Module Interface Specification for Software Engineering

Team #1, Sanskrit Ciphers
Omar El Aref
Dylan Garner
Muhammad Umar Khan
Aswin Kuganesan
Yousef Shahin

November 13, 2025

1 Revision History

Date	Version	Notes
Nov. 13, 2025	1.0	Added Section 2: Symbols, Abbreviations and Acronyms.
Nov. 13, 2025	1.1	Added Section 3: Introduction.
Nov. 13, 2025	1.2	Added Section 4: Notation.
Nov. 13, 2025	1.3	Added Section 5: Module Decomposition.
Nov. 13, 2025	1.4	Added Sections 6–29: Module Interface Specifications for all modules.
Nov. 13, 2025	1.5	Added Reflection and References

2 Symbols, Abbreviations and Acronyms

The following symbols, abbreviations, and acronyms are used throughout this Module Interface Specification (MIS):

Project-Specific

- **SMFRP** — Sanskrit Manuscript Fragment Reconstruction Platform
- **SRS** — System Requirements Specification
- **MG** — Module Guide
- **MIS** — Module Interface Specification

Layer & Module Prefixes

- **HW** — Hardware-Hiding Level
- **UI** — User Interface Layer
- **Svc** — Service Layer
- **ML** — Machine Learning Layer
- **Data** — Data Management Layer
- **Util** — Utility / Software Decision Level

UI / Web Terms

- **DOM** — Document Object Model
- **UI** — User Interface
- **HTML** — HyperText Markup Language
- **API** — Application Programming Interface
- **URL** — Uniform Resource Locator

Data Types & Structures

- **ID** — Identifier (e.g., ImageID, FragmentID)
- **MB** — Megabyte
- **JSON** — JavaScript Object Notation
- **HTTP** — Hypertext Transfer Protocol
- **REST** — Representational State Transfer

Exceptions / Errors

- **UIRenderError** — User Interface Rendering Error
- **NetworkError** — Network Communication Error
- **ValidationError** — Input Validation Error
- **UploadError** — File Upload Failure
- **AuthError** — Authentication Error
- **NotFoundError** — Missing Resource
- **StorageWriteError** / **StorageReadError** — File I/O Problems

Machine Learning Terms

- **OCR** — Optical Character Recognition
- **ML** — Machine Learning

Contents

List of Tables

3 Introduction

The following document details the Module Interface Specifications (MIS) for the Sanskrit Manuscript Fragment Reconstruction Platform (SMFRP). This system supports researchers and scholars in reconstructing fragmented Sanskrit manuscripts by providing a unified digital workspace that integrates fragment management, intelligent matching, and image-based reconstruction tools.

The MIS defines the external behavior and interfaces for each software module in the system. Each module is specified in terms of its responsibilities, provided services, hidden state (secrets), environment variables, and exported access programs. These specifications are written at the right level of abstraction, focusing on logical interfaces and responsibilities rather than low-level implementation details such as programming language syntax or data structures.

This document ensures that the system achieves consistency, and minimal coupling across modules. The defined interfaces promote high cohesion, and generality, enabling independent development, verification, and maintenance of each module.

Complementary Documents

- **System Requirements Specification (SRS)** – Defines the system's functional and non-functional requirements. The SRS (Ciphers, 2025b) can be accessed at: SRS
- **Module Guide (MG)** – Outlines the overall software architecture, module decomposition, and dependency hierarchy.

Repository

The full project documentation and implementation can be found in the team's GitHub repository

4 Notation

In this section, we define the notation used throughout the Module Interface Specification. The purpose of this notation is to describe, in a clear and consistent manner, what each module provides and requires, without referencing any specific programming language or implementation detail.

Each module in this MIS is described in terms of:

- **State Variables** – The module's internal data that is not visible externally.
- **Environment Variables** – Data received from or affecting the external environment (e.g., databases, file storage, APIs).

- **Exported Constants and Access Programs** – The operations, services, or functions the module provides to other modules.

This approach focuses on interface behavior, not implementation syntax (such as JSON or Python code), ensuring that the interfaces remain consistent.

4.1 Primitive Data Types

Table 1: Primitive Data Types

Data Type	Notation	Description
Character	`char`	A single symbol or letter.
Integer	`int`	Whole numbers with no fractional part.
Real	`float`	Numbers with decimal or fractional parts.
Boolean	`bool`	Represents logical truth values (true or false).

4.2 Derived Data Types

In addition to the primitive types above, the system specification also makes use of the following derived or composite types:

- **Sequence:** An ordered collection of elements of the same data type. Example: A list of fragment identifiers or saved user sessions.
- **String:** A sequence of characters, typically used for textual input, labels, or metadata fields.
- **Tuple:** An ordered collection of values that may be of different data types. Commonly used to represent grouped entities such as `(FragmentId, ConfidenceScore)` or `(x, y, width, height)`.
- **Record / Object:** A structured type that groups named fields with defined data types, such as `{id: FragmentId, script: String, confidence: Float}`.
- **Function:** A mapping defined by the types of its inputs and outputs. Functions are specified by their type signatures (inputs/outputs) and behavior (preconditions, postconditions, and effects).

4.3 Interface and Parameter Conventions

This section explains how interfaces and operations are written in the MIS. The goal is to make them easy to read and understand while staying general enough to apply to any implementation.

- **Inputs and Outputs:** Labeled as `in:` and `out:` in operation definitions. Example: `search(in: SearchQuery, out: SearchResult)`
- **Operation Format:** `functionName(in: InputType, out: OutputType) → may throw ErrorType`
- **Module Naming Convention:** Each module name follows the format `Layer.Module`, for example: `UI.Canvas`, `Svc.MatchOrch`, or `Data.FragmentStore`.
- **Boolean Conditions:** Represented using logical symbols such as $\wedge$ (and), $\vee$ (or), $\neg$ (not), and $\implies$ (implies).

5 Module Decomposition

The following section summarizes the module hierarchy for the *Sanskrit Manuscript Fragment Reconstruction Platform*. It is consistent with the **Module Guide (MG)** for this project and represents how the system has been decomposed into modules at different abstraction levels.

Each level in the hierarchy captures a different design concern:

- **Hardware-Hiding Level:** Interfaces directly with the system's physical or platform-dependent resources.
- **Behaviour-Hiding Level:** Encapsulates the main functional behavior of the system, implementing the use cases described in the SRS.
- **Software Decision Level:** Contains lower-level utility modules and software decisions that simplify implementation but are not visible to the user.

