# System Verification and Validation Plan for Room8

Team 19
Mohammed Abed
Maged Armanios
Jinal Kasturiarachchi
Jane Klavir
Harshil Patel

2024-11-04

# Revision History

| Date | Version | Notes |
|---|---|---|
| 2024-11-04 | 0.0 | Document Created |
| 2025-04-04 | 1.0 | Changed Multiple FR and NFR test cases to remove mention of score and add tasks instead |

# Contents

# List of Tables



# List of Figures

# 1 Symbols, Abbreviations, and Acronyms

| **Acronym** | **description** |
|---|---|
| SRS | Software Requirements Specification |
| VnV | Verification and Validation |
| CI/CD | Continuous Integration and Continuous deployment |
| API | Application Programming Interface |

This document provides the VnV plans for all relevant aspects of this project. These include the VnV plans for the SRS, system architecture, implementation, and requirements. In addition, this document outlines the expected software and tools to be used in the VnV and the expected roles/responsibilities of the project team.

This document begins with a General Information Section that acts as a primer to the project and contains all the background information needed to understand the document.

Following the primer, Section 3 outlines the VnV plans for project and is followed by System Tests for all the requirements outlined in the SRS. In future revisions, a 5th section outlining detailed unit tests will be added to this document, but at the time of writing the detailed design had not been completed.

# 2 General Information

## 2.1 Summary

Shared living environments often create tension between roommates. It can be inconvenient and sometimes awkward to explicitly reach out to a roommate about the messes they leave behind in a shared space, concerns over bill payments, household upkeep, and more. Our project, Room8, aims to prevent these cumbersome interactions by providing an application and the necessary hardware components to help monitor the cleanliness of shared spaces, schedule tasks, track shared billing cycles, and alert roommates of any issues within the shared house.

## 2.2 Objectives

The objective for this document, and the VnV plan that it outlines, is to support the idea that all software built for the project is correct, complete, and provides a satisfying user experience. The reason why these qualities are emphasized are due to the fact that the outputs of our system have real-life implications (i.e. cleanliness detector determining the incorrect roommate made a mess, bill splitter incorrectly calculating amounts due, etc.), making

correctness and completion essential. Additionally, since the application is very user-centric, the user experience is central to the project's success and must be treated as such. Since this will be the main focus of our testing efforts, there are certain elements of our project which will not be tested extensively by our own team. For example, any external libraries that will be used as a part of Room8 will be assumed to have rigorous verification and validation already completed by the implementation team who created it. Additionally, extreme scalability/load testing will not be completed as part of the initial VnV. This is due to the hypothetical nature of the project in its current state, which doesn't require the system to be able to withstand many concurrent users at a time.

### 2.3 Challenge Level and Extras

The challenge level of our project is general. The extras that will be completed as part of this project include additional features related to student housing and user documentation. Since the main functionality of Room8 is the cleanliness detector and image processing is not a novel concept (mess detecting software already exists), the challenge level is not advanced. However, the program still presents a variety of difficulties on the implementation side, preventing it from being classified as basic. The extras include the user-facing software, which provides all of the other student housing features (i.e. bill splitter, scheduler, etc.), and a thorough guide for the application and installation of the hardware will be delivered upon the project's completion.

### 2.4 Relevant Documentation

The SRS will be referenced throughout the VnV process, as it contains all of the system requirements, which will be verified and validated through testing. Every requirement in the SRS will be referenced by one or more test(s) in this document to ensure that the system is functioning as expected and that everything required from the system is being completed.

## 3 Plan

The following section outlines the execution plan for verifying project documentation and validating the software. The sections begin outlining the

verification and validation team. Following section 3.1, sections 3.2 and 3.4 outline the plans for verifying documentation, while sections 3.5-3.7 discuss verification and validation for the system.

## 3.1 Verification and Validation Team

Table 1: **Verification and Validation Team**

| **Teammate** | **Position** | **Project Role** |
|---|---|---|
| Mohammed Abed | Developer | Mohammed's role will be focused on automating test cases using online tools such as GitHub actions and any other tools the team chooses to use. Since Mohammed's strengths and past experiences are in web development, he will also provide support in developing tests for the user-facing application. |
| Maged Armanios | Developer | Maged's role will focus on developing unit and usability tests for the user-facing application and its backend. These tests will be then automated by Mohammed using tools like GitHub Actions. |
| Jinal Kasturiarachchi | Developer | Jinal will verify the usability requirements for the system. In addition, due to the expected workload of each role being unknown at the time of writing, Jinal will support any other developer who has received a larger-than-average workload. |
| Jane Klavir | Developer | Jane's role will have her verify the cleanliness management system meets the requirements outlined. This includes that it'll be able to detect created messes, cleaned messes, and states of no change. |

| Teammate | Position | Project Role |
|---|---|---|
| Harshil Patel | Developer | Harshil will focus on testing the safety and security of the system. This includes ensuring all data is stored securely, access tokens are confidential, and authorization is secure on the user's end. |
| Dr. R Tharmarasa | Supervisor | Dr. Tharmarasa's role will be to provide the team with support in making decisions related to the cleanliness management algorithm. Additionally, Dr. Tharmarasa will provide his input on the feasibility of requirements related to the cleanliness management system using his research experience in classification and related topics. |

**Note about project roles**: While each team member has specific roles, all members of the team are expected to be able to support each other when required. The roles defined in the table above are simply the optimal roles for each member based on their expertise and do not restrict the work they are able to undergo to only what's outlined in the table above.

