Module Guide 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 13, 2025	1.0	Initial Revision
January 17, 2025	1.1	Final changes made according to rubric
March 31, 2025	1.2	Updated Document with changes according to feedback

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Software Engineering	Explanation of program name
UC	Unlikely Change

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in scientific computing, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules laid out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed. The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- **New project members**: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- **Maintainers**: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- **Designers**: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: Supporting additional hardware, such as new touchscreen displays or tablets, to ensure compatibility with future device upgrades and varying specifications.

AC2: Modifying input data formats to align with updated composite structures or metadata provided by Lifetouch.

AC3: Enhancing data processing workflows to adapt to different composite formats and improve OCR preprocessing accuracy.

AC4: Scaling the system to include graduation composites from additional faculties or departments across the university.

AC5: Updating the OCR model to accommodate higher-resolution composites or address accuracy improvements in text extraction.

AC6: Adding new user roles, such as external recruiters or event organizers, to expand the system's usability and access controls.

AC7: Redesigning the user interface to meet evolving accessibility standards and usability feedback for a better experience.

AC8: Extending the database schema to store new metadata, such as additional student achievements or career milestones.

AC9: Integrating the system with McMaster's existing alumni and event management platforms to streamline user engagement.

AC10: Adjusting backend processes to comply with updates to privacy policies, legal regulations, or internal university standards.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: The source of the input data will always be from predefined repositories, such as LifeTouch or McMaster databases.

UC2: The system will always display graduation composites on touchscreen interfaces and other devices as defined during development.

UC3: The primary goal of the system will remain focused on facilitating the browsing and searching of graduation composites for students, alumni, and staff.

UC4: The search functionality will always rely on a combination of Optical Character Recognition (OCR) data and metadata stored within the system.

UC5: The backend database architecture will remain relational (or NoSQL, as selected initially) to meet system scalability and storage requirements.

UC6: The system will only support English as the primary language for search and display functionalities.

UC7: The project will be exclusively deployed on McMaster's internal infrastructure, adhering to university IT policies and avoiding external hosting solutions.

UC8: The privacy and security protocols implemented will always follow GDPR and university policies, ensuring compliance with data handling standards.

5 Module Hierarchy

The GradSight system is modularized based on the principle of *information hiding* to ensure each module encapsulates specific responsibilities and secrets. The hierarchy is structured to support ease of development, future scalability, and independent module testing. The table below summarizes the system's decomposition:

Table 1: Module Hierarchy

Level 1	**Level 2**
Hardware-Hiding	M1: Cloud Module
Behaviour-Hiding M3: Data Upload Module M5: Output Storage Module	M2: Input Module M4: OCR Processing Module
Software Decision M7: Graphical User Interface Module	M6: UI Parsing Module

This hierarchy supports a *layered design* where each module at a higher level may depend on the services of lower-level modules. For example, the UI Parsing module (M6) depends on the Upload (M3), OCR (M4), and Output Storage (M5) modules to function correctly. The Cloud Module (M1) hides low-level cloud interaction details from all others.

6 Connection Between Requirements and Design

The design of GradSight ensures that every requirement identified in the Software Requirements Specification (SRS) is addressed by one or more modules in the system. Each requirement category has been mapped to modules based on their responsibilities and interactions. This section provides a brief overview of how the major functional and nonfunctional requirements are supported by the system design.

- **Functional Requirements (FR-1 to FR-5):** These requirements involve uploading, parsing, storing, and viewing composite data. Modules M2 (Input), M3 (Upload), M4 (OCR), M5 (Output), M6 (UI Parsing), and M7 (GUI) collectively fulfill these tasks. For example, M3 handles the file upload, M4 extracts text and position data, and M7 provides the user interface to search and view composites.
- **Nonfunctional Requirements (NFR-1 to NFR-3):** These are addressed through efficient cloud operations and performance optimization in modules M1 (Cloud), M4 (OCR), and M5 (Output). For instance, fast image parsing is ensured through optimized OCR and AWS Lambda functions (M4), and concurrent access is handled via scalable database storage in M5.
- **Security and Privacy (SEC, PRIV):** Input validation (M2, M6), secure data handling (M1, M5), and access control via the admin interface (M6, M7) ensure the system is GDPR-compliant and secure against unauthorized access. Logging of failed attempts and fallback mechanisms are also integrated.