5.1 Module Hierarchy

Table 2: Module Hierarchy

Level 1	Level 2 Modules
Hardware-Hiding (HW)	Image Storage Interface
Behaviour-Hiding	**User Interface (UI):** `UI.Canvas`, `UI.Search`, `UI.Auth`, `UI.Upload` **Service Layer (Svc):** `Svc.API`, `Svc.Search`, `Svc.AuthZ`, `Svc.Session`, `Svc.ProcOrch`, `Svc.MatchOrch` **Data Management (Data):** `Data.FragmentStore`, `Data.Catalog`, `Data.User`, `Data.Project` **Machine Learning Modules (ML):** `ML.EdgeMatch`, `ML.Damage`, `ML.ScriptClass`, `ML.OCR`, `ML.TextSim`
Software Decision	Utility and Support Modules (Error Handling, Logging, Configuration Management, Test Stubs)

5.2 Description of Levels

5.2.1 Hardware-Hiding Level

This level hides all hardware and platform dependencies from the rest of the system. It includes interfaces that communicate with the file storage system, image repositories, and cloud infrastructure. These components provide a uniform interface for accessing images and metadata regardless of the underlying storage medium.

5.2.2 Behaviour-Hiding Level

This level defines the functional behavior of the system, implementing the requirements outlined in the SRS. It includes the user-facing modules (UI), the application logic (Service Layer), data persistence (Data Layer), and intelligent processing (ML Layer). These modules interact through clearly defined service interfaces, promoting modularity and separation of concerns.

5.2.3 Software Decision Level

This level includes supporting components that are primarily design decisions rather than functional necessities. These modules include configuration handlers, utility libraries, and testing or logging frameworks that support development, debugging, and maintenance.

6 MIS of HW.ImageStorage

6.1 Module

HW.ImageStorage (Image Storage Interface)

This module provides a uniform interface for reading, writing, and managing manuscript fragment image files and related metadata in the underlying file system or cloud storage. It hides all platform-specific file operations from higher-level modules and ensures consistent access and naming conventions.

6.2 Uses

- `Data.FragmentStore` – for linking stored image references to fragment metadata records.
- `Svc.ProcOrch` – for initiating preprocessing or orientation correction workflows once an image is stored.
- External file system or object storage API.

6.3 Syntax

6.3.1 Exported Constants

- `MAX_FILE_SIZE = 50MB`
- `SUPPORTED_FORMATS = {.jpg, .png, .tiff}`
- `STORAGE_PATH = "/images/"`

6.3.2 Exported Access Programs

Table 3: HW.ImageStorage - Exported Access Programs

Name	In	Out	Exceptions
`storeImage`	`file: ImageFile, metadata: Record`	`ImageID`	`InvalidFormatError, StorageWriteError`
`retrieveImage`	`id: ImageID`	`ImageFile`	`NotFoundError, StorageReadError`
`deleteImage`	`id: ImageID`	`bool`	`NotFoundError, StorageWriteError`
`getMetadata`	`id: ImageID`	`Record`	`NotFoundError`

6.4 Semantics

6.4.1 State Variables

- `storageIndex: Map[ImageID, FilePath]`
- `metadataCache: Map[ImageID, Record]`

6.4.2 Environment Variables

- `FileSystem / CloudStorage` – external persistence for image binaries

6.4.3 Assumptions

- The underlying storage service guarantees atomic read/write operations.
- ImageIDs are globally unique.

6.4.4 Access Routine Semantics

`storeImage(file, metadata)`:

- transition: file is written to storage; `storageIndex` and `metadataCache` updated with new entry.
- output: a newly generated `ImageID`.
- exception: `InvalidFormatError` if file type not in `SUPPORTED_FORMATS`; `StorageWriteError` if write fails.

`retrieveImage(id)`:

- output: image file associated with `id`.
- exception: `NotFoundError` if `id` absent from `storageIndex`; `StorageReadError` if read fails.

`deleteImage(id)`:

- transition: remove image file from storage; delete entries from `storageIndex` and `metadataCache`.
- output: `true` if deletion successful.
- exception: `NotFoundError` if `id` not present; `StorageWriteError` if deletion fails.

`getMetadata(id)`:

- output: metadata record associated with `id`.
- exception: `NotFoundError` if no record exists.

6.4.5 Local Functions

- `generateImageID(file: ImageFile): ImageID`
- `validateFormat(file: ImageFile): bool`

7 MIS of UI.Auth

7.1 Module

UI.Auth (Authentication Interface)

This module provides the browser-facing login and logout experience. It collects user credentials, displays relevant error messages, and forwards authentication requests to the backend service. It does not perform authentication logic itself.

7.2 Uses

- `Svc.AuthZ` – for validating credentials and issuing session tokens.
- `Svc.Session` – for managing client-side session lifecycle.

7.3 Syntax

7.3.1 Exported Constants

- `MAX_LOGIN_ATTEMPTS = 5`

7.3.2 Exported Access Programs

Table 4: UI.Auth - Exported Access Programs

Name	In	Out	Exceptions
`renderLoginForm`	`none`	`none`	`UIRenderError`
`submitCredentials`	`creds: UserCredentials`	`AuthResult`	`AuthError, NetworkError, InboxOverflowError`
`logoutUser`	`none`	`bool`	`NetworkError`

7.4 Semantics

7.4.1 State Variables

- `loginAttempts: int` – count of consecutive failed attempts in current session.
- `authErrorMessage: String` – message shown on failed login.

7.4.2 Environment Variables

- Browser window and Document Object Model(DOM).

7.4.3 Assumptions

- Network connectivity is available for requests to `Svc.AuthZ`.
- Password input is masked and handled securely by the browser.

7.4.4 Access Routine Semantics

`renderLoginForm()`:

- transition: updates DOM to show login form, clears previous error message.
- output: none.
- exception: `UIRenderError` if DOM cannot be updated.

`submitCredentials(creds)`:

- transition: sends credentials to `Svc.AuthZ`; updates `loginAttempts` and `authErrorMessage` based on response.
- output: `AuthResult` indicating success or failure.
- exception: `AuthError` if credentials rejected; `NetworkError` if backend unreachable.

`logoutUser()`:

- transition: clears client-side session state and redirects to login view.
- output: `true` if logout succeeds.
- exception: `NetworkError` if logout call to backend fails.

7.4.5 Local Functions

- `displayError(msg: String)` – updates UI error region.
- `resetForm()` – clears form fields.

8 MIS of UI.Upload

8.1 Module

UI.Upload (Fragment Upload Interface)

This module allows users to select, preview, and submit manuscript fragment images for processing. It performs basic client-side validation and delegates storage and preprocessing to backend services.

8.2 Uses

- `Svc.API` – for upload endpoints.
- `HW.ImageStorage` – indirectly, via backend, for persistent image storage.

