System Verification and Validation Plan for Software Engineering

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

April 1, 2025

Revision History

Date	Version	Notes
Nov 4, 2024	1.0	Initial Document
March 31 2025	1.1	Updated to meet rubric feedback

Contents

List of Tables

1 General Information

1.1 Summary

GradSight is designed as a user-friendly, digital platform for interacting with graduation composites, modernizing what has traditionally been a static, physical display. This platform enables users to search and filter through alumni composites by year, program, and individual names, providing a richer, more accessible experience. By replacing physical displays with a digital interface, GradSight allows broader access, enhanced usability, and the preservation of alumni records in a format more suited to future integration and scalability.

1.2 Objectives

The key objectives of this Verification and Validation (VnV) Plan focus on ensuring GradSight's functionality, security, usability, and reliability. The prioritized objectives are as follows:

- **Build Confidence in Software Correctness:** Verify that core functionalities, including data retrieval, user search, and profile interaction, are functioning as specified.

- **Demonstrate Usability:** Validate that GradSight's interface is intuitive, allowing users to efficiently navigate, search, and interact with alumni composites.

- **Verify Data Security Compliance:** Confirm that GradSight's data management aligns with McMaster's privacy and security standards, focusing on role-based access control, data encryption, and secure data transmission protocols to protect sensitive information.

Out-of-scope objectives due to resource limitations:

- **Extensive Validation of External Libraries:** Since GradSight relies on TensorFlow for OCR functionality, this plan assumes that the library has been rigorously tested by its developers. Our testing will focus on how GradSight integrates with TensorFlow, without delving into TensorFlow's internal operations.

By prioritizing these objectives, the VnV Plan ensures that the most critical qualities of GradSight receive focused attention, balancing rigorous validation with awareness of resource constraints.

1.3 Challenge Level and Extras

The project is classified as a general level challenge.

Additional extras include enhanced accessibility features all aimed at meeting Web Content Accessibility Guidelines (WCAG).

1.4 Relevant Documentation

This VnV plan is supported by key project documents that provide foundational information and context:

- **Software Requirements Specification (SRS):** The SRS offers a detailed breakdown of GradSight's functional and nonfunctional requirements (Wajdan et al. 2024).
- **Hazard Analysis:** This document identifies potential risks to system security, privacy, and data integrity, highlighting areas that need rigorous testing and validation (Wajdan et al. 2024).
- **Development Plan:** The Development Plan describes the project's timeline, team roles, and workflow structure (Wajdan et al. 2024).
- **Problem Statement:** The Problem Statement outlines the motivation behind GradSight's development, detailing the limitations of the current physical composites and the anticipated advantages of a digital solution (Wajdan et al. 2024).

Each of these documents underpins the VnV plan by providing specific requirements and design considerations that guide test case development and validation activities.

2 Plan

This section outlines the strategy our team will use to verify and validate the project's requirements, design, and implementation to ensure quality and alignment with our objectives. The following sections detail our verification and validation team, our approach to verifying the Software Requirements Specification (SRS), and our design verification plan.

2.1 Verification and Validation Team

The verification and validation team comprises six members, including our supervisor, with each member bringing specific areas of expertise to ensure thorough project verification. All members, excluding the supervisor, will be responsible for creating and executing tests in their specific areas. Those who are comfortable with other areas can swap roles after completing their verifications to re-assess other verifications for full coverage.

Team Member	**Role**
Hammad Pathan	Verify documentation accuracy and consistency, including checklists and procedures for SRS and design reviews.
Wajdan Faheem	Oversee system architecture design and ensure components are working together.
Willie Pai	Coordinate all verification activities and development of components.
Henushan Balachandran	Ensure security and privacy measures are in place and sound.
Zahin Hossain	Lead technical validation and ensure requirements are feasible for implementation.
Meggie MacDougall	Provide feedback on SRS and design documents, validating alignment with the project's overall objectives.

Table 1: Verification and Validation Team Roles

2.2 SRS Verification Plan

For our Software Requirements Specification (SRS) verification, we will use a combination of structured reviews with our supervisor, peer feedback, TA feedback, and checklist verification.

Structured Review Meeting with Supervisor:
We will organize a meeting with our supervisor to go over the SRS docu-

ment in detail. During this meeting, we will present the main requirements, focusing on the most important areas.

- **Pre-meeting Preparation:** Prior to the meeting, we will provide our supervisor with a list of key requirements, questions to address potential unclear aspects, and any known issues.
- **Task-based Inspection:** Our supervisor will be given a checklist for reviewing the SRS. This checklist will guide her in examining key aspects like completeness, feasibility, and clarity of requirements.
- **Follow-up Actions:** After the meeting, we will document any feedback and issues in our project's issue tracker for action and further discussion.

