Smart Doctor Appointment System

Submitted By
S.M. Jubail Islam
Student Id : 221-35-1048
Department of Software Engineering
Daffodil International University

Supervised By
Dr. Shabnom Mustary
Assistant Professor
Faculty of Science and Information Technology
Daffodil International University

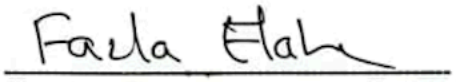
This project report has been submitted in fulfilment of the requirements for the degree a **Bachelor of Science in Software Engineering**

Department of Software Engineering
Daffodil International University
Fall 2025

Approval

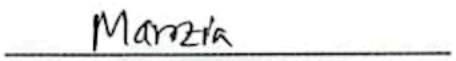
This thesis titled on **Smart Doctor Appointment System**, submitted by **S.M. Jubail Islam (ID: 221-35-1048)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents

BOARD OF EXAMINERS



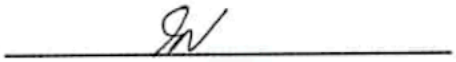
Chairman

Dr. Fazla Ealhe
Assistant Professor & Associate Head
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University



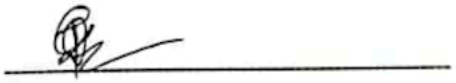
Internal Examiner 1

Dr. Marzia Ahmed
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University



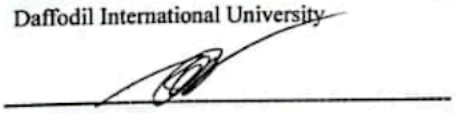
Internal Examiner 2

Dr. Shabnom Mustary
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University



Internal Examiner 3

Md. Rajib Mia
Lecturer (Senior Scale)
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University



External Examiner

Mohammod Abul Kashem, PhD
Professor
Department of Computer Science and Engineering
DUET, Bangladesh

Declaration

I hereby declare that the work presented in this project report titled "Smart Doctor Appointment System" is the result of my own effort and has not been submitted, in whole or in part, to any other institution for any academic qualification.

I confirm that all materials drawn from other sources have been properly acknowledged and cited in accordance with academic standards. I also affirm that this project was carried out under the supervision of Dr. Shabnom Mustary, Assistant Professor, Department of Software Engineering, Daffodil International University.

I further declare that this report is an original contribution completed by me, S.M. Jubail Islam (ID: 221-35-1048, Batch-38), and any errors or oversights within the work are entirely my responsibility.

Submitted by:

Date:

Jubail

S.M. Jubail Islam

ID: 221-35-1048

Department of Software Engineering

Daffodil International University

Certified by:

Date:

Dr. Shabnom Mustary

Assistant Professor

Faculty of Science and Information Technology

Department of Software Engineering

Daffodil International University

Acknowledgement

First and foremost, I express my deepest gratitude to the Almighty for granting me the strength, patience, and opportunity to complete this project successfully.

I would like to extend my sincere appreciation to my supervisor, **Dr. Shabnom Mustary, Assistant Professor, Department of Software Engineering, Daffodil International University,** for her continuous guidance, valuable suggestions, and constructive feedback throughout the development of this project. Her encouragement and expert supervision played a vital role in shaping the final outcome of this work.

I am also grateful to all faculty members of the **Department of Software Engineering** for providing the academic foundation and inspiration that supported my learning journey. Their teachings and dedication greatly contributed to the successful completion of this project.

My special thanks go to my friends and classmates for their cooperation, helpful discussions, and continuous motivation during the project's development and documentation phase.

Finally, I would like to acknowledge my family for their unconditional support, prayer, and encouragement, which have always been a source of strength and inspiration for me.

Table of Contents

List of Tables

List of Figures

Chapter 1 : Introduction

1.1 Project Overview

The rapid advancement of digital technologies has significantly transformed the healthcare sector, enabling faster, more efficient, and more accessible services for patients and medical professionals. Traditional appointment booking methods—such as walk-in scheduling, manual registration, or telephone-based reservations—often result in long waiting times, administrative burdens, and miscommunication between patients and healthcare providers. These inefficiencies highlight the need for an integrated digital platform capable of streamlining appointment management and improving the continuity of care.

The Doctor Appointment System is proposed as a web-based solution designed to automate and simplify the appointment scheduling process within clinical and hospital environments. By utilizing modern web technologies, real-time communication tools, and a structured database-driven backend, the system aims to eliminate conventional scheduling challenges and enhance the overall patient experience. Furthermore, it supports healthcare providers by offering centralized control over time slots, appointments, patient records, and communication flows.

This project focuses on creating a user-friendly, secure, and scalable system that enables patients to book appointments conveniently and allows doctors to manage their schedules with minimum administrative effort. The inclusion of additional features—such as real-time messaging, medical record uploads, automatic reminders, and notification management—further strengthens the system's capability to support modern healthcare operations.

Overall, the introduction of this system aligns with the increasing demand for digital healthcare services. It addresses critical gaps in traditional appointment management processes while ensuring improved accuracy, accessibility, and operational efficiency. This chapter provides a foundational overview of the project, outlining its rationale, purpose, planning activities, feasibility considerations, and the targeted users who will interact with the system.

1.2 Project Purpose

The Doctor Appointment System is a web-based platform designed to simplify and modernize the process of booking and managing medical appointments. The system provides dedicated interfaces for patients, doctors, and administrators, enabling each user group to perform their tasks efficiently through a centralized and secure environment.

Patients can view available doctors, check schedules, book appointments, communicate through real-time chat, and access their medical records. Doctors can manage their availability, view upcoming appointments, upload medical documents, and respond to patient messages. Administrators oversee user management, monitor system activity, and ensure proper operation of appointment workflows.

