Verification and Validation Report: SFWRENG 4G06A

Team #25, RapidCare
Pranav Kalsi
Gurleen Rahi
Inreet Kaur
Moamen Ahmed

March 10, 2025

1 Revision History

Date	Version	Notes
2025-03-10	1.0	Initial Report

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
SRS	Software Requirements Specification
MG	Module Guide
MIS	Module Interface Specification
FR	Functional Requirements
NFR	Non-Functional Requirements
UI	User Interface
MFA	Multi-Factor Authentication
MG	Module Guide
M	Module
MIS	Module Interface Specification
API	Application Programming Interface
MFA	Multi-Factor Authentication
IR	Integrity Requirement
AR	Access Requirement

Contents

List of Tables

List of Figures

This document summarizes the results of series of tests as outlined in the VnV Plan, demonstrating the system's compliance with the specified requirements and highlighting areas of success as well as any identified issues.

3 Functional Requirements Evaluation

3.1 Add a document to the database

This subsection covers FR1, FR4, and FR8 from of the SRS document by testing that the system is able to add a document to the database only when a valid input is provided.

1. test-FR1,4,8-1

   Initial State: The system is set up, open to the relevant section (e.g., Healthcare Network, Healthcare Professional, or Patient Records), and ready to take the user input.

   Input: Correct and complete input data for all required fields.

   Expected Output: A confirmation message and a new entry is added to the appropriate database.

   Actual Output: A confirmation message and a new entry is added to the appropriate database.

   Result: Pass

2. test-FR1,4,8-2

   Initial State: The system is set up, open to the relevant section (e.g., Healthcare Network, Healthcare Professional, or Patient Records), and ready to take the user input.

   Input: Invalid input data.

   Expected Output: An error message outlining the invalid fields along with a possible steps to guide the users to recover from the error state.

   Actual Output: An error message outlining the invalid fields along with a possible steps to guide the users to recover from the error state.

   Result: Pass

3.2 Remove a document to the database

This subsection covers FR2, FR5, and FR9 from of the SRS document by testing that the system is able to remove a document to the database only when a valid input idetifier is provided.

1. test-FR2,5,9-1

   Initial State: The system is set up, open to the relevant section (e.g., Healthcare Network, Healthcare Professional, or Patient Records), and ready to take the user input. A document exists in the appropriate database.

   Input: A correct identifier for the document to be deleted.

   Expected Output: A confirmation message and relevant entry no longer exist in the database.

   Actual Output: A confirmation message and relevant entry no longer exist in the database.

   Result: Pass

2. test-FR2,5,9-2

   Initial State: The system is set up, open to the relevant section (e.g., Healthcare Network, Healthcare Professional, or Patient Records), and ready to take the user input. A document exists in the appropriate database.

   Input: An invalid identifier for the document to be deleted.

   Expected Output: An error message outlining the invalid input along with a possible steps to guide the users to recover from the error state.

   Actual Output: An error message outlining the invalid input along with a possible steps to guide the users to recover from the error state.

   Result: Pass

3.3 Update a document to the database

This subsection covers FR3, FR6, FR10, and FR11 from of the SRS document by testing that the system is able to update a document to the database only when a valid input idetifier is provided.

1. test-FR3,6,10,11-1

   Initial State: The system is set up, open to the relevant section (e.g., Healthcare Network, Healthcare Professional, or Patient Records), and ready to take the user input.

   Input: A correct identifier for the document to be updated.

   Expected Output: A confirmation message and relevant entry shows the updated data in the database.

   Actual Output: A confirmation message and relevant entry shows the updated data in the database.

   Result: Pass

2. test-FR3,6,10,11-2

   Initial State: The system is set up, open to the relevant section (e.g., Healthcare Network, Healthcare Professional, or Patient Records), and ready to take the user input.

   Input: Invalid input data.

   Expected Output: An error message outlining the invalid fields along with a possible steps to guide the users to recover from the error state.

   Actual Output: An error message outlining the invalid fields along with a possible steps to guide the users to recover from the error state.

   Result: Pass

3.4 Login for valid/invalid credentials

This subsection covers FR7 from of the SRS document by testing that the system is able to allow to access the database only when a valid input credentials is provided.

1. test-FR7-1

   Initial State: The system is set up, and the user is on the login page, ready to enter their credentials.

   Input: The correct credentials for login.

   Expected Output: A confirmation message and user is logged into the system.

   Actual Output: A confirmation message and user is logged into the system.

   Result: Pass

2. test-FR7-2

   Initial State: The system is set up, and the user is on the login page, ready to enter their credentials.

   Input: Invalid credentials for login.

   Expected Output: An error message outlining the invalid fields.

   Actual Output: An error message outlining the invalid fields.

   Result: Pass

3.5 Voice-to-text-transcription check

This subsection covers FR7 from of the SRS document by testing that the system is able to transcribe audio data to the written text.

1. test-FR11-1

   Initial State: The user has successfully logged in and is on the appropriate view to take audio input.

   Input: A valid audio chunk from the conversation between the healthcare professional and the patient.