Peer Review and TA feedback:
We will also have classmates and other teams review the SRS for additional input, focusing on areas like requirement clarity, usability, and potential technical risks. In addition, we will create notes on all feedback given to us by our TA from our in person meetings, and the deliverables marked online.

SRS Verification Checklist:
A structured checklist will be used to verify that the SRS meets all necessary criteria.

Table 2: SRS Verification Checklist

Check	**Criteria**	**Description**
1. Completeness and Clarity		
1.1	Requirements are complete	All functional and non-functional requirements are included.
1.2	Requirements are specific and unambiguous	No vague language; each requirement is clear and understandable.
1.3	Requirements are concise	Each requirement is necessary, and redundancy is minimized.
1.4	User needs are accurately captured	The requirements reflect the intended user experience and interface expectations.
1.5	External dependencies are defined	Any dependencies (e.g., third-party data or resources) are documented and included.
2. Functional and Non-Functional Requirements		

Check	Criteria	Description
2.1	Functional requirements are well-defined	Each required function is fully described, including expected behavior and constraints.
2.2	Non-functional requirements are detailed	Non-functional needs like performance, security, and usability are clearly stated.
2.3	Prioritization of requirements is clear	High-priority and low-priority requirements are identified.
2.4	Requirements are measurable and testable	All requirements are phrased in a way that allows for straightforward testing and verification.
2.5	Requirements are feasible	All requirements are realistic within the scope and resources of the project.
3. Legal and Compliance Considerations		
3.1	Legal obligations are considered	Requirements address copyright, intellectual property, and user privacy as per relevant regulations.
3.2	Data protection standards	Requirements include necessary steps to comply with data protection laws and university policies.
3.3	Permissions for third-party data	If external data (e.g., alumni photos) is used, permissions and licensing requirements are documented.
4. Traceability		
4.1	Traceability matrix included	A traceability matrix is present to map each requirement to the specific SRS sections and design components.
4.2	Clear labeling of requirements	Each requirement has a unique identifier for easy referencing in future documentation.
4.3	Linkage to project objectives	Each requirement is linked back to the overarching project goals and objectives.

2.3 Design Verification Plan

For our Design verification, we will use a checklist verification, peer reviews and a structured review meeting with Michael Curwin, Associate Director of Information Technology and Services.

Structured Meeting with Associate Director of Information Technology and Services:

We will organize a meeting with Michael Curwin to go over the Design in detail. During this meeting, we will present the main system design.

- **Pre-meeting Preparation:** Prior to the meeting, we will provide Michael with a list of key design ideas, questions to address potential unclear aspects, and any known issues.
- **Task-based Inspection:** Michael will be given a checklist for reviewing the design. This checklist will guide him in examining key aspects like compatibility, security, feasibility and scalability.
- **Follow-up Actions:** After the meeting, we will document any feedback and issues in our project's issue tracker for action and further discussion.

Peer Review:
We will also have classmates and other teams review the design for additional input. We will take notes on their feedback and work to implement improvements to our design.

Design Verification Checklist:
A structured checklist will be used to verify that the design meets all requirements, is feasible and follows best practices under McMaster's IT department.

Table 3: Design Verification Checklist

Check	**Criteria**	**Description**
1. Completeness and Consistency		
1.1	Design completeness	All modules, components, and their interactions are described in detail, covering all key functionalities.
1.2	Consistency with SRS	Each design component aligns with the SRS requirements; any discrepancies are documented and resolved.
1.3	Modular design structure	The system is broken down into distinct, modular components for clarity and maintainability.
2. Traceability and Documentation		

Check	Criteria	Description
2.1	Traceability matrix updated	Each design element is mapped to corresponding requirements in the SRS traceability matrix.
2.2	Detailed component diagrams	Clear diagrams, such as flowcharts, sequence diagrams, and data flow diagrams, are included to illustrate component interactions.
2.3	Documentation up to date	Design documentation is thorough, with explanations for decisions, assumptions, and alternatives considered.
3. Security Requirements		
3.1	Authentication mechanisms	Design includes secure authentication methods (e.g., role-based access control) that align with the project's security requirements.
3.2	User data protection	The design accounts for privacy and data protection, ensuring minimal data access based on user roles.
4. Version Control Management		
4.1	Version control strategy defined	A clear version control workflow (e.g., Git branching strategy) is outlined to ensure smooth collaboration and tracking.
4.2	Commit guidelines	All team members follow commit message guidelines to maintain a clear history of changes.
4.3	Versioning of design documents	Design documents are version-controlled, with clear labeling for each major update or change.
5. Usability and Performance		
5.1	Usability considerations	User interface design aligns with usability best practices, with an emphasis on simplicity, responsiveness, and accessibility.
5.2	Performance benchmarks set	Key performance indicators (e.g., load time, responsiveness) are defined and accounted for in the design.

