Project Title: System Verification and Validation Plan for UnderTree

Team 22, Capstoners
Palanichamy Veerash
Kannammalil Kevin
Qureshi Eesha
Ahmed Faiq

April 5, 2023

1 Revision History

Date	Version	Notes
October 31	1.0	Created Project Template
November 1	1.1	Started filling in system tests
November 2	1.2	Finished system tests
November 2	1.3	Finished rest of documentation aside from unit tests
March 1	1.4	Added unit tests
March 3	1.5	Modified unit tests to remove GitHub integration related features
April 5	1.5	Final document changes

Contents

List of Tables

2 Symbols, Abbreviations and Acronyms

Please take a look at the naming convention and terminology section of the SRS.

This document outlines the testing approach for the validation and verification of UnderTree. The document begins with an introduction to the verification and validation team, then a brief overview of the plans that will be followed to verify the SRS, design, and implementation of the project. The tools that will be used for automated testing are also discussed. This is followed by a section detailing functional and non-functional system test cases that are grouped by component. Finally the unit testing strategy is discussed.

3 General Information

3.1 Summary

This document covers the test plan for UnderTree, a collaborative LaTeX editing and compiling software that supports version control with Git integration. The software will allow for multiple members of a team to view, edit and compile the same **tex** files at the same time while seeing the changes made by others in real time. This is especially useful for teams preparing project documentation where multiple members need to contribute to the documents in parallel, however it is widely intended for the development of any LaTeX document by any number of contributors for any reason.

The LaTeX editor will support version control management through Git integration so that the version history of any document created will be easily accessible. This is especially useful for documents that are likely to change over time due to revisions, such as software documentation and release notes

3.2 Objectives

The objective of this test plan is to outline the test cases that will build confidence in the correctness of the software and verify that the intended requirements are met as defined in the SRS, as well as to ensure that the requirements to be followed are correct..

3.3 Relevant Documentation

The following documents are referenced in this test plan:

- Software Requirements Specification (SRS)
- MIS (to be added)

- MG (to be added)

Additionally it may be useful to take a look at the Hazard Analysis document to get a greater understanding of the modules, boundaries, and constraints of this system.

4 Plan

This section will introduce the Verification and Validation team, followed by summaries of the SRS, design, implementation and software verification plans, as well as an outline of testing tools that will be used.

4.1 Verification and Validation Team

The verification and validation team for this projects consists of the developers Kevin Kannammallil, Veerash Palanichamy, Eesha Qureshi and Faiq Ahmed as well as the Capstone course instructor Dr.Spencer Smith and his instructional assistant Samuel Crawford.The developers will be in charge of the quality assurance engineering of the software and will develop, implement and verify test suites that cover the functional and non-functional requirements outlined in the SRS. The instructors will be responsible for providing feedback and ensuring that standards and expectations are met by the project. Additionally, potential users will provide feedback about the functionality and quality of the software to validate the product. Finally colleagues and peers will provide regular feedback towards to the documentation and design which will be taken into account. More specifically, the testing will be broken down into both front-end and back-end testing, as well as integrated testing and manual testing. All four team members will be responsible for end to end testing from the back-end to the front-end for certain features of the project. Back-end testing will include data validation, testing queries and API testing while front-end tests will include content rendering testing, flow testing, input validation testing and so on.

4.2 SRS Verification Plan

The SRS will be verified through peer reviews, stakeholder feedback and by using the SRS checklist as well the the learning objectives. The peer reviews will be taken in the form of surveys from a sample population whose results will then be aggregated, charted and incorporated into the next revision. Surveys will ask questions regarding

visual design, usability, and overall impressions. Peers will also be asked to walk-through tasks and provide feedback. Other stakeholders who will be surveyed include potential users of the product (i.e people who use LaTeX editors) and the Capstone Supervisor, Dr.Spencer Smith as well as Samuel Crawford.

4.3 Design Verification Plan

The design verification will be done by the developers of the project by assessing the design architecture of the software and verifying that it meets the qualities addressed in the SRS. The module guide (MG) and mathematical specification (MIS) will be the most helpful tool in the design review as they will break down the design architecture into components, that can be further detailed and assessed.

4.4 Implementation Verification Plan

The implementation will be verified for behavioural correctness by implementing the functional tests outlined in this document as well as the unit tests to ensure all implementations are performing as expected. In addition, frequent code walkthroughs, reviews and pair programming by the developers will verify the quality and efficiency of the implemented code.

4.5 Automated Testing and Verification Tools

The main tools used for automated UI testing will be Jest and Selenium to test the front-end Javascript components. JSlint will be used for linting the front-end code. In the Java backend, we will use JUnit for unit testing and Java assertions, and the built in Intellij linter. CI/CD will be set up to run unit tests using Terraform.

4.6 Software Validation Plan

To validate the system, we would be getting feedback from stakeholders using interviews and formal reviews. These interviews and formal review would be focused on ensuring that first all necessary business events for the software are included in the SRS, and secondly that all business events and requirements have been correctly implemented.

5 System Test Description

5.1 Tests for Functional Requirements

5.1.1 GitHub Integration

The following section covers all system tests related to the GitHub integration, these would encompass are functional requirements covered by business events 3, 4, 17 and 19 in the SRS.

1. ST-1

   Control: Manual

   Initial State: The user has already logged into the system using the GitHub auth and is connected to their Git account.

   Input: The user creates a project through the system, adds collaborators and provides an valid name, "New Project".

   Output: A new project is created and displayed on the system which is connected to a new Git repository, with the name "New Project".

   Test Case Derivation: When a user creates a project, the Git repository is also created in their GitHub account which should be visible by the user.

   How test will be performed: The tester will press the button to create a project, add collaborators and then create a name for the project that is not already being used on their account. They will then validate that the project shows up on their homepage and the corresponding Git repository is on GitHub as well.

2. ST-2

   Control: Manual

   Initial State: The user has already logged into the system using the GitHub auth and is connected to their Git account.

   Input: The user creates a project through the system, adds collaborators and provides an invalid name.

   Output: An error pops up indicating that the user tried to enter a name for the project which is already taken in a previous Git repository of theirs.

Test Case Derivation: When a user creates a project with an invalid name, the process should end and they should encounter an error.

How test will be performed: The tester will press the button to create a project, add collaborators, and then add a name that already exists in their GitHub and submit. They will validate that the process ends there and an error comes up. They will also validate that the project wasn't created on their homepage along with the Git repository on their GitHub.

3. ST-3

   Control: Manual

   Initial State: The user has already logged into the system using the GitHub auth and is connected to their Git account. The user has a preexisting Git repository in their GitHub to import.

   Input: The user clicks the button to import a project using the interface and selects among the available existing Git repositories in their GitHub to import.

   Output: The user sees a new project on their homepage with all the tex files in it, along with the collaborators added.