The primary purpose of the project is to replace traditional manual scheduling methods with a streamlined, automated solution that reduces administrative workload, minimizes scheduling errors, and improves communication between patients and healthcare providers. By integrating modern digital tools, the system aims to enhance service accessibility, operational efficiency, and overall patient experience.

1.3 Project Planning and Initiation

The development of the Doctor Appointment System began with a structured planning process aimed at defining the project's objectives, requirements, and overall scope. This phase involved identifying the key problems in the traditional appointment system, determining the primary stakeholders, and outlining the essential features needed to improve efficiency and user experience. Clear timelines, resource needs, and development responsibilities were also established to ensure smooth project execution.

Project initiation included gathering preliminary information from potential users, reviewing existing appointment systems, and selecting suitable technologies for implementation. This phase

ensured that the project was aligned with user needs, technically feasible, and achievable within the planned time frame.

1.3.1 Feasibility Study

A feasibility study was conducted to determine whether the proposed system could be practically developed and successfully deployed. The analysis evaluated several dimensions—including market, technical, financial, and operational feasibility—to confirm the viability of the Doctor Appointment System.

Table 1.1 : Feasibility Summary

Feasibility Area	Key Findings
Market	Addresses a growing need for faster, more organized, and digitalized medical appointment services.
Technical	Utilizes a modern, secure, and low-risk technology stack (Next.js, MongoDB, Node.js, Pusher), ensuring reliable system performance.
Financial	Highly cost-effective due to the use of open-source technologies and minimal infrastructure costs.
Operational	Easily integrates with existing clinical workflows, reducing manual workload and improving service efficiency.

Phase 1: Preliminary Analysis & Project Scope

a. Problem Statement

Table 1.2 : Problem Statement

Actor / Stakeholder	Problem	Description / Impact
Patient	Difficulty booking appointments	Patients often face long waiting times, manual queues, and unclear scheduling information, leading to inconvenience and service delays.
Doctor	Inefficient schedule management	Doctors struggle to manage their time slots manually, causing overlapping appointments and administrative burden.
Admin / Clinic Staff	Lack of central control	Manual record-keeping makes it difficult to track doctor availability, patient bookings, and system-wide operations.
Healthcare System	Limited communication flow	There is no streamlined channel for real-time updates, notifications, or patient–doctor messaging, resulting in miscommunication and missed appointments.

b. Proposed Solution

I. **For Patients:**

The system will provide an easy-to-use online platform where patients can browse available doctors, view schedules, book appointments, and receive automated reminders. Patients will also have access to a secure medical records section and a real-time chat interface to communicate with their assigned doctors.

II. **For Doctors:**

The system will provide a dedicated dashboard that allows doctors to manage their availability, review upcoming appointments, upload medical documents, and interact with

patients through a built-in chat feature. Doctors can easily update their schedule, minimize administrative workload, and maintain organized records of patient interactions.

III. **For Administrators:**

The system will provide a centralized management panel where administrators can oversee users, monitor appointments, validate doctor profiles, and manage operational workflows. They can view system statistics, handle scheduling issues, and ensure smooth functioning across all user groups.

c. Project Scope

In-Scope Features:

I. **User Authentication:** A secure sign-up and login system with three user roles: Patient, Doctor, and Admin.

II. **Patient Dashboard:** Allows patients to browse doctors, view available schedules, book appointments, and access medical records.

III. **Doctor Schedule Management:** Tools for doctors to create, update, and manage their available time slots and appointments.

IV. **Real-Time Chat:** A communication module enabling patients and doctors to exchange messages within an active appointment session.

V. **Medical Record Management:** A system for doctors to upload medical documents and for patients to view their stored health records.

VI. **Admin Dashboard:** Allows administrators to manage users, oversee appointments, and monitor system activity.

VII. **Notification System:** In-app and real-time alerts for appointment confirmations, reminders, and chat messages.

VIII. **Automated Reminders:** A scheduled reminder system to alert patients and doctors about upcoming appointments.

Out-of-Scope Features:

I. Online payment processing or billing integration.

II. Video-based telemedicine consultations or virtual appointments.

III. AI-powered medical analysis, diagnosis, or automated decision-making.

IV. Integration with external hospital management systems or government health databases.
V. Mobile app version (Android/iOS) beyond the web platform.
VI. Multi-language support or translation features.
VII. Prescription generator or pharmaceutical automation.

Phase 2: Market Feasibility

The market feasibility analysis examines the demand, relevance, and viability of implementing a web-based Doctor Appointment System within the healthcare context of Bangladesh. The goal is to determine whether the system addresses real market needs, offers competitive value, and aligns with national digital transformation initiatives.

Table 1.3 : Market Feasibility

Factor	Analysis (Bangladesh Perspective)	Viability
Target Market	Hospitals, clinics, diagnostic centers, and private chambers that require digital appointment management.	**High**
Market Need	Growing demand for quick appointment access, reduced waiting times, and digital healthcare services among urban and semi-urban populations.	**High**
Competitive Advantage	Provides an integrated platform with doctor scheduling, real-time chat, medical record sharing, and automated reminders — features not commonly bundled in local systems.	**High**
Alignment	Supports "Smart Bangladesh" and national digital health initiatives by reducing manual paperwork and modernizing healthcare operations.	**High**

Phase 3: Technical Feasibility

The technical feasibility assessment evaluates whether the required technologies, tools, and development skills are available to build and support the Doctor Appointment System. Since the project uses widely adopted web technologies, scalable cloud services, and secure backend frameworks, the system is technically practical, maintainable, and highly viable within the context of Bangladesh.