8.3 Syntax

8.3.1 Exported Constants

- `MAX_FILES_PER_UPLOAD = 20`

8.3.2 Exported Access Programs

Table 5: UI.Upload - Exported Access Programs

Name	In	Out	Exceptions
`renderUploadPanel`	`none`	`none`	`UIRenderError`
`selectFiles`	`files: Sequence[ImageFile]`	`none`	`ValidationError`
`submitUpload`	`none`	`UploadResult`	`NetworkError, UploadError`

8.4 Semantics

8.4.1 State Variables

- `pendingFiles: Sequence[ImageFile]`
- `uploadErrorMessage: String`

8.4.2 Environment Variables

- Browser file picker and local file system access (sandboxed).

8.4.3 Assumptions

- The browser enforces configured file size constraints on selection when possible.

8.4.4 Access Routine Semantics

`renderUploadPanel()`:

- transition: renders the upload panel, including file input control and status indicators.
- output: none.

- exception: `UIRenderError` if DOM cannot be updated.

`selectFiles(files)`:

- transition: validates file count and basic properties; if valid, stores them in `pendingFiles`.
- output: none.
- exception: `ValidationError` if any file exceeds size or format constraints.

`submitUpload()`:

- transition: sends `pendingFiles` to backend via `Svc.API`; clears `pendingFiles` on success.
- output: `UploadResult` containing list of created fragment IDs.
- exception: `NetworkError` if request fails; `UploadError` if backend rejects the upload.

8.4.5 Local Functions

- `validateFiles(files)` – checks count, format, and size bounds.
- `previewThumbnails()` – generates and displays thumbnails for selected files.

9 MIS of UI.Canvas

9.1 Module

UI.Canvas (Interactive Fragment Workspace)

This module provides the interactive canvas where scholars can view, move, rotate, and arrange multiple fragments simultaneously. It displays match suggestions and visual overlays for edges, damage patterns, or text regions.

9.2 Uses

- `Svc.MatchOrch` – for requesting match suggestions.
- `Svc.Session` – for loading and saving canvas layouts.
- `Svc.API` – for fetching fragment images and overlays.

9.3 Syntax

9.3.1 Exported Constants

- `MAX_FRAGMENTS_ON_CANVAS = 50`

9.3.2 Exported Access Programs

Table 6: UI.Canvas - Exported Access Programs

Name	In	Out	Exceptions
renderCanvas	state: CanvasState	none	UIRenderError
placeFragment	id: FragmentID, position: Tuple	none	LayoutError
rotateFragment	id: FragmentID, angle: float	none	LayoutError
requestMatches	id: FragmentID	none	NetworkError
saveLayout	none	LayoutID	NetworkError

9.4 Semantics

9.4.1 State Variables

- currentState: CanvasState – positions, rotations, and visibility for fragments.
- selectedFragment: FragmentID or null
- activeMatchOverlay: MatchResultList

9.4.2 Environment Variables

- Browser rendering surface (HTML canvas or equivalent).

9.4.3 Assumptions

- Device supports required rendering capabilities for zoom and pan.

9.4.4 Access Routine Semantics

renderCanvas(state):

- transition: updates the visual layout to match state.
- output: none.
- exception: UIRenderError if rendering fails.

placeFragment(id, position):

- transition: updates currentState with new position for id.
- output: none.

- exception: `LayoutError` if fragment exceeds bounds or limit exceeded.

`rotateFragment(id, angle)`:

- transition: updates rotation of `id` in `currentState`.
- output: none.
- exception: `LayoutError` if rotation cannot be applied.

`requestMatches(id)`:

- transition: sends match request to `Svc.MatchOrch`; updates `activeMatchOverlay` when response is received.
- output: none.
- exception: `NetworkError` if request fails.

`saveLayout()`:

- transition: persists `currentState` via `Svc.Session`.
- output: a `LayoutID` for later retrieval.
- exception: `NetworkError` if persistence fails.

9.4.5 Local Functions

- `snapToEdges(id)` – adjusts position so nearby edges align visually.
- `highlightMatches(id)` – visually emphasizes suggested neighbors.

10 MIS of UI.Search

10.1 Module

UI.Search (Database Search and Exploration Interface)

This module provides the interface for querying the fragment database, filtering results, and viewing metadata. It supports search by script type, line count, fragment size, and other criteria.

10.2 Uses

- `Svc.Search` – for executing search queries.
- `Svc.API` – for retrieving fragment summaries and export payloads.

10.3 Syntax

10.3.1 Exported Constants

- `RESULTS_PAGE_SIZE = 25`

10.3.2 Exported Access Programs

Table 7: UI.Search - Exported Access Programs

Name	In	Out	Exceptions
`renderSearchPanel`	`none`	`none`	`UIRenderError`
`submitQuery`	`q: SearchQuery`	`none`	`ValidationError`, `NetworkError`
`nextPage`	`none`	`none`	`NetworkError`
`exportResults`	`format: ExportFormat`	`none`	`NetworkError`, `ExportError`

10.4 Semantics

10.4.1 State Variables

- `currentQuery: SearchQuery`
- `currentPage: int`
- `results: SearchResultPage`

10.4.2 Environment Variables

- Browser viewport for displaying tables and filters.

10.4.3 Assumptions

- Backend search service responds within acceptable time bounds defined in the SRS.

10.4.4 Access Routine Semantics

`renderSearchPanel()`:

- transition: displays search form, filters, and an empty or last-used results view.
- output: none.
- exception: `UIRenderError` if rendering fails.

`submitQuery(q)`:

- transition: validates query parameters; if valid, sends query to `Svc.Search` and updates `results` and `currentPage`.
- output: none.
- exception: `ValidationError` if query malformed; `NetworkError` if backend unreachable.

`nextPage()`:

- transition: increments `currentPage` and requests next result page.
- output: none.
- exception: `NetworkError` if request fails.

`exportResults(format)`:

- transition: sends export request for current query to backend.
- output: none at UI level; triggers file download in browser.
- exception: `NetworkError` if backend unreachable; `ExportError` if export fails.

10.4.5 Local Functions

- `renderResultsTable(results)` – updates DOM table from latest `SearchResultPage`.
- `validateSearchQuery(q)` – checks filter ranges and required fields.

11 MIS of Svc.API

11.1 Module

Svc.API (Backend API Gateway)

This module exposes the backend functionality of the Sanskrit Manuscript Fragment Reconstruction Platform as a REST-style service layer. It routes incoming HTTP requests from the UI to the corresponding internal services, enforces basic request validation, and standardizes API responses.