   Test Case Derivation: The new project created will be the imported Git repository data that the user requested.

   How test will be performed: The tester will create a repository in GitHub with collaborators and add a tex file with some dummy data. The tester will then log into the system with GitHub auth and then press import to transfer the tex files from the GitHub repository to the system. The tester will then validate if the system has all the correct tex files and collaborators.

4. ST-4

   Control: Manual

   Initial State: The user has already logged into the system using the GitHub auth and is connected to their Git account. The user has a preexisting project in the system.

   Input: The user clicks on the desired project and then the revision history button

   Output: The user is able to view the updated revision history along with the author that made the commit.

Test Case Derivation: The user will be able to see all the latest commit history by clicking the button, which tells the system to pull all the latest commits from GitHub

How test will be performed: The tester will press the revision history button on a preexisting project in the system and view the current commits. Then the tester will create a commit in the GitHub repository and then navigate back to the system and click on the revision history button again to validate that the system does pull all commits from the Git repository.

5. ST-5

   Control: Manual

   Initial State: The user has already logged into the system using the GitHub auth and is connected to their Git account. The user has a preexisting project in the system.

   Input: The user has made some changes to the tex file in a project and then enters a commit message, then clicks commit.

   Output: The commit message is displayed with the changes in Github.

   Test Case Derivation: Any committed changes should have a commit message that can be modified by the user

   How test will be performed: The tester will have a project that they have already created. They will make some new changes and then create a commit through the system. They will confirm they are able to specify and edit a commit message.

6. ST-6

   Control: Manual

   Initial State: The user has already logged into the system using the GitHub auth and is connected to their Git account. The user has a preexisting project in the system.

   Input: The user has made some changes to the tex file in a project and then starts the process to commit the changes.

   Output: The user is able to view the change that was made in the system in the GitHub repository.

Test Case Derivation: Any committed changes should be reflected accurately to GitHub along with the correct user that made the changes.

How test will be performed: The tester will have a project that they have already created. They will make some new changes and then create a commit through the system with an appropriate message. The tester then selects the specific changes that they desire to commit. Once the tester has committed them, the system should push them to GitHub repository. The tester will then validate if all the changes in the GitHub repository is accurately reflecting what was on the system.

5.1.2 Authentication

The following section covers all system tests related to authentication, these would encompass all functional requirements covered by business event 2 in the SRS.

1. ST-7

   Control: Manual

   Initial State: The user has an existing GitHub account and is on the log in page.

   Input: The user fills in the GitHub third party authentication form with the correct credentials

   Output: A valid GitHub API token and JSON Web Token is returned and the user is redirected to the projects page corresponding to their profile

   Test Case Derivation: Logging in through the GitHub third party authentication system should give us access to the api token that can be used to retrieve the a username that can will be mapped to a corresponding project page in our system which is then displayed.

   How test will be performed: The tester will log in with a valid GitHub account and validate that they are redirected to the projects page corresponding to their profile. Additionally the tester would also validate that a valid GitHub API token and JSON Web Token is returned.

2. ST-8

   Control: Manual

Initial State: The user has an existing GitHub account

Input: The user fills in the GitHub third party authentication form with the incorrect credentials

Output: The user is informed that they were not able to successfully authenticate and returned to the log in page.

Test Case Derivation: The user will not be able to get the GitHub API token necessary for this application if they cannot log in through GitHub's third party authentication system.

How test will be performed: The tester will try to log in with an invalid GitHub account and validate that they are returned to the log in page and the correct error is displayed.

3. ST-9

   Control: Manual

   Initial State: The user is already logged in and has valid GitHub API and JSON Web Tokens.

   Input: The user attempts to access the log in page

   Output: The user is redirected to the projects page

   Test Case Derivation: The user should not be able to access the log in page when they are already logged in. Thus they are returned to main projects page instead.

   How test will be performed: The tester will try to access the log in page while logged in, and validate that they are redirected to the projects page.

4. ST-10

   Control: Manual

   Initial State: The user is logged in with an invalid JSON Web Token or GitHub API Token

   Input: The user attempts to access the user specific page such as projects, or files

   Output: The user is redirected to the log in page and the JSON Web Token and GitHub API Token is removed

Test Case Derivation: A user with an invalid JSON Web Token and GitHub API token would not be able to prove that the information they are trying to access belongs to them and would also be unable to make the neccesary GitHub API calls needed for the application. Thus we would need to ask the user to log in again and revoke their tokens.

How test will be performed: The tester will manually add invalid API and JSON Web tokens as cookies to their browsers and try to access as restricted url. The tester will then confirm that they are redirected to the login page and the tokens have been revoked.

5. ST-11

   Control: Manual

   Initial State: The user is already logged in and has valid GitHub API and JSON Web Tokens.

   Input: The user clicks the log out button

   Output: The user is redirected to the log in page and the JSON Web Token and GitHub API Token is removed

   Test Case Derivation: The user is trying to log out so all information that connects them to an account needs to be revoked. Thus we would need to ask the user to log in again and revoke their tokens.

   How test will be performed: The tester will manually click the log out button. The tester will then confirm that they are redirected to the login page and the tokens have been revoked.

5.1.3 Editor

The following section covers all system tests related to editing files, these would encompass some of the functional requirements covered by business event 9-14 in the SRS.

1. ST-83

   Control: Manual

   Initial State: User has a file open in a project

   Input: A different user deletes the project from the project menu

Output: The editor page should be redirected to the projects menu page with a message stating the current project no longer exists

Test Case Derivation: Deleting a project should be reflected for all collaborators.

How test will be performed: A user will create a project and add a collaborator, then open the project, create a file and open the file. The collaborator will then delete the project while the other user still has it open.

2. ST-12

   Control: Automated

   Initial State: User has the editor open for a project

   Input: User creates files

   Output: The file list in the editor should display all the files that were created and the file that already existed in alphabetical order

   Test Case Derivation: The file list in the editor view of a project should display the files in the project so that the user can edit them.

   How test will be performed: An automated script will create a new project, and will try to edit it. In the editor, the script will create new files. The script will then confirm that the newly created file and the default file are listed in the file list alphabetically.

