System Verification and Validation Plan for SFWRENG 4G06A

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

April 2, 2025

Revision History

Date	Version	Notes
04-11-2024	1.0	Initial Version
10-03-2025	1.1	Updates in Accordance to VnV report
31-03-2025	1.2	Updated NFR and Security requirement tests to reflect changes in SRS and Hazard Analysis
31-03-2025	1.2	Updates to reflect changes in VnV Report
01-04-2025	1.3	Added Unit testing and updated usability survey

Contents

List of Tables

1 Symbols, Abbreviations, and Acronyms

symbol	description
SRS	Software Requirements Specification
MG	Module Guide
MIS	Module Interface Specification
FR	Functional Requirements
NFR	Non-Functional Requirements
UI	User Interface
PIPEDA	Personal Information Protection and Electronic Documents Act
EHR	Electronic Healthcare Record
API	Application Programming Interface
IR	Integrity Requirement
AC	Access Requirement

2 General Information

2.1 Summary

This document provides a comprehensive description of the system tests specific to all requirements for a software application, RapidCare, that aims to streamline the healthcare documentation process aimed to be run as a web application. This document will be used as a verification tool to ensure that system fulfills all the requirements and needs of the user. This document will allow for an in-depth description of system tests for functional, non-functional, safety and security requirements. Additionally, it will outline the verification plan for design and implementation which will make sure that all requirements are fulfilled.
Along with this, the document will also include the software validation plan that will validate the requirements by making sure that the client who intends to use this application is satisfied with its features.
The system will allow the users to automate the documentation process by transcribing audio to text and generating reports based on the transcribed data. The system will also provide diagnostics and treatment plan suggestions based on the reports to further optimize the process.

2.2 Objectives

The objective of verification and validation plan is to build confidence in the correctness of the software and decide if the software is correctly following the requirements. Additionally, we want to achieve adequate usability of our software by making sure it satisfies with the user's needs, thus speeding up the documentation process. Another goal of this document is to test the accuracy of the input data to reflect that the software suggests correct diagnosis based on the text that has been transcribed from the audio conversation. Since the safety and security of the patient's information is a priority for this software, the plan is to conduct tests to make sure patient's information remains safe and secure.
The main focus will be on verifying the system's essential features to make sure it complies with safety regulations and protects patient information and healthcare professionals' confidence. Assuming that the external libraries are already tested by their implementation team, the priority of this document will be to test the requirements of the software to ease the patient care.

Due to time constraints and limited resources, we are not going to test the quality of usability. While the usability is important to test, the priority of this document is to test the fundamental requirements to ensure the software is accurate and reliable.

2.3 Challenge Level and Extras

In terms of the project challenge level, the project will come in the general category. The reasoning for this choice is supported below:

- **Domain Knowledge** – The documentation process has a lot of ins and outs which may differ between health organizations. Our supervisors and stakeholders will provide us with a base of the requirements, but further elicitation will be required to ensure that the requirements reflect a problem that truly exists. Additionally, since the whole patient journey is tracked we will have to survey other hospital staff as well to gain a further understanding.
- **Implementation Challenges** – There will be quite a few microservices required for this project where each microservice has high complexity. The performance of the microservices must be high as this use case requires quick response time. Additionally, since we are dealing with patient data security and privacy must be upheld. Lastly, the integration between all of the parts must be secure and undergo rigorous integration testing.

As part of the extras for this project, we will accomplish the following extras:

- **Usability testing** – This is designed for the users to assess how easily they can navigate and use the software. This will ensure that the requirements of the software perfectly align with the user's needs.
- **User Manual** – This will include instructions that will help the user to get started with the software. It will have an overview of the software along with some help resources for setup instructions and easy navigation for the user.

2.4 Relevant Documentation

The documents that are relevant to this project are:

- SRS
- Hazard analysis
- MG
- MIS

SRS and Hazard analysis documents list the FR, NFR, safety and security requirements which will assist us in developing tests for each of them. Moreover, MG and MIS documents give a structured approach of system's architecture and interface, ensuring that the system is thoroughly tested. This will help us to develop test cases for vital areas of the software.

3 Plan

This section will include details about the verification and validation team. In addition to this, this section will include SRS, design, validation and verification plan, and implementation verification plan. Lastly, it will outline details about automated testing and verification tools and software validation plan.