11.2 Uses

- `Svc.AuthZ` – for authentication and authorization checks.
- `Svc.Session` – for resolving current user/session context.
- `Svc.ProcOrch` – for preprocessing and normalization requests.
- `Svc.MatchOrch` – for fragment match discovery.
- `Svc.Search` – for database search and exploration.
- `Data.FragmentStore`, `Data.User`, `Data.Project`.

11.3 Syntax

11.3.1 Exported Constants

- `API_VERSION = "v1"`
- `MAX_REQUEST_SIZE` – logical upper bound on payload size (aligned with SRS).

11.3.2 Exported Access Programs

Table 8: Svc.API - Exported Access Programs

Name	In	Out	Exceptions
handleAuthRequest	req: HttpRequest	HttpResponse	AuthError
handleUploadRequest	req: HttpRequest	HttpResponse	ValidationError, ServiceError
handleMatchRequest	req: HttpRequest	HttpResponse	ServiceError
handleSearchRequest	req: HttpRequest	HttpResponse	ValidationError, ServiceError
handleSessionRequest	req: HttpRequest	HttpResponse	ServiceError

11.4 Semantics

11.4.1 State Variables

- `routeTable: Map[Endpoint, Handler]` – mapping between URL paths and internal handlers.

11.4.2 Environment Variables

- HTTP server environment providing incoming requests and outgoing responses.

11.4.3 Assumptions

- All requests include sufficient information to resolve user/session context when required.
- Lower-level services enforce their own invariants in addition to basic checks at this layer.

11.4.4 Access Routine Semantics

`handleAuthRequest(req)`:

- transition: forwards credentials/token to `Svc.AuthZ`; logs outcome.
- output: `HttpResponse` with status code and (if successful) session token or user info.
- exception: `AuthError` if authentication fails or token invalid.

`handleUploadRequest(req)`:

- transition: validates payload size and type; forwards work to `Svc.ProcOrch` for ingestion.
- output: `HttpResponse` including created fragment IDs or error details.
- exception: `ValidationError` if request violates constraints; `ServiceError` if downstream failure occurs.

`handleMatchRequest(req)`:

- transition: parses fragment identifier and forwards to `Svc.MatchOrch`.
- output: `HttpResponse` containing ranked match suggestions.
- exception: `ServiceError` if orchestrator fails or times out.

`handleSearchRequest(req)`:

- transition: validates search query parameters; forwards to `Svc.Search`.
- output: `HttpResponse` containing search results page.
- exception: `ValidationError` for malformed queries; `ServiceError` on backend failures.

`handleSessionRequest(req)`:

- transition: routes session save/load actions to `Svc.Session`.
- output: `HttpResponse` with session payload or confirmation.
- exception: `ServiceError` if the session operation fails.

11.4.5 Local Functions

- `parseJSONBody(req)` – extracts and validates JSON payload.
- `buildErrorResponse(code, msg)` – standardizes error responses.

12 MIS of Svc.AuthZ

12.1 Module

Svc.AuthZ (Authentication and Authorization Service)

This module validates user credentials, issues and verifies session tokens, and enforces access control for protected operations.

12.2 Uses

- `Data.User` – for user credentials and role lookup.
- Cryptographic utilities (hashing, token signing).

12.3 Syntax

12.3.1 Exported Constants

- `TOKEN_TTL_MINUTES` – session token lifetime.
- `MAX_FAILED_ATTEMPTS` – lockout threshold.

12.3.2 Exported Access Programs

Table 9: Svc.AuthZ - Exported Access Programs

Name	In	Out	Exceptions
`authenticate`	`creds: UserCredentials`	`AuthResult`	`AuthError`
`validateToken`	`token: String`	`UserContext`	`AuthError`
`checkAccess`	`ctx: UserContext, action: String`	`bool`	`AuthError`

12.4 Semantics

12.4.1 State Variables

- `failedAttempts: Map[UserID, int]`

12.4.2 Environment Variables

- Secure storage for password hashes and secret keys.

12.4.3 Assumptions

- Passwords are stored as salted hashes only.
- Time and randomness sources are trustworthy for token generation.

12.4.4 Access Routine Semantics

`authenticate(creds)`:

- transition: verifies credentials against stored hash; updates `failedAttempts`.
- output: `AuthResult` containing success flag and token on success.
- exception: `AuthError` if user locked out or credentials invalid.

`validateToken(token)`:

- output: `UserContext` if token valid and not expired.
- exception: `AuthError` if token invalid or expired.

`checkAccess(ctx, action)`:

- output: `true` if user role in `ctx` is allowed to perform `action`.
- exception: `AuthError` if context invalid.

12.4.5 Local Functions

- `hashPassword(pw)` – hashes password for comparison.
- `generateToken(user)` – constructs signed token.

13 MIS of Svc.Session

13.1 Module

Svc.Session (Session and Workspace Management Service)

This module manages user sessions, saved canvas layouts, and ongoing reconstruction projects. It provides persistent storage and retrieval of user workspace state.

13.2 Uses

- `Data.Project` – for project-level data and references.
- `Data.User` – for associating sessions with users.

13.3 Syntax

13.3.1 Exported Constants

- `MAX_SAVED_LAYOUTS_PER_USER`

13.3.2 Exported Access Programs

Table 10: Svc.Session - Exported Access Programs

Name	In	Out	Exceptions
`saveCanvasState`	`ctx: UserContext, state: CanvasState`	`LayoutID`	`SessionError`
`loadCanvasState`	`ctx: UserContext, id: LayoutID`	`CanvasState`	`NotFoundError, SessionError`
`listLayouts`	`ctx: UserContext`	`Sequence[LayoutSummary]`	`SessionError`

13.4 Semantics

13.4.1 State Variables

- `layoutStore: Map[LayoutID, (UserID, CanvasState)]`

13.4.2 Environment Variables

- Persistent database for layouts and project metadata.

13.4.3 Assumptions

- `UserContext` is valid and has been authenticated by `Svc.AuthZ`.

13.4.4 Access Routine Semantics

`saveCanvasState(ctx, state)`:

- transition: creates or updates an entry in `layoutStore` for the user.
- output: generated `LayoutID`.
- exception: `SessionError` if write fails or limit exceeded.

`loadCanvasState(ctx, id)`:

- output: `CanvasState` associated with user and layout ID.
- exception: `NotFoundError` if layout does not exist or not owned by user; `SessionError` on read failure.

`listLayouts(ctx)`:

- output: summaries of layouts owned by the user.
- exception: `SessionError` if retrieval fails.

13.4.5 Local Functions

- `generateLayoutID()` – creates unique identifier for layouts.

