**Test Plan Document**

Group 9 Moon Eyes: Case Management System for Metro Detective Agency

Senior Design II Fall 2023

By: Kylie Callison, Erika Valle-Baird, Kendall Gesek & Pamela Shahu

**Table of Contents**

[1.0 Introduction](#_ernmnz26gnhj) 2

[1.1 Goals and objectives](#_6949hjx8kah5) 2

[1.2 Statement of scope](#_lyizptamkcc3) 2

[1.3 Major constraints](#_igcusbwzyd8d) 2

[2.0 Test Plan](#_lt8cyjlx7qt6) 2

[2.1 Software (SCIís) to be tested](#_8esg9cbdi8ph) 3

[2.2 Testing strategy](#_4uic7gcu8zzt) 3

[2.2.1 Unit testing](#_hn7vdzr91viv) 3

[2.2.2 Integration testing](#_koe39fcd7qyy) 3

[2.2.3 Validation testing](#_qzbdh6w9f0xl) 3

[2.2.4 High-order testing](#_uajj4mvkme) 3

[2.3 Testing resources and staffing](#_etjczq75bkbc) 3

[2.4 Test work products](#_fu92r3i3t20q) 4

[2.5 Test record keeping](#_bfq035d3dnrp) 4

[2.6 Test metrics](#_6plq0conbgtz) 4

[2.7 Testing tools and environment](#_k5h5lvyk39d9) 4

[2.8 Test schedule](#_3zpvx76um8d8) 4

[3.0 Test Procedure](#_qcppr6o2qwcb) 4

[3.1 Software (SCIís) to be tested](#_pqoxc177wxcp) 4

[3.2 Testing procedure](#_td4l3h4do6vk) 5

[3.2.1 Unit test cases](#_g3lrlhgdbgpx) 5

[3.2.1.2 Stubs and/or drivers for component i](#_uhqtvs85a7x4) 5

[3.2.1.3 Test cases component i](#_w4kyt7d2v1cl) 5

[3.2.1.4 Purpose of tests for component i](#_8skh6wwy85ls) 5

[3.2.1.5 Expected results for component i](#_tlwfhll3vdkh) 5

[3.2.2 Integration testing](#_ipimpblb8ccy) 5

[3.2.2.1 Testing procedure for integration](#_ardcban6k6v2) 6

[3.2.2.2 Stubs and drivers required](#_qqplummh7vcy) 6

[3.2.2.3 Test cases and their purpose](#_z6xj70ndbwuh) 6

[3.2.2.4 Expected results](#_ui3vlk1i6ptf) 6

[3.2.3 Validation testing](#_lhs1o94bhj9c) 6

[3.2.3.1 Testing procedure for validation](#_nyq9gwbha5om) 6

[3.2.3.3 Expected results](#_wyxy37xngtqt) 6

[3.2.3.4 Pass/fail criterion for all validation tests](#_6zdw5nc8umb2) 6

[3.2.4 High-order testing (a.k.a. System Testing)](#_34vp5kgmcvy7) 6

[3.2.4.1 Recovery testing](#_t36e6l292ahc) 7

[3.2.4.2 Security testing](#_59jvfg25pcd) 7

[3.2.4.3 Stress testing](#_qeesqs71690u) 7

[3.2.4.4 Performance testing](#_3fi1r3pcqdkn) 7

[3.2.4.5 Alpha/beta testing](#_e2v6btc4xolr) 7

[3.2.4.6 Pass/fail criterion for all validation tests](#_go5q2rfievkq) 7

[3.3 Testing resources and staffing](#_2ox6bwsvcty9) 7

[3.4 Test work products](#_8jaq6w54yvz5) 7

[3.5 Test record keeping and test log](#_hf2793xkeigm) 7

## 

## 1.0 Introduction

### 1.1 Goals and objectives

MoonEyes: a case management system for a private detective agency. The main purpose of this project is to create a process that allows agents from Metro Detective Agency to submit and track case information (including client, subject and agent details). This system will be accessible remotely to agents using Amazon EC2. This will allow agents the ability to update case information in real time from anywhere. The testing process for MoonEyes will ensure that the system is performing as intended, no major errors or lags in the system, and the system has met all standards and requirements stated in the Product Backlog. The system must also have an aesthetically pleasing interface, be able to be high performing, and be easy for an agent to navigate and use.

### 1.2 Statement of scope

The scope of the testing will include black-box testing, unit testing, integration testing, performance and smoke testing, and validation testing. Black-Box testing is used to ensure the software is working from a user’s perspective. This will include a series of test cases that are designed from the user’s perspective to guarantee the system is operating as intended and satisfies the business requirements. Unit testing will ensure that each component of the system is working properly before moving on to integration testing. This will include testing boundaries, such as validating information entered into the database, automatic sign out after inactivity, and ensuring the search parameters are displaying the correct information. Integration testing will follow the unit testing to test the interaction between the database, cloud-based web application, and the user interface. This will verify that the information being entered or changed is updating correctly in the database, and the information can be successfully and efficiently queried. The performance testing is to test that the system is able to perform with multiple agents using it at the same time and the information is able to be accessible and updated for all agents. Smoke testing will be used after the system is deployed to verify the application is running correctly in the environment and to determine if more testing is necessary. Lastly, the use of validation testing will ensure that all requirements of MoonEyes have been met and properly implemented in the system, and the application is functioning as intended.