- **Adaptability and Maintainability (ADP-1, SUP-1):** Modular decomposition was performed with future changes in mind. For example, M5 is designed to support new metadata fields without affecting other modules, and M1 allows new composite files to be added without downtime. Support documentation and system logs are provided through M6 and M1.

- **Traceability:** A full traceability matrix is presented in Section 8, which explicitly maps each requirement to its responsible module(s). Anticipated and unlikely changes are listed in Section 4, showing how likely evolution points were considered during decomposition.

7 Module Decomposition

Modules are decomposed according to the principle of information hiding. The Secrets field highlights the design decisions encapsulated by the module. The Services field summarizes the visible functionality provided. The Implemented By field explains what technology or component is responsible for implementing the module. The table in Section 5 provides a high-level overview of how modules are grouped.

7.1 7.1 Hardware-Hiding Modules

7.1.1 Cloud Module (M1)

Secrets: The SDKs, APIs, and configurations used to interact with AWS services (S3, Lambda, DynamoDB), including authentication, network protocols, and security settings.
Services: Abstracts cloud infrastructure, enabling the system to upload files, run serverless functions, and store metadata.
Implemented By: AWS SDK and Services
Type of Module: Abstract Object

7.2 7.2 Behaviour-Hiding Modules

7.2.1 Input Module (M2)

Secrets: Format expectations for the uploaded image and metadata (e.g., JSON keys, file types).
Services: Converts uploaded data into a standardized format suitable for processing and storage.
Implemented By: Back-End Logic
Type of Module: Library

7.2.2 Data Upload Module (M3)

Secrets: Details of the file upload logic, AWS S3 SDK usage, error handling and retries.
Services: Uploads validated image files and metadata to the cloud.
Implemented By: AWS S3 and Back-End Entity
Type of Module: Abstract Object

7.2.3 OCR Processing Module (M4)

Secrets: OCR library configuration (e.g., Tesseract or AWS Textract) and preprocessing rules.
Services: Extracts text and coordinates (e.g., names and center points) from uploaded images.
Implemented By: AWS Lambda
Type of Module: Library

7.2.4 Output Storage Module (M5)

Secrets: Schema design for metadata storage (e.g., keys, indexes, data types in DynamoDB).
Services: Stores parsed data in DynamoDB for later querying by the frontend or admin.
Implemented By: AWS DynamoDB
Type of Module: Abstract Data Type

7.3 Software Decision Modules

7.3.1 UI Parsing Module (M6)

Secrets: Logic for upload flow, fallback behavior, and file validation via the frontend.
Services: Validates, logs, and sends files for processing. Generates fallback data if needed.
Implemented By: Back-End Logic and AWS Lambda
Type of Module: Library

7.3.2 Graphical User Interface Module (M7)

Secrets: Visual layout rules, user interaction events, view state management.
Services: Provides a touch-based interface to view composites and search for student data.
Implemented By: React
Type of Module: Library

8 Traceability Matrix

This section provides traceability between the system's requirements and the corresponding modules in the design. It also identifies how anticipated changes may impact the modular structure of the system. This traceability is critical for ensuring design alignment with stakeholder needs, system safety, and future scalability.

