System Verification and Validation Plan for MacONE

Team #5, Overwatch League
Jingyao, Qin
Qianni, Wang
Qiang, Gao
Chenwei, Song
Shuting, Shi

April 3, 2024

Revision History

Date	Version	Notes
Nov 3, 2023	1.0	Initial Draft
April 2, 2024	2.0	Final Version

Contents

List of Tables

1 Symbols, Abbreviations, and Acronyms

symbol	**description**
UI	User Interface
ML	Machine Learning
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
PDF	Portable Document Format
.csv	Comma-Separated Values
.txt	Text file

This document outlines the Verification and Validation (V&V) plan for the MacONE project developed by Team #5, Overwatch League. The V&V plan is a critical component of our project management and quality assurance processes, ensuring that MacONE not only meets its specified requirements but also fulfills the needs and expectations of its users and stakeholders.

Roadmap The V&V plan is structured as follows:

1. **Symbols, Abbreviations, and Acronyms**
2. **General Information**
3. **Plan**
4. **System Test Description**
5. **Unit Test Description**

2 General Information

2.1 Summary

MacONE is a tailored online platform aimed at streamlining and enhancing the McMaster student experience. It enables students to directly extract and integrate course information into their schedules, creating an individualized to-do list. Additionally, MacONE provides easy access to vital university resources, a forum for student interaction, a Pomodoro timer for effective study sessions, a feedback box for users to share their insights directly with developers, and a function allowing students to calculate their cumulative GPA by uploading their transcripts, all designed to simplify student life at McMaster University.

2.2 Objectives

This VnV plan would direct us through the testing process to ensure that our system would accomplish all the intended functions correctly and efficiently. Adequate tests would be planned according to functional and non-functional requirements stated in our SRS to ensure our project exhibits adequate qualities and functions. We would prioritize and polish core functions including PDF unloading and analysis, integration between task management and Pomodoro, and those associated non-functional requirements. If time allows, we would also aim to develop tests for AI attention detection and the social component.

2.3 Relevant Documentation

2.3.1 Development plan

As part of the development process, the writing of this VnV plan implementation of all the test cases shall follow the development process specified in the development plan.

2.3.2 SRS

The functional and non-functional requirements and their corresponding fit criteria would lead the test plan and be used as a guideline to make sure tests could cover all the documented requirements.

2.3.3 Hazard Analysis

Additional requirements related to software failures should also be covered in testing.

2.3.4 VnV Report

The result from executing this VnV plan would be recorded in the VnV report. The VnV plan could be revised from the results.

3 Plan

This section outlines the comprehensive strategy for verifying and validating the MacONE software, ensuring it aligns with specified requirements and design standards. The plan spans from team roles in verification to the utilization of various testing and verification tools.

3.1 Verification and Validation Team

Name	Role and Specific Duties
Jinyao Qin	**Lead Verifier**: Oversees the entire process, coordinates with other team members, and ensures all verification steps are followed diligently.
Qianni Wang	**Implementation Specialist**: Reviews the codebase to ensure it aligns with the documented requirements, also verifies the code's functionality, performance, and security aspects.
Qiang Gao	**Implementation Specialist**: same as Qianni Wang
Chenwei Song	**Manual Test Engineer**: Responsible for manual test cases, ensuring that all tests run in different environments, and reporting the results in an understandable format for the team.
Shuting Shi	**Test Automation Engineer**: Responsible for automating test cases, ensuring that all tests run in different environments, and reporting the results in an understandable format for the team.

Table 1: Verification and Validation Team Members and Their Roles

3.2 SRS Verification Plan

For the verification of the Software Requirements Specification (SRS) document, the following approaches will be adopted:

1. **Peer Review:** The SRS will be reviewed by team members and classmates to identify any inconsistencies, ambiguities, or missing requirements.
2. **Expert Review:** Experts in software development will be consulted to ensure the requirements are complete and feasible.
3. **Supervisor Review:** The SRS will be reviewed by our supervisor, who can provide valuable insights from a strategic and technical perspective.
4. **Client Feedback:** The document will be shared with the client or stakeholders for their feedback, ensuring alignment with their expectations and needs.

5. **Automated Analysis Tools:** Tools such as requirement management software will be used for tracing and managing requirements systematically.

Additionally, an SRS checklist will be utilized to systematically verify the content of the SRS document: Please see our SRS.pdf for more details.

3.3 Design Verification Plan

The design verification for our project will focus on ensuring that the design is user-friendly, intuitive, and aligns with the architectural requirements specified in the SRS. The verification plan will include the following key activities:

1. **Peer Reviews:** The design documents and models will be reviewed by team members and classmates to critique and provide feedback on the design's usability, intuitiveness, and adherence to architectural requirements.
2. **Supervisor Review:** The design will be presented to the project supervisor for a thorough review, focusing on adherence to technical specifications and project objectives.
3. **Design Walkthroughs:** Scheduled sessions where the design team presents the design to the stakeholders, including peers and supervisors, for feedback and suggestions.
4. **Prototype Testing:** Early versions of the design will be tested to gather quick feedback on the design's effectiveness and user experience.
5. **Consistency Check:** Ensuring that the design remains consistent with the requirements and objectives outlined in the SRS document.