### 1.3 Major constraints

**Funding**

As mentioned, we will need to deploy MoonEyes on a cloud-based hosting service. This means it will cost anywhere from $10-50/month for our client. There could be additional costs through Amazon for more cybersecurity protection for the system. That responsibility will fall on the client if they want to put additional funding to more cybersecurity measures, if they deem that is necessary over time.

**Time**

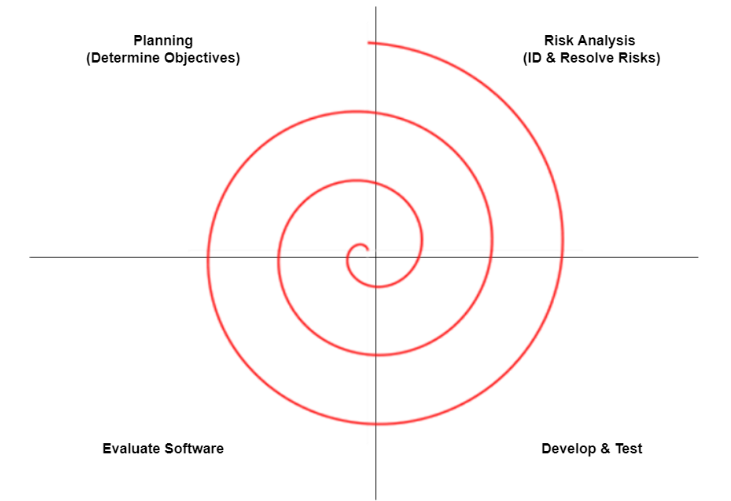
Because this project is for our senior design project we have a strict eight-month development timeline. It is possible that we might want to make improvements or add additional features down the road that are incompatible with the amount of time left. Additionally, all members of the team will also be balancing class and work responsibilities along with our project. Though there is an extensive test plan, the time constraints could limit the amount of testing done on the application.

**Experience**

Because our group consists of students with limited professional development experience, we may have to do more independent and team research than a typical development team. While the team does have experience in the cybersecurity industry, we do not have access to an experienced hacker to test our product. We will have to rely on previous cybersecurity and testing knowledge as well as built in security features from Amazon EC2 and RDS systems to ensure the application is secure and performing as intended.

## 2.0 Test Plan

Due to our client’s evolving and changing needs, we have adopted the spiral approach for both our software development and testing. This approach is depicted below. Such an approach allows us to identify requirements not satisfied for our clients, along with allowing us to both change and implement any future developing customer needs. The advantage of this approach is that by getting a model out sooner, testing can be done much more rapidly and efficiently, with each new iteration of the spiral. This also allows our clients access to previous prototypes and continual access to the system as we identify bugs and test new models. Due to the high uncertainty related to our clients, this model allows us to accommodate any desired changes for testing purposes. This also allows us to utilize requirements-based testing discussed later in this document.



### 2.1 Software (SCIís) to be tested

*Individual engine components will be tested separately. Interface components to be unit tested include:*

Wireframing

New Project Wizard

Database Construction

Database Communication with Interface

Exporting PDF File Ability

*In addition, the following are engine components to be tested are:*

Object Handler

Text Handler

Logic Handler

Attribute Handler

Unit Pathing

File I/O Parser

*Finally, components to enable our clients informational help when needed that will be tested involve:*

Interface Help

Engine Help

Database Help

Tutorials

### 2.2 Testing strategy

*Black-Box Testing*

The use of functional Black-Box testing will focus on how the software is working from a user’s perspective. This will include a series of test cases that are designed from each use case to guarantee that the system is operating as intended and satisfying the business requirements . Since it will be from a user-perspective, this testing will be performed later in the development process.

*Performance & Smoke Testing*

Performance testing will be used to ensure the system can perform with multiple agents using it at a time and the data is readily available to access. Smoke testing will be used after the system is deployed to ensure the application is running properly in the environment and to determine if more testing is needed.

In conjunction with this strategy for testing, we will utilize the following-

*Flexibility for New Functionality*

Ensuring the system is capable of accommodating new functionality and features with ease, facilitating future enhancements.

*Auditing*

Auditing will be instituted via inspecting work submitted against a set of standard processes to determine if they were followed.

*Reviewing*

Meetings are set up to review all software processes.

*Code Inspection*

Static testing will be established to find bugs and avoid defects.

*Design Inspection*

A checklist will be employed to ensure design inspection can be done efficiently. Such items will include general requirements, logic, structures, interfaces, and requirement traceability.

*Simulation*

A simulation will be completed through the creation of a prototype.

*Functional Testing*

Black box testing will be conducted to test the system specifications and features.

*Standardization (IEEE)*

Standardization will be required for all documentation and work completion to institute consistent standards and practices.

*Walkthroughs*

Software walkthroughs will be arranged to guide all team members in the development process.

*Unit testing*

To secure complete code coverage, White Box Testing will be executed on each path, branch, and condition.

*Stress testing*

To ensure our product works under normal conditions, stress testing will be completed to test the robustness of our system.