3. ST-13

   Control: Manual

   Initial State: Multiple users have the same LaTeX file open in the editor

   Input: All the users click some text position to move their cursor

   Output: All the users cursor's current position should be shown

   Test Case Derivation: When multiple users are editing a LaTeX file, it is necessary to see where each user's text cursor position is located

   How test will be performed: The tester will have to log into different testing accounts using different web browsers. The tester will then move the text cursor from each different session to a different location. At the end, the tester will observe that all the text cursor positions are show in each session

4. ST-14

   Control: Manual

   Initial State: User opens a LaTeX file in the editor

   Input: User types LaTeX specific code

   Output: The LaTeX code should be highlighted accordingly

   Test Case Derivation: When a user types LaTeX specific code, the system should highlight its such that it helps the user understand the LaTeX file

   How test will be performed: The tester will open a LaTeX file, and type LaTeX functions and environments. The tester will then confirm that the LaTeX functions are highlighted such that they can be differentiated from normal text

5. ST-15

   Control: Manual

   Initial State: User opens a LaTeX file in the editor

   Input: User types normal text with some spelling errors, "Hkello there"

   Output: The words with the spelling error, "Hkello", should be highlighted

   Test Case Derivation: When a user does a spelling error, the system should highlight the error so that the user can fix it

   How test will be performed: The tester will open a LaTeX file in the editor, and types text with some spelling error . The tester will then confirm that the words with the spelling errors are highlighted

6. ST-16

   Control: Manual

   Initial State: Multiple users have their editor open for the same project

   Input: All the different users make an different edit to the same files using the editor

   Output: The updated file should have all those changes and should be the same for all the users

   Test Case Derivation: When multiple users make changes to a LaTeX file, those changes need to be synchronized for all the users to see

How test will be performed: The tester will have to log into different testing accounts using different web browsers. The tester will then make a change from each of the open sessions. At the end, the tester will observe that all the files are the same and have all the changes made

7. ST-17

   Control: Manual

   Initial State: User has a project open in the editor with a file to delete

   Input: User selects the option to delete one of the file from the file list

   Output: A confirmation dialog confirming the user to delete the file should appear

   Test Case Derivation: When a user tries to delete a file, there should be a confirmation screen to make sure they want to delete the file.

   How test will be performed: An automated scrip will open the editor with a file to delete, the scrip will then attempt to delete the file. It will then confirm that a confirmation dialog confirming the choice of the user to delete the file shows up.

8. ST-18

   Control: Manual

   Initial State: Multiple users have their editor open for the same project

   Input: One of the user deletes a file

   Output: All of the users should not longer be able to see the file in the editor

   Test Case Derivation: When a user deletes a file, that file should also be deleted for other users

   How test will be performed: The tester will have to log into different testing accounts using different web browsers. The tester will then delete a file from one of the open session. The tester can create a file to delete if there are none. At the end, the tester will observe that the file he deleted is also deleted in all the other sessions.

9. ST-19

Control: Automated

Initial State: User has a project open in the editor with a file to rename

Input: User clicks the edit icon and enters a new name for the file

Output: The file is displayed with the new name

Test Case Derivation: When a user renames a file, they should be allowed to input the new name

How test will be performed: Automated testing script will try renaming one of the file, and will confirm that the file is renamed in the file list according to the new name.

10. ST-20

    Control: Automated

    Initial State: User has a project open in the editor with a file to rename

    Input: User selects the option to rename one of the file from the file list, and renames it to a different name

    Output: This event of the file being renamed should be recorded in the client side storage alongside the user who made the change

    Test Case Derivation: When a user renames a file, this event should be recorded so that it can be used when committing these changes

    How test will be performed: Automated testing script will try renaming one of the file, and will confirm that the data structure keeping track of the events contains the this renaming event.

11. ST-21

    Control: Manual

    Initial State: Multiple users have their editor open for the same project

    Input: One of the user renames a file

    Output: All of the users should should see the updated file name in the editor

    Test Case Derivation: When a user renames a file, that file should also be renamed for other users

    How test will be performed: The tester will have to log into different testing accounts using different web browsers. The tester will then rename a file from

one of the open session. The tester can create a file to rename if there are none. At the end, the tester will observe that the file he renamed is also renamed in all the other sessions.

12. ST-22

    Control: Manual

    Initial State: User has the editor open for a project

    Input: User selects the option to create a new file

    Output: The system should present the user with modal that prompts the user for the name and the extension of the file, with default name and .tex extension filled in

    Test Case Derivation: When a user tries to create a new file, they should be allowed to input the name and extension of the file, or they can use the default values

    How test will be performed: The tester will open the editor for a project, and select the option to create a new file. The tester will then confirm, that the new file creation modal appears with default values for the name and the extension of the file. The tester will also confirm they are able to modify these values.

13. ST-23

    Control: Manual

    Initial State: User has selected the option to create a new file in the editor and has filled in the name and the extension of the file

    Input: User selects confirm on the new file creation modal

    Output: The empty file should be added to the editor, and should also be visible to other users

    Test Case Derivation: When a user clicks confirm on the new file creation modal, the file should be created according to the information given from the modal

    How test will be performed: The tester will open the editor for a project, and select the option to create a new file. The tester will fill in the name of the file and the extension accordingly, and finally click confirm. The tester will then confirm that the file shows up in the file section of the editor with the

same name and extension that was specified during file creation. The tester will also open new browser sessions with different test account to access the same project, and will confirm the file exists for those users too.

14. ST-24

    Control: Manual

    Initial State: Multiple users have their editor open for the same project

    Input: Each user creates a new file

    Output: Each user should be able to see all the files created by the other users in the editor

    Test Case Derivation: When a user creates a file, that file should be accessible and shown to all the different users editing that project

    How test will be performed: The tester will have to log into different testing accounts using different web browsers. The tester will then create a file from each of the open sessions. At the end, the tester will observe that all the files he created in different sessions is visible from all those sessions

15. ST-25

    Control: Manual

    Initial State: User has a the editor open for a project

    Input: User selects the option to upload a file

    Output: The user should be presented a modal to choose their local file

    Test Case Derivation: When a user chooses the option to upload a file, they should be allowed to find and upload their local file.

    How test will be performed: The tester will open the editor for a project, and select the option to upload a new file. The tester will confirm a modal appears from which they can browse through and find their local file.

16. ST-26

    Control: Manual

    Initial State: User has file upload modal open

Input: User browses through and chooses a local file to upload

Output: The file should appear in the file list window with the same name and extension as the local file chose to upload with the same file content

Test Case Derivation: When a user chooses to upload a file, the uploaded file should have the same name, extension and content

How test will be performed: The tester will open the editor for a project, and select the option to upload a new file. The tester will the choose a local test file, and confirm the upload. Lastly, the tester will make sure the file name, extension and the content is the same as the local file.

17. ST-27

    Control: Manual

    Initial State: User has a LaTeX file open in the editor

    Input: User tries to insert, delete and modify the text in the LaTeX file

    Output: Each action should be carried out without any issues

    Test Case Derivation: This test case makes sure the basic functionality of the editor to insert, delete and modify test works without any issues.

    How test will be performed: The tester will open a LaTeX file in the editor , and try editing it by inserting, deleting and modifying text. The tester will confirm all those functionalities work without any issues.

18. ST-28

    Control: Manual

    Initial State: User has a LaTeX file open in the editor

    Input: User tries to insert, delete and modify the text in the LaTeX file

    Output: This event of the file being edited should be saved alongside the user who made the change

    Test Case Derivation: When a user edits a file, this event should be recorded so that it can be used when committing these changes

    How test will be performed: Automated testing script will try editing one of the file, and will confirm that the data structure keeping track of the events contains the edits done as an event.