## 3.2 SRS Verification Plan

To verify our SRS, the development team will meet with the project supervisor, dr. Tharmarasa. In this meeting, the group will present a summarized version of the SRS which outlines the components, functionality, constraints, and assumptions. This presentation will be accompanied by some form of presentation of the use-case scenarios outlined in the SRS. This can be done with either a low-fidelity mock-up or a live demo of the system if sufficient development has been completed. After the presentation, we will ask the supervisor whether they found any use cases unimportant and requirements unverifiable. After meeting with our supervisor, the team will redraft the requirements and use cases according to his feedback.

In the event we are unable to meet with our supervisor for review, an alternative approach would be to seek peer review and feedback from our primary project reviewers. The meeting would have a similar structure to that which,

would be done with Dr. Tharmarasa.

Additionally, the development team will review the SRS document to verify each requirement:

- Is free of implementation details
- Does not include acceptance criteria
- Does not conflict with any other requirement

## 3.3 Design Verification Plan

The system architecture will be verified by the development team in a formal meeting. The meeting will have each member read through the system architecture and verify it according to the following checklist:

- For each requirement and/or use case defined in the SRS, does the architecture enable the system to do so?
- Is there no overlap between the modules (separation of concerns)?
- Is the system easy to scale (consider both horizontal and vertical scaling)? If you were to scale the system, how would you do so?
- Does the architecture include security measures to satisfy the non-functional requirements in the SRS?
- Does the architecture support unit testing?
- Does the architecture support the addition of new modules (extension)?

## 3.4 Verification and Validation Plan Verification Plan

The Verification and Validation Plan will be read by each member of the team to confirm the following:

- Each teammember referenced has clear, defined roles with a clause for role switching and overlapping roles
- Each test satisfies the following:

- The test corresponds to a requirement in the SRS
- Functional requirements have test cases specific enough that a test can be built with the data. Including clearly defined inputs and outputs
- Functional test cases are deterministic
- Nonfunctional test cases are specific and provide enough information that someone can run the test once the product is built.

## 3.5 Implementation Verification Plan

The system implementation will be verified using a plethora of tools and techniques. First, the implementation of the system will be verified using the system tests and unit tests outlined in sections 4 and 5 (5 is omitted until the detailed design is complete). Additionally, the development team will create integration tests to see if the interactions between different modules are expected. Any tests that can be used during CI/CD use will be done with them to ensure changes to the source code do not create a loss of functionality. Finally, the team will perform an acceptance test, running the application and acting out the key use-case scenarios outlined in the SRS.

During development, the team will use static analysis tools to help minimize common development mistakes such as referencing non-existent variables, incorrect type operations, etc.

Additional tests that can be executed if required are:

- Code walkthroughs with the development team
- Review sessions with our team's primary reviewers

## 3.6 Automated Testing and Verification Tools

At the time of writing, the programming languages have not been finalized. Nonetheless, this section outlines the expected tools that will be utilized for the expected programming languages and frameworks. Currently, the expected programming languages and frameworks are expected to be JavaScript/TypeScript for the frontend application, utilizing the React framework, and Python for the cleanliness management system.

**TypeScript**:

- Jest: The preferred framework for testing React-based applications
- Puppeteer: Provides high-level APIs to control Chrome and allows for automating UI testing, form submissions, and keyboard inputs easily
- ESLint: A code linter that helps catch common programming mistakes and can be enhanced to enforce code styling. Automates the detection of potential vulnerabilities and life-cycle methods for React-based projects.

**Python**:

- Pylance: A language server providing tools features like IntelliSense, logical linting, code actions, code navigation, and semantic colorization. These features can assist developers in catching errors while developing.
- Flake8: A command-line utility for enforcing style consistency across Python projects. Includes lint checks.
- Pytest and PyUnit: Testing frameworks.

**CI/CD Tools**:

- GitHub Actions: Allows for the automation of builds and tests

**API Testing Tools**:

- Postman: Can run API functional and performance tests to determine average and median API response times. Postman can also be used to ensure the correctness of an API's response

## 3.7 Software Validation Plan

In the SRS verification plan, a meeting with the project supervisor, dr. Tharmarasa was mentioned, which had him review the use cases and requirements of the project. This session will also act as part of the software validation plan since Dr. Tharmarasa is independent of the development and project team. Additionally, feedback from primary reviewers and primary stakeholders (people living in shared living spaces) will be taken into consideration and possibly used to revise some requirements.

# 4 System Tests

This section outlines the tests that will be used to verify and validate that Room8 is meeting its requirements. 4.1 and 4.2 designate tests for functional and nonfunctional requirements respectively, and tests are furthermore split into Room8's various modules for ease of readability and traceability. The output section of each test case describes the pass condition, thus this is the criteria the app developers will use to confirm that the app aligns with requirements.