*Process Approach*

By enlisting a process approach, we can establish activities as processes, allowing our team to prioritize improvements, and manage activities.

*Evidence-Based Decision Making*

Decisions will be made based on analysis, appropriate methods, and employed tools.

*Relationship Management*

By constant communication with our clients, our relationship management will consist of honest and open dialogue.

#### 2.2.1 Unit testing

Unit testing is a crucial step in the development of MoonEyes. This type of testing will be used in the development of the system to ensure that each component is working properly before moving on to integration testing, as well as testing boundaries. Some of these boundaries include validating the information being entered by an agent; security measures such as an automatic sign out after 60 minutes of inactivity; and ensuring that the search parameters are displaying the correct information. Black box testing will be conducted to test each function and component of the software. Entry and exit conditions will be identified to safeguard the software components from compatibility issues.

#### 

#### 2.2.2 Integration testing

After the successful development of unit testing, the focus will shift to integration testing between multiple units. The integration testing will be used to test the interaction between the database, cloud-based web application, and the user interface. The main focal point of the integration testing is to validate the information the agent is entering or changing is updating correctly in the database and the information can be successfully and efficiently queried.

As our software is implemented on a cloud-based web application, we do not have to test at any physical client location as a cloud-based system can be testing from any physical location with the same results. We will however, ensure that there are no software, cloud, or network components that could possibly conflict and cause issues.

Integration testing will include protecting the database from any loss should any deviation from the norm occur. This will be conducted by beginning with our login window, then moving through each part of the software application.

#### 2.2.3 Validation testing

The use of validation testing will validate that all of the requirements of the MoonEyes have been met and implemented in the system. As well as the system is functioning as intended.

Mock data files have been created for testing purposes. This was done to establish the validity of software for our clients. With this mock data, we can obtain results for testing data. These results will be inspected for congruence with System Requirements Specifications. Validation testing will be rendered on all components and subcomponents of the software.

#### 2.2.4 High-order testing

High-Order testing will be conducted on the completed system. Such testing includes beta testing of our product. Best testing will be conducted in an attempt to run through a somewhat fully functioning Metro Detective Agency mock session. Coupled with this, we will be blending recovery, security, and stress testing in our high-order testing.

*Recovery Testing*

As the Metro Detective Agency database must maintain information stored in it, the software’s ability to retrieve any data lost is crucial to our clients. Should the system shut down, we must ensure data was not lost.

*Stress Testing*

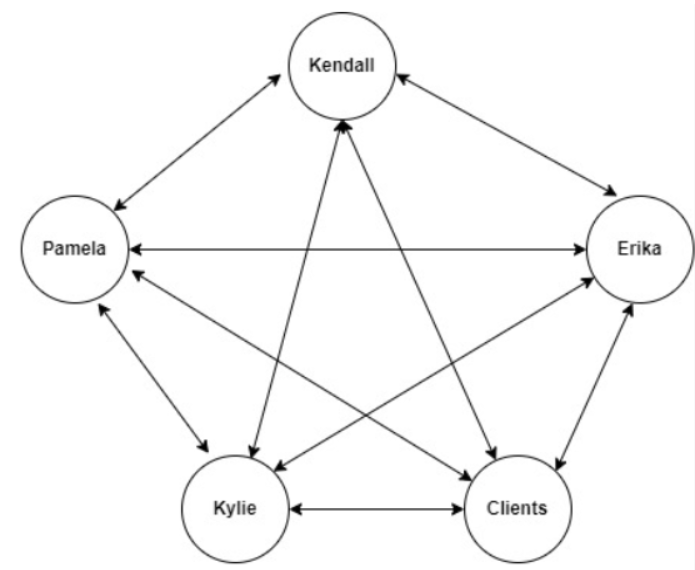
Employing the stress testing method allows us to monitor any stress our system may encounter. Should multiple administrators or users be concurrently logged into the system, we must ensure system abilities.

*Security Testing*

As our system employs secure client information, security testing must be conducted to ensure data security. This data is crucial to our client’s business, and therefore of the utmost importance in security testing concerns.

### 2.3 Testing resources and staffing

Due to our small team size, we will all take part in managing, designing, preparing, executing, and resolving relevant issues and test activities. To ensure issues are resolved, frequent meetings are set up to review and discuss outstanding and resolved bugs, decisions, goals, as well as any changes the client would like to have. Therefore, our testing employs an egoless and democratic structure as is illustrated below. The smaller group size allows for open communication among the team regarding the project decisions, products, and goals.



Constraints our team will encounter include both time and limited knowledge, as all members are currently students at the University of Michigan, Dearborn. Other constraints are described in detail below-

*Funding*

As mentioned, we will need to deploy MoonEyes on a cloud-based hosting service. This means it will cost anywhere from $10-50/month for our client.

*Time*

Because this project is for our senior design project we have a strict eight-month development timeline. It is possible that we might want to make improvements or add additional features down the road that are incompatible with the amount of time left. Additionally, all members of the team will also be balancing class and work responsibilities along with our project.