3.1 Verification and Validation Team

Here is a basic outline of the verification and validation team and their responsibilities. Please note that team members will switch roles between the tasks to ensure a well-rounded verification of the system. All team members will work collectively toward user documentation for this project.
Gurleen Rahi: Will focus on functional testing, specifically for transcription and report generation module. Will also focus on usability survey and design verification throughout the project.
Pranav Kalsi: Will focus on functional testing, specifically functional tests related to machine learning models. Also, will focus on integration testing and code verification throughout the project.
Inreet Kaur: Will focus on functional testing, specifically for various components in the data layer. Will also focus on safety requirement testing for

this project.
Moamen Ahmed: Will focus on functional testing specifically for authentication and account management components. Will also focus on non-functional testing of the system.
Kristen Burrows (Supervisor): Will help the team verify and provide feedback to improve the SRS, design, and verification and validation plan verification plan.

3.2 SRS Verification Plan

Each team member will perform a detailed review of the SRS to verify clarity, feasibility, and consistency across requirements. Identified issues will be added to the appropriate GitHub issue of the specific section. The team members can then make appropriate changes based on the feedback. Each team member will complete functional requirement tests for their assigned components as well as non-functional and safety requirement testing (as assigned). Once completed, at least two other team members will review and verify the tests using the sample input for the tests and verify the desired behavior. We will also ask for feedback from our peers, i.e. have another team review SRS and update it based on the feedback in the GitHub issues. In addition to this, we will then conduct a structured meeting review with our supervisor to gain feedback on the correctness and relevance of the requirements of the SRS document. During this meeting, we will have a walkthrough of the key sections of the SRS, outline any challenging or critical requirements, and discuss any issues flagged during internal reviews.
Once the system is complete, we will then have the supervisor and other stakeholders verify the system's functionality using specified inputs. They will also complete the usability survey (section 7.2) for the system.

3.3 Design Verification Plan

Our design verification plan ensures that the system design meets requirements for consistency, maintainability, scalability, and usability. This process will involve systematic reviews, integration testing, and structured feedback sessions with the supervisor and stakeholders.
Each team member will review the design of specific components to verify consistency with project requirements, maintainability, and scalability.

Identified issues will be added to the appropriate GitHub issue of the specific section. The team members can then make appropriate changes based on the feedback. The team will also conduct integration testing to check the compatibility of different components of the system and compatibility with the hardware.
Once the system is completed, the team will conduct a review session with the supervisor. In this meeting, we will provide a high-level overview of architecture and key components. We will present a sample user journey and gather feedback on the system's design for improvement.

Following is a checklist for the design verification plan:

- Verify the design meets all requirements identified in SRS and hazard analysis
- Verify each module is modular and maintainable
- Verify the design meets the coding standards
- Verify hardware and software compatibility
- Verify alignment with project objectives and stakeholder needs

3.4 Verification and Validation Plan Verification Plan

This document must be verified to make sure it is robust and reliable. The plan must have extensive coverage to ensure that all requirements are covered. The two main methods we will be using are the following:

- **VnV Reviews**: We will identify gaps in the Validation and Verification plan through reviews that we conduct with our peers and supervisor. This will help ensure completeness, accuracy, and feasibility of the plan.
  - **Checklist for Review:**
    * Verify that all requirements are covered in the plan
    * Ensure that the plan adheres to industry standards
    * Check for consistency and clarity in the plan
    * Ensure that the plan is feasible and can be carried out by any reader

    - Post review send respective items to developers (Pranav, Inreet, Gurleen, Moamen) to be resolved.

Through both of these processes we will be able to ensure that the plan covers all input and is clear and consistent such that it can be carried out by any reader.

3.5 Implementation Verification Plan

The implementation verification plan is really a lot of static review this is include code walkthroughs and code inspections in conjunction with static analyzers. The plan will be as follows:

1. Static analysis will be present at the CI/CD pipeline level, utilizing tools like Super-Linter in our GitHub Actions sequence, to catch code errors or style violations whenever code is committed or a pull request is created, this will also expedite the code review process as we can have a level of checking to catch errors.

2. Next, the team will conduct a high-level code walkthrough to identify any high-level logic errors. This will also verify the service interaction flows to ensure the desired business logic is fulfilled.

3. Finally, in any critical sections or features, we will perform a thorough code inspection, using the following checklist to ensure the implementation is verified:

    - Functionality aligns with requirements
    - Error handling is implemented correctly
    - Code is modular, maintainable and extendable

    Finally, the way critical sections will be identified is through the requirements, as well as reviews with our supervisor to ensure the critical use case flows are highlighted and verified.