5.1.4 Chat

The following section covers all system tests related to the chat available in the editor, these would encompass some of the functional requirements covered by business event 18 in the SRS.

1. ST-29

   Control: Manual

   Initial State: User has the editor screen open

   Input: User opens the project screen containing the chat

   Output: The chat-box should show up with all the messages exchanged in this project

   Test Case Derivation: When a user opens up the chat box, they should be presented with a chat-box

   How test will be performed: The tester will open the editor for a project and open the chat-box. The tester will then confirm that the chat-box and messages show up.

2. ST-30

   Control: Manual

   Initial State: User has the chat-box open

   Input: User sends a message in the chat

   Output: The sent message gets displayed in the chat for all the users editing the project

   Test Case Derivation: When a user wants to communicate through the chat-box, they should be able to send messages which should be displayed to every user

   How test will be performed: The tester will open the editor for a project and open the chat-box through different test account in different web browsers. The tester will type a random test message, and send it. The tester will then check that the message is shown in all the sessions

5.1.5 Projects Menu

The following section covers all system tests related to the projects menu, these would encompass are functional requirements covered by business events 3, 4, 5, 6, and 7 in the SRS.

1. ST-31

   Control: Manual

   Initial State: The user has the create project screen open.

   Input: The user enters a valid project name into the appropriate form field and clicks the create project button.

   Output: No error is given for creating the project.

   Test Case Derivation: A valid project name should not give any errors.

   How test will be performed: The tester will open the create project screen, enter a valid name and finish the task, then ensure that no error is displayed.

2. ST-84

   Control: Manual

   Initial State: The user has the create project screen open.

   Input: The user enters a project name with Arabic letters into the project name form field and clicks the create project button.

   Output: An error is given stating project names must be English alpha-numeric characters.

   Test Case Derivation: An invalid project name should give an errors.

   How test will be performed: The tester will open the create project screen, enter the invalid name and ensure the process could not be completed.

3. ST-32

   Control: Manual

   Initial State: The user has the create project screen open.

   Input: The user enters an invalid project name into the appropriate form field and tries to create the project.

Output: An error message is displayed indicating that the project was not created due to invalid project name.

Test Case Derivation: Invalid names are not permitted for projects.

How test will be performed: The tester will open the create project screen, enter an invalid name and finish the task, then confirm that the appropriate error message is displayed and that a new project was not created.

4. ST-33

   Control: Manual

   Initial State: The user has a project edit menu open on their screen

   Input: The user enters a valid username of the collaborator they want to add into the appropriate form field and clicks the add button.

   Output: The users that were added are listed as a collaborator to the project get an invite to join the project from GitHub and can see the project in their projects menu.

   Test Case Derivation: Users added as collaborators to a project should be listed as collaborators to the project and should be able to access the project

   How test will be performed: The tester will add a test user as a collaborator to a project in their repository. Then they will open the list of collaborators for the project and ensure the test user is listed. Finally they will log into the test user account and ensure that the project they were added to shows up in their repository.

5. ST-34

   Control: Manual

   Initial State: The user has a project open on their screen and a collaborator with the given name already added to the project

   Input: The user clicks on the add collaborators icon and enters the username of the collaborator that is already in the project

   Output: The user must be informed that the collaborator already exists in the project.

   Test Case Derivation: User should not be able to add an existing collaborator to the project twice.

How test will be performed: The tester will try to add a collaborator to the project twice and ensure that an error is given when they try to add the collaborator the second time

6. ST-35

   Control: Manual

   Initial State: The user has a project open on their screen

   Input: The user clicks on the add collaborators icon and enters an invalid username of the collaborator then clicks the done icon.

   Output: An error message is displayed indicating that the collaborator name is invalid and no collaborators were added to the project.

   Test Case Derivation: Users should not be able to add users that do not exist to their repositories.

   How test will be performed: The tester will enter an invalid name into the collaborator form field and ensure that the appropriate error message shows up.

7. ST-36

   Control: Manual

   Initial State: The user has the create new project screen open.

   Input: The user creates a new project with an appropriate name then clicks done.

   Output: A new repository with the appropriate name is created in the database and the user's GitHub account that is linked to their profile.

   Test Case Derivation: A new project being created must trigger the creation of a corresponding Git repository.

   How test will be performed: The tester will create a new repository with a valid name from the create new project screen. Then they will navigate to their GitHub account and ensure a new repository with the matching name has been created in their directory.

8. ST-37

Control: Manual

Initial State: The user has the create new project screen open.

Input: The user clicks import from repository option.

Output: The screen displays a list of repositories from the user's GitHub directory to choose from.

Test Case Derivation: When importing a repository, the user should be prompted with the list of existing repositories in their GitHub to select from.

How test will be performed: The tester will create a new project with a valid name from the create new project screen. Then they will navigate to their GitHub account and ensure a new repository with the matching name has been created in their directory.

9. ST-38

   Control: Manual

   Initial State: The user has the create new project screen open.

   Input: The user selects import from repository option and chooses a repository to import, then clicks done.

   Output: A new project is created that lists and contains all the TeX and image files that are present in the GitHub repository that was imported.

   Test Case Derivation: When creating a project from a pre-existing repository, all and only the existing Tex files should be imported into the Tex editor.

   How test will be performed: The tester will create a GitHub repository that contains multiple Tex files. Then they will import the repository into a new project in the Tex editor and ensure that all Tex files are imported along with their original and complete content.

10. ST-39

    Control: Manual

    Initial State: The user has the create new project screen open.

    Input: The user selects import from repository option and chooses a repository to import, then clicks done.

Output: A new project is created that lists and contains all the collaborators that are present in the GitHub repository that was imported with appropriate access.

Test Case Derivation: When creating a project from a pre-existing repository, all and only the existing collaborators should be imported into the new project.

How test will be performed: The tester will create a GitHub repository that contains multiple collaborators with read-only and read-write access. Then they will import the repository into a new project in the Tex editor and ensure that all collaborators are imported along with their appropriate access permissions.

11. ST-40

    Control: Manual

    Initial State: The user has created a project with collaborators in their directory, and the directory is displayed on screen.

    Input: The user selects delete project and clicks confirm.

    Output: The project is no longer listed in the project directory nor in any of the collaborators' directory, and the content is no longer stored in the Tex editor.

    Test Case Derivation: Deleting a project should remove it and it's contents for both the owner and the collaborators.

    How test will be performed: The tester will create a project that contains multiple collaborators. Then they will delete the project from their directory and ensure that it no longer exists in their own directory nor the directories of the collaborators.

12. ST-41

    Control: Manual

    Initial State: The user has a non-empty directory containing projects they own and projects they have been added as a collaborator to.

