# Document 6- Please prepare a use case diagram, activity diagram and a use case specification document.

### Use Case Diagram 1. Login Function



### Use Case Diagram 2: Search Doctor Function

###

### Activity Diagram : User Registration

###

# Use case specification document.

**Use Case 1: Employee Login**

| **Parameter** | **Details** |
| --- | --- |
| **Use Case Name** | Employee Login |
| **Use Case Description** | This use case allows medical representatives to securely log in to the app using their credentials to access the app's functionalities. |
| **Actors** | **Primary Actor**: Medical Representative**Secondary Actor**: Admin (for managing accounts and permissions) |
| **Basic Flow** | 1. Open the app. 2. Enter username and password. 3. Click "Login". 4. If credentials are correct, access is granted and the dashboard is shown. |
| **Alternate Flow** | **A1**: If the login fails, an error message is displayed, prompting the user to retry. |
| **Exceptional Flows** | **E1**: App fails to load due to connectivity issues. **E2**: Incorrect credentials or user not found. |
| **Pre-Conditions** | - The medical representative has valid login credentials. - The app is installed on the device. |
| **Post-Conditions** | - The representative successfully logs in and is redirected to the dashboard. |
| **Assumptions** | - The user has a registered account with valid credentials. - The representative has internet access for verification (unless offline login is supported). |
| **Constraints** | - Login can only occur when credentials are correct. - The app requires an active internet connection for authentication. |
| **Dependencies** | - User authentication service. - Centralized user management system. |
| **Inputs and Outputs** | **Inputs**: Username, Password. **Outputs**: Authentication response (success or failure). |
| **Business Rules** | - A representative can log in only if they have a valid account. - Multiple failed login attempts may result in a temporary account lockout. |
| **Miscellaneous Info** | - The login screen should be user-friendly with options for "Forgot Password" and "Help". |

**Use Case 2: Doctor Search**

| **Parameter** | **Details** |
| --- | --- |
| **Use Case Name** | Doctor Search |
| **Use Case Description** | This use case allows medical representatives to search for doctors by name, location, or specialty to view their details and schedule visits. |
| **Actors** | **Primary Actor**: Medical Representative **Secondary Actor**: Admin (for managing doctor data) |
| **Basic Flow** | 1. Open the app and log in. 2. Navigate to the "Doctor Search" section. 3. Enter search criteria (name, specialty, location). 4. View the search results with doctor details. 5. Select a doctor for scheduling a visit or logging feedback. |
| **Alternate Flow** | **A1**: If no results are found, show a message: "No doctors found matching the criteria". **A2**: Search results can be filtered by multiple criteria (e.g., location + specialty). |
| **Exceptional Flows** | **E1**: Search fails due to connection issues. **E2**: The system cannot retrieve doctor data due to an error in the backend. |
| **Pre-Conditions** | - The medical representative is logged into the app. - The app has access to the doctor database. |
| **Post-Conditions** | - The representative can view and select a doctor for scheduling or further interaction. |
| **Assumptions** | - The doctor data is regularly updated in the system. - The representative is familiar with the doctor search feature. |
| **Constraints** | - Doctor data may be incomplete or outdated. - Search functionality depends on a central database. |
| **Dependencies** | - Doctor data stored in the centralized database. - Search functionality relies on connectivity. |
| **Inputs and Outputs** | **Inputs**: Doctor name, location, specialty. **Outputs**: Search results (doctor list with details). |
| **Business Rules** | - Doctor data must be updated regularly to reflect the correct information. - Search results must only show doctors relevant to the search criteria. |
| **Miscellaneous Info** | - The search feature should support auto-completion or suggestions for easier user input. |