This strategy will be implemented in section 4.

3.6 Automated Testing and Verification Tools

An outline of the CI/CD strategy can be found in section 7 of the Development Plan. A list of relevant frameworks will be found in section 10 of the same document.

3.7 Software Validation Plan

Validation of the project is critical and our supervisor will be vital for this process. Our supervisor will make sure that our software solves the user groups needs. Our supervisor is in an excellent position to validate the software as not only are they a domain expert they also fit the potential user group.

In the review sessions we intend on hold with our we ensure to validate the following to ensure the right software is built:

- Validate that the requirements cover all user needs and expectation (walkthrough of complete SRS).
- Confirm the feasibility of the requirements such that we can validate if they can even be built in the real-world.
- Does the software align with the expectations of stakeholders.
- Highlight gaps in compliance and fulfillment of industry standards.
- Post review send respective items to developers (Pranav, Inreet, Gurleen, Moamen) to be resolved.

Completing this checklist will ensure the software is validated.

4 System Tests

This section will include system tests for functional, non-functional, and safety and security requirements. In addition to this, we will include a traceability table for test cases and requirements.

4.1 Tests for Functional Requirements

This section contains the tests for the Functional Requirements. The subsections for these tests were created based on the subsections of the Functional Requirements listed in the SRS. Traceability for these requirements and tests can be found in the section 4.4.

4.1.1 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

   Control: Manual

   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.

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

   Test Case Derivation: The system will validate the input data, accept a complete and correct data input, and confirm a successful addition to the database.

   How test will be performed: The test controller will input a valid data object and check if the system is able to validate and accept the valid input. The controller will verify that a confirmation message appears and there is a valid new entry in the appropriate database.

2. test-FR1,4,8-2

   Control: Manual

   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.

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

Test Case Derivation: The system will validate the input data and prompt an error message outlining why the system is not able to accept the input data. No new document is added to the database.

How test will be performed: The test controller will input an invalid data object and check if the system is able to validate and reject the invalid input. The controller will also verify that an error message appears and there is no new entry in the appropriate database.

4.1.2 Remove a document from 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

   Control: Manual

   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.

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

   Test Case Derivation: The system should allow the deletion of an existing document when the correct identifier is provided.

   How test will be performed: The test controller will input a valid identifier and check if the system is able to validate and accept the valid input. The controller will verify that a confirmation message appears and the associated document no longer exists in the database.

2. test-FR2,5,9-2

   Control: Manual

   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.

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

Test Case Derivation: The system should be able to validate the provided identifier and prevent the deletion of any existing document.

How test will be performed: The test controller will input an invalid identifier and check if the system is able to validate and reject the invalid identifier. The controller will verify that an error message appears, and no document is deleted from the database.

4.1.3 Update document in 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

   Control: Manual

   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.

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

   Test Case Derivation: The system should allow to update the existing document when the correct identifier is provided.

   How test will be performed: The test controller will input a valid identifier and check if the system is able to validate and accept the valid input. The controller will verify that a confirmation message appears and the current document is updated in the database.

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

   Control: Manual

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.

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

Test Case Derivation: The system should be able to validate the provided input and prevent any invalid updates to the document.

How test will be performed: The test controller will input an invalid input and check if the system is able to validate and reject the invalid inputs. The controller will verify that an error message appears, and no document is updated in the database.

4.1.4 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

   Control: Manual

   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.

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

   Test Case Derivation: The system should be able to validate the provided credentials and allow to login to the system.

   How test will be performed: The test controller will input the valid credentials and check if the system is able to validate and accept the valid input. The controller will verify that a confirmation message appears and the user is able to login the database.

2. test-FR7-2

   Control: Manual

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.

Output: An error message outlining the invalid fields.

Test Case Derivation: The system should be able to validate the provided credentials and prevent unauthorized access to the database.

How test will be performed: The test controller will input any invalid credentials and check if the system is able to validate and reject the invalid credentials. The controller will verify that an error message appears and unauthorized access will be denied.

4.1.5 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 text.

1. test-FR11-1

   Control: Manual

   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.

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

   Test Case Derivation: The system should be able to validate the voice input by processing it and transcribe to text in real-time.

   How test will be performed: The test controller will input a valid audio chunk and check if the system is able to transcribe the valid input. The controller will verify that a confirmation message appears and the voice input is transcribed to text.

