Deliverable 4: Testing

SCRIBBLES

Team Members:
Sanjana Nambiar, Symbat Bekzhigit, Saamia Shafqat and Levith Andrade Cuellar

Section 1: Introduction

1.1 Test Project Name

Scribbles Application: Testing Phase

1.2 Summary of the Rest of the Test Plan

For the later stages of testing the Scribbles Application, we executed a comprehensive plan covering multiple key functionalities crucial for healthcare professionals, particularly doctors and nurses. Our approach integrated a variety of testing techniques, including Black Box Testing, Load Testing, and Exploratory Testing, to assess features like messaging, paging, electronic health record (EHR) access, reminder management, and AI-powered requests. Each test case was meticulously designed to simulate real-world scenarios, such as messaging under different network conditions, paging with incomplete data, accessing EHRs with varying user permissions, and interacting with AI for data queries. We employed methods like Boundary Value Analysis and Equivalence Partitioning to ensure robust coverage across potential use cases. By combining these methods, we aimed to uncover any issues from functionality to security and usability, enhancing the platform's reliability and effectiveness in a clinical setting. This structured yet flexible testing approach was crucial for validating the application's performance under varied and complex conditions, preparing it for safe and efficient deployment in a healthcare environment.

Section 2: Feature Description

USE CASE (1): MESSAGE A USER
Primary Actor: Doctor / Nurse
Main success scenario:

1. **Doctor / Nurse** accesses the platform's messaging section.
2. **System** displays the current chats and displays an option to start a new conversation.
3. **Doctor / Nurse** starts a new conversation.
4. **System** displays the hospital's directory composed of other doctors and nurses (name, photo, specialty).
5. **Doctor / Nurse** selects the doctor or nurse they'd like to start a conversation with.
6. **System** displays a chat interface, a location for input, a keyboard, an option to include emojis and a send button.
7. **Doctor / Nurse** inputs their message and sends it.
8. **System** encrypts the message content using the relevant encryption standards.
9. **System** delivers the message to the recipient and updates the interface with the sent message.

Extensions:

<u>Searching Directory:</u>

3a. **System** displays the profiles in alphabetical order.

4a. **Doctor / Nurse** searches for a person that isn't registered under the hospital's directory.

4b. **System** alerts the user that it was unable to locate the person they were searching for.

<u>Acknowledgment Receipts:</u>

8a. **System** displays if the message has been delivered.

8b. **System** displays if the message has been read.

USE CASE (2): PAGE A USER (Assuming the user is logged into the application)

Primary Actor: Doctor / Nurse

Main success scenario:

1. **Doctor / Nurse** accesses the platform's pager section.
2. **System** displays a set of options to construct the request: recipient name, location, notification type, and custom message.
3. **Doctor / Nurse** searches for the recipient's name using the hospital's directory.
4. **Doctor / Nurse** selects a location for the request and a type of notification using a set of options.
5. **Doctor / Nurse** inputs text for the custom message.
6. **Doctor / Nurse** sends the request.

7. **System** delivers the request to the designated recipient.

Extensions:

Request Drafting:

5a. **System** alerts the doctor/nurse when the word count is reached.

6a. **System** notifies the doctor/nurse that there is missing information.

Notification Failure

7a. **System** alerts the doctor/nurse they're offline and asks them to use a different method for processing an urgent request.

7b. **System** logs the notification failure.

Notification Acknowledgement

8a. **System** forwards the notification to another healthcare professional in case the original recipient does not acknowledge the request in a timely manner.

USE CASE (3): VIEW AN ELECTRONIC HEALTH RECORD (EHR)

Primary Actor: Doctor / Nurse

Main success scenario:

1. **Doctor / Nurse** accesses the platform's records section.
2. **System** displays a list of patients (name, photo, age, gender) that the Doctor / Nurse is currently treating.
3. **System** presents the user with options to search for patient data by various criteria, such as patient name, medical record number, or admission date.
4. **Doctor / Nurse** selects a patient whose information they'd like to view.
5. **System** displays the contents of the patient's Electronic Health Record (EHR): personal information, medical history, diagnoses, etc.

Extensions:

2a. **System** displays the list in order of treatment start date.

Invalid Search Criteria:

4a. **Doctor / Nurse** searches for a patient that they're not assigned to.

4b. **System** alerts the user that it was unable to locate the patient they are looking for.

4c. **System** allows the user to revise their search criteria or initiate a new search.

Database Connection Failure:

5a. **System** encounters issues connecting to the hospital's database during the data retrieval

process.

5b. **System** displays an error message indicating the database connection failure and advises the user to check their internet connection or try again later.

5c. **Doctor / Nurse** can attempt to reconnect to the database.

USE CASE (4): CREATE A REMINDER

Primary Actor: Doctor / Nurse

Main success scenario:

1. **Doctor / Nurse** accesses the platform's reminder section.
2. **System** displays the current reminders and displays an option to create a new reminder.
3. **Doctor / Nurse** selects an option to create a new reminder.
4. **System** displays a set of options to construct the reminder: title, type, notes, date, time, and the reminder's recurrence.
5. **Doctor / Nurse** inputs text for the title and notes.
6. **Doctor / Nurse** selects the appropriate date, time and type associated with the reminder.
7. **Doctor / Nurse** set a reminder preference, including the method of notification.
8. **Doctor / Nurse** adds the reminder.
9. **System** saves the reminder and displays it with the other current reminders.

Extensions:

<u>Arranging existing reminders</u>

2a. **System** displays the current reminders in chronological order.

3a. **Doctor / Nurse** selects the option to delete a reminder.

3b. **Doctor / Nurse** selects the option to update a reminder.

<u>Invalid Reminder Details</u>

5a. **Doctor / Nurse** enters incomplete or invalid information for the reminder.

5b. **System** alerts the doctor/nurse when the notes word count is reached.

6a. **System** verifies the chosen date is in the future.

9a. **System** displays an error message prompting the user to provide valid details.

USE CASE (5): PERFORM AI-POWERED REQUEST

Primary Actor: Doctor / Nurse

Main success scenario:

1. **Doctor / Nurse** accesses the platform’s records section.
2. **Doctor / Nurse** selects a specific patient's record to work with.
3. **System** presents predetermined prompts with possible queries that can be made to the AI module.
4. **Doctor / Nurse** selects the query.
5. **System** utilizes the AI module to process the user's query.
6. **System** displays the AI-generated query results.
7. **Doctor / Nurse** reviews the query results.

Extensions:

Inaccurate Summarization:

7a. **Doctor / Nurse** may choose to manually review the patient records for accuracy or request further clarification from the AI module with additional queries.

Unresponsive AI:.

1a. **System** displays an error message indicating the unavailability of the AI service and suggests trying again later.

2a. **Doctor / Nurse** may opt to wait for the AI service to become responsive again or proceed with manual review of the patient records in the meantime.

Section 3. Assumptions

3.1 Test Case Exclusions

Some of the main test case scenarios that are not covered in our testing due to various reasons such as execution limitations, resource limitations, etc. are provided below:

1. **System Testing Under Unusual Operating Conditions:** While there are some functional tests for system testing under unusual operating conditions, there is limited information on testing the application under unusual operating conditions such as system overload, operating system updates, or hardware compatibility issues.

2. **User Acceptance Testing:** The document does not specify scenarios where end-users (doctors, nurses) test the application in a real-world setting to validate the usability and overall experience.

3. **Security and Vulnerability Testing**: Apart from general functionality, there are no explicit test cases addressing security protocols, data encryption beyond the application layer, or vulnerability assessments against external threats.

4. **Cross-Platform Compatibility:** Testing the application's performance and functionality across different devices and operating systems is not detailed, which could be crucial given the variety of devices in a healthcare setting.

5. **Long-Term Stability and Memory Leak Tests:** Due to the execution limitations, there are no test cases addressing the application's performance over extended periods or its behavior under continuous use conditions, which could potentially identify memory leaks or degradation of performance.

3.2 Test Tool

From the testing tools, we used an online pairwise generator tool pairwise.teremokgames.com for a few of our test cases, and we conducted most of our testing manually.

Section 4: Test Approach

4.1 Test Strategy

The following are some testing techniques we used. Each test case is labeled with the technique that it represents.

1. **Black Box Testing** is a software testing method where the goal is to evaluate the functionality of an application without knowing pretty much anything about its internal structure or code. This technique is used to check how the application behaves externally according to the requirements.
    a. **Boundary Value Analysis** is a black box testing technique in which tests are

designed to include representatives of boundary values. The technique runs on the assumption that errors often occur at the boundaries of input domains. The boundaries are often defined in the requirements.

b. **Equivalence Partitioning** is a testing method that divides input data into valid and invalid groups or partitions. The rationale is that a single test case for each partition should be representative of the entire partition. If a test case in a partition passes, the whole partition is more than likely to pass.
c. **Truth Table** is used to test logical operations. It lists all possible combinations of inputs and their corresponding or expected outputs. Truth tables come in handy when dealing with complex decision-making operations.
d. **Pairwise Testing** is a technique where every possible finite combination of pairs of input parameters is tested at least once. This technique helps when the number of inputs is substantial, and testing for all combinations would be impractical or simply inefficient. Pairwise testing helps to identify errors caused by the interaction of two factors, reducing the number of test cases significantly.
e. **Error guessing** is a type of black box testing technique and unlike systematic testing methods that follow a strict procedure, error guessing is more informal and relies on the experience and intuition of the tester to anticipate the kinds of errors that may occur in a software application. This technique is especially useful when it's difficult to identify all possible test cases through the more formal methods.

2. **Load Testing** is a software testing method used to determine a system's behavior under different conditions. The main purpose of load testing is to locate performance bottlenecks in a system and to ensure it can handle certain loads correctly.

3. **Exploratory Testing** is a software testing method that is unstructured. In exploratory testing, testers engage with the software without following predefined tests and instead explore its functionalities using their intuition. Exploratory testing allows testers to uncover subtle bugs and issues that can be overlooked by other testing techniques.

4.2 Rationale

1. Testing Tools and Techniques:

- **Manual Testing:** Primarily, the testing approach was manual, which allowed us to interact with the application intuitively and detect issues that automated tests might overlook.
- **Pairwise Testing:** We utilized Pairwise Testing for generating test cases that ensure all possible pairs of input parameters were covered at least once. This is efficient in scenarios where the number of inputs is substantial, and full combinatorial testing would be impractical.
- **Boundary Value Analysis and Equivalence Partitioning:** Boundary Value Analysis and Equivalence Partitioning techniques were used to define valid and invalid input ranges and to test edge cases, ensuring that the application handles data limits and expected input types correctly.

2. Automation and Scripts:

- **Pairwise Generators:** We used tools like pairwise.teremokgames.com for generating test case variants for pairwise testing, reducing the complexity and effort needed to ensure comprehensive parameter coverage.