Check	Criteria	Description
5.3	User flow consistency	The user flow is logical, with clear navigation paths that align with the SRS-defined user scenarios.

2.4 Verification and Validation Plan Verification Plan

The verification and validation (V&V) process for GradSight explicitly encompasses both dynamic and non-dynamic testing methodologies to ensure thorough assessment and quality assurance.

Dynamic Testing

Dynamic testing will involve systematic execution of tests including:

- Unit tests (Section 4)
- Integration tests (Section 3.1)
- System and acceptance tests (Section 3.1, 3.2)

Each dynamic test will explicitly follow documented test cases, providing clear inputs, expected outputs, and pass criteria.

Non-dynamic (Static) Testing

Static testing methodologies will be rigorously conducted through structured code reviews, inspections, and document walkthroughs. The detailed approach for non-dynamic testing is outlined explicitly below:

Code Reviews and Inspections
Each pull request (PR) will undergo formal code reviews, adhering strictly to the following checklist:

- Reviews will be documented explicitly through GitHub PR comments and tracked within project management tools.

Document Walkthroughs
Periodic walkthroughs will be performed for essential documents (e.g., SRS, MIS, MG, Hazard Analysis, VnV Plan) to confirm clarity, accuracy, and consistency:

- Walkthrough outcomes and identified improvements will be explicitly logged and tracked through versioned documents.

Static Analysis Tools

Static analysis tools will continuously be integrated into the development process to detect potential errors early:

- ESLint: Enforce JavaScript coding standards and detect potential runtime issues
- Prettier: Ensure consistent code formatting
- PyLint: Maintain coding quality in Python modules

Static analysis tool outputs will explicitly be included in automated GitHub Actions CI/CD pipelines, ensuring continuous compliance and immediate feedback to developers.

2.5 Implementation Verification Plan

This section outlines the approach for verifying that each module of GradSight is implemented correctly according to its design specification (MIS), and adheres to the requirements outlined in the SRS and MG documents. Both dynamic and static testing techniques will be applied.

Module-Level Verification Overview

Module Name	**Verification Method**	**Responsible Team Member(s)**	**Evidence Collected**
User Auth (M01)	Manual testing, Code review	Hammad Pathan	Login test logs, screenshots, code inspection checklist
Composite Upload (M02)	Functional testing, Static inspection	Wajdan Faheem	Uploaded files, metadata JSON, inspection notes
Search & Filter (M03)	Automated UI test, API response validation	Zahin Hossain	Console logs, response time reports

OCR Integration (M04)	Dynamic test w/ test image set	Willie Pai	Extracted names, OCR logs, error messages
Logging & Audit (M05)	Static analysis, Log trace validation	Henushan Balachandran	Log file samples, encryption metadata, data access patterns
Composite Viewer UI (M06)	UI usability tests, Responsiveness tests	All team members	User feedback forms, mobile/desktop screenshots
Admin Tools (M07)	Role-based access test, Input validation	Zahin Hossain	Access logs, incorrect-input test report

Test Data and Inputs

- **User Credentials:** Valid, invalid (non-McMaster), blank, and malformed inputs
- **Composite Files:** Various resolutions, file types, and naming conventions
- **Search Terms:** Edge cases (empty search, partial year, invalid program names)
- **Touch Interaction:** Simulated gesture inputs (swipe, pinch, tap)
- **OCR Inputs:** Blurry and clear composite samples for name extraction

Verification Activities

- **Code Review Checklists:** Used to inspect adherence to modular structure, error handling, and data validation
- **Static Analysis:** Verifies absence of code smells and enforces naming/formatting conventions
- **Manual Testing:** Includes walkthroughs for login, upload, search, and view features
- **Dynamic UI Testing:** Real-time validation of zoom, filter, navigation, and responsiveness

- **API Testing:** Ensure backend endpoints return expected data formats and handle errors gracefully

Pass/Fail Criteria

- Each module must pass all its defined test cases in Section 3.1 or 3.2
- No critical errors or crashes observed under normal conditions
- All user flows must return the expected output or error messages
- Static review must yield zero high-priority defects
- UI tests must meet accessibility and consistency standards

Documentation of Evidence

- All test outcomes, screenshots, API logs, user feedback, and walk-through notes will be archived under the GradSight VnV folder
- Each defect or failure will be recorded in the team issue tracker (GitHub) and linked to its associated module and test ID

2.6 Automated Testing and Verification Tools

To ensure thorough and efficient verification of GradSight, the following automated testing tools and verification strategies will be utilized. These tools have been explicitly selected based on the implementation technologies used.