    Input: The user opens the projects menu.

    Output: A list is displayed on the screen of all the projects that the user has created or been added as a collaborator to with their Titles and date modified, in descending order from date modified.

Test Case Derivation: All projects that a user is involved in should be listed for the user to view in correct order.

How test will be performed: The tester will create a few projects, as well as add themselves as a collaborator to a project created from a test account. Then they will click on projects and ensure that all the projects are listed in the opposite order they created them.

13. ST-42

    Control: Manual

    Initial State: The user has a project containing at least one collaborator

    Input: The user selects the edit icon and selects remove collaborator, then selects a collaborator from the list and clicks confirm.

    Output: The collaborator that was removed is no longer listed as a collaborator and can no longer view the project.

    Test Case Derivation: Removing a collaborator should remove them from the project and revoke their access.

    How test will be performed: The tester will create a project and add a test account as a collaborator. Then they will click edit and remove the test user as a collaborator. Then they will confirm that the test user is no longer listed as a collaborator, as well as confirm that the test account no longer has access to the project.

5.1.6 LaTeX Compilation

The following section covers all system tests related to LaTeX compilation, these would encompass are functional requirements covered by business events 15 and 16 in the SRS.

1. ST-43

    Control: Manual

    Initial State: Empty LaTeX file

    Input: User wants to view a LaTeX file that has not been compiled yet

    Output: An empty PDF file is displayed

Test Case Derivation: An empty LaTeX file should map to an empty PDF file

How test will be performed: The tester will log in to a testing account and manually click the compile button after opening an empty LaTeX file. The tester will then verify that an empty PDF file is displayed.

2. ST-44

   Control: Manual

   Initial State: A valid LaTeX file with data in it

   Input: User clicks the compile LaTeX button

   Output: A successfully compiled PDF file with the correct corresponding data that maps to the LaTeX code is displayed

   Test Case Derivation: A valid LaTeX file should not give any errors and should display the correct corresponding pdf

   How test will be performed: The tester will log in to a testing account and manually click the compile button on a LaTeX file that contains valid LaTeX data. The tester will then verify that a valid and corresponding PDF file is displayed.

3. ST-45

   Control: Manual

   Initial State: A valid LaTeX file with data in it that has already been compiled and is being displayed as a PDF

   Input: User updates the LaTeX file to a different valid file and clicks the compile LaTeX button

   Output: A successfully compiled PDF file with the correct corresponding and different data from the previous PDF file should be displayed

   Test Case Derivation: This test is to confirm that the LaTeX file updates to the latest version of the compiled PDF.

   How test will be performed: The tester will log in to a testing account, verify that a PDF is displayed for the current valid LaTeX file, update the LaTeX file to a different valid file and compile it. The tester will then verify that a valid and corresponding PDF file that is different from the previous PDF is displayed.

4. ST-46

   Control: Automated

   Initial State: A LaTeX file with invalid data in it

   Input: User clicks the compile button

   Output: A correct and corresponding error that points to the specific issue with the LaTeX file that prevented it from compiling is displayed.

   Test Case Derivation: An invalid LaTeX file should not be compiled and instead display an error that points to the issue.

   How test will be performed: This test can be performed with a front-end testing framework such as Selenium or Jest which can click the compile button and verify that an error is displayed as text in the correct place on the user interface.

5. ST-47

   Control: Automated

   Initial State: A LaTeX file with valid data in it

   Input: The function to compile the PDF is triggered

   Output: A PDF corresponding to the LaTeX file and sharing the name of the LaTeX file is created and saved in the system

   Test Case Derivation: A valid LaTeX should successfully compile and a PDF should be saved in the system with the same name as the LaTeX file.

   How test will be performed: This test can performed using unit testing frameworks that input a valid LaTeX input and trigger the function responsible for compiling it, then verify that a PDF with the correct name and data in the s

5.1.7 Instructions

The following section covers all system tests related to viewing instructions, these would encompass are functional requirements covered by business event 1 in the SRS.

1. ST-48

Control: Automated

Initial State: The application page is open on the screen

Input: User clicks on instructions icon

Output: Instructions screen is displayed on the screen

Test Case Derivation: Instructions should be displayed when requested by the user

How test will be performed: Automated testing script will select instructions icon and trigger a click. Elements on resulting page will be matched against intended instructions page contents

2. ST-49

Control: Automated

Initial State: Instructions page is open and displayed on screen

Input: Close icon is clicked

Output: Instructions screen is no longer displayed

Test Case Derivation: Instructions screen should no longer be displayed once closed

How test will be performed: Automated testing script will trigger a button click on the close icon.

5.2 Tests for Nonfunctional Requirements

5.2.1 Look and Feel Requirements

1. ST-50

Type: Dynamic, Manual

How test will be performed: Users will be asked a question in a server to rate how minimalist the design is. The average rating given by the users must be greater than the simplicity_rating symbolitic parameter that is defined in the SRS.

2. ST-51

Type: Dynamic, Manual

How test will be performed: The code responsible for the styling of the website will be reviewed by all developers to ensure that it follows the specified color scheme. The total amount of distinct colors used in the design must be less than the num_colors symbolitic parameter that is defined in the SRS.

3. ST-52

Type: Static, Manual

How test will be performed: The code responsible for the selecting cursors colors will be reviewed by all developers to ensure that it will create distinct colors for up to 20 users.

5.2.2 Usability and Humanity Requirements

1. ST-53

Type: Static, Manual

How test will be performed: Users will be asked a series of questions about how long it took them to be able to effectively use the system, if they are already familiar with LaTeX, and how easy the material was to understand. Only submissions for users who claim they are familiar with LaTeX will be considered as this application is targeted towards such users, the average learning_time that is taken must be less than and the easiness_rating must be greater than their respective symbolic parameters in the SRS.

2. ST-54

Control: Automated, Dynamic

Initial State: Light mode is enabled on

Input: The button to switch to dark mode is clicked

Output: The colors of the application are switched to darker colors that fit the new theme

Test Case Derivation: Clicking the dark mode and light mode button should change the styling of the application to fit the current theme.

How test will be performed: Automated testing script will click the button and then verify that the styling of the specific elements contain the correct colors.

5.2.3 Performance Requirements

1. ST-55

   Type: Static, Manual

   How test will be performed: Users will be asked a question about how responsive the application feels in a survey. An average rating of all users will be taken and must be greater than the responsiveness_rating symbolic parameter that is defined in the SRS.

2. ST-56

   Control: Automatic

   Initial State: A LaTeX file with valid input is ready to be compiled

   Input: The compilation button is triggered multiple times

   Output: The time taken to receive the response is taken multiple times

   Test Case Derivation: We want to get the response time of the compilation with multiple trials, thus we need to output the response time after the button is clicked.

   How test will be performed: Automated testing script will click the button and then take the average of all the response times. The response time must be less than the compile_time symbolic parameter that is defined in the SRS.

