VnV Plan

Team 5, GradSight

Hammad Pathan
Zahin Hossain
Willie Pai
Henushan Balachandran
Wajdan Faheem

Contents

This Verification and Validation (VnV) Plan is designed to ensure that GradSight—the digital composite display system for McMaster University—meets its design goals in functionality, security, usability, and reliability. By implementing structured testing and validation steps, this document lays out a comprehensive approach for evaluating how well GradSight fulfills its intended purposes. This plan will also serve as a tool for building confidence in GradSight's performance, security, and longevity.

The VnV Plan is organized as follows:

General Information: This section provides an overview of GradSight, including its objectives, scope, and any limitations in testing coverage. Some other things that are identified include the primary goals of verification, such as functionality, security, and usability, as well as the resources and strategies that support these goals.

Plan: The plan section describes the organization of the verification and validation team, detailing each member's responsibilities. It also outlines specific plans for verifying the Software Requirements Specification (SRS) and design documents, as well as the approach to validating the VnV Plan itself. Lastly, this section includes our strategies for implementation verification, automated testing, and the use of validation tools.

System Tests: Here, we present detailed test cases for both functional and nonfunctional requirements, directly mapping these to the SRS. This section includes tests for GradSight's core functionalities, such as data retrieval, user interactions, and interface responses, as well as nonfunctional tests focused on performance, reliability, and usability.

Unit Tests: The Unit Test Description section discusses our approach to validating individual software components, ensuring that each module performs accurately and integrates smoothly within the larger system. These tests are crucial for identifying any isolated issues before full system integration.

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 focus 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.

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 document 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.

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	
	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.

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	
	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.
	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

Team members in the group will perform peer reviews of the V&V plan itself. This could be in the form of cross-checking each aspect of the plan and ensuring it aligns in our current project requirements and goals. Additionally, it could involve comparing requirements to industry standards. The standards in the review include completeness, traceability, and clarity. Completeness can focus on ensuring all implementation and testing stages are included with specific guidelines and clear objectives. Testing stages can include unit, integration, system and functional testing. Traceability will focus on each requirement containing a subset of corresponding test cases and coverage. The system requirements can and should be mapped to testing suites. Any missing requirements will immediately result in lack of coverage. Finally, clarity can be tested by providing the document to stakeholders and testing their understanding and scope of our requirements and goals. The document should be understandable to both technical and non-technical stakeholders.

An additional review can be from stakeholders. Engaging project stakeholders, in our case, faculty members and graduate students, to review the V&V will provide some external insights. They will be able to review for alignment with their own expectations, goals and usability standards. Customer obsession!

Mutation testing can be applied to assess the robustness and accuracy of the test cases within the V&V plan. This process will involve introducing small changes in the code to see if the existing test cases can detect these modifications. It's essentially trying to make our product work incorrectly to see if it will. This is important as it will reveal any weak test cases and any potential edge cases. If test cases don’t catch errors due to minor code changes, they need refinement. Additionally, it inevitably checks if all parts of the code are tested, providing a simple coverage analysis.

We can perform mutation testing by slightly modifying key components within the code (changing a condition or a variable) to create “mutants” and run test cases against these changes. Effective test cases should identify these intentional faults, indicating robust validation. Missed mutations highlight areas in the V&V plan that need additional or improved test cases. Additionally, we can scale this testing to component level, by removing small components within the system to see how it would handle these changes. If anything, this resembles more real-world scenarios where certain components will be unavailable due to server outages.

2.5 Implementation Verification Plan

The unit testing plan covers individual components of GradSight, with each unit tested in isolation to ensure it performs as expected

Specific key components to test in isolation

- **Database operations**: Verifying that all CRUD (Create, Read, Update, Delete) operations on the alumni and composite data work as expected and handle edge cases
- Testing the OCR functionality for digitizing student names and verifying that OCR reads are accurate across varying image resolutions. The most important one and should be the easiest to test, if done statically.
- **UI elements**: ensuring that each UI component, including search bars, image zoom, and navigation buttons, responds correctly to user interactions

Getting into specifics, if python is used as a back-end, PyTest will be the primary framework for unit testing. Additionally, if JavaScript is used in the front-end, Jest will be chosen.

Code walkthroughs

- We can perform code walk-throughs during team meetings within ourselves and meetings with the TA and other teams
- Meetings will focus on critical parts of the codebase and key features
    - Database operations and interactions, data access layer
    - Image handling, reading from composites and breaking down data
    - User interface. Can get figma designs and iterations
- Walk throughs can be with other teams who have similar verification plans and can cross-check verify plans.