To comprehensively verify the design, the following checklist will be used:

1. **Design Documentation Review:**
    - Check if the design documentation is complete and clearly describes the architecture, components, and interfaces.
    - Ensure that the design aligns with the project's objectives and requirements specified in the SRS.
2. **User Interface (UI) and User Experience (UX) Evaluation:**

- Verify that the UI design is intuitive and user-friendly.
- Ensure UI consistency across different parts of the application.
- Assess the UX for compliance with common usability standards and practices.

3. **Architectural Conformity:**
   - Confirm that the system architecture supports all the required functionalities.
   - Check for scalability, maintainability, and flexibility of the design.

4. **Performance and Security Review:**
   - Ensure that the design incorporates adequate performance optimizations.
   - Review the design for potential security vulnerabilities and data protection measures.

5. **Compliance with Standards:**
   - Verify adherence to relevant industry and design standards.

6. **Feedback Integration:**
   - Check that feedback from previous reviews (by classmates, peers, or stakeholders) has been adequately incorporated into the design.

3.4 Verification and Validation Plan Verification Plan

The verification and validation (V&V) plan for our project includes ensuring the integrity and effectiveness of the V&V processes themselves. Given the importance of this plan in the overall project quality assurance, the following approaches will be employed:

1. **Peer Review:** The V&V plan will be reviewed by team members and classmates to identify any omissions or areas needing improvement.

2. **Mutation Testing:** This technique will be applied to evaluate the ability of our test cases to detect faults deliberately injected into the code.

3. **Iterative Feedback Incorporation:** Feedback from all review sessions and testing phases will be systematically incorporated to refine the V&V plan.

To systematically verify the V&V plan, the following checklist will be used:

- Is the plan comprehensive, covering all aspects of software verification and validation?
- Are the responsibilities and roles in the V&V process clearly defined?
- Does the plan include a variety of testing methods (e.g., unit testing, integration testing, system testing)?
- Is there a clear process for incorporating feedback and continuous improvement in the V&V process?
- Are there criteria defined for the success of each testing phase?
- Is mutation testing included to assess the thoroughness of the test cases?
- Are there measures in place to track and resolve any identified issues during the V&V process?
- Does the plan align with the project's schedule, resources, and constraints?

3.5 Implementation Verification Plan

The Implementation Verification Plan will ensure that the software implementation adheres to the requirements and design specifications outlined in the SRS. Key components of this plan include:

- **Unit Testing:** A comprehensive suite of unit tests, as detailed in the project's test plan, will validate individual components or modules of the software. /textitPyTest, a flexible and powerful testing tool, will be used for writing and executing these tests.
- **Static Analysis:** /textitPylint and /textitFlake8 will be employed for static code analysis to identify potential bugs, security vulnerabilities, and issues with code style and complexity.
- **Code Reviews and Walkthroughs:** Regularly scheduled code reviews and walkthroughs with team members and supervisors to inspect code quality, readability, and adherence to the Flask framework's best practices and design patterns.

- **Continuous Integration:** Automated build and testing processes will be implemented using tools like GitHub Actions, to ensure continuous code quality, integration, and deployment.
- **Performance Testing:** The use of tools like Locust for load testing will help evaluate the application's performance under various conditions, particularly focusing on how the Flask application handles concurrent requests and data processing.

3.6 Automated Testing and Verification Tools

For automated testing and verification in our Flask/Python project, the following tools will be employed:

- **Unit Testing Framework:** /textitPyTest will be used for developing and running unit tests.
- **Profiling and Performance Tools:** Tools like /textitcProfile for Python will assist in identifying performance bottlenecks and optimizing code efficiency.
- **Static Code Analyzers:** /textitPylint and /textitFlake8 will be used to analyze Python code quality, adherence to coding standards, and identification of potential errors.
- **Continuous Integration:** GitHub Actions will automate the build, testing, and deployment process, ensuring continuous integration and delivery of the Python codebase.
- **Linters:** /textitFlake8 will be used to enforce coding standards.

3.7 Software Validation Plan

The Software Validation Plan will focus on ensuring that the final product meets the requirements and expectations of the stakeholders. Key strategies include:

- **Beta Testing:** Involvement of selected users in the beta testing phase to provide real-world feedback on the software's functionality and usability.
- **Stakeholder Review Sessions:** Regular review meetings with stakeholders to confirm that the software meets the intended requirements and use cases.

- **Demo to Supervisor:** A demonstration of the software to the project supervisor following the Rev 0 demo for feedback and validation.
- **Reference to SRS Verification:** Aligning the validation activities with the SRS verification efforts to ensure consistency in meeting the documented requirements.

4 System Test Description

4.1 Tests for Functional Requirements

1. TFR1-EXTRACT1

   Type: Functional, Manual, Dynamic

   Initial State: Tester is on the course information extraction page.

   Input/Condition: Tester uploads their course syllabi in PDF format.

   Output/Result: The system extracts key details such as professor's email addresses, assignment weightings, and MSAF policy, displaying them to the tester.

   How test will be performed: A tester uploads multiple course syllabi PDFs, and the system's ability to accurately extract and display the specified information is evaluated.

2. TFR2-TASK1

   Type: Functional, Manual, Dynamic Initial State: The system has successfully extracted course information from an uploaded course outline.

   Input/Condition: Course information is processed by the system.

   Output/Result: Tasks are automatically created and added to the tester's to-do list based on the extracted course information.

   How test will be performed: Course outlines are uploaded, and the automatic generation of tasks in the to-do list is verified for correctness and completeness.

3. TFR3-ADD1

   Type: Functional, Manual, Dynamic

Initial State: Tester is on the course addition page.

Input/Condition: Tester inputs a valid course code to add a new course.

