Module Interface Specification for Software Engineering

Team 5, GradSight
Willie Pai
Hammad Pathan
Wajdan Faheen
Henushan Balachandran
Zahin Hossain

April 1, 2025

1 Revision History

Date	Version	Notes
January 17, 2025	1.0	Rev 0

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/PaisWillie/Digital-Composite/blob/main/docs/SRS-Volere/SRS.pdf

Contents

3 Introduction

The following document details the Module Interface Specifications for Digital Composite

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/PaisWillie/Digital-Composite/tree/main.

4 Notation

The structure of the MIS for modules follows standard conventions from software engineering design, specifically based on information hiding principles and modular interface specifications. The templates used are adapted from classical software specification methods. The mathematical notation follows commonly accepted formal methods for data types, logical rules, and function semantics.

For instance, the symbol `:=` is used to denote multiple assignment, and conditional expressions follow the structure:

$$(c_1 \Rightarrow r_1 \mid c_2 \Rightarrow r_2 \mid \cdots \mid c_n \Rightarrow r_n)$$

The table below summarizes the primitive data types used throughout this specification:

Data Type	Notation	Description
character	`char`	A single symbol or digit
integer	$\mathbb{Z}$	A whole number in $(-\infty, \infty)$
natural number	$\mathbb{N}$	A positive integer in $[1, \infty)$
real number	$\mathbb{R}$	Any real number in $(-\infty, \infty)$

In addition to primitive types, the specification uses several derived data types:

- **Sequences**: Ordered lists of elements of the same type
- **Strings**: Sequences of characters
- **Tuples**: Ordered collections of elements, potentially of different types

Functions are described using their input and output types. Local functions are specified using a signature followed by their semantics in English or mathematical form.

5 Module Decomposition Table

The modules for GradSight are organized according to the principles of information hiding and responsibility-driven design. The decomposition separates system infrastructure, processing logic, and presentation layers into three abstraction levels. Each module is labeled with an identifier (M1–M7) and follows the hierarchy introduced in the Module Guide (MG).

Table 1: Revised Module Decomposition Table

Level 1	Level 2 (Modules)
Hardware-Hiding	M1: Cloud — abstracts interaction with AWS S3, Lambda, and DynamoDB
Behaviour-Hiding	M2: Input — parses and validates uploaded files M3: Upload — uploads images and metadata to cloud storage M4: OCR — extracts names and coordinates from composite images M5: Output — stores and queries parsed metadata from DynamoDB M6: UI Parsing — handles fallback parsing and error display logic
Software Decision	M7: GUI — provides the touch-based interface for navigation and search

6 Module M1 – Cloud

Module Responsibilities

This module abstracts the cloud infrastructure of the system. It handles all interaction with Amazon Web Services (AWS), including S3 for image storage, Lambda for triggering OCR pipelines, and DynamoDB for persistent metadata storage.

Secrets

- The structure and credentials for accessing AWS services - Cloud function triggers and permissions - DynamoDB table and S3 bucket configuration

Services

- Upload image files to S3 - Retrieve stored image files - Store parsed OCR metadata in DynamoDB - Fetch parsed metadata by user query

Information Hiding

This module hides the implementation details of AWS interaction, such as authentication, request handling, and region setup.

Assumptions

- AWS credentials are stored securely in the environment - DynamoDB, Lambda, and S3 services are already configured - Lambda function expects a valid image path from S3

State Variables

- `bucketName: String` — Identifier for the S3 bucket

- `ddbTableName: String` — Name of the DynamoDB table used for storing parsed metadata

Environment Variables

- `AWS_REGION: String` — Region where services are hosted
- `AWS_CREDENTIALS: JSON` — Keys and secret tokens for service access

Uses

None (Cloud is the hardware-hiding module — other modules use it, but it does not call others directly.)