Table 1.4 : Technical Feasibility

Component	Chosen Technology	Feasibility & Rationale
Frontend	React (Next.js)	**High:** Modern, fast, and widely supported framework suitable for building responsive user interfaces. Strong local and global community support.
Backend	Next.js API Routes	**High:** Efficient, scalable, and capable of handling real-time features and secure server-side operations.
Database	MongoDB (Atlas)	**High:** Flexible, schema-friendly NoSQL database offering high scalability, low maintenance, and cloud accessibility.
Real-Time Communication	Pusher	**High:** Provides reliable, low-latency real-time messaging for chat and notifications without complex server setup.
Authentication	NextAuth	**High:** Secure and stable authentication library supporting role-based access and session management.
File Handling	Cloud Storage	**High:** Supports uploading and retrieving medical records efficiently with low infrastructure overhead.
UI & Styling	Tailwind CSS	**High:** Enables rapid UI development with a clean, utility-first approach and excellent community support.

Phase 4: Financial Feasibility

The financial feasibility assessment examines the estimated costs, required resources, and overall affordability of developing and maintaining the Doctor Appointment System. Since the project uses open-source technologies, cloud-based services, and minimal infrastructure, the overall

financial burden is low. This makes the system highly suitable for healthcare organizations seeking a cost-effective digital solution.

Smart Doctor Appointment System

Economic Feasibility (Cost Calculated in USD)
Expected 10,000 Users (Initial Deployment Scenario)

Financial Item	Complete Budget	Initial Budget	Duration
Developer	$3,500.00	$0.00	6–8 Months
Tester	$400.00	$0.00	1–2 Weeks
Operational Cost	$250.00	$10.00	Monthly
Maintenance & Support	$350.00	$0.00	Monthly
Hosting & Infrastructure	$200.00	$5.00	Monthly
Total Estimated Cost	$4,700.00	$15.00	—

Figure 1.1 : Financial Feasibility For This System

1.4 Target User Profile and Tentative Elicitation Process

Target User 1: Patient

Table 1.5 : User Profile Patient

Aspect	Details
Profile	Individuals seeking medical consultations, including general patients, chronic patients, and first-time visitors using online appointment services.

Goals	To easily find a suitable doctor, check available schedules, book appointments online, receive reminders, and communicate with doctors when needed.
Elicitation	Insights were collected through informal conversations with frequent clinic visitors and online survey responses about common booking challenges. Observations of real clinic queues were also used to understand pain points such as waiting time and manual registration delays.

Target User 2: Doctor

Table 1.6 : User Profile Doctor

Aspect	**Details**
Profile	Licensed medical professionals working in hospitals, clinics, or private chambers who manage daily appointments and patient consultations.
Goals	To manage their appointment schedule efficiently, update availability, review patient bookings, upload medical records, and communicate with patients through real-time messaging.
Elicitation	Requirements were gathered through discussions with practicing doctors and medical interns to understand workflow challenges such as schedule overlap, high administrative load, and difficulty tracking patient information. Their feedback guided the design of the scheduling and communication features.

Target User 3: Admin

Table 1.7 : User Profile Admin

Aspect	**Details**
Profile	Administrative staff responsible for overseeing system operations, managing user accounts, and ensuring smooth coordination between patients and doctors.

Goals	To monitor system usage, manage user roles, verify doctor information, handle appointment issues, and maintain the overall integrity and functionality of the platform.
Elicitation	Requirements were collected through interviews with clinic administrators and staff to understand challenges in manual record-keeping, appointment coordination, and user management. Their input shaped features related to admin control, monitoring, and system supervision.

1.5 System Requirements

1.5.1 Development Environment Requirements

Table 1.8 : Development Environment Requirements

Category	Requirements
Operating System	Windows, macOS, or Linux
Software	1. Node.js (version 18.x or later) 2. Code editor (Visual Studio Code recommended) 3. Git and a command-line interface (CLI) 4. Web browser (Chrome, Edge, or Firefox)
Accounts	1. GitHub (source code hosting and version control) 2. Vercel (deployment of the Next.js application) 3. MongoDB Atlas (cloud database service) 4. Pusher account (real-time messaging and notifications) 5. Email / notification service accounts, if external providers are used

1.5.2 User Environment Requirements

Table 1.9 : User Environment Requirements

Category	Requirements
Operating System	Windows, macOS, Linux, or Android/iOS (via mobile browser)
Web Browser	Any modern browser such as Google Chrome, Mozilla Firefox, Microsoft Edge, or Safari (latest version recommended)
Device Requirements	Desktop, laptop, tablet, or smartphone with stable internet access
Network Requirements	Minimum 2–5 Mbps internet connection for smooth navigation, real-time chat, and timely notification updates
Additional Requirements	JavaScript enabled in browser; pop-ups and notifications allowed for real-time updates

1.6 Project Scheduling

1.6.1 Gantt Chart

The development of the Doctor Appointment System is planned across a 12-month timeline. The schedule includes requirement gathering, analysis, design, development, testing, deployment, and documentation. The table below presents a Gantt chart–style breakdown of the project phases.