**Use Case 3: Call Logging**

| **Parameter** | **Details** |
| --- | --- |
| **Use Case Name** | Call Logging |
| **Use Case Description** | This use case allows the medical representative to log calls made to doctors, including call duration and purpose. |
| **Actors** | **Primary Actor**: Medical Representative **Secondary Actor**: Admin (for viewing reports) |
| **Basic Flow** | 1. Open the app and log in. 2. Go to the "Call Logging" section. 3. Select the doctor or enter new details. 4. Log call duration, purpose, and notes. 5. Save the entry. |
| **Alternate Flow** | **A1**: If the doctor is not found, the representative can enter the call details manually. **A2**: Calls can be edited if they are logged incorrectly. |
| **Exceptional Flows** | **E1**: Call logging fails due to a network issue. **E2**: The app crashes while logging the call. |
| **Pre-Conditions** | - The representative is logged into the app. - The representative has made a call to a doctor. |
| **Post-Conditions** | - The call is logged in the system with all relevant details (duration, purpose, doctor name). |
| **Assumptions** | - The representative has made a call to a doctor. - The app allows call logging with date and time accuracy. |
| **Constraints** | - The app must ensure that call logs are correctly saved and stored in the system. - Call logs may not sync immediately without connectivity. |
| **Dependencies** | - Call logs are synced to the central database. - App needs internet for saving logs. |
| **Inputs and Outputs** | **Inputs**: Call duration, purpose, doctor name. **Outputs**: Logged call details (duration, purpose, notes). |
| **Business Rules** | - Call logs must be saved with all required fields (doctor, duration, notes). - Call logging should be completed immediately after the call. |
| **Miscellaneous Info** | - Call logging should have a quick entry format to facilitate rapid documentation. |

**Use Case 4: Product Discussion Log**

| **Parameter** | **Details** |
| --- | --- |
| **Use Case Name** | Product Discussion Log |
| **Use Case Description** | This use case enables the medical representative to log product discussions with doctors, including product information shared, feedback, and interest level. |
| **Actors** | **Primary Actor**: Medical Representative **Secondary Actor**: Doctor, Admin |
| **Basic Flow** | 1. Open the app and log in. 2. Select the doctor or visit entry. 3. Enter the product discussed. 4. Record the doctor's feedback and level of interest. 5. Save the log. |
| **Alternate Flow** | **A1**: If the product discussed is not listed, the representative can manually enter the product name. **A2**: If feedback is incomplete, representative can return to update it later. |
| **Exceptional Flows** | **E1**: Product details fail to load from the system. **E2**: The discussion log fails to save due to network connectivity issues. |
| **Pre-Conditions** | - The representative is logged into the app. - The representative has met with the doctor and discussed a product. |
| **Post-Conditions** | - The product discussion and feedback are saved in the system. - The log is available for future reference and reporting. |
| **Assumptions** | - The representative has access to product information. - The app allows easy entry of discussion points and feedback. |
| **Constraints** | - Limited to logging discussions with doctors who are already in the app's database. - Discussions may not be logged without product information. |
| **Dependencies** | - Product data available in the app. - Syncing capability with the central system to log and track product feedback. |
| **Inputs and Outputs** | **Inputs**: Product name, discussion details, doctor feedback. **Outputs**: Saved product discussion logs. |
| **Business Rules** | - Product discussion logs should be updated in real-time. - Feedback must be detailed and entered as soon as possible after the interaction. |
| Miscellaneous Info | Product discussions should be searchable by product name or doctor name for future reference. |

**Use Case 5: Visit Duration Tracking**

| **Parameter** | **Details** |
| --- | --- |
| **Use Case Name** | Visit Duration Tracking |
| **Use Case Description** | This use case tracks the total time spent by the medical representative during each doctor visit, helping with performance analysis and reporting. |
| **Actors** | **Primary Actor**: Medical Representative **Secondary Actor**: Admin (for report generation) |
| **Basic Flow** | 1. Open the app and log in. 2. Select the doctor and start the visit timer. 3. During the visit, the timer runs and tracks the duration. 4. Stop the timer at the end of the visit. 5. Save the visit duration. |
| **Alternate Flow** | **A1**: The visit timer can be paused and resumed if the representative needs to step away. **A2**: In case of app crash, the visit time is saved when resumed. |
| **Exceptional Flows** | **E1**: Timer fails to start or stop due to app malfunction. **E2**: Duration is not recorded due to connectivity loss (if cloud syncing is required). |
| **Pre-Conditions** | - The representative has started a visit. - The app is running and has access to the device's timer functionality. |
| **Post-Conditions** | - The visit duration is logged and available for reporting. - The data syncs with the backend if necessary. |
| **Assumptions** | - The representative will manually start and stop the timer. - The app has access to the device's time-tracking feature. |
| **Constraints** | - Timer functionality requires active device resources. - Duration tracking may not be precise if the app is not properly synced. |
| **Dependencies** | - Device clock and time tracking functionality. - Centralized system for storing visit logs. |
| **Inputs and Outputs** | **Inputs**: Start and stop visit time. **Outputs**: Recorded visit duration. |
| **Business Rules** | - The visit duration should be accurately tracked from the time the visit starts until the representative marks the visit complete. |
| **Miscellaneous Info** | - The app should allow for manual editing of visit durations in case of errors or exceptions. |