Output/Result: The course is successfully added to the tester's profile.

How test will be performed: Tester attempt to add various courses by providing valid course codes, and the system's response to correctly add those courses is assessed.

4. TFR4-REMOVE1

   Type: Functional, Manual, Dynamic

   Initial State: Tester's profile contains one or more courses.

   Input/Condition: Tester selects a course to remove from their profile.

   Output/Result: The selected course is removed from the tester's profile.

   How test will be performed: Tester with multiple courses on their profile attempts to remove one, and the successful removal and profile update are verified.

5. TFR5-POMODORO1

   Type: Functional, Manual, Dynamic

   Initial State: Tester is on the Pomodoro timer page.

   Input/Condition: Tester initiates a study session using the Pomodoro timer.

   Output/Result: A Pomodoro study session starts.

   How test will be performed: Tester navigates to the Pomodoro page and starts a study session, verifying the timer's functionality and session tracking.

6. TFR6-POMODORO2

   Type: Functional, Manual, Dynamic

   Initial State: Tester's to-do list contains at least one task.

   Input/Condition: Tester starts a Pomodoro timer by selecting a specific task from the to-do list. Output/Result: A Pomodoro study session specific to the selected task begins.

How test will be performed: Tester selects a task from their to-do list to start a Pomodoro session, and the system's ability to associate the session with the selected task is evaluated.

7. TFR7-MUSIC1

   Type: Functional, Manual, Dynamic

   Initial State: A Pomodoro study session is active.

   Input/Condition: Tester selects to play background music during the study session.

   Output/Result: Chosen background music plays during the Pomodoro session.

   How test will be performed: During an active Pomodoro session, the tester selects different music options to play in the background, and the system's ability to play and manage music selection is assessed.

8. TFR8-TOMATO1

   Type: Functional, Manual, Dynamic

   Initial State: A Pomodoro study session has just been completed.

   Input/Condition: Completion of a Pomodoro study session.

   Output/Result: A tomato icon is generated and displayed in the tester's "harvest bucket".

   How test will be performed: After completing a Pomodoro session, the presence of a new tomato icon in the harvest bucket is verified to assess the system's reward mechanism.

9. TFR9-TODO11

   Type: Functional, Manual, Dynamic

   Initial State: Tester is on the to-do list page. Input/Condition: Tester adds a new task to the to-do list.

   Output/Result: The new task appears on the to-do list.

   How test will be performed: Tester attempts to add a task by specifying task details, and the addition of the task to the list is confirmed.

10. TFR10-TODO2

    Type: Functional, Manual, Dynamic

    Initial State: Tester's to-do list contains at least one task.

    Input/Condition: Tester selects a task to view its details.

    Output/Result: Details of the selected task are displayed, including due date, associated course, weight, and other relevant information.

    How test will be performed: Tester clicks on various tasks to view their details, verifying that all relevant information is accurately presented.

11. TFR11-TODO3

    Type: Functional, Manual, Dynamic

    Initial State: Tester's to-do list contains at least one task.

    Input/Condition: Tester selects an existing task to edit in the to-do list.

    Output/Result: The task is updated with new information as provided by the tester.

    How test will be performed: Tester edits multiple tasks with different pieces of information, confirming that each change is reflected in the to-do list.

12. TFR12-TODO4

    Type: Functional, Manual, Dynamic

    Initial State: Tester's to-do list contains several tasks.

    Input/Condition: Tester views the to-do list.

    Output/Result: Tasks are organized and displayed in three separate columns: To Do, In Progress, and Done.

    How test will be performed: Tester verifies the organization of tasks into the specified columns and assesses the ease of understanding the task's status.

13. TFR13-TODO5

    Type: Functional, Manual, Dynamic

    Initial State: Tester is on the to-do list page.

    Input/Condition: Tester toggles between Kanban-style and calendar layout views.

Output/Result: The to-do list is displayed according to the selected layout preference.

How test will be performed: Tester switches between the two layout options multiple times to ensure the layout changes are applied correctly and persistently.

14. TFR14-FORUM1

    Type: Functional, Manual, Dynamic

    Initial State: Tester is on the forum page.

    Input/Condition: Tester creates a new discussion topic.

    Output/Result: The new topic is successfully created and visible in the forum.

    How test will be performed: Tester attempts to create multiple discussion topics with different themes, verifying that each is added to the forum.

15. TFR15-FORUM2

    Type: Functional, Manual, Dynamic

    Initial State: Forum contains multiple discussion topics.

    Input/Condition: Tester searches for a topic using keywords.

    Output/Result: Topics matching the keywords are displayed to the tester.

    How test will be performed: Tester performs several searches using different keywords, and the accuracy and relevance of search results are evaluated.

16. TFR15-FORUM3

    Type: Functional, Manual, Dynamic

    Initial State: Tester is viewing a specific discussion topic in the forum.

    Input/Condition: Tester submits a comment on the topic.

    Output/Result: The comment is added to the discussion.

    How test will be performed: Tester comments on various topics and verifies that their comments are correctly posted and visible to others.

17. TFR17-FORUM4

    Type: Functional, Manual, Dynamic

    Initial State: Tester is viewing a comment within a discussion topic in the forum.

    Input/Condition: Tester replies to a comment. Output/Result: The reply is posted as part of the discussion thread.

    How test will be performed: Tester replies to multiple comments in different discussion threads, checking for the proper organization and visibility of their replies.

