

**Doctor Appointment System (DAS)**

**By:**

*Names: Ziad Elshazly Mohamed Magdy Merna Wesa Katreen William*

*IDs: 221001682*  *221001891 221002000 221001947*  **Under Supervision Of:**

*Prof. Mohamed Hassan Mahmoud Elgazzar Eng. Ahmed Jamal Metwaly*

[1.1 Project Overview: 2](#_Toc345532763)

[1.2 Problem Statement: 2](#_Toc1739266724)

[1.3 Proposed Solution: 3](#_Toc427666815)

[2.1.1 Patient: 4](#_Toc1580452527)

[2.1.3 Admin: 5](#_Toc2125705298)

[3.1.1 Functional Requirements: 5](#_Toc1279903179)

[3.1.2 Functional Requirements: 6](#_Toc857988003)

[3.1.3 Functional Requirements: 6](#_Toc1272596377)

[3.1.4 Functional Requirements: 6](#_Toc5533122)

[3.1.5 Functional Requirements: 7](#_Toc1274575151)

[3.1.6 Functional Requirements: 7](#_Toc1234316460)

[3.1.7 Functional Requirements: 7](#_Toc1825479798)

[3.1.8 Functional Requirements: 8](#_Toc568581107)

[4. Non-Functional Requirements 8](#_Toc1907036188)

[5.System Architecture & Design 10](#_Toc1164080500)

[5.1 Architecture Overview: 10](#_Toc436292692)

[ The Clinical Management System will use a three-layer architecture: 10](#_Toc364811532)

[5.3.1 UC-01: Admin: Manage Specializations (Add/Update/Delete) 12](#_Toc434434812)

[5.3.1.1 Admin: Manage Specializations (Add/Update/Delete) 13](#_Toc777089765)

[Goal: 13](#_Toc10324050)

[Assumptions: 13](#_Toc1815717071)

[Preconditions: 13](#_Toc1707114108)

[Postconditions: 13](#_Toc555492661)

[Main Success Scenario: 13](#_Toc873331095)

[Extensions: 13](#_Toc804721944)

[5.3.2 UC-02: Doctor: Manage Appointments (View/Change Status) 14](#_Toc661009454)

[Goal: 14](#_Toc1385635787)

[Assumptions: 14](#_Toc977435007)

[Preconditions: 14](#_Toc1197818043)

[Postconditions: 14](#_Toc1494017119)

[Main Success Scenario: 14](#_Toc912371002)

[Extensions: 15](#_Toc815854063)

[5.3.3 UC-03: Book Appointment 15](#_Toc16463567)

[Goal: 15](#_Toc1245807652)

[Assumptions: 15](#_Toc361053649)

[Preconditions: 15](#_Toc859310453)

[Postconditions: 15](#_Toc2013146485)

[Main Success Scenario: 15](#_Toc76630881)

[Extensions: 16](#_Toc1019821308)

[5.3.4 UC-04: Patient: Search Appointment 16](#_Toc917745956)

[Goal: 16](#_Toc210235701)

[Assumptions: 16](#_Toc1463133304)

[Preconditions: 16](#_Toc1060562322)

[Postconditions: 16](#_Toc2087993676)

[Main Success Scenario: 16](#_Toc1585150304)

[Extensions: 17](#_Toc168719404)

[5.3.5 UC-05: Doctor: Prescribe Medicine 17](#_Toc1667068075)

[Goal: 17](#_Toc486284013)

[Preconditions: 17](#_Toc1081232555)

[Postconditions: 17](#_Toc891285724)

[Main Success Scenario: 17](#_Toc1714959444)

[Assumptions: 18](#_Toc2097635987)

[Extensions: 18](#_Toc518583012)

[5.3.6 UC-05: Admin: Generate Doctor Registration Report 18](#_Toc1060268513)

[Goal: 18](#_Toc1193909671)

[Preconditions: 19](#_Toc714457477)

[Postconditions: 19](#_Toc1320861333)

[Main Success Scenario: 19](#_Toc421230914)

[Assumptions: 19](#_Toc1323937991)

[Extensions: 19](#_Toc292196086)

[6.2. Test Case: 32](#_Toc1129512908)

[Title: Profile View 32](#_Toc1539655641)

[6. 3. Test Case: 33](#_Toc917308591)

[Title: Login View 33](#_Toc1086061578)

[6.4. Test Case 33](#_Toc441443547)

[Title: Logout View 34](#_Toc10416288)

[6.5. Test Case 34](#_Toc114434347)

[Title: Create Appointment View 34](#_Toc1247831876)

[6.6 Test Case 35](#_Toc196976515)

[Title: User Search Appointments View 35](#_Toc1925131481)

[6.7. Test Case 35](#_Toc1191702671)

[Title: View Appointment Details View 35](#_Toc363988195)

[6.8. Test Case 36](#_Toc8538752)

[Title: Change Password View 36](#_Toc1919674460)

**1. Introduction**

# 1.1 Project Overview:

The **Doctor Appointment System** is a web-based application designed to simplify and enhance the operational processes of a healthcare clinic. It offers core functionalities such as patient online reservation, user role-based access control, and appointment scheduling. This project is developed using Agile methodology to ensure iterative development and efficient delivery.

# 1.2 Problem Statement:

* **Description of the Problem**: Doctors often struggle with managing appointments efficiently, keeping patient records organized, and maintaining secure access to sensitive information. Manual systems or generic solutions lead to mismanagement of appointments, delays in accessing patient information, and potential data security issues.
* **Proposed Solution**: The Doctor Appointment System will provide an online reservation feature for patients, allowing them to book appointments and view their own appointment securely. The system will also assign specific roles to clinic staff, ensuring that only authorized users can access or modify information. This tailored solution will reduce administrative burden, improve data security, and enhance patient satisfaction.

# 1.3 Proposed Solution:

The Doctor Appointment System will provide an online reservation feature for patients, allowing them to book appointments and view their own medical records securely. The system will also assign specific roles to clinic staff, ensuring that only authorized users can access or modify information. This tailored solution will reduce administrative burden, improve data security, and enhance patient satisfaction.

**1.4 Used Technologies:**

The clinical Management System is a web application developed using the following technologies:

* Python
* Django
* JavaScript
* CSS
* HTML
* Bootstrap
* MySQL
* Draw.io

**1.5 Intended Audience:**

This document is intended for:

* Developers: To understand the system requirements and guide the implementation process.
* Project Managers: To plan and track the progress of the project.
* Testers: To understand the system’s functionality and develop appropriate test cases.
* clients: To understand the capabilities and limitations of the proposed system.

**2.Overall Description**

**2.1User Personas:**

#### **2.1.1 Patient:**

* **Needs**: Ability to book through an intuitive interface.
* **Goals**: Reduce waiting times and improve convenience by managing appointments online.
* **Permissions**: Can only interact with their own data.