## 4.1 Tests for Functional Requirements

Functional requirements pertain to the core functionality of the app. Room8 has the core functionality of registering users into a home, deploying a ChatBot for notifications, establishing a cleanliness managegement system, providing an event scheduler, and implementing a bill-splitter. The following tests will provide evidence of these integral components being successfully fulfilled.

### 4.1.1 User Authentication and House Management

When a group of students chooses to adopt Room8 to help with their house, the first step is to register all the members and connect them into a common home group through the app. The tests in this section realize this fundamental first step, and are in conjunction with requirements FR211-FR218.

| **FST-UAHM-1** | **Profile Creation** |
|---|---|
| **Description** | Tests if the system is able to handle creating user profiles in the system from frontend inputs |
| **References** | FR211 |
| **Type** | System Test (dynamic, automated) |
| **Initial State** | Frontend inputs are empty. User profile does not exist in the system |
| **Input** | Valid Google account sign in |
| **Output** | User profile entry is created in the system using the Google account. Frontend redirects user to the dashboard |
| **Procedure** | Testing agent navigates to the account creation page of the application. Agent uses exisiting Google account. Agent submits account creation |

| **FST-UAHM-2** | **System Login** |
|---|---|
| **Description** | Tests if the system is able to recognize a user profile and to authenticate them to the system |
| **References** | FR212 |
| **Type** | System Test (dynamic, automated) |
| **Initial State** | Testing profile exists in the system. System does not have the profile currently authenticated |
| **Input** | Valid Google account matching the profile in the system |
| **Output** | User profile is authenticated with a new session entry. Frontend redirects user to the dashboard |
| **Procedure** | Testing agent navigates to the account login page of the application. Agent enters a valid testing Google account. Agent submits account login |

| FST-UAHM-3 | System Logout |
|---|---|
| **Description** | Tests if the system logs the user out, invalidating their session and returning them to the homepage |
| **References** | FR213 |
| **Type** | Functional Test (dynamic, automated) |
| **Initial State** | Testing profile is logged in with an active session |
| **Input** | User clicks on the logout button |
| **Output** | The system invalidates the session and redirects the user to the homepage |
| **Procedure** | Testing agent clicks the logout button and confirms that the user is logged out and redirected to the homepage |

| FST-UAHM-4 | Create Home Instance |
|---|---|
| **Description** | Tests if the system allows users to create a new home group with specified details |
| **References** | FR214, FR216 |
| **Type** | System Test (dynamic, automated) |
| **Initial State** | User is logged in, and no home group exists for their account |
| **Input** | Home name, address, and number of roommates |
| **Output** | A new home group instance is created with the specified details, and the user is added as a member |
| **Procedure** | Testing agent navigates to the home creation page, enters the home name, address, and number of roommates, and submits the form. Confirm that the home instance is created and the user is added as a member |

| FST-UAHM-5 | Invite and Remove Users From Home Instance |
|---|---|
| **Description** | Tests if the system allows users to invite others to join a home group and remove them as needed |
| **References** | FR215, FR217, FR218 |
| **Type** | Integration Test (dynamic, automated) |
| **Initial State** | A home group exists with only the logged-in user as a member |
| **Input** | Valid email address of another user to invite; selection of a user to remove |
| **Output** | The invited user receives an invitation, joins the home group upon acceptance, and can be removed by the home admin |
| **Procedure** | Testing agent invites another user via their email. Once accepted, confirm the user is added to the home group. Then, the agent removes the user and confirms their removal from the group. |

| FST-UAHM-6 | Leave Home Instance |
|---|---|
| **Description** | Tests if the system allows users to leave a home group as needed |
| **References** | FR216, FR218 |
| **Type** | Functional Test (dynamic, automated) |
| **Initial State** | A home group exists with user as a member |
| **Input** | ID of a home group to leave |
| **Output** | User receives feedback on their action to leave the home group |
| **Procedure** | Testing agent views home group then selects the option to leave the home group. The testing agent then verifies that the user has left the home group. |

### 4.1.2 ChatBot Configuration

The ChatBot is configured to send necessary reminders and notifications to the group. To ensure the ChatBot is accomplishing its role, the following tests for FR221-FR224 will be used.

| **FST-CC-1** | **Update Chatbot Settings** |
|---|---|
| **Description** | Tests if the chatbot settings can be updated successfully by an authorized user |
| **References** | FR221 |
| **Type** | Functional Test (dynamic, automated) |
| **Initial State** | Chatbot is configured with default settings |
| **Input** | Randomized array of include and exclude inputs |
| **Output** | ChatBot settings entry in database reflect the array of inputs |
| **Procedure** | Testing agent navigates to the chatbot settings page, iterates through each setting checkbox corresponding to randomized boolean input array. Agent saves changes. Database is verified to contain the updated array of inputs |

| **FST-CC-2** | **GroupChat Created with ChatBot** |
|---|---|
| **Description** | Tests if all users attached to a home instance are added to a groupchat alongside the ChatBot |
| **References** | FR225 |
| **Type** | Integration Test (dynamic, manual) |
| **Initial State** | Agent has user profile and is part of a home instance with more than 2 users linked. Chatbot phone number is not present in the group chat. ChatBot has no corresponding groupchat entry |
| **Input** | User clicks "Create GroupChat" button |
| **Output** | All users are added to the group chat alongside the ChatBot |
| **Procedure** | Testing agent navigates to the group chat creation page and clicks the "Create GroupChat" button. Agent verifies that all users are added to the group chat alongside the ChatBot |

