; omitting them does not affect functional correctness.
- Assumes the input caption text is available and properly formatted for validation.

7.4.4 Access Routine Semantics

initialize():

- transition:
  - Initialize WCAG rule set and validation thresholds
- output: None.

checkCompliance(CaptionText):

- transition:
  - (Optional) `lastCheckedCaption` := CaptionText
  - Validate basic WCAG-aligned constraints (e.g. length bounds using `MIN_CAPTION_LEN` and `MAX_CAPTION_LEN`)
  - Construct a `ComplianceReport` summarizing pass/fail, warnings, and violations
  - (Optional) `lastReport` := ComplianceReport
- 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

Called by the `WCAGCompliance` module after a caption has been validated, or by the `CaptionGeneration` module when compliance checks are explicitly 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
initialize	-	-	OutputException
generateOutput	FinalCaption:String	DisplayConfirmation:$\mathbb{B}$	OutputException

8.4 Semantics

8.4.1 State Variables

None.

8.4.2 Environment Variables

`DisplayChannel` – User-facing interface or backend component responsible for presenting captions.

8.4.3 Assumptions

- `initialize` is called before `generateOutput`.
- Assumes `FinalCaption` has already been designated as compliant or approved by the calling module.
- Assumes the display channel is active and able to receive caption text.

8.4.4 Access Routine Semantics

initialize():

- transition:
  - Establish connection with display or backend output channel
- output: None.

generateOutput(FinalCaption):

- transition:
  - Receive `FinalCaption` from the calling module
  - Transmit `FinalCaption` to `DisplayChannel` for presentation
- output: Returns `DisplayConfirmation` indicating successful delivery.
- exception: Raises `OutputException` if transmission to the display channel 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

`DEFAULT_MODEL_ID`: String – Default model identifier/architecture used for training (may be hardcoded).
`MAX_EPOCHS`: $\mathbb{N}$ – Maximum number of training epochs (may be hardcoded).
`BATCH_SIZE`: $\mathbb{N}$ – Training batch size (may be hardcoded).
`LEARNING_RATE`: $\mathbb{R}$– Default learning rate (may be hardcoded).
`CHECKPOINT_DIR`: String – Directory where checkpoints are written (may be hardcoded).
`CHECKPOINT_FREQ`: $\mathbb{N}$ – Save checkpoint every k steps/epochs (may be hardcoded).
`RANDOM_SEED`: $\mathbb{N}$ – Seed for reproducibility (optional; may be omitted in early versions).

9.3.2 Exported Access Programs

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

9.4 Semantics

9.4.1 State Variables

`trainedModel`: Stores the trained model instance.
`checkpointDir`: String. This stores the checkpoint directory path.
`modelSavePath`: String. This stores the file path used to save the final trained model.

9.4.2 Environment Variables

`cpu`: Device. This is the CPU execution device.
`gpu`: Device. This is the GPU execution device (it may be unavailable).
`checkpointDir`: String. This is the writable checkpoint directory path.
`modelSavePath`: String. This is the writable file path for the final trained model.

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 (may be a default/hardcoded configuration in early versions).
- Constants may be hardcoded in the current implementation and moved to configuration later.
- `initializeModel` is called before `trainModel` and `saveCheckpoint`.

9.4.4 Access Routine Semantics

initializeModel(configPath):

- transition: Loads or initializes the model structure and related parameters based on `configPath` (or defaults).
- output: None.
- exception: Raises `ConfigFileNotFoundException` if the configuration is missing, unreadable, or invalid.

trainModel(trainingData):

- transition: Trains the model on provided data and updates `trainedModel`.
- output: Returns the trained model.
- exception: Raises `TrainingException` if training fails due to invalid data, insufficient resources, or runtime errors.

saveCheckpoint(progress):

- transition: Saves the model's current state to `CHECKPOINT_DIR` for recovery and future use.
- output: None.