3. Testing Each System Part:

- **Messaging Feature:** Tests include sending messages under different network conditions, at different times of the day, and with varying message lengths.
- **Paging System:** This is tested for various scenarios like sending pages with normal and missing information, ensuring the system alerts the user appropriately when information is incomplete.
- **EHR Viewing:** Boundary value analysis is used to simulate database connection issues and access restrictions, testing the system's ability to fetch and display EHR data correctly under different access levels.
- **AI-Powered Requests:** Testing involves ensuring that the AI correctly processes and summarizes EHR data.

4. Rationale for Approach:

- **Efficiency:** Techniques like pairwise testing ensure efficient use of testing resources.
- **Real-World Simulation:** Testing in scenarios that simulate real-world use cases (like network variations for messaging) ensures that the application will be reliable in actual use environments.

Section 5: Test Cases

5.1 Test Cases

<table>
<tr><th colspan="2">Test Case 1.1</th></tr>
<tr><td>Test Case Title</td><td>Messaging Feature: Sending Messages - Different Connection Levels</td></tr>
<tr><td>Use Case Tested</td><td>MESSAGE A USER</td></tr>
<tr><td>Technique Used</td><td>Pairwise Testing</td></tr>
<tr><td>Pre-Condition</td><td>Doctor / Nurse is logged into the application.
Doctor / Nurse has accessed the messaging feature.
Doctor / Nurse has typed a message.</td></tr>
<tr><td>Proof</td><td>Pairwise Testing is concerned with generating “test case variants that guarantee ‘all-pairs’ coverage”.

To test the performance of our messaging feature under different levels of internet connection we can employ Pairwise Testing by running through the following scenarios:

I first identified the parameters and their potential values.
<table>
<tr><th>Message Length</th><th>Connection Strength</th></tr>
<tr><td>Short (50 characters max)</td><td>No Connection</td></tr>
<tr><td>Medium (50 - 200 characters)</td><td>Low Connection</td></tr>
<tr><td>Long (200+ characters)</td><td>Strong Connection</td></tr>
</table>
I then entered this information in a Pairwise Generator to obtain the following test case variants:
<table>
<tr><th>#</th><th>Message Length</th><th>Connection Strength</th></tr>
<tr><td>1</td><td>Short</td><td>No Connection</td></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td></td>
<td>
<table>
<tr><td>2</td><td>Short</td><td>Low Connection</td></tr>
<tr><td>3</td><td>Short</td><td>Strong Connection</td></tr>
<tr><td>4</td><td>Medium</td><td>No Connection</td></tr>
<tr><td>5</td><td>Medium</td><td>Low Connection</td></tr>
<tr><td>6</td><td>Medium</td><td>Strong Connection</td></tr>
<tr><td>7</td><td>Long</td><td>No Connection</td></tr>
<tr><td>8</td><td>Long</td><td>Low Connection</td></tr>
<tr><td>9</td><td>Long</td><td>Strong Connection</td></tr>
</table>
</td>
</tr>
<tr>
<td>Input</td>
<td>
Short length message:
[under No Connection] “Got it, see you at 7! 👍”
[under Low Connection] “Got it, see you at 7! 👍”
[under Strong Connection] “Got it, see you at 7! 👍”

Medium length message:
[under No Connection] “Reminder: Meeting tomorrow at 10 AM in Conference Room Saadiyat. Please bring the revised medical reports and be ready to discuss.”
[under Low Connection] “Reminder: Meeting tomorrow at 10 AM in Conference Room Saadiyat. Please bring the revised medical reports and be ready to discuss.”
[under Strong Connection] “Reminder: Meeting tomorrow at 10 AM in Conference Room Saadiyat. Please bring the revised medical reports and be ready to discuss.”

Long message:
[under No Connection] “Hi everyone, just wanted to share a quick update on the medical reports. We've completed the initial phases ahead of schedule, which is great news! However, we're facing some delays with hospital administration, so I'm adjusting the timeline accordingly.
</td>
</tr>
</table>

	Let's discuss our next steps in detail during tomorrow's Zoom meeting. Thanks!" [under Low Connection] "Hi everyone, just wanted to share a quick update on the medical reports. We've completed the initial phases ahead of schedule, which is great news! However, we're facing some delays with hospital administration, so I'm adjusting the timeline accordingly. Let's discuss our next steps in detail during tomorrow's Zoom meeting. Thanks!" [under Strong Connection] "Hi everyone, just wanted to share a quick update on the medical reports. We've completed the initial phases ahead of schedule, which is great news! However, we're facing some delays with hospital administration, so I'm adjusting the timeline accordingly. Let's discuss our next steps in detail during tomorrow's Zoom meeting. Thanks!"
Steps to Execute	*<u>Note:</u> Repeat for all connection levels.* *<u>Note:</u> To verify that a notification was sent and delivered correctly we need two devices.* 1. Prepare a script that feeds custom input to the application. 2. Physically relocate to a spot where there is {no connection, low connection, strong connection}. 3. Load the application and sign-in with a dummy account. 4. Open the messaging feature and run the script. 5. If there is a connection, await the notification on a test recipient device. 6. If there is a connection, verify that the notification was delivered correctly.
Expected Results	If there is **<u>no connection</u>**, the system should fail to send / deliver the message and should alert the user that their message could not be delivered. If there is **<u>low connection</u>** or **<u>strong connection</u>**, the system should be able to send / deliver the message (with marginal differences in send

	time depending on message length).

Test Case 1.2	
Test Case Title	Messaging Feature: Message Input - Composition
Use Case Tested	**MESSAGE A USER**
Technique Used	**Equivalence Partitioning**
Pre-Condition	**Doctor / Nurse** is logged into the application. **Doctor / Nurse** has accessed the messaging feature. **Doctor / Nurse** has typed a message.
Proof	Equivalence Partitioning involves separating the input “into a finite number of subdomains” where each “subdomain is known as an equivalence class, and it serves as a source of at least one test input”. To test how our messaging handles different kinds of input we can employ Equivalence partitioning by taking into account the following guidelines: **Valid Partition** / **Invalid Partition** A message may have . . . / A message must not have . . . • Letters • Numbers • Special Characters • Emojis / • Executables • Documents • Images • Videos
Input	*These are some example of the input we can try:* [Valid Partition] “Hey Doctor” [Valid Partition] “1234567890”

	[Valid Partition] “Great work! Did you finish the report? @Nurse Johnson.” [Valid Partition] “Happy Birthday Doctor! 🎂.” [Valid Partition] “こんにちは” [Invalid Partition] *executable.py* [Invalid Partition] *image.png* [Invalid Partition] *video.mov* [Invalid Partition] *document.pdf*
Steps to Execute	1. Prepare a script that feeds custom input to the application. 2. Load the application and sign-in with a dummy account. 3. Open the pager feature and run the script.
Expected Results	If the input qualifies as part of the **valid partition**, the system sends the message correctly. If the input qualifies as part of the **invalid partition**, the system is able to identify the invalid input, alerts the user and does not send the message.

Test Case 1.3	
Test Case Title	Messaging Feature: Message Sending - Times of Day
Use Case Tested	**MESSAGE A USER**
Technique Used	**Pairwise Testing**
Pre-Condition	**Doctor / Nurse** is logged into the application. **Doctor / Nurse** has accessed the messaging feature. **Doctor / Nurse** has typed a message.
Proof	Pairwise Testing is concerned with generating “test case variants that guarantee ‘all-pairs’ coverage”. To test the performance of our messaging feature during different times of day we can employ Pairwise Testing by running through the following

<table>
<tr>
<td></td>
<td>scenarios:

I first identified the parameters and their potential values.
<table>
<tr><th>Recipient</th><th>Message Length</th><th>Time of Day</th></tr>
<tr><td>Doctor</td><td>Short</td><td>Morning</td></tr>
<tr><td>Nurse</td><td>Medium</td><td>Afternoon</td></tr>
<tr><td>None</td><td>Long</td><td>Night</td></tr>
</table>
I then entered this information in a Pairwise Generator to obtain the following test case variants:
<table>
<tr><th>#</th><th>Recipient</th><th>Message Length</th><th>Time of Day</th></tr>
<tr><td>1</td><td>Doctor</td><td>Short</td><td>Morning</td></tr>
<tr><td>2</td><td>Doctor</td><td>Medium</td><td>Afternoon</td></tr>
<tr><td>3</td><td>Doctor</td><td>Long</td><td>Night</td></tr>
<tr><td>4</td><td>Nurse</td><td>Medium</td><td>Night</td></tr>
<tr><td>5</td><td>Nurse</td><td>Long</td><td>Morning</td></tr>
<tr><td>6</td><td>Nurse</td><td>Short</td><td>Afternoon</td></tr>
<tr><td>7</td><td>None</td><td>Long</td><td>Afternoon</td></tr>
<tr><td>8</td><td>None</td><td>Short</td><td>Night</td></tr>
<tr><td>9</td><td>None</td><td>Medium</td><td>Morning</td></tr>
</table>
</td>
</tr>
<tr>
<td>Input</td>
<td>[during the Morning, for a Doctor] “Got it, see you at 7! 👍”
[during the Afternoon, for a Doctor] “Reminder: Meeting tomorrow at 10 AM in Conference Room Saadiyat. Please bring the revised medical reports and be ready to discuss.”
[during the Night, for a Doctor] “Hi everyone, just wanted to share a quick update on the medical reports. We've completed the initial phases ahead of schedule, which is great news! However, we're facing some delays with hospital administration, so I'm adjusting the timeline</td>
</tr>
</table>

	accordingly. Let's discuss our next steps in detail during tomorrow's Zoom meeting. Thanks!” [during the Night, for a Nurse] “Reminder: Meeting tomorrow at 10 AM in the Conference Room Saadiyat. Please bring the revised medical reports and be ready to discuss.” [during the Morning, for a Nurse] “Hi everyone, just wanted to share a quick update on the medical reports. We've completed the initial phases ahead of schedule, which is great news! However, we're facing some delays with hospital administration, so I'm adjusting the timeline accordingly. Let's discuss our next steps in detail during tomorrow's Zoom meeting. Thanks!” [during the Afternoon, for a Nurse] “Got it, see you at 7! 👍” [during the Afternoon, for None] “Hi everyone, just wanted to share a quick update on the medical reports. We've completed the initial phases ahead of schedule, which is great news! However, we're facing some delays with hospital administration, so I'm adjusting the timeline accordingly. Let's discuss our next steps in detail during tomorrow's Zoom meeting. Thanks!” [during the Night, for None] “Got it, see you at 7! 👍” [during the Morning, for None] “Reminder: Meeting tomorrow at 10 AM in the Conference Room Saadiyat. Please bring the revised medical reports and be ready to discuss.”
Steps to Execute	*<u>Note:</u> Repeat for all times of day.* *<u>Note:</u> To verify that a notification was sent and delivered correctly we need two devices.* 1. Prepare a script that feeds custom input to the application. 2. Wait until the appropriate time of day {morning, afternoon, night}. 3. Load the application and sign-in with a dummy account. 4. Open the messaging feature and run the script.