*Experience*

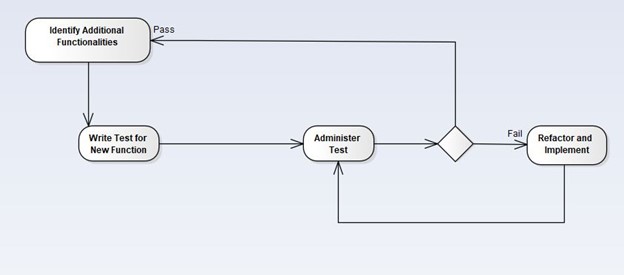
Because our group consists of students with limited professional development experience, we may have to do more independent and team research than a typical development team.

### 2.4 Test work products

Test procedures and sequencing of those test procedures are incorporated in test implementation work products. Our team has integrated Requirements-based testing due to our clients evolving needs and specifications.

*Requirements-based testing*

As new functionalities and requirements are needed for Metro Detective Agency, test driven development will be implemented. This allows our team to focus on the newly requested features before writing our code. This guarantees such needs are met. In addition to this, traceability can be supported throughout the software item lifecycle for test cycles and cases.



*Steps for development are comprised of*-

* Identifying additional functionalities.
* Writing a test for this new functionality.
* Administering the test.
  + As the test has not yet been implemented, it initially fails.
* Refactoring, Implementing, and Re-running the Test.

Upon successful test runs, the next functionality determined begins the test development stage.

*Work products produced*

The producer will make an informal change request.

During the review the recorder will take notes and reference the checklist. A review summary report will be made with details on the decisions made and if any changes need to be made before the modification is pushed.

*Test monitoring and control work products-*

Monitoring of tests and control work products will include test progress reports and test summary reports.

*Test Execution of work products*

Our team will log the status of individual test cases. These logs will include labels on such cases like “ready to run”, “pass”, “fail”, and “deliberately skipped”. Furthermore, defect reports will be maintained for proper documentation and regulation. Such documentation will include which test items, objects, tools, and ware were involved in any testing performed.

*Test Completion work products*

Test summary reports will be completed upon completion of tests for future reference. This helps our team track defects and issues, along with how they were resolved, in a manner that helps future testing.

### 2.5 Test record keeping

As aforementioned, our team will log the status of individual test cases. Such logs will be maintained and stored. Here, the log will also allow us to monitor any tests that have been conducted and applied. Furthermore, these reports audit any utilized tests. We will also store and evaluate test records by the following devices.

*Audit Logs*

Audit logs keep all the records of user activities on the system. Admin can view and monitor all the activities of users and can take necessary actions.

*Session Management*

This is to control the system access to only authorized users. When the user login to the system it starts a session, and all the activities are held within the session by the specific user and maintain the user’s rights. The session ends when a user logouts.

*Data Validation and Error Handling*

To maintain the quality performance of the system, error handling and error prevention are implemented. It ensures that the users enter the correct information, for example creating a case where the entered data is correct and in the right data fields.

*Access Control List*

The system uses the ACL system to control the permission that which user can perform which tasks. For example, only the admin can do user management. Unauthorized users cannot perform any task. Both user and admin can update the account information users cannot update the case information.

### 2.6 Test metrics

*Metrics used during testing include*:

*Schedule Adherence*

The purpose of this metric is to determine the difference between scheduled expected and actual execution times. This metric enables our team to estimate the progress, quality, and health of our software. In addition to this, it enables us to adhere to project schedules as this project must adhere to a tight finishing deadline.

*Defects finding rate*

This metric aids in measuring the quality of our software product and testing capabilities. It also aids in recognizing patterns of flaws our product is facing over a specified time period.

*Defect fixing time*

In order to ensure timeliness of defect fixes, a defect fixing time metric will be employed. This will measure the amount of time required to correct the issue.

*Test coverage*

Aiding in verification of coding sequences as well as testing functionality, we will use a test coverage metric. The number of test cases assigned to the program will be indicated. This metric will make certain that testing is completed thoroughly.

*Fault Profiles*

As our clients have a prioritization of needs, we will employ the fault profile metric to both categorize and prioritize any discovered errors.

### 2.7 Testing tools and environment

*Hardware Requirements:*

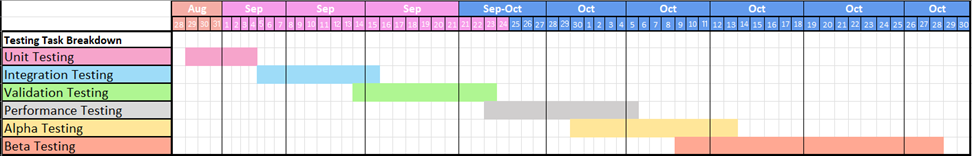
* PC or Laptop with
* Minimum 8GB RAM
* Access to Web Browser
* Internet Connection

*Software Requirements:*

* Visual Studio
  + Or Visual Studio Code
* Access to MySQL

* The test environment will be provided via the testers.
* The tester must have the ability to connect to the database server.
* In addition, an outstanding account with Github.com needs to be created before access to the code can occur.
* The tester can obtain the most recent build through Github.com
* New builds will not be pushed to the page until consistent testing has been conducted.

### 2.8 Test schedule

Below, is an outline of the most recent test schedule for project Mooneyes. 

## 

## **3.0 Test Procedure**

This section describes a detailed test procedure including test tactics and test cases for the software.