- exception: Raises `WriteErrorException` if the checkpoint cannot be written (e.g., permission issues, missing directory, or I/O failure).

saveTrainedModel():

- transition: Serializes `trainedModel` to `modelSavePath`.
- output: None.
- exception: Raises `WriteErrorException` if the model cannot be written to `modelSavePath`.

9.4.5 Local Functions

- **splitData(NewFilePath:String) -> TestPath:String, TrainPath:String, ValPath:String**: Divides input data into training, validation, and testing 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

`MAX_CAPTION_LEN`: $\mathbb{N}$ – Maximum generated caption length (tokens/words) (may be hard-coded).
`MIN_CAPTION_LEN`: $\mathbb{N}$ – Minimum caption length to avoid under-description (optional).
`DECODE_STRATEGY`: String – Decoding strategy (e.g., {"greedy","beam"}) (may be hard-coded).

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

`cpu`: Device. This is the CPU execution device.
`gpu`: Device. This is the GPU execution device (it may be unavailable).
`device`: Device. This is the selected device used for inference.

10.4.3 Assumptions

- A valid trained model is available and compatible with the input format.
- Input images are properly preprocessed.
- Constants may be hardcoded in the current implementation and moved to configuration later.

10.4.4 Access Routine Semantics

loadModel(modelPath):

- transition: Loads the trained model from `modelPath` into memory as `loadedModel`.
- output: None.
- exception: Raises `ModelLoadException` if the model file is missing, unreadable, or incompatible.

generateCaption(image):

- transition: Uses `loadedModel` to generate a caption for `image` using `DECODE_STRATEGY`.
- output: `caption`.
- exception: Raises `GenerationException` if inference fails or a valid caption cannot be produced.

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

`SCORE_RANGE`: Tuple – Allowed score range (e.g. (0, 100)); may be hardcoded.
`PASS_THRESHOLD`: $\mathbb{N}$ – Minimum acceptable aggregate score; may be hardcoded.
`METRIC_WEIGHTS`: Tuple – (Optional) weighting factors for aggregate scoring; may be omitted in early versions.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
initialize	-	-	-
evaluateCaption	CaptionText:String	PerformanceScores: List[Tuple[String, Float]]	EvaluationException

11.4 Semantics

11.4.1 State Variables

`lastScores`: List – (Optional) caches the most recently produced `PerformanceScores`.
`aggregateScore`: $\mathbb{Z}$ – (Optional) most recent computed overall score.

11.4.2 Environment Variables

None.

11.4.3 Assumptions

- `initialize` is called before `evaluateCaption`.
- Constants may be hardcoded in the current implementation and moved to configuration later.

- State variables are optional; omitting them does not affect functional correctness.
- Assumes generated captions are available and that the chosen evaluation metrics are accessible.

11.4.4 Access Routine Semantics

initialize():

- transition:
  - Initialize evaluation metrics and scoring weights
- output: None.

evaluateCaption(CaptionText):

- transition:
  - Compute performance indicators according to the selected metrics
  - (Optional) compute `aggregateScore` and clamp to `SCORE_RANGE`
  - (Optional) `lastScores` := PerformanceScores(List[Tuple[String, Float]])
- output: Returns `PerformanceScores`, a list of numerical scores summarizing caption quality and accessibility.
- exception: Raises `EvaluationException` if evaluation cannot be completed.

11.4.5 Local Functions

None.

12 MIS of FeedbackLoop 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
initialize	-	-	-
updateModel	PerformanceScores: (List[Tuple[String, Float]])	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

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

12.4.4 Access Routine Semantics

initialize():

- transition:
  - Initialize feedback integration state and model update hooks
- output: None.

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 Feedback Module

13.1 Module

`ModFeedback`

13.2 Uses

Receives generated captions and performance scores for storage, and provides them to other modules when requested.

- Receives captions and performance scores from the `FeedbackMetrics` module.
- Provides stored feedback data to the `FeedbackLoop` module for model updates.

13.3 Syntax

13.3.1 Exported Constants

`MAX_STORAGE`: $\mathbb{N}$ – Maximum number of feedback entries to store.