Unit Testing Tools

- **Jest:** Unit testing of JavaScript and React components
  Modules: M2 (Input Validation), M3 (Upload Handling), M6 (UI Parsing)
- **React Testing Library:** Testing user interactions and DOM updates
  Modules: M7 (GUI)
- **PyTest:** Python-based unit testing
  Modules: M4 (OCR Module)

System and Integration Testing Tools

- **Playwright:** End-to-end testing of user interfaces and interactions
  Modules: M3, M5, M6, M7
- **Postman:** Testing API endpoints and backend integration
  Modules: M1 (Cloud Module), M5 (Output Module)
- **Mockaroo / Fixture Scripts:** Generate mock data for testing OCR outputs and UI fallback entries
  Modules: M4 (OCR Module), M6 (UI Parsing)

Static and Non-dynamic Testing Tools

- **ESLint:** Enforce JavaScript and TypeScript coding standards
- **Prettier:** Code formatting consistency
- **VS Code Extensions:** Real-time linting and formatting checks during development
- **Manual Review Checklist:** Structured peer reviews focusing on logic correctness, code quality, and security concerns

Continuous Integration and Automation

- **GitHub Actions:**
  - Automated unit tests executed on each pull request
  - Linting and formatting automatically validated
  - Optional integration testing (Playwright) executed headlessly

2.7 Software Validation Plan

The purpose of validation is to ensure that the GradSight system fulfills the needs of its stakeholders, including faculty, students, and administrative staff, and satisfies all system-level requirements described in the SRS. Validation focuses on checking that the right product was built, not just whether it was built correctly.

Validation Objectives

- Confirm that all user goals (e.g., uploading, viewing, and searching composites) can be achieved with minimal training.
- Ensure that only authorized users can access restricted features.
- Validate usability under real-world constraints like touchscreen kiosks and limited lighting.
- Confirm that system behavior aligns with expectations derived from the SRS and Hazard Analysis.

Stakeholders Involved

Stakeholder	Role in Validation
Faculty members	Upload composites and verify proper role-based access
Students	Search and view composites; provide feedback on usability
Admins (project team)	Simulate kiosk environment and conduct technical walkthroughs
Reviewers (TA/instructor)	Confirm alignment with SRS and hazard controls

Validation Activities

Activity	Description	When
Scenario-based walkthroughs	Stakeholders follow predefined tasks (e.g., upload, search, zoom, tap profile) to simulate real usage	After implementation of each milestone
User feedback survey	Users rate ease of use, clarity, and accessibility (Likert scale + open-ended questions)	During usability testing
Acceptance tests	Functional tests run from the user's perspective to confirm system meets acceptance criteria	Final phase of development
SRS compliance review	Map each FR/NFR from the SRS to actual working features and validate coverage	Midway and end of implementation

Kiosk simulation	Full interaction tested on touchscreen setup under various lighting conditions	Prior to deployment

Validation Inputs

- Composite images of various qualities
- Pre-filled metadata for testing upload accuracy
- Non-McMaster and malformed emails for validation testing
- Simulated real user tasks from the Hazard Analysis doc (e.g., touching sensitive areas, zoom overflow)

Success Criteria

- All core tasks can be completed by a new user within 2 minutes
- No task requires more than 2 taps/clicks to locate critical functionality
- At least 80% user satisfaction in usability survey
- All validation tasks yield expected results with no critical system errors
- All validation logs, screenshots, and feedback stored in GradSight evidence repository

3 System Tests

This section provides detailed system tests that cover all requirements specified in Software Requirements Specification (SRS) for the GradSight project. The tests are designed to ensure that the system functions correctly, and meets the needs of its users.

3.1 Tests for Functional Requirements

The following test cases are organized according to the functional requirements outlined in the SRS. It covers all aspects of the system's functionality, from user authentication to composite uploads and viewings. The test cases are detailed to allow precise testing of the system once built completely.

3.1.1 User Registration and Authentication

Test Case: Student Registration **Requirement ID:** FR-UR-01 (User Registration)
Control: Manual
Initial State: User "Jane Smith" does not exist in the database.
Input:

- Navigate to registration page (/register)
- Enter the following:
  - First Name: Jane
  - Last Name: Smith
  - Email: jane.smith@mcmaster.ca
  - Password: SecurePass123!
- Click "Register"

Expected Output:

- "Registration successful! Please check your email to confirm your account." displayed
- User account created in database (with email confirmation status: pending)

Test Steps:

- Navigate manually to /register
- Enter provided details
- Click "Register"
- Verify confirmation message
- Check database for new user

Test Case: Successful Login **Requirement ID:** FR-UR-01 (User Login)
Control: Manual
Initial State: User exists:

- Email: john.doe@mcmaster.ca
- Password: Password123!