**2.1.2 Doctor:**

* **Needs**: Access patient records, update notes, and view scheduled appointments.
* **Goals**: Streamline patient information access to prepare for appointments efficiently.
* **Permissions**: Can view and update medical records and write prescriptions for assigned patients but does not have full administrative privileges.

#### **2.1.3 Admin:**

* **Needs**: Manage system-wide settings, including user role assignments and security protocols.
* **Goals**: Maintain system security and oversee smooth operations by managing user access, and other things like manage doctors' data.
* **Permissions**: Full access to all features and modules, including role management and system settings.

**3. Functional Requirements**

### **3.1.1 Functional Requirements:**

* **Title:** User Registration
* **Description:** The system will allow users to create an account and enter the system. Once authenticated, users will be directed to specific pages based on their roles.
* **Required Information:**

Username

Password

Role (e.g., Admin, Doctor, Patient)

### **3.1.2 Functional Requirements:**

* **Title:** Login
* **Description:** The system will authenticate users using their username and password to grant access to their respective accounts. After authentication, users will be directed to their role-specific dashboards.

### **3.1.3 Functional Requirements:**

* **Title:** Role-Based Authentication
* **Description:** All users must log in using credentials tied to their roles. The system will authenticate the credentials and grant access based on the user’s assigned role, ensuring the correct permissions are applied.

### **3.1.4 Functional Requirements:**

* **Title:** Online Appointment Booking (for Patients)
* **Description:** Patients can book appointments online by selecting a doctor, choosing an available time and date slot, and submitting the request. They can also view the status of their appointments.

### **3.1.5 Functional Requirements:**

* **Title:** Appointment Management (for Admin)
* **Description:** Admins can manage all patient appointments, including scheduling new appointments, viewing all appointments, and processing cancellations. They will have access to modify appointment statuses.

### **3.1.6 Functional Requirements:**

* **Title:** Patient Record Management (for Doctors)
* **Description:** Doctors can view, and update patient records based on their appointments. They can also prescribe medicine, recommend tests, and track patient progress.

**3.1.7 Functional Requirements:**

* **Title:** Role Management (for Admin)
* **Description:** Admin users can assign and manage roles for other users. Admins will have the authority to modify permissions and ensure that users have appropriate access based on their role within the system.

#### **4. Non-Functional Requirements**

**4.1 Security**:

Ensure secure data handling with encryption and role-based access control (e.g., admin)

**4.2 Scalability**:

Design the system to accommodate future growth in terms of features and users.

In addition to Support up to 1000 concurrent users without performance degradation, ensuring response times stay below 5 second for 95% of requests.

**4.3 Reliability**:

Minimize downtime and provide a stable user experience (Maintain an uptime of 99.9% monthly,

allowing for a maximum of 43 minutes of downtime).

**4.5 Usability**:

Ensure an intuitive interface that can be navigated easily by users of varying technical backgrounds.

# 5.System Architecture & Design

#### 

#### **5.1 Architecture Overview:**

#### The Doctor Appointment System will use a three-layer architecture:

1. **Presentation Layer**: Web-based interface for interacting with the system. This layer will differ depending on the user role, with specific views for patients, doctors, and admin users.
2. **Application Layer**: Manages business logic, handling requests like booking an appointment, viewing records, and role-based permissions.
3. **Database Layer**: Stores user credentials, appointment details, patient records, and other essential data.

**5.2 Use-Case Diagram:**

**5.3 Use-Case Diagram Scenarios**

#### **5.3.1 UC-01: Admin: Manage Specializations (Add/Update/Delete)**

1. **Actors**: Admin
2. **Objects**: Specialization Page, Specialization Database
3. **Steps**:
   1. Admin logs into the system (Authenticate Credentials).
   2. Admin selects "Manage Specializations."
   3. **Condition**:
      1. If **Add Specialization**:
         1. Admin inputs specialization details.
         2. Specialization Page sends the data to the database.
         3. Database confirms specialization added.
      2. If **Update Specialization**:
         1. Admin selects a specialization to update.
         2. Admin edits and submits the changes.
         3. Specialization Page sends the updated data to the database.
         4. Database confirms the update.
      3. If **Delete Specialization**:
         1. Admin selects a specialization to delete.
         2. Specialization Page requests confirmation.
         3. Admin confirms deletion.
         4. Specialization Page sends a delete request to the database.
         5. Database confirms the deletion.

**In other words, or more summary:**

### **5.3.1.1 Admin: Manage Specializations (Add/Update/Delete)**

#### **Goal:**

* Allow the Admin to manage specializations in the system by adding, updating, or deleting entries.

#### **Assumptions:**

* Admin is authenticated.
* The database for storing specializations is accessible.

#### **Preconditions:**

* Admin must be logged in.
* Admin must have access rights to manage specializations.

#### **Postconditions:**

* Specialization data is updated in the database.
* Confirmation messages are displayed.

#### **Main Success Scenario:**

1. Admin selects the "Manage Specializations" option.
2. Admin adds, updates, or deletes specialization data.
3. The database reflects the changes and confirms success.

#### **Extensions:**

* If the Admin inputs invalid data, an error message is displayed.
* If the database is inaccessible, the system shows an error message.

### **5.3.2 UC-02: Doctor: Manage Appointments (View/Change Status)**

#### **Goal:**

* Allow the doctor to view and update appointment statuses.

#### **Assumptions:**

* Doctor is authenticated.
* Appointments exist in the database.

#### **Preconditions:**

* The doctor is logged in and authorized to manage appointments.

#### **Postconditions:**

* Appointment statuses are updated in the database.

#### **Main Success Scenario:**

1. Doctor views appointments.
2. The doctor changed the status of the appointment.
3. The system updates the status and confirms success.

#### **Extensions:**

* If an invalid status is selected, an error message is displayed.
* If the database is inaccessible, an error message is displayed.

### **5.3.3 UC-03: Book Appointment**

#### **Goal:**

* Allow the patient to book an appointment with a doctor.

#### **Assumptions:**

* Patient does not need to register.
* Doctors and availability data exist in the database.

#### **Preconditions:**

* Patients must provide valid details.

#### **Postconditions:**

* The appointment is booked and confirmed.

#### **Main Success Scenario:**

1. Patient selects "Book Appointment."
2. Patient enters details and selects a doctor.
3. The system checks the doctor's availability.
4. The appointment is successfully booked.

#### **Extensions:**

* If the doctor is unavailable, the system prompts the patient to choose another time.
* If patient details are invalid, an error message is displayed.

### **5.3.4 UC-04: Patient: Search Appointment**

#### **Goal:**

* Allow the patient to search for their appointment.

#### **Assumptions:**

* Appointments exist in the database.

#### **Preconditions:**

* The patient must provide a valid appointment number or name.

#### **Postconditions:**

* Appointment details are retrieved and displayed.

#### **Main Success Scenario:**

* Patients enter an appointment number or name.
* The system retrieves and displays the appointment.

#### **Extensions:**

* If the appointment does not exist, an error message is displayed.