	5. Await the notification on a test recipient device.
Expected Results	If a **recipient was indicated**, the system should be able to send / deliver the message (with marginal differences in send time depending on the time of day and message length). If a **recipient was not indicated**, the system should not be able to send / deliver the message.

Test Case 1.4	
Test Case Title	Message Length Boundary Testing
Use Case Tested	**MESSAGE A USER**
Technique Used	**Boundary Value Analysis**
Pre-Condition	The user is on the chat user interface, and the text box is ready to accept input.
Proof	For message length boundary testing, the maximum allowed length of a text message serves as the upper boundary of the input range. By attempting to input a message with the maximum allowed length plus one word/character, we are intentionally testing the behavior of the application when it encounters this upper boundary.
Input	Text message of maximum length (max_length) + 1 word/character.
Steps to Execute	1. Type a message in the chat interface until it reaches the maximum allowed length. 2. Attempt to type one additional word/character beyond the maximum allowed length. 3. Observe the behavior of the application.
Expected Results	The application should display a prompt indicating that the maximum message length has been reached. The user should be prevented from

	entering any additional text beyond the maximum allowed length.

Test Case 1.5	
Test Case Title	Empty Message Testing
Use Case Tested	**MESSAGE A USER**
Technique Used	**Boundary Value Analysis**
Pre-Condition	The user is on the chat user interface, and the text box is ready to accept input.
Proof	In this test case, we are testing the behavior of the application when it encounters the lower boundary of the input range, represented by an empty message. This scenario is essential to ensure that the application behaves correctly even in edge cases.
Input	Pressing send button without entering anything in the text box.
Steps to Execute	1. Leave the chat interface text box empty. 2. Press the send button to attempt sending the empty message. 3. Observe the behavior of the application.
Expected Results	The application should display a prompt indicating that the message being sent does not include any text. Sending an empty message should not be allowed, and the user should be prompted to enter at least one word before sending the message.

Test Case 1.6	
Test Case Title	Sending a message to a newly added user.
Use Case Tested	**MESSAGE A USER**
Technique Used	**Equivalence Partitioning**

<table>
<tr><td>Pre-Condition</td><td>A new user has been added to the hospital directory.</td></tr>
<tr><td>Proof</td><td>
<table>
<tr><th>User added to the directory within last 10 minutes</th><th>No change to the User in the past 10 minutes</th><th>User removed from the directory within the last 10 minutes</th></tr>
<tr><td>This partition focuses on testing the scenario where a user has been recently added to the directory. It ensures that the system accurately reflects the addition of new users and allows for their selection when sending messages.</td><td>This partition addresses the scenario where a user’s information remains unchanged within a specific timeframe. It validates the system’s ability to maintain consistency in displaying user information and handling interactions with users who have not been recently modified.</td><td>This partition examines the scenario where a user has been removed from the directory within the last 10 minutes. It ensures that the system correctly updates its records to reflect the removal of users and appropriately handles attempts to interact with removed users.</td></tr>
</table>
</td></tr>
<tr><td>Input</td><td>Searching and selecting the newly added user from the directory.</td></tr>
<tr><td>Steps to Execute</td><td>1. Access the hospital directory section of the application.
2. Initiate a search for the newly added user by name or other relevant criteria.
3. Locate the newly added user in the search results.
4. Select the newly added user as a contact to start a conversation.
5. Attempt to send a message to the selected user.
6. Observe the behavior of the application.</td></tr>
</table>

Expected Results	The newly added user should appear in the search results within a reasonable time frame after being added to the directory. The user should be selectable as a contact for messaging, and the message should be sent successfully without any errors.

Test Case 1.7			
Test Case Title	Searching and sending a message to a recently removed user from the hospital directory.		
Use Case Tested	**MESSAGE A USER**		
Technique Used	**Equivalence Partitioning**		
Pre-Condition	A user has been recently removed from the hospital directory.		
Proof			

User added to the directory within last 10 minutes	**No change to the User in the past 10 minutes**	**User removed from the directory within the last 10 minutes**
This partition focuses on testing the scenario where a user has been recently added to the directory. It ensures that the system accurately reflects the addition of new users and allows for their selection when	This partition addresses the scenario where a user's information remains unchanged within a specific timeframe. It validates the system's ability to maintain consistency in displaying user	This partition examines the scenario where a user has been removed from the directory within the last 10 minutes. It ensures that the system correctly updates its records to reflect the removal of

	<table><tr><td>sending messages.</td><td>information and handling interactions with users who have not been recently modified.</td><td>users and appropriately handles attempts to interact with removed users.</td></tr></table>
Input	Selecting the removed user as a contact and attempting to send a message.
Steps to Execute	1. Access the messaging section of the application. 2. Attempt to select the recently removed user from the hospital directory as a contact to start a conversation. 3. Type and send a message to the removed user. 4. Observe the behavior of the application.
Expected Results	The removed user should not appear in the list of available contacts for messaging, or if somehow selected, there should be an indication that the user is no longer part of the hospital directory. Attempting to send a message to the removed user should result in an error message or notification indicating that the user cannot be reached.

Test Case 1.8	
Test Case Title	Normal Input for Messaging
Use Case Tested	**MESSAGE A USER**
Technique Used	**Equivalence Partitioning**
Pre-Condition	User is logged in and at the message input screen.
Proof	Equivalence Classes: ● Class 1: Valid input (1-1000 characters) ● Class 2: Invalid input (0 characters)

	• Class 3: Invalid input (>1000 characters)
Input	"Hello, this is a test message!"
Steps to Execute	1. Enter text "Hello, this is a test message!" in the input field (25 characters long, falls in Class 1). 2. Press send.
Expected Results	Message is sent successfully and appears in the chat interface. No error messages should be displayed.

Test Case 1.9	
Test Case Title	Message Delivery Time Analysis
Use Case Tested	**MESSAGE A USER**
Technique Used	**Error Guessing**
Pre-Condition	Users have active accounts on the messaging platform. Users are connected to the internet.
Proof	This test case uses error guessing to anticipate potential issues in the messaging system, focusing on delays and inconsistencies in message delivery. Users expect messages to arrive promptly and in order, regardless of time or message frequency. By considering real-world scenarios and past experiences, the test ensures the system reliably delivers messages and maintains conversation order.
Input	Send a series of messages from one user to another at different times of the day.
Steps to Execute	1. Log in to the messaging application with the sender's account. 2. Navigate to the conversation with the recipient. 3. Compose a message and send it to the recipient. 4. Repeat steps 2-3 multiple times, sending messages at varying

	intervals. 5. Log in to the messaging application with the recipient's account. 6. Monitor the arrival time of each message in the conversation. 7. Note any delays in message delivery or discrepancies in the message order.
Expected Results	• Messages should be delivered promptly, typically within a few seconds to a minute. • Messages should be displayed in the correct order, reflecting the sequence in which they were sent by the sender. • There should be no significant delays or inconsistencies in message delivery, regardless of the time of day or frequency of messages sent.

Test Case 2.1	
Test Case Title	Pager Feature: Pager request with a Normal Input for Messaging
Use Case Tested	**PAGE A USER**
Technique Used	**Equivalence Partitioning**
Pre-Condition	**Doctor / Nurse** is logged into the application. **Doctor / Nurse** has accessed the pager feature. **Doctor / Nurse** has selected a recipient, room number and notification type.
Proof	Equivalence Partitioning helps us to identify and test for classes with correct and incorrect input separately. Equivalence classes for message input in this case would be: • Class 1: Valid input (1-50 characters) • Class 2: Invalid input (0 characters) • Class 3: Invalid input (>50 characters)

Input	Message with 30 characters: "Hello, this is a test message!"
Steps to Execute	1. Enter text "Hello, this is a test message!" in the input field (30 characters long, falls in Class 1). 2. Press send.
Expected Results	• Pager Request is sent to the designated recipient. • Confirmation is displayed to the sender that the pager request has been sent.

Test Case 2.2	
Test Case Title	Pager Feature: Room Selection - Selection Functionality
Use Case Tested	**PAGE A USER**
Technique Used	**Boundary Value Analysis**
Pre-Condition	**Doctor / Nurse** is logged into the application. **Doctor / Nurse** has accessed the pager feature. **Doctor / Nurse** has selected a recipient.
Proof	Boundary Value Analysis is concerned with evaluating the “behavior of a system at the boundaries of input domains”. To test the pager’s room selection functionality we can use Boundary Value Analysis by preparing the following input: *In a small hospital there are approximately 100 beds.* *Let’s assume there are 100 rooms as well.*

Invalid (min - 1)	**Valid (min, min+, max, max-)**	**Invalid (max + 1)**
Room _____ :		
No Room Selected	Room 1, Room 2, Room 100, Room 99	Room 101

Input	• Select: No Room • Select: Room 1 • Select: Room 2 • Select: Room 100 • Select: Room 99 • Select: Room 101
Steps to Execute	1. Prepare a script that feeds custom input to the application. 2. Load the application and sign-in with a dummy account. 3. Open the pager feature and run the script.
Expected Results	If a **valid room number** was selected, the application should allow the user to send the pager request. If an **invalid room number** was selected (which would be rare given that the user is given fixed options), the application should alert the user and prevent them from sending the pager request. If **no room number** was selected, the application should alert the user and prevent them from sending the pager request.

Test Case 2.3	
Test Case Title	Pager Feature: Notification System - Urgency
Use Case Tested	**PAGE A USER**
Technique Used	**Truth Table**
Pre-Condition	**Doctor / Nurse** is logged into the application. **Doctor / Nurse** has accessed the pager feature. **Doctor / Nurse** has selected a recipient, room number and notification type.
Proof	A Truth or Decision table displays “a combination of conditions to be met and actions to be taken”. For our pager to work as intended it must be able to deliver notifications accurately, with the level of urgency indicated

<table>
<tr>
<td></td>
<td>by the user. To test this we can use a Decision Table for different kinds of pager requests:

<table>
<tr><th></th><th colspan="8">Rules</th></tr>
<tr><th>Conditions</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th></tr>
<tr><td>Doctor</td><td>T</td><td>T</td><td>T</td><td>T</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Nurse</td><td></td><td></td><td></td><td></td><td>T</td><td>T</td><td>T</td><td>T</td></tr>
<tr><td>Custom Message</td><td>T</td><td>F</td><td>T</td><td>F</td><td>T</td><td>F</td><td>T</td><td>F</td></tr>
<tr><td>Urgent Notification</td><td>T</td><td>T</td><td></td><td></td><td>T</td><td>T</td><td></td><td></td></tr>
<tr><td>Semi-Urgent Notification</td><td></td><td></td><td>T</td><td>T</td><td></td><td></td><td>T</td><td>T</td></tr>
<tr><th>Actions</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th></tr>
<tr><td>Deliver Urgently</td><td>✔</td><td>✔</td><td></td><td></td><td>✔</td><td>✔</td><td></td><td></td></tr>
<tr><td>Deliver Semi-Urgently</td><td></td><td></td><td>✔</td><td>✔</td><td></td><td></td><td>✔</td><td>✔</td></tr>
</table>
</td>
</tr>
<tr>
<td>Input</td>
<td>

A request sent by a [Doctor] with a [Custom Message] and [Urgent Notification].
A request sent by a [Doctor] with [No Custom Message] and [Urgent Notification].
A request sent by a [Doctor] with a [Custom Message] and [Semi-Urgent Notification].
A request sent by a [Doctor] with [No Custom Message] and [Semi-Urgent Notification].
A request sent by a [Nurse] with a [Custom Message] and [Urgent Notification].
A request sent by a [Nurse] with [No Custom Message] and [Urgent Notification].