| FST-CC-3 | ChatBot Sends Messages to Groupchats |
|---|---|
| **Description** | Tests if the ChatBot is able to send messages to a group chat as per configuration |
| **References** | FR222, FR223, FR224 |
| **Type** | System Test (dynamic, automated) |
| **Initial State** | Chatbot is added to a group chat with configured message triggers |
| **Input** | New database entry created for a chore, cleanliness task, and a shared expense |
| **Output** | Chatbot sends the appropriate configured messages to the group chat |
| **Procedure** | Testing agent creates a new database entry for a chore, cleanliness task, and a shared expense. Agent verifies that the ChatBot sends the appropriate messages to the group chat according to the configured ChatBot settings |

### 4.1.3 Cleanliness Manager

The cleanliness manager is the unique selling point of the Room8 app. It assesses the cleanliness of spaces by running an algorithm to output a list of cleanliness tasks which consists of added, moved and removed objects. To ensure this functionality is working properly (i.e. FR231-FR235), the following tests are used.

| **FST-CM-1** | **System Evaluates Cleanliness** |
|---|---|
| **Description** | Tests if the system can evaluate and record changed objects based on captured images |
| **References** | FR231 |
| **Type** | System Test (dynamic, manual) |
| **Initial State** | Cleanliness management camera is setup and connected to home instance |
| **Input** | N/A |
| **Output** | Cleanliness tasks are updated based on user actions |
| **Procedure** | Home resident enters camera view and creates a mess. Resident verifies that the cleanliness tasks are updated to reflect the mess. Resident then cleans the mess and updates the cleanliness task to reflect the clean state |

| **FST-CM-2** | **Display Cleanliness Tasks** |
|---|---|
| **Description** | Tests if the system displays cleanliness tasks to users in a group |
| **References** | FR232, FR233, FR234, FR235 display requirements |
| **Type** | Functional Test (dynamic, automated) |
| **Initial State** | Cleanliness management system is setup for the group |
| **Input** | N/A |
| **Output** | Cleanliness tasks are displayed accurately to the user |
| **Procedure** | Testing agent navigates to the cleanliness management page and verifies that the tasks are shown accurately |

### 4.1.4 Schedule Configuration

Room8 provides a convenient calendar to allow users to schedule chores and events and stay organized, as outlined by requirements FR241-FR245. The tests here will be used to ensure this component of the app is functioning correctly.

| **FST-CM-1** | **System Evaluates Cleanliness** |
|---|---|
| **Description** | Tests if the system can evaluate and record cleanliness tasks based on captured images |
| **References** | FR231 |
| **Type** | System Test (dynamic, manual) |
| **Initial State** | Cleanliness management camera is setup and connected to home instance |
| **Input** | N/A |
| **Output** | Cleanliness tasks are updated based on user actions |
| **Procedure** | Home resident enters camera view and creates a mess. Resident verifies that the cleanliness tasks are updated to reflect the mess. Resident then cleans the mess and verifies that the cleanliness tasks are updated to reflect the clean state |

| FST-CM-2 | Display Cleanliness Tasks |
|---|---|
| **Description** | Tests if the system displays cleanliness tasks to users in a group |
| **References** | FR232, FR233, FR234, FR235 |
| **Type** | Functional Test (dynamic, automated) |
| **Initial State** | Cleanliness mangement system is initialized and ready for use |
| **Input** | N/A |
| **Output** | Cleanliness tasks are displayed accurately to the user |
| **Procedure** | Testing agent navigates to the cleanliness tasks display page and verifies that the tasks are shown accurately |

### 4.1.5 Schedule Configuration

Room8 provides a convenient calendar to allow users to schedule chores and events and stay organized, as outlined by requirements FR241-FR245. The tests here will be used to ensure this component of the app is functioning correctly.

| **FST-SC-1** | **Adding Event to Schedule** |
|---|---|
| **Description** | Tests that new event appears in calendar after it is inputted. |
| **References** | FR241, FR242, FR243, FR245 |
| **Type** | Functional Test (dynamic, automated) |
| **Initial State** | Calendar displaying chores and events pertaining to the house. |
| **Input** | Title, date, time, and duration. |
| **Output** | Calendar displaying new chore/event with input parameters. |
| **Procedure** | Testing agent navigates to calendar, generates a series of events with random inputs, and adds them to the schedule. Agent verifies that the events with the same input parameters exist in the schedule. |

| **FST-SC-2** | **Removing Event from Schedule** |
|---|---|
| **Description** | Tests that event disappears from calendar after existing event is removed. |
| **References** | FR244 |
| **Type** | Functional Test (dynamic, automated) |
| **Initial State** | Calendar displaying chores and events pertaining to the house. |
| **Input** | Chore/event from calendar represented by name, date, time, and duration. |
| **Output** | Chore/event that was removed is no longer displayed in its previous timeblock in the calendar. |
| **Procedure** | Testing agent navigates to calendar, and removes the series of events it generated in FST-SC-1. Agent verifies that the events it removed do not exist in the schedule. |