Code Inspections

- Team members can conduct detailed code inspection, which can be automated in the PR stage of features. This is done through the approval step before merging feature branches into main branch
- Code readability - ensuring code is well-documented, easy to follow, and adheres to the project's guide
- Error Handling - Verifying that exceptions and edge cases are properly managed, especially api calls and queries
- Resource management - memory and data connections are efficiently managed. Time-to-live features to ensure downtime rarity.
-

Static Analyzers

- There are tools and libraries within IDE's and languages that help improve code quality and standards.
- ESLint for JavaScript and SonarQube for overall code quality. This can help find code vulnerabilities and automatically check for syntax issues

2.6 Automated Testing and Verification Tools

The below is briefly mentioned in the development plan as well as the SRS. The information below will summarize and provide additional info on the tools that GradSight will use.

Unit testing Framework

- PyTest: This will the primary unit testing framework for the python back-end
- Jest: This will be for the front-end. This will handle UI behavior testing

CI Tools

- GitHub Actions: It makes sense to keep automation in a central tool. We are already using GitHub and GitHub Issues for repository management and task allocation, respectively. We will use GitHub Actions to trigger builds and run tests on each commit. Automated workflow and pipeline

Code coverage tools

- Coverage.py: tool for measuring code coverage of Python programs. It monitors your program, noting which parts of the code have been executed, then analyzes the source to identify code that could have been executed but was not. Additionally, it can be integrated with GitHub Actions as a failure in build if coverage doesn't meet a threshold.

Linters and code style enforces

- ESLint for JavaScript: Since our UI will be written in JavaScript, ESLint is a strong tool to help enforce coding standards for the interface. We can modify and add rules as needed for the scope of the project, along with security rules to further

provide secure and safe user interactions. It will additionally help maintain consistent syntax within the codebase.
- If python enters the picture, with the need for language models, Flake8 is a strong tool for a scripting language like python which will enforce PEP 8 style guidelines. Refer here

Static Analyzers
- SonarQube: Will be used in both front-end and back-end. It provides insights into code quality, identifying vulnerabilities, code smells, and potential security risks
- Chrome Dev Tools: Could be useful in debugging front-end issues in the DOM

2.7 Software Validation Plan

There shouldn't be too much validation. We can manually check the composites that LifeTouch provides us to see if it is optimal resolution and meets quality standards. Additionally, any past alumni data can be cross checked manually by team members to see if they provide the metadata we need. Once we check for one, we can ensure the rest are provided in the same format.

We'll conduct task-based inspections with key stakeholders, including representatives from the Faculty of Engineering and Alumni Relations. Each stakeholder will perform tasks specified in the SRS, such as viewing composites, using search features, and zooming in on images. Stakeholders will provide feedback on whether GradSight meets the **appearance** (Section 10), **performance** (Section 12), and **accessibility requirements** (Section 11.5) specified in the SRS.

We can provide a Rev 0 demo to the supervisor and participating parties to gather more feedback and completely validate the requirements.

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 for Registration of New Student User

1. FR-UR-01
   Control: Manual
   Initial State:
   - The system is running
   - The user "John Doe" is not registered in the system

   Input:
   - User navigates to registration page
   - User fills in the registration form with the following data:
     - First name: John
     - Last Name: Doe
     - Email: john.doe@mcmaster.ca
     - Password: Password123!

   Output:
   - A new user account for "John Doe" is created with the student role
   - System displays confirmation message for successful registration
   - Email confirmation sent to john.doe@mcmaster.ca (if implemented)

   How the test will be performed:
   - Manually perform steps outlines in the input
   - Observe system's response for confirmation message
   - Verify no errors are displayed

- Attempt to log in with newly created credentials to confirm successful registration

Test for Successful Login of a Registered User

2. FR-UR-01
   Control: Manual
   Initial State:
   - User “John Doe” is registered in the system with:
     - Email: john.doe@mcmaster.ca
     - Password: Password123!