</td>
</tr>
</table>

	• A request sent by a [Nurse] with a [Custom Message] and [Semi-Urgent Notification]. • A request sent by a [Nurse] with [No Custom Message] and [Semi-Urgent Notification].
Steps to Execute	*<u>Note:</u> To verify that a notification was sent and delivered correctly we need two devices.* 1. Prepare a script that feeds custom input to the application. 2. Load the application and sign-in with a dummy account. 3. Open the pager feature and run the script. 4. Await the notification on a test recipient device. 5. Verify that the notification was delivered correctly.
Expected Results	If the notification was supposed to be of **<u>high urgency</u>**, the system should deliver the notification within 5 seconds, and should label the notification as such. If the notification was supposed to be of **<u>medium urgency</u>**, the system should deliver the notification within 10 seconds, and should label the notification as such.

Test Case 2.4	
Test Case Title	Pager Feature: Paging Request with Missing Information
Use Case Tested	**PAGE A USER**
Technique Used	**Pairwise Testing**
Pre-Condition	**Doctor/Nurse** is logged into the platform.
Proof	Since we have 4 input parameters each of which has 2 input options, there are a lot of possible combinations expected for an incomplete input case. Pairwise Testing efficiently addressed this by generating a test suite that covers all possible pairs of parameter values at least once. This approach ensures that the interactions between any two

<table>
<tr>
<td></td>
<td>parameters are tested, which helps in identifying and isolating faults related to parameter interactions, without the need to test every possible combination. This makes our testing process both effective and efficient.

A sample pairwise distribution table is provided below:

<table>
<tr><td>Conditions</td><td>Recipient</td><td>Location</td><td>Notification type</td><td>Message</td></tr>
<tr><td rowspan="8"></td><td>T</td><td>F</td><td>F</td><td>F</td></tr>
<tr><td>F</td><td>F</td><td>T</td><td>T</td></tr>
<tr><td>F</td><td>T</td><td>T</td><td>F</td></tr>
<tr><td>F</td><td>T</td><td>F</td><td>T</td></tr>
<tr><td>T</td><td>T</td><td>F</td><td>T</td></tr>
<tr><td>T</td><td>T</td><td>T</td><td>F</td></tr>
<tr><td>T</td><td>F</td><td>T</td><td>T</td></tr>
<tr><td>T</td><td>F</td><td>F</td><td>F</td></tr>
<tr><td>Actions</td><td colspan="4"></td></tr>
<tr><td>Proceed with the request</td><td>No</td><td>No</td><td>No</td><td>No</td></tr>
<tr><td>Display error message</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td></tr>
</table>
</td>
</tr>
<tr>
<td>Input</td>
<td>

Recipient: "Nurse B",
Notification Type: "Urgent",
Message: "Need assistance"

</td>
</tr>
</table>

Steps to Execute	1. Access the Paging section. 2. Fill in requested fields except for the location field. 3. Attempt to send the request.
Expected Results	• System alerts the sender with the message: "Missing information" and asks to complete all required fields. • Doctor/nurse is forwarded to the section where they need to add the missing information.

Test Case 2.5	
Test Case Title	Pager Feature: Notification Acknowledgment Delay
Use Case Tested	**VIEW AN EHR**
Technique Used	**Truth Table**
Pre-Condition	**Doctor/Nurse** is logged into the platform, and a paging request is sent to another **Doctor/Nurse** (i.e. “Doctor A”). Notification Time: Assume current time is 12:00 PM
Proof	Using a truth table to test interaction between delayed acknowledgment and notification forwarding. Sample truth table is provided below:

Conditions		
On time acknowledgement	T	F
Delayed or No acknowledgement	F	T
Actions		
Forward to designated alternate	No	Yes

	<table><tr><td>No forward action taken</td><td>Yes</td><td>No</td></tr></table>
Input	• Recipient: "Doctor A", • Location: "ER", • Notification Type: "Urgent", • Message: "Need assistance"
Steps to Execute	1. Send a paging notification to "Doctor A" at 12:00 PM. 2. Monitor the system for acknowledgement from "Doctor A". 3. “Doctor A” fails to acknowledge the request in a timely manner.
Expected Results	• System notifies the sender that the recipient failed to respond to notification in a timely manner. • System asks the sender to choose another recipient. • System logs notification failure status for “Doctor A”.

Test Case 2.6	
Test Case Title	Pager Feature: Complete Paging Request
Use Case Tested	**PAGE A USER**
Technique Used	**Truth Table**
Pre-Condition	Doctor/Nurse is logged in and on the Paging section.
Proof	• We can use a decision table to verify behavior associated with all required input fields being correctly filled.

<table>
<tr>
<td></td>
<td>

Sample Decision table is provided below (it doesn’t cover the cases when two or more conditions are false at the same time for the simplicity purposes):

<table>
<tr><td>Conditions</td><td colspan="7">Is provided?</td></tr>
<tr><td>Recipient</td><td>T</td><td>T</td><td>T</td><td>T</td><td>F</td><td>F</td><td>T</td></tr>
<tr><td>Location</td><td>T</td><td>F</td><td>T</td><td>T</td><td>F</td><td>T</td><td>T</td></tr>
<tr><td>Notification type</td><td>T</td><td>T</td><td>F</td><td>T</td><td>F</td><td>T</td><td>T</td></tr>
<tr><td>Message</td><td>T</td><td>T</td><td>T</td><td>F</td><td>F</td><td>T</td><td>T</td></tr>
<tr><td>Actions</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Proceed with the request</td><td>Yes</td><td>No</td><td>No</td><td>No</td><td>No</td><td>No</td><td>Yes</td></tr>
<tr><td>Display error message</td><td>No</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td><td>No</td></tr>
</table>
</td>
</tr>
<tr>
<td>Input</td>
<td>

Recipient: "Nurse B",
Location: "ER",
Notification Type: "Urgent",
Message: "Need assistance"

</td>
</tr>
<tr>
<td>Steps to Execute</td>
<td>

Fill in all requested fields as specified.
Send the pager request.

</td>
</tr>
<tr>
<td>Expected Results</td>
<td>

Pager Request is sent to the designated recipient.
Confirmation is displayed to the sender that the pager request has been sent.

</td>
</tr>
</table>

<table>
<tr><th colspan="2">Test Case 2.7</th></tr>
<tr><td>Test Case Title</td><td>Pager Feature: Pager request with Empty Message Field</td></tr>
<tr><td>Use Case Tested</td><td>PAGE A USER</td></tr>
<tr><td>Technique Used</td><td>Boundary Value Analysis</td></tr>
<tr><td>Pre-Condition</td><td>Doctor / Nurse is logged into the platform and on the Paging Section.
Doctor / Nurse populates normally all other fields except for “message” field.</td></tr>
<tr><td>Proof</td><td>Boundary value analysis method allows us to check for off-by-one errors at the boundary, in this case, the minimum boundary of input length (0 characters).

Distribution of boundary values is provided below:

<table>
<tr><th>Invalid
(min - 1)</th><th>Valid
(min, min+, max, max-)</th><th>Invalid
(max + 1)</th></tr>
<tr><td colspan="3">Message with _____ characters:</td></tr>
<tr><td>0</td><td>1, 2, 50, 49</td><td>51</td></tr>
</table>
</td></tr>
<tr><td>Input</td><td>An empty string: ""</td></tr>
<tr><td>Steps to Execute</td><td>1. Enter no text in the message input field.
2. Press send.</td></tr>
<tr><td>Expected Results</td><td>● System alerts the sender that the “message” field cannot be empty.
● Doctor/nurse is forwarded back to the section where they need to re-enter the message.</td></tr>
</table>

<table>
<tr><th colspan="2">Test Case 2.8</th></tr>
<tr><td>Test Case Title</td><td>Pager Feature: Pager request with Message Exceeding the Maximum Length Boundary</td></tr>
<tr><td>Use Case Tested</td><td>PAGE A USER</td></tr>
<tr><td>Technique Used</td><td>Boundary Value Analysis</td></tr>
<tr><td>Pre-Condition</td><td>Doctor / Nurse is logged into the platform and on the Paging Section. Doctor / Nurse populates normally all other fields except for the “message” field.</td></tr>
<tr><td>Proof</td><td>Boundary value analysis method allows us to check for off-by-one errors at the boundary, in this case, the maximum boundary of input length (>50 characters).

Sample distribution of boundary values is provided below:

<table>
<tr><th>Invalid
(min - 1)</th><th>Valid
(min, min+, max, max-)</th><th>Invalid
(max + 1)</th></tr>
<tr><td colspan="3">Message with _____ characters:</td></tr>
<tr><td>0</td><td>1, 2, 50, 49</td><td>51</td></tr>
</table>
</td></tr>
<tr><td>Input</td><td>A string with 51 characters: “This is a test message with exactly fifty-one chars”.</td></tr>
<tr><td>Steps to Execute</td><td>1. Enter the above message in the message input field.
2. Press send.</td></tr>
</table>

Expected Results	• System alerts the sender that the “message” entered exceeds the maximum length boundary for message input. • Doctor/nurse is forwarded back to the section where they need to re-enter the message.

Test Case 2.9	
Test Case Title	Pager Feature: Pager Request to User Who is Logged Out of The System
Use Case Tested	**PAGE A USER**
Technique Used	**Truth Table**
Pre-Condition	**Doctor/Nurse** is logged into the platform, and a paging request is sent to another **Doctor/Nurse** (i.e. “Doctor A”).
Proof	Truth table would be useful in testing interaction between the availability status of the recipient and request status update. Sample truth table is provided below:

Conditions		
Recipient is logged in to their account (active in the system)	T	F
Recipient is logged out of their account in their device	F	T
Actions		
Send the request	Yes	Yes

	<table><tr><td>Alert the sender that the recipient is currently unavailable</td><td>No</td><td>Yes</td></tr></table>
Input	• Recipient: "Doctor A", • Location: "ER", • Notification Type: "Urgent", • Message: "Need assistance"
Steps to Execute	1. Send a paging notification to "Doctor A". 2. Monitor the system for acknowledgement from "Doctor A". 3. “Doctor A” does not receive the pager notification because he is logged out of his account.
Expected Results	• System notifies the sender that the system failed to send notification to the recipient because of his logged out status. • System asks the sender to choose another recipient. • System logs notification failure status for “Doctor A”.