| **FST-SC-3** | **Editing Event in Schedule** |
|---|---|
| **Description** | Tests that changes can be made to an already-existing event. |
| **References** | FR244 |
| **Type** | Functional Test (dynamic, automated) |
| **Initial State** | Calendar displaying chores and events pertaining to the house. |
| **Input** | Chore/event from calendar represented by name, date, time, and duration. |
| **Output** | Chore/event displayed in calendar after edits contains updated information (i.e. name, date, time, and/or duration). |
| **Procedure** | Testing agent navigates to calendar, and adds an event with random input. Then, testing agent edits that event by changing name, date, time, and duration one-by-one, verifying each time that the parameter edited has changed. Then, testing agent edits 2 random parameters simultaneously, then 3 random parameters simultaneously, and finally all the parameters simultaneously, verifying after each step. |

| FST-SC-4 | Display Event Schedule |
|---|---|
| **Description** | Tests that all events displayed in calendar view. |
| **References** | FR245 |
| **Type** | Functional Test (dynamic, automated) |
| **Initial State** | User is logged in and registered in home group |
| **Input** | User requests to view calendar |
| **Output** | Calendar displaying chores and events pertaining to the home group |
| **Procedure** | Testing agent navigates to calendar and confirms that chores and events are displayed accurately. |

#### 4.1.6 Bill Splitter Configuration

Various expenses will inevitably accumulate for a group of roommates. The bill splitter keeps track of expenses and amounts owed by each member. The Room8 app must be able to deal with expenses accurately (as covered by FR251-FR255) and these tests will serve as confirmation.

| FST-BSC-1 | Add and Split Expense |
|---|---|
| **Description** | Tests if the system allows users to add a new expense and splits it among group members |
| **References** | FR251 |
| **Type** | System Test (dynamic, automated) |
| **Initial State** | No expenses recorded for the current month |
| **Input** | New expense with an amount and shared expense roommates |
| **Output** | Expense is split equally among group members and recorded |
| **Procedure** | Testing agent navigates to the add expense page, enters details, and verifies that the expense is split among group members |

| FST-BSC-2 | Display User Expenses |
|---|---|
| **Description** | Tests if the system displays all expenses associated with a user |
| **References** | FR252, FR253. FR255 |
| **Type** | Functional Test (dynamic, automated) |
| **Initial State** | Recorded expenses are available in the system |
| **Input** | User navigates to their expenses page |
| **Output** | System displays all expenses associated with the user |
| **Procedure** | Testing agent navigates to the user expenses page and verifies that all expenses are displayed correctly |

| FST-BSC-3 | Mark Expense as Paid |
|---|---|
| **Description** | Tests if the system allows users to mark an expense as paid |
| **References** | FR253, FR254 |
| **Type** | Functional Test (dynamic, automated) |
| **Initial State** | An unpaid expense is available in the system |
| **Input** | User navigated to an expenses page and marks an expense as paid |
| **Output** | System updates the UI state and displays all expenses associated with the user |
| **Procedure** | Testing agent navigates to the user expenses page and marks an expense as paid. The system then refreshes the UI and the testing agent will verify that the marked expense is displayed as paid |

## 4.2 Tests for Nonfunctional Requirements

Nonfunctional requirements are the quality attributes of the system. Covering nonfunctional requirements ensures that Room8 is operating well, and this subsection contains such tests for the nonfunctional requirements outlined in the SRS.

### 4.2.1 Area of Testing: User Authentication and House Management

This section covers encryption standards for authentication and house management data of the application, as presented in SRS, Section S.2.1.

| **NFST-UAHM-1** | **Encryption of Authentication Data** |
|---|---|
| **Description** | Tests if system encrypts data in transit and when stored |
| **References** | NFR211 |
| **Type** | System Test (Automated) |
| **Initial State** | Frontend inputs for login are filled with user data. User profile exists in the system. |
| **Input** | User inputted data in login elements. |
| **Output** | Stored and data in transit are encrypted. |
| **Procedure** | Capture data in transit between client and server, use network monitoring tools to inspect data transmission and check database storage for encryption compliance. |

| **NFCT-UAHM-2** | **Sensitive Information Error Management** |
|---|---|
| **Description** | Tests if the system does not reveal sensitive user information during login attempts |
| **References** | NFR212 |
| **Type** | Compliance Test (Manual) |
| **Initial State** | Login elements are filled with user data. User profile exists in system. |
| **Input** | Valid email and incorrect password. |
| **Output** | Error message that avoids revealing sensitive information |
| **Procedure** | Attempts to login with incorrect password but correct email, observing error message that is displayed on frontend. |

| **NFST-UAHM-3** | **Encryption of House Data** |
|---|---|
| **Description** | Tests if system encrypts all data related to houses in transit and at rest. |
| **References** | NFR213 |
| **Type** | System Test (Automated) |
| **Initial State** | House data stored in the database and transmitted over the network. |
| **Input** | View or edit house data. |
| **Output** | Verification that data in transit and stored data are encrypted. |
| **Procedure** | Use network monitoring tools to inspect data transmission and check database storage for encryption compliance. |

| **NFPT-UAHM-4** | **Authentication Response Time** |
|---|---|
| **Description** | Tests if system authenticates a user within a median response time of under 1 second. |
| **References** | NFR214 |
| **Type** | Performance Test (Automated) |
| **Initial State** | On login page and user profile exists. |
| **Input** | Correct user login credentials filled in login elements. |
| **Output** | Authentication process completed within one second. |
| **Procedure** | Measure the time from when login credentials are submitted until a response is received using performance monitoring tools. |