## Non-Functional Requirements

* **Security**: The app must be secured with encryption and multi-factor authentication for sensitive data (e.g., doctor details, feedback).
* **Performance**: The app must load quickly, even with a large number of records.
* **Usability**: The app must have a simple and intuitive interface suitable for field use.
* **Compatibility**: The app should be compatible with both Android and iOS platforms.
* **Notifications**: The app must support push notifications for visit reminders and follow-ups.
* **Offline Functionality**: The app should support offline functionality, allowing representatives to continue working without a constant internet connection, syncing data when the connection is restored.

## Document 7- Screens and pages

|  |  |
| --- | --- |
| Home Page | Login Page |
| New Mockup 1.png | New Mockup 2.png |

|  |  |
| --- | --- |
| My HQ Dashboard | Select Doctor |
| New Mockup 3.png | New Mockup 4.png |

## Doctor Profile

## New Mockup 5.png

# Q8. Write a paragraph on your experience using Visio and Axure for the project.

* Visio is ideal for users who need to create diagrams and flowcharts for technical, business, or system documentation. It’s a great tool for process mapping and other technical visualizations.
* Axure, on the other hand, is a more specialized tool for UX/UI design and interactive prototyping, allowing designers to create high-fidelity, clickable prototypes that simulate real user experiences and flows.

During the project, I had the opportunity to use **Visio** and **Axure** for creating detailed **UI mockups**, **diagrams**, and **interactive prototypes**, which significantly enhanced the design and planning process.

**Visio** was particularly useful for creating **flowcharts** and **use case diagrams**, providing a clear visual representation of the system's workflows and user interactions. I appreciated Visio’s versatility in drafting complex diagrams, allowing me to quickly iterate and adjust the structure. Its intuitive interface, with drag-and-drop functionality, made it easy to organize different components and maintain clarity throughout the design process.

On the other hand, **Axure** was invaluable for building interactive, high-fidelity **prototypes**. It enabled me to not only design static UI screens but also add interactive elements like **clickable buttons**, **hover effects**, and **form validations**. This helped in visualizing the user experience and allowed stakeholders to interact with the prototype early in the development phase. Axure's ability to create dynamic interactions and its flexible design tools made it easy to iterate on user flows and gather feedback quickly.

Together, Visio and Axure streamlined communication between the design and development teams, ensuring that the functional requirements were met while delivering a more refined user experience. Both tools provided a comprehensive environment for designing and testing the app’s interface and functionality before implementation, saving time and resources.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Visio** | **Axure** |
| **Primary Purpose** | **Diagramming and Flowcharts**: Used for creating detailed **flowcharts**, **network diagrams**, **organizational charts**, and **system architectures**. | **Prototyping and Interaction Design**: Primarily used for creating **interactive wireframes**, **high-fidelity prototypes**, and **user interface designs**. |
| **User Interface Design** | Focuses on static visual representations and **basic layouts** without interactive elements. | Allows for dynamic, **interactive design**, where users can click, hover, and simulate user flows. |
| **Interactivity** | Limited interactivity (such as hyperlinks or basic actions). Primarily for **static diagrams**. | Full interactivity with **clickable elements**, **dynamic content**, **animations**, and **conditional logic**. |
| **Use Cases** | **Business Process Mapping**, **Flowcharts**, **Network Diagrams**, **Organizational Charts**, and **Technical Schematics**. | **UX/UI Design**, **Wireframing**, **Prototyping**, **User Testing**, and **Interactive Prototypes**. |
| **Learning Curve** | Relatively simple and quick to learn for creating basic diagrams, as it’s mostly drag-and-drop for shapes and connectors. | Steeper learning curve due to its focus on creating interactive and dynamic prototypes, but more powerful for **user experience design**. |
| **Design Fidelity** | Primarily low-fidelity; mostly focuses on schematic diagrams and layout planning rather than detailed visuals. | High-fidelity design tool for creating detailed, interactive UI elements with design consistency. |
| **Collaboration Features** | Basic collaboration through shared files and comments in a **Microsoft ecosystem**. | Advanced collaboration options, including **real-time collaboration**, **version control**, and **feedback tools** directly within the prototype. |
| **Integration with Other Tools** | Integrates well with **Microsoft Office** and other **Visio-based tools**, but limited integration with UX design tools. | Integrates with tools like **Sketch**, **Photoshop**, and other UX design and collaboration platforms. |
| **Output Format** | **Diagrams**, **PDFs**, **Visio files**, and **images**. Suitable for presentations and technical documentation. | **Interactive prototypes** that can be shared and tested in a browser or exported to various formats like **HTML**, **PDF**, or **Axure files** for review. |
| **Cost** | More affordable for basic diagramming needs, with licensing available through **Microsoft Office**. | Axure typically has a higher cost, but it’s designed specifically for UX/UI prototyping, which makes it a more specialized tool for designers. |