Interface Syntax

- `uploadToS3(image: BinaryImage) → url: String`
- `triggerOCRLambda(s3Url: String) → lambdaStatus: Integer`
- `storeMetadata(data: JSON) → success: Boolean`
- `fetchMetadataByQuery(query: JSON) → result: JSON[]`

Semantics

uploadToS3

- **Transition:** Uploads the given image to the S3 bucket using a secure HTTP PUT request
- **Output:** Returns the public URL of the uploaded image
- **Exceptions:** S3UploadError if upload fails

triggerOCRLambda

- **Transition:** Calls AWS Lambda to parse the image at the given S3 path
- **Output:** Returns HTTP status code (200 = success)
- **Exceptions:** LambdaInvocationError if function cannot be triggered

storeMetadata

- **Transition:** Saves parsed data (name, program, coordinates, etc.) to DynamoDB
- **Output:** Returns `true` if write is acknowledged

- **Exceptions:** DDBWriteError if write fails

fetchMetadataByQuery

- **Transition:** Queries the DynamoDB table for entries matching user query (e.g., year = 2024, name = Zahin)
- **Output:** Returns an array of matching JSON objects (name, coordinates, etc.)
- **Exceptions:** DDBReadError or MalformedQueryError

7 Module M2 – Input

Module Responsibilities

This module is responsible for validating the structure and format of composite images before they are uploaded. It ensures that the uploaded file meets size, resolution, and file-type constraints as outlined in the SRS.

Secrets

- Validation thresholds (e.g., max file size, supported file types) - Image resolution rules specific to OCR accuracy requirements

Services

- Accepts image files from the Admin UI (M6) - Performs pre-upload validation checks - Reports formatted validation results to the Upload module (M3)

Information Hiding

This module hides the internal rules used to verify the image's acceptability for parsing and storage.

Assumptions

- Input is a single image file in a supported format (.png or .jpeg) - File is not corrupted and can be decoded - File is being uploaded through the Admin interface

State Variables

- `acceptedTypes: List<String>` — Allowed file extensions (e.g., `[‘‘png’’, ‘‘jpg’’]`)
- `maxFileSize: Integer` — Maximum file size in bytes
- `minResolution: Tuple<Integer, Integer>` — Minimum resolution (width, height)

Environment Variables

- `MAX_UPLOAD_SIZE: Integer` — System-wide upper limit on uploads

Uses

- M3: Upload Module (sends validated image forward)

Interface Syntax

- `validateImage(file: BinaryImage) → result: ValidationReport`
- `getAcceptedTypes() → List<String>`
- `getMaxFileSize() → Integer`

Semantics

validateImage

- **Transition:** Verifies that the image meets file type, size, and resolution constraints
- **Output:** Returns a `ValidationReport` object containing status, error messages (if any), and extracted metadata
- **Exceptions:** FileFormatError, FileTooLargeError, ResolutionTooLowError

getAcceptedTypes

- **Transition:** Returns a list of supported file extensions
- **Output:** List of strings (e.g., [“png”, “jpeg”])

getMaxFileSize

- **Transition:** Fetches the current maximum file size threshold
- **Output:** Integer value in bytes

8 Module M3 – Upload

Module Responsibilities

This module handles the secure upload of validated image files to the cloud storage system (S3) via M1. It ensures that data is correctly packaged and transferred, handles upload confirmations, and triggers cloud-side processing via Lambda.

Secrets

- The method of authentication used for upload - Retry logic and error thresholds for failed transfers - Upload buffering and compression strategy

Services

- Accepts validated files from M2 (Input) - Uploads images to M1 (Cloud) - Initiates post-upload processes (e.g., OCR trigger) - Returns upload status and error details (if any)

Information Hiding

The module hides details about how images are encoded, compressed, and pushed to the cloud backend.

Assumptions

- The image passed from M2 has already been validated - M1 services (S3 + Lambda) are available and authorized - This module is invoked only through the Admin path in the UI

State Variables

- `uploadBuffer: BinaryImage` — Holds the image during processing
- `lastUploadURL: String` — Cloud path returned by S3 after successful upload

Environment Variables

- `UPLOAD_TIMEOUT`: Integer – Max allowable upload duration (ms)
- `RETRY_COUNT: Integer` – Number of times to retry a failed upload

Uses

- M1: Cloud Module (for upload + OCR triggering) - M2: Input Module (receives validated files)

Interface Syntax

- `initiateUpload(file: BinaryImage) → status: UploadReport`
- `getLastUploadURL() → String`
- `retryUpload() → Boolean`