2. test-FR11-2

   Control: Manual

   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.

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

Test Case Derivation: The system should be able to validate the invalid input data format.

How test will be performed: The test controller will input an invalid data format and check if the system is able to validate and reject the input. The controller will verify that an error message appears.

4.1.6 Validate output of correct diagonsis and treatment plan

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 treatment plans with a high accuracy and based on the context.

1. test-FR12,13-1

   Control: Automatic

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

   Input: Transcribed text from patient-doctor interaction.

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

   Test Case Derivation: The system will create a validation set of data, where the model will be evaluated.

   How test will be performed: When the model is deployed it will be automatically evaluated such that it will have to have a classification accuracy greater or equal than 85%.

4.1.7 Validate input data for models

This subsection covers FR12 and FR13 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.

2. test-FR12,13-2

   Control: Automatic

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

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

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

   Test Case Derivation: The system will test if the chart data is relevant and correct.

   How test will be performed: When the system is deployed the model input will be tested such that it only accepts valid inputs.

3. test-FR12,13-3

   Control: Automatic

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

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

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

   Test Case Derivation: The system will test if the chart data is relevant and correct.

   How test will be performed: When the system is deployed the model input will be tested such that it only accepts valid inputs. It will output a error message if the output is wrong.

4.2 Tests for Nonfunctional Requirements

4.2.1 Look and Feel Requirement (NFR1)

1. test-AD1

   Type: Non-functional, Dynamic, Manual

   Initial State: UI is operational and accessible to users.

   Input: Users view the UI under normal operating conditions.

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

How this test will be performed: A test group of users will be given the system, and a set of routine UI interactions to perform. They will be asked to complete these interactions using the interface. After they do this, they will be given a Usability survey to verify that the interface meets aesthetic appeal and simplicity requirements.

2. test-AD2

   Type: Non-functional, Dynamic, Manual

   Initial State: UI is operational and accessible to users.

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

   Output: Feedback collected on satisfaction with the UI design.

   How this test will be performed: A test group of users will be given the system, and a set of common tasks to perform. They will be asked to complete these tasks using the interface. After they do this, they will be given a Usability survey to verify that at least 80% of users are satisfied with the design.

4.2.2 Validate AI Assistant Results

This subsection covers FR-14 from of the SRS document by testing that the system is able to effectively query the patient data.

1. test-FR14,14-1

   Control: Manual

   Initial State: The patient's profile is loaded and AI

   Input: Query regarding the patient is present.

   Output: A answer directly from the patient data (demographic information, SOAP Notes, etc...)

   Test Case Derivation: Manually users will test if the chart data is relevant and correct.

   How test will be performed: User will perform 5 common queries which will be common queries healthcare professionals need, phrased 2 different ways each.

4.2.3 Usability Requirement (NFR2)

1. test-UR1

   Type: Non-functional, Dynamic, Manual

   Initial State: System is deployed and accessible.

   Input: Users navigate and explore the system features independently.

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

   How this test will be performed: A test group of users will be given the system, and a set of core functions to locate. They will be asked to find these functions without guidance. After they do this, they will be given the usability survey to verify that users could locate core functions independently.

4.2.4 Performance Requirement (NFR3)

1. test-PR1

   Type: Non-functional, Dynamic, Manual

   Initial State: Transcription interface open and ready.

   Input: Real-time voice input provided by the user.

   Output: Real-time transcription displayed within an acceptable time frame (should not exceed 2 minutes).

   How this test will be performed: A test group of users will be given the system, and a set of voice inputs to transcribe. They will be asked to speak these inputs for real-time transcription. After they do this, the transcription delay will be noted and should not exceed the specified time frame.

4.2.5 Operational Requirement (NFR4)

1. test-OR1

   Type: Non-functional, Dynamic, Manual

   Initial State: System is deployed and accessible.

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.

How test will be performed: A test group of users will be given the system, and a set of common tasks to perform. After they complete these tasks, any abnormal behavior and errors will be noted.

4.2.6 Maintainability Requirement (NFR5)

1. test-MR1

   Type: Non-functional, Dynamic, Manual

   Initial State: System is deployed and accessible.

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

   Output: System successfully applies updates without impacting stability.

   How this test will be performed: A set of software updates will be applied, and it will be verified that system functionality remained stable after updates, ensuring that the system supports regular updates for bug fixes and feature enhancements.