# Document 9- BA experience

My experience as BA in following phases:

## Requirement gathering:

**Requirement Gathering**

To ensure we captured the correct and comprehensive requirements for the doctor application, I utilized the **MOSCOW technique** (Must have, Should have, Could have, Won’t have). This allowed us to prioritize the needs and functionalities critical for medical representatives' daily activities.

**Examples:**

* **Must Have:**
	1. **Login & Authentication:** The application should allow medical representatives to securely log in with two-factor authentication (2FA) for confidentiality.
	2. **Doctor Database Search:** Medical representatives must be able to search and filter doctors by specialty, location, and contact information in real-time.
* **Should Have:**
	1. **Appointment Scheduling:** It should allow reps to schedule meetings with doctors and receive reminders for upcoming appointments.
	2. **Prescription History:** The app should provide a view of the doctor’s past prescription history (e.g., drugs prescribed, quantity, frequency).
* **Could Have:**
	1. **Push Notifications:** The app could notify the rep of any updates or changes in the doctor's availability or appointment.
	2. **Sales Reports:** The app could include basic analytics features showing sales performance metrics for the rep.
* **Won’t Have:**
	1. **Real-time video consultation integration:** This feature would be out of scope for the initial release as the focus is on scheduling and information access.

**Client Availability Challenges**

During this phase, our primary client contact was unavailable for a certain period, creating a gap in direct communication. As the BA, I had to identify alternative points of contact within the client organization to continue gathering information without delays.

**Examples:**

* **Information for Medical Representative Workflow:**
While the client was unavailable, I worked closely with the project manager from the client side to gather data on typical daily activities of medical reps (e.g., how often reps need to update doctor profiles, preferred notification channels, etc.).
* **Data Security Requirements:**
Since the client’s main point of contact was unavailable, I sourced a security expert from the client side to discuss the data protection and privacy regulations that would apply to the app, such as HIPAA compliance for patient data.
* **Feedback from Sales Managers:**
I reached out to regional sales managers to get feedback on pain points that reps experience in the field, such as the lack of a unified doctor database and difficulties in tracking doctor engagement.

**Requirement Validation**

To ensure that the gathered requirements were clear, accurate, and aligned with the end goals of the project, I applied the **FURPS** technique (Functionality, Usability, Reliability, Performance, and Supportability).

**Examples:**

* **Functionality:**
	+ The app should support real-time syncing of doctor details, meaning any update made by a rep is reflected immediately on all devices.
	+ The application should allow reps to generate PDF versions of doctor visit reports with the click of a button.
* **Usability:**
	+ The interface should be simple and intuitive, with easy navigation to avoid time lost in learning the system.
	+ Medical reps should be able to filter doctor lists by specialty and proximity with a single tap on the screen.
* **Reliability:**
	+ The application should be able to handle at least 10,000 simultaneous users without crashing or slowdowns.
	+ The system should guarantee 99.9% uptime, ensuring that critical functions like doctor database access and appointment scheduling remain uninterrupted.
* **Performance:**
	+ The application should load data within 2 seconds, even with a large database of doctors, to enhance the user experience.
	+ Responses from the backend (e.g., search results, doctor data) should be processed within 1 second.
* **Supportability:**
	+ The system should include an integrated help section with FAQs, troubleshooting guides, and a way for reps to directly contact support if necessary.
	+ The application should be able to support future updates without requiring complete system overhauls (e.g., adding new doctor specialties, integrating new reporting tools).

**Identifying and Removing Redundancies**

During the requirement gathering process, we identified duplicated or repeated requirements that could lead to inefficiency in development.