   Expected Output: A confirmation message indicating successful transcription and the screen shows the transcribed text.

   Actual Output: A confirmation message indicating successful transcription and the screen shows the transcribed text.

   Result: Pass

2. test-FR11-2

   Initial State: The user has successfully logged in and is on the appropriate view to take audio input.

   Input: An invalid data format for transcription.

   Expected Output: An error message saying that the file or data format is not valid.

   Actual Output: An error message saying that the file or data format is not valid.

   Result: Pass

3.6 Validate output of correct diagnosis and medication

This subsection covers FR12 and FR13 from of the SRS document by testing that the system is able to create predictions on the diagnosis and medicines with a high accuracy and confidence.

1. test-FR12,13-1

   Initial State: Patient's blank chart is present, and a transcription of the Patient-Healthcare Professional conversation is present.

   Input: Correct and complete input data where all the symptoms fields of the chart are filled out.

   Expected Output: A suggestion on what the diagnosis should be based on the symptoms, then based on the diagnosis provide possible medicine and a plan of treatment.

   Actual Output: A suggestion on what the diagnosis should be based on the symptoms, then based on the diagnosis provide possible medicine and a plan of treatment.

   Result: Pass

3.7 Validate input data for models

This subsection covers FR12, FR13, and IR5 from of the SRS document by testing that the system is able to validate the input data in the charts such that it may be inputted into the prediction module.

1. test-FR12,13,IR5-1

   Initial State: The patient's blank chart is present.

   Input: Correct and complete input data where the transcription is present.

   Expected Output: A suggestion on what the diagnosis should be based on the symptoms, then based on the diagnosis provide possible medicine.

   Actual Output: A suggestion on what the diagnosis should be based on the symptoms, then based on the diagnosis provide possible medicine.

   Result: Pass

2. test-FR12,13,IR5-2

   Initial State: The patient's blank chart is present.

   Input: Incorrect and incomplete transcription field as it was not recorded/complete.

   Expected Output: An error message relaying the error along with a HTTP status code indicating incomplete request.

   Actual Output: An error message relaying the error along with a HTTP status code indicating incomplete request.

   Result: Pass

4 Nonfunctional Requirements Evaluation

4.1 Aesthetic and Design

1. test-AD1

   Initial State: UI is designed and implemented.

   Input: Team members and peers view the UI under normal operating conditions.

   Output: Feedback collected on UI's aesthetic appeal and simplicity.

   Result: Pass

2. test-AD2

   Initial State: UI is operational and accessible to users.

   Input: Users interact with the UI during routine tasks and provide feedback.

   Output: Team members and peers report satisfaction with the UI design.

   Result: Pass

4.2 Usability

1. test-UR1

   Initial State: System is up.

   Input: Users navigate and explore system features independently.

   Output: Majority of users can locate core functions without additional guidance. System reflects clear discoverability, affordances, and visibility.

   Result: Pass

4.3 Performance

1. test-PR1

   Initial State: Transcription interface open and ready.

   Input: Real-time voice input provided by healthcare professionals.

   Output: Real-time transcription displayed within a 2-second delay.

   Result: Pass

4.4 Operational

1. test-OR1

   Initial State: System is running in a controlled test environment.

   Input: Multiple test users perform basic interactions (e.g., logging in, recording voice input, retrieving stored data).

   Output: The system responds within an acceptable time frame, and no critical errors occur.

   Result: Pass

4.5 Maintainability

1. test-MR1

   Initial State: System is running in the development environment.

   Input: Review and modify system components (e.g., refactoring code, adding new functionality).

   Output: Changes are successfully integrated without breaking existing functionality, verified through automated/unit tests.

   Result: Pass

2. test-MR2

   Initial State: Codebase is under active development.

   Input: Perform a code maintainability review i.e. checking adherence to coding standards, documentation completeness.

   Output: Code follows maintainability best practices, including clear documentation and modular structure.

   Result: Pass

4.6 Security

1. test-SR1

   Initial State: System fully operational with access logs enabled.

   Input: Simulate unauthorized access attempts.

   Output: Unauthorized attempts are blocked; access logs capture details.

   Result: Pass

4.7 Cultural

1. test-CR1

   Initial State: System operational with default settings.

   Input: User interacts with the system in a culturally diverse context, including input that may contain sensitive language.

   Output: System processes the input and responds appropriately, ensuring cultural sensitivity and avoiding any inappropriate language in its output.

   Result: Pass

4.8 Legal

1. test-LR1

   Initial State: System is fully functional and contains patient records.

   Input: Conduct a compliance audit with PIPEDA and other relevant data protection standards.

   Output: System passes all compliance checks with no exceptions.

   Result: Pass

2. test-LR2

   Initial State: System storing and transmitting patient data over a network.

Input: Monitor data handling and transfer processes during operation.

Output: All patient data is handled in compliance with regulations, without any unauthorized access or data breaches.

Result: Pass

4.9 Scalability

1. test-S1

   Initial State: System deployed on a test server environment capable of scaling horizontally.