4.2.7 Security Requirement (NFR6)

1. test-SR1

   Type: Non-functional, Dynamic, Manual

   Initial State: System is deployed and accessible.

   Input: Simulate unauthorized access attempts.

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

   How this test will be performed: A test group of users will be given the system, and a set of unauthorized access scenarios. They will be asked to attempt these scenarios while monitoring system responses. After they do this, it will be verified that all unauthorized attempts were properly blocked.

4.2.8 Cultural Requirement (NFR7)

1. test-CR1

   Type: Non-functional, Dynamic, Manual

   Initial State: System is deployed and accessible.

   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.

   How this test will be performed: A test group of users will be given the system and a set of culturally diverse input scenarios. Users will interact with the system using these inputs and it will be verified that the system responds in a culturally appropriate manner.

4.2.9 Legal Requirement (NFR8)

1. test-LR1

   Type: Non-functional, Dynamic, Manual

   Initial State: System is deployed and accessible.

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

   Output: System passes all compliance checks and shows relevant disclaimers with no exceptions.

   How test will be performed: A certified compliance auditor verifies data handling and data access policies, ensuring that the system displays appropriate medical disclaimers and feedback to the users.

4.2.10 Scalability Requirement (NFR9)

1. test-S1

   Type: Non-functional, Dynamic, Automated

   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.

How test will be performed: Use load testing tools like Apache to simulate concurrent user traffic and monitor system response times, server load, and throughput during the test.

4.3 Tests for Safety and Security Requirements

4.3.1 AC1

1. test-AC1-1

   Type: Functional, Dynamic, Manual

   Initial State: System is deployed and accessible.

   Input: Attempt to access system with invalid credentials.

   Output: All unauthorized access attempts are blocked.

   How this test will be performed: A test group of users will be given the system. They will be asked to attempt access the system with invalid credentials and it will be verified that access was properly blocked.

4.3.2 AC2

1. test-AC2-1

   Type: Functional, Dynamic, Manual

   Initial State: System is deployed and accessible.

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

   Output: System correctly displays UI based on the role and access permissions.

   How this test will be performed: A test group of users will be given the system, and a set of non-admin user credentials. They will be asked to attempt various administrative operations using these credentials and it will be verified that the users were not able to access those options based on the permissions.

4.3.3 IR1

1. test-IR1-1

   Type: Non-functional, Dynamic, Automated

   Initial State: System is deployed and accessible.

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

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

   How this test will be performed: A test group of users and the supervisor will be given the system, and a set of test credentials for concurrent authentication testing. They will be asked to perform multiple simultaneous authentication attempts and it will be verified that all credentials remained intact and system stability was maintained.

4.3.4 IR2

1. test-IR2-1

   Type: Functional, Dynamic, Manual

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

   Input: Submit an invalid data input.

   Output: An error message is displayed to the user with feedback to recover from error state. No data is added to any database.

   How test will be performed: The test controller will input an invalid data input and check if the system is able to validate and reject the invalid data. The controller will verify that an error message appears.

4.3.5 IR3

1. test-IR3-1

   Type: Functional, Dynamic, Automated

   Initial State: The prediction model is ready for to predict trestment plan and diagnosis.

Input: A test set of various patient medical charts.

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

How test will be performed: Automated testing of model outputs and confidence scores this assists in understanding how sure the model is in its classification.

4.3.6 IR4

1. test-IR4-1

   Type: Functional, Dynamic, Manual

   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.

   How test will be performed: The test controller will have a new input with the same data as an existing document and check if the system is able to validate and reject the duplicate record. The controller will verify that an error message appears.

4.3.7 IR5

1. test-IR5-1

   Type: Functional, Dynamic, Manual

   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 transcribed text from the input data matches the conversation with minimal to no interference.

   How test will be performed: The test controller will have a new data input and check if the system is able to validate and classify the data accurately. The controller will verify that the generated text matches

the spoken conversation with 90% percent accuracy. The accuracy will be assessed by comparing the system's output to the human transcription and calculating the percentage of correct words.