**Examples:**

* **Duplicate Doctor Search Features:**
	+ Initially, there were two separate requirements that asked for similar doctor search functionalities: one for searching by name and one for searching by specialty. These were combined into a single, comprehensive search feature with multiple filters (name, specialty, location, etc.).
* **Appointment Reminder:**
	+ One requirement stated that the app should send notifications for upcoming appointments, while another requirement mentioned that the app should include a calendar with reminders. These were consolidated into a single requirement for appointment reminders with a calendar integration.
* **Sales Report Features:**
	+ There were multiple mentions of sales performance reports, some focused on the overall sales performance and others on individual rep metrics. These were unified into a single requirement for a sales performance dashboard, which can display both individual and team-level statistics.

**Prototyping for Clarity**

To refine and specify the requirements further, I used **prototyping** to build wireframes and mock-ups of the application, which helped visualize key features and provided an opportunity for early feedback.

**Examples:**

* **Doctor Profile Screen:**
	+ I developed a prototype for the doctor profile page, showcasing the doctor’s contact details, prescribing history, and a scheduling feature. This allowed stakeholders to see the design and provide feedback on the interface before full development began.
* **Appointment Booking Flow:**
	+ I created a mock-up of the appointment booking flow, allowing medical reps to select a date and time, send a request to the doctor, and receive confirmations. This visual representation helped clarify user expectations for booking features and highlighted potential usability issues, such as unclear button labels or redundant steps.
* **Sales Report Dashboard:**
	+ A prototype of the sales performance dashboard was created, where reps could view their sales numbers, product distribution, and visit success rates. Stakeholders provided feedback on the layout and which metrics were most important to display, ensuring that the final product would be both useful and easy to navigate.

## Requirement Analysis:

During the **Requirement Analysis** phase of the Doctor Reporting App, I translated business requirements into clear specifications using various diagrams and documents, ensuring effective communication with stakeholders and the development team.

### 1. ****UML Diagrams:****

I created **UML diagrams** to visually represent the system requirements:

1. **Use Case Diagram:** Illustrated interactions like "Login," "Search Doctor," and "Generate Report" to define key user actions.
2. **Class Diagram:** Mapped data structures such as "Doctor" and "Appointment" to help the development team understand relationships.
3. **Sequence Diagram:** Showed the flow of actions in generating daily reports, highlighting interactions between the system and users.

**2. Activity Diagrams:**

**I used Activity Diagrams to describe key processes:**

* **Appointment Scheduling Flow:** Visualized the steps from selecting a doctor to receiving an appointment confirmation, identifying decision points.
* **Daily Reporting Process:** Mapped out the process of logging visits, adding prescriptions, and generating a report to streamline the user experience.

**3. Communicating Diagrams:**

I presented these diagrams to the team, adjusting them based on feedback:

* Presented diagrams to the team for feedback. For example, updated the **Use Case Diagram** to include “Cancel Appointment” based on developer input, and adjusted the **Class Diagram** to account for doctor specializations after a team discussion.

**4. Documentation (BRS & SRS):**

I prepared the **Business Requirements Specification (BRS)** and **Software Requirements Specification (SRS)**:

* **BRS:** Outlined the business goals, such as improving reporting efficiency and appointment scheduling.
* **SRS:** Detailed system functionalities like doctor data management, report generation, and performance requirements.

**Summary:**

I utilized **UML and Activity Diagrams** to define system requirements and processes, communicated effectively with the team to refine them, and created the **BRS** and **SRS** to ensure the development aligned with both business and technical needs.

## 3. Design:

**BA Experience in Design for Daily Doctor Reporting App:**

During the **Design** phase of the Doctor Reporting App, my role focused on ensuring that all requirements were accurately translated into testable components and that the design and solution were aligned with client expectations.

**1. Preparing Test Cases from Use Case Diagrams:**

Using the **Use Case Diagrams** as a foundation, I prepared **test cases** to validate system functionality. These test cases covered both positive and negative scenarios to ensure comprehensive testing.

* **Positive Test Cases:** For example, testing if a medical representative can successfully log in, search for a doctor, and generate a report.
* **Negative Test Cases:** Ensured that invalid logins, incorrect search criteria, or missing data inputs would result in appropriate error messages or system behavior.

**2. Communicating with the Client:**

I regularly communicated with the client to review the **design and solution documents** to ensure alignment with their expectations. I gathered feedback and made necessary adjustments to the design based on their input, ensuring the solution met their business needs.