   Input: Simulate an increasing number of concurrent users.

   Output: System maintains consistent performance and response times without degradation.

   Result: Pass

4.10 Tests for Safety and Security Requirements

4.10.1 Access Requirements Tests (AC1)

1. test-AC1-1

   Initial State: System deployed with authentication module enabled and test user accounts configured.

   Input: Attempt to access protected resources with invalid credentials.

   Output:

   - System logs each failed attempt.
   - User receives failed to login notification.

   Result: Pass

2. test-AC1-2

   Initial State: System operational with authentication logs enabled.

   Input: Attempt to access system resources without authentication.

   Output:

   - All unauthorized access attempts are blocked.
   - Each attempt is logged with timestamp, IP address, and attempted resource.

   Result: Pass

3. test-AC2-1

   Initial State: System operational with standard user and admin accounts configured.

   Input: Attempt to create, update, and delete user accounts using non-admin credentials.

   Output:

   - All unauthorized actions are blocked.
   - Actions are logged with user details.
   - Security team can review blocked attempts.

   Result: Pass

4.10.2 Integrity Requirements Tests (IR1)

1. test-IR1-1

   Initial State: System operational with test user accounts and authentication database.

   Input: Simulate multiple concurrent failed login attempts while monitoring credential storage.

   Output: User credentials remain unchanged, and system maintains stability.

   Result: Pass

2. test-IR2-1

   Initial State: System is set up, open to the relevant section, and ready to take user input.

   Input: Submit an invalid data input.

   Output: An error message is displayed to the user. No data is added to any database.

   Result: Pass

3. test-IR3-1

   Initial State: The prediction model is ready to predict medication and diagnosis.

   Input: A test set of various patient medical charts.

   Output: The prediction includes a diagnosis and medication prediction, each with a confidence score exceeding 85%.

   Result: Pass

4. test-IR4-1

   Initial State: System is set up, open to the relevant section, and ready to take user input. A document already exists in the relevant database.

Input: A new input with the same data as an existing document.

Output: An error message is displayed. No new document is added to the database.

Result: Pass

5. test-IR5-1

   Initial State: System is set up, open to the relevant section, and ready to take user input. A document already exists in the relevant database.

   Input: Written text transcribed from the audio data.

   Output: The data is correctly classified in the generated report with no diagnosis errors.

   Result: Pass

6. test-IR6-1

   Initial State: System is set up, open to the relevant section, and ready to take user input.

   Input: Audio conversation between the patient and healthcare professional.

   Output: The written text transcribed from the input data matches the conversation with minimal to no interference.

   Result: Pass

5 Comparison to Existing Implementation

N/A

6 Unit Testing

6.1 Behaviour-Hiding Module

6.1.1 User Authentication Module

Control: Manual
Initial State: The default email and password are initialized as variables and given a default value.
Test Case Derivation: The expected behavior is derived from the correct registration and login functionality of Firebase Authentication, ensuring valid user creation and authentication processes, as well as error handling for invalid credentials.
Test Procedure: The tests are performed as follows:

1. Sign up using valid credentials:

- **Input:** The input for this test are the valid credentials of email and password.
- **Output:** The user object is returned by the signup function.
- **Test Derivation:** Verifies that the signup function correctly works and creates a new user account when the valid credentials are provided.
- **Result:** Pass

2. Sign up throwing errors with invalid credentials:

   - **Input:** The input for this test are the invalid credentials of email and password.
   - **Output:** An error thrown up by the signup function.
   - **Test Derivation:** Verifies that the signup function correctly throws error when the invalid credentials are provided.
   - **Result:** Pass

3. Sign in using valid credentials:

   - **Input:** The input for this test are the valid credentials of email and password.
   - **Output:** The user object is returned by the signin function.
   - **Test Derivation:** Verifies that the signin function correctly authenticates the user when valid credentials are provided.
   - **Result:** Pass

4. Sign in throwing errors with invalid credentials:

   - **Input:** The input for this test are the invalid credentials of email and password.
   - **Output:** An error thrown up by the signin function.
   - **Test Derivation:** Verifies that the signin function correctly throws error when the invalid credentials are provided.
   - **Result:** Pass

6.1.2 Administrator Account Management Module and Patient Account Management Module

Control: Manual
Initial State: Provides mock data of healthcare professionals, healthcare networks and patients.
Test Case Derivation: The expected behavior is derived from the correct functionality of onboarding, updating, and removing the healthcare networks, healthcare professionals and patient records.
Test Procedure: The tests are performed as follows:

1. Adding valid documents to the database:

- **Input:** Mock data of the healthcare professionals, healthcare networks and patients are given as input.
- **Output:** Successful addition of valid information of healthcare networks, healthcare professionals and patients in the database.
- **Test Derivation:** Verifies that addHealthcareProfessional, addHospital and addPatient functions work correctly ensuring proper document creation. The functions addHealthcareProfessional and addHospital add the documents if they don't exist in the database. However, if they do, then these functions update the data in those documents. The function addPatient is only used to add a new patient record in the database.
- **Result:** Pass

2. No duplicate addition of patient document in the system:

   - **Input:** Mock data of the patients are given as input.
   - **Output:** No specific output is returned by the test function itself. However, the test asserts a specific state in the database.
   - **Test Derivation:** Verifies that the addPatient function correctly handles attempts to add duplicate patient records. It ensures that only one patient record is created even if the addPatient function is called multiple times.
   - **Result:** Pass

   Note: Test for invalid document format:
   Since, we have implemented validate input function in the UI module, this ensures that no invalid output is taken by the system. In addition to this, TypeScript has type checking on the parameters therefore invalid objects cannot be passed.