Test Case 3.1	
Test Case Title	EHR Viewing Feature: Database - Responses
Use Case Tested	**VIEW AN EHR**
Technique Used	**Boundary Value Analysis**
Pre-Condition	**Doctor / Nurse** is logged into the application. **Doctor / Nurse** has accessed the EHR viewing feature.
Proof	Boundary Value Analysis is concerned with evaluating the “behavior of a

<table>
<tr><td></td><td>system at the boundaries of input domains”. To test the EHR feature’s database retrieval capabilities we can use Boundary Value Analysis by preparing the following input:

While the database that the EHR will pull from will depend on each individual hospital, we can test how our system will perform when pulling from a Google Firebase Database (a giant in the industry).

Firebase can handle approximately 100,000 read operations sent by the server per second. (Link)
<table>
<tr><th>Under
(min - 1)</th><th>Normal Load
(min, min+, max, max-)</th><th>Excess Load
(max + 1)</th></tr>
<tr><td colspan="3">_____ read operations per second.</td></tr>
<tr><td>N/A</td><td>1, 2, 100,000, 99,000</td><td>100,001</td></tr>
</table>
</td></tr>
<tr><td>Input</td><td>Conduct 1 read operation per second.
Conduct 2 read operations per second.
Conduct 100,000 read operations per second.
Conduct 99,000 read operations per second.
Conduct 100,001 read operations per second.</td></tr>
<tr><td>Steps to Execute</td><td>1. Prepare a script that executes custom requests.
2. Load the application and sign-in with a dummy account.
3. Open the EHR feature and run the script.</td></tr>
<tr><td>Expected Results</td><td>Under <u>normal load</u>, the EHR feature should be able to properly retrieve data.
Under <u>excess load</u>, the EHR feature should be able to properly retrieve data, albeit with some delay.</td></tr>
</table>

<table>
<tr><th colspan="2">Test Case 3.2</th></tr>
<tr><td>Test Case Title</td><td>EHR Viewing Feature: Accessing Records - Access Levels</td></tr>
<tr><td>Use Case Tested</td><td>VIEW AN EHR</td></tr>
<tr><td>Technique Used</td><td>Equivalence Partitioning</td></tr>
<tr><td>Pre-Condition</td><td>Doctor / Nurse is logged into the application.
Doctor / Nurse has accessed the EHR viewing feature.</td></tr>
<tr><td>Proof</td><td>Equivalence Partitioning involves separating the input “into a finite number of subdomains” where each “subdomain is known as an equivalence class, and it serves as a source of at least one test input”.

To test how our EHR feature handles different access levels we can employ Equivalence Partitioning by taking into account the following guidelines:

Only the healthcare professionals that are treating a patient should have access to their records, in compliance with HIPAA.
<table>
<tr><th>Valid Partition</th><th>Invalid Partition</th></tr>
<tr><td>A user that is authorized . . .</td><td>A user that is not authorized . . .</td></tr>
<tr><td>● Patient’s Doctor
● Patient’s Nurse</td><td>● Not Patient’s Doctor
● Not Patient’s Nurse</td></tr>
</table>
</td></tr>
<tr><td>Input</td><td>We may need to expand, but for now let’s run through a sample scenario:
Doctor A and Nurse A treat Patient A.
Doctor B and Nurse B treat Patient B.</td></tr>
</table>

	Request: Doctor A tries to access Patient A's records. Request: Nurse A tries to access Patient A's records. Request: Doctor B tries to access Patient B's records. Request: Nurse B tries to access Patient B's records. Request: Doctor A tries to access Patient B's records. Request: Nurse A tries to access Patient B's records. Request: Doctor B tries to access Patient A's records. Request: Nurse B tries to access Patient A's records.
Steps to Execute	1. Prepare a script that executes custom requests. 2. Load the application and sign-in with a dummy account. 3. Open the EHR feature and run the script.
Expected Results	If the **user has permission** to access the records, the application should allow the user to see the requested records. If the **user doesn't have permission** to access the records, the application should notify the user and forbid them from seeing them.

Test Case 3.3	
Test Case Title	EHR Viewing Feature: Accessing Records - Type of Data / Time of Day
Use Case Tested	**VIEW AN EHR**
Technique Used	**Pairwise Testing**
Pre-Condition	**Doctor / Nurse** is logged into the application. **Doctor / Nurse** has accessed the EHR viewing feature.
Proof	Pairwise Testing is concerned with generating "test case variants that guarantee 'all-pairs' coverage". To test the performance of our EHR viewing feature at displaying certain kinds of data for different users and at different times of day we can

<table>
<tr>
<td></td>
<td>employ Pairwise Testing by running through the following scenarios:

I first identified the parameters and their potential values.
<table>
<tr><th>User</th><th>Type of Data</th><th>Time of Day</th></tr>
<tr><td>Doctor</td><td>Textual</td><td>Morning</td></tr>
<tr><td>Nurse</td><td>Numerical</td><td>Afternoon</td></tr>
<tr><td></td><td>Photographical</td><td>Night</td></tr>
</table>
I then entered this information in a Pairwise Generator to obtain the following test case variants:
<table>
<tr><th>#</th><th>User</th><th>Type of Data</th><th>Time of Day</th></tr>
<tr><td>1</td><td>Doctor</td><td>Textual</td><td>Morning</td></tr>
<tr><td>2</td><td>Doctor</td><td>Numerical</td><td>Afternoon</td></tr>
<tr><td>3</td><td>Doctor</td><td>Photographical</td><td>Night</td></tr>
<tr><td>4</td><td>Nurse</td><td>Numerical</td><td>Night</td></tr>
<tr><td>5</td><td>Nurse</td><td>Photographical</td><td>Morning</td></tr>
<tr><td>6</td><td>Nurse</td><td>Textual</td><td>Afternoon</td></tr>
<tr><td>7</td><td>Doctor</td><td>Photographical</td><td>Afternoon</td></tr>
<tr><td>8</td><td>Doctor</td><td>Textual</td><td>Night</td></tr>
<tr><td>9</td><td>Doctor</td><td>Numerical</td><td>Morning</td></tr>
</table>
</td>
</tr>
<tr>
<td>Input</td>
<td>Request: [during the Morning] Doctor retrieves an EHR with textual data.
Request: [during the Afternoon] Doctor retrieves an EHR with numerical data.
Request: [during the Night] Doctor retrieves an EHR with photographical data.
Request: [during the Night] Nurse retrieves an EHR with numerical data.
Request: [during the Morning] Nurse retrieves an EHR with photographical data.</td>
</tr>
</table>

	Request: [during the Afternoon] Nurse retrieves an EHR with textual data. Request: [during the Afternoon] Doctor retrieves an EHR with photographical data. Request: [during the Night] Doctor retrieves an EHR with textual data. Request: [during the Morning] Doctor retrieves an EHR with numerical data.
Steps to Execute	1. Prepare a script that executes custom requests. 2. Load the application and sign-in with a dummy account. 3. Open the EHR feature and run the script.
Expected Results	**Irrespective of the kind of data being retrieved, the user requesting it or the time of day**, the system should be able to retrieve the requested EHR correctly.

Test Case 3.4	
Test Case Title	EHR Database Connection Failure
Use Case Tested	**VIEW AN EHR**
Technique Used	**Boundary Value Analysis**
Pre-Condition	**Doctor/Nurse** is logged into the platform.
Proof	Simulating a database connection issue during data retrieval.
Input	Patient name "John Doe".
Steps to Execute	1. Access the records section. 2. Search for "John Doe".
Expected Results	System encounters a database connection issue and displays an appropriate error message.

Test Case 3.5	
Test Case Title	View EHR with Valid Patient Name
Use Case Tested	**VIEW AN EHR**
Technique Used	**Equivalence Partitioning**
Pre-Condition	A valid query to retrieve a specific EHR.

Proof	• Using the patient name as input, this tests the scenario with a valid patient name.
	• Equivalence classes for patient name:
	  ○ One equivalence class for each name in the set of valid patient names {name1},{name2},{name3},...,{namen}.
	  ○ One equivalence class for invalid patient names (i.e. IDs outside the set of valid IDs {name1, name2,...,namen}).

Input	Valid patient ID, e.g., "abc".
Steps to Execute	1. Enter valid patient name "abc". 2. Request EHR.
Expected Results	EHR information for the patient "abc" is displayed correctly.

Test Case 3.6	
Test Case Title	View EHR with Invalid Patient name
Use Case Tested	**VIEW AN EHR**
Technique Used	**Equivalence Partitioning**
Pre-Condition	User is logged in and on the EHR retrieval screen.
Proof	• Using the patient name as input, this tests the scenario with an invalid patient name.

	● Equivalence classes for patient name: ○ One equivalence class for each name in the set of valid patient names {name1},{name2},{name3},...,{namen}. ○ One equivalence class for invalid patient names (i.e. names outside the set of valid names {name1, name2,...,namen}).
Input	Invalid patient name, e.g., "abc".
Steps to Execute	1. Enter invalid patient name "abc". 2. Request EHR.
Expected Results	System alerts "Patient not found".

Test Case 3.7	
Test Case Title	Response to System Extension Scenarios
Use Case Tested	**VIEW AN EHR**
Technique Used	**Truth Table**
Pre-Condition	**Doctor / Nurse** is logged into the application and has accessed the EHR viewing feature.
Proof	Decision Table Testing is used to handle different combinations of inputs and system states effectively. Testing how the system sorts and displays treatment dates verifies that the system is able handle different scenarios correctly and follows the user expectations for data organization.
Input	Access a user's EHR records with different treatment start dates.
Steps to Execute	1. Request to view a list of patients treated by the healthcare professional. 2. System should display the list in order of treatment start date, as

	per system extension scenario.
Expected Results	• The list of patients should be correctly ordered by the treatment start date. • The system should allow the user to easily identify the earliest and the latest treatments.

Test Case 3.8	
Test Case Title	System Performance under Concurrency
Use Case Tested	**VIEW AN EHR**
Technique Used	Load Testing
Pre-Condition	**Doctor / Nurse** is logged into the application and has accessed the EHR viewing feature.
Proof	Load testing evaluates the system's performance under certain levels of user traffic to make sure that the system can handle certain loads without reducing functionality. This test explores what happens when two users try to access the same feature concurrently. **User 1** | **User 2** Access Records. | Access Records.
Input	User 1 and User 2 simultaneously access the EHR feature.
Steps to Execute	1. Simulate two users logging in and accessing the EHR simultaneously. 2. Each user tries to fetch different patient records at the same time.
Expected Results	The system should **maintain operational performance** without

	significant problems, confirming the application's scalability.