**3. Comprehensive Test Case Coverage:**

I ensured that no test case was overlooked; understanding that missing even a single test case could have a significant impact later in the development cycle. For example, failing to test edge cases like "doctor availability during scheduling" could cause major issues during system integration or user acceptance testing.

**4. Test Data Preparation:**

I worked on **preparing test data** to validate the system. This included creating sample doctor profiles, appointment schedules, and representative data that accurately reflected real-world usage, ensuring thorough testing of all app functions.

**5. Updating RTM (Requirements Traceability Matrix):**

I kept the **RTM** up-to-date, ensuring that all requirements were linked to corresponding test cases. This was crucial for tracking the progress of development and confirming that all business requirements were met before the system was delivered.

**Summary:**

I prepared and validated **test cases** based on use case diagrams, communicated with the client on design documents, wrote both **positive and negative test cases**, and ensured full test coverage. Additionally, I prepared test data and kept the **RTM** updated, ensuring that all requirements were met and accounted for in the testing phase. This meticulous approach helped ensure the app was robust and met business expectations.

## 4. Development:

During the **Development** phase of the Doctor Reporting App, my role involved ensuring effective communication between the business, technical teams, and the client, helping resolve queries, and fostering a collaborative environment to ensure smooth project execution.

**1. Organizing JAD (Joint Application Development) Sessions:**

I organized and facilitated **JAD sessions** to gather feedback and clarify requirements. These sessions allowed the business and technical teams to collaborate closely, ensuring the development team understood the client’s expectations and requirements.

**Examples and Time Intervals:**

* **Session 1 (2 hours):** We discussed the user authentication flow, focusing on the security requirements and how the app would manage user logins (e.g., two-factor authentication, password reset).
* **Session 2 (1.5 hours):** We reviewed the doctor's profile management feature, confirming what data needed to be captured (e.g., specialties, contact details, prescriptions) and how it would be displayed in the app.

**2. Clarifying Queries from the Tech Team During Coding:**

As the development progressed, the technical team had several **queries** related to the business logic, especially in areas such as appointment scheduling and doctor availability. I acted as a bridge to clarify these queries, ensuring alignment between technical implementation and business needs.

**Examples:**

* **Query:** A developer asked for clarification on how to handle overlapping appointment requests from different medical representatives. I clarified that the system should flag conflicts and prevent double-booking.
* **Query:** A developer asked if a medical rep could update doctor profiles in real-time during visits. I explained that only certain fields (e.g., contact details) should be editable, and other fields should be locked for security reasons.

**3. Handling Disagreements and Ensuring Cooperation in JAD Sessions:**

During some **JAD sessions**, I encountered team members who were resistant to certain design decisions or didn’t cooperate fully. As a BA, I addressed these issues by handling the situation diplomatically, ensuring that all voices were heard, and fostering a cooperative environment.

**Examples:**

* One developer disagreed with a proposed design for the doctor search functionality, feeling it would be too complex. I had a **one-on-one discussion** with them after the session, explained the business need for a multi-filter search feature, and demonstrated how it aligned with project goals. I also suggested simplifying the UI to make the feature more user-friendly.
* Another team member was not actively participating in a JAD session on **report generation** due to confusion about the system’s requirements. I set up a **one-on-one meeting** with them, clarified their concerns, and explained how their input was crucial for designing the report generation logic.

**4. Referring Diagrams to Code the Unit:**

To ensure that the development team followed the business requirements and design, I regularly referred to **UML diagrams** (use case, class, and activity diagrams) during the coding phase. These diagrams helped the developers understand the system flow and business logic, ensuring the application was being built as intended.

**Examples:**

* Developers referred to the **Use Case Diagram** to understand the sequence of actions for the “Generate Daily Report” feature and how it should interact with doctor and appointment data.
* They used the **Activity Diagram** for the appointment booking process to correctly implement validation checks, such as ensuring that time slots were available before scheduling.

**5. Conducting Regular Meetings with the Tech Team and Client:**

I organized **regular meetings** with both the technical team and the client to track progress, address issues, and gather feedback. These meetings were sometimes challenging, especially when team members were unavailable or the client had conflicting feedback.

**Examples and Time Intervals:**

* **Weekly Tech Team Meeting (1 hour):** Discussed progress on key features such as doctor search, report generation, and appointment scheduling. Addressed issues like bugs and bottlenecks.
* **Bi-Weekly Client Meeting (1.5 hours):** Reviewed design mockups, updated the client on development progress, and discussed feedback on specific features (e.g., appointment confirmation alerts).