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
initialize	-	-	-
storeFeedback	Caption:String, Scores:List[Tuple[String, Float]]	FeedbackID:String	StorageException
getFeedback	FeedbackID:String	FeedbackRecord	RetrievalException
getAllFeedback	-	List[FeedbackRecord]	-

13.4 Semantics

13.4.1 State Variables

`feedbackDB`: Map[String, FeedbackRecord] – Stores feedback records keyed by FeedbackID.

13.4.2 Environment Variables

None.

13.4.3 Assumptions

- `initialize` is called before other operations.
- Feedback data comes from `FeedbackMetrics` module.
- Storage capacity is sufficient for system operation.

13.4.4 Access Routine Semantics

initialize():

- transition: Initializes `feedbackDB` as empty.
- output: None.

storeFeedback(Caption, Scores):

- transition: Creates new entry in `feedbackDB` with unique FeedbackID.
- output: Returns generated FeedbackID.
- exception: Raises `StorageException` if storage limit exceeded.

getFeedback(FeedbackID):

- transition: None.
- output: Returns the FeedbackRecord for given ID.
- exception: Raises `RetrievalException` if ID not found.

getAllFeedback():

- transition: None.
- output: Returns all stored FeedbackRecords.
- exception: None.

13.4.5 Local Functions

None.

14 MIS of BackendController Module

14.1 Module

`BackendController`

14.2 Uses

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

- Receives authentication and session status from the AuthenticationService Module (see Section 15) and SessionManagement module (see Section 16), respectively.
- Sends uploaded images to the ImageValidation module (see Section 17) 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 18)
- Receives the generated alt text from the ModelOutput module (see Section 8) to be presented to the user.

14.3 Syntax

14.3.1 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.

14.3.2 Exported Access Programs

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

14.4 Semantics

14.4.1 State Variables

None.

14.4.2 Environment Variables

None.

14.4.3 Assumptions

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

14.4.4 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: Returns the generated alt text or corresponding error message to the UserInterfaceInteractions module to be displayed to the user.
- output: None.
- exception: `FrontendUnreachableException` is raised if the Frontend 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.

15 MIS of AuthenticationService Module

15.1 Module

`AuthenticationService`

15.2 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.

15.3 Syntax

15.3.1 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.

15.3.2 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

15.4 Semantics

15.4.1 State Variables

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

15.4.2 Environment Variables

None.

15.4.3 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.

15.4.4 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.

15.4.5 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.

16 MIS of SessionManagement Module

16.1 Module

`SessionManagement`

16.2 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.

16.3 Syntax

16.3.1 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.

16.3.2 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

16.4 Semantics

16.4.1 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.

16.4.2 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.

16.4.3 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.

16.4.4 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.

16.4.5 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.

17 MIS of ImageValidation Module

17.1 Module

`ImageValidation`

17.2 Uses

None.

17.3 Syntax

17.3.1 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.

17.3.2 Exported Access Programs

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

17.4 Semantics

17.4.1 State Variables

None.

17.4.2 Environment Variables

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

17.4.3 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.

17.4.4 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.

17.4.5 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.

18 MIS of Logging Module

18.1 Module

`Logger`

18.2 Uses

None.

18.3 Syntax

18.3.1 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`.

18.3.2 Exported Access Programs

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

18.4 Semantics

18.4.1 State Variables

None.

18.4.2 Environment Variables

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

18.4.3 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.

18.4.4 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.

18.4.5 Local Functions

None.

19 MIS of UserInterfaceInteractions Module

19.1 Module

`InteractionsUI`

19.2 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.

19.3 Syntax

19.3.1 Exported Constants

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

19.3.2 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

19.4 Semantics

19.4.1 State Variables

None.

19.4.2 Environment Variables

None.

19.4.3 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.
- Each exported access program represents an event handler that is invoked in response to user-triggered actions such as mouse clicks and keyboard presses on the user interface.
- `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.
- Each exported access program represents an event handler that is invoked in response to user-triggered actions such as mouse clicks and keyboard presses on the user interface.