8.1 8.1 Trace Between Requirements and Modules

Table 2: Trace Between Requirements and Modules

Requirement ID	**Modules**
FR-1: Admin uploads composite image via UI	M2, M3, M6, M7
FR-2: System parses composite image using OCR	M1, M4
FR-3: System stores metadata and position data	M1, M5
FR-4: User browses composite by year/program	M5, M6, M7
FR-5: User searches by name (OCR-based)	M4, M5, M7
NFR-1: System uptime 99% during business hours	M1
NFR-2: Image parsing takes 3 seconds per composite	M4, M5
NFR-3: System supports multiple concurrent viewers	M1, M5, M7
SEC-1: Unauthorized users cannot upload or modify composites	M2, M6
SEC-2: System logs failed uploads and parsing attempts	M6
PRIV-1: Personal data stored securely and compliant with GDPR	M1, M5
ADP-1: System supports addition of future composites without downtime	M1, M5
RR-1: Admin access panel available for data updates	M6, M7
SUP-1: System includes documentation for admin and IT support	M1, M6

8.2 8.2 Trace Between Anticipated Changes and Modules

Table 3: Trace Between Anticipated Changes and Modules

Anticipated Change (AC)	Modules Affected
AC1: Support for new display hardware (e.g., tablets)	M7
AC2: Change in composite image structure or metadata	M2
AC3: OCR model enhancement or replacement	M4
AC4: Adding additional faculties/programs	M1, M5
AC5: High-resolution image support for better accuracy	M4
AC6: New user roles and permissions	M2, M6, M7
AC7: UI updates based on accessibility feedback	M7
AC8: New metadata fields (e.g., awards, clubs)	M5
AC9: Integration with McMaster's alumni/event systems	M1
AC10: Privacy/legal policy updates (GDPR, ISO)	M1, M2, M5

9 Use Hierarchy Between Modules

This section describes the *use* relationship between modules. Module A **uses** Module B if the correct operation of A depends on services provided by B. This relationship implies a direct dependency in implementation, such as function calls or API usage.

Figure 1 illustrates the directed acyclic graph (DAG) of use relationships among modules. The design ensures that upper-layer modules are simpler and rely on well-encapsulated lower layers.