### 4.2.2 Area of Testing: ChatBot Configuration

This section covers the ChatBot's compliance to not disclose sensitive data and minimize user interaction, as presented in SRS, Section S.2.2.

| **NFCT-CC-1** | **Sensitive Information Non-Disclosure** |
|---|---|
| **Description** | Tests if the chatbot avoids disclosing sensitive user data. |
| **References** | NFR221 |
| **Type** | Compliance Test (Manual) |
| **Initial State** | ChatBot configured and messaging application chat with ChatBot open. |
| **Input** | Message box with query prompting to reveal sensitive data. |
| **Output** | ChatBot responds without sensitive information like addresses or full names. |
| **Procedure** | Send message to ChatBot asking for sensitive information and verify reply from ChatBot complies with non-disclosure requirements. |

| NFST-CC-2 | User Messaging Frequency |
|---|---|
| **Description** | Tests ChatBot to ensure it reasonably messages user and not too frequently. |
| **References** | NFR222 |
| **Type** | System Test (Manual) |
| **Initial State** | ChatBot configured and active for user interaction. |
| **Input** | Trigger multiple notifications. |
| **Output** | Verify notifications are not excessively frequent. |
| **Procedure** | Initiate different prompts and events to trigger notifications and monitor frequency to ensure user is not annoyed. |

### 4.2.3 Area of Testing: Cleanliness Manager

This section covers the Cleanliness Manager's compliance, encrypt and store pictures of high quality safely and correctly process and detect messes within specified time, as presented in SRS, Section S.2.3.

| NFST-CM-2 | Encryption of Images and Videos |
|---|---|
| **Description** | Tests system to ensure encryption and secure storage of images and videos. |
| **References** | NFR233 |
| **Type** | System Test (Automatic) |
| **Initial State** | Cleanliness manager is initialized and ready for use. |
| **Input** | Captured image and recording of shared space. |
| **Output** | Image recording in storage are encrypted. |
| **Procedure** | Capture image and record space using the system and verify encryption in the database. |

| **NFST-CM-3** | **Image Quality Validation** |
|---|---|
| **Description** | Test to check if captured image have sufficient quality. |
| **References** | NFR234 |
| **Type** | System Test (Manual) |
| **Initial State** | Cleanliness manager is initialized and ready for use. |
| **Input** | Captured image of shared space. |
| **Output** | Image of shared space. |
| **Procedure** | Capture image and manually verify clarity and object visibility. |

| **NFPT-CM-4** | **Image Processing Time** |
|---|---|
| **Description** | Test to ensure images are processed within time frame. |
| **References** | NFR235 |
| **Type** | Performance Test (Automatic) |
| **Initial State** | Image is taken and sent over to server/database ready for processing. |
| **Input** | Uploaded image of shared space. |
| **Output** | Image after process of cleanliness manager within time of 30 minutes. |
| **Procedure** | Tracking time between start of image process to end with processing done within 30 minutes. |

| NFCT-CM-5 | False Event Prevention |
|---|---|
| **Description** | Test to check if system avoids false report. |
| **References** | NFR236 |
| **Type** | Compliance Test (Manual) |
| **Initial State** | Cleanliness manager is initialized and ready for use. |
| **Input** | User uses shared space but does not make a mess. |
| **Output** | Cleanliness manager does not output change to tasks. |
| **Procedure** | User activities are simulated where no mess is made and the cleanliness manager outputs cleanliness tasks and verify tasks are not added. |

| NFST-CM-6 | End of Use |
|---|---|
| **Description** | Test to check if system detects that the user is no longer using the space. |
| **References** | NFR237 |
| **Type** | System Test (Manual) |
| **Initial State** | Cleanliness manager is initialized and ready for use. |
| **Input** | User uses shared space and leaves after use. |
| **Output** | Final picture. |
| **Procedure** | After simulated use of the shared space and no activity has been detected for a determined period of time, system declares the space is no longer in use and takes final picture. |

### 4.2.4 Area of Testing: Schedule Configuration

This section covers the schedule's ability to accurately store, handle and schedule events with respect to time zones, as presented in SRS, Section

S.2.4.