19.4.4 Access Routine Semantics

uploadImage(imageFilePath):

- transition: The HTTP client 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 is set to `altText`.
- output: returns `true` iff clipboard write succeeds, returns `false` otherwise.
- exception: `ClipboardError`.

editText(altText,editedText):

- transition: The HTTP client 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`.

19.4.5 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.

20 MIS of MainScreen Module

20.1 Module

`MainScreen`

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
renderMain	-	-	RenderError, AccessibilityError

20.4 Semantics

20.4.1 State Variables

None.

20.4.2 Environment Variables

None.

20.4.3 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.

20.4.4 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`.

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.
- **announceMainLoaded() ->** $\mathbb{B}$: calls `AccessibilityUI` to announce screen load.

21 MIS of ShowHistoryScreen Module

21.1 Module

`ShowHistoryScreen`

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
showHistory	-	-	RenderError, AccessibilityError

21.4 Semantics

21.4.1 State Variables

None.

21.4.2 Environment Variables

None.

21.4.3 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.

21.4.4 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`.

21.4.5 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.

22 MIS of LogInScreen Module

22.1 Module

`LogInScreen`

22.2 Uses

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

22.3 Syntax

22.3.1 Exported Constants

None.

22.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderLogIn	-	-	RenderError, AccessibilityError

22.4 Semantics

22.4.1 State Variables

None.

22.4.2 Environment Variables

None.

22.4.3 Assumptions

- `AccessibilityUI` is used for announcements and focus.

22.4.4 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`.

22.4.5 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.

23 MIS of UserInterfaceAccessibility Module

23.1 Module

`AccessibilityUI`

23.2 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.

23.3 Syntax

23.3.1 Exported Constants

23.3.2 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

None.

23.3.3 Environment Variables

None.

23.3.4 Assumptions

23.3.5 State Variables

None.

23.3.6 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

23.3.7 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.

23.3.8 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`.

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

24 Appendix

Appendix — Reflection

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:

Dhruv Sardana - Reflection

1. **What went well while writing this deliverable?**
   During the writing of this deliverable, it helped me understand our system design and coding to a much better extent where I was able to identify the modules and their types more clearly. The low level design became more apparent and it helped us understand the flow of data between the modules. It gave me clarification on what kind of data tables we needed to create for each module.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   During the writing of this deliverable, one of the pain points I experienced was understanding how to deploy the CI.CD pipeline for our project. I was not very familiar with GitHub actions and how to implement it. I researched about it and found some tutorials online that helped me understand how to set it up. I also discussed with my team members. Another thing that I researched was what kind of databases we would be needing and how it would affect our modules. Such having a signup verification module or just keeping it under authentication module. I discussed with backend subteam and we came to a conclusion to keep it under authentication module

Casey Francine Bulaclac - Reflection

1. **What went well while writing this deliverable?**
   When writing the Rev 0 design document deliverable, what went well was applying the TA and our peer review group's feedback. The feedback we were given from Rev -1 was very helpful in learning what was missing and/or what may have been confusing with this deliverable. This allowed our group to think more thorougly about our modules and how each one connects to each other.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   When writing the Rev 0 design document deliverable, a pain point was trying to correctly identify the type of each module. This was quite difficult at the beginning, for example, trying to understand what an abstract object is took our group a long time. To resolve this, we went over our types with our TA to ensure we understood it correctly, and asked for extra clarification regarding the module types.

Nawaal Fatima - Reflection

1. **What went well while writing this deliverable?**
   During this deliverable, knowing precisely what had to be changed was good. It provided me with direction and structure. I wasn't left feeling frustrated on why we had lost marks on certain sections due to the very detailed feedback. Also, communication among the team vastly improved and expectations were properly met. We had more consistent updates between meetings in our team, and toour supervisor.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   Juggling between coding and fixing past mistakes, then inplementing into the code any changes in an agile way was a bit hectic for me. I had to think ahead and 2 steps behind at the same time which is still hard for me to do but Im learning! To resolve these issues, I first revisited all that had to be done and then made a sequential plan to iterate through changes, discussing with the team/subteam when required. It taught me to be agile. I'm also slowly learning to not be a perfect coder - I have to make it exist first before it can be "good" and even then there's a ton of trade offs!