18. TFR18-QUICKLINKS1

    Type: Functional, Manual, Dynamic

    Initial State: Tester is on the dashboard or home page.

    Input/Condition: Tester navigates to the quick links section.

    Output/Result: The system displays key university websites and other relevant tools.

    How test will be performed: Tester checks the quick links section for accessibility and relevance of the provided links to university websites and other educational tools.

19. TFR19-QUICKLINKS2

    Type: Functional, Manual, Dynamic

    Initial State: Quick links are displayed on the tester's dashboard or home page.

    Input/Condition: Tester attempts to drag and rearrange the order of website links.

    Output/Result: The order of the links changes according to the tester's customization.

    How test will be performed: Tester rearranges the quick links in a new order, verifies the order is saved and persists across sessions.

20. TFR20-FEEDBACK1

    Type: Functional, Manual, Dynamic

    Initial State: Tester is logged in and on the feedback submission page.

    Input/Condition: Tester submits feedback through the feedback form.

    Output/Result: Feedback is successfully submitted to the developers.

    How test will be performed: Tester fills out and submits the feedback form with various types of feedback, confirming submission success and receipt by the development team.

21. TFR21-FEEDBACK2

    Type: Functional, Manual, Dynamic

    Initial State: Tester has previously submitted feedback.

    Input/Condition: Tester navigates to the feedback history page.

    Output/Result: The system displays all past feedback submitted by the tester.

    How test will be performed: Tester reviews their feedback history to ensure all previously submitted feedback is visible and accurate.

22. TFR22-FEEDBACK3

    Type: Functional, Manual, Dynamic

    Initial State: Tester has submitted feedback and is on the feedback status page.

    Input/Condition: Tester checks the status of their submitted feedback.

    Output/Result: The system shows the current status of each piece of feedback submitted by the tester.

    How test will be performed: Tester submits feedback and later checks the status of their submissions, verifying that the system accurately tracks and displays the status.

23. TFR23-PROFILE1

    Type: Functional, Manual, Dynamic

    Initial State: Tester is on their profile settings page.

Input/Condition: Tester attempts to change their username.

Output/Result: The system updates the tester's username across the platform.

How test will be performed: Tester changes their username in profile settings, then navigates the platform to ensure the new username is displayed and used consistently.

24. TFR24-GPA1

    Type: Functional, Manual, Dynamic

    Initial State: Tester is on the GPA calculation page.

    Input/Condition: Tester upload the transcript to calculate their cumulated GPA.

    Output/Result: The system calculates and displays the tester's cumulated GPA.

    How test will be performed: upload the transcript, checking for accuracy of the calculated GPA against manual calculations.

4.2 Tests for Nonfunctional Requirements

4.2.1 Look and feel

(a) TAR-1

    Type: Dynamic, Manual

    Initial State: The web application is launched.

    Input/Condition: Tester are asked to read the text on each page.

    Output/Result: `MIN_UNDERSTAND%` of tester reporting that all the texts are legible.

    How test will be performed: The web application displayed with screens with various resolutions and sizes. `MIN_TESTER_NUM` Tester are asked to read the texts and report any unclear typography.

(b) TAR-2

    Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to observe color on each page.

Output/Result: `MIN_UNDERSTAND%` of testers comfortable with the color used.

How test will be performed: `MIN_TESTER_NUM` Tester are asked to observe the color and report any distracting or overwhelming color palette.

(c) TAR-3

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to interpret icons and graphics on each page.

Output/Result: `MIN_UNDERSTAND%` of tester able to understand the meaning of each icon and graphic as designed.

How test will be performed: `MIN_TESTER_NUM` Tester are asked what they think each icon and graphic means. Any mismatch from the UI designer's intention would be recorded.

(d) TAR-4

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to observe the layout of each page.

Output/Result: `MIN_UNDERSTAND%` of tester comfortable with the payout.

How test will be performed: The web application is launched with different devices including smartphones, tablets, and desktops. `MIN_TESTER_NUM` Tester are asked to point out any unresponsive, out-of-margin, or compressed elements.

(e) TSTR-1

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to point out the menus and navigation bars.

Output/Result: `MIN_UNDERSTAND%` of tester able to locate menus and navigation tools without confusion.

How test will be performed: `MIN_TESTER_NUM` Tester are asked to point out the menus and navigation bars without assistance.

(f) TSTR-2

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to point out inconsistent UI elements.

Output/Result: `MAX_ELEMENT_BAD` inconsistent UI elements found.

How test will be performed: `MIN_TESTER_NUM` Tester are asked to explore the application and point out any inconsistent UI elements that break the harmony.

(g) TSTR-3

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to click each interactive UI element.

Output/Result: `MAX_ELEMENT_BAD` interactive UI elements being unresponsive.

How test will be performed: `MIN_TESTER_NUM` Tester are asked to explore the application and report any unresponsive interactive elements.

(h) TSTR-4

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to adjust font size.

Output/Result: `MAX_UNSATISFIED%` of tester reporting unsuitable font size.

How test will be performed: `MIN_TESTER_NUM` Tester are asked to report how they like the default font size and attempt to adjust the font size to their preference. Report if the adjusted font still causes difficulty in reading.

(i) TSTR-5

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to perform a task while an animation is displayed.

Output/Result: `MIN_UNDERSTAND%` of tester who could perform the task distracted.

How test will be performed: `MIN_TESTER_NUM` Tester are asked to perform a task that involves the animation display part of the screen. They are asked to report if they feel the animation is intrusive or distracting.