### **5.3.5 UC-05: Doctor: Prescribe Medicine**

#### **Goal:**

* To allow the doctor to create and save a prescription for a patient after viewing their appointment details.

#### **Preconditions:**

1. The doctor is logged into the system.
2. The patient has an appointment with the doctor.
3. The system has the patient's medical records available.

#### **Postconditions:**

1. The prescription is saved in the patient database.
2. The patient can view their prescription (if applicable in your system design).

#### **Main Success Scenario:**

1. The doctor logs into the system.
2. The doctor selects a specific patient from the **"Patient List"**.
3. The doctor clicks **"Prescribe Medicine"** and enters prescription details (e.g., medicine name, dosage, duration, notes).
4. The system saves the prescription in the database.
5. The doctor receives a success confirmation message.

#### **Assumptions:**

1. The doctor's account and patient appointment details are valid and up to date.
2. The prescription form includes all necessary fields for a valid prescription.

#### **Extensions:**

1. **Invalid Inputs:**

* If the doctor enters incomplete or invalid prescription details, the system prompts the doctor to correct the errors before saving.

1. **Database Error:**

* If the database fails to save the prescription, the system displays an error message and prompts the doctor to try again.

1. **No Appointment Found:**

* If no valid appointment exists for the selected patient, the system notifies the doctor.

### **5.3.6 UC-05: Admin: Generate Doctor Registration Report**

#### **Goal:**

* To enable the admin to generate a report of doctors registered within a specified date range.

#### **Preconditions:**

1. The admin is logged into the system with valid credentials.
2. There is at least one doctor registered in the database.

#### **Postconditions:**

1. The system generates a report containing all doctors registered in the specified date range.
2. The admin can view or download the report.

#### **Main Success Scenario:**

1. The admin logs into the system.
2. The admin selects the **"Doctor Registration Report"** option.
3. The admin enters the start and end dates for the report.
4. The system retrieves doctor registration data from the database.
5. The system generates the report and displays it to the admin.
6. The admin optionally downloads the report.

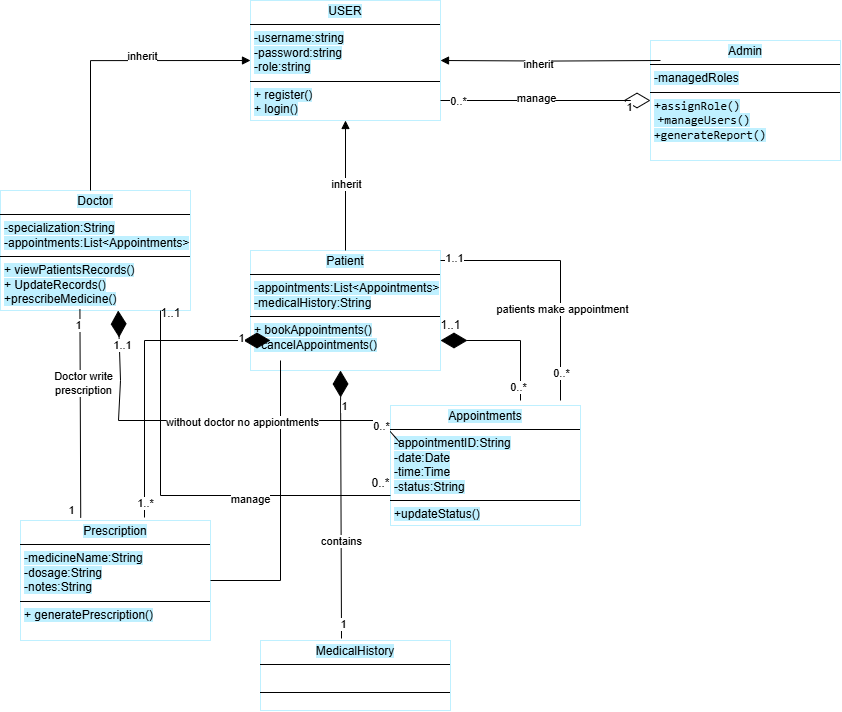
#### **Assumptions:**

1. The system has doctor registration data stored in the database.
2. The date range entered by the admin is valid.

#### **Extensions:**

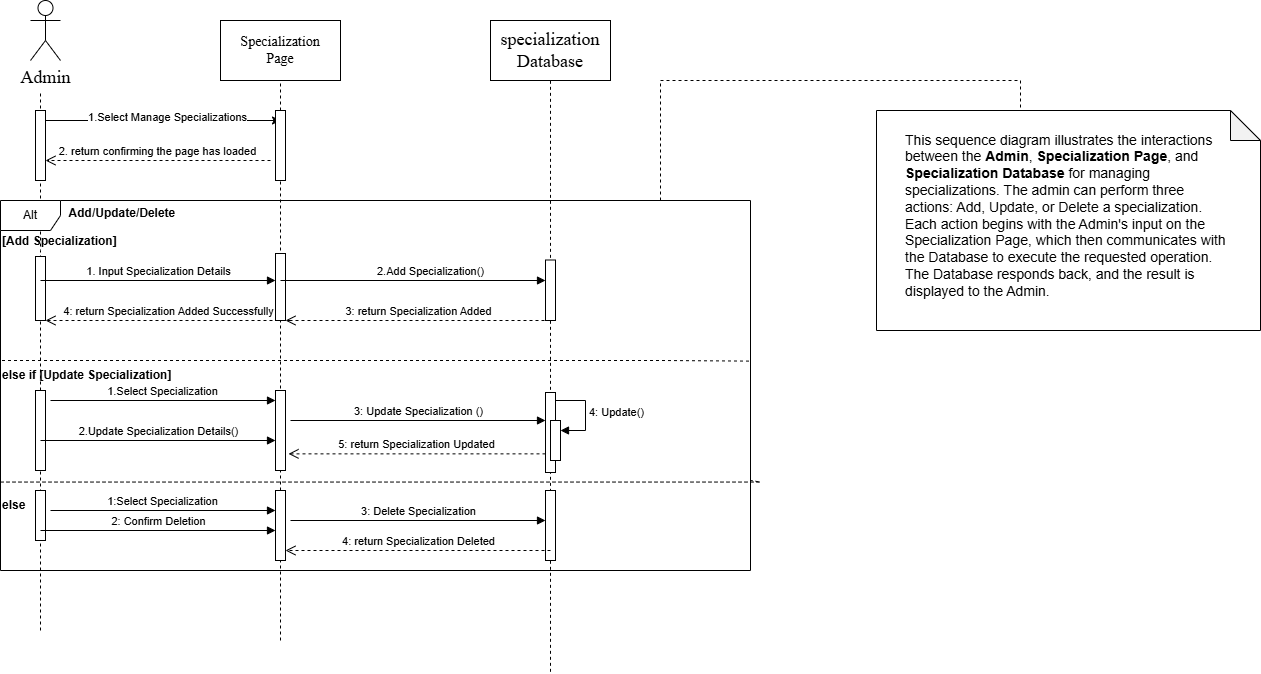
1. **No Doctors Found:**
   1. If no doctors were registered in the specified date range, the system displays a message: *"No data available for the selected date range."*
2. **Database Error:**
   1. If the database fails to retrieve the data, the system displays an error message and asks the admin to try again.
3. **Invalid Date Input:**
   1. If the admin enters an invalid date range (e.g., end date before start date), the system prompts the admin to correct it.