### 3.1 Software (SCIís) to be tested

For a detailed description of all individual software component items of our Process-Based Estimation Assistant please refer to 2.1 Software (SCIís) to be tested

### 3.2 Testing procedure

The overall procedure for software testing is described.

#### 3.2.1 Unit test cases

The procedure for unit testing is described for each software component (that will be unit tested) is presented. This section is repeated for all components i.

3.2.1 Unit test cases: **Sign In Component**

3.2.1.2 Stubs and/or drivers for Sign In component

* Stubs: Mock authentication service to simulate user authentication.
* Drivers: Simulated user interactions triggering the Sign In Button functionality.

3.2.1.3 Test cases component

**Test Case 1: Successful Sign In**

* **Description:** Validate the functionality of the "Sign In" button for successful authentication.
* **Steps:**

1. Enter a valid username and password.
2. Click on the "Sign In" button.

* **Expected Outcome:** The system authenticates the user and grants access to the dashboard. The user should be redirected to the dashboard page.

**Test Case 2: Incorrect Password**

* **Description:** Test the behavior of the "Sign In" button when an incorrect password is entered.
* **Steps:**

1. Enter a valid username and an incorrect password.
2. Click on the "Sign In" button.

* **Expected Outcome:** The system should display an error message indicating that the password is incorrect. The user should remain on the sign-in page.

**Test Case 3: Non-Existent Username**

* **Description:** Validate the behavior of the "Sign In" button when a non-existent username is entered.
* **Steps:**

1. Enter a non-existent username and a valid password.
2. Click on the "Sign In" button.

* **Expected Outcome:** The system should display an error message indicating that the username does not exist. The user should remain on the sign-in page.

**Test Case 4: Empty Fields**

* **Description:** Test the "Sign In" button behavior when both username and password fields are left empty.
* **Steps:**

1. Leave both username and password fields empty.
2. Click on the "Sign In" button.

* **Expected Outcome:** The system should display an error message indicating that both fields are required. The user should remain on the sign-in page.

3.2.1.4 Purpose of tests for Sign In component

* Ensure that the "Sign In" button accurately handles different scenarios, including successful sign-ins, incorrect passwords, non-existent usernames, and empty fields. Validate that the system provides appropriate feedback to the user based on their input.

3.2.1.5 Expected results for Sign In component

For each test case, the system should behave as follows:

* Test Case 1: The user is successfully authenticated, and they are redirected to the dashboard.
* Test Case 2: The system displays an error message about the incorrect password, and the user remains on the sign-in page.
* Test Case 3: The system displays an error message about the non-existent username, and the user remains on the sign-in page.
* Test Case 4: The system displays an error message about the empty fields, and the user remains on the sign-in page.

3.2.1 Unit test cases: **Home Component**

3.2.1.2 Stubs and/or drivers for component i

* Stubs: Mock API responses for simulating backend interactions.
* Drivers: Simulated user interactions triggering the Home Button functionality and data display.

3.2.1.3 Test cases component

**Test Case 1: View Open Cases**

* **Description:** Validate the functionality of the "Home" button for displaying open cases.
* **Steps:**

1. Click on the "Home" button.

* **Expected Outcome:** The system displays a table of open cases, including Case ID, Title, Purpose, Associated Agents, and Status (Open). The information is accurate and up-to-date.

**Test Case 2: View New Entries**

* **Description:** Validate the functionality of displaying new case entries.
* **Steps:**

1. Click on the "Home" button.
2. Navigate to the "New Entries" section.

* **Expected Outcome:** The system displays a table with new case entries, including Case ID, Purpose, Title, Associated Agents, and Status. The information is accurate and up-to-date.

**Test Case 3: View Recently Added Clients**

* **Description:** Validate the functionality of displaying recently added clients.
* **Steps:**

1. Click on the "Home" button.
2. Navigate to the "Recently Added Clients" section.

* **Expected Outcome:** The system displays a table with recently added clients, including Client ID, Name, Email, Associated Cases, and Lawyer details. The information is accurate and up-to-date.

**Test Case 4: View Recently Added Subjects**

* **Description:** Validate the functionality of displaying recently added subjects.
* **Steps:**

1. Click on the "Home" button.
2. Navigate to the "Recently Added Subjects" section.

* **Expected Outcome:** The system displays a table with recently added subjects, including Subject ID, Name, Associated Cases, and Lawyer details. The information is accurate and up-to-date.

3.2.1.4 Purpose of tests forHome component

* Ensure that clicking the "Home" button and accessing associated tables display accurate and up-to-date information. Validate the correct operation of viewing open cases, new entries, recently added clients, and recently added subjects.

3.2.1.5 Expected results for Home component

For each test case, the system should behave as follows:

* Test Case 1: The system displays a table of open cases with accurate information.
* Test Case 2: The system displays a table of new case entries with accurate information.
* Test Case 3: The system displays a table of recently added clients with accurate information.
* Test Case 4: The system displays a table of recently added subjects with accurate information.

3.2.1 Unit test cases: **Add Button**    
3.2.1.2 Stubs and/or drivers for component

* Stubs: Mock API responses for simulating backend interactions
* Drivers: Simulated user interactions triggering Add Button functionality.