4.2.2 Usability and Humanity

(a) TUHR-1

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to navigate to a required page.

Output/Result: `MIN_UNDERSTAND%` of tester able to navigate to a required page without confusion.

How test will be performed: In an obeservational study, `MIN_TESTER_NUM` Tester are asked to navigate to a required page without assistance within a reasonable amount of time undergoing minimum trial and error.

(b) TUHR-2

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Testers are asked to perform a series of specified core functions, including:

i. Navigating to and uploading a course syllabus in PDF format.
ii. Starting a Pomodoro study session from the dashboard.
iii. Posting a message in the forum section.

Output/Result: `MIN_UNDERSTAND%` of testers are able to perform each specified tasks without confusion or significant difficulty.

How the test will be performed: In an observational study, `MIN_TESTER_NUM` testers are given a sequence of tasks that encompass the specified core functions. They are observed as they attempt to complete each task within a reasonable amount of time, with the goal of undergoing minimum trial and error. The observer records the ease with which each tester completes the tasks, noting any areas of confusion or difficulty. This data is then used to evaluate the usability and intuitiveness of the web application's core functionalities.

(c) TUHR-3

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to inspect the grammar of texts on the page.

Output/Result: `MAX_BAD_GRAMMAR` grammar mistakes found.

How test will be performed: In an obeservational study, `MIN_TESTER_NUM` Tester are asked to report grammar mistakes in all the text visible in this web application.

(d) TUHR-4

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to inspect for offensive messages on the page.

Output/Result: `MAX_OFFENSIVE` offensive messages found.

How test will be performed: In an obeservational study, `MIN_TESTER_NUM` Tester are asked to report offensive messages in all the text visible in this web application.

(e) TUHR-5

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to observe color combinations on the page.

Output/Result: `MAX_COLOR_AMBIGUOUS` indistinguishable color combinations found.

How test will be performed: In an obeservational study, `MIN_TESTER_NUM` Tester are asked to report indistinguishable color combinations in this web application. This user group should include people with color blindness.

4.2.3 Performance

(a) TSLR-1

Type: Dynamic, Automatic

Initial State: The web application is launched.

Input/Condition: The processing time for a core function.

Output/Result: It takes `MAX_TIME_PROCESS` for the application to finish a core function.

How test will be performed: An automated script is used to perform and time a core function, including uploading syllabuses, generating tasks, and prioritizing tasks.

4.2.4 Safety-Critical

(a) TSCR-1

Type: Dynamic, Automatic

Initial State: The web application is launched.

Input/Condition: A pair of new credentials are added.

Output/Result: Format of credentials in database.

How test will be performed: An automated script is used to add a pair of new credentials and check whether plain text is stored in the database.

4.2.5 Precision and Accuracy

(a) TPAR-1

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Course outlines in PDF are uploaded

Output/Result: `MIN_PRECISION%` of coincidence between algorithm result and human result.

How test will be performed: A manual test is performed by one of our developers to upload multiple course outlines and manually record the percentage of course information that are correctly extracted.

4.2.6 Robustness and Fault-Tolerance

(a) TRFTR-1

Type: Dynamic, Automated

Initial State: The web application is launched.

Input/Condition: Incorrect tester input.

Output/Result: `MIN_OPERABLE%` of the system operating.

How test will be performed: An automated script is used to put false input in every interactive element, including wrong file format, special characters, out-of-boundary data, and malicious commands.

4.2.7 Capacity

(a) TCR-1

Type: Dynamic, Automated

Initial State: The web application is launched.

Input/Condition: Massive course data input.

Output/Result: It takes `MAX_TIME_PROCESS` to finish a core function.

How test will be performed: An automated script is used to inject a large amount of course data. Run TSLR-5.

4.2.8 Operational and Environmental

(a) TOER-1

Type: Dynamic, Automated

Initial State: The web application is launched.

Input/Condition: Send requests to supported calendar APIs.

Output/Result: `MIN_API_SUCCESS%` of successful API requests.

How test will be performed: An automated script is used to send requests to calendar APIs and record the result.

(b) TOER-2

Type: Dynamic, Automated

Initial State: The web application is launched.

Input/Condition: Run regression test suites.

Output/Result: MIN_REGRESSION_PASS% of passed regression tests.

How test will be performed: An automated script is used to run regression tests.

4.2.9 Maintainability and Support

(a) TSPR-1

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to contact support.

Output/Result: MAX_SUPPORT_STEP steps needed to reach help.

How test will be performed: MIN_TESTER_NUM Tester are asked to reach the helpdesk through email, phone, and chatbot and record the steps it takes from the main page to help.

(b) TSPR-2

Type: Dynamic, Automatic

Initial State: The web application is launched.

Input/Condition: Tester feedback is inputted.

Output/Result: Tester input in database.

How test will be performed: A script is used to input tester feedback and check if this feedback gets stored in the database.

4.2.10 Security

(a) TSR-1

Type: Dynamic, Manual

Initial State: The web application is launched.

Input/Condition: Tester are asked to modify the database.

Output/Result: MAX_ATTACK_SUCCESS Tester successfully modified the database.

How test will be performed: MIN_TESTER_NUM Tester are asked to taint the database as unauthorised entity.

(b) TSR-2

Type: Dynamic, Automatic

Initial State: The web application is launched.

Input/Condition: A series of specific tester actions, including but not limited to, logging in, uploading a document, editing a profile, posting on a forum, and deleting an item from the to-do list.