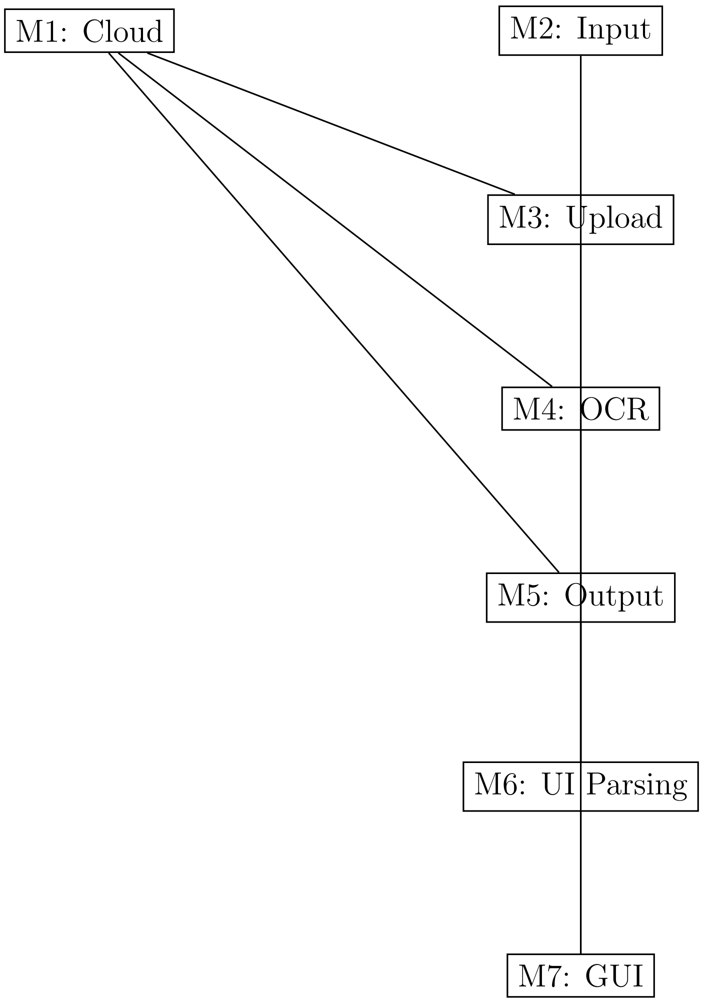


Figure 1: Use Hierarchy Among Modules

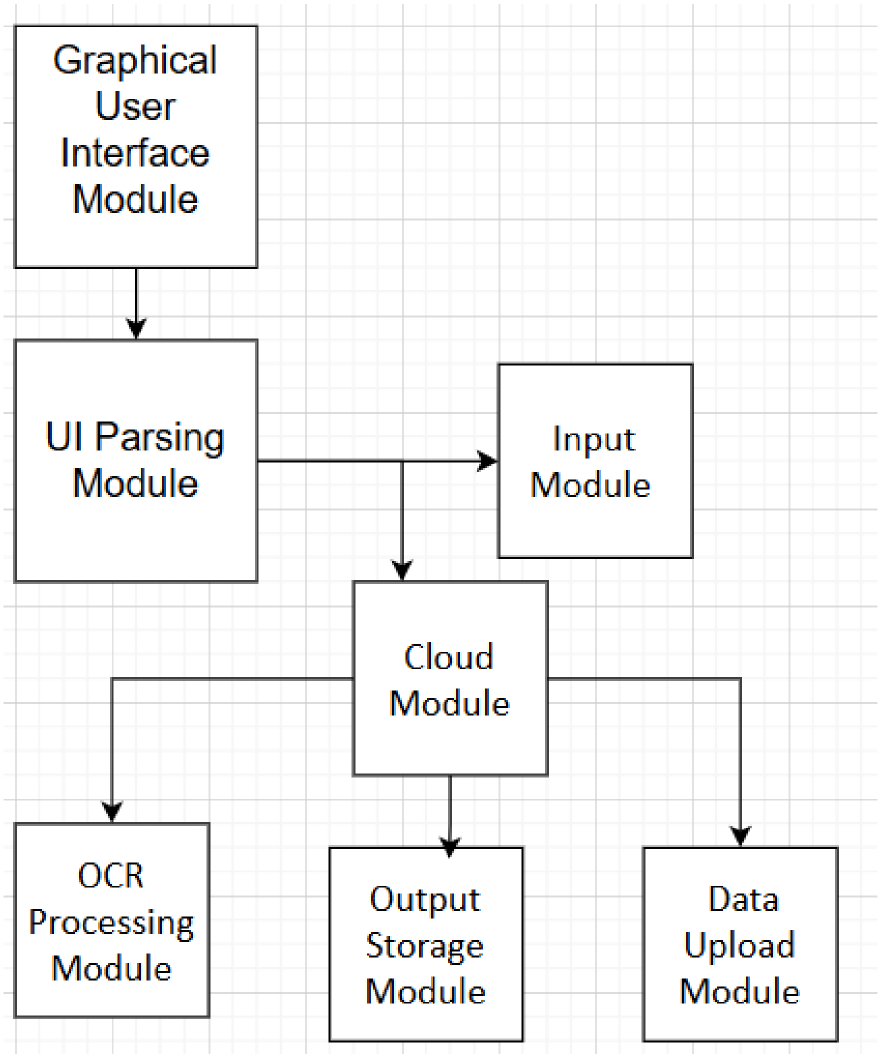


Figure 2: Use hierarchy among modules

10 User Interfaces

This section presents the graphical user interfaces (GUIs) designed for GradSight. The system is built for a touchscreen display, enabling users to upload, search, and view composite images interactively. The interfaces are powered primarily by Modules M6 (UI Parsing) and M7 (GUI), with support from backend modules for data retrieval and processing.