3.2.1.3 Test cases component   
**Test Case 1: Add a New Case**

* **Description:** Verify the functionality of the "Add a case" button in the MoonEyes dashboard.
* **Steps:**

1. Click on the "Add" button.
2. Select "Add a case" from the dropdown menu.
3. Fill in the necessary case details ( Case ID, Title, Purpose of Case…)
4. Click on the "Submit" button.

* **Expected Outcome:** The case information is successfully submitted, stored in the database, and displayed in the system.

**Test Case 2: Add a New Client**

* **Description:** Test the functionality of the "Add a client" button in the MoonEyes dashboard.
* **Steps:**

1. Click on the "Add" button.
2. Select "Add a client" from the dropdown menu.
3. Enter client details (Client ID, Name, E-mail, Phone number…)
4. Click on the "Save" button.

* **Expected Outcome:** The client information is saved in the system, and the client is listed in the database.

**Test Case 3: Add a New Subject**

* **Description:** Validate the functionality of the "Add a subject" button.
* **Steps:**

1. Click on the "Add" button.
2. Select "Add a subject" from the dropdown menu.
3. Enter subject details (Subject ID, Name, Address, Vehicle Information…)
4. Click on the "Save" button.

* **Expected Outcome:** The subject details are successfully added, stored, and accessible in the system.

##### **Test Case 4: Add a New Tag**

##### **Description:** Test the functionality of the "Add a Tag" button.

##### **Steps:**

##### Click on the "Add" button.

##### Select "Add a Tag" from the dropdown menu.

##### Enter tag details.

##### Click on the "Create" button.

##### **Expected Outcome:** The tag information is created, stored, and available for associating with cases or subjects.

##### 3.2.1.4 Purpose of tests for Add Button Component:

* Validate the correct operation of the "Add" button and its sub-options. Ensure that adding new cases, clients, subjects, and tags works seamlessly without errors or data loss.

##### 3.2.1.5 Expected results for component for Add Button Component:

* For each test case, the corresponding data (case, client, subject, or tag) is successfully added and stored in the system's database. The system should provide appropriate success messages or notifications after each operation, indicating the successful addition of the respective data.

3.2.1 Unit test cases: **Search Button Component**

3.2.1.2 Stubs and/or drivers for component Search button Component

* Stubs: Mock API responses for simulating backend interactions.
* Drivers: Simulated user interactions triggering the Search Button functionality.

3.2.1.3 Test cases for Search button component

**Test Case 1: Search for New Cases**

* **Description:** Validate the functionality of the "Search" button for finding new cases.
* **Steps:**

1. Enter relevant search criteria (e.g., case number, client name, subject name).
2. Click on the "Search" button.

* **Expected Outcome:** The system displays a list of new cases matching the search criteria, including case numbers, client names, and subject details.

**Test Case 2: Search for Old Cases**

* **Description:** Test the functionality of the "Search" button for finding old cases.
* **Steps:**

1. Enter relevant search criteria (e.g., case number, client name, subject name) for a previously resolved case.
2. Click on the "Search" button.

* **Expected Outcome:** The system displays the details of the old case matching the search criteria, including case number, client name, subject details, and resolution status.

3.2.1.4 Purpose of tests for Search button component

* Validate the correct operation of the "Search" button. Ensure that users can search for both new and old cases effectively. Verify that the search results match the specified search criteria accurately.

3.2.1.5 Expected results for Search button component

For each test case, the system should display the correct search results based on the entered criteria. The displayed information should match the search query, including case details, client names, subject details, and resolution status for old cases.

3.2.1 Unit test cases: **Admin Controls Component**

3.2.1.2 Stubs and/or drivers for Admin Controls component

* Stubs: Mock API responses for simulating backend interactions.
* Drivers: Simulated user interactions triggering the Admin Controls functionality

3.2.1.3 Test cases component

**Test Case 1: View Audit Logs**

* Description: Validate the functionality of viewing audit logs in the Admin Controls.
* Steps:

1. Click on the "Admin Controls" button.
2. Select "View Audit Logs."

* Expected Outcome: The system displays a log of recent activities and interactions, including user actions, timestamps, and descriptions of the performed tasks.

**Test Case 2: View Recently Added Agents**

* **Description:** Validate the functionality to view recently added agents in a table.
* **Steps:**

1. Click on the "Admin Controls" button.
2. Select "View Recently Added Agents."

* **Expected Outcome:** The system displays a table with agent details, including Agent ID, name, email, and associated cases. The table is accurate and up-to-date.

**Test Case 4: Add a New Agent**

* **Description:** Test the functionality of adding a new agent through Admin Controls.
* **Steps:**

1. Click on the "Admin Controls" button.
2. Select "Add New Agent."
3. Fill in the agent's details and click "Save."

* **Expected Outcome:** The new agent is successfully added to the system, and the agent's details are displayed in the recently added agents table.

**Test Case 5: View Metrics - Cases Solved Last Month**

* **Description:** Validate the functionality of viewing metrics, specifically the number of cases solved in the last month.
* **Steps:**

1. Click on the "Admin Controls" button.
2. Select "View Metrics."

* **Expected Outcome:** The system displays the number of cases solved in the last month. The metric is accurate and reflects the correct data.