   Input:
   - User navigates to login page
   - User enters:
     - Email: john.doe@mcmaster.ca
     - Password: Password123!
   - User clicks login button

   Output:
   - User is successfully authenticated
   - USer is redirected to dashboard

   How the test will be performed:
   - Manually perform login steps
   - Ensure system does not display any error messages
   - Verify that user is redirected appropriately

Test for Unauthorized Access Attempt

3. FR-AC-01
   Control: Manual
   Initial State:
   - User “John Doe” is logged in with student role

   Input:
   - User attempts to access a faculty-only page by directly navigating to the url

   Output:
   - System denies access to the page
   - Error message is displayed telling them that they do not have permissions to view the page

   How the test will be performed:
   - While logged in as “John Doe”, attempt to access the restricted URL
   - Observe system’s response
   - Verify that access is denied and correct error message is displayed

3.1.2 Uploading Graduation Composites

Test for Uploading of Digital Graduation Composite

4. FR-UP-01
   Control: Manual
   Initial State:
   - User “Dr. Smith” is registered with:
     - Email: smith@mcmaster.ca
     - Password: SecurePassword123!
     - Role: Faculty
   - “Dr. Smith” is logged in

   Input:
   - User navigates to upload page
   - User fills in upload form:
     - Select file: Chooses image file “Composite2025.jpg”
     - Graduation: Enter “2025”
     - Program: Selects “Software Engineering” from dropdown list
   - User clicks Upload button

   Output:
   - System uploads and stores “Composite2025.jpg”
   - Metadata is stored and associated with the composite
   - Successful confirmation message displayed

   How the test will be performed:
   - Log in as “Dr. Smith”
   - Navigate to upload page and fill out form as specified
   - Observe system’s response after clicking “upload”
   - Verify confirmation message appears
   - Check database/composite listings to confirm that composite and metadata are stored

Test for Verifying Storage of Composite Metadata

5. FR-MD-01
   Control: Automatic
   Initial State:
   - Composite “Composite2025.jpg” has been uploaded as per FR-UP-01

   Input:
   - Execute a API call to retrieve the composite data associated with “Composite2025.jpg”

   Output:
   - Retrieved data includes:

- Composite ID: Some unique identifier
- File Name: “Composite 2025.jpg”
- Year of Graduation: “2025”
- Program: “Software Engineering”

How the test will be performed:

- Use automated testing tools to query API
- Compare retrieved data with expected values
- Confirm metadata fields match input provided during upload

3.1.3 Viewing and Interacting with Composites

Test for Viewing Graduation Composites

6. FR-VZ-01
   Control: Manual
   Initial State:
   - Composite “Composite2025.jpg” is available in the system

   Input:
   - User selects filter of:
     - Year: 2025
     - Program: Software Engineering
   - Composite for 2025 Software Engineering is displayed
   - User interacts with composite:
     - Zoom controls to zoom in on the image
     - Scrolls or pans to navigate different areas of the composite
     - Clicks on an individual profile within the composite

   Output:
   - Composite image is displayed clearly
   - Zoom functionality works smoothly without lag or distortion
   - Clicking on individual profile (e.g., “Alice Smith) will display a popup with:
     - Name: “Alice Smith”
     - Program: Software Engineering
     - Any additional available information

   How the test will be performed:
   - Manually perform steps outlined
   - Observe responsiveness of the display
   - Verify that zooming in and out works
   - Ensure clicking on profile brings up correct information
   - Check for any visual or functional glitches during interaction

3.1.4 Searching Graduation Composites

Test for Searching Composites by Year and Program

7. FR-SR-01
   Control: Manual
   Initial State:
   - System contains multiple composites for various years and programs, including Composite2025.jpg

   Input:
   - User navigates to search function
   - User enters search criteria:
     - Year: selects 2025 from dropdown
     - Program: selects Software Engineering dropdown
   - User clicks Search button

   Output:
   - System displays search results matching the criteria:
     - Composite: Composite2025.jpg for Software Engineering
   - If multiple composites match, only those matching all criteria are displayed

   How the test will be performed:
   - Manually input search criteria as specified
   - Observe search results
   - Verify only composites matching all entered criteria are displayed
   - Repeat tests with different search criteria combinations to ensure it works accurately

3.1.5 Logging User Activities

Test for Logging User Activities

8. FR-LG-01
   Control: Automatic
   Initial State:
   - User “Dr. Smith” is registered and not currently logged in
   - System logging is enabled

   Input:
   - “Dr. Smith” logs into the system
   - “Dr. Smith” navigates to composite upload and uploads “Composite2025.jpg”

   Output:
   - System logs following events with timestamps:
     - Login event:

        - User ID: “smith@mcmaster.ca”
        - Event type: login
        - Timestamp
    - Composite Upload Event:
        - User ID: “smith@mcmaster.ca”
        - Event type: View composite
        - Composite ID: Unique identifier
        - Timestamp