Fiza Sehar Individual - Reflection

1. **What went well while writing this deliverable?**
   Since we had already discussed many implementation details while building, the TA feedback was clear and actionable. I refined the environment variables to be more specific and consistent across modules by separating `cpu` and `gpu`, and replacing vague terms like "Filesystem" with explicit paths such as `checkpointDir` and `modelSavePath`. I also tightened the access routine semantics so the outputs match what is declared in the access program tables.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   The main pain point was deciding how detailed the MIS should be without adding unnecessary complexity. In particular, it was unclear whether model save locations should be treated as parameters or environment variables, and how to describe CPU/GPU usage consistently. I resolved this by aligning directly with the TA comments, making dependencies explicit by naming the file paths, standardizing device naming (`cpu`, `gpu`, `device`), and adding a simple precondition assumption (model initialized or loaded before use) to remove ambiguity.

Moly Mikhail Individual - Reflection

1. **What went well while writing this deliverable?**
   I believe many things went well while writing this deliverable. Firstly having to iterations was greatly beneficial. This allowed us to iterate on our design and make improvements based on numerous things. We made changes based on our TA feedback from the Rev0 Design Document deliverable, as well as peer feedback. This was really helpful as a lot of the feedback pointed out inconsistences and gaps within our design. I also found it helpful to reflect on our design after beginning the implementation process. Often when beginning to code and implement our design, small differences were revealed. It was helpful to compare our initial design to how we were implementing and make decisions on how on the best way to proceed.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   The first pain point I experienced during this deliverable was reviewing the module types we had initially chosen and ensuring they made sense and aligned with the definition of the types. Looking specifically, at abstract objects it was initially confusing what an initializer method was and the difference between a constrcutor. However, after meeting with our TA, I was able to ask about this and get clarification, which was really helpful.

Team Reflection

1. **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?**
   When designing the frontend modules, including MainScreen, ShowHistoryScreen, and LogInScreen, our supervisor, Jing, provided very useful guidance on the layout, logic and style of these interfaces. Jing had experience in working with our potential users so she was able to give us advice on how to design our frontend modules. These suggestions helped ensure the design was optimized for accessibility.
   For the AI, our team chose to train non Large Language Model (LLM) models. This is because of training constraints (training would be done primarily on our personal computers using open sourced datasets) and to address privacy concerns. Additionally, Jing expressed that the learning material would be the intellectual property of the professors/instructors so the optimal choice would be not to save these materials.
   The AuthenticationService module was also designed upon getting feedback from Jing. Initially, the team planned to have a single sign on but after discussing, we learned that this may result in extra steps for our potential users.
   For design decisions that did not stem from speaking to clients or proxies, our team kept in mind software design principles such as single responsibility.

2. **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?**

When creating the Module Guide and Module Interface Specification documents, specifically for Rev -1, we had to make modifications to our Verification and Validation (VnV) document. Upon creating the SessionManagement module, the team received feedback and discussed that the definition of a "session" was not clear. Therefore, we added a more clear definition for this in our VnV as this is an important terminology in our system. Other than this small addition, nothing had to be changed in the other documents while creating the design document as the team tried to stick with the original requirements written.

3. **What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)**
   One limitation of our solution is that the training of the AI model is constrained by available computational resources and data for training the model. With access to more powerful resources such as GPUs, RAM, and greater storage capacity, we could train more complex models in a faster, more efficient manner. Additionally, access to more professors data (academic learning materials) would allow the model to learn better and improve accuracy when generating alternative text for images. Furthermore, having human in the loop for captions generated would highly improve how our model understands language. Additionally, a limitation in our current backend system is the database restriction (maximum 500 rows per table) which limits the amount of users that can use the web tool. This also restricts the systems scalability in the future if needed.