Semantics

initiateUpload

- **Transition:** Sends image to M1 for upload to S3, then invokes Lambda OCR function
- **Output:** Returns an `UploadReport` object with upload success/failure, timestamp, and S3 URL
- **Exceptions:** UploadTimeoutError, InvalidCredentialsError, CloudServiceUnavailableError

getLastUploadURL

- **Transition:** Retrieves the cloud URL of the most recently uploaded file
- **Output:** String (URL)

retryUpload

- **Transition:** Retries the last failed upload using the stored buffer
- **Output:** Returns `true` if retry succeeded, `false` otherwise
- **Exceptions:** MaxRetryExceededError

9 Module M4 – OCR

Module Responsibilities

This module performs OCR on the uploaded composite image and extracts student names along with their bounding box coordinates. It transforms visual data into structured metadata that can be stored in the database and rendered in the user interface.

Secrets

- OCR engine configuration (e.g., AWS Textract, Tesseract) - Heuristics for bounding box filtering - Preprocessing techniques for image enhancement

Services

- Accepts an image URL from the Upload module (M3) - Applies OCR to extract text and location data - Returns results to the Output module (M5) - Raises errors for unsupported image structures

Information Hiding

This module hides the internal OCR pipeline and pre-processing steps such as image sharpening, binarization, and margin cropping.

Assumptions

- The input image is accessible via a valid S3 URL - The image contains rows/columns of names with consistent spacing - Text is horizontally aligned for best OCR performance

State Variables

- `parsedMetadata: JSON` — Stores extracted text and position data from the latest OCR run
- `imageURL: String` — Holds the current image being processed

Environment Variables

- `OCR_ENGINE: String` — Selected OCR tool (e.g., "textract" or "tesseract")
- `MIN_CONFIDENCE: Float` — Confidence threshold for valid text extraction

Uses

- M1: Cloud Module (to retrieve image from S3) - M3: Upload Module (passes S3 URL to M4) - M5: Output Module (receives parsed metadata)

Interface Syntax

- `runOCR(imageURL: String) → metadata: JSON`
- `getParsedMetadata() → JSON`
- `clearBuffer() → Boolean`

Semantics

runOCR

- **Transition:** Downloads the image from S3 and applies OCR using the configured engine. Extracts names and coordinates and stores the result in `parsedMetadata`.
- **Output:** JSON object in the format: `[ { "name": String, "x": Int, "y": Int }, ... ]`

- **Exceptions:** ImageNotFoundError, OCRFailureError, LowConfidenceWarning

getParsedMetadata

- **Transition:** Returns the latest parsed metadata
- **Output:** JSON object with OCR results

clearBuffer

- **Transition:** Empties internal storage for parsedMetadata and imageURL
- **Output:** `true` on success

10 Module M5 – Output

Module Responsibilities

This module handles persistent storage of parsed composite data and processes queries for retrieving student name information. It provides the backend functionality for the search and display features seen in the GUI (M7).

Secrets

- The database schema and indexing strategy - Optimization strategies for filtering and querying - Any redundancy or fallback keys in data storage

Services

- Stores OCR results provided by M4 - Accepts user-defined queries from M6 (UI Parsing) or M7 (GUI) - Retrieves relevant entries from the database - Returns data in a structured, displayable format

Information Hiding

This module hides the structure of the database, indexing techniques, and optimization details.

Assumptions

- The database is pre-configured and connected - Inputs passed from OCR and GUI are valid and in correct format - The data schema is consistent with the metadata generated by OCR

State Variables

- `dbClient: DDBClient` — DynamoDB interface object
- `lastQueryResult: JSON[]` — Stores the most recent query output for caching

Environment Variables

- `DB_TABLE_NAME: String` — Target table for storing and retrieving OCR metadata
- `MAX_QUERY_LIMIT: Integer` — Cap on number of entries returned per query

Uses

- M1: Cloud Module (interfaces with DynamoDB) - M4: OCR Module (receives parsed data)
- M6: UI Parsing Module (fetches name data)

Interface Syntax

- `storeData(metadata: JSON) → success: Boolean`
- `queryByField(field: String, value: String) → results: JSON[]`
- `getLastQueryResult() → JSON[]`