3. ST-57

   Type: Static, Manual

   How test will be performed: Metrics from the hosting platform of the application will be looked at on a weekly basis to ensure that the up-time of the application is greater than or equal to the the uptime symbolic parameter that is defined in the SRS.

4. ST-58

Type: Static, Manual

How test will be performed: All code and infrastructure for the data storage will be reviewed by all developers to ensure that the code saves data to a remote database, and the database infrastructure is scalable both vertically and horizontally.

5. ST-59

   Type: Static, Manual

   How test will be performed: All code and infrastructure will be reviewed by all developers to ensure that it follows the best practices for maintainability.

5.2.4 Operational and Environmental Requirements

1. ST-60

   Type: Static, Manual

   How test will be performed: The application will be tested with a wide variety of devices with different screen ratios and peripherals. The application should look and perform the same on all configurations of devices and peripherals.

2. ST-61

   Type: Static, Manual

   How test will be performed: Developers will keep close track of all milestones in the development of the project to ensure that the project is on track with the road-map and will be completed by March 20th 2023.

5.2.5 Maintainability and Support Requirements

1. ST-62

   Type: Static, Manual

   How test will be performed: A third party stakeholder will review all documents and diagrams that are included in the system and validate whether they can fully understand the system using the documentation and diagrams.

2. ST-63

   Type: Static, Manual

   How test will be performed: The application will be manually tested on google chrome, firefox, and safari to ensure that it performs and looks the same on all browsers.

5.2.6 Security Requirements

1. ST-64

   Type: Static, Manual

   How test will be performed: The developers will validate and confirm that the source code for the system and the Git repository is only view-able to the public and limits manipulation of source code to only maintainers.

2. ST-65

   Type: Static, Manual

   How test will be performed: The testers will have multiple public user accounts that they can use to create projects with and then validate that each user can only access projects they are authorized for.

3. ST-66

   Type: Static, Manual

   How test will be performed: The testers will confirm that the data that is being received from requests are all necessary for the user's request and there is no extra data lying around.

4. ST-67

   Type: Static, Manual

   How test will be performed: Users will be asked a question about how communicative and appropriate the errors that they encounter are and if there is anything missing.

5. ST-68

   Type: Static, Manual

   How test will be performed: The testers will validate whether a confirmation modal comes up for modifications that are significant, like creating and deleting a project and committing changes.

6. ST-69

   Type: Static, Manual

   How test will be performed: The testers will validate that the system modifications and manipulations do not affect the user's data against their request or desire.

7. ST-70

   Type: Static, Manual

   How test will be performed: The testers will perform security threats against the system to validate whether the system provides a reasonable responds as desired.

8. ST-71

   Type: Static, Manual

   How test will be performed: The testers will intentionally run procedures against the database to see whether the database still persists the backed up data as expect.

9. ST-72

   Type: Static, Manual

   How test will be performed: The tester will delete a project that exists in their homepage. Then confirm whether the project appears in the trash section and restores it to validate that the projects are retained.

10. ST-73

    Type: Static, Manual

    How test will be performed: The tester will disconnect their internet connection and continue working on the project and then reload the page. The tester will then validate whether any new changes still persisted.

11. ST-74

    Type: Static, Manual

    How test will be performed: The testers will try out edge cases against the system to validate that there are necessary measures to prevent unintentional behaviour.

12. ST-75

    Type: Static, Manual

    How test will be performed: The tester will validate that the only way to access the system is by creating an account, if one does not already exist. The tester will also validate that the user data that is collected is minimal and for only what is required and adheres to the general privacy standards.

13. ST-76

    Type: Static, Manual

    How test will be performed: The developers will review the code to validate that there are no security concerns with the method that the system is using to store user credentials and if it is up to par with industry standards.

14. ST-77

    Type: Static, Manual

    How test will be performed: The developers will confirm and validate that the API keys being used by the system are changing at a reasonable interval.

15. ST-78

    Type: Static, Manual

    How test will be performed: The tester will validate that the system does log out user's after a certain period of time or when deemed necessary.

5.2.7 Cultural Requirements

1. ST-80

   Type: Static, Manual

   How test will be performed: A culture review will be performed on the product by taking a sample population from various demographics and recording their feedback

5.2.8 Legal Requirements

1. ST-81

   Type: Static, Manual

   How test will be performed: The validity of the user agreement will be assessed by a legal professional

2. ST-82

   Type: Static, Manual

   How test will be performed: Tester collects all the media on the application that is there by default and confirm all media is copy-right free

5.2.9 Health and Safety Requirements

N/A

5.3 Traceability Between Test Cases and Requirements

Table 1: Traceability Matrix between FR & Test cases

Functional Requirement	Test Case
FR1	ST-48
FR2	ST-49
FR3	ST-7, ST-8, ST-9
FR4	ST-10, ST-11
FR5	ST-1, ST-2, ST-31, ST-32
FR6	ST-1, ST-33, ST-34, ST-35
FR7	ST-1, ST-36
FR8	ST-3, ST-37
FR9	ST-3, ST-38
FR10	ST-3, ST-39
FR11	ST-40
FR12	ST-40
FR13	ST-41
FR14	ST-41
FR15	ST-41
FR16	ST-33
FR17	ST-42
FR18	ST-12
FR19	ST-12
FR20	ST-13
FR21	ST-16, ST-18, ST-21, ST-24
FR22	ST-14
FR23	ST-15
FR24	ST-22
FR25	ST-22

FR26	ST-22, ST-23, ST-24
FR27	ST-25
FR28	ST-26
FR29	ST-17
FR30	ST-18
FR31	ST-19
FR32	ST-19, ST-21
FR33	ST-20
FR34	ST-27
FR35	ST-28
FR36	ST-43, ST-44, ST-45, ST-46
FR37	ST-46
FR38	ST-47
FR39	ST-43, ST-44, ST-45
FR40	ST-4
FR41	ST-4
FR42	ST-29
FR43	ST-30
FR44	ST-30
FR45	ST-5
FR46	ST-6
FR47	ST-6

Table 2: Traceability Matrix between NFR & Test cases

Non-Functional Requirement	Test Case
NFR1	ST-50
NFR2	ST-50, ST-51
NFR3	ST-50, ST-52
NFR4	ST-53
NFR7	ST-53
NFR8	ST-53
NFR10	ST-55
NFR11	ST-56
NFR16	ST-59
NFR20	ST-61
NFR21	ST-62
NFR22	ST-62
NFR23	ST-63
NFR26	ST-65
NFR28	ST-67
NFR29	ST-68
NFR34	ST-73
NFR36	ST-75
NFR38	ST-76
NFR39	ST-77
NFR40	ST-78
NFR41	ST-80
NFR42	ST-81
NFR43	ST-82

6 Unit Test Description

Test cases will be selected to cover the core functionality of each module and would be done with boundary unit testing to minimize the number of unit tests required to fully validate the functionality of each module.