How the test will be performed:

- Perform user actions as Dr. Smith
- Access system logs through administrative tools
- Verify each activity is correctly logged with accurate details
- Ensure that sensitive information is not logged
- Check timestamps are accurate and consistent with actions performed

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 for Display Quality

Test ID: **NFR-LF-01**

- **Type**: Nonfunctional, Dynamic, Manual
- **Initial State**: System displaying a sample composite image.
- **Input/Condition**: Load composite images with varying resolutions and pixelation levels.
- **Output/Result**: The system displays high-resolution images with minimal pixelation.
- **How the test will be performed**: Manually check image clarity on the interface to ensure composite images remain clear and without pixelation at various zoom levels.

Test for Consistent Screen Layout

Test ID: **NFR-LF-02**

- **Type**: Nonfunctional, Dynamic, Manual
- **Initial State**: Interface on different pages (home, search, and view composite).
- **Input/Condition**: Navigate through different sections of the application.
- **Output/Result**: Consistent layout across all pages, with search/filter options and navigation accessible from any page.
- **How the test will be performed**: Navigate manually through the interface to check layout consistency across all pages.

Test for Color Scheme

Test ID: **NFR-LF-03**

- **Type**: Nonfunctional, Static, Inspection
- **Initial State**: Interface displayed on the screen.

- **Input/Condition**: Visual inspection of the interface color scheme on each page.
- **Output/Result**: The color palette does not distract from viewing digital composites.
- **How the test will be performed**: Perform a static review to verify that colors do not interfere with viewing composite content.

Test for Font Choice

Test ID: **NFR-LF-04**

- **Type**: Nonfunctional, Static, Inspection
- **Initial State**: System interface on various screens.
- **Input/Condition**: Adjust screen size and observe font readability.
- **Output/Result**: A legible, sans-serif font that scales properly on varying screen sizes.
- **How the test will be performed**: Perform inspection by adjusting the display size and verifying readability at various scales.

Test for Iconography

Test ID: **NFR-LF-05**

- **Type**: Nonfunctional, Static, Inspection
- **Initial State**: Interface displayed with icons.
- **Input/Condition**: Observe the icon display for intuitiveness and clarity.
- **Output/Result**: Icons are easily identifiable and convey intended actions.
- **How the test will be performed**: Conduct visual inspection to confirm that icons are intuitive and consistent across all interface sections.

Test for Consistency in Interface Elements

Test ID: **NFR-LF-06**

- **Type**: Nonfunctional, Static, Inspection
- **Initial State**: Interface displayed with various buttons and layout elements.
- **Input/Condition**: Compare buttons, margins, padding, etc., across pages.
- **Output/Result**: Uniform style in buttons, margins, and padding throughout the application.
- **How the test will be performed**: Inspect and review all screens to confirm consistent use of interface elements.

3.2.2 Usability and Humanity Requirements

Test for Touch Sensitivity

Test ID: **NFR-UH-01**

- **Type**: Nonfunctional, Dynamic, Manual
- **Initial State**: System is displayed on a touchscreen.
- **Input/Condition**: Test touch interactions such as tapping, scrolling, and swiping.
- **Output/Result**: The system responds accurately to touch gestures without delays or errors.
- **How the test will be performed**: Conduct manual tests on a touchscreen device to verify responsiveness to touch input.

Test for Minimal Learning Curve

Test ID: **NFR-UH-02**

- **Type**: Nonfunctional, Usability Test, Manual
- **Initial State**: New user interacting with the interface.
- **Input/Condition**: Observe initial interaction with no guidance.
- **Output/Result**: Users can successfully navigate and complete basic tasks without assistance.
- **How the test will be performed**: Observe a new user's interaction with the interface to gauge ease of use and any points of confusion.

Test for On-Screen Guidance

Test ID: **NFR-UH-03**

- **Type**: Nonfunctional, Usability Test, Manual
- **Initial State**: User is accessing the system for the first time.
- **Input/Condition**: Observe if on-screen guidance appears when required.
- **Output/Result**: Contextual tooltips or guidance are displayed as expected, improving ease of use for first-time users.
- **How the test will be performed**: Observe user interaction and verify the presence and effectiveness of guidance features.

Test for Clarity of Instructions

Test ID: **NFR-UH-04**

- **Type**: Nonfunctional, Usability Test, Inspection
- **Initial State**: Interface loaded with instructions displayed.
- **Input/Condition**: Observe clarity of provided instructions.