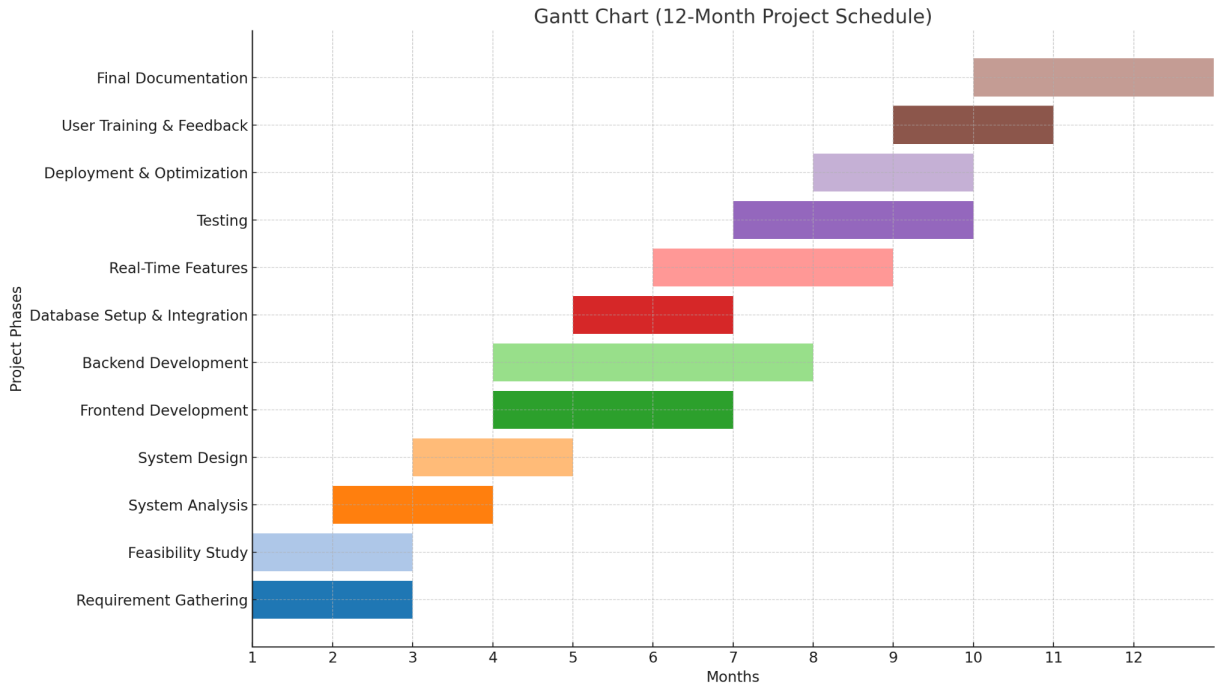


Figure 1.2 : Gantt Chart

1.6.2 Risk Management

Risk management helps identify potential challenges that may affect the successful completion of the Doctor Appointment System. By assessing risks early, the development team can implement strategies to reduce their likelihood and minimize impact. The following table outlines the key risks, their severity, and the mitigation strategies planned for this project.

1.6.3 Risk Management Table

Table 1.10 : Risk Management

Risk Factor	Severity	Possible Impact	Mitigation Strategy
Technical Issues	High	System downtime, feature delays, or integration failures.	Conduct regular testing, maintain backups, use stable frameworks, and establish recovery procedures.

Data Security Risks	High	Unauthorized access, data leaks, or compromised patient records.	Implement encryption, strong authentication, role-based access control, and periodic security audits.
Schedule Delays	Medium	Extended development timeline and delayed deployment.	Maintain a clear project timeline, conduct weekly reviews, and apply agile iteration methods.
Insufficient User Adoption	Medium	Users may resist new technology or prefer traditional methods.	Provide training materials, ensure a simple UI, and gather feedback for usability improvements.
Budget Constraints	Low	Limited resources for additional tools or services.	Use open-source technologies, optimize cloud usage, and monitor expenses regularly.
Technical Skill Gaps	Low	Developers may require additional time to learn tools.	Conduct early training sessions and use well-documented frameworks.

Chapter 2 : Design and Implementation

2.1 Functional Requirements

Functional Requirements define what the system must do to meet the needs of patients, doctors, and administrators. These requirements describe the core operations, user interactions, and system behaviors necessary for the Smart Doctor Appointment System to function effectively.

The key functional requirements of the proposed system are outlined below:

Table 2.1 : FR User Registration

FR01	**User Registration**
Description	The system shall allow new users (Patient, Doctor, Admin) to create an account, validate unique email addresses, and securely store hashed passwords.
Stakeholders	Patients, Doctors, Admin

Table 2.2 : FR User Login & Dashboard Redirection

FR02	**User Login & Dashboard Redirection**
Description	The system shall allow users to log in using email and password, authenticate credentials, and redirect them to dashboards based on their role.
Stakeholders	Patients, Doctors, Admin

Table 2.3 : FR Role-Based Access Control

FR03	**Role-Based Access Control**
Description	The system shall enforce three roles (Admin, Doctor, Patient) and restrict access to pages, features, and APIs based on role permissions.
Stakeholders	All Users, System Administrator

Table 2.4 : FR View Doctors

FR04	**View Doctors**
Description	The system shall allow patients to view available doctors with their specialization, available days, and available time slots.
Stakeholders	Patients, Doctors

Table 2.5 : FR View Available Time Slots

FR05	**View Available Time Slots**
Description	The system shall allow patients to check available appointment time slots for a selected doctor.
Stakeholders	Patients

Table 2.6 : FR Book Appointment

FR06	**Book Appointment**
Description	The system shall allow patients to book an appointment by selecting date and time and shall send booking notifications.
Stakeholders	Patients, Doctors

Table 2.7 : FR Manage Patient Appointments

FR07	**Manage Patient Appointments**
Description	The system shall allow patients to view upcoming, completed, and canceled appointments and cancel appointments when needed.
Stakeholders	Patients, Doctors

Table 2.8 : FR Patient–Doctor Chat

FR08	**Patient–Doctor Chat**
Description	The system shall allow patients and doctors to engage in real-time chat within active chat sessions and notify users of new or unread messages.
Stakeholders	Patients, Doctors

Table 2.9 : FR View Patient Medical Records

FR09	**View Patient Medical Records**
Description	The system shall allow patients to view medical records uploaded by doctors.
Stakeholders	Patients, Doctors

Table 2.10 : FR Manage Doctor Schedule

FR10	**Manage Doctor Schedule**
Description	The system shall allow doctors to create, edit, delete, and mark appointment time slots as unavailable.
Stakeholders	Doctors