3. Deleting existing documents from the database:

   - **Input:** Mock data of the healthcare professionals, healthcare networks and patients are given as input.
   - **Output:** Successful deletion of valid information of healthcare networks, healthcare professionals and patients in the database.
   - **Test Derivation:** Verifies that deleteHealthCareProfessional, deleteHospital and deletePatient functions work correctly ensuring proper document deletion. item **Result:** Pass

4. Updating existing patient documents in the database:

   - **Input:** Mock data of the patients are given as input.
   - **Output:** Successful update of valid information of the patients in the database.

- **Test Derivation:** Verifies that updatePatient function works correctly ensuring proper patient record update.
- **Result:** Pass

5. Retrieve correct existing document:

   - **Input:** Mock id of the patients are given as input.
   - **Output:** Retrieval of the correct patient object from the database.
   - **Test Derivation:** Verifies that the getPatient function correctly retrieves an existing patient document from the database after providing patient id.
   - **Result:** Pass

6. Retrieve non-existing document:

   - **Input:** Non-existent mock id of the patients are given as input.
   - **Output:** Returns null.
   - **Test Derivation:** Verifies that the getPatient function correctly handles situations when a non-existing patient document from the database is requested to retrieve after providing a non-existing patient id.
   - **Result:** Pass

6.1.3 Administrator View Module and Patient View Module

Control: Manual
Initial State: No specific initialization is required as tests solely focus on validating fields.
Test Case Derivation: The expected behavior is derived from the correct implementation of data validation rules and age calculation logic, ensuring accurate identification of invalid input, acceptance of valid input, and precise calculation of age based on provided dates.
Test Procedure: The tests are performed as follows:

1. Should return appropriate error for invalid field input:

   - **Input:** The inputs for this test are the actual value of the field and a string representing the field type.
   - **Output:** Returns appropriate error messages on receiving invalid input.
   - **Test Derivation:** Verifies that the validateField function correctly identifies invalid input values for different field types. It ensures that the function returns appropriate error messages for invalid email formats, phone numbers, dates, and numerical values for age, weight, and height.
   - **Result:** Pass

2. Should return an empty string for valid field input:

    - **Input:** The inputs for this test are the actual value of the field and a string representing the field type.
    - **Output:** Returns an empty string on receiving invalid input.
    - **Test Derivation:** Verifies that the validateField function correctly identifies valid input values for different field types. It ensures that the function returns an empty string for valid email formats, phone numbers, dates, and numerical values for age, weight, and height.
    - **Result:** Pass

3. Correct calculation of age:

    - **Input:** The input for this test is the date of birth.
    - **Output:** Returns a number representing the calculated age.
    - **Test Derivation:** Verifies that the calculateAge function correctly calculates the age for a given date of birth.
    - **Result:** Pass

4. Calculation of age for future dates:

    - **Input:** The input for this test is the date of birth in the future.
    - **Output:** Returns 0.
    - **Test Derivation:** Verifies that the calculateAge function correctly handles the age for a given date of birth in the future.
    - **Result:** Pass

5. Calculation of age for the present date:

    - **Input:** The input for this test is the date of birth representing today's date.
    - **Output:** Returns 0.
    - **Test Derivation:** Verifies that the calculateAge function correctly handles the age for a given date of birth with today's date.
    - **Result:** Pass

6.2 Software Decision Module

6.2.1 Diagnosis Prediction Module and Medicine Prediction Module

Control: Automatic
Initial State: A predefined set of conversation is used for testing. It also assumes that

Open API Key is configured correctly.
Test Case Derivation: The expected behavior is derived from the correct functioning of an API endpoint that processes patient inputs and provides relevant responses, ensuring accurate handling of valid and invalid input data, as well as a high confidence level in generating appropriate responses. We will review the manual test by our supervisor. **Test Procedure:** The tests are performed as follows:

1. Confidence score test:
    - **Input:** The input for this test is the list of strings which represent patient conversation.
    - **Output:** There is no direct output. This test asserts a condition based on the number of successful evaluations.
    - **Test Derivation:** Verifies that the testConfidence function correctly sends each conversation to the API endpoint. The test calculates the success rate based on the number of relevant responses and asserts that the success rate is above a certain threshold (0.85). Due to the output of this test not being the deterministic, we have used a strategy that involves using a third independent AI agent to evaluate the confidence. This allows for an initial benchmark of the confidence on the service's performance.
    - **Result:** Pass
2. Valid data input test:
    - **Input:** A dictionary that contains a valid transcription of a patient conversation.
    - **Output:** Returns a response object from API endpoint.
    - **Test Derivation:** Verifies that the API endpoint correctly handles and processes valid input data. It sends POST request with valid transcription to the API endpoint and asserts that the response status code is 200, indicating successful processing of the input.
    - **Result:** Pass
3. Invalid data input test:
    - **Input:** A dictionary that contains an empty string of invalid transcription of a patient conversation.
    - **Output:** Returns a response object from API endpoint.
    - **Test Derivation:** Verifies that the API endpoint correctly handles and processes invalid input data. It sends POST request with invalid and missing transcription to the API endpoint and asserts that the response status code is 400, successful identification of invalid input.
    - **Result:** Pass

6.2.2 Transcription Module

Control: Manual
Initial State: The audio files are provided as input for the tests and Socket.IO Server is running at a specified host.
Test Case Derivation: The expected behavior is derived from the correct functioning of a Socket.IO server that receives audio chunks, performs transcription, and emits transcription results or errors. The tests ensure that the server correctly transcribes valid audio data, handles invalid audio input, and emits appropriate events with accurate transcription results or error indicators.
Test Procedure: The tests are performed as follows:

1. Test for valid audio file:
    - **Input:** The input for this test is the path to the valid audio file.
    - **Output:** Returns a string containing transcribed text from the audio file.
    - **Test Derivation:** Verifies the audio transcription functionality of the Socket.IO server. It establishes a connection with the server and sends the audio data in chunks and then emits the transcribed text.
    - **Result:** Pass
2. Test for invalid audio file:
    - **Input:** The input for this test is the path to the invalid audio file.
    - **Output:** Returns an empty string containing transcribed text from the audio file.
    - **Test Derivation:** Verifies that the API endpoint correctly handles invalid input data. It establishes the connection with the server and asserts that transcription_text remains empty indicating that the invalid input is correctly handled.
    - **Result:** Pass

6.2.3 Classification Module

Control: Automatic
Initial State: A list of pre-defined conversation containing patient's symptoms, allergies, medications is initialized.
Test Case Derivation: The expected behavior is derived from the correct functioning of an API endpoint that processes patient conversations, extracts key information (symptoms, reason for visit, allergies, current medications), and provides relevant responses. The tests ensure that the API accurately extracts information from various conversation formats, handles both valid and invalid input data, and maintains a high level of confidence in generating appropriate responses.

Test Procedure: The tests are performed as follows:

1. Confidence score test:
    - **Input:** The input for this test is the list of strings representing the patient conversation consisting of medications, allergies and current medications.
    - **Output:** There is no direct output. This test asserts a condition based on the number of successful evaluations.
    - **Test Derivation:** Verifies the confidence of the API in correctly extracting information from patient conversations. It iterates through the list sending conversation to the API endpoint. We will test it manually with our supervisor. Due to the output of this test not being the deterministic, we have used a strategy that involves using a third independent AI agent to evaluate the confidence. This allows for an initial benchmark of the confidence on the service's performance.
    - **Result:** Pass
2. Test for valid input:
    - **Input:** The input for this test is the valid patient transcription.
    - **Output:** Returns a response object from the API endpoint.
    - **Test Derivation:** Verifies that the API endpoint correctly handles and processes valid input data. It sends POST request with valid input to the API endpoint and asserts that the response status code is 200, indicating successful processing of the input.
    - **Result:** Pass
3. Test for invalid input:
    - **Input:** The input for this test is an empty dictionary.
    - **Output:** Returns a response object from the API endpoint.
    - **Test Derivation:** Verifies that the API endpoint correctly handles empty input data. It sends POST request with empty input to the API endpoint and asserts that the response status code is 400, indicating successful processing of the input.
    - **Result:** Pass

7 Changes Due to Testing

Based on the feedback received during the Rev 0 demo, we focused on ensuring that we prioritize critical and necessary requirements in the final implementation. We incorporated user feedback messages in accordance with the functional tests. Additionally, we updated

the functions responsible for validating input data to cover the edge cases discovered during testing, ensuring that the system can handle unexpected or invalid inputs gracefully. Furthermore, we placed a strong emphasis on UI and usability testing. This involved gathering feedback on the aesthetic appeal and functionality of the user interface, ensuring that users could navigate the system intuitively and efficiently. Iterative testing process allowed us to refine the system significantly and make several design adjustments leading to a more user-friendly application.

8 Automated Testing

We useD GitHub Actions to automate critical aspects of our development workflow, primarily focusing on testing and code quality. This automation ensures that every code change undergoes a rigorous validation process before being integrated into the main branch. The execution of suite of unit and integration tests forms the basis of our continuous integration approach. This makes sure that any new code modifications don't cause regressions or interfere with already-existing functionality. These tests are automatically executed on each pull request, allowing us to promptly detect and fix any possible problems early in the development cycle. Our team has been continuously running into linting problems on all pull requests lately. Even though our tests pass GitHub Actions CI / Explore-GitHub-Actions (push) test, it fails the Lint code base (push) test. The screenshot below show an example of lint-codebase error. We hope to resolve this issue before final demonstration by fixing all the lint errors in the code.