4.4 Traceability Between Test Cases and Requirements

Test ID	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FR12	FR13	FR14
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-FR12,13-2												×	×	
test-FR12,13-3												×	×	
test-FR14-1														×

Table 1: **Functional Requirements Tests Traceability**

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

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

Test ID	AC1	AC2	IR1	IR2	IR3	IR4	IR5
test-AC1-1	×						
test-AC2-1		×					
test-IR1-1			×				
test-IR2-1				×			
test-IR3-1					×		
test-IR4-1						×	
test-IR5-1							×

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

5 Unit Test Description

5.1 Unit Testing Scope

6 Unit Testing Scope

While comprehensive testing will e performed on several critical modules, certain modules like Serivce Layer module, Patient and Administrator model modules, and views in Patient View and Adminstrator view modules will not tested. This is due to various reasons.
Service Layer Module was not included in the unit testing scope as it used primarily for routing interactions and focus was placed on testing the core functionalities rather than the service layer, which is expected to be stable and well-defined.
Patient Model and Administrator Model Modules were not included in unit testing scope as they just provide the data structure for for various objects. The tests for integrity of the data are included in the Data Layer Module tests.
UI Components in Administrator View and Patient View Modules were not included in unit testing. This is because user acceptance testing and usability testing would provide better insights for overall user experience and interaction with the system can be evaluated in a more holistic manner.
By prioritizing the verification of core functionalities, we aimed to ensure that the most essential aspects of the system were thoroughly tested.

6.1 Tests for Functional Requirements

This section will include unit tests for functional requirements organised by different modules Details of the functional requiremnts can be found in SRS and details about different modules can be found in MIS and MG.

6.1.1 User Authentication Module

This subsection covers User Authentication Module from MIS. This module focuses on the authentication mechanism, including secure storage and validation of user credentials, and session management. The tests outlined below are designed to verify both the functionality and security of the authentication mechanisms. These tests were selected based on the situations users will

encounter under normal conditions and also considers potential edge cases to ensure comprehensive coverage of the authentication functionality.

Control: Automatic

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.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will simulate the input of valid email and password into the signup function. The system will be monitored to ensure that a user object is returned.
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 an error when the invalid credentials are provided.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will simulate the input of invalid email and password into the signup function. The system will be monitored to ensure that an error is thrown.
3. Login 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.
- **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will simulate the input of valid email and password into the signin function. The system will be monitored to ensure that a user object is returned.

4. Login 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 an error when the invalid credentials are provided.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will simulate the input of invalid email and password into the signin function. The system will be monitored to ensure that an error message is displayed.

6.1.2 Data Layer Module

This subsection covers Data Layer Module from MIS. The Data Layer Module is responsible for managing the storage and retrieval of data, including patient records and healthcare professionals records. The tests in this section were selected to validate the core functionalities of adding, updating, deleting, and retrieving data. They encompass both typical user interactions and potential error conditions to ensure that the data layer operates reliably and securely.

Control: Automatic

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.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will call the addHealthcareProfessional, addHospital, and addPatient functions with valid mock data. The database will be checked to confirm that the records have been added successfully.

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.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will call the addPatient function multiple times with the same mock data. The database will be checked to ensure that only one record exists.

    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.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will call the deleteHealthCareProfessional, deleteHospital, and deletePatient functions with valid mock data. The database will be checked to confirm that the records have been deleted successfully.

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.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will call the updatePatient function with valid mock data. The database will be checked to confirm that the record has been updated successfully.

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.

- **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will call the getPatient function with a valid patient id and check that the returned object matches the expected patient data.

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.
   - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will call the getPatient function with a non-existing patient id or invalid patient id and check that the returned value is null.

6.1.3 Administrator View Module and Patient View Module

This subsection covers Administrator View Module and Patient View Module from MIS. These modules provide the user interface for both administrators and patients. The tests included in this section were used to verify correct behaviour of different functions implemented to verify input in different fields ensuring that the system correctly rejects an invalid input by the user and cover all edge cases.

Control: Automatic

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.
- **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will input various invalid values for different field types and check that the appropriate error messages are returned.

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 valid 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.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will input various valid values for different field types and check that an empty string is returned for the error.

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.

- **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will input a valid date of birth and check that the returned age is correct.

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.
   - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will input a future date of birth and check that the returned age is 0.

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.
   - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will input today's date as the date of birth and check that the returned age is 0.

6.1.4 Diagnosis and Treatment Plan Prediction Module

This subsection covers Diagnosis and Treatment Plan Prediction Module from MIS. This module is responsible for providing diagnosis and treatment plan based on the transcribed text from the patient doctor interaction. The tests below were selected to verify the correctness and accuracy of the prediction based on the given context. However, these tests are just preliminary analysis; further stress testing would be required to verify the accuracy of the predictions.