Table 2.11 : FR Doctor Appointment Management

FR11	**Doctor Appointment Management**
Description	The system shall allow doctors to view appointments assigned to them and filter them by date.
Stakeholders	Doctors

Table 2.12 : FR Medical Record Management (Doctor)

FR12	**Medical Record Management (Doctor)**
Description	The system shall allow doctors to upload medical files with notes, store file metadata, and delete or replace existing medical records.
Stakeholders	Doctors, Patients

Table 2.13 : FR User Management (Admin)

FR13	**User Management (Admin)**
Description	The system shall allow admins to view all registered users and activate or deactivate user accounts.
Stakeholders	Admin

Table 2.14 : FR Appointment Management (Admin)

FR14	**Appointment Management (Admin)**
Description	The system shall allow admins to view all appointments, cancel or delete them, and create schedules on behalf of doctors.
Stakeholders	Admin, Doctors

Table 2.15 : FR Admin Notifications

FR15	**Admin Notifications**

Description	The system shall allow admins to send system-wide notifications and reminders to users.
Stakeholders	Admin, All Users

Table 2.16 : FR Appointment Validation (No Overlap)

FR16	**Appointment Validation (No Overlap)**
Description	The system shall prevent overlapping appointments and automatically mark selected slots as booked.
Stakeholders	Patients, Doctors

Table 2.17 : FR Appointment Status Tracking

FR17	**Appointment Status Tracking**
Description	The system shall support and maintain appointment statuses: Pending, Completed, Cancelled, Cancelled by Admin, and No-show.
Stakeholders	Patients, Doctors, Admin

Table 2.18 : FR Appointment Reminder System

FR18	**Appointment Reminder System**
Description	The system shall send automated reminders to patients and doctors before appointment time and prevent duplicate reminders (e.g., with a reminderSent flag).
Stakeholders	Patients, Doctors

Table 2.19 : FR Notification System

FR19	**Notification System**
Description	The system shall provide an in-app notification panel, show unread notification counts, allow marking notifications as read, and support real-time alerts for chats, appointment updates, medical record uploads, and announcements (with support for Email/SMS/WhatsApp via APIs).
Stakeholders	All Users

Table 2.20 : FR System Security & Data Management

FR20	**System Security & Data Management**
Description	The system shall store all system data (users, appointments, schedules, medical records), hash passwords, enforce authentication on sensitive routes, and validate uploaded files for allowed MIME types.
Stakeholders	System Admin, All Users

2.2 Non-Functional Requirements

Table 2.21 : Non-Functional Requirements

ID	**Category**	**Description**
NFR01	**Performance**	The system shall load all major pages (doctor list, appointments, dashboard) within **3 seconds** under normal network conditions. Real-time features (chat and notifications) shall deliver updates within **1 second**.
NFR02	**Security**	The system shall enforce secure authentication, password hashing, role-based access control, and validation of all file uploads. All

		sensitive data (user profiles, medical records, appointment details) shall be encrypted in transit using HTTPS.
NFR03	**Usability**	The system shall provide a user-friendly interface that is easy to navigate for patients, doctors, and admins. Key functions (booking, chat, record viewing) shall be accessible in no more than **three clicks**.
NFR04	**Scalability**	The system shall support growth in users, appointments, chat messages, and medical records without performance degradation. It shall be able to scale horizontally to handle increasing traffic and database load.
NFR05	**Reliability**	The system shall maintain **99% uptime**, ensure consistent delivery of notifications, and prevent data loss during operations. Appointment bookings shall remain accurate even under high load conditions.
NFR06	**Maintainability**	The system shall use modular and well-documented code to support easy updates, debugging, and addition of new features (e.g., new notification channels or medical record formats). All major components shall follow clean architecture principles.

2.3 Object-oriented System Design using UML

2.3.1 Use Case Diagram

Figure 2.1 : Use Case Diagram

2.3.2 Case Description

UC1 – User Registration

Table 2.22 : Case Description for User Registration

Use Case	User Registration
Goal	Allow new users (Patient, Doctor, Admin) to create an account in the system.
Precondition	User is not yet registered. Registration page is available.
Success End Condition	A new user record is stored in the database with hashed password and valid, unique email.
Failed End Condition	Account is not created and the user is informed of the error (e.g., duplicate email, invalid data).
Primary Actors:	Patient, Doctor, Admin
Secondary Actors:	Database system
Description / Main Success Scenario	User opens the registration page. User enters required details (name, email, password, role, etc.). System validates input and checks email uniqueness. System hashes the password. System stores the new user record in the database. System displays a successful registration message and may redirect to login.
Alternative Flows	A1: Email already exists → System shows an error and asks for a different email.

	A2: Required fields missing/invalid → System highlights fields and requests correction.
Quality Requirements	Secure password hashing (e.g., bcrypt/argon2). Validation and response within ~3 seconds. Clear, user-friendly error messages.

UC2 – User Login & Dashboard Access

Table 2.23 : Case Description for User Login & Dashboard Access

Use Case	User Login & Dashboard Access
Goal	Authenticate users and provide access to the correct dashboard based on their role.
Precondition	User account exists and is active.
Success End Condition	User is authenticated and redirected to the appropriate dashboard (Patient, Doctor, Admin).
Failed End Condition	User remains unauthenticated and sees a clear error (wrong credentials, deactivated account).
Primary Actors:	Patient, Doctor, Admin
Secondary Actors:	Authentication service / session manager
Description / Main Success Scenario	User navigates to the login page. User enters email and password. System validates input and checks credentials. System verifies account status (active).

	System creates a session / token for the user. System determines role and redirects to the appropriate dashboard.
Alternative Flows	A1: Incorrect email or password → System shows an error and allows retry. A2: Account deactivated → System informs user that the account is inactive.
Quality Requirements	Login response within ~2 seconds. All communications over HTTPS. Session/token managed securely (no exposure of credentials).