Semantics

storeData

- **Transition:** Inserts the provided JSON metadata (name, x/y position, year, program, etc.) into the DynamoDB table
- **Output:** Returns `true` on successful write
- **Exceptions:** WriteFailedError, SchemaMismatchError

queryByField

- **Transition:** Queries the database for records where `field = value`; stores the result in `lastQueryResult`
- **Output:** Array of matching entries in JSON
- **Exceptions:** FieldNotFoundError, QueryLimitExceededError

getLastQueryResult

- **Transition:** Retrieves the most recently stored query result
- **Output:** JSON array of composite data

11 Module M6 – UI Parsing

Module Responsibilities

This module manages user interface-level parsing logic, including collecting manually entered names when OCR fails and formatting data for display. It serves as an intermediary between the backend data (from M5) and the visual components (in M7).

Secrets

- Admin input validation logic - Fallback handling strategy and overwrite rules - Mapping of metadata fields to UI labels

Services

- Receives fallback name inputs from the Admin UI - Combines OCR results with manually entered data - Formats result sets for display in the GUI - Sends structured data to M7 for rendering

Information Hiding

Hides the logic used to reconcile OCR vs manual inputs, formatting rules, and display metadata preparation.

Assumptions

- The OCR module may fail to parse certain names - Admin input is assumed to be valid and minimal in volume - The GUI expects fully structured metadata

State Variables

- `fallbackEntries: JSON[]` — Stores admin-entered name/location data
- `mergedResults: JSON[]` — Combines fallback and OCR results

Environment Variables

- `MAX_FALLBACK_ENTRIES: Integer` — Cap on number of names that can be manually added

Uses

- M4: OCR Module (receives original parsed results) - M5: Output Module (fetches DB entries) - M7: GUI Module (sends formatted data for display)

Interface Syntax

- `submitFallbackEntry(data: JSON) → Boolean`
- `mergeAndFormatData(ocr: JSON[], fallback: JSON[]) → JSON[]`
- `getFormattedOutput() → JSON[]`

Semantics

submitFallbackEntry

- **Transition:** Adds a new fallback entry (name + coordinates) to `fallbackEntries`
- **Output:** `true` if entry stored successfully
- **Exceptions:** FallbackLimitExceededError

mergeAndFormatData

- **Transition:** Combines `ocr` and `fallback` data into a unified list, resolving duplicate names if needed
- **Output:** Formatted JSON array ready for UI rendering

getFormattedOutput

- **Transition:** Returns the final data array for GUI display
- **Output:** JSON array containing names, coordinates, and optional metadata

12 Module M7 – Graphical User Interface

Module

Provides the interactive user interface for GradSight. Displays the Admin and User modes, handles navigation, and renders composite search and view results. Interfaces with backend modules to fetch and display information.

Uses

- M5: Output Module (composite metadata)
- M6: UI Parsing Module (fallback + formatted metadata)

Syntax

Exported Constants

None

Exported Access Programs

Name	In	Out	Exceptions
renderHomeScreen	None	Void	RenderFailureError
renderSearchResults	data: JSON[]	Void	DataFormatError
getUserSelection	None	SelectionData	InputCaptureError

Semantics

State Variables

- currentView: String — Tracks the currently displayed screen (e.g., "home", "upload", "results")
- displayedData: JSON[] — Data currently shown to the user

Environment Variables

- SCREEN_SIZE: Tuple¡Integer, Integer¿ — Resolution of the touchscreen
- INPUT_MODE: String — Either "admin" or "user"

Assumptions

- The GUI is deployed on a touchscreen device with a modern browser
- Backend modules (M5, M6) return properly formatted JSON

Access Routine Semantics

renderHomeScreen()

- **transition:** Switches currentView to "home" and renders mode selection buttons
- **output:** None
- **exception:** RenderFailureError if rendering fails

renderSearchResults(data)

- **transition:** Sets currentView to "results", stores data in displayedData, and renders result list + highlights

- **output:** None
- **exception:** DataFormatError if data is missing required fields

getUserSelection()

- **transition:** Waits for user tap input on the screen and returns interaction details
- **output:** Object with coordinates and selected name/program
- **exception:** InputCaptureError if input device is unresponsive

Local Functions

- navigateTo(view: String): Transitions the view and triggers re-render
- highlightResult(result: JSON): Adds visual highlight to selected name on the composite

13 Enough to Build

The GradSight system is structured so that each module (M1–M7) encapsulates a distinct responsibility and can be implemented independently by a developer with access to this specification, the SRS, and the module guide. The interfaces defined in this document include state variables, environment dependencies, typed inputs and outputs, exceptions, and well-scoped responsibilities. This ensures that an independent developer does not require informal knowledge or direct communication with the original design team to begin development.