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 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.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will send the predefined text to the API endpoint and verify if the the success rate of the responses meets the specified threshold.
2. Valid data input test:
    - **Input:** A dictionary that contains a valid transcription of a patient conversation.
    - **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 transcription to the API endpoint and asserts that the response status code is 200, indicating successful processing of the input.

- **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will send a POST request with valid transcription data to the API endpoint and check that the response status code is 200.

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 the 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.
   - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will send a POST request with invalid transcription data to the API endpoint and check that the response status code is 400.

6.1.5 Transcription Module

This subsection covers Transcription Module from MIS. The Transcription Module is responsible for converting audio input from patient interactions into text format. The tests in this section were selected to verify the accuracy and reliability of the transcription process. The tests include both normal and edge scenarios to ensure that the system can handle a range of valid and invalid inputs effectively and meet the critical need for accurate transcriptions in healthcare settings.

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 to match the asserted string, 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:** Asserts the transcribed text equal to the string containing expected output.
    - **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.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will connect to the Socket.IO server, send the valid audio file, and check that the emitted transcribed text matches the expected string.
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.
    - **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.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will connect to the Socket.IO server, send the invalid audio file, and check that the emitted transcribed text is empty.

6.1.6 Classification Module

This subsection covers Classification Module from MIS. The Classification Module processes the transcribed text to automatically fill patients charts with appropriate values for each field such as symptoms, allergies, and medications. The tests included in this section were chosen to assess whether the service can accurately classify and extract relevant data ensuring that it meets the necessary performance and accuracy standards.

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 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.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will send the predefined conversations to the API endpoint and monitor the success rate of the responses.
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.
- **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will send a POST request with valid input data to the API endpoint and check that the response status code is 200.

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.
    - **How test will be performed:** This test is automated and will be executed by running the unit test suite, which will send a POST request with empty input data to the API endpoint and check that the response status code is 400.

7 Appendix

7.1 Symbolic Parameters

N/A

7.2 Usability Survey Questions?

The following multiple choice questions will be presented to the user who will be using this software.

1. What operating system do you use?
   (a) Mac
   (b) Windows
2. Is this system compatible with the hardware?
   (a) Yes
   (b) No
3. Does this system satisfy all the needs of the user?
   (a) Yes
   (b) No
4. How intuitive is the UI? Rate performance.
   (a) Very confusing
   (b) Mostly confusing
   (c) Intuitive
   (d) Mostly intuitive
   (e) Very intuitive
5. How satisfied are you with the AI-assist feature of the system?
   (a) Not satisfied
   (b) Less satisfied

(c) Neutral

(d) Mostly satisfied

(e) Highly satisfied

6. How satisfied are you with the diagnosis and action plan based on the symptoms?

   (a) Not satisfied

   (b) Less satisfied

   (c) Neutral

   (d) Mostly satisfied

   (e) Highly satisfied

7. Are there any other functionalities that you want to list which can further improve the performance of this software?

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?
   This document has let us build more on the rough ideas of the requirements that we had brainstormed initially. While going through the outline of this document, we were able to provide the tests for functional requirements, non-functional requirements, and safety and security requirements. It also made us better understand the detailed procedure of system testing and plan for design validation and verification.

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. We had to collectively make a list of tests that are needed to be conducted for functional requirements, non-functional requirements, and safety and security requirements. We also needed to create a schedule to contribute to the template and review each other's work in the best way possible.

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.
As different abilities are added to the overall development plan, the project progresses more quickly thanks to the diversified knowledge of the team members. Technical expertise such as static and dynamic testing knowledge are very helpful to come up with a successful verification and validation plan for our project. Knowledge about how the external libraries can be used is also an asset for validating the system. Finally, being able to work on backend programming with an understanding of different server-side technologies, like Python, and React.js will be helpful in creating a dynamic database that secures patient data.

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?
   One can use a variety of resources to learn Python from Flask to become effective in backend programming. With these resources, developers can gain practical experience by building web applications that offer them flexibility. Using LeetCode to practice coding skills is an additional strategy. This allows users to modify the difficulty of the problems and proceed with their solution. Moreover, setting goals and taking regular pauses between tasks might help with time management and allow one to work more effectively. It's critical that each member of the team grasp every talent for everyone to be moving at the same speed. This is because it's a fantastic chance to learn and implement it in practical situations.