Input:

- Navigate to login page (/login)
- Enter credentials above
- Click "Login"

Expected Output:

- Redirected to Dashboard (/dashboard)
- "Welcome, John Doe" message displayed

Test Steps:

- Manually enter credentials
- Click "Login"
- Confirm dashboard redirection

Test Case: Invalid Login Attempts **Requirement ID:** FR-UR-01 (Login Error Handling)
Control: Manual
Initial State: System operational, valid user: john.doe@mcmaster.ca
Inputs and Expected Outputs:

Scenario	Email Input	Password Input	Expected Output
A	nonmcmaster@gmail.com	randompass	"Invalid email or password"
B	abc123	wrongpass	"Invalid email or password"
C	john.doe@mcmaster.ca	WrongPass123	"Invalid email or password"

Test Steps:

- Enter each scenario manually
- Attempt login
- Confirm error message matches exactly

Test Case: Unauthorized Access Requirement ID: FR-AC-01 (Access Control)
Control: Manual
Initial State: User logged in as student (student@mcmaster.ca)
Input: Attempt to visit admin-only URL (/admin/upload)
Expected Output:

- Error message: "You do not have permission to access this page"
- User redirected to the dashboard or login page

Test Steps:

- Manually navigate to restricted URL
- Confirm appropriate error message and redirection

3.1.2 Uploading Graduation Composites

Test Case: Composite Upload Requirement ID: FR-UP-01 (Composite Upload)
Control: Manual
Initial State: User logged in as admin (admin@mcmaster.ca)
Input:

- File: Composite2025.jpg (JPEG, 1920x1080, ¡500KB)
- Year: 2025
- Program: Software Engineering

Expected Output:

- Message: "Composite for Software Engineering 2025 uploaded successfully."
- Composite stored in AWS S3 at /composites/2025/SoftwareEngineering/
- Database entry added in DynamoDB with correct metadata

Test Steps:

- Navigate to /admin/upload
- Enter provided details
- Upload file
- Verify confirmation message and storage/database entries

Test Case: Metadata Retrieval **Requirement ID:** FR-MD-01 (Metadata Storage)
Control: Automated (Postman or Curl)
Initial State: Composite (Composite2025.jpg) previously uploaded
Input: GET request to /api/composite?id=Composite2025
Expected Output: JSON response:

```
{
  "file": "Composite2025.jpg",
  "year": "2025",
  "program": "Software Engineering"
}
```

Test Steps:

- Send GET request
- Verify response content matches exactly

3.1.3 OCR and Manual Corrections

Test Case: OCR Name Parsing **Requirement ID:** FR-OCR-01
Control: Automated (Unit test)
Initial State: OCR module active
Input: Upload `test_composite_ocr.jpg` (contains clearly readable names)
Expected Output: OCR response: JSON array of name-coordinate pairs
Test Steps:

- Upload test image
- Confirm OCR output against expected test data

Test Case: Manual Name Fallback Entry Requirement ID: FR-UP-04
Control: Manual
Initial State: OCR failed parsing certain names
Input:

- Manually enter:
  - Name: Bob Patel
  - Coordinates: X=450, Y=300

Expected Output:

- Name and coordinates saved successfully
- Appears on composite UI upon user interaction

Test Steps:

- Enter data through UI
- Click composite to confirm visibility

3.1.4 Viewing and Searching Composites

Test Case: Composite Viewing and Interaction Requirement ID: FR-VI-01
Control: Manual
Initial State: Composite2025.jpg uploaded with metadata
Input:

- Filter:
  - Year: 2025
  - Program: Software Engineering
- Interactions:

    - Zoom
    - Pan
    - Click profile: Alice Smith

Expected Output:

- Composite image displays smoothly
- Popup displays correct profile information (Alice Smith, Software Engineering)

Test Steps:

- Manually perform interactions
- Confirm visual and functional correctness

Test Case: Composite Search **Requirement ID:** FR-SR-01
Control: Manual
Initial State: Composites exist for various years and programs
Input:

- Year: 2025
- Program: Software Engineering
- Click "Search"

Expected Output:

- Composite2025.jpg listed clearly
- Only relevant results shown

Test Steps:

- Enter criteria and search
- Verify correct results listed

3.2 Tests for Nonfunctional Requirements

The nonfunctional aspects of reliability, data accuracy, usability, and code quality require extensive testing of the GradSight digital composite display system. This section gives our structured approach toward the assessment of such non-functional requirements. Each of the test cases described in this section will support the quality of GradSight regarding the standards of the project and the norms for data security and usability at McMaster University.