**5.4 UML Diagram:**

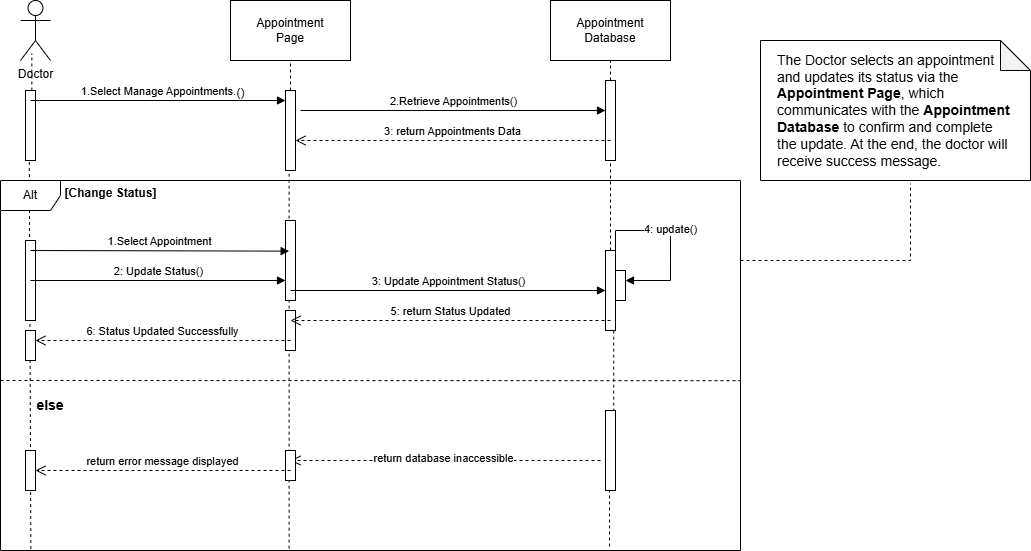


**5.5 Sequence Diagram:**

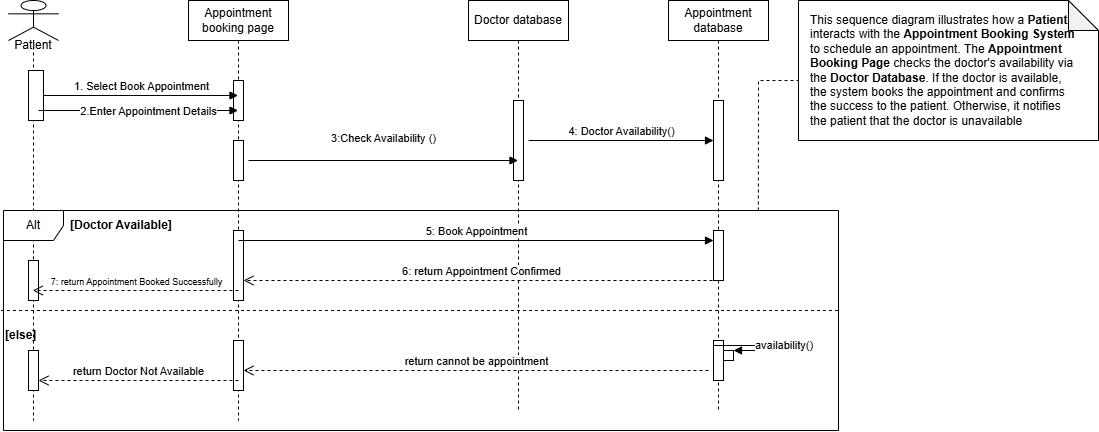
**5.5.1 Sequence Diagram:**



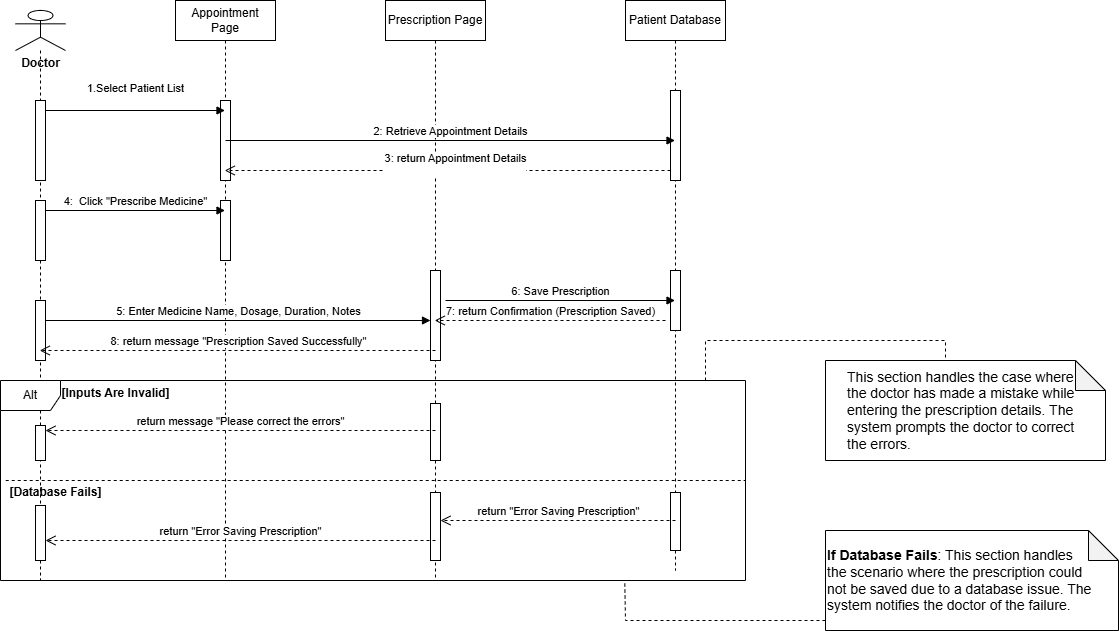
**5.5.2 Sequence Diagram:**



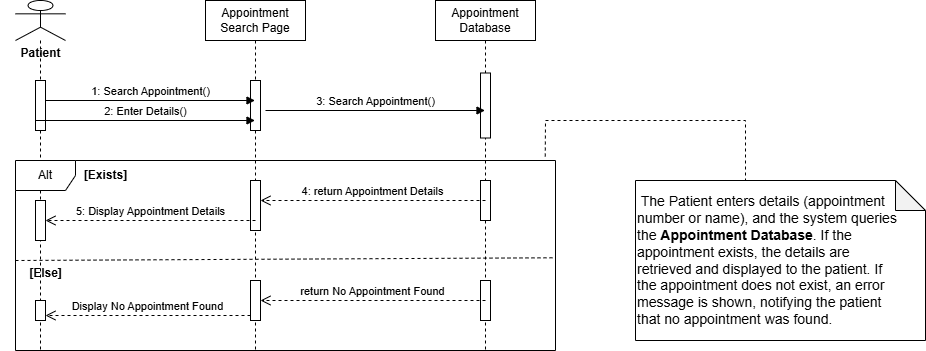
**5.5.3 Sequence Diagram:**



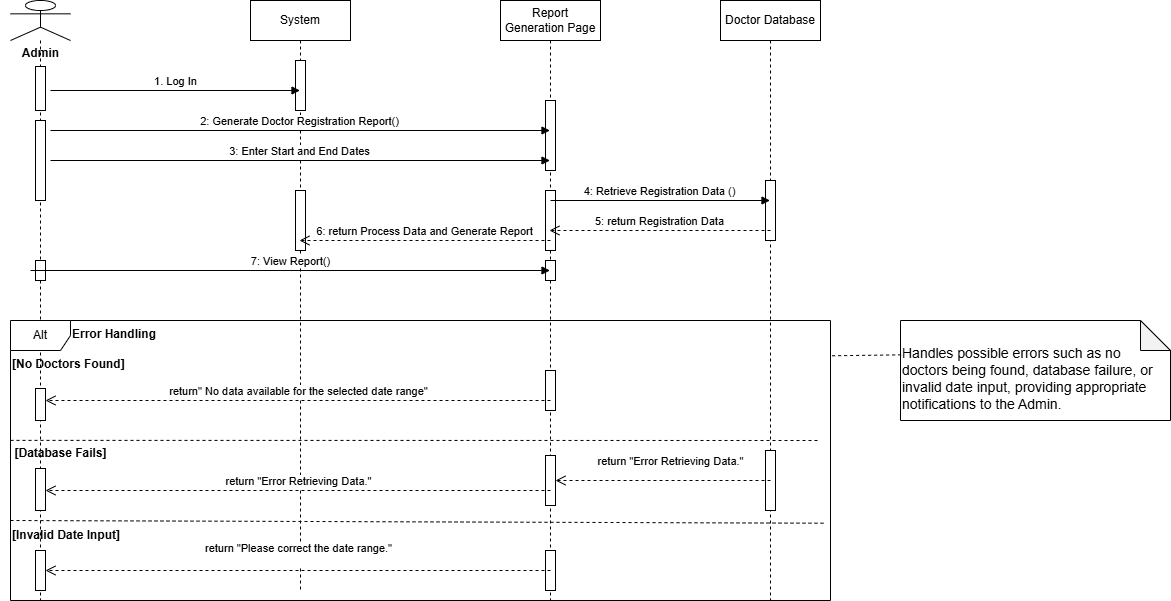
**5.5.4 Sequence Diagram:**



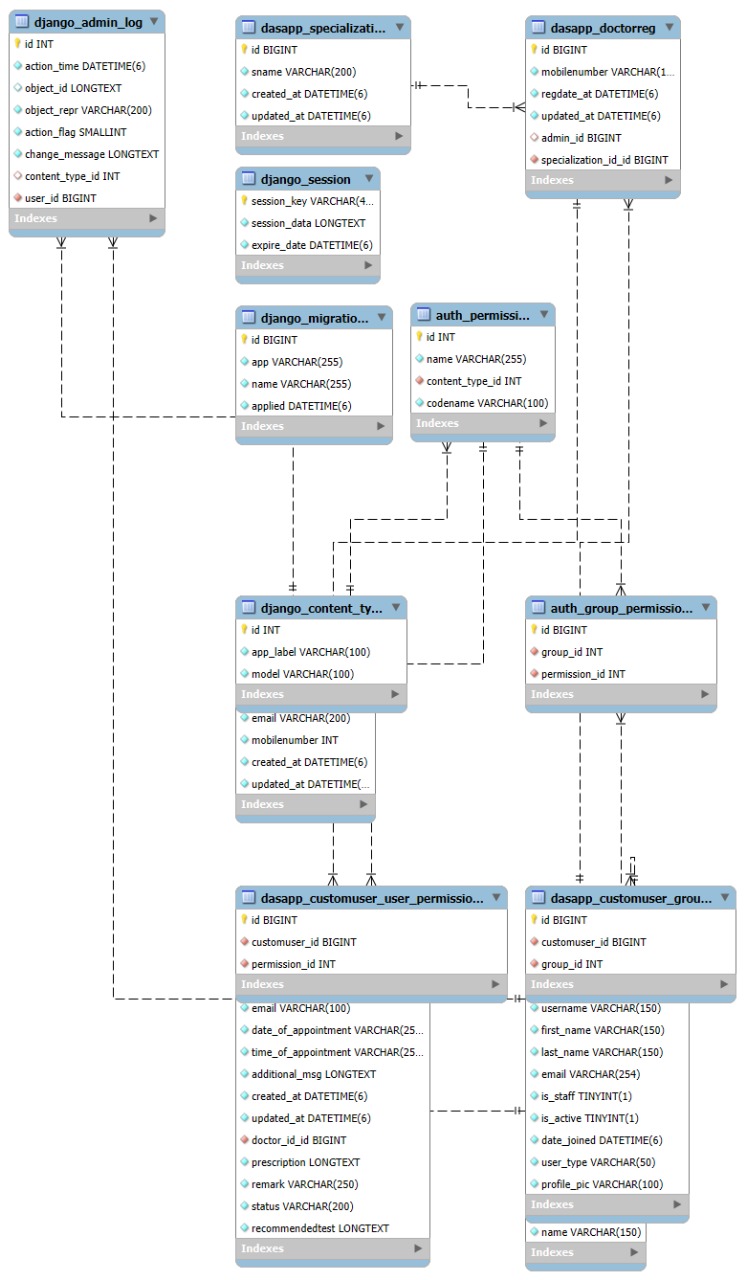
**5.5.5 Sequence Diagram:**



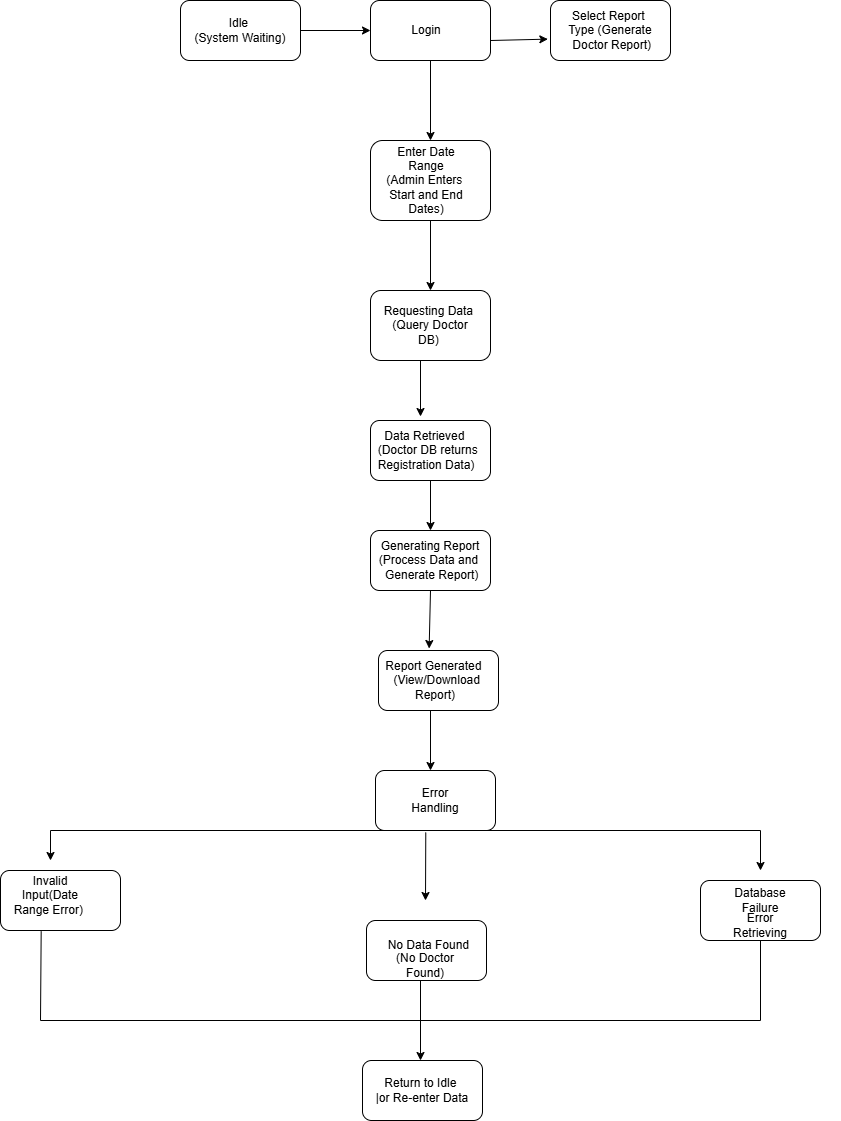
**5.5.6 Sequence Diagram:**



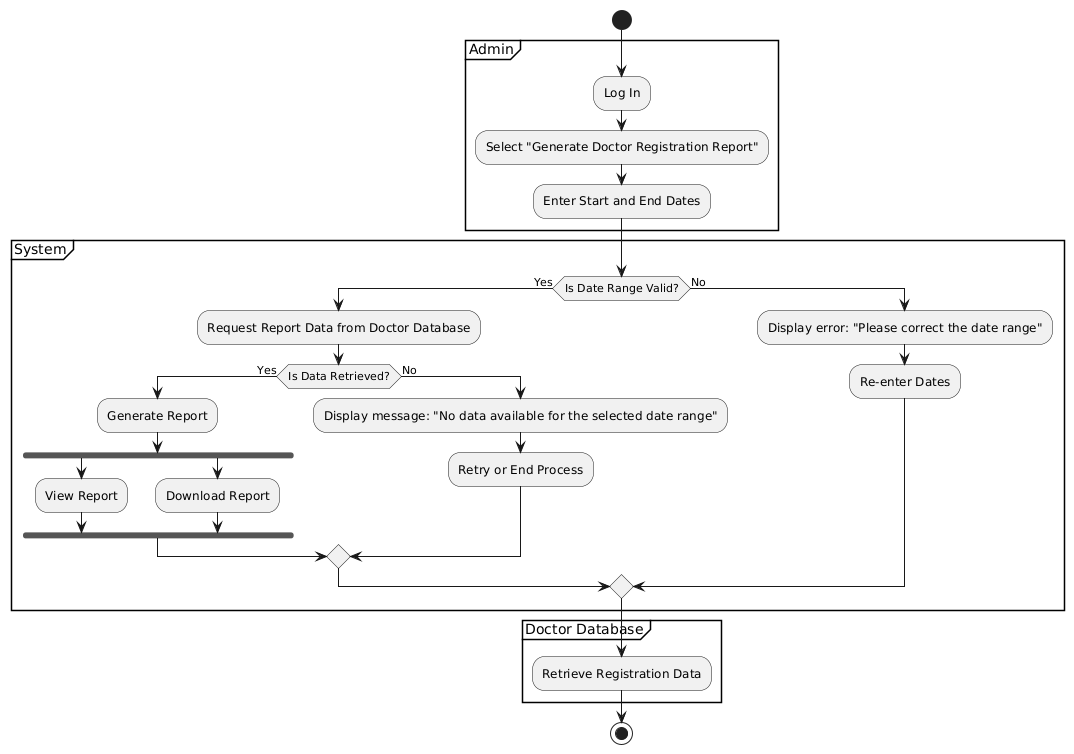
**5.6 ERD Diagram:**



**5.7 State Diagram:**



**5.8 Activity Diagram:**



**6.Testcases**

**6.1: Testcase**

**Title: Index View**

* **Test Objective**: To ensure that the Index view renders the correct template and provides the expected context data.
* **Test Steps**:
  + Access the Index view using the GET method.
  + Verify the HTTP status code is 200.