Figure 1: Lint codebase error

9 Trace to Requirements

See SRS Documentation at SRS document for detailed Information.

Test ID	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FR12	FR13
test-FR1,4,8-1	×			×				×					
test-FR1,4,8-2	×			×				×					
test-FR2,5,9-1		×			×				×				
test-FR2,5,9-2		×			×				×				
test-FR3,6,10,11-1			×			×				×	×		
test-FR3,6,10,11-2			×			×				×	×		
test-FR7-1							×						
test-FR7-2							×						
test-FR11-1											×		
test-FR11-2											×		
test-FR12,13-1												×	×
test-FR12,13,IR5-1												×	×
test-FR12,13,IR5-2												×	×

Table 1: **Functional Requirements Tests Requirement Traceability**

TestID	NFR1	NFR2	NFR3	NFR4	NFR5	NFR6	NFR7	NFR8	NFR9
test-AD1	×								
test-AD2	×								
test-UR1		×							
test-PR1			×						
test-OR1				×					
test-MR1					×				
test-MR2					×				
test-SR1						×			
test-CR1							×		
test-LR1								×	
test-LR2								×	
test-S1									×

Table 2: **Non-Functional Requirements Tests Requirement Traceability**

Test ID	AC1	AC2	IR1	IR2	IR3	IR4	IR5	IR6	IR7
test-AC1-1	×								
test-AC1-2	×								
test-AC2-1		×							
test-IR1-1			×						
test-IR2-1				×					
test-IR3-1					×				
test-IR4-1						×			
test-FR12,13,IR5							×		
test-IR6-1							×		
test-IR7-1								×	

Table 3: **Safety and Security Requirements Tests Requirement Traceability**

10 Trace to Modules

See MG Documentation at MG for detailed Information.

Test ID	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
test-FR1,4,8-1		×	×								×	×
test-FR1,4,8-2		×	×								×	×
test-FR2,5,9-1		×	×								×	×
test-FR2,5,9-2		×	×								×	×
test-FR3,6,10,11-1		×	×								×	×
test-FR3,6,10,11-2		×	×								×	×
test-FR7-1	×	×	×			×						
test-FR7-2	×	×	×			×						
test-FR11-1			×			×	×	×				
test-FR11-2			×			×	×	×				
test-FR12,13-1			×			×			×	×		
test-FR12,13,IR5-1			×			×			×	×		
test-FR12,13,IR5-2			×			×			×	×		

Table 4: **Functional Requirements Tests Module Traceability**

TestID	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
test-AD1		×	×									
test-AD2		×	×									
test-UR1	×	×	×	×	×	×	×	×	×	×	×	×
test-PR1			×			×	×	×				
test-OR1	×	×	×	×	×	×	×	×	×	×	×	×
test-MR1	×	×	×	×	×	×	×	×	×	×	×	×
test-MR2	×	×	×	×	×	×	×	×	×	×	×	×
test-SR1	×	×				×						
test-CR1		×	×									
test-LR1						×					×	×
test-LR2						×					×	×
test-S1	×	×	×	×	×	×	×	×	×	×	×	×

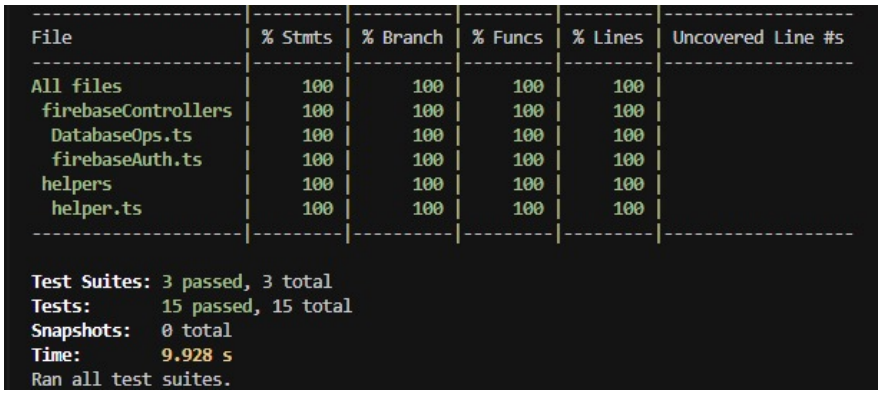
Table 5: **Non-Functional Requirements Tests Module Traceability**

Test ID	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
test-AC1-1	×											
test-AC1-2	×											
test-AC2-1	×											
test-IR1-1	×	×	×								×	×
test-IR2-1		×	×								×	×
test-IR3-1									×	×		
test-IR4-1		×	×								×	×
test-IR6-1							×	×				
test-IR7-1							×	×				

Table 6: **Safety and Security Requirements Tests Module Traceability**

11 Code Coverage Metrics

The image below displays the code coverage matrix for UI, Patient Account Management, Administrator Account Management, and Authentication Modules. As can been in the figure below, the code coverage for all the files we created the unit tests for is 100%.