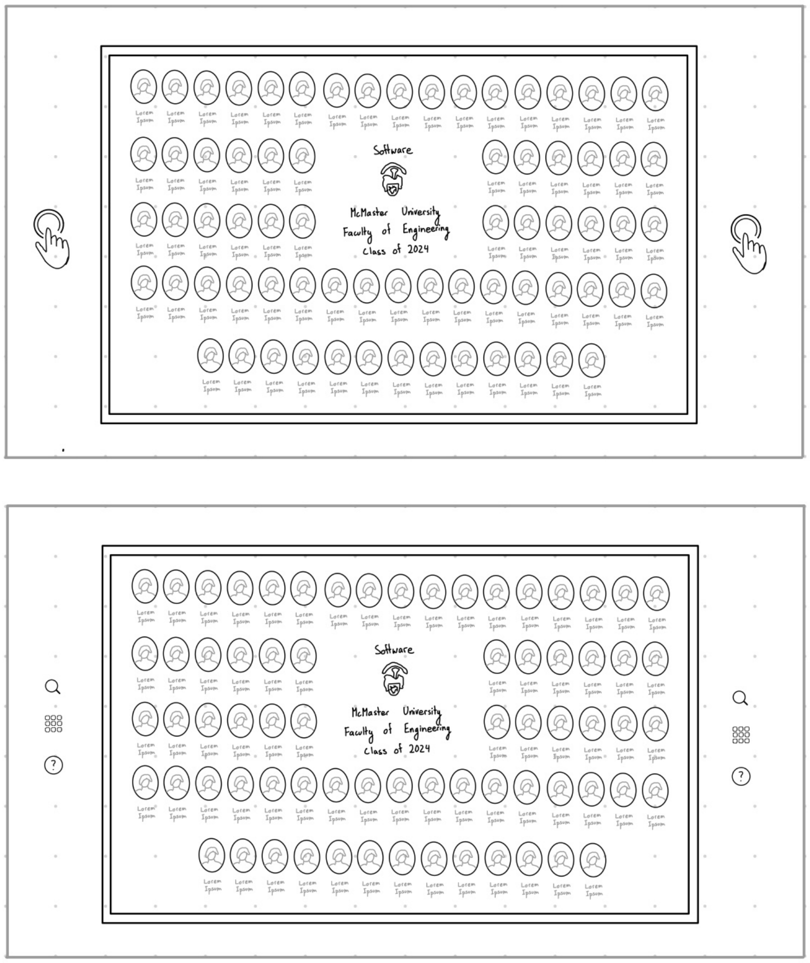
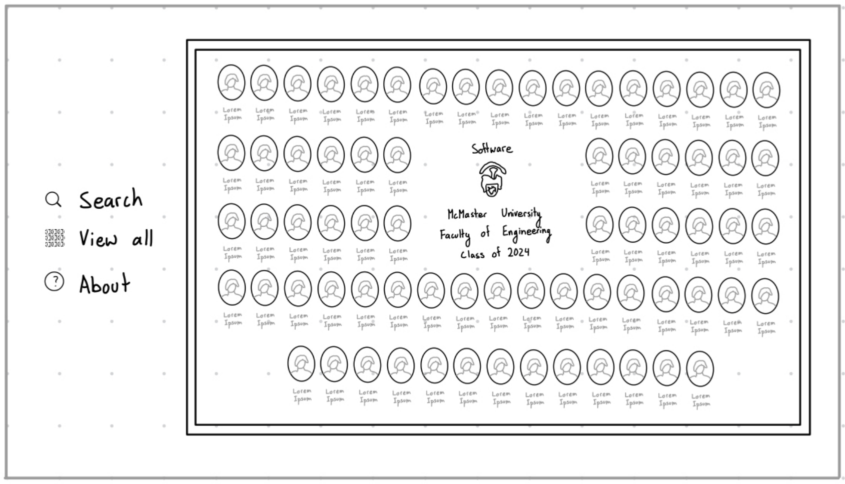


Figure 3: Home Page Interface — The user is presented with an initial menu to choose between Admin and User modes. This screen is managed by M7 (GUI) and determines system flow depending on user role.