| **NFIT-SC-1** | **Calendar Event Time Storage and Display** |
|---|---|
| **Description** | Tests calendar to ensure event time in database is in UTC but on user's application is displayed in user's local time. |
| **References** | NFR241, NFR242 |
| **Type** | Integrational Test (Automatic) |
| **Initial State** | User is on calendar section of application. |
| **Input** | Event scheduled with specific time and date. |
| **Output** | Event in database in UTC and on user's calendar in their local time. |
| **Procedure** | Local timezone on test device is not UTC, event is created in the app, verify event can be seen on calendar in local time, verify in database event is in UTC. |

| **NFST-SC-2** | **Event Granularity** |
|---|---|
| **Description** | Test calendar event scheduler to ensure events have granularity of five minutes. |
| **References** | NFR243 |
| **Type** | System Test (Automatic) |
| **Initial State** | User is on calendar section of application. |
| **Input** | Creating event and setting time. |
| **Output** | Options to set minutes should be multiples of five. |
| **Procedure** | In calendar screen create event, in setting event time section verify the setting the minute of the event is in intervals of five. |

| NFST-SC-3 | Events Are Encrypted |
|---|---|
| **Description** | Tests if system encrypts stored events. |
| **References** | NFR244 |
| **Type** | System Test (Automatic) |
| **Initial State** | Calendar is active and has events created. |
| **Input** | Creating event. |
| **Output** | Event is encrypted in transit and storage. |
| **Procedure** | Check event data in the database and transit and verify encryption using network monitoring tools. |

### 4.2.5 Area of Testing: Bill Splitter Configuration

This section covers the bill splitters's ability to accurately and precisely handle data, as presented in SRS, Section S.2.5.

| NFST-BSC-1 | Events Are Encrypted |
|---|---|
| **Description** | Tests if system encrypts bill splitter events. |
| **References** | NFR251 |
| **Type** | System Test (Automatic) |
| **Initial State** | Bill splitter is active and has history of events. |
| **Input** | Bill splitting is used. |
| **Output** | Event is encrypted in transit and storage. |
| **Procedure** | Check bill splitting event data in the database and transit and verify encryption using network monitoring tools. |

| NFCT-BSC-2 | Precision of Numerical Values |
|---|---|
| **Description** | Tests bill splitter's ability to calculate and display numerical values with preicion of two decimal places. |
| **References** | NFR252 |
| **Type** | Compliance Test (Automatic) |
| **Initial State** | Bill splitter is active . |
| **Input** | Prices entered to split between two users. |
| **Output** | Price each user owes is display. |
| **Procedure** | Price entered does not evenly divide by two (e.g. 33.15), verify price displayed is rounded to two decimal places and is stored in database with two decimals of precision. |

## 4.3 Traceability Between Test Cases and Requirements

Here are the traceability matrices for the functional and non-functional requirements to the test cases. The requirements are listed in the rows and the test cases are listed in the columns. An "X" indicates the functional or non-functional requirement is tested by the test case.

| | FST-UAHM-1 | FST-UAHM-2 | FST-UAHM-3 | FST-UAHM-4 | FST-UAHM-5 | FST-UAHM-6 | FST-CC-1 | FST-CC-2 | FST-CC-3 | FST-CM-1 | FST-CM-2 | FST-SC-1 | FST-SC-2 | FST-SC-3 | FST-SC-4 | FST-BSC-1 | FST-BSC-2 | FST-BSC-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FR211 | X | | | | | | | | | | | | | | | | | |
| FR212 | | X | | | | | | | | | | | | | | | | |
| FR213 | | | X | | | | | | | | | | | | | | | |
| FR214 | | | | X | | | | | | | | | | | | | | |
| FR215 | | | | | X | | | | | | | | | | | | | |
| FR216 | | | | X | | X | | | | | | | | | | | | |
| FR217 | | | | | X | | | | | | | | | | | | | |
| FR218 | | | | | X | X | | | | | | | | | | | | |
| FR221 | | | | | | | X | | | | | | | | | | | |
| FR222 | | | | | | | | X | X | | | | | | | | | |
| FR223 | | | | | | | | | X | | | | | | | | | |
| FR224 | | | | | | | | | X | | | | | | | | | |
| FR231 | | | | | | | | | | X | | | | | | | | |
| FR232 | | | | | | | | | | | X | | | | | | | |
| FR233 | | | | | | | | | | | X | | | | | | | |
| FR234 | | | | | | | | | | | X | | | | | | | |
| FR235 | | | | | | | | | | | X | | | | | | | |
| FR241 | | | | | | | | | | | | X | | | | | | |
| FR242 | | | | | | | | | | | | X | | | | | | |
| FR243 | | | | | | | | | | | | X | | | | | | |
| FR244 | | | | | | | | | | | | | X | X | | | | |
| FR245 | | | | | | | | | | | | X | | | X | | | |
| FR251 | | | | | | | | | | | | | | | | X | | |
| FR252 | | | | | | | | | | | | | | | | | X | |
| FR253 | | | | | | | | | | | | | | | | | X | X |
| FR254 | | | | | | | | | | | | | | | | | | X |
| FR255 | | | | | | | | | | | | | | | | | X | |