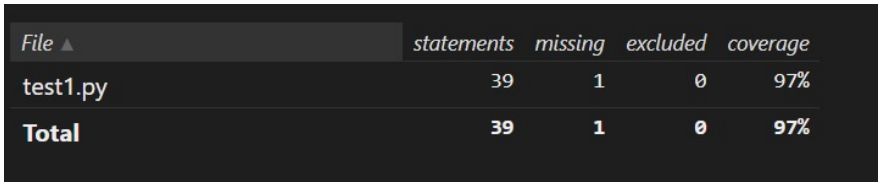


File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
firebaseControllers	100	100	100	100	
DatabaseOps.ts	100	100	100	100	
firebaseAuth.ts	100	100	100	100	
helpers	100	100	100	100	
helper.ts	100	100	100	100	

Test Suites: 3 passed, 3 total
Tests: 15 passed, 15 total
Snapshots: 0 total
Time: 9.928 s
Ran all test suites.

Figure 2: Code Coverage Metrics

We also created unit tests for software decision modules. As can be seen in the figures below, we could not obtain 100% coverage for all the tests because according to the report, the only thing missing is the run command for the tests.



File ▲	statements	missing	excluded	coverage
test1.py	39	1	0	97%
Total	**39**	**1**	**0**	**97%**

Figure 3: Code Coverage Metrics for Classification Module

The below is the code coverage metrics for diagnosis prediction and medicine prediction modules. As can be seen in the figure below, we could not obtain 100% coverage for all the tests because the catch that is not covered is alternatively covered by the 'else' statement which both yield the same http response code indicating a failure.

The below is the code coverage metrics for transcription module. As can be seen in the figure below, we could not obtain 100% coverage for all the tests because according to the report, the only thing missing is the run command for the tests.

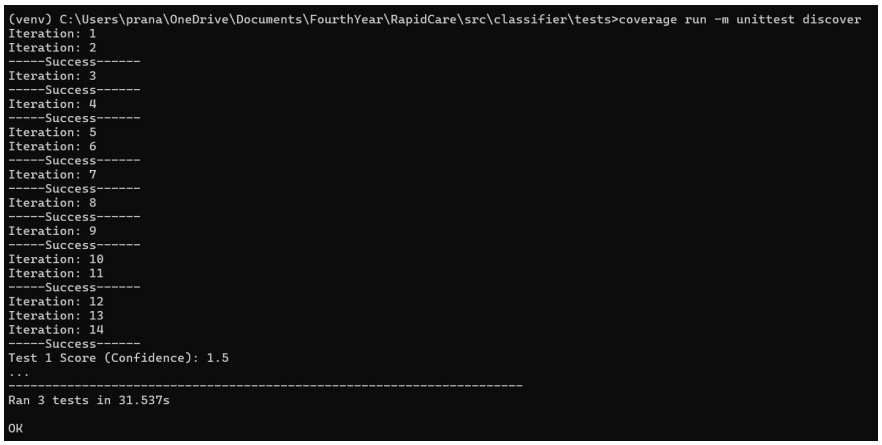


Figure 4: Tests for Classification Module

Appendix — Reflection

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

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?
   This document has let us strengthen the understanding about unit testing of all functional requirements as well as verification of all non-functional qualities of the system. While going through the outline of the document, we were able to conduct unit testing and compare it with the existing implementation. It also made us better execute the detailed procedure of system testing along with performing validation and verification of the system.

2. What pain points did you experience during this deliverable, and how did you resolve them?
   There are obstacles in any team project that must be overcome for it to proceed successfully. To ensure seamless operations, we had to develop a strategy for contributions.