4. **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)**
   One of the design solutions the team considered was making our own authentication system. However, in our chosen design, the team decided to use a third party authentication service (Supabase) to authenticate our users. This design decision allowed us to easily set up the databases and collaborate during development rather than having everything locally. It is also more secure and helps the team take steps towards preparing for production. If the team had made our own authentication system, it would be easier to control during development and have likeliness restrictions on the database (i.e. rows, number of users, etc.).
   Another design solution considered was using all STEM diagrams for training. In our chosen design, however, the team decided to narrow it down to physics diagrams. This is because the open-sourced datasets available were to generic and fine-tuning would be a problem unless the scope was narrowed down. Had we used all STEM diagrams, it would be applicable to more students but our team wants to prioritize the accuracy of our generated alt text over diverse consumers. We hope to build on this model given more data in the future.

5. **(After you have implemented another team's module, which means this**

isn't filled in until after the original deadline). 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 another team's logging module helped us interpret the effectiveness of our peer's design documentation. It also helped us reflect on our own design document to double check any areas of confusion. Overall, the Logger MIS by Team 17 was very well-structured and clearly written, and most of the required functionality was straightforward to implement directly from their specification. We didn't ask the team questions directly as most of their MIS was mostly clear, however, we made a few assumptions.

Some areas required assumptions or interpretation due to a few missing specification or confusion. One major assumption involved log rotation and archival behaviour. The MIS states that logs are "rotated" once they exceed a maximum size but it doesn't define how to rotate these files when archiving. We assumed a rolling log shere older log files are shifted (e.g., file 1 $\rightarrow$ file 2 $\rightarrow$ file 3) within the same log directory. Additionally, although a `logDir` environment variable is referenced, its path and structure were not specified, so we assumed that all log files reside within a single configurable directory called "/log". The representation of a `LogEntry` was another area of interpretation. While the MIS defines its fields abstractly, it does not specify how entries are stored. We assumed a dictionary-based representation serialized to JSON, which is shown in our implementation.

Some details required assumptions during implementation. For example, we assumed that the limit parameter in `getRecentLogs` is an integer that specifies how many log entries should be returned. To efficiently retrieve recent logs from large log files, we implemented a helper function (`read_last_n_lines`) that reads the file in fixed-size chunks instead of loading the entire file into memory. Since the MIS did not specify how file reading should be handled, we assumed a reasonable chunk size and defined a constant `block_size` = 4096 bytes as part of the implementation.

Multiple access to logging was also an assumption and a design consideration that we implemented for the team. We assumed that multiple API requests could call logging simultaneously, which explains our use of locking during archival operations to ensure thread safety. Additionally, the purpose of the `emitAlert` local function was a little unclear, as severe events already raise errors. Therefore, interpreted `emitAlert` as an optional feature for notifying external monitoring or alerting systems. However, its role was unclear since errors are already raised through exceptions. Finally, while the MIS is generally thorough, implementation could have been eased by including explicit data

types directly in the exported access program table and by further clarifying helper function behavior such as `formatLogEntry`.

We believe theres changes that could be made to the documentation to ensure outsiders can better implement the logger module, with more clarity and confidence. Looking at both groups that implemented our modules, it is clear they determined the appropriate format and structure of the logs. For example, both groups included a time stamp in their log entries even though this was not explicitly stated in our MIS. Therefore, in order to make the process go more smoothly for outsiders, we should update the documentation to explicitly define the log entry format and required fields. Furthermore, one external group implemented a `RotatingFileHandler` for log rotation. This highlights that our documentation could have better specified how edge cases should be handled, such as the log file exceeding the maximum allowed size. Beyond this, we believe our documentation correctly specified the intended behavior and functionality of the Logging module. This is evident in that both external teams were able to correctly implement the modules that could be used the `Reading4all` system without major assumptions made.