3.2.1 Look and Feel Requirements

Test Case: Display Quality (NFR-LF-01)

- Images load in high resolution (minimum 1080p)
- No pixelation or distortion visible at normal viewing distance
- Images consistent across multiple devices

Test Case: Consistent Screen Layout (NFR-LF-02)

- All primary screens share the same layout structure
- Navigation menus identical on all pages
- Element positioning consistent with the design mockups

3.2.2 Usability and Humanity Requirements

Test Case: Touch Sensitivity (NFR-UH-01)

- Touch interactions respond within 200 milliseconds
- Swipe gestures smoothly recognized
- Pinch-to-zoom gestures responsive without jitter or delay

Test Case: Minimal Learning Curve (NFR-UH-02)

- Users successfully complete key tasks (search, upload, view composites) unassisted
- All essential tasks completed within 5 minutes
- Users report confidence in independently using the application again

Survey:

- Rate ease of task completion (1–5 scale)
- Rate confidence in future app use (1–5 scale)

Test Case: On-Screen Guidance (NFR-UH-03)

- Tooltips available and correctly displayed on hover
- First-time users see instructional overlays clearly
- Help icons effectively provide concise guidance

Test Case: Clarity of Instructions (NFR-UH-04)

- Instructions clearly visible on every relevant screen
- Instructions free of technical jargon
- Text instructions concise and grammatically correct

Test Case: Politeness in Error Messages (NFR-UH-05)

- Error messages clearly indicate what went wrong
- Provide clear guidance on resolving the issue
- Tone remains polite and helpful throughout

3.2.3 Performance Requirements

Test Case: Search Speed (NFR-PR-01)

- Search returns results within 2 seconds under typical conditions
- Search returns results within 5 seconds under simulated high-load conditions

Test Case: Page Load Time (NFR-PR-02)

- Page load time does not exceed 2 seconds in normal operation
- Page load time does not exceed 4 seconds under peak simulated usage

3.2.4 Reliability Requirements

Test Case: System Uptime and Crash Recovery (NFR-RB-01)

- Simulated system crash automatically recovers without manual intervention
- No loss of data occurs after crash recovery
- System maintains an uptime of at least 99.5% over one-week testing period

3.2.5 Maintainability and Support Requirements

Test Case: Adaptability (NFR-MS-02)

- UI elements scale properly on screen sizes from 10” tablets to 27” monitors
- Interface remains fully usable at 150% and 200% browser zoom settings
- Layout adjusts proportionally for 1080p and 4K screen resolutions

3.2.6 Security Requirements

Test Case: Data Security (NFR-SC-01)

- Data encrypted at rest using AES-256 standard
- Data encrypted during transmission using TLS
- Unauthorized access attempts properly logged and alerted

Test Case: Access Controls (NFR-SC-01)

- Verify role-based permissions (Student, Faculty, Admin)
- Unauthorized access attempts effectively denied and logged

Test Case: Audit Log Integrity (NFR-SC-04)

- Logs record accurate timestamps, user IDs, and detailed actions
- Logs are protected from unauthorized modifications or deletions

3.2.7 Compliance Requirements

Test Case: Legal and Standards Compliance (NFR-CO-01, NFR-CO-02)

- Application adheres to applicable intellectual property and privacy laws
- Fully complies with Web Content Accessibility Guidelines (WCAG)
- Meets McMaster University's data governance and compliance standards

3.3 Traceability Between Test Cases and Requirements

The following table shows the relationship between the system's requirements and the corresponding test cases defined in Sections 3.1 (Functional) and 3.2 (Nonfunctional). This mapping ensures completeness, traceability, and test coverage. All requirement identifiers follow the format and numbering from the SRS document.