6.1 Unit Testing Scope

The real time editing aspect of the editor module is out of scope since the functionality of this module is mostly provided by libraries, however integration with these libraries will still need to be tested. Many of the GitHub integration are also out of scope as they would require the use of integration tests to ensure that changes happen on GitHub as well when our service makes a specific change, and as a result these can be better tested with manual system tests.

6.2 Tests for Functional Requirements

6.2.1 Project Services

The subsections below covers the Project Services module in the MIS which is where all the logic for the remaining project modules that can be unit tested lies. We will be testing the boundaries for each core functionality of the module, there will be one basic black box test to ensure that the core functionality of the module is working as intended and the remaining test would be white box boundary tests to ensure that the implementation of the module does not have any invalid logic that interferes with its functionality. These tests will be derived to closely follow the functional requirements of the module as well.

1. PS-1

   Type: Automatic

   Initial State: User is logged in and has not created a project with selected name

   Input: User inputs a project name, and no collaborators

   Output: A project with the given name is saved in the database

   Test Case Derivation: This test is to verify that a user can create a project and no issues occur from not adding any collaborators to the project

How test will be performed: The test will be performed using NodeJS testing framework Jest.

2. PS-2

   Type: Automatic

   Initial State: User is logged in and has not created a project with selected name

   Input: User inputs a project name, and a valid GitHub username as a collaborator

   Output: A project with the given name and collaborators is saved in the database

   Test Case Derivation: This test is to verify that a user can create a project and no issues occur from adding a collaborator to the project

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

3. PS-3

   Type: Automatic

   Initial State: User is logged in and has not created a project with the selected name

   Input: User inputs the name of the existing project to try and recreate it

   Output: The user gets an error that the project with the given name already exists

   Test Case Derivation: This test is to verify that a user cannot recreate an existing project

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

4. PS-4

   Type: Automatic

   Initial State: User is logged in and has existing projects in their repository that have not been imported before

Input: User imports a project

Output: A project with the correct name, collaborators, and files is added to the database

Test Case Derivation: This test is to verify that a user can import a project without any issues

How test will be performed: The test will be performed using NodeJS testing framework Jest.

5. PS-5

   Type: Automatic

   Initial State: User is logged in and wants to delete a project that exists in the database

   Input: User deletes a project

   Output: The project no longer exists in the database

   Test Case Derivation: This test is to verify that a user can delete a project without any issues

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

6. PS-6

   Type: Automatic

   Initial State: User is logged in and wants to delete a project that does not exist in the database

   Input: User deletes a project

   Output: The user gets an error stating that the project does not exist

   Test Case Derivation: This test is to verify that the backend will not behave unexpectedly if we try to delete a project that does not exist.

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

7. PS-7

Type: Automatic

Initial State: User is logged in and has created a project and is also a collaborator of a project. Additionally there should be a project that the user is not a part of in any way.

Input: User asks to see all their projects

Output: The user get both the project that they have created and the project that they are a collaborator of, but not the project that they are not a part of.

Test Case Derivation: This test is to verify that the user can access all projects that they are a part of and all projects that they are a collaborator in.

How test will be performed: The test will be performed using NodeJS testing framework Jest.

8. PS-8

   Type: Automatic

   Initial State: User is logged in and has created a project

   Input: User adds a collaborator with a valid username to the project

   Output: The collaborator is added to the project in the database.

   Test Case Derivation: This test is to verify that the user can add collaborators to their project. We cannot verify that a collaborator is added on GitHub since that would require the collaborator to accept the invitation first. This is not possible with unit tests and thus will be fully tested with system tests.

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

9. PS-9

   Type: Automatic

   Initial State: User is logged in and has created a project

   Input: User removes an existing collaborator from their project

   Output: The collaborator is removed from the project in the database.

   Test Case Derivation: This test is to verify that the user can remove collaborators from their project.

How test will be performed: The test will be performed using NodeJS testing framework Jest.

6.2.2 File Services

The subsections below covers the File Services module in the MIS which is where all the logic for the remaining file modules that can be unit tested lies. We will be testing the boundaries for each core functionality of the module, there will be one basic black box test to ensure that the core functionality of the module is working as intended and the remaining test would be white box boundary tests to ensure that the implementation of the module does not have any invalid logic that interferes with its functionality. These tests will be derived to closely follow the functional requirements of the module as well.

1. FS-1

   Type: Automatic

   Initial State: User is logged in and has an empty TeX file that they want to compile

   Input: User compiles the TeX file

   Output: The user should be given an error that the TeX file is empty

   Test Case Derivation: This test is to verify that the LaTeX compiler can handle and alert the user of errors

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

2. FS-2

   Type: Automatic

   Initial State: User is logged in and has an valid TeX file that they want to compile

   Input: User compiles the TeX file

   Output: The user is given a success message for the LaTeX file being successfully compiled

   Test Case Derivation: This test is to verify that the LaTeX compiler can compiled valid files correctly.

How test will be performed: The test will be performed using NodeJS testing framework Jest.

3. FS-3

   Type: Automatic

   Initial State: User is logged in and does not have a file compiled with a given name

   Input: User wants to access the PDF for a file that is not compiled yet

   Output: The user gets an empty PDF file returned to them

   Test Case Derivation: This test is to verify that the system returns an empty PDF for non-compiled files as specified in the functional requirements.

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

4. FS-4

   Type: Automatic

   Initial State: User is logged in and has file compiled with a given name

   Input: User wants to access the PDF for a file that is compiled already

   Output: The user gets a valid PDF for the compiled file

   Test Case Derivation: This test is to verify that the system returns the correct PDF file for the compiled LaTeX.

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

6.2.3 Chat Services

The subsections below covers the Chat Services module in the MIS which is where all the logic for the remaining chat modules that can be unit tested lies. We will be testing the boundaries for each core functionality of the module, there will be one basic black box test to ensure that the core functionality of the module is working as intended and the remaining test would be white box boundary tests to ensure that the implementation of the module does not have any invalid logic that interferes with its functionality. These tests will be derived to closely follow the functional requirements of the module as well.

1. CS-1

   Type: Automatic

   Initial State: A project has two collaborators

   Input: One user types a message and presses Send

   Output: The message content and metadata (sender, timestamp, UUID) are stored in the database

   Test Case Derivation: This test is to verify that the chat database service is properly storing the user input in the backend

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

2. CS-2

   Type: Automatic

   Initial State: A project has two collaborators

   Input: One user types a message and presses Send

   Output: The user's Github profile picture is displayed next to their message

   Test Case Derivation: This test is to verify that the chat service is correctly pulling the profile picture images from Github corresponding to the user

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

3. CS-3

   Type: Automatic

   Initial State: A project has two collaborators

   Input: One user types a message and presses Send

   Output: The message is displayed in the chat-box

   Test Case Derivation: This test is to verify that the UI is reflecting the user input and updating accordingly

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

6.2.4 Auth Services

The subsections below covers the Auth Services module in the MIS which is where all the logic for the remaining auth modules that can be unit tested lies. We will be testing the boundaries for each core functionality of the module, there will be one basic black box test to ensure that the core functionality of the module is working as intended and the remaining test would be white box boundary tests to ensure that the implementation of the module does not have any invalid logic that interferes with its functionality. These tests will be derived to closely follow the functional requirements of the module as well.