  + Ensure the template used is index.html.
  + Check if the context contains doctorview and page.
  + Validate that the correct doctorview and page objects are passed in the context.
* **Expected Result**:
  + The page loads successfully with the index.html template.
  + The context contains doctorview with the created doctor and page with the created page.

#### **6.2. Test Case:**

#### **Title: Profile View**

* **Test Objective**: To check if the Profile view correctly displays the user's profile information.
* **Test Steps**:
  + Log in as a test user.
  + Access the Profile view using the GET method.
  + Verify the HTTP status code is 200.
  + Ensure the correct template profile.html is used.
  + Validate that the context contains the correct user object.
* **Expected Result**:
  + The profile page loads successfully.
  + The logged-in user's information is displayed correctly.

#### **6. 3. Test Case:**

#### **Title: Login View**

* **Test Objective**: To verify that the Login view functions correctly for both GET and POST requests.
* **Test Steps**:
  + Test GET request to ensure the login page is rendered with the correct template.
  + Test POST request by submitting valid user credentials.
  + Verify successful login (302 redirect) and that the user is authenticated.
* **Expected Result**:
  + GET request displays the login.html template.
  + POST request successfully logs the user in and redirects them.
  + The user is authenticated.

#### **6.4. Test Case**

#### **Title: Logout View**

* **Test Objective**: To ensure that the Logout view correctly logs the user out.
* **Test Steps**:
  + Log in as a test user.
  + Access the Logout view using the GET method.
  + Verify the HTTP status code is 302 (redirect after logout).
  + Ensure the user is logged out (not authenticated).
* **Expected Result**:
  + The user is successfully logged out and redirected.