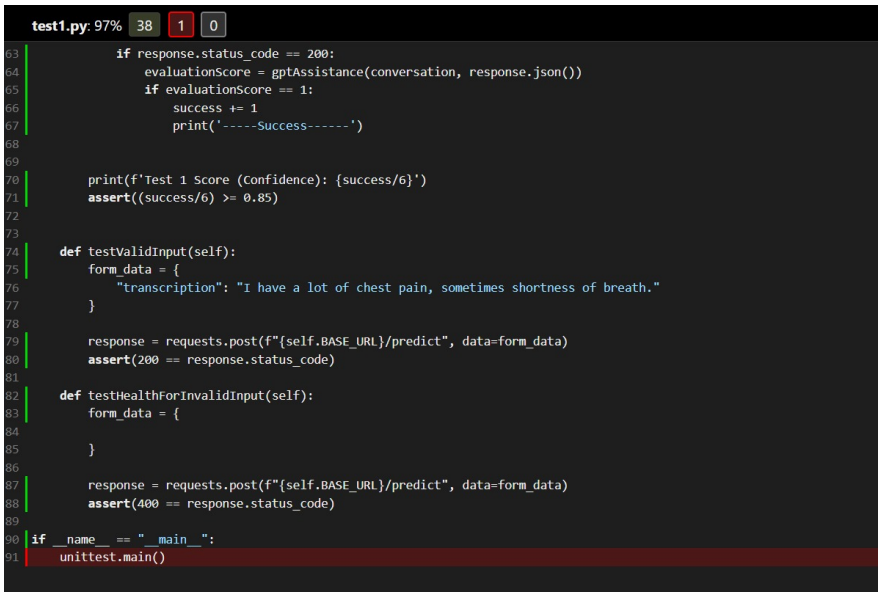


Figure 5: Tests for Classification Module

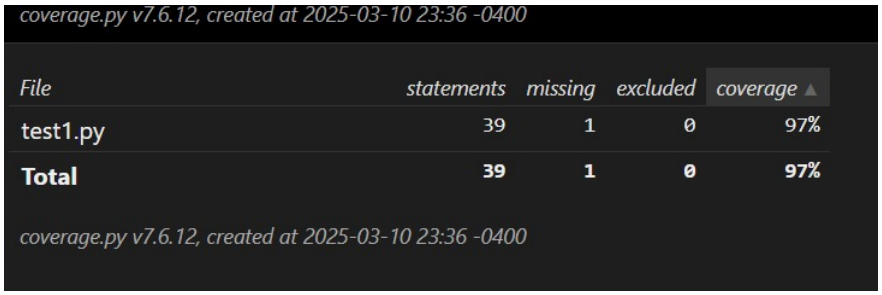


Figure 6: Code Coverage Metrics for Diagnosis Prediction and Medicine Prediction Module

Since we already had the list of tests that were needed to be conducted for both functional and non-functional requirements, we divided them equally among all the team members. We also needed to create a schedule to contribute to the template and review each other's work in the best way possible.

3. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why?
The client's feedback helped us to understand the changes due to testing the requirements. Before Rev 0 demo, the team gave a demo to the supervisor about the system pretending the supervisor as the healthcare professional using the system. The supervisor then gave us some possible ways that the healthcare professional would like to use the system. This helped us to improve some features of the system such as whether prescription of a patient is being saved or not. If yes, then is it being stored within the system or on local machine. Furthermore, the section such as unit testing of all

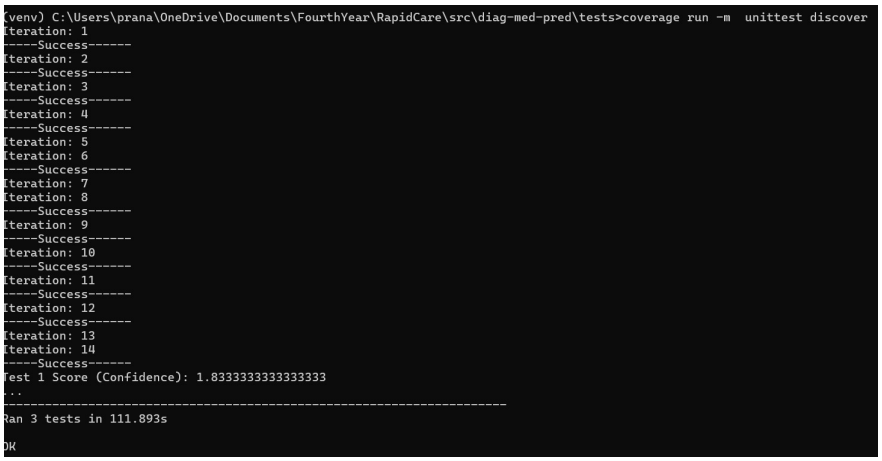


Figure 7: Tests for Diagnosis Prediction and Medicine Prediction Module

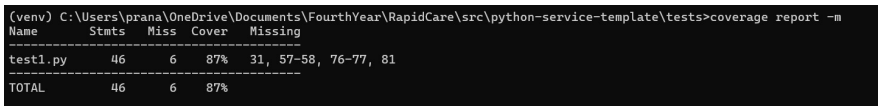


Figure 8: Code Coverage Metrics for Transcription Module

functional requirements stemmed from the collective ideas of our peers. We discussed different test cases that each requirement could possibly have along with how they will be implemented. In this way, we got diverse test cases and all team members were able to contribute especially in this critical section of the report.

4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)
   VnV Plan was little bit different from the activities that were actually conducted and recorded in VnVReport. This is because VnV Plan includes one extra functional requirement test which needs to be deleted as the team has decided to not pursue with that functional requirement. Therefore, the team will modify the VnV Plan to remove that requirement. Additionally, as part of the modifications, the team removed some Non-Functional Requirements tests to reduce redundancy, such as test-S2, test-CR2, and test-PR1, and updated the traceability to the requirements for both Non-Functional Requirements tests and Safety and Security tests. These changes were made to streamline the testing process and ensure alignment with the system's goals. This change occurred as we went through every detail of our system and it turns out that functional requirement is not necessary for the system to accomplish its goal. Yes, we

will be able to anticipate thus change in the future projects as going through every detail of the project review the necessary and non-necessary features of the product.