- **Output/Result**: Instructions are clear and easily understandable to users.
- **How the test will be performed**: Conduct a static review and gather feedback from users on the clarity of instructions.

Test for Politeness in Error Messages

Test ID: **NFR-UH-05**

- **Type**: Nonfunctional, Usability Test, Inspection
- **Initial State**: System in error state.
- **Input/Condition**: Trigger different error states (e.g., failed login, search error).
- **Output/Result**: Error messages are polite, informative, and guide users on corrective actions.
- **How the test will be performed**: Review error messages generated in various scenarios to ensure they are polite and helpful.

3.2.3 Accessibility Requirements

Test for Accessibility Features

Test ID: **NFR-AC-01**

- **Type**: Nonfunctional, Accessibility Test, Manual
- **Initial State**: Interface with accessibility settings enabled.
- **Input/Condition**: Test accessibility features such as large buttons and touch-friendly text.
- **Output/Result**: The interface displays larger, accessible buttons and text for touch interactions.
- **How the test will be performed**: Conduct manual inspection and interaction to confirm the accessibility features meet accessibility standards.

3.2.4 Performance Requirements

Test for Search Speed

Test ID: **NFR-PR-01**

- **Type**: Nonfunctional, Dynamic, Manual
- **Initial State**: Search page ready for input.
- **Input/Condition**: Enter search criteria and measure response time.

- **Output/Result**: Search results appear within 1-2 seconds of input.
- **How the test will be performed**: Use a timer to measure the time taken for search results to appear upon input.

Test for Page Load Time

Test ID: **NFR-PR-02**

- **Type**: Nonfunctional, Dynamic, Manual
- **Initial State**: User navigates to a new page.
- **Input/Condition**: Trigger page load and measure load time.
- **Output/Result**: New screens load within 2 seconds.
- **How the test will be performed**: Measure load time with a stopwatch or automated tool to verify it meets the standard.

Test for Smooth Animations

Test ID: **NFR-PR-03**

- **Type**: Nonfunctional, Dynamic, Manual
- **Initial State**: User navigates between sections with transitions enabled.
- **Input/Condition**: Observe animation quality during transitions.
- **Output/Result**: Transitions are smooth without lag or stutter.
- **How the test will be performed**: Visually observe transitions between screens and assess smoothness.

3.2.5 Safety-Critical Requirements

Test for Data Security

Test ID: **NFR-SC-01**

- **Type**: Nonfunctional, Security Test, Static and Dynamic
- **Initial State**: System ready with data entry fields for personal information.
- **Input/Condition**: Attempt unauthorized data access and simulate secure data transfers.
- **Output/Result**: System prevents unauthorized access and encrypts sensitive data.
- **How the test will be performed**: Conduct penetration tests and verify encryption protocols in use.

3.2.8 Capacity Requirements

Test for Concurrent Users

Test ID: **NFR-CR-01**

- **Type**: Nonfunctional, Load Test, Automated
- **Initial State**: System operational with no active users.
- **Input/Condition**: Simulate multiple users accessing the system on separate touchscreens.
- **Output/Result**: System supports multiple users without performance degradation.
- **How the test will be performed**: Simulate multiple simultaneous user interactions and monitor system performance metrics like response time and system load handling.

3.2.9 Scalability or Extensibility Requirements

Test for Scalability

Test ID: **NFR-SE-01**

- **Type**: Nonfunctional, Load Test, Automated
- **Initial State**: System database with limited number of composite records.
- **Input/Condition**: Gradually add more composite records and monitor system performance.
- **Output/Result**: System performance does not degrade significantly as more records are added.
- **How the test will be performed**: Incrementally increase database entries and measure response times and search efficiency.

Test for Extensibility

Test ID: **NFR-SE-02**

- **Type**: Nonfunctional, Static, Inspection
- **Initial State**: System codebase available for review.
- **Input/Condition**: Inspect modular structure and code design for extensibility.
- **Output/Result**: System supports addition of new features without substantial changes to existing code.
- **How the test will be performed**: Perform a code inspection to verify modular design and potential for future feature integration.