Output/Result: An audit log containing records of these actions.

How test will be performed: A script is used to automatically perform a predefined series of tester actions on the web application. These actions include:

- Uploading a document (e.g., a PDF syllabus) to the application.
- Editing the tester's user profile information.
- Posting a message on the forum section of the application.
- Adding a new task to the to-do list and subsequently deleting it.

After these actions are performed, the script will access the application's audit log to verify that each action is recorded with accurate timestamps and relevant metadata, such as the user ID, action type, and any pertinent details related to the action (e.g., the name of the uploaded document, the updated profile information, the content of the forum post, and the details of the added and deleted tasks).

4.2.11 Complience

(a) TCPR-1

Type: Dynamic, Manual

Initial State: Development container is launched.

Input/Condition: A PR containing local changes is pushed.

Output/Result: Workflow result.

How test will be performed: A PR containing local changes is pushed, which triggers a workflow linter check.

4.3 Traceability Between Test Cases and Requirements

All requirements refer to SRS.pdf.

Test Case #	Requirement #																							
	AR1	AR2	AR3	AR4	STR1	STR2	STR3	STR4	STR5	UHR1	UHR2	UHR3	UHR4	UHR5	SLR1	SCR1	PAR1	RFTR1	CR1	CR2	SER1	OER1	OER2	OER3
TAR-1	X																							
TAR-2		X																						
TAR-3			X																					
TAR-4				X																				
TSTR-1					X																			
TSTR-2						X																		
TSTR-3							X																	
TSTR-4								X																
TSTR-5									X															
TUHR-1										X														
TUHR-2											X													
TUHR-3												X												
TUHR-4													X											
TUHR-5														X										
TSLR-1															X									
TSCR-1																X								
TPAR-1																	X							
TRFTR-1																		X						
TCR-2																				X				
TOER-1																						X		
TOER-2																							X	

Table 2: Traceability Matrix: Test Cases to Non-Functional Requirements

Test Case #	Requirement #																							
	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FR12	FR13	FR14	FR15	FR16	FR17	FR18	FR19	FR20	FR21	FR22	FR23	FR24
TFR1-EXTRACT1	X																							
TFR2-TASK1		X																						
TFR3-ADD1			X																					
TFR4-REMOVE1				X																				
TFR5-POMODORO1					X																			
TFR6-POMODORO2						X																		
TFR7-MUSIC1							X																	
TFR8-TOMATO1								X																
TFR9-TODO1									X															
TFR10-TODO2										X														
TFR11-TODO3											X													
TFR12-TODO4												X												
TFR13-TODO5													X											
TFR14-FORUM1														X										
TFR15-FORUM2															X									
TFR16-FORUM3																X								
TFR17-FORUM4																	X							
TFR18-QUICKLINKS1																		X						
TFR19-QUICKLINKS2																			X					
TFR20-FEEDBACK1																				X				
TFR21-FEEDBACK2																					X			
TFR22-FEEDBACK3																						X		
TFR23-PROFILE1																							X	
TFR24-GPA1																								X

Table 3: Traceability Matrix: Test Cases to Functional Requirements

5 Unit Test Description

5.1 Unit Testing Scope

Unit testing will focus on individual units or components of the software to determine if they operate correctly. Each test case is designed based on a functional requirement from the SRS.

(a) UT-1

Objective: To verify that the system correctly extracts and displays key course information from uploaded syllabi.

Input: Tester uploads a course syllabus in PDF format.

Expected Output: Key details such as professor's email addresses, assignment weightings, and MSAF policy are extracted and displayed to the tester.

Requirement Reference: FR1 - Course Information Extraction.

(b) UT-2

Objective: To verify the system automatically creates tasks from extracted course information and adds them to the tester's to-do list.

Input: Course information is processed by the system after a syllabus upload.

Expected Output: Tasks are automatically created and added to the tester's to-do list based on the extracted course information.

Requirement Reference: FR2 - Task Creation from Course Information.

(c) UT-3

Objective: To verify that tester can successfully add a new course by providing a valid course code.

Input: Tester inputs a valid course code to add a new course.

Expected Output: The new course is added to the tester's profile.

Requirement Reference: FR3 - Add New Course.

(d) UT-4

Objective: To verify that tester can remove a course from their profile.

Input: Tester selects a course to remove from their profile.

Expected Output: The selected course is removed from the tester's profile.

Requirement Reference: FR4 - Remove Course.

(e) UT-5

Objective: To verify that the system allows tester to start a study session using the Pomodoro timer.

Input: Tester initiates a Pomodoro timer on the Pomodoro page.

Expected Output: A Pomodoro study session starts and is tracked by the system.

Requirement Reference: FR5 - Pomodoro Timer Start from Page.

(f) UT-6

Objective: To verify that tester can start a Pomodoro study session by selecting a specific task from the to-do list.

Input: Tester starts a Pomodoro timer by clicking a specific task in their to-do list.

Expected Output: A Pomodoro study session specific to the selected task begins.

Requirement Reference: FR6 - Pomodoro Timer Start from Task.

(g) UT-7

Objective: To verify that tester can select and play different music as background during their study session.

Input: Tester selects a music option to play during an active Pomodoro study session.

Expected Output: The selected music plays in the background during the study session.

Requirement Reference: FR7 - Music Selection for Pomodoro Session.

(h) UT-8

Objective: To verify the system generates and displays a tomato icon in the harvest bucket upon completion of a study session.