Test Case 3.9	
Test Case Title	General Exploratory Testing
Use Case Tested	**VIEW AN EHR**
Technique Used	Exploratory Testing
Pre-Condition	**Doctor / Nurse** has accessed the EHR viewing feature.
Proof	Exploratory testing can be useful in situations where the application has experienced changes or additions (or when its newly built, like in our case). It provides an opportunity for testers to interact with the application in an unscripted manner and effectively uncover unexpected behaviors or bugs that meticulously planned tests might not reveal.
Input	The user interacts with all integrated features including the ability to click and access certain records, and the ability to view records of different patients.
Steps to Execute	1. Doctor / Nurse starts with a review of the features to understand their intended functionality. 2. Doctor / Nurse interacts with all features in a random manner, trying various combinations of actions that a typical user might perform. 3. Tester notes any errors, unexpected results, or challenges encountered during the interactions of the Doctor / Nurse.
Expected Results	● Functional, usability, or interface issues with the implemented features are identified. ● Feedback and comments for improving the existing features are compiled.

<table>
<tr><th colspan="2">Test Case 4.1</th></tr>
<tr><td>Test Case Title</td><td>Reminder Feature: Time Settings - Years</td></tr>
<tr><td>Use Case Tested</td><td>CREATE A REMINDER</td></tr>
<tr><td>Technique Used</td><td>Boundary Value Analysis</td></tr>
<tr><td>Pre-Condition</td><td>Doctor / Nurse is logged into the application.
Doctor / Nurse has accessed the Reminder feature.
Doctor / Nurse has started the creation of a reminder.</td></tr>
<tr><td>Proof</td><td>Boundary Value Analysis is concerned with evaluating the “behavior of a system at the boundaries of input domains”. To test the Reminder feature’s time settings we can use Boundary Value Analysis by preparing the following input:

We are in the year 2024. Let’s test for that year as our base. We can’t test for complete dates since dates are linearly dependent (we need more advanced testing methods).
<table>
<tr><th>Invalid Year
(min - 1)</th><th>Valid
(min, min+, max, max-)</th><th>Invalid Year
(max + 1)</th></tr>
<tr><td colspan="3">The reminder is for the year _____</td></tr>
<tr><td>2023</td><td>2024, 2025, 2038, 2037</td><td>2039</td></tr>
<tr><td>Last Year</td><td>To avoid Y2038 problem.</td><td>Beyond Y2038.</td></tr>
</table>
</td></tr>
<tr><td>Input</td><td>[in the date field] MM/DD/2023
[in the date field] MM/DD/2024
[in the date field] MM/DD/2025
[in the date field] MM/DD/2038
[in the date field] MM/DD/2037
[in the date field] MM/DD/2039</td></tr>
</table>

Steps to Execute	*<u>Preparation:</u> Prepare the rest of the reminder information* 1. Prepare a script that feeds custom input to the application. 2. Load the application and sign-in with a dummy account. 3. Open the reminder feature and run the script.
Expected Results	If the **<u>year is in the past, or conflicts with the Y2038 problem</u>**, the application notifies the user and prevents them from creating the reminder. If the **<u>year is valid</u>**, the application allows the user to create the reminder.

Test Case 4.2	
Test Case Title	Reminder Feature: Recurrence - Frequency Test
Use Case Tested	**CREATE A REMINDER**
Technique Used	**Truth Table**
Pre-Condition	**Doctor / Nurse** is logged into the application. **Doctor / Nurse** has accessed the Reminder feature. **Doctor / Nurse** has started the creation of a reminder.
Proof	A Truth or Decision table displays “a combination of conditions to be met and actions to be taken”. For our Reminder feature to work as intended it must be able to send reminders to the user at the correct times. To test this we can use a Decision Table for different kinds of reminder frequencies:

	Rules					
Conditions	**1**	**2**	**3**	**4**	**5**	**6**
Doctor	T	T	T			
Nurse				T	T	T

<table>
<tr><td></td><td>
<table>
<tr><td>Hourly Recurrence</td><td>T</td><td>F</td><td>F</td><td>T</td><td>F</td><td>F</td></tr>
<tr><td>Daily Recurrence</td><td>F</td><td>T</td><td>F</td><td>F</td><td>T</td><td>F</td></tr>
<tr><td>Weekly Recurrence</td><td>F</td><td>F</td><td>T</td><td>F</td><td>F</td><td>T</td></tr>
<tr><td>Actions</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Hourly Reminder</td><td>✔</td><td></td><td></td><td>✔</td><td></td><td></td></tr>
<tr><td>Daily Reminder</td><td></td><td>✔</td><td></td><td></td><td>✔</td><td></td></tr>
<tr><td>Weekly Reminder</td><td></td><td></td><td>✔</td><td></td><td></td><td>✔</td></tr>
</table>
</td></tr>
<tr><td>Input</td><td>

A Reminder created by a [Doctor] with [Hourly] recurrence.
A Reminder created by a [Doctor] with [Daily] recurrence.
A Reminder created by a [Doctor] with [Weekly] recurrence.
A Reminder created by a [Nurse] with [Hourly] recurrence.
A Reminder created by a [Nurse] with [Daily] recurrence.
A Reminder created by a [Nurse] with [Weekly] recurrence.

</td></tr>
<tr><td>Steps to Execute</td><td>

Prepare a script that feeds custom input to the application.
Load the application and sign-in with a dummy account.
Open the reminder feature and run the script.
Await the reminder notification.

</td></tr>
<tr><td>Expected Results</td><td>If a reminder was <u>set with an hourly recurrence</u>, the reminder is sent every hour.
If a reminder was <u>set with a daily recurrence</u>, the reminder is sent daily.
If a reminder was <u>set with a weekly recurrence</u>, the reminder is sent weekly.</td></tr>
</table>

<table>
<tr><th colspan="2">Test Case 4.3</th></tr>
<tr><td>Test Case Title</td><td>Reminder Feature: Reminding System - Notifications</td></tr>
<tr><td>Use Case Tested</td><td>CREATE A REMINDER</td></tr>
<tr><td>Technique Used</td><td>Boundary Value Analysis</td></tr>
<tr><td>Pre-Condition</td><td>Doctor / Nurse is logged into the application.
Doctor / Nurse has accessed the Reminder feature.
Doctor / Nurse has started the creation of a reminder.</td></tr>
<tr><td>Proof</td><td>Boundary Value Analysis is concerned with evaluating the “behavior of a system at the boundaries of input domains”. To test the Reminder feature’s notification receipts we can use Boundary Value Analysis.

A reminder notification must be delivered within 10 seconds.
<table>
<tr><th>Invalid Time
(min - 1)</th><th>Valid
(min, min+, max, max-)</th><th>Invalid Time
(max + 1)</th></tr>
<tr><td colspan="3">The reminder is received in _____ .</td></tr>
<tr><td>N/A</td><td>1 second, 2 seconds, 10 seconds, 9 seconds</td><td>11 seconds</td></tr>
</table></td></tr>
<tr><td>Input</td><td>A Reminder set with [Hourly] recurrence.
A Reminder set with [Daily] recurrence.
A Reminder set with [Weekly] recurrence.</td></tr>
<tr><td>Steps to Execute</td><td>1. Prepare a script that feeds custom input to the application.
2. Load the application and sign-in with a dummy account.
3. Open the reminder feature and run the script.
4. Await the reminder notification.</td></tr>
<tr><td>Expected Results</td><td><u>Irrespective of the reminder recurrence</u>, the reminder notification must be delivered within 10 seconds of being sent.</td></tr>
</table>

Test Case 4.4	
Test Case Title	Handling Duplicate Reminder Creation
Use Case Tested	**CREATE A REMINDER**
Technique Used	**Error Guessing**
Pre-Condition	The user has access to the reminder system and intends to create a new reminder.
Proof	This test case was generated through error guessing, a technique based on anticipating potential issues that may arise in the system. In this case, the anticipation stems from understanding the nature of reminder systems and the potential consequences of duplicate reminders. The underlying assumption is that the system should maintain a clean and organized record of reminders, without allowing duplicates. Allowing duplicate reminders could lead to confusion for users and clutter the reminder system, making it difficult to manage and prioritize tasks effectively. By considering the logical expectations of how the reminder system should function, it is inferred that the system should have mechanisms in place to prevent the creation of duplicate reminders. Therefore, the test case focuses on verifying whether the system accurately identifies duplicate reminders and provides appropriate options for resolution.
Input	Attempting to create a reminder with the same date/time and title as an existing reminder in the system.
Steps to Execute	1. Open the interface of the reminder system. 2. Select the option to create a new reminder. 3. Input a date/time and title that match an existing reminder in the

	system. 4. Attempt to save the reminder.
Expected Results	The system should validate the input and detect that the reminder is a duplicate of an already existing reminder in the system. The user should be presented with an option to either overwrite the existing reminder or cancel the creation of the duplicate reminder. If the user chooses to overwrite, the system should update the existing reminder with the new information provided. If the user chooses not to create the duplicate reminder, the system should cancel the creation process and return the user to the reminder creation interface.

Test Case 4.5	
Test Case Title	Create Reminder with Future Date
Use Case Tested	**CREATE A REMINDER**
Technique Used	**Equivalence Partitioning**
Pre-Condition	Doctor/Nurse is logged into the platform.
Proof	**Partition 1**: Valid Input - All fields (title, notes, date, time, type, recurrence) are filled correctly and completely. **Partition 2**: Invalid Input - One or more required fields are missing or filled out incorrectly. For example, setting a reminder with a past date, leaving the title blank, or omitting the recurrence information.
Input	Title "Follow-Up", Notes "Check patient progress", Date "Next Month", Time "10:00 AM", Type "Check-up", Recurrence "Weekly"

Steps to Execute	1. Access the reminder section. 2. Fill out all fields with the above information. 3. Add the reminder.
Expected Results	System saves and displays the new reminder among current reminders.

Test Case 4.6	
Test Case Title	Delete an Existing Reminder
Use Case Tested	**CREATE A REMINDER**
Technique Used	**Truth Table**
Pre-Condition	Doctor/Nurse is logged into the platform and reminders exist.
Proof	Truth Table testing in this scenario can be used to verify whether the system correctly handles the deletion of reminders across various conditions or attributes associated with those reminders.
Input	Select an existing reminder.
Steps to Execute	1. Access the reminder section. 2. Select an existing reminder. 3. Choose the option to delete the reminder.
Expected Results	System deletes the selected reminder and updates the list.

Test Case 4.7	
Test Case Title	Update an Existing Reminder
Use Case Tested	**CREATE A REMINDER**
Technique Used	**Equivalence Partitioning**
Pre-Condition	Doctor/Nurse is logged into the platform, and reminders exist.

Proof	Testing update functionality by categorizing updates into different equivalence classes based on input validity.
Input	Select an existing reminder, new date and time
Steps to Execute	1. Access the reminder section. 2. Select an existing reminder. 3. Update the date and time. 4. Save the changes.
Expected Results	The system should correctly update the reminder with new details if the inputs are valid, and it should reject the update or show an error message if the inputs are invalid.