14 MIS of Svc.ProcOrch

14.1 Module

Svc.ProcOrch (Preprocessing and Normalization Orchestrator)

This module coordinates image preprocessing workflows such as orientation correction, denoising, and resolution standardization before downstream analysis.

14.2 Uses

- `HW.ImageStorage` – for reading and writing image files.
- `ML.EdgeMatch`, `ML.Damage` – where preprocessing parameters are shared.
- External image processing library (for example, OpenCV).

14.3 Syntax

14.3.1 Exported Constants

- `TARGET_RESOLUTION` – canonical resolution for normalized images.
- `MAX_ORIENTATION_ERROR` – tolerated post-correction skew.

14.3.2 Exported Access Programs

Table 11: Svc.ProcOrch - Exported Access Programs

Name	In	Out	Exceptions
enqueuePreprocessing	id: ImageID	JobID	PreprocError
runPreprocessing	job: JobID	PreprocResult	PreprocError

14.4 Semantics

14.4.1 State Variables

- jobQueue: Sequence[JobID]

14.4.2 Environment Variables

- Execution environment for batch or async jobs.

14.4.3 Assumptions

- Input image already stored by HW.ImageStorage.

14.4.4 Access Routine Semantics

enqueuePreprocessing(id):

- transition: creates a preprocessing job for id and appends it to jobQueue.
- output: JobID for tracking.
- exception: PreprocError if job cannot be created.

runPreprocessing(job):

- transition: retrieves image by ImageID, applies normalization steps, and writes results back to storage.
- output: PreprocResult containing normalized image reference and metrics (for example, estimated orientation error).
- exception: PreprocError if any step fails.

14.4.5 Local Functions

- correctOrientation(img) – aligns text horizontally.
- normalizeResolution(img) – resamples to TARGET_RESOLUTION.

15 MIS of Svc.MatchOrch

15.1 Module

Svc.MatchOrch (Fragment Matching Orchestrator)

This module coordinates AI models for fragment matching, aggregates model outputs, and produces ranked match suggestions for a given fragment.

15.2 Uses

- `ML.EdgeMatch`, `ML.Damage`, `ML.TextSim`.
- `Data.FragmentStore` – for retrieving fragment metadata.

15.3 Syntax

15.3.1 Exported Constants

- `MAX_TOP_MATCHES = 10`
- `MATCH_CONFIDENCE_THRESHOLD = 0.6`

15.3.2 Exported Access Programs

Table 12: Svc.MatchOrch - Exported Access Programs

Name	In	Out	Exceptions
`generateMatches`	`frag: FragmentID`	`MatchResultList`	`MatchError`
`aggregateScores`	`scores: Sequence[ModelScore]`	`float`	`AggregationError`

15.4 Semantics

15.4.1 State Variables

- `recentResults: Map[FragmentID, MatchResultList]`

15.4.2 Environment Variables

- ML model serving endpoints.

15.4.3 Assumptions

- Each model returns normalized scores in $[0, 1]$.

15.4.4 Access Routine Semantics

`generateMatches(frag)`:

- transition: queries all relevant ML models; aggregates and sorts candidate matches; updates `recentResults`.
- output: `MatchResultList` of top candidates above threshold.
- exception: `MatchError` if models unavailable or response invalid.

`aggregateScores(scores)`:

- output: combined score (for example, weighted mean) over model scores.
- exception: `AggregationError` if sequence empty or weights invalid.

15.4.5 Local Functions

- `filterByThreshold(results)` – drops candidates below `MATCH_CONFIDENCE_THRESHOLD`.
- `sortByScore(results)` – orders candidates descending by final score.

16 MIS of Svc.Search

16.1 Module

Svc.Search (Search and Exploration Service)

This module executes queries over stored fragments and metadata, providing paginated and filterable search results to the UI.

16.2 Uses

- `Data.FragmentStore`, `Data.Catalog`.

16.3 Syntax

16.3.1 Exported Constants

- `DEFAULT_PAGE_SIZE = 25`

16.3.2 Exported Access Programs

Table 13: Svc.Search - Exported Access Programs

Name	In	Out	Exceptions
executeQuery	q: SearchQuery, page: int	SearchResultPage	SearchError
exportResults	q: SearchQuery, format: ExportFormat	ExportPayload	SearchError, ExportError

16.4 Semantics

16.4.1 State Variables

- none

16.4.2 Environment Variables

- Database engine for indexing and querying fragment records.

16.4.3 Assumptions

- All searchable fields are indexed appropriately for performance.

16.4.4 Access Routine Semantics

`executeQuery(q, page)`:

- output: `SearchResultPage` containing fragment summaries for the requested page.
- exception: `SearchError` if query invalid or backend failure occurs.

`exportResults(q, format)`:

- output: `ExportPayload` that can be streamed to client as CSV or PDF.
- exception: `SearchError` on query failure; `ExportError` if export generation fails.

16.4.5 Local Functions

- `buildQueryPlan(q)` – compiles filter and sort options into database query.
- `formatExport(results, format)` – converts results into specified output format.

17 MIS of Data.FragmentStore

17.1 Module

Data.FragmentStore (Fragment Metadata Store)

This module is responsible for storing and retrieving logical fragment entities. It maintains metadata such as fragment IDs, associated image IDs, script labels, line counts, and status flags, and acts as the core record of fragments used throughout the system.

17.2 Uses

- `HW.ImageStorage` – via stored `ImageID` references.
- `Data.Catalog` – for linking to external catalogue identifiers when available.

17.3 Syntax

17.3.1 Exported Constants

- `MAX_LABEL_LENGTH`

17.3.2 Exported Access Programs

Table 14: Data.FragmentStore - Exported Access Programs

Name	In	Out	Exceptions
`createFragment`	`rec: FragmentRecord`	`FragmentID`	`FragmentError`
`getFragment`	`id: FragmentID`	`FragmentRecord`	`NotFoundError`
`updateFragment`	`id: FragmentID, rec: FragmentRecord`	`bool`	`FragmentError, NotFoundError`
`deleteFragment`	`id: FragmentID`	`bool`	`FragmentError, NotFoundError`

17.4 Semantics

17.4.1 State Variables

- `fragmentTable: Map[FragmentID, FragmentRecord]`

17.4.2 Environment Variables

- Relational database or equivalent persistent storage.

17.4.3 Assumptions

- Fragment IDs are globally unique and stable over time.

17.4.4 Access Routine Semantics

`createFragment(rec)`:

- transition: inserts a new record into `fragmentTable`.
- output: new `FragmentID`.
- exception: `FragmentError` if insert fails or constraints violated.

`getFragment(id)`:

- output: record associated with `id`.
- exception: `NotFoundError` if no record exists.