3.2.1.4 Purpose of tests for Admin Controls component

* Validate the correct operation of the Admin Controls, including viewing audit logs, sending emails, viewing recently added agents, adding new agents, and viewing metrics. Ensure that administrative tasks can be performed accurately and efficiently.

3.2.1.5 Expected results for Admin Controls component

* For each test case, the system should perform the specified administrative task accurately. The audit logs, emails, agent details, and metrics should be displayed correctly, and any actions performed through the Admin Controls should have the intended effect without errors or inconsistencies.

3.2.1 Unit test cases: **Logout Button**

3.2.1.2 Stubs and/or drivers for Logout component

* Stubs: Mock session management to simulate user logout.
* Drivers: Simulated user interactions triggering the Log Out Button functionality.

3.2.1.3 Test cases component

**Test Case: User Logout**

* **Description:** Validate the functionality of the "LogOut" button.
* **Steps:**

1. Click on the "LogOut" button.

* **Expected Outcome:** The user session is terminated, and the user is redirected to the login page. Accessing any secured page should prompt the user to log in again.

3.2.1.4 Purpose of tests for Log Out component

* Ensure that clicking the "LogOut" button results in the immediate termination of the user session, providing enhanced security and preventing unauthorized access to the system.

3.2.1.5 Expected results for Log Out component

* After clicking the "LogOut" button, the user should be logged out, and any attempts to access secure pages should redirect the user to the login page. The user's session should be fully terminated, preventing access to any authenticated features until the user logs in again.

#### 3.2.2 Integration testing

The integration testing procedure is specified.

##### 3.2.2.1 Testing procedure for integration

##### The main testing procedure for integration in MoonEyes involves testing individual components together. Each component is tested in conjunction with others to identify any issues arising during their integration. This includes testing components that are tied into other components within the case management system.

##### 3.2.2.2 Stubs and drivers required

##### No stubs or drivers are required for the integration testing of MoonEyes components.

##### 3.2.2.3 Test cases and their purpose

##### **Test Case 1: Add Case and Update Client Information**

##### **Description:** Validate the integration between the "Add Case" functionality and "Update Client" feature.

##### **Steps:**

##### Add a new case with client details.

##### Update the client information related to the added case.

##### **Purpose:** Ensure that adding a new case and updating client details work cohesively without data discrepancies.

##### **Test Case 2: Search for Case and Modify Subject Information**

##### **Description:** Validate the integration between the "Search" functionality and "Modify Subject" feature.

##### **Steps:**

##### Search for a specific case.

##### Modify subject information related to the searched case.

##### **Purpose:** Verify that searching for cases and modifying subject details function together accurately, updating the system in real-time.

##### **Test Case 3: Add Tag to Case and View Audit Logs**

##### **Description:** Validate the integration between the "Add Tag" function and "View Audit Logs" feature.

##### **Steps:**

##### Add a tag to a selected case.

##### View audit logs to confirm if the tag addition is logged.

##### **Purpose:** Ensure that adding tags to cases is appropriately logged in the audit trail, demonstrating the integration between tagging and audit logging.

##### 

##### 3.2.2.4 Expected results

* Test Case 1: The new case is added, and client details are successfully updated, ensuring data consistency between cases and clients.
* Test Case 2: The searched case is accurately modified, and subject details are updated accordingly, reflecting real-time changes in the system.
* Test Case 3: The audit logs display the addition of the tag to the case, confirming that tag-related actions are logged correctly for auditing purposes.

#### 3.2.3 Validation testing

The validation testing procedure is specified.

##### 3.2.3.1 Testing procedure for validation

* The validation testing procedure involves systematically testing all components of the MoonEyes project to ensure they are functioning correctly and as intended.

##### 3.2.3.2 Testing cases and their purpose

**Test Case 1: Submit Case and Verify Data Integrity**

* **Description:** Validate that submitted case data remains intact and accurate.
* **Purpose:** Ensure that all details entered during case submission are retained without data loss or corruption.

**Test Case 2: Update Client Information and Confirm Persistence**

* **Description:** Update client information and verify that changes are saved.
* **Purpose:** Confirm that client data modifications persist, demonstrating the system's ability to store and retrieve updated client information.

**Test Case 3: Add Tag to Case and Check Audit Trail**

* **Description:** Add a tag to a case and review the audit trail.
* **Purpose:** Verify that tag-related actions are logged in the audit trail, ensuring a record of all tagging activities.

**Test Case 4: Search for Case and Validate Search Results**

* **Description:** Search for a specific case and validate the accuracy of search results.
* **Purpose:** Ensure that the search functionality retrieves the correct case information based on the provided search criteria.

##### 3.2.3.3 Expected results

##### Test Case 1: The submitted case details are retained without any loss or corruption.

##### Test Case 2: Updated client information persists and is accurately retrieved when accessed.

##### Test Case 3: Adding a tag results in the corresponding entry in the audit trail, confirming proper logging.

##### Test Case 4: The search function accurately retrieves the specified case based on the provided search parameters.

##### 

##### 3.2.3.4 Pass/fail criterion for all validation tests

#### Pass Criteria:

#### All entered data is retained and displayed accurately.

#### Tags are successfully added and recorded in the audit trail.

#### The search function retrieves correct results based on the search query.

#### Fail Criteria:

#### Data is lost, corrupted, or inaccurately displayed during submission or retrieval.

#### Tag-related actions are not logged in the audit trail, indicating a logging failure.

#### Search results do not match the specified criteria, indicating a search functionality issue.

#### 

#### 

#### 3.2.4 High-order testing (a.k.a. System Testing)

The high-order testing procedure is specified. For each of the high order tests specified below, the test procedure, test cases, purpose, specialized requirements and pass/fail criteria are specified. It should be noted that not all high-order test methods noted in Sections 3.2.4.n will be conducted for every project.