Test Case 4.8	
Test Case Title	Create Reminder with Invalid Details
Use Case Tested	**CREATE A REMINDER**
Technique Used	**Equivalence Partitioning**
Pre-Condition	Doctor/Nurse is logged into the platform.
Proof	Attempting to create a reminder with invalid or incomplete details.
Input	Title "", Type "Meeting", Notes "", Date "Past Date", Time "Invalid Time", Recurrence "Once"
Steps to Execute	1. Access the reminder section. 2. Attempt to add a new reminder with the above details.
Expected Results	System alerts about invalid or incomplete details and prevents reminder

	creation.

Test Case 4.9	
Test Case Title	Create a New Reminder with Incomplete Details
Use Case Tested	**CREATE A REMINDER**
Technique Used	**Pairwise Testing**
Pre-Condition	User is logged in and on the reminder creation screen.
Proof	Since we have 6 input parameters each of which has 2 input options, there are a lot of possible combinations expected for an incomplete input case. All-Pairs Testing efficiently addresses this by generating a test suite that covers all possible pairs of parameter values at least once.

Conditions	Title	Type	Notes	Date	Time	Recurrence
Is it provided?	F	F	T	F	T	T
	F	T	F	T	T	F
	T	T	F	F	T	T
	T	T	T	T	F	F
	T	F	F	T	T	T
	F	T	T	T	F	F
	F	T	F	F	T	T
	F	F	T	T	T	F
	F	T	T	F	F	T
	T	T	F	T	F	T
	T	T	T	F	T	F

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>T</td><td>F</td><td>T</td><td>T</td><td>F</td><td>T</td></tr>
<tr><td></td><td>T</td><td>T</td><td>F</td><td>T</td><td>T</td><td>F</td></tr>
<tr><td></td><td>T</td><td>F</td><td>T</td><td>F</td><td>T</td><td>T</td></tr>
<tr><td>Actions</td><td colspan="6"></td></tr>
<tr><td>Proceed with the request</td><td>No</td><td>No</td><td>No</td><td>No</td><td>No</td><td>No</td></tr>
<tr><td>Display error message</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td></tr>
</table>
</td></tr>
<tr><td>Input</td><td>Title: "Consultation", Type: "Consultation", Notes: "Consult with specialist", Date: "2024-04-30".</td></tr>
<tr><td>Steps to Execute</td><td>1. Input all details except for the time and recurrence.
2. Attempt to add the reminder.</td></tr>
<tr><td>Expected Results</td><td>System displays an error message prompting the user to provide valid details.</td></tr>
</table>

Test Case 5.1	
Test Case Title	AI-Powered Request: API Interaction - Request Limit
Use Case Tested	**PERFORM AN AI-POWERED REQUEST**
Technique Used	**Boundary Value Analysis**
Pre-Condition	**Doctor / Nurse** is logged into the application. **Doctor / Nurse** has accessed the EHR viewing feature.

<table>
<tr><td></td><td>Doctor / Nurse has accessed a specific patient’s EHR.</td></tr>
<tr><td>Proof</td><td>Boundary Value Analysis is concerned with evaluating the “behavior of a system at the boundaries of input domains.” To test our application’s integration of AI and its request limit as it employs the OpenAI API, we can use Boundary Value Analysis by preparing the following input:

<table>
<tr><th>None
(min - 1)</th><th>Within Quota
(min, min+, max, max-)</th><th>Exceeds Quota
(max + 1)</th></tr>
<tr><td colspan="3">_____ requests per day:</td></tr>
<tr><td>0</td><td>1, 2, 200, 199</td><td>201</td></tr>
</table>
</td></tr>
<tr><td>Input</td><td>Number of Requests: 0
Number of Requests: 1
Number of Requests: 200
Number of Requests: 199
Number of Requests: 201</td></tr>
<tr><td>Steps to Execute</td><td>1. Prepare a script that conducts API requests with loaded EHRs.
2. Load the application and sign-in with a dummy account.
3. Open the EHR feature and run the script.</td></tr>
<tr><td>Expected Results</td><td>If the number of requests <u>exceeds the quota</u> imposed by OpenAI, then the application should notify the user, prevent them from querying the API again and should not return a summary.
If the number of requests is <u>within the quota</u> imposed by OpenAI, then the application should correctly perform the query and display the results to the user.</td></tr>
</table>

Test Case 5.2	
Test Case Title	AI-Powered Request: API Performance - Summarizing EHRs

<table>
<tr><td>Use Case Tested</td><td>PERFORM AN AI-POWERED REQUEST</td></tr>
<tr><td>Technique Used</td><td>Pairwise Testing</td></tr>
<tr><td>Pre-Condition</td><td>Doctor / Nurse is logged into the application.
Doctor / Nurse has accessed the EHR viewing feature.
Doctor / Nurse has accessed a specific patient’s EHR.</td></tr>
<tr><td>Proof</td><td>Pairwise Testing is concerned with generating “test case variants that guarantee ‘all-pairs’ coverage”.

To test the performance of the OpenAI API on EHR data we can employ Pairwise Testing by running through the following scenarios:

I first identified the parameters and their potential values.
<table>
<tr><th>Style</th><th>Information</th><th>Summary</th></tr>
<tr><td>“Straightforward”</td><td>Limited</td><td>“Quick”</td></tr>
<tr><td>“Ambiguous”</td><td>Average</td><td>“In-Depth”</td></tr>
<tr><td></td><td>Expansive</td><td></td></tr>
</table>
I then entered this information in a Pairwise Generator to obtain the following test case variants:
<table>
<tr><th>#</th><th>Style</th><th>Information</th><th>Summary</th></tr>
<tr><td>1</td><td>“Straightforward”</td><td>Limited</td><td>“Quick”</td></tr>
<tr><td>2</td><td>“Straightforward”</td><td>Average</td><td>“In-Depth”</td></tr>
<tr><td>3</td><td>“Ambiguous”</td><td>Average</td><td>“Quick”</td></tr>
<tr><td>4</td><td>“Ambiguous”</td><td>Expansive</td><td>“Quick”</td></tr>
<tr><td>5</td><td>“Ambiguous”</td><td>Limited</td><td>“In-Depth”</td></tr>
<tr><td>6</td><td>“Straightforward”</td><td>Expansive</td><td>“In-Depth”</td></tr>
<tr><td>7</td><td>“Straightforward”</td><td>Average</td><td>“Quick”</td></tr>
</table>
</td></tr>
</table>

Input	Quick Summary Option: An EHR with [Straightforward] style and [Limited] information. In-Depth Summary Option: An EHR with [Straightforward] style and [Average] information. Quick Summary Option: An EHR with [Ambiguous] style and [Average] information. Quick Summary Option: An EHR with [Ambiguous] style and [Expansive] information. In-Depth Summary Option: An EHR with [Ambiguous] style and [Limited] information. In-Depth Summary Option: An EHR with [Straightforward] style and [Expansive] information. Quick Summary Option: An EHR with [Straightforward] style and [Average] information.
Steps to Execute	1. Prepare a script that conducts API requests with loaded EHRs. 2. Load the application and sign-in with a dummy account. 3. Open the EHR feature and run the script.
Expected Results	**Irrespective of the EHR's quality**, the API must be able to return an accurate summary *(accuracy as defined by healthcare professionals will need to be included in the testing process)*.

Test Case 5.3	
Test Case Title	AI-Powered Request: - Application Security - Levels of Access
Use Case Tested	**PERFORM AN AI-POWERED REQUEST**
Technique Used	**Truth Table**
Pre-Condition	**The doctor / Nurse** is logged into the application. **The doctor / Nurse** has accessed the EHR viewing feature. **The doctor / Nurse** has accessed a specific patient's EHR.

<table>
<tr><td>Proof</td><td>A Truth or Decision table displays “a combination of conditions to be met and actions to be taken”. For our AI feature to work as intended it must not be able to summarize or simply access data restricted to the user. To test this we can use a Decision Table for different kinds of users and their levels of access.

<table>
<tr><td></td><td colspan="4">Rules</td></tr>
<tr><td>Conditions</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>Doctor</td><td>T</td><td>T</td><td></td><td></td></tr>
<tr><td>Nurse</td><td></td><td></td><td>T</td><td>T</td></tr>
<tr><td>Restricted Access</td><td>T</td><td>F</td><td>T</td><td>F</td></tr>
<tr><td>Unrestricted Access</td><td>F</td><td>T</td><td>F</td><td>T</td></tr>
<tr><td>Actions</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Access Granted</td><td>✔</td><td></td><td>✔</td><td></td></tr>
<tr><td>Access Denied</td><td></td><td>✔</td><td></td><td>✔</td></tr>
</table>
</td></tr>
<tr><td>Input</td><td>

Logged in as a [Doctor] attempt to summarize a patient’s file which has [Restricted Access].
Logged in as a [Doctor] attempt to summarize a patient’s file which has [Unrestricted Access].
Logged in as a [Nurse] attempt to summarize a patient’s file which has [Restricted Access].
Logged in as a [Nurse] attempt to summarize a patient’s file which has [Unrestricted Access].

</td></tr>
<tr><td>Steps to Execute</td><td>

Prepare a script that conducts API requests with loaded EHRs.
Load the application and sign-in with a dummy account.

</td></tr>
</table>

	3. Open the EHR feature and run the script.
Expected Results	If the **user has restricted access** to a patient's file, the application should not allow the user to summarize or access the file. If the **user has unrestricted access** to a patient's file, the application should allow the user to both summarize and access the file.

Test Case 5.4	
Test Case Title	Simultaneous AI Query and Record Access Test
Use Case Tested	**PERFORM AN AI-POWERED REQUEST**
Technique Used	**Boundary Value Analysis**
Pre-Condition	Multiple users (doctors/nurses) are logged into the platform.
Proof	Testing how the system handles simultaneous operations of different users performing typical tasks.
Input	● User 1 performs an AI query "Summary of patient's last visit" ● User 2 accesses a different patient's records
Steps to Execute	1. Coordinate multiple users to log into the system simultaneously. 2. User 1 accesses the records section, selects a patient, and inputs the AI query "Summary of patient's last visit." 3. Simultaneously, User 2 accesses the records section and selects a different patient to view their medical history. 4. All users review the system's responses and changes made to assess accuracy and performance. 5. Review the AI-generated results.
Expected Results	The system should handle all requests accurately without delays or errors. The AI provides an accurate summary for User 1.

	User 2 sees the correct records.

Test Case 5.5	
Test Case Title	Handle AI Service Downtime
Use Case Tested	**PERFORM AN AI-POWERED REQUEST**
Technique Used	**Equivalence Partitioning**
Pre-Condition	**Doctor / Nurse** is logged into the platform, and AI service is down.
Proof	• **Partition 1: AI Service Available -** The system should process AI queries as normal, returning accurate responses based on the AI's analysis. • **Partition 2: AI Service Unavailable -** The system should recognize the service downtime and appropriately handle this by informing the user of the unavailability.
Input	Any AI query.
Steps to Execute	1. Access the records section. 2. Select a patient. 3. Input an AI query.
Expected Results	System displays an error message indicating the unavailability of the AI service.