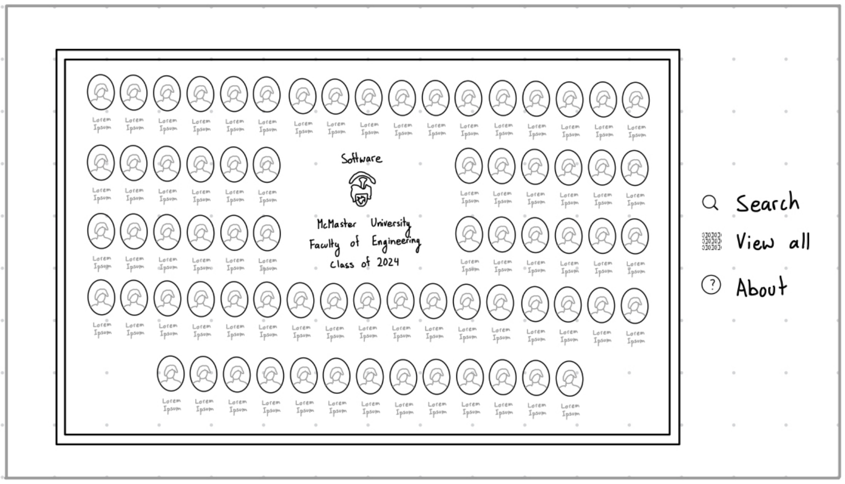


Figure 4: Admin Upload Interface — Admins can upload composite images through this interface. M6 handles file parsing, while M2 and M3 handle input validation and upload logic.

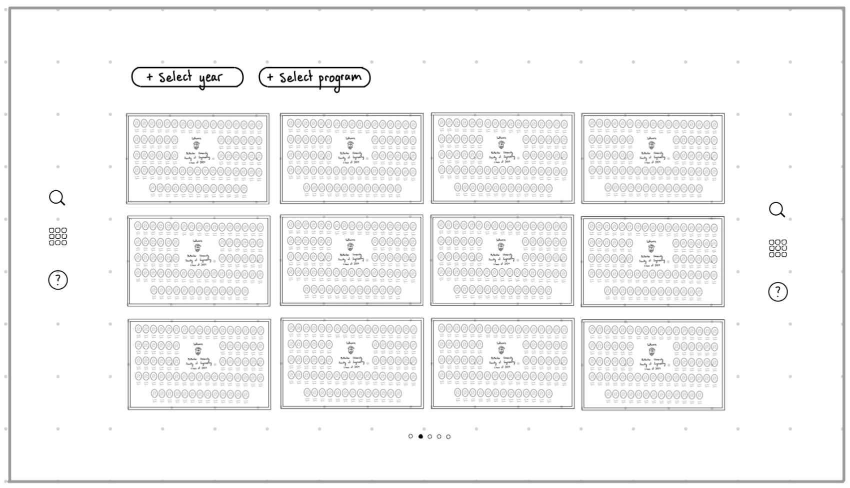


Figure 5: Composite Search Interface — Users can search for individuals by name, program, or year. M7 handles the interactive UI and form input, while M5 retrieves matching data.