`updateFragment(id, rec)`:

- transition: replaces existing record for `id` with `rec`.
- output: `true` if update successful.
- exception: `NotFoundError` if `id` unknown; `FragmentError` on write failure.

`deleteFragment(id)`:

- transition: removes record associated with `id`.
- output: `true` if deletion successful.
- exception: `NotFoundError` if `id` unknown; `FragmentError` on delete failure.

17.4.5 Local Functions

- `validateFragment(rec)` – checks required fields and constraints.

18 MIS of Data.Catalog

18.1 Module

Data.Catalog (External Catalogue Link Store)

This module maintains mappings between internal fragments and external catalogue identifiers from libraries, collections, or previous editions.

18.2 Uses

- `Data.FragmentStore` – for core fragment records.

18.3 Syntax

18.3.1 Exported Constants

- `MAX_CATALOG_ID_LENGTH`

18.3.2 Exported Access Programs

Table 15: Data.Catalog - Exported Access Programs

Name	In	Out	Exceptions
`linkFragmentToCatalog`	`frag: FragmentID, catID: String`	`bool`	`CatalogError`
`getCatalogIDs`	`frag: FragmentID`	`Sequence[String]`	`CatalogError`
`findByCatalogID`	`catID: String`	`Sequence[FragmentID]`	`CatalogError`

18.4 Semantics

18.4.1 State Variables

- `catalogMap: Map[FragmentID, Sequence[String]]`

18.4.2 Environment Variables

- Same underlying database as the fragment store or a logically related schema.

18.4.3 Assumptions

- External catalogue identifiers are opaque strings; uniqueness is handled externally.

18.4.4 Access Routine Semantics

`linkFragmentToCatalog(frag, catID)`:

- transition: appends `catID` to the list of identifiers for `frag`.
- output: `true` on success.
- exception: `CatalogError` on write failure.

`getCatalogIDs(frag)`:

- output: all catalogue IDs associated with `frag`.
- exception: `CatalogError` if retrieval fails.

`findByCatalogID(catID)`:

- output: all `FragmentID` values linked to `catID`.
- exception: `CatalogError` if query fails.

18.4.5 Local Functions

- `normalizeCatalogID(catID)` – optional normalization (trim, case).

19 MIS of Data.User

19.1 Module

Data.User (User Account Store)

This module stores user accounts, credentials (hashed), roles, and profile information required for authentication and access control.

19.2 Uses

- Cryptographic utilities for password hashing.

19.3 Syntax

19.3.1 Exported Constants

- `MAX_USERNAME_LENGTH`
- `MAX_EMAIL_LENGTH`

19.3.2 Exported Access Programs

Table 16: Data.User - Exported Access Programs

Name	In	Out	Exceptions
createUser	rec: UserRecord	UserID	UserError
getUserByID	id: UserID	UserRecord	NotFoundError
getUserByEmail	email: String	UserRecord	NotFoundError
updateUser	id: UserID, rec: UserRecord	bool	UserError, NotFoundError

19.4 Semantics

19.4.1 State Variables

- `userTable: Map[UserID, UserRecord]`

19.4.2 Environment Variables

- Persistent user database with appropriate access controls.

19.4.3 Assumptions

- Email addresses are unique per user.

19.4.4 Access Routine Semantics

`createUser(rec)`:

- transition: inserts new user record; hashes password fields before storage.
- output: generated `UserID`.
- exception: `UserError` if constraints violated or write fails.

`getUserByID(id)`:

- output: user record with `id`.
- exception: `NotFoundError` if no such user.

`getUserByEmail(email)`:

- output: user record whose email matches `email`.
- exception: `NotFoundError` if not found.

`updateUser(id, rec)`:

- transition: updates record for `id`.
- output: `true` on success.
- exception: `NotFoundError` if `id` unknown; `UserError` on failure.

19.4.5 Local Functions

- `validateUser(rec)` – checks email format, password policy etc.

20 MIS of Data.Project

20.1 Module

Data.Project (Project and Reconstruction Session Store)

This module records reconstruction projects, including project metadata, associated fragments, and saved reconstruction sessions.

20.2 Uses

- `Data.User` – to associate projects with owners or collaborators.
- `Data.FragmentStore`.

20.3 Syntax

20.3.1 Exported Constants

- `MAX_PROJECT_NAME_LENGTH`

20.3.2 Exported Access Programs

Table 17: Data.Project - Exported Access Programs

Name	In	Out	Exceptions
`createProject`	`rec: ProjectRecord`	`ProjectID`	`ProjectError`
`getProject`	`id: ProjectID`	`ProjectRecord`	`NotFoundError`
`updateProject`	`id: ProjectID, rec: ProjectRecord`	`bool`	`ProjectError, NotFoundError`
`listProjectsForUser`	`user: UserID`	`Sequence[ProjectSummary]`	`ProjectError`

20.4 Semantics

20.4.1 State Variables

- `projectTable: Map[ProjectID, ProjectRecord]`

20.4.2 Environment Variables

- Persistent project database.

20.4.3 Assumptions

- Each project belongs to at least one user.

20.4.4 Access Routine Semantics

`createProject(rec)`:

- transition: inserts new row in `projectTable`.
- output: created `ProjectID`.
- exception: `ProjectError` on constraint violation or failure.

`getProject(id)`:

- output: project record matching `id`.
- exception: `NotFoundError` if none exists.

`updateProject(id, rec)`:

- transition: updates stored record.
- output: `true` on success.
- exception: `NotFoundError` or `ProjectError`.

`listProjectsForUser(user)`:

- output: summaries of all projects owned by or shared with `user`.
- exception: `ProjectError` if query fails.

20.4.5 Local Functions

- `validateProject(rec)` – checks required fields and name length.

21 MIS of ML.EdgeMatch

21.1 Module

ML.EdgeMatch (Edge Pattern Matching Model)

This module provides model-based similarity estimates based on fragment edge contours and shapes.

21.2 Uses

- `HW.ImageStorage` – to retrieve normalized images.

21.3 Syntax

21.3.1 Exported Constants

- `EDGE_SIGNATURE_DIM`

21.3.2 Exported Access Programs

Table 18: ML.EdgeMatch - Exported Access Programs

Name	In	Out	Exceptions
`computeEdgeSignature`	`id: ImageID`	`EdgeSignature`	`ModelError`
`scoreEdgeSimilarity`	`a: EdgeSignature, b: EdgeSignature`	`float`	`ModelError`

21.4 Semantics

21.4.1 State Variables

- Optional cached model weights and configuration.

21.4.2 Environment Variables

- Model runtime (for example, PyTorch or TensorFlow backend).

21.4.3 Assumptions

- Input images have been preprocessed and normalized by `Svc.ProcOrch`.