When team members couldn’t attend a meeting, I **recorded the session** and sent the video along with meeting notes. Later, I scheduled **one-on-one discussions** with the absent team members to go over key points and address any questions they had.

**Example:**

* In one **weekly tech team meeting**, the lead developer was absent, so I recorded the session and shared it. Later, I had a follow-up meeting with the lead developer to discuss the feedback on the "doctor search" functionality and align on any changes needed.

# 5. Testing:

During the **Testing** phase of the Doctor Reporting App, I worked closely with both the technical team and the client to ensure that all requirements were thoroughly tested, and the system met business expectations.

**1. Prepare Test Cases from Use Cases:**

I derived detailed **test cases** from the use case diagrams to ensure comprehensive test coverage.

#### Test Case 1: Login Functionality

**Test Case ID:** TC\_Login\_01
**Test Case Title:** Verify User Login with Valid Credentials
**Module:** User Authentication
**Tested By:** [Tester Name]
**Date:** [Test Date]
**Test Priority:** High
**Pre-Conditions:**

* User has a valid username and password.
* The application is running, and the login page is accessible.

**Test Data:**

* Username: testuser
* Password: Test@1234

**Test Steps:**

1. Open the app’s login page.
2. Enter the valid username testuser in the username field.
3. Enter the valid password Test@1234 in the password field.
4. Click the “Login” button.
5. Wait for the page to load and verify the redirect to the homepage.

**Expected Result:**

* User should be logged in successfully and redirected to the homepage/dashboard.
* The user's name should appear at the top right corner (if implemented).

**Actual Result:**

* [To be filled by tester]

**Pass/Fail Criteria:**

* **Pass:** The user is successfully logged in and redirected to the homepage.
* **Fail:** The user is not logged in, or any error message is displayed.

**Comments:**

* [Any additional comments from the tester]

#### Test Case 2: Appointment Scheduling – Valid Input

**Test Case ID:** TC\_App\_Schedule\_01
**Test Case Title:** Verify Medical Representative can Schedule an Appointment with a Doctor
**Module:** Appointment Scheduling
**Tested By:** [Tester Name]
**Date:** [Test Date]
**Test Priority:** High
**Pre-Conditions:**

* User is logged in successfully.
* At least one doctor has available time slots for scheduling.

**Test Data:**

* Doctor: Dr. John Doe
* Patient Name: Jane Smith
* Time Slot: 2025-01-08, 10:00 AM

**Test Steps:**

1. Navigate to the “Schedule Appointment” page.
2. Search for and select Dr. John Doe from the list of doctors.
3. Choose the available time slot 10:00 AM on 2025-01-08.
4. Enter the patient's name: Jane Smith.
5. Click the “Schedule Appointment” button.
6. Verify that the appointment is confirmed and saved in the system.

**Expected Result:**

* The appointment is scheduled successfully and displayed in the user’s appointment list.
* A confirmation message or notification should appear, such as “Appointment scheduled successfully.”

**Actual Result:**

* [To be filled by tester]

**Pass/Fail Criteria:**

* **Pass:** Appointment is successfully scheduled and confirmed.
* **Fail:** Appointment is not saved, or an error occurs during the scheduling process.

**Comments:**

* [Any additional comments from the tester]

#### Test Case 3: Doctor Search – Invalid Input

**Test Case ID:** TC\_Doctor\_Search\_01
**Test Case Title:** Verify Doctor Search with Invalid Input
**Module:** Doctor Search
**Tested By:** [Tester Name]
**Date:** [Test Date]
**Test Priority:** Medium
**Pre-Conditions:**

* User is logged in.
* The search functionality is working.

**Test Data:**

* Invalid Doctor Name: Dr. XYZ

**Test Steps:**

1. Navigate to the “Doctor Search” page.
2. Enter an invalid doctor name Dr. XYZ in the search field.
3. Click the “Search” button.
4. Verify the system response to the invalid search query.

**Expected Result:**

* The system should display an error message such as “No results found” or show an empty list with a message indicating no matching results.

**Actual Result:**

* [To be filled by tester]

**Pass/Fail Criteria:**

* **Pass:** The system returns no results and displays an appropriate error message.
* **Fail:** The system shows incorrect results or crashes.