##### 3.2.4.1 Recovery testing

##### Test Procedure: Simulate system failure and recovery to assess the system's ability to resume normal operation after a crash.

##### Test Cases: Trigger system failure intentionally and observe the recovery process.

##### Purpose: Evaluate the system's resilience and ability to recover gracefully from unexpected failures.

##### Specialized Requirements: None specified.

##### Pass/Fail Criteria: Pass if the system successfully recovers and restores functionality after a simulated failure; fail if the system cannot recover or experiences data loss.

##### 

##### 3.2.4.2 Security testing

##### For security testing, MoonEyes will undergo rigorous evaluations to ensure the confidentiality, integrity, and availability of the data stored in the system. Several aspects will be considered:

##### **SQL Injections and Cross-Site Scripting (XSS) Testing:** SQL Injections and XSS attacks are common security vulnerabilities. We will thoroughly test the system to ensure that it is protected against these attacks. SQL Injection involves injecting malicious SQL code into input fields, potentially granting unauthorized access to the database. XSS attacks involve injecting malicious scripts into web pages, which can compromise user data. By validating inputs and implementing proper sanitization techniques, MoonEyes will defend against these threats.

##### **Authentication Testing:** MoonEyes will undergo authentication testing to ensure that only authorized users can access specific resources. Different user roles will be defined (such as regular agents and admin users), and the system will be tested to confirm that each user can only access functionalities and data pertinent to their role. For instance, admin users will have exclusive access to the admin panel, ensuring that sensitive functionalities are only accessible to authorized personnel.

##### By addressing these security concerns, MoonEyes aims to provide a robust and secure environment for data management and user interactions. Regular security audits and updates will be conducted to maintain the system's integrity and protect against emerging threats.

##### 

##### 3.2.4.3 Stress testing

##### Test Procedure: Input a large volume of data into the system, including extensive case details, client information, and subject records.

##### Test Cases: Populate the system with a high volume of data, approaching or exceeding the system's documented capacity.

##### Purpose: Evaluate system stability and responsiveness under stress conditions with a substantial amount of data.

##### Specialized Requirements: None specified.

##### Pass/Fail Criteria: Pass if the system handles the stress conditions without crashing, slowing down unreasonably, or freezing; fail if the system fails to respond adequately under stress, indicating a performance issue.

##### 

##### 3.2.4.4 Performance testing

##### Test Case: Measure the time taken to load and display the main dashboard upon login.

##### Purpose: Ensure the system loads and displays the main interface within a reasonable timeframe (ideally within 9 seconds).

##### 

##### 

##### 3.2.4.5 Alpha/beta testing

##### Test Procedure: Release the system to a limited audience of internal users (Alpha) and a broader group of external users (Beta) for real-world usage and feedback.

##### Test Cases: Gather user feedback, identify bugs, and assess user satisfaction and system performance.

##### Purpose: Collect valuable insights from end-users, uncover potential issues, and ensure the system's readiness for widespread deployment.

##### Specialized Requirements: None specified.

##### Pass/Fail Criteria: Pass if feedback from Alpha and Beta testing indicates a positive user experience, minimal critical issues, and high user satisfaction; fail if significant issues are reported, requiring further adjustments and refinements before the full release.

##### 

##### 3.2.4.6 Pass/fail criterion for all validation tests

* Security Testing: Pass if no alerts pop up; fail if any <script> tags trigger an alert.
* Stress Testing: Pass if the system handles the stress conditions without crashing; fail if the system freezes or crashes under stress.
* Performance Testing: Pass if the dashboard loads within 9 seconds; fail if it takes longer.

### 3.3 Testing resources and staffing

* **Group Members:** All team members will actively participate in testing MoonEyes. They will utilize their personal PCs with Visual Studio Code and GitHub for testing purposes.
* **Open-Source Tools**: Utilization of open-source tools such as Visual Studio Code and GitHub for development and testing.
* **Communication Platform:** WhatsApp group chat will be used for real-time bug reporting and communication among team members.
* **Documentation and Logging:** Google Sheets will serve as a platform for documenting test cases, recording test results, and maintaining a test log.

### 3.4 Test work products

* Test Specification Document: This document, detailing the testing strategies, procedures, and scenarios for MoonEyes.
* Test Log: A Google Sheets document where all testing activities, including test cases and results, will be recorded.
* Final specification and test cases and results

### 3.5 Test record keeping and test log

* Google Sheets will be used as a central repository to log all testing activities, including test cases, test execution results, and any identified issues.
* Testers are responsible for promptly reporting test failures and errors via the WhatsApp group chat. Detailed information regarding the issue, steps to reproduce, and context will be communicated to facilitate quick resolution.