Input: Completion of a Pomodoro study session.

Expected Output: A tomato icon is generated and displayed in the tester's harvest bucket.

Requirement Reference: FR8 - Tomato Harvesting Post Study Session.

(i) UT-9

Objective: To verify that tester can add a task to their to-do list successfully.

Input: Tester adds a new task with details to their to-do list.

Expected Output: The new task is visible on the to-do list with the specified details.

Requirement Reference: FR9 - Adding a Task to the To-Do List.

(j) UT-10

Objective: To verify that tester can view details of a task including due date, associated course, and weight.

Input: Tester selects a task to view its details.

Expected Output: Detailed information about the task is displayed to the tester.

Requirement Reference: FR10 - Task Detail Viewing.

(k) UT-11

Objective: To verify that tester can edit an existing task in their to-do list.

Input: Tester edits an existing task by changing its details.

Expected Output: The task is updated with the new information provided by the tester.

Requirement Reference: FR11 - Editing an Existing Task.

(l) UT-12

Objective: To verify that tasks are organized into 'To Do', 'In Progress', and 'Done' columns.

Input: Viewing the to-do list with multiple tasks in different stages.

Expected Output: Tasks are correctly organized and displayed in their respective columns.

Requirement Reference: FR12 - Task Organization in Columns.

(m) UT-13

Objective: To verify the functionality of creating a new topic in the forum for discussion.

Input: Tester submits a new discussion topic with title and description.

Expected Output: The new topic is successfully created and visible in the forum.

Requirement Reference: FR14 - Forum Topic Creation.

(n) UT-14

Objective: To verify that tester can search for topics in the forum using keywords.

Input: Tester enters keywords in the forum search field.

Expected Output: Topics matching the keywords are displayed to the tester.

Requirement Reference: FR15 - Forum Topic Search.

(o) UT-15

Objective: To verify that tester can comment on a specific topic in the forum.

Input: Tester submits a comment on a forum topic.

Expected Output: The comment is added to the discussion under the topic.

Requirement Reference: FR16 - Forum Topic Commenting.

(p) UT-16

Objective: To verify that tester can reply to a comment within a forum topic.

Input: Tester submits a reply to an existing comment.

Expected Output: The reply is correctly threaded under the original comment.

Requirement Reference: FR17 - Comment Replies in Forum.

(q) UT-17

    Objective: To verify that tester can access key university websites and other relevant tools from the quick links section.

    Input: Tester navigates to the quick links section on the dashboard.

    Expected Output: Key university websites and other relevant tools are displayed and accessible.

    Requirement Reference: FR18 - Quick Links Access.

(r) UT-18

    Objective: To verify that tester can customize the order of website links in the quick links section.

    Input: Tester drags and rearranges the order of quick links.

    Expected Output: The order of the links changes and persists according to tester customization.

    Requirement Reference: FR19 - Customizing Order of Quick Links.

(s) UT-19

    Objective: To verify that the system allows tester to submit feedback through the feedback form.

    Input: Tester fills out and submits the feedback form.

    Expected Output: Feedback is successfully submitted and received by the system.

    Requirement Reference: FR20 - Feedback Submission.

(t) UT-20

    Objective: To verify that tester can view their previously submitted feedback.

    Input: Tester navigates to the feedback history page.

    Expected Output: All past feedback submitted by the tester is visible and accurately displayed.

    Requirement Reference: FR21 - Viewing Past Feedback.

(u) UT-21

Objective: To verify that tester can check the status of their submitted feedback.

Input: Tester checks the status of submitted feedback in the feedback section.

Expected Output: The current status of each submitted feedback is accurately displayed.

Requirement Reference: FR22 - Checking Feedback Status.

(v) UT-22

Objective: To verify that the system allows tester to change their username through their profile settings.

Input: Tester changes their username in the profile settings.

Expected Output: The system updates the tester's username across the platform.

Requirement Reference: FR23 - Username Change.

(w) UT-23

Objective: To verify that the system allows tester to calculate their cumulative GPA by inputting grades and credit hours.

Input: Tester inputs their grades and credit hours on the GPA calculation page.

Expected Output: The system calculates and displays the tester's cumulative GPA.

Requirement Reference: FR24 - GPA Calculation.

5.2 Traceability Between Test Cases and Modules

Test Case #	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FR12	FR13	FR14	FR15	FR16	FR17	FR18	FR19	FR20	FR21	FR22	FR23	FR24
UT-1	X																							
UT-2		X																						
UT-3			X																					
UT-4				X																				
UT-5					X																			
UT-6						X																		
UT-7							X																	
UT-8								X																
UT-9									X															
UT-10										X														
UT-11											X													
UT-12												X												
UT-13													X											
UT-14														X										
UT-15															X									
UT-16																X								
UT-17																	X							
UT-18																		X						
UT-19																			X					
UT-20																				X				
UT-21																					X			
UT-22																						X		
UT-23																							X	
UT-24																								X

Table 4: Traceability Matrix: Unit Test Cases to Functional Requirements

6 Static Testing Plan

Static testing examine the software's code, documentation, and design without executing the program. The primary goal is to detect errors early in the development process, improve quality, and ensure compliance with coding standards and design guidelines.

6.0.1 Scope

The static testing in our project will focus on the following areas:

- **Code Review:** To identify syntax errors, dead code, security vulnerabilities, and deviations from coding standards.
- **Design Review:** To ensure the design documents accurately reflect the software requirements and align with the established design principles.
- **Documentation Review:** To verify that all documentation is clear, complete, and consistent with the software implementation and user expectations.