3.2.10 Longevity Requirements

Test for Maintenance Ease

Test ID: **NFR-LR-01**

- **Type**: Nonfunctional, Static, Inspection
- **Initial State**: System codebase and documentation available.
- **Input/Condition**: Inspect the modular structure and review documentation completeness.
- **Output/Result**: Code and documentation are modular and comprehensive, facilitating minimal maintenance.
- **How the test will be performed**: Review system code and documentation for modularity, readability, and completeness to support ease of future maintenance.

3.2.11 Operational and Environmental Requirements

Test for Indoor Placement Durability

Test ID: **NFR-OE-01**

- **Type**: Nonfunctional, Dynamic, Environmental Test
- **Initial State**: System installed in an indoor public area.
- **Input/Condition**: Simulate frequent interaction over an extended period.
- **Output/Result**: System withstands frequent use and remains durable indoors.
- **How the test will be performed**: Conduct prolonged usage tests to evaluate system durability in a public indoor setting.

Test for Lighting Conditions

Test ID: **NFR-OE-02**

- **Type**: Nonfunctional, Dynamic, Environmental Test
- **Initial State**: System screen displayed in varying light conditions.
- **Input/Condition**: Test system display under different lighting intensities.
- **Output/Result**: Screen visibility is maintained across lighting conditions.
- **How the test will be performed**: Evaluate screen readability in low and bright light settings to confirm visibility.

Test for Building Infrastructure Compatibility

Test ID: **NFR-OE-03**

- **Type**: Nonfunctional, Static, Inspection
- **Initial State**: Campus infrastructure requirements available for review.
- **Input/Condition**: Inspect system installation requirements (power, network) for compatibility with campus infrastructure.
- **Output/Result**: System installation requirements align with existing campus infrastructure.
- **How the test will be performed**: Review the installation requirements against campus infrastructure specifications for compatibility.

3.2.12 Maintainability and Support Requirements

Test for Supportability

Test ID: **NFR-MS-01**

- **Type**: Nonfunctional, Inspection, Static
- **Initial State**: System logs and documentation available.
- **Input/Condition**: Inspect logs, documentation, and support FAQs.
- **Output/Result**: Logs, FAQs, and documentation support diagnostic analysis effectively.
- **How the test will be performed**: Conduct an inspection of diagnostic logs and support documentation to ensure they facilitate effective troubleshooting.

Test for Adaptability

Test ID: **NFR-MS-02**

- **Type**: Nonfunctional, Static, Inspection
- **Initial State**: System design and hardware specifications available.
- **Input/Condition**: Inspect design adaptability for varying hardware and screen resolutions.
- **Output/Result**: System adapts to different display types and resolutions without configuration issues.
- **How the test will be performed**: Review system adaptability for compatibility across multiple screen sizes and types.

3.2.13 Security Requirements

Test for Access Controls

Test ID: **NFR-SC-01**

- **Type**: Nonfunctional, Security, Dynamic
- **Initial State**: System with user roles configured.
- **Input/Condition**: Attempt to access restricted areas with various user roles.
- **Output/Result**: Access is restricted based on role, and data transmission is encrypted.
- **How the test will be performed**: Perform tests with various user roles to ensure role-based access is functioning and verify encrypted data transmission.

Test for Data Integrity

Test ID: **NFR-SC-02**

- **Type**: Nonfunctional, Security, Dynamic
- **Initial State**: System with data input forms active.
- **Input/Condition**: Enter data with potential errors and attempt data modification.
- **Output/Result**: System validates input and maintains data integrity.
- **How the test will be performed**: Validate input handling and test version control to ensure data integrity.

Test for Privacy Compliance

Test ID: **NFR-SC-03**

- **Type**: Nonfunctional, Security, Inspection
- **Initial State**: System privacy policy and consent mechanisms active.
- **Input/Condition**: Review privacy policy, user consent, and data encryption methods.
- **Output/Result**: System adheres to privacy regulations and secures user data.
- **How the test will be performed**: Review system compliance with GDPR and other data privacy standards.

Test for Audit Log Integrity

Test ID: **NFR-SC-04**

- **Type**: Nonfunctional, Security, Inspection
- **Initial State**: System audit logs enabled.
- **Input/Condition**: Access and attempt modification of audit logs.
- **Output/Result**: Audit logs are tamper-proof and alert on suspicious activity.

- **How the test will be performed**: Test audit log integrity by reviewing storage, access controls, and tamper detection.

Test for Immunity Against Cyber Threats

Test ID: **NFR-SC-05**

- **Type**: Nonfunctional, Security, Penetration Testing
- **Initial State**: System in live environment.
- **Input/Condition**: Simulate cyber-attacks like SQL injection, XSS, and DDoS.
- **Output/Result**: System resists common cyber threats and maintains integrity.
- **How the test will be performed**: Conduct penetration testing to identify and mitigate potential vulnerabilities.