System Overview

GradSight is a modular web-based system designed to allow users to upload, parse, search, and view graduating student composite images. It includes a touch-enabled frontend (M7) and a cloud-integrated backend (M1–M6) with OCR, data processing, and fallback handling.

Implementation Entry Point

A developer may begin building from any module:

- **Frontend (M7)**: Requires M6 outputs to render UI. Can be built using sample data.
- **Cloud (M1)**: Operates independently to upload/retrieve files and handle Lambda/DynamoDB.
- **OCR (M4)**: Can be developed using any test image and outputs in JSON format.

Module Independence and Cohesion

Modules are organized by information hiding principles:

- **M1 (Cloud)** handles all AWS interaction — no other module needs cloud credentials or SDK logic.
- **M2–M3** are responsible for input validation and upload logic before cloud transfer.
- **M4** performs parsing using OCR with documented JSON output format.
- **M5** stores/query composite metadata, with full key-based search support.
- **M6** formats fallback data + OCR results for M7 to render.
- **M7** consumes only structured UI-ready data, rendering it without deep backend knowledge.

Data Formats and Contracts

All inter-module communication is done via defined JSON structures. Examples include:

- `OCR Output:` { name: "Zahin H", x: 102, y: 220 }
- `Fallback Entry:` { name: "Willie P", location: [140, 330] }
- `Query Result:` JSON[] of name/location objects returned to GUI

Integration Readiness

Each module contains:

- Clearly specified access routines with types and exceptions
- Defined state/environment variables
- Interface examples and transition semantics

This documentation enables new developers to build and test individual modules using mocks or provided formats, with no ambiguity about scope, responsibilities, or expected behavior.

14 Specification Formalization

This section provides a formal specification of the `runOCR()` access program from the OCR Module (M4). The purpose is to clearly define the function's behavior using a mathematical model that is precise and unambiguous.

Formalized Function: runOCR(imageURL)

Inputs

Let:

- $I \in \mathbb{S}$ be a valid S3 image URL such that the image is accessible and decodable
- $OCR_{engine} : \mathbb{S} \to \mathcal{M}$ be the OCR function that maps a string (URL) to metadata

Output

Let the output $M \in \mathcal{M}$ be a set of metadata tuples:

$$M = \{(n_i, x_i, y_i) \mid n_i \in \texttt{String}, x_i, y_i \in \mathbb{N}\}$$

Each tuple represents a name and its coordinates extracted from the image.

Preconditions

$$\texttt{imageURL} \in \mathbb{S} \land \texttt{reachable(imageURL)} = \texttt{true}$$

Postconditions

$$runOCR(I) = OCR_{engine}(I)$$

Exceptions

- `OCRFailureError` if $OCR_{engine}(I)$ fails or returns an empty result
- `ImageNotFoundError` if $\neg\texttt{reachable}(I)$

References

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:

1. What went well while writing this deliverable?

   - Everybody understood the structure of the design very well. We were all on the same page which let us complete this deliverable with confidence.

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

   - Some pain points we experienced with this deliverable were understanding how to break up and modularize our design components that we created before. We solved this by speaking with our TA and clarifying how we can divide them.

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

   - The design decision to use cloud stemmed from speaking with our client and other stakeholders due to the nature of the hosting availability at McMaster. For those decisions that did not stem, they came from previous experiences in how to setup the infrastructures.

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?

   - None, if they need to we will go back to change them after getting reviewed.

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)

- Some limitations of our solution are the security and reliability. Spend more time perfecting the modularization, as well as understanding the most secure ways to setup the architecture with help from professionals. While we are secure and reliable enough now, in terms of scaling we don't know.

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)

    - Not many other design solutions were considered. The only other design solution was to directly host a database on McMaster's Servers but this came with many complications as well as setting up meetings with busy individuals who were in charge of the departments. Due to how easy setting up cloud was in comparison, we went with that route.