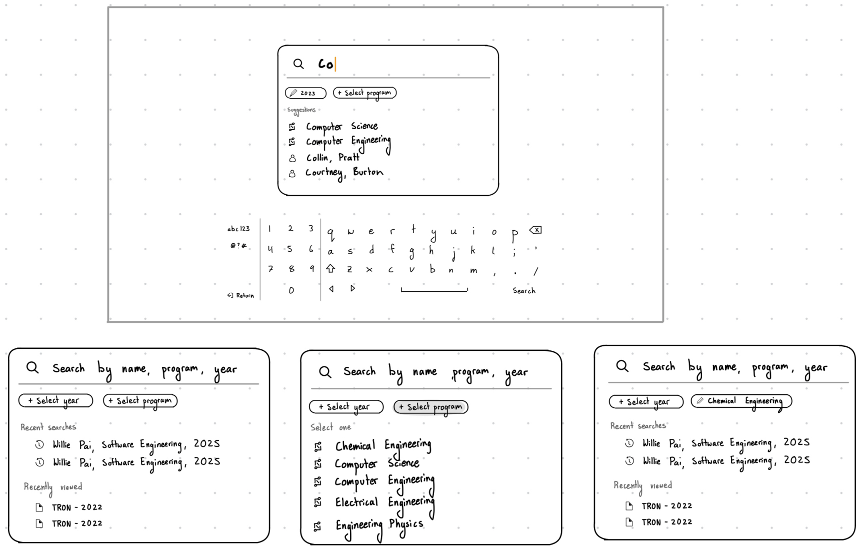
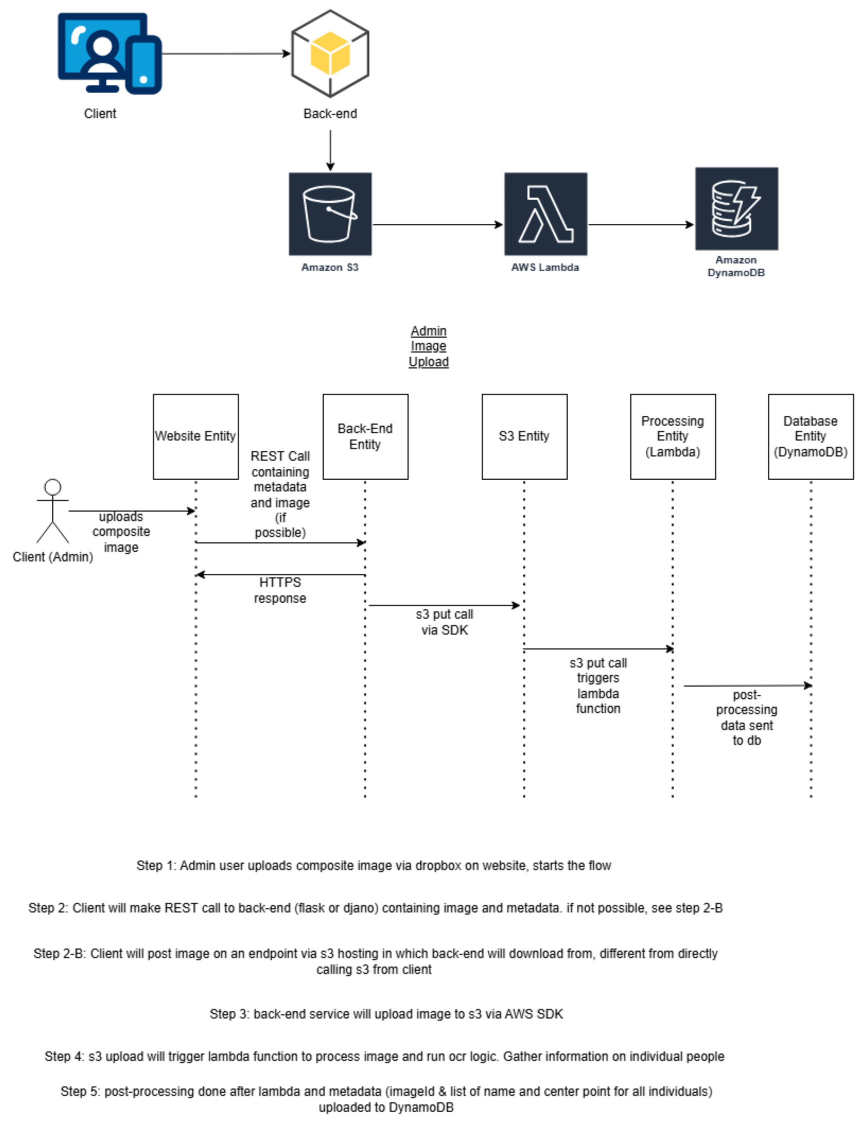


Figure 6: Composite Viewer Interface — The selected composite is displayed with highlights for identified names. This view is rendered by M7 and populated by data from M4 (OCR) and M5 (Output).