UC3 – Book Appointment

Table 2.24 : Case Description for Book Appointment

Use Case	Book Appointment
Goal	Allow a patient to book a non-overlapping appointment with a doctor and receive reminders/notifications.
Precondition	Patient is logged in. Doctor has an existing schedule with available slots.
Success End Condition	Appointment is created, slot is marked as booked, and confirmation notification is sent.
Failed End Condition	Appointment is not created and patient is informed of the reason (e.g., conflict, invalid data).
Primary Actors:	Patient
Secondary Actors:	Doctor Appointment system

	Notification system
Description / Main Success Scenario	Patient opens “Book Appointment” page. Patient selects a doctor. System displays doctor’s available dates and time slots. Patient selects a date and time. System checks the selected slot for conflicts/overlaps. System creates the appointment with initial status (e.g., Pending). System marks that slot as booked. System sends an in-app notification (and scheduled reminder) to patient and doctor.
Alternative Flows	A1: Selected time slot has just been booked → System shows conflict message and refreshes availability. A2: Patient cancels before saving → No appointment is stored. A3: Validation error (missing date/time) → System prompts patient to provide required info.
Quality Requirements	Real-time availability (minimal race conditions). Conflict check and booking within ~2–3 seconds. Reliable reminder mechanism (no duplicate reminders).

UC4 – Patient–Doctor Chat

Table 2.25 : Case Description for Patient–Doctor Chat

Use Case	Patient–Doctor Chat

Goal	Provide real-time messaging between a patient and an assigned doctor during an active chat session.
Precondition	Patient and doctor are logged in. A valid appointment and chat session exist and are not expired.
Success End Condition	Messages are delivered and displayed to both participants in real time.
Failed End Condition	Message is not delivered; user is notified (connection issue or session expired)
Primary Actors:	Patient, Doctor
Secondary Actors:	Real-time messaging service (e.g., Pusher/WebSocket) Notification system
Description / Main Success Scenario	Patient or doctor opens the chat linked to a specific appointment. System verifies that both users are the assigned patient and doctor and that the session is active. User types a message and clicks send. System sends the message via real-time messaging service. The receiver's chat window displays the new message. System updates message status (e.g., delivered/read).
Alternative Flows	A1: Chat session expired → System shows message that session is closed; sending disabled. A2: Network connection lost → System shows error and retries sending when connection resumes.
Quality Requirements	Message latency typically < 1 second.

	Only authorized users can access a given chat. Chat history stored securely and associated with the appointment.

UC5 – Manage Medical Records

Table 2.26 : Case Description for Manage Medical Records

Use Case	Manage Medical Records
Goal	Allow doctors to upload, view, and replace medical records related to a patient's appointment; allow patients to view them.
Precondition	Doctor or patient is logged in. Relevant appointment exists.
Success End Condition	Medical record is stored/updated with metadata and is viewable by authorized users.
Failed End Condition	File is not stored or retrieved; user is notified of the error (e.g., invalid type, system error).
Primary Actors:	Doctor
Secondary Actors:	Patient File storage subsystem Database
Description / Main Success Scenario	Doctor selects an appointment from the list. Doctor opens the "Medical Records" section. Doctor uploads one or more medical files and optionally adds notes. System validates file size and MIME type. System stores the file and its metadata (name, type, size, appointmentId).

	Patient later logs in and opens the same appointment. Patient views the list of associated medical records and opens/downloads them.
Alternative Flows	A1: File type not allowed → System rejects upload and shows reason. A2: Doctor chooses to replace existing record → System removes old record (or marks as superseded) and saves new one. A3: Record not found when patient tries to view → System shows "No records available".
Quality Requirements	Secure storage (access control by patient/doctor only). Reasonable upload time; large file handling. Auditability (who uploaded/modified records).

UC6 – Manage Schedules (Doctor)

Table 2.27 : Case Description for Manage Schedules (Doctor)

Use Case	Manage Schedules (Doctor)
Goal	Enable doctors to create, edit, delete, and mark appointment time slots as unavailable.
Precondition	Doctor is logged in.
Success End Condition	Updated schedule is saved and visible in patient booking interface.
Failed End Condition	Changes are not saved due to validation errors or system issues; doctor is notified.
Primary Actors:	Doctor

Secondary Actors:	Admin (may adjust schedules on behalf of doctor) Database / scheduling system
Description / Main Success Scenario	Doctor opens the schedule management page. Doctor selects a date range or specific day. Doctor adds new available time slots or edits existing ones. System validates that new slots do not overlap with existing booked appointments. Doctor (optionally) marks certain slots as unavailable. System saves the updated schedule. Patients see updated availability when booking.
Alternative Flows	A1: Doctor tries to delete a slot that already has a booked appointment → System blocks deletion and notifies doctor. A2: Time range invalid (end before start) → System shows validation error.
Quality Requirements	Intuitive UI for time-slot editing. Robust validation to avoid inconsistencies. Schedule updates propagated quickly to patient views.

UC7 – Admin User & Appointment Management

Table 2.28 : Case Description for Admin User & Appointment Management

Use Case	Admin User & Appointment Management
Goal	Allow administrators to manage users, appointments, and (if necessary) doctors' schedules.