6.0.2 Walk-through

Facilitate a discussion around the documents or code to gather feedback and find errors. The author of the document walks the team through the content, explaining the logic and decisions. Participants ask questions and give feedback based on their understanding and expertise.

6.0.3 Inspection

Conduct a thorough examination of code and documents to detect and correct issues. An inspection team (different from the authors) conducts a detailed examination. The findings should be documented for correction.

6.0.4 Process Split Up

- Establish a review process for all major documents and code with clear guidelines on how reviews, walk-through, and inspections are conducted.

- Using specific tools like flake8 or using documentation/manual as reference.
- Integrate static analysis into the CI/CD pipeline to ensure continuous code quality and security checks.
- Monitor and iterate on the process, and continuously optimize the process.

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

parameter	value	unit	description
`MIN_UNDERSTAND%`	90	N/A	the minimum percentage of testers who can understand among all testers
`MAX_UNSATISFIED%`	5	N/A	the maximum percentage of testers reporting uncomfortable
`MAX_TRIAL_TIME`	1200	`s`	the maximum allowed trial time
`MIN_TESTER_NUM`	10	N/A	the minimum number of testers needed
`MAX_BAD_GRAMMAR`	0	N/A	the maximum occurrence of grammar mistakes allowed
`MAX_OFFENSIVE`	0	N/A	the maximum occurrence of offensive messages allowed
`MAX_ELEMENT_BAD`	0	N/A	the maximum occurrence of inconsistent or unresponsive elements allowed
`MAX_ATTACK_SUCCESS`	0	N/A	the maximum occurrence of successful attack

`MAX_COLOR_AMBIGUOUS`	0	N/A	the maximum occurrence of indistinguishable color combinations allowed
`MAX_TIME_PROCESS`	2	s	the maximum processing time for a core function
`MIN_PRECISION%`	90	N/A	the minimum precision of the algorithm
`MIN_OPERABLE%`	95	N/A	the minimum percentage of system being operable
`MIN_API_SUCCESS%`	95	N/A	the minimum percentage of successful API calls
`MIN_REGRESSION_PASS%`	100	N/A	the minimum percentage of successful API calls
`MAX_RESPONSE_TIME`	24	h	The maximum issue resolve response time
`MIN_LANGUAGE`	5	N/A	The minimum number of languages that can be translated
`MAX_SUPPORT_STEP`	5	N/A	The maximum steps needed for asking for support

7.2 Survey Questions

(a) Are you a student? If so, are you from McMaster?

(b) Which following current feature in the website attracts you the most?

(c) Please select from the options below that you think may be used less often?

(d) We are considering adding some new features to the website to improve your experience. Please select from the options below the features that you think will make you use our site more frequently.

(e) Is the content on our website clear and understandable?

(f) On a scale of 1 to 5, how easy was it to navigate our website overall?

(g) On a scale of 1 to 5, how would you rate the ease of the priority generation feature?

(h) On a scale of 1 to 5, how visually appealing do you find our website?

(i) Were you able to find all the current features we have? if not, please describe the problem you met.

(j) On a scale of 1 to 5, how would you rate the accuracy of the information provided on our website overall?

(k) On a scale of 1 to 5, how would you rate the task priority feature?

(l) have you encountered any other inaccuracies while using our website? if yes, please provide examples.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

(a) 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 etc. You should look to identify at least one item for each team member.

- UI/UX usability validation tools such as *UserTesting*, *Lookback.io.* to better evaluate our product is user-friendly in a couple of perspectives: effective, learnable, and user-friendly.
- Dynamic Testing Tools such as *Behave*, which is a tool that allows users to write the test cases in human languages to test for python-system framework.
- AI Model Validation Frameworks such as *Snitch AI* and *scikit-learn* which can help our trained morel enhance quality and troubleshoot quickly.
- Static Code Analysis Tools such as *SonarQube* to ensure the code quality which also can be integrated with *CI/CD* for continuous development
- Enhance continuous delivery/deployment by exploring the *Actions* features in *GitHub* Pro to build custom workflow pipeline.

(b) 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?

Knowledge or Skills	**Approaches**	**Assigned Team Member**	**Reason**
UI/UX Usability validation	Use *ChatGPT*, *Google*, watch online tutorials, or ask supervisor for help	Shuting, Shi	Working on the initial UI design, familiar with the key features and the components of website. Therefore, can detect the usability requirements of our target user groups and easy to make modifications accordingly
Dynamic Testing Tools	Use *ChatGPT*, *Google*, watch online tutorials	Qiang, Gao	Have the related experience in the previous co-op work terms, implemented similar functionality in previous project. Strong interest in the dynamic testing section.
AI Model Validation Framework	Use *ChatGPT*, *Google*, watch online tutorials	Qianni, Wang	Experience with many ML projects where these libraries are being used in AI programs and previous co-op work terms. Working on the model training, data-sets selection and integration, familiar with the model algorithm, easy to do modifications if encounters specific model bias.
Static Code Analysis Tools	Use *ChatGPT*, *Google*, watch online tutorials	Chenwei, Song	Experience in enhancing clean code in previous co-op work terms. Strong interest in the code analysis section.
GitHub Action Feature	Use *ChatGPT*, *Google*, and watch online tutorials	Jingyao, Qin	Strong interest in GitHub features, have related experience in the previous coop term, quick to hand on this technique.