1. AS-1

   Type: Automatic

   Initial State: User is on the homepage and not authenticated

   Input: User clicks Login

   Output: The user is redirected to GitHub's login page to enter their credentials and then redirected back to the user's project page

   Test Case Derivation: This test is to verify that a user is able to access the UnderTree page when successfully authenticated using GitHub

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

2. AS-2

   Type: Automatic

   Initial State: User is not logged in

   Input: User tries to enter a restricted page using the URL

   Output: The user is redirected back to the home page to log in

   Test Case Derivation: This test is to verify that the auth service is correctly verifying users and their authorization when completing an operation or accessing a webpage.

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

3. AS-3

   Type: Automatic

   Initial State: User is logged in

   Input: User clicks the log out button

   Output: The user is redirected back to the home page

   Test Case Derivation: This test is to verify that the user can successfully exit out of the system and remove any authentication like cookies.

   How test will be performed: The test will be performed using NodeJS testing framework Jest.

...

References

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

Reference the Symbolic parameters section in the SRS.

7.2 Usability Survey Questions

The Usability Survey contains the following questions:

- On a scale of 1 to 10, how minimalist and simplistic would you rate the design to be? (1 being very convoluted and 10 being very simple)
- Are you familiar with LaTeX? (Yes or No question)
- Approximately how long in minutes did it take you to learn the system?
- On a scale of 1 to 10, how easy would you say the system material was to understand? (1 being difficult and 10 being very easy to understand)
- On a scale of 1 to 10, how responsive would you say the system is? (1 being unresponsive and 10 being very responsive)
- On a scale of 1 to 10, how appropriate and informative were the errors you seen? (1 being very little and 10 being very much)
- Are there any errors you think of as important that the system does not notify?

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:

1. **What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project? Examples of possible knowledge to acquire include domain specific knowledge from the domain of your application, or software engineering knowledge, mechatronics knowledge or computer science knowledge. Skills may be related to technology, or writing, or presentation, or team management, etc. You should look to identify at least one item for each team member.**

   As a collective the team will need acquire the following sets of skills to be able to successfully validate and test the UnderTree Application:

   - Testing concurrent system applications
   - System testing using selenium
   - Testing front-end and UI using Jest.

   Additionally, while working on the documentation for the project for the months and finally at a time where implementation of the documentation is needed for the proof of concept demonstration. Each team member also identified an individual skill that needs to be improved or acquired to complete the implementation of the project and future documentation successfully. The general knowledge and individual skills that each team member requires is:

   - **Faiq:** Time Management
   - **Veerash:** Leadership
   - **Kevin:** Organization
   - **Eesha:** Initiative

2. **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?**

Since all the technical skills for testing are coding related, the 3 best approaches to learn these skills would be to read documentation on them, try a very simple project that utilizes them or read an article/tutorial.

For each of the technical skills each team member chose to use the following approach:

- **Faiq**: I believe for testing concurrent systems, the knowledge needed is more focused on the actual theory rather than the implementation. As a result for this, it would be better to read documentation and articles to better understand how to properly test concurrent systems and how to resolve the many issues we may face when doing so. Meanwhile on the other hand, since testing with Selenium and Jest simply requires the actual knowledge about the framework, it would be easier to understand them by doing a small project.
- **Veerash**: For front-end & system testing, I would prefer to use official documentation and read up articles. The reasoning behind it is that I have worked with selenium and other front-end testing frameworks before, and I would expect Jest to work similarly, thus what is important for me is to figure out the syntax and names for the various functions available which can easily be done through official documentation and articles. On the other hand, testing concurrent system application would require me to set up a small project which I can start to learn testing with. The reasoning behind is that I cannot jump into testing our capstone project right away since it is a big project with many intermingles components. A good learning point for me will be to set a predictable small project and truly understand how concurrent testing works.
- **Eesha**: For testing concurrent systems I would take the approach of trying a beginner project. This is because concurrency is a complex topic and can easily become overcomplicated with theory so I think that taking an application approach would better help to understand the implementation. Conversely I think that front end testing requires minimal theory and would also be best demonstrated through a practise project. I believe reading documentation for system testing would be a good idea because it is more open ended and would benefit from multiple perspectives.
- **Kevin**: I would use the approach of reading an article/tutorial to master the technical skills of system testing with Selenium and testing front end and UI with Jest. This is mainly because I have already had experience

working with these skills a bit and need more of a refresher. So for me, documentation is enough to recap topics and I'm familiar enough to understand advanced topics. However, I would use the method of making a simple project that utilizes the skill of testing concurrent systems since this is a new topic because in my opinion I feel like concurrent systems have behaviour that's hard to predict and understand in theory, whereas if I were to practically apply it with a simple use case to learn, I would learn the skill a lot more thoroughly and better.

As for the general soft skills needed, each team member identified the following approaches needed to improve in their general skills and which approach they prefer:

- **Faiq**: The two approaches that I identified for better time management are creating a schedule to properly allocate time for capstone along with other classes, or completing assignments and projects as soon as I can rather than waiting for deadline. I believe the first approach is better since it gives me fixed deadlines within my schedule. The second approach is much less rigid on the other hand, and leaves room for me to slack and not make any noticeable improvement in time management.
- **Veerash**: The two approaches that I identified for improving my leadership skill is to have more introspection regarding my decisions I make with this capstone group and to improve my discipline. I go back to situations and think about the way they were handled, figure out what could have been done better, and improve those things the next time something similar comes up. This iterative process is what will help me improve my leadership skills over time. Good leaders tend to have good discipline, and I am lacking proper discipline like working on tasks during allocated time. Improving my discipline will then improve my leadership skills, therefore I will start out by being more disciplined for small tasks, and try to build a habit.
- **Eesha**: Two approaches I identified to build initiative is to take charge of group meetings, and to get started on deliverables without any instruction from other group members. I prefer taking charge of group meetings because it will simultaneously also build leadership skill. Also, sometimes starting a project without other group members will have a negative impact on teamwork.

- **Kevin**: The two approaches that I identified to improve in is the skill of organization are to use a Calendar and to set reasonable deadlines for each task. These two methods go hand in hand because first the calendar would be vital in laying out what tasks exist that need to be done within the week and then setting deadlines that seem reasonable for each task so that I am able to have ample time to complete each task.