Precondition	Admin is logged in with appropriate privileges.
Success End Condition	Selected user/appointment/schedule changes are applied successfully
Failed End Condition	Changes are not applied; admin is informed of the problem.
Primary Actors:	Admin
Secondary Actors:	Patient, Doctor Database system
Description / Main Success Scenario	Admin opens the admin dashboard. Admin views lists of users and appointments. Admin selects a user to activate/deactivate or edit basic info. Admin selects an appointment to view details. Admin cancels or deletes an appointment if necessary (e.g., doctor unavailable). Admin optionally creates or modifies a schedule on behalf of a doctor. System saves changes and, where relevant, notifies affected users.
Alternative Flows	A1: Admin attempts to modify non-existent record → System shows "Record not found". A2: Admin cancels appointment close to appointment time → System warns and asks for confirmation.
Quality Requirements	Strong access control (only admins have these options). All admin actions logged for auditing. Interface responsive and able to handle large data sets.

UC8 – Notification Handling

Table 2.29 : Case Description for Notification Handling

Use Case	Notification Handling
Goal	Generate and deliver notifications for key events (bookings, updates, messages, uploads, announcements) through in-app and external channels.
Precondition	Triggering event occurs (e.g., appointment created, message sent). User has at least one notification channel active.
Success End Condition	Notification is created, delivered to user, and appears in notification panel with correct status.
Failed End Condition	Notification is not delivered or not recorded; system logs the failure.
Primary Actors:	Patient, Doctor, Admin (as recipients)
Secondary Actors:	Notification service (in-app, email, SMS, WhatsApp, push provider)
Description / Main Success Scenario	A triggering event occurs (e.g., appointment booked). System creates a notification entry with type, recipient, and message. System pushes the notification to the in-app notification panel. System optionally sends external notifications via configured channels (email/SMS/WhatsApp). User opens the notification panel and views new notifications. User marks notifications as read; system updates unread count.
Alternative Flows	A1: External channel API fails → System retries or logs error and still shows in-app notification.

	A2: User has disabled a specific channel → System only uses allowed channels.
Quality Requirements	Real-time or near real-time delivery (< 2 seconds for in-app). Accurate unread counts. High reliability for external channels with retry mechanisms.

UC9 – System Security & Data Management

Table 2.30 : Case Description for System Security & Data Management

Use Case	System Security & Data Management
Goal	Ensure secure storage, access, and handling of all system data and enforce security controls.
Precondition	System is deployed and running with configured database and security modules
Success End Condition	Data is stored safely; only authorized users access protected information; file uploads are validated.
Failed End Condition	Unauthorized access attempt detected or insecure data handling; operation is blocked and logged.
Primary Actors:	System Admin
Secondary Actors:	All users (indirectly) Authentication/authorization services Database and file storage systems
Description / Main Success Scenario	User attempts to access a protected resource (API/page). System verifies authentication token/session.

	System checks user role and permissions. System grants or denies access accordingly. When storing credentials, system hashes passwords before saving. When a file is uploaded, system validates MIME type and size before storing. System regularly backs up critical data (appointments, users, records).
Alternative Flows	A1: Authentication fails or token expired → System returns unauthorized error and redirects to login. A2: User lacks permission for requested resource → System denies access and logs the attempt. A3: File fails validation → System rejects upload and shows error.
Quality Requirements	All traffic over HTTPS. Strong password hashing and secure token/session management. Role-based authorization checks on all sensitive endpoints. Regular backups and secure storage for medical data (HIPAA-like sensitivity, if applicable).