3.2.14 Cultural Requirements

Test for Language Support

Test ID: **NFR-CU-01**

- **Type**: Nonfunctional, Usability Test, Manual
- **Initial State**: Interface displays names with special characters or accents.
- **Input/Condition**: Load names with diverse linguistic characters.
- **Output/Result**: System displays names accurately in English with proper character handling.
- **How the test will be performed**: Verify that names with special characters or accents display correctly in the English-language interface.

3.2.15 Compliance Requirements

Test for Legal Compliance

Test ID: **NFR-CO-01**

- **Type**: Nonfunctional, Static, Inspection
- **Initial State**: System using third-party images.
- **Input/Condition**: Check intellectual property compliance of third-party data usage.
- **Output/Result**: System adheres to intellectual property laws and has necessary licenses.

- **How the test will be performed**: Conduct a review of third-party data usage and licensing compliance.

Test for Standards Compliance

Test ID: **NFR-CO-02**

- **Type**: Nonfunctional, Static, Inspection
- **Initial State**: System with data protection and software quality measures.
- **Input/Condition**: Review system adherence to ISO/IEC 25010 and GDPR standards.
- **Output/Result**: System complies with ISO/IEC 25010 for software quality and GDPR for data privacy.
- **How the test will be performed**: Inspect system compliance with international standards and data privacy regulations.

3.3 Traceability Between Test Cases and Requirements

Requirement Category	Requirement	Test Case ID
9: Functional Requirements	FR1: Upload of digital graduate composites	FR-UP-01
	FR2: Store metadata with each composite	FR-MD-01
	FR3: Grant access to different features based on role	FR-AC-01
	FR4: View and zoom into individual profiles on composites	FR-VZ-01
	FR5: Allow searching for composites based on criteria	FR-SR-01
	FR6: Log user activities (logins, composite views, feedback submissions)	FR-LG-01
10.1: Appearance Requirements	NFR1: Display (high-res images with minimal pixelation)	NFR-LF-001
	NFR2:: Screen layout (consistent layout, accessible nav)	NFR-LF-002

10.2: Style Requirements	NFR3: Color Scheme (non-distracting)	NFR-ST-001
	NFR4:Font Choice (legible, scalable sans-serif)	NFR-ST-002
	NFR5: Iconography (intuitive, recognizable icons)	NFR-ST-003
	NFR6: Consistency (uniform style across elements)	NFR-ST-004
11.1: Usability and Humanity Requirements	NFR7: Touch Sensitivity (responsive to gestures)	NFR-US-001
	NFR8: Navigation (easy, intuitive with gestures)	NFR-US-002
11.3. Learning Requirements	NFR9: Minimal Learning Curve	NFR-LR-001
	NFR10: On-Screen Guidance (tooltips on first interaction)	NFR-LR-002
11.4 Understandability and Politeness	NFR11: Clarity (clear and concise instructions)	NFR-UP-001
	NFR12: Politeness (informative error messages)	NFR-UP-002
11.5 Accessibility Requirements	NFR13: Accessibility Features (large buttons, text)	NFR-AC-001
12.1 Performance Requirements	NFR14: Search Speed (1-2 seconds for results)	NFR-PR-001
	NFR15: Page Load Time (under 2 seconds)	NFR-PR-002
	NFR16: Smooth Animations (no lag in transitions)	NFR-PR-003
12.2 Safety-Critical Requirements	NFR17: Data Security (protection from unauthorized access)	NFR-SC-001
12.5 Capacity Requirements	NFR21: Concurrent Users	NFR-CP-001