Step 1: Admin user uploads composite image via dropbox on website, starts the flow

Step 2: Client will make REST call to back-end (flask or djano) containing image and metadata. if not possible, see step 2-B

Step 2-B: Client will post image on an endpoint via s3 hosting in which back-end will download from, different from directly calling s3 from client

Step 3: back-end service will upload image to s3 via AWS SDK

Step 4: s3 upload will trigger lambda function to process image and run ocr logic. Gather information on individual people

Step 5: post-processing done after lambda and metadata (imageId & list of name and center point for all individuals) uploaded to DynamoDB

Figure 7: Interface Software Architecture — This diagram outlines how the UI interacts with backend services. M6 connects with M2–M5, and M7 manages the front-facing display and user flow.

11 Timeline

The development of the GradSight system follows a structured, phase-based schedule designed to ensure a reliable Rev 0 implementation by March. Each phase includes specific deliverables, personnel assignments, and dates. The design emphasizes modular parallelism so team members can develop and test individual components without bottlenecks. Testing is embedded in each phase where appropriate to support early validation.

11.1 Requirements Gathering and System Design

- **Sept 23 – Oct 9:** Finalization of software requirements (SRS), hazard analysis, and preliminary architecture planning.
- **Tasks:**
  - Define FRs, NFRs, and design constraints.
  - Complete FMEA and anticipated change table.
  - Identify modules and interfaces.
- **Leads:** Wajdan (System), Zahin (Hazards), Willie (Comms)

11.2 Front-End and Back-End Development

Each developer is assigned one or more modules to implement in parallel. Cloud services, OCR pipelines, and database schema are set up early to unblock downstream modules. User interface work starts once the backend API is ready.

Table 4: Module-Level Development Timeline and Responsibilities

Module	**Developer(s)**	**Timeline**	**Responsibility**
M1: Cloud	Henushan	Oct 9–14	Set up S3, Lambda functions, and DynamoDB with API integration
M2: Input	Wajdan	Oct 12–16	Validate file structure and convert raw upload into standardized input format
M3: Upload	Zahin, Henushan	Oct 14–18	Implement frontend upload logic and S3 upload pipeline with error handling
M4: OCR	Hammad, Wajdan	Oct 18–22	Integrate OCR engine (e.g., Tesseract or Textract) to extract names/coordinates
M5: Output	Henushan	Oct 22–26	Design database schema and write DynamoDB interface for storing parsed results
M6: UI Parsing	Willie, Hammad	Oct 26–31	Implement fallback name collection and integrate OCR and database results
M7: GUI	Willie, Zahin	Nov 1–10	Develop touchscreen interface to navigate and search composite data

11.3 Proof of Concept and Testing

- **Nov 22 – Feb 3:** Internal testing of core system features. Each module is tested with mock data and then integrated with upstream/downstream modules.
- **Tasks:**
  - Unit testing (M2–M5): Hammad
  - UI testing (M6–M7): Willie
  - OCR and cloud validation: Zahin
- Integration testing begins once the full data pipeline is connected end-to-end.

11.4 Final Deployment and Demonstration

- **Feb 3 – Mar 20:** Final feature polishing, performance improvements, and deployment to production environment.
- **Mar 21 – Mar 30:** Capstone demo preparation, walkthrough testing, UI/UX feedback, poster creation, and video submission.

11.5 Post-Deployment Support and Documentation

- **Mar 30 – Apr 2:** Admin onboarding guide, system support documentation, and user-facing instructions.
- **Primary Author:** Hammad (Support), reviewed by Zahin and Willie