2.3.3 Activity Diagram

- User Registration

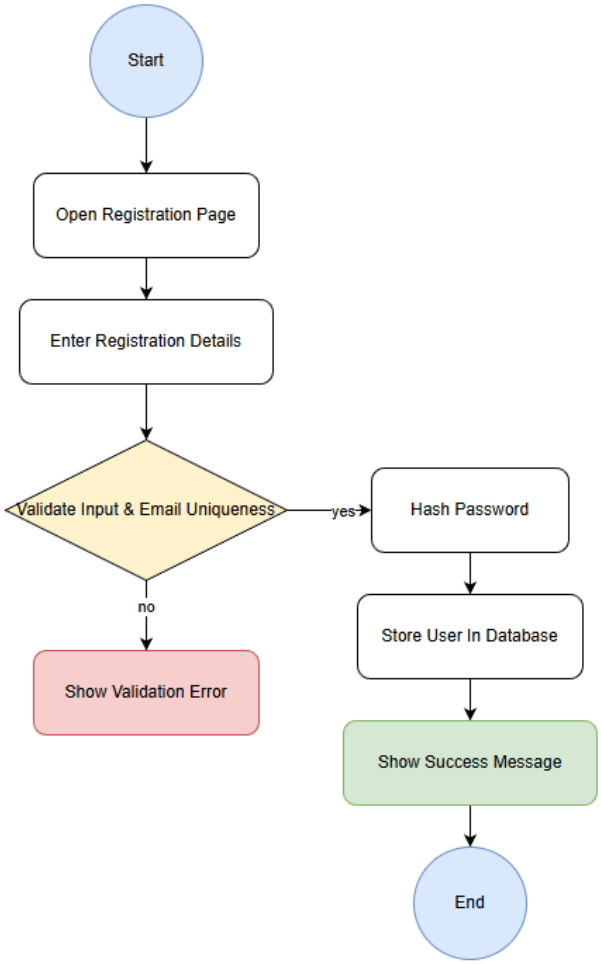


Figure 2.2 : Activity Diagram for User Registration

- User Login & Dashboard Access

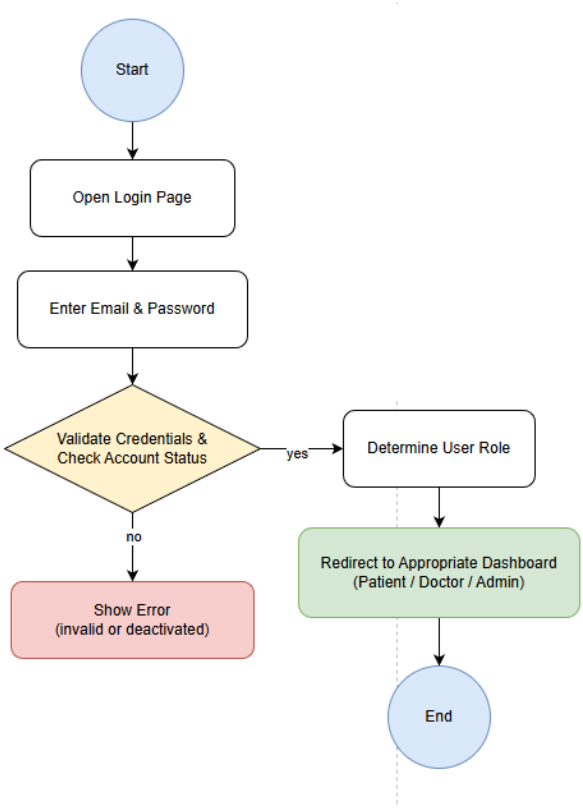


Figure 2.3 : Activity Diagram for User Login & Dashboard Access

- Book Appointment

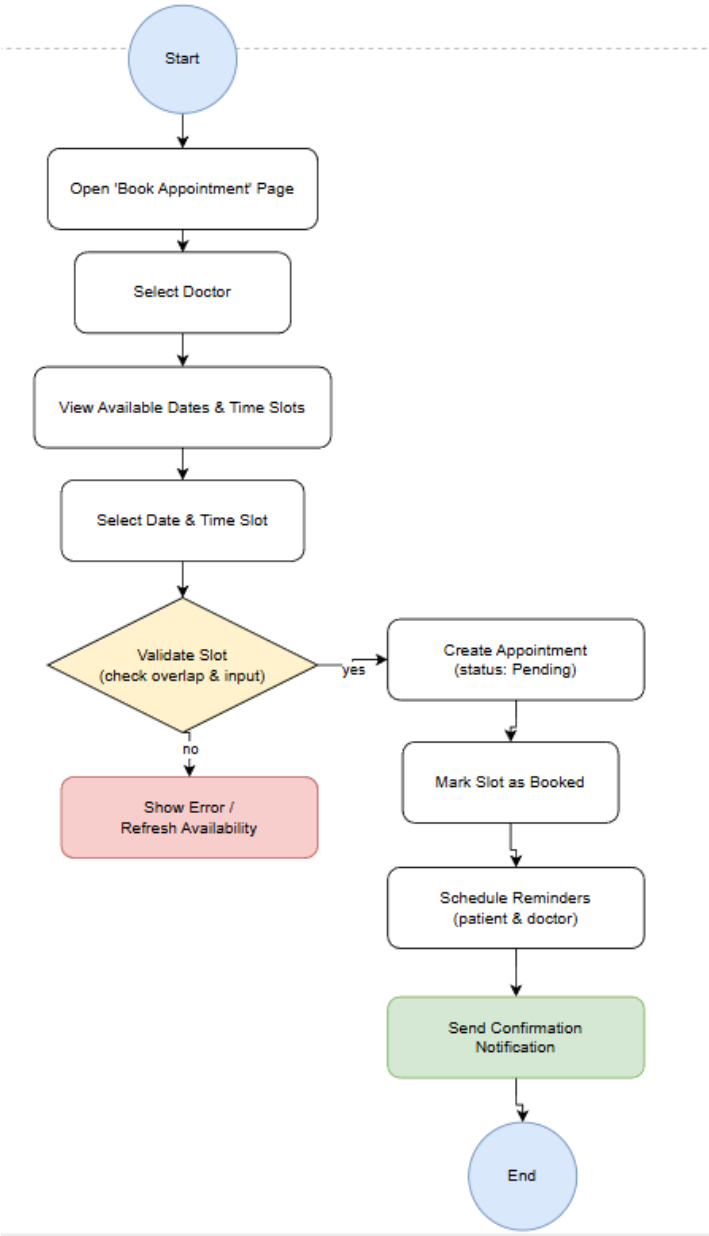


Figure 2.4 : Activity Diagram for Book Appointment

- Patient-Doctor Chat

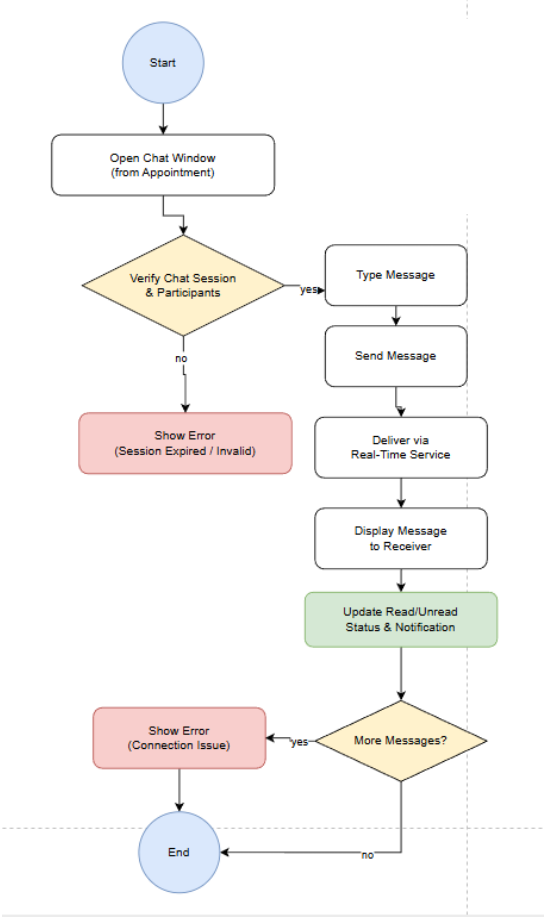


Figure 2