Test Case 5.6	
Test Case Title	Valid AI Query
Use Case Tested	**PERFORM AN AI-POWERED REQUEST**

Technique Used	**Equivalence Partitioning**
Pre-Condition	User is logged in, has selected a patient's EHR, and the AI service is operational.

Proof	• Using the EHR query as input, this tests the scenario with a valid EHR query.
	• One equivalence class for each EHR query in the set of valid EHR queries {Query1}, {Query2}, {Query3},..., {Query1n} which are predefined valid queries that the system is expected to process correctly. Examples include:
	  ◦ "Summarize patient history"
	• One equivalence class for invalid EHR queries that are not recognized or supported by the system (i.e. queries outside the set {Query1, Query2, Query3,..., Query1n}). Examples could include:
	  ◦ "Calculate future health risks"

Input	AI query for "summarize patient history".
Steps to Execute	1. Enter the AI query "summarize patient history". 2. Submit the query for processing.
Expected Results	AI processes the query and returns a concise summary of the patient's history.

Test Case 5.7	
Test Case Title	Predefined Button Functionality Check
Use Case Tested	**PERFORM AN AI-POWERED REQUEST**
Technique Used	**Equivalence Partitioning**
Pre-Condition	Doctor/Nurse is logged into the application and has accessed a specific patient’s EHR.

Proof	Using equivalence partitioning to test each predefined button to ensure they are mapped to the correct queries and produce expected outputs. The partitions are: functional buttons and non-responsive buttons.
Input	Click each predefined button (assuming names like "Summarize Patient History", "Latest Lab Results", "Medication List").
Steps to Execute	1. Log into the application using a dummy account. 2. Access the EHR feature. 3. Click each predefined query button sequentially and observe the response.
Expected Results	Each button should trigger the corresponding AI query without error. The results should match expected outputs for each query type, showing correct data from the patient's EHR.

Test Case 5.8	
Test Case Title	AI Response Time for Predefined Queries
Use Case Tested	**PERFORM AN AI-POWERED REQUEST**
Technique Used	**Boundary Value Analysis**
Pre-Condition	High load conditions simulated with multiple users logged in and making simultaneous requests.
Proof	Testing system response at the boundaries of normal operational capacity to ensure it can handle peak loads.
Input	Simultaneous use of predefined query buttons by multiple users (e.g., 50 users clicking "Summarize Patient History" at the same time).
Steps to Execute	1. Coordinate multiple users to log into the system simultaneously. 2. Direct each user to access the EHR section and use the same predefined button at exactly the same time.

	3. Monitor and record the response time for each request.
Expected Results	The system should manage concurrent queries efficiently, returning responses within a predetermined acceptable timeframe without server errors or significant delays.

Test Case 5.9	
Test Case Title	Error Handling and Recovery for AI Requests
Use Case Tested	**PERFORM AN AI-POWERED REQUEST**
Technique Used	**Boundary Value Analysis**
Pre-Condition	Doctor/Nurse is logged into the application, has accessed a specific patient's EHR, and is in a situation where network fluctuations or API errors might occur.
Proof	Boundary value analysis is used here to assess how the system handles edge cases such as minimal network connectivity or when the AI service API fails or returns an error. Testing how the system copes with these issues is critical for ensuring reliability and user satisfaction.
Input	● Simulate minimal network connectivity while pressing a predefined button like "Summarize Patient History." ● Simulate an API failure scenario when another predefined button like "Medication List" is used.
Steps to Execute	1. Prepare to simulate network issues and API failure scenarios either through software tools or network configuration adjustments. 2. Log into the application using a dummy account. 3. Execute the predefined queries under these simulated conditions.

	4. Observe how the application responds, noting any error messages displayed and the recovery options offered to the user.
Expected Results	• When network connectivity is poor, the application should display a clear and informative error message, possibly suggesting the user to try again later or check their connection. • In case of an API failure, the system should handle the error gracefully without crashing, providing feedback to the user about the issue and steps for recovery. • The application should maintain its stability and functionality, allowing the user to attempt the request again or perform other actions without needing to restart or experience further errors.

5.2 Traceability Matrix

Use Case 1: Messaging	Use Case 2: Paging	Use Case 3: EHR Viewing	Use Case 4: Reminders	Use Case 5: AI Integration
Test Case 1.1	Test Case 2.1	Test Case 3.1	Test Case 4.1	Test Case 5.1
Test Case 1.2	Test Case 2.2	Test Case 3.2	Test Case 4.2	Test Case 5.2
Test Case 1.3	Test Case 2.3	Test Case 3.3	Test Case 4.3	Test Case 5.3
Test Case 1.4	Test Case 2.4	Test Case 3.4	Test Case 4.4	Test Case 5.4
Test Case 1.5	Test Case 2.5	Test Case 3.5	Test Case 4.5	Test Case 5.5
Test Case 1.6	Test Case 2.6	Test Case 3.6	Test Case 4.6	Test Case 5.6
Test Case 1.7	Test Case 2.7	Test Case 3.7	Test Case 4.7	Test Case 5.7
Test Case 1.8	Test Case 2.8	Test Case 3.8	Test Case 4.8	Test Case 5.8
Test Case 1.9	Test Case 2.9	Test Case 3.9	Test Case 4.9	Test Case 5.9

Section 6: Testing Results

Use Case	Test Case	Outcome	Notes
Use Case 1: Messaging	Test Case 1.1	Passed	Messages are stored in the database when the internet is off and are successfully delivered to the recipient once the internet connection is restored.
	Test Case 1.2	Passed	The message "hiii23132…????///,,,<<===++__" was delivered successfully.
	Test Case 1.3	Passed	Testing for all different times of the day was not possible.
	Test Case 1.4	Passed	System does not allow sending messages when it is greater than the fixed length of message.
	Test Case 1.5	Passed	System does not allow sending empty messages.
	Test Case 1.6	Passed	Newly registered users are promptly added to the messaging directory and can receive messages normally.
	Test Case 1.7	Passed	Users removed from the directory are immediately eliminated from the list of possible message recipients, even if a message is sent during the removal process.
	Test Case 1.8	Passed	Messages are delivered successfully without issues.
	Test Case 1.9	Passed	Messages are delivered almost instantaneously.

Use Case	Test Case	Result	Notes
Use Case 2: Paging	Test Case 2.1	Passed	The application handles correct and incorrect messages for paging
	Test Case 2.2	Passed	The user can select the rooms from

			the available roms mentioned in the system
	Test Case 2.3	Passed	The user can specify the urgency of the notification while creating the paging request
	Test Case 2.4	Passed	The system doesn't allow users to send notification without filling the values in the respective fields
	Test Case 2.5	Failed	Cannot test this test cases in our current phase of implementation of the app.
	Test Case 2.6	Passed	The user is able to send the notification after entering details into all the required fields
	Test Case 2.7	Passed	Yes the user wont be able to send the notification without entering the message or the reason for sending.
	Test Case 2.8	Passed	The system checks whether the message is within the message limit specified in our app
	Test Case 2.9	Failed	Cannot test this test cases in our current phase of implementation of the app.

Use Case	Test Case	Result	Notes
Use Case 3: EHR Viewing	Test Case 3.1	Partially	We tested this test case with a max of 4 requests and were able to retrieve data at the same time.
	Test Case 3.2	Passed	
	Test Case 3.3	Passed	Our implementation of EHR data records only handles data for textual and numerical containing data.
	Test Case 3.4	Passed	Our App handles this by showing an

			error page in showing the EHR record.
	Test Case 3.5	Passed	Our App displays all the names of the patient that the doctor has access to.
	Test Case 3.6	Passed	Our App only displays names of the valid patients.
	Test Case 3.7	Issue	This is out of the scope of our use case and hence our implementation has not passed this test case.
	Test Case 3.8	Passed	Simultaneous viewing of EHR is possible through our app.
	Test Case 3.9	Passed	Yes, the system is user friendly for the users to understand and test the features by themselves.

Use Case	Test Case	Result	Notes
Use Case 4: Reminders	Test Case 4.1	Passed	Since the date format is not manually entered but rather relies on an international dependency and the built-in Flutter date and time selector, date formatting does not pose any issues.
	Test Case 4.2	Issue	Recurrence functionality has not yet been implemented.
	Test Case 4.3	Passed	Notifications appear immediately as expected.
	Test Case 4.4	Passed	The system checks for duplicate entries when new reminders are added and prevents duplicate creations.
	Test Case 4.5	Passed	Reminders set for future dates are successfully created without issues.

	Test Case 4.6	Passed	Reminders are successfully removed from both the list and the database.
	Test Case 4.7	Passed	Reminders are updated immediately after modifications in the UI, with synchronous changes reflected in the database.
	Test Case 4.8	Passed	The system does not allow the creation of reminders set for past times, ensuring data integrity.
	Test Case 4.9	Passed	Title and time are mandatory fields for reminder creation. The system defaults to 12:00 AM for any selected date. For reminders set on the current day, the earliest possible time is set to the current time to prevent setting reminders in the past.

Use Case	Test Case	Result	Notes
Use Case 5: AI Integration	Test Case 5.1	Partially	Due to financial constraints related to making API calls to the AI, we haven't fully tested this aspect because we are unable to reach the limit of allowed API calls.
	Test Case 5.2	Passed	We are successfully able to make API calls and retrieve meaningful summaries from the AI.
	Test Case 5.3	Passed	Our app is specifically designed so that only doctors and nurses with access to an EHR record can view and query it using the AI.
	Test Case 5.4	Passed	The application supports multiple users making AI queries simultaneously without issues.
	Test Case 5.5	Passed	We have tested the scenario with incorrect API tokens to ensure robust

			error handling within the code structure.
	Test Case 5.6	Passed	The app is designed to provide predefined AI queries, and it successfully passes this test case.
	Test Case 5.7	Passed	The query buttons are functioning well and meet the requirements of the test cases.
	Test Case 5.8	Passed	The response time is minimal due to the AI's capability to efficiently handle requests from multiple users.
	Test Case 5.9	Passed	This case is managed effectively by displaying an error message when issues arise.

Section 7: Recommendation on Software Quality

1. **Enhanced Messaging Features:**
    a. **Read Receipts:** Integrate read receipts that show when a message is delivered and read by the recipient. This feature will enhance communication transparency and urgency in a healthcare setting.
    b. **Message Forwarding:** Add a feature that allows users to forward messages to other users within the app. This can be particularly useful for sharing patient updates quickly without the need to retype or take screenshots.
2. **EHR Usability Improvements:**
    a. **Sorting Capability:** Implement functionalities to sort EHRs based on patient names or birth dates. This would help healthcare providers find patient records more efficiently, improving workflow and patient care.
3. **Advanced Reminder Settings:**
    a. **Recurring Reminders:** Incorporate options for setting reminders that recur at specified intervals, such as daily, weekly, or monthly. This feature would be highly beneficial for managing recurring tasks and medication schedules.

These enhancements are aimed at improving the user experience, increasing the efficiency of medical staff, and ensuring critical health information is communicated effectively and promptly within your application.