#### **6.5. Test Case**

#### **Title: Create Appointment View**

* **Test Objective**: To verify that the Create Appointment view works as expected for both GET and POST requests.
* **Test Steps**:
  + Test GET request to ensure the appointment creation page is displayed correctly.
  + Test POST request by submitting valid appointment data.
  + Check that an appointment is created and saved in the database.
  + Verify the HTTP status code is 302 (redirect after successful appointment creation).
* **Expected Result**:
  + The page renders correctly.
  + The appointment is successfully created and stored in the database.

#### **6.6 Test Case**

#### **Title: User Search Appointments View**

* **Test Objective**: To validate that the User Search Appointments view displays the correct appointments for the logged-in user.
* **Test Steps**:
  + Log in as a test user with existing appointments.
  + Access the User Search Appointments view.
  + Verify that the appointments are correctly displayed.
  + Ensure the context contains appointments and that they match the expected details.
* **Expected Result**:
  + The search results display the correct appointments for the logged-in user.

#### **6.7. Test Case**

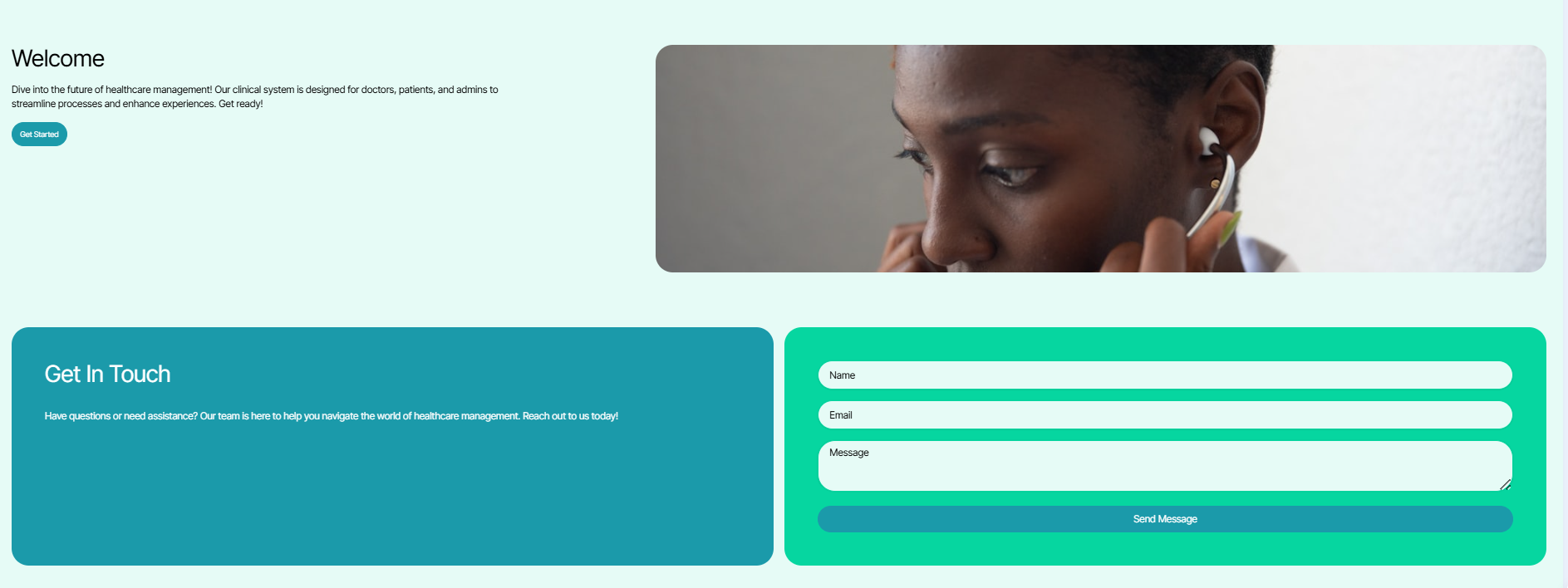
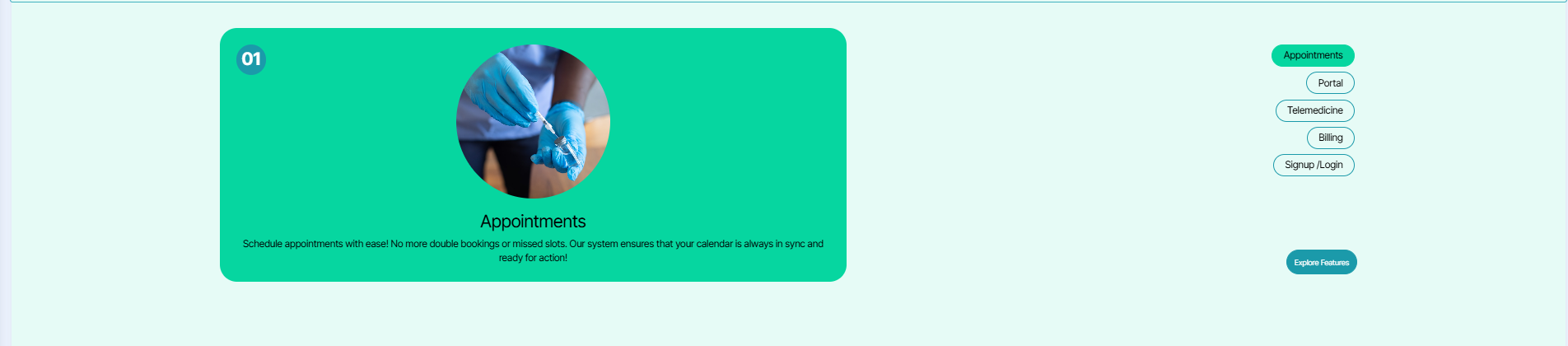
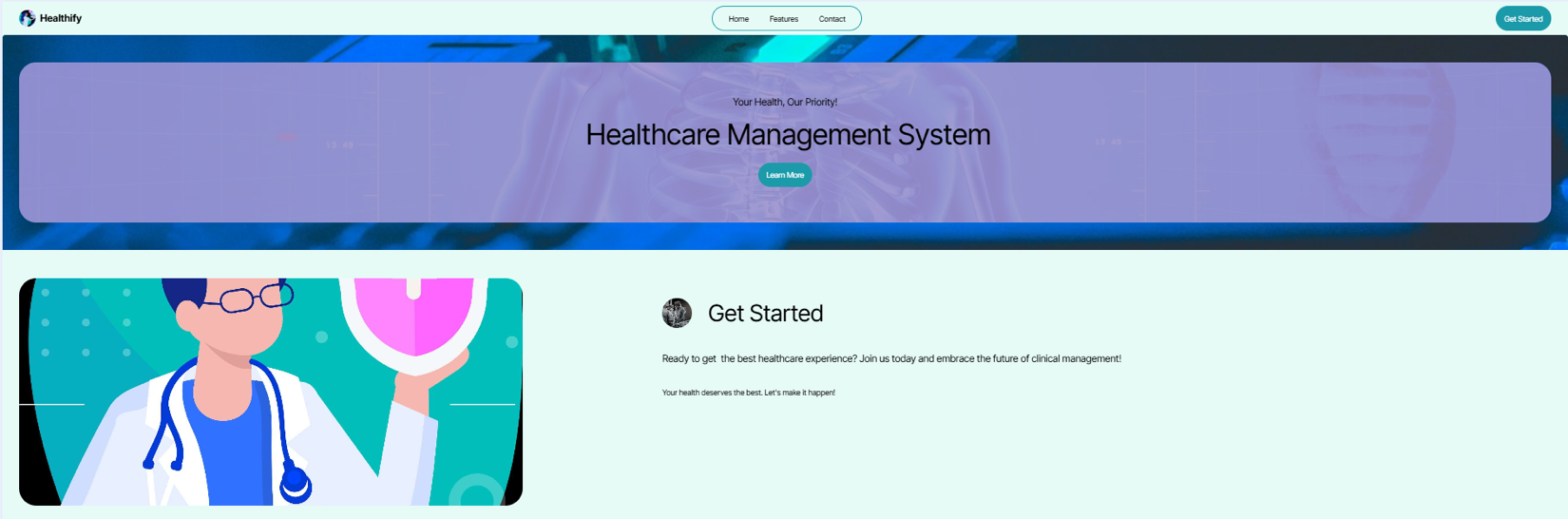
#### **Title: View Appointment Details View**