Requirement ID	Requirement Description	Test Case(s)
FR-UR-01	Student registration and login	TC-UR-01, TC-UR-02, TC-UR-03
FR-UR-02	Faculty registration and login	TC-UR-04, TC-UR-05
FR-AC-01	Unauthorized access control	TC-AC-01
FR-UP-01	Faculty uploads graduation composites	TC-UP-01
FR-MD-01	Metadata stored with composites	TC-UP-02
FR-VZ-01	View, zoom, and interact with composites	TC-VZ-01
FR-SR-01	Search composites by year/program	TC-SR-01
FR-LG-01	Log user actions like login and uploads	TC-LG-01
NFR-LF-01	High-resolution composite display	TC-LF-01
NFR-LF-02	Consistent interface layout	TC-LF-02
NFR-LF-03	Non-distracting color scheme	TC-LF-03
NFR-LF-04	Legible, scalable fonts	TC-LF-04
NFR-LF-05	Intuitive icons and labels	TC-LF-05
NFR-LF-06	Consistency across UI elements	TC-LF-06
NFR-UH-01	Touchscreen support	TC-UH-01
NFR-UH-02	Minimal learning curve	TC-UH-02
NFR-UH-03	On-screen guidance for users	TC-UH-03
NFR-UH-04	Clear instructions	TC-UH-04
NFR-UH-05	Polite and informative error messages	TC-UH-05
NFR-AC-01	Accessibility (large buttons, fonts)	TC-AC-01
NFR-PR-01	Fast search response (under 2s)	TC-PR-01
NFR-PR-02	Page load time under 2s	TC-PR-02
NFR-PR-03	Smooth screen transitions	TC-PR-03
NFR-SC-01	Secure data storage and encryption	TC-SC-01
NFR-SC-02	Input validation and data integrity	TC-SC-02
NFR-SC-03	GDPR and privacy compliance	TC-SC-03
NFR-SC-04	Secure audit logging	TC-SC-04
NFR-SC-05	Protection from cyber threats	TC-SC-05
NFR-CR-01	Support for multiple concurrent users	TC-CR-01
NFR-SE-01	Scales with added composite data	TC-SE-01
NFR-SE-02	Supports future extensibility	TC-SE-02

NFR-LR-01	Easy to maintain (modular, documented)	TC-LR-01
NFR-OE-01	Durable indoor placement	TC-OE-01
NFR-OE-02	Display readable in various lighting	TC-OE-02
NFR-OE-03	Compatible with building infra	TC-OE-03
NFR-MS-01	Logging and diagnostics support	TC-MS-01
NFR-MS-02	Adaptable to different screens/devices	TC-MS-02
NFR-CU-01	Unicode support for names	TC-CU-01
NFR-CO-01	Compliance with copyright/IP law	TC-CO-01
NFR-CO-02	ISO/IEC 25010 + GDPR compliance	TC-CO-02

Table 8: Requirements-to-Test Case Mapping

4 Unit Test Description

This section outlines the comprehensive set of unit tests designed to verify individual modules in the GradSight system. These tests focus on ensuring both functional correctness and nonfunctional behavior (e.g., performance and maintainability) based on the MIS (Modular Interface Specification) design document.

4.1 Unit Testing Scope

All core GradSight modules will undergo unit testing. No modules are considered out-of-scope, although modules with simple utility roles or minimal logic may have fewer test cases due to their lower complexity.

Modules are prioritized as follows:

- **High Priority:** User Authentication (M01), Composite Upload (M02), Search
  Filter (M03), OCR Integration (M04)
- **Medium Priority:** Logging
  Audit (M05), Composite Viewer UI (M06)
- **Lower Priority:** Admin Tools (M07)

The rationale is based on the complexity and criticality of each module as defined in the MIS. Higher-priority modules involve sensitive operations, external integrations, or key user workflows.

4.2 Tests for Functional Requirements

Most unit tests will be automated using Jest, PyTest, or React Testing Library depending on the module's technology. Black-box and white-box strategies will be employed.

4.2.1 Module 1: User Authentication (M01)

Responsible for login, logout, and role-based access.

1. UT-M01-01
   Type: Functional, Dynamic, Automatic
   Initial State: User not logged in
   Input: Valid login credentials
   Output: Login success, session created
   Test Case Derivation: Verifies that correct user credentials allow access
   How test will be performed: Automated test with Jest using mock API

2. UT-M01-02
   Type: Functional, Dynamic, Automatic
   Initial State: User not logged in
   Input: Invalid password
   Output: Login failure message
   Test Case Derivation: Checks failure on incorrect input
   How test will be performed: Use React Testing Library to simulate input and assert response

4.2.2 Module 2: Composite Upload (M02)

Handles file uploads, year/program metadata.

1. UT-M02-01
   Type: Functional, Dynamic, Automatic
   Initial State: Admin logged in
   Input: Valid JPEG file, metadata

Output: Upload success message
Test Case Derivation: Normal behavior with valid inputs
How test will be performed: Playwright test for form submission

4.2.3 Module 3: Search and Filter (M03)

Searches based on year, name, or program.