**Comments:**

* [Any additional comments from the tester]

#### Test Case 4: Generate Daily Report – Correct Data

**Test Case ID:** TC\_Report\_Generation\_01
**Test Case Title:** Verify Daily Report Generation for Medical Representative
**Module:** Reporting
**Tested By:** [Tester Name]
**Date:** [Test Date]
**Test Priority:** High
**Pre-Conditions:**

* User is logged in and has completed at least one appointment for the day.
* Appointment data is present in the system.

**Test Data:**

* Date: 2025-01-08
* Medical Rep: John Doe

**Test Steps:**

1. Navigate to the “Generate Report” section.
2. Select the date 2025-01-08 for the report.
3. Ensure the system pulls in all appointments and prescription data for the selected date.
4. Click the “Generate Report” button.
5. Verify that the system generates a report in PDF format.

**Expected Result:**

* A **PDF report** is generated containing the correct appointment details, such as patient names, doctors, time slots, and prescriptions for the day.

**Actual Result:**

* [To be filled by tester]

**Pass/Fail Criteria:**

* **Pass:** The report is generated correctly with accurate data.
* **Fail:** The report contains incorrect or missing data.

**Comments:**

* [Any additional comments from the tester]

#### Test Case 5: Appointment Cancellation

**Test Case ID:** TC\_App\_Cancel\_01
**Test Case Title:** Verify Appointment Cancellation Functionality
**Module:** Appointment Management
**Tested By:** [Tester Name]
**Date:** [Test Date]
**Test Priority:** High
**Pre-Conditions:**

* User is logged in and has at least one scheduled appointment.

**Test Data:**

* Appointment to Cancel: Dr. John Doe, 2025-01-08, 10:00 AM

**Test Steps:**

1. Navigate to the “My Appointments” section.
2. Select the appointment scheduled with Dr. John Doe for 2025-01-08, 10:00 AM.
3. Click on the “Cancel Appointment” button.
4. Confirm the cancellation in the popup or confirmation window.
5. Verify that the appointment is removed from the list of upcoming appointments.

**Expected Result:**

* The appointment should be successfully canceled and removed from the schedule. A confirmation message such as “Appointment successfully canceled” should appear.

**Actual Result:**

* [To be filled by tester]

**Pass/Fail Criteria:**

* **Pass:** Appointment is canceled and removed from the list.
* **Fail:** Appointment is not canceled, or an error occurs.

**Comments:**

* [Any additional comments from the tester]

**2. Perform High-Level Testing:**

I performed **high-level testing** to verify that the core functionalities of the app were working as expected, focusing on critical features such as **login**, **doctor search**, and **report generation**.

**3. Test Data Requested from Client:**

I worked with the client to obtain **test data** (e.g., doctor profiles, sample appointments, prescriptions) needed for validation, ensuring it was realistic and covered various use cases.

* **Example:** Requested test data on **doctor availability** and **medical rep visit logs** to validate the appointment scheduling functionality.

**4. Updated RTM (Requirements Traceability Matrix):**

I kept the **RTM** up to date to ensure that all requirements were mapped to corresponding test cases, tracking coverage and ensuring that no requirements were overlooked during testing.

**5. Take Signoff from Client:**

Once testing was completed, I gathered client feedback and obtained **signoff** on the tested features to ensure they met expectations.

* **Example:** After testing the **doctor report generation** feature, I obtained client approval, confirming that the generated reports were accurate and met their requirements.

**6. Prepare Client for UAT (User Acceptance Testing):**

I worked with the client to prepare them for **UAT**, providing necessary documentation, test cases, and support to ensure a smooth process.

 I worked closely with the client to ensure they were ready to validate the system’s functionality.

1. **Providing Documentation and Test Cases:**
I prepared detailed **UAT test cases** based on business requirements, covering key features like **doctor data management** and **appointment scheduling**. I provided a **UAT plan** outlining testing steps and criteria.
2. **Conducting Walkthrough Sessions:**
I conducted a **UAT walkthrough** with the client, explaining how to execute test cases, validate results, and report issues. For example, I demonstrated how to **add/edit doctor profiles** and ensured the client understood validation rules.
3. **Ongoing Support and Monitoring:**
I supported the client during UAT by answering questions, troubleshooting issues, and ensuring tests were performed smoothly. I tracked progress and helped log defects for resolution.

In summary, I ensured the client was well-prepared by providing **clear documentation**, **training**, and **ongoing support**, ensuring a successful UAT process.