21.4.4 Access Routine Semantics

`computeEdgeSignature(id)`:

- output: fixed-length vector representing edge features for the image.
- exception: `ModelError` if image cannot be processed.

`scoreEdgeSimilarity(a, b)`:

- output: similarity score in $[0, 1]$ reflecting how well edges match.
- exception: `ModelError` if signatures invalid or incompatible.

21.4.5 Local Functions

- `extractContours(img)` – local function to compute edge features.

22 MIS of ML.Damage

22.1 Module

ML.Damage (Damage Pattern Recognition Model)

This module characterizes damage patterns (tears, stains, burn marks) and computes similarity between fragments based on shared damage signatures.

22.2 Uses

- `HW.ImageStorage`.

22.3 Syntax

22.3.1 Exported Constants

- `DAMAGE_SIGNATURE_DIM`

22.3.2 Exported Access Programs

Table 19: ML.Damage - Exported Access Programs

Name	In	Out	Exceptions
`computeDamageSignature`	`id: ImageID`	`DamageSignature`	`ModelError`
`scoreDamageSimilarity`	`a: DamageSignature, b: DamageSignature`	`float`	`ModelError`

22.4 Semantics

22.4.1 State Variables

- Optional cached model configuration.

22.4.2 Environment Variables

- Same model runtime as other ML modules.

22.4.3 Assumptions

- Damage features can be extracted from normalized images.

22.4.4 Access Routine Semantics

`computeDamageSignature(id)`:

- output: feature vector encoding damage pattern characteristics.
- exception: `ModelError` on failure.

`scoreDamageSimilarity(a, b)`:

- output: similarity score in $[0, 1]$.
- exception: `ModelError` if input invalid.

22.4.5 Local Functions

- `segmentDamageRegions(img)` – internal helper for feature extraction.

23 MIS of ML.ScriptClass

23.1 Module

ML.ScriptClass (Script Classification Model)

This module classifies the script type of a fragment (for example, Gupta Brāhmī, Tibetan Uchen, Chinese) and returns confidence scores.

23.2 Uses

- `HW.ImageStorage`.

23.3 Syntax

23.3.1 Exported Constants

- `SUPPORTED_SCRIPTS: Sequence[String]`

23.3.2 Exported Access Programs

Table 20: ML.ScriptClass - Exported Access Programs

Name	In	Out	Exceptions
`classifyScript`	`id: ImageID`	`ScriptPrediction`	`ModelError`

23.4 Semantics

23.4.1 State Variables

- Model weights for script classification.

23.4.2 Environment Variables

- Model runtime and hardware accelerators (if available).

23.4.3 Assumptions

- Input image includes at least some legible text.

23.4.4 Access Routine Semantics

`classifyScript(id)`:

- output: `ScriptPrediction` containing predicted script label and confidence.
- exception: `ModelError` if classification fails or confidence undefined.

23.4.5 Local Functions

- `preprocessForScript(img)` – applies script-specific preprocessing pipeline.

24 MIS of ML.OCR

24.1 Module

ML.OCR (Sanskrit OCR and Transcription Model)

This module performs OCR on manuscript fragments, returning transcribed text and character-level confidence scores.

24.2 Uses

- `HW.ImageStorage`.

24.3 Syntax

24.3.1 Exported Constants

- `MIN_ACCEPTABLE_CONFIDENCE` – used to flag low-confidence characters.

24.3.2 Exported Access Programs

Table 21: ML.OCR - Exported Access Programs

Name	In	Out	Exceptions
`runOCR`	`id: ImageID`	`OCRResult`	`ModelError`
`estimateAccuracy`	`gt: String, pred: String`	`float`	`None`

24.4 Semantics

24.4.1 State Variables

- OCR model weights and configuration.

24.4.2 Environment Variables

- Model runtime and any external OCR library.

24.4.3 Assumptions

- Input images obey preprocessing constraints (orientation, resolution).

24.4.4 Access Routine Semantics

`runOCR(id)`:

- output: `OCRResult` containing recognized text and confidence scores.
- exception: `ModelError` if inference fails.

`estimateAccuracy(gt, pred)`:

- output: approximate accuracy value, for example using character-level Levenshtein-based metric.
- exception: none.

24.4.5 Local Functions

- `normalizeText(str)` – harmonizes text for fair comparison.

25 MIS of ML.TextSim

25.1 Module

ML.TextSim (Textual Content Similarity Model)

This module computes semantic similarity between text segments derived from OCR or manual transcription.

25.2 Uses

- `ML.OCR` – as a source of text.

25.3 Syntax

25.3.1 Exported Constants

- `EMBEDDING_DIM`

25.3.2 Exported Access Programs

Table 22: ML.TextSim - Exported Access Programs

Name	In	Out	Exceptions
`embedText`	`text: String`	`TextEmbedding`	`ModelError`
`scoreTextSimilarity`	`a: TextEmbedding, b: TextEmbedding`	`float`	`ModelError`

25.4 Semantics

25.4.1 State Variables

- Embedding model weights.

25.4.2 Environment Variables

- Model runtime.

25.4.3 Assumptions

- Input text is normalized (script, encoding).

25.4.4 Access Routine Semantics

`embedText(text)`:

- output: fixed-length embedding vector.
- exception: `ModelError` if model fails.

`scoreTextSimilarity(a, b)`:

- output: similarity score in $[0, 1]$.
- exception: `ModelError` if inputs invalid.

25.4.5 Local Functions

- `cosineSimilarity(a, b)` – core similarity computation.

26 MIS of Util.Error

26.1 Module

Util.Error (Error Type and Exception Utilities)

This module defines standard error types and helpers for constructing and handling errors in a consistent way across modules.

26.2 Uses

- None at the architectural level; referenced by many other modules.

26.3 Syntax

26.3.1 Exported Constants

- Standard error codes or identifiers (for example, `ERR_NOT_FOUND`, `ERR_VALIDATION`).

26.3.2 Exported Access Programs

Table 23: Util.Error - Exported Access Programs

Name	In	Out	Exceptions
`buildError`	`code: String, msg: String`	`ErrorObject`	`None`
`toClientMessage`	`err: ErrorObject`	`String`	`None`

26.4 Semantics

26.4.1 State Variables

- None.

26.4.2 Environment Variables

- None.

26.4.3 Assumptions

- All modules reference the same canonical error types.

26.4.4 Access Routine Semantics

`buildError(code, msg)`:

- output: structured error object capturing code and message.
- exception: none.

`toClientMessage(err)`:

- output: safe message for display at UI or API boundary.
- exception: none.