1. UT-M03-01
   Type: Functional, Dynamic, Automatic
   Initial State: Composite data seeded
   Input: Year = 2025
   Output: Correct list of results
   Test Case Derivation: Tests accuracy of filtering
   How test will be performed: Jest and mock API calls

4.3 Tests for Nonfunctional Requirements

4.3.1 Module 4: OCR Integration (M04)

Assesses name parsing reliability and timing.

1. UT-M04-01
   Type: Nonfunctional, Dynamic, Automatic
   Initial State: OCR module ready
   Input/Condition: Blurry input image
   Output/Result: Proper error handling or partial results
   How test will be performed: Python test using PyTest with preloaded test image

4.3.2 Module 5: Composite Viewer UI (M06)

Tests usability and responsiveness.

1. UT-M06-01
   Type: Nonfunctional, Dynamic, Automatic
   Initial State: App running
   Input: Touch gesture (zoom/pan)
   Output: Responsive animation and layout
   How test will be performed: Playwright test on emulated touchscreen browser

4.4 Traceability Between Test Cases and Modules

- **M01 - User Authentication:** UT-M01-01, UT-M01-02
- **M02 - Composite Upload:** UT-M02-01
- **M03 - Search Filter:** UT-M03-01
- **M04 - OCR Integration:** UT-M04-01
- **M06 - Composite Viewer UI:** UT-M06-01

All modules in the MIS are covered with at least one unit test. Non-functional tests also validate performance and usability for mission-critical components.

References

[1] Hammad Pathan, Zahin Hossain, Willie Pai, Henushan Balachandran, Wajdan Faheem, *Software Requirements Specification (SRS)*, 2024. https://github.com/PaisWillie/Digital-Composite/blob/main/docs/SRS-Volere/SRS.pdf

[2] Hammad Pathan, Zahin Hossain, Willie Pai, Henushan Balachandran, Wajdan Faheem, *Hazard Analysis*, 2024. https://github.com/PaisWillie/Digital-Composite/blob/main/docs/HazardAnalysis/HazardAnalysis.pdf

[3] Hammad Pathan, Zahin Hossain, Willie Pai, Henushan Balachandran, Wajdan Faheem, *Development Plan*, 2024. https://github.com/PaisWillie/Digital-Composite/blob/main/docs/DevelopmentPlan/DevelopmentPlan.pdf

[4] Hammad Pathan, Zahin Hossain, Willie Pai, Henushan Balachandran, Wajdan Faheem, *Problem Statement and Goals*, 2024. https://github.com/PaisWillie/Digital-Composite/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf

Appendix — Reflection

What went well while writing this deliverable?

Things that went well while writing this deliverable was the team's clear role distribution and effective communication. Each member had specific responsibilities, which streamlined the document creation process, and helped refine objectives and requirements, ensuring we met all necessary criteria. Moreover, the team was on the same page when it came to how the project should be structured, so coming to decisions on certain topics did not take long. This helped move the progress on this document as it had a lot of decisions along the way that needed to be addressed.

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

We faced challenges in defining and categorizing test cases, especially for nonfunctional requirements like usability and accessibility, which require subjective evaluation. Moreover, we still have a lot of unknowns in the project when it comes to LifeTouch and the technology we will be using. To avoid issues with this, we decided to stick to certain guidelines and stay on the same page, so there aren't any inconsistencies in the document.

What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project?

- **Hammad** — Learning how to use Jest and set up automated tests.
- **Wajdan** — Understanding system architecture validation to ensure component-level compatibility.
- **Zahin** — Developing Dynamic Testing Knowledge to conduct runtime tests effectively.
- **Henushan** — Gaining skills in security testing and vulnerability assessment.
- **Willie** — Learning to work with PyTest and develop backend tests.

For each knowledge area, what are two approaches to mastering it? Which one will be pursued and why?

- **Hammad:**
  - Watch YouTube tutorials and read documentation on Jest.
  - Ask friends/acquaintances for help and hands-on guidance.
    Chosen: Second approach — direct visuals and guidance provide faster learning.
- **Zahin:**
  - Follow online courses and tutorial videos on dynamic testing.
  - Practice dynamic testing on small sample applications.
    Chosen: First approach — structured tutorials ensure thorough understanding.
- **Wajdan:**
  - Enroll in formal online courses (e.g., Coursera).
  - Read system design blogs and crash courses.
    Chosen: Second approach — better suited to team members' time constraints.
- **Henushan:**
  - Take a security fundamentals course or certification.
  - Use gamified platforms to learn through hands-on exercises.
    Chosen: Second approach — more engaging and time-efficient.
- **Willie:**
  - Research online resources, documentation, and video tutorials on PyTest.
  - Learn through building tests for small-scale personal backends.
    Chosen: First approach — ensures foundational knowledge before applying it to project.