12.6 Scalability or Extensibility	NFR22: Scalability (additional years without performance drop)	NFR-SE-001
	NFR23: Extensibility (future feature support)	NFR-SE-002
12.7 Longevity Requirements	NFR24: Maintenance (minimal required as system scales)	NFR-LG-001
13.1 Operational/Environmental	NFR25: Indoor Placement (durability for frequent use)	NFR-OE-001
	NFR26: Lighting (visible in varying light conditions)	NFR-OE-002
13.2 Wider Environment Requirements	NFR27: Building Infrastructure (minimal installation needs)	NFR-WE-001
	NFR28: Security (physical security from tampering)	NFR-WE-002
14.1 Maintainability and Support	NFR30: Maintenance (modular codebase, documentation)	NFR-MS-001
	NFR31: Supportability (detailed logs, FAQs, clear error messages)	NFR-MS-002
14.3 Adaptability Requirements	NFR32: Adaptability (supports hardware/software changes)	NFR-AD-001
15 Security Requirements	NFR33: Access Controls (role-based and encrypted)	NFR-SR-001
	NFR34: Integrity (input validation, version control)	NFR-SR-002
	NFR35: Privacy (GDPR compliance, data minimization)	NFR-SR-003
	NFR36: Audit (tamper-proof logs, integrity checks)	NFR-SR-004
	NFR37: Immunity (protection from cyber threats)	NFR-SR-005

16 Cultural Requirements	NFR38: Language Support (English with special characters)	NFR-CR-001
17 Compliance Requirements	NFR39: Legal Compliance (intellectual property, LifeTouch)	NFR-CO-001
	NFR40: Standards Compliance (ISO/IEC, GDPR)	NFR-CO-002

4 Unit Test Description

This section will be filled in after the MIS (detailed design document) has been completed. For now we do not have enough information to develop unit tests for our capstone as they will not be accurate and will change almost on a daily basis.

References

Hammad Pathan, Zahin Hossain, Willie Pai, Henushan Balachandran, Wajdan Faheem, https://github.com/PaisWillie/Digital-Composite/blob/main/docs/SRS-Volere/SRS.pdf, 2024

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

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

Hammad Pathan, Zahin Hossain, Willie Pai, Henushan Balachandran, Wajdan Faheem, https://github.com/PaisWillie/Digital-Composite/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf, 2024

5 Appendix

Appendix — Reflection

1. 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.

2. 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.

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

   Hammad - One skill that our team will collectively need to acquire to successfully complete this capstone project will be learning how to use Jest and how to set up automated tests.

   Wajdan - An essential skill for Wajdan would be understanding system architecture validation. This involves ensuring that all components work together effectively, which is key to verifying that the system structure meets project specifications and integrates seamlessly.

   Zahin - One skill our team will need to acquire is Dynamic Testing Knowledge. We need to conduct runtime tests ensuring feature functionality.

Henushan - A vital skill will be security testing, specifically knowledge in assessing privacy measures and vulnerability to security threats. This will ensure that the application not only protects user data but also complies with institutional privacy and security standards.

Willie - To successfully complete this capstone project, our team will need to know how to work with PyTest, the backend framework. We will need to understand how to create the tests and implement them effectively with our backend.

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

   Hammad - For Jest, one approach to this would be to search up videos and tutorials on youtube as well as going over its documentation to see how to set it up. The other approach would be to contact friends and acquaintances to have them show and teach me how to work the framework . I will choose the second approach because I can learn more in depth through direct visuals and watching someone else do it. This is the fastest way to learn and my friends and acquaintances are already familiar with the framework.

   Zahin - To build Dynamic Testing Knowledge, we can follow two approaches. First, we could take online courses or watch tutorial videos focused on dynamic testing techniques, including unit and integration testing, to understand best practices and common tools. Second, we could conduct hands-on testing sessions on small sample applications, allowing us to gain practical experience in a controlled environment. We'll focus on the tutorial approach. By taking online courses and watching tutorial videos on dynamic testing techniques, we can understand best practices and learn to use relevant tools at our own pace. This structured learning method ensures we cover essential concepts thoroughly, building a solid foundation that we can then apply to our project.

   Wajdan - To master system architecture validation, one approach is to use online resources like Udacity or Coursera courses that delve into architecture principles and validation methods. A second approach is to read blogs rather than a strict regimented course, which serves more like a crash course than a complete guide. The team will choose the second option due to time constraints in the schedules of most members.

Henushan - For security testing skills, one approach is to complete a certification or course on cybersecurity fundamentals, which will cover essential security testing techniques. Another approach is to use websites that help teach these skills through activities like games and competitions. The team will be opting to do the activity based learning because a full certificate or course will take too much time out of everyone's schedule.

Willie - To acquire the skill of using PyTest, one approach would be to research any of the widely available online resources on the topic. Where tutorial videos and documentation is readily available to learn from, I will be able to understand how to develop and implement the tests for our backend. Another approach would be to learn by practice, attempting to develop smaller-scale tests for less complicated backends I have created in the past; until I am more comfortable with the knowledge, so I can apply it towards the success of our capstone project.