27 MIS of Util.Logging

27.1 Module

Util.Logging (Logging Utility)
This module provides structured logging for events, errors, and performance metrics.

27.2 Uses

- Underlying logging framework or standard output.

27.3 Syntax

27.3.1 Exported Constants

- Log levels such as `INFO`, `WARN`, `ERROR`.

27.3.2 Exported Access Programs

Table 24: Util.Logging - Exported Access Programs

Name	In	Out	Exceptions
`logInfo`	`msg: String`	none	None
`logWarn`	`msg: String`	none	None
`logError`	`msg: String, err: ErrorObject`	none	None

27.4 Semantics

27.4.1 State Variables

- Optional runtime configuration (log level, sinks).

27.4.2 Environment Variables

- File system, console, or external logging service.

27.4.3 Assumptions

- Logging operations are non-fatal and do not throw.

27.4.4 Access Routine Semantics

`logInfo(msg)`:

- output: none; writes informational log entry.
- exception: none.

`logWarn(msg)`:

- output: none; writes warning log entry.
- exception: none.

`logError(msg, err)`:

- output: none; writes error entry including details from `err`.
- exception: none.

28 MIS of Util.Config

28.1 Module

Util.Config (Configuration Management)

This module provides read-only access to configuration values such as database connection strings, model paths, and feature flags.

28.2 Uses

- Environment variables or configuration files.

28.3 Syntax

28.3.1 Exported Constants

- None fixed; keys are used to look up values.

28.3.2 Exported Access Programs

Table 25: Util.Config - Exported Access Programs

Name	In	Out	Exceptions
`getValue`	`key: String`	`String`	`ConfigError`
`getBool`	`key: String`	`bool`	`ConfigError`

28.4 Semantics

28.4.1 State Variables

- `configCache: Map[String, String]`

28.4.2 Environment Variables

- Config file or environment variable store.

28.4.3 Assumptions

- Configuration is loaded at startup and remains static during runtime.

28.4.4 Access Routine Semantics

`getValue(key)`:

- output: configuration value associated with `key`.
- exception: `ConfigError` if key missing.

`getBool(key)`:

- output: parsed boolean value.
- exception: `ConfigError` if key missing or value not parseable.

29 MIS of Util.TestStub

29.1 Module

Util.TestStub (Testing Stubs and Fakes)

This module provides stub implementations of selected services and ML components for use in automated tests.

29.2 Uses

- Core service and ML interfaces for type compatibility.

29.3 Syntax

29.3.1 Exported Constants

- None.

29.3.2 Exported Access Programs

Table 26: Util.TestStub - Exported Access Programs

Name	In	Out	Exceptions
`getFixedOCRStub`	`none`	`ML.OCR-like object`	`None`
`getFixedMatchStub`	`none`	`Svc.MatchOrch-like object`	`None`

29.4 Semantics

29.4.1 State Variables

- Predefined responses used for deterministic tests.

29.4.2 Environment Variables

- None.

29.4.3 Assumptions

- Stubs are only used in test environments.

29.4.4 Access Routine Semantics

`getFixedOCRStub()`:

- output: object implementing `runOCR` that returns predictable results for known inputs.
- exception: none.

`getFixedMatchStub()`:

- output: object implementing `generateMatches` with fixed candidate lists.
- exception: none.

29.4.5 Local Functions

- `loadStubFixtures()` – loads predefined responses from static files.

References

Team Sanskrit Ciphers. Hazard analysis document. https://github.com/DylanG5/sanskrit-cipher/blob/43484da11e4fb0f879f7bbf2377c21900e5b9838/docs/HazardAnalysis/HazardAnalysis.pdf, 2025a. Identifies potential hazards, risks, and mitigation strategies.

Team Sanskrit Ciphers. Software requirements specification (srs). https://github.com/DylanG5/sanskrit-ciphers-requirements/blob/9938018b682709551d0a6248a0a9f6e5794112ee/index.pdf, 2025b. Specifies functional and non-functional requirements for the system.

Appendix — Reflection

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

1. What went well while writing this deliverable?

   One aspect that went particularly well was identifying the appropriate notation and determining which modules needed to be included in the design. Our team communicated effectively throughout this process, which helped us maintain a shared understanding of the system and its structure. Because we already had a strong grasp of the project vision, it was relatively easy to map our ideas into a formal design representation. As a result, we were able to present a clear and coherent picture of the system architecture that reflects both the requirements and the intended functionality.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   The main challenge we encountered was thinking through the lower-level implementation details. At this stage, we have not yet progressed far into the actual coding, so some of the design decisions, especially those related to specific algorithms, data structures, and system interactions required educated assumptions rather than concrete experience. We addressed this by focusing on higher-level abstractions and deferring detailed implementation decisions until we gain more clarity during development. This allowed us to complete the deliverable while still leaving room for necessary adjustments later.

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

   Many of our design decisions, especially those involving the user interface, were directly influenced by conversations with our supervisor, Dr. Shayne. Since he does not come from a technical background, he primarily expressed his needs in terms of how he wanted to interact with the system. This naturally shaped the UI and the workflow we designed for scholars. Additionally, the inclusion of segmentation-based machine learning came from his explicit request; he emphasized that automated assistance in identifying and matching fragments would significantly improve his research workflow. These inputs helped anchor our design choices in real user needs.

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?

   Very few changes were needed to our previous documents. Because this is a pre-approved project with a well-defined problem and scope, many of the requirements and expectations were already laid out for us in the project description. As a result, our earlier documents, such as the SRS (Ciphers, 2025b) and Hazard Analysis (Ciphers,

2025a), aligned naturally with the design work we did here. We did not need to revisit or adjust them significantly, beyond minor wording refinements and consistency updates.

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)

   With unlimited resources, one of the most impactful improvements would be investing in more powerful AI infrastructure, such as GPUs dedicated to training and evaluating models. This would allow us to train new segmentation or matching models more frequently and at a much larger scale. We would also aim to build a more advanced fragment-matching pipeline with higher accuracy and more robust behaviour across different manuscript types. Additionally, we would elevate the quality of the UI significantly, improving performance, adding more interactive tools, and making the interface cleaner and more intuitive. Overall, the entire system could operate at a higher level of precision, speed, and sophistication.

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)

   We did not explore many alternative designs because, from early discussions, the entire team already shared a clear mental model of how the system should operate. Our frequent conversations kept us aligned on the major components and workflow, so we naturally converged on a single coherent design. Although this meant we did not formally evaluate multiple competing solutions, the advantage was that we were able to focus our efforts on developing one strong, detailed design rather than splitting time considering options that did not align with the client's expectations. Given the clarity of the problem and the constraints provided, the chosen design was the most straightforward and appropriate path forward.