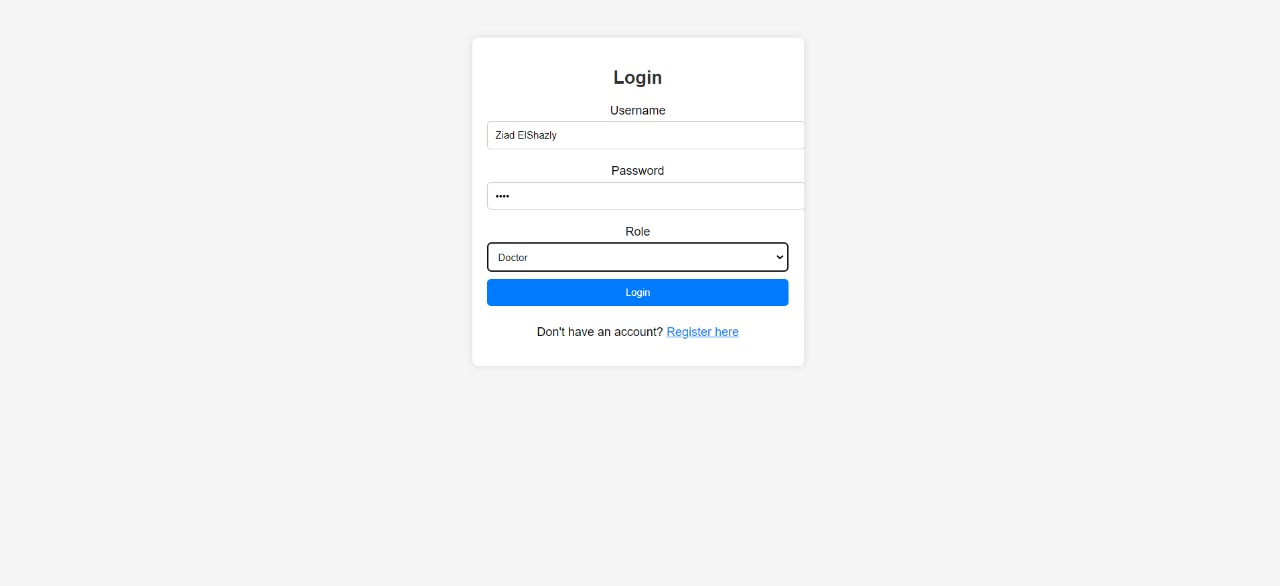
* **Test Objective**: To ensure that the View Appointment Details view displays the correct details for a given appointment.
* **Test Steps**:
  + Log in as a test user.
  + Access the View Appointment Details view using a valid appointment ID.
  + Verify the appointment details are rendered correctly.
* **Expected Result**:
  + The appointment details are displayed as expected.

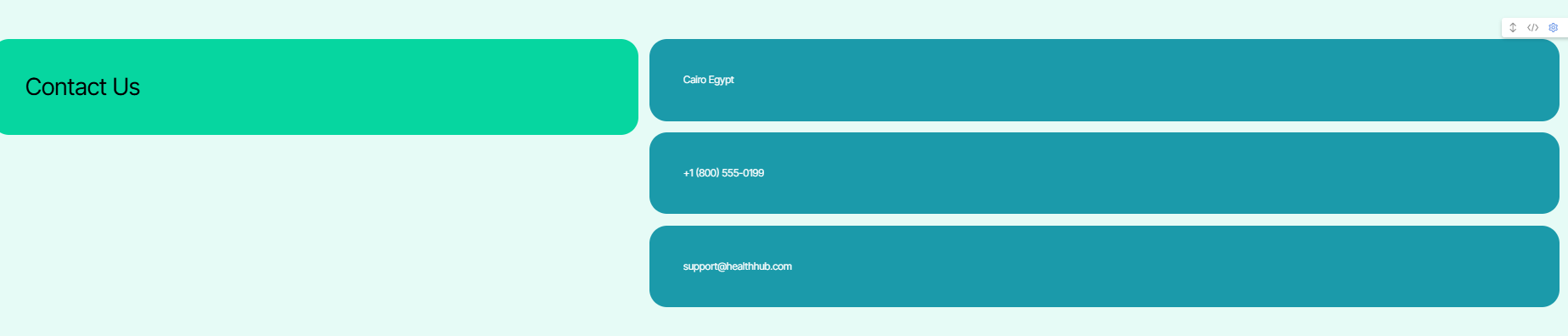
#### **6.8. Test Case**

#### **Title: Change Password View**

* **Test Objective**: To verify that the Change Password view works correctly, allowing users to change their password.
* **Test Steps**:
  + Log in as a test user.
  + Access the Change Password view.
  + Submit valid and invalid password changes.
  + Ensure the password is updated correctly when valid credentials are provided.
  + Verify error messages for invalid inputs.
* **Expected Result**:
  + The user is able to change their password successfully.
  + Appropriate error messages are shown for invalid inputs.

**7.1 Wireframes (Frontend Design Drawings)** 





**8.Conclusion:**

The **Doctor Appointment System** is designed to address key operational challenges faced by clinics, providing an efficient solution for managing patient appointments, records, and secure user access. By implementing role-based access and à user-friendly online reservation feature, this system aims to enhance patient convenience, streamline clinic workflows, and improve data security. This proposal outlines the initial Project structure, user personas, requirements, and system design, forming a solid foundation for development under the Agile method. Moving forward, our team will focus on delivering a Minimum Viable Product (MVP) in Phase One, with plans for iterative enhancements in Phase Two based on feedback and testing outcomes.