Figure 1: Functional requirements - tests traceability

| | NFST-UAHM-1 | NFST-UAHM-2 | NFST-UAHM-3 | NFST-UAHM-4 | NFCT-CC-1 | NFCT-CC-2 | NFCT-CM-1 | NFCT-CM-2 | NFCT-CM-3 | NFCT-CM-4 | NFCT-CM-5 | NFCT-CM-6 | NFIT-SC-1 | NFIT-SC-2 | NFIT-SC-3 | NFST-BSC-1 | NFST-BSC-2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NFR211 | X | | | | | | | | | | | | | | | | |
| NFR212 | | X | | | | | | | | | | | | | | | |
| NFR213 | | | X | | | | | | | | | | | | | | |
| NFR214 | | | | X | | | | | | | | | | | | | |
| NFR221 | | | | | X | | | | | | | | | | | | |
| NFR222 | | | | | | X | | | | | | | | | | | |
| NFR231 | | | | | | | X | | | | | | | | | | |
| NFR232 | | | | | | | X | | | | | | | | | | |
| NFR233 | | | | | | | | X | | | | | | | | | |
| NFR234 | | | | | | | | | X | | | | | | | | |
| NFR235 | | | | | | | | | | X | | | | | | | |
| NFR236 | | | | | | | | | | | X | | | | | | |
| NFR237 | | | | | | | | | | | | X | | | | | |
| NFR241 | | | | | | | | | | | | | X | | | | |
| NFR242 | | | | | | | | | | | | | X | | | | |
| NFR243 | | | | | | | | | | | | | | X | | | |
| NFR244 | | | | | | | | | | | | | | | X | | |
| NFR251 | | | | | | | | | | | | | | | | X | |
| NFR252 | | | | | | | | | | | | | | | | | X |

Figure 2: Non-functional requirements - tests traceability

# 5 Unit Test Description

To be included in future revisions.

# 6 Appendix

# Appendix — Reflection

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

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

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

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

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

Answers:

1. While writing this deliverable, the team was able to start much earlier and each individual member of the team took more initiative. This lead to the deliverable being close to completion much earlier (relative to the deadline) than other deliverables for the project. Additionally, we had planned some questions before our meeting with our TA which when answered gave us a lot of clarity about this deliverable so we were able to work with less confusion. Finally, the work items for this deliverable were separated very fairly, with the core sections of this deliverable having multiple people working on them and other sections being delegated to a single person.

2. There were not any major pain points in this deliverable. Some minor pain points that did occur include some confusions about what do with section 3.3 and section 5, as they relate to the design document which at the time of writing, has not been completed. Another minor paint point was that the this document required the SRS to be modified in situtations which we discovered new requirements and since our SRS, this was cumbersome and difficult to coordinate for the multiple group members developing test cases who may have done duplicate work. To deal with this, we created a thread in our group chat on discord where we notified each other when modifications were being done to the SRS.

3. The team will need to learn how to use various tools and testing libraries to complete the verification of the project. Some of these tools and libraries include the automation of tests on GitHub Actions, testing libraries for specific programming languages (Pytest, Jest, etc), and Postman. In addition to just tools and libraries, the team will also need to learn and improve their analytical skills for software. This project involves a lot of analysis on all aspects of it such as requirements, code, and use cases. Learning and improving the teams code review / walkthrough skills, to effectively evaluate / communicate what the code would / should do is essential.

4. For learning a new testing tool or library, our team has decided to rely on the countless resources available on the internet, as well as an opportunity for transferable knowledge. Applying these methods will help with integrating automated testing into our software effectively.
   One approach for mastering a new tool or library is to follow a tutorial or short course readily available online, for instance this Intro

to Postman Youtube series or this Unit Testing in Python course from LinkedIn Learning. Maged Armanios will use this approach because by following a tutorial created by an expert, he can understand important nuances of the software. In addition, he enjoys following tutorials and learns well when doing a guided lesson.

Another approach is to create a small, simple project for unit testing and using the documentation/reference material from the library chosen, integrate the tool on a small scale and keep increasing the coverage until it is adequate for the entire project. For our project, we will use the proof of concept (POC) as our basic starting point to add test automation, and then expand into other modules as we gain familiarity with the chosen tool. Harshil Patel will lead this effort because he has experience working in software testing and test automation. Harshil's foundation in congruent skills gives him confidence that he can successfully integrate a testing library with just the distributor's reference material.

Improving software analytical skills will really improve the functionality of the software and ensure the code is concise, optimal, and scalable/maintainable. There are countless approaches to developing these skills but we believe the ones identified below are most applicable to our project.

The first approach would be to gather all the diagrams from all our documentation, and create more as necessary to really harbour the understanding of exactly how the code will be used and how functions will be fulfilled. Such diagrams include case diagrams, sequence diagrams, and flow charts. This is delegated to Zayn Abed because Zayn is strong with organization and attention-to-detail. He will be able to lay out the flow and functionality of the app and ensure there are no gaps.

The second approach expands on the first one, and it is to use the findings to draw common themes and create a software architecture that incorporates design patterns to accomplish the purpose of each module. Material from previous courses will be used to aid in ultimately creating a UML diagram that encapsulates our software. Using design patterns and principles will ensure scalability, reusability, modularity, testability, and more qualities of good design. Jane Klavir will follow this approach because Jane is passionate about design and bringing together individual parts to make a cohesive whole.

A third approach is to conduct code walkthroughs for our supervisor,

Dr. Tharmarasa, as new elements are implemented. While these walkthroughs do not have to be perfect or formal, they should clearly get the point across to our supervisor so that he can analyse the software effectively and provide his expert evaluation. Jinal Kasturiarachchi will be responsible for doing walkthroughs in meetings with Dr. Tharmarasa because Jinal enjoys presenting and opening the discussion for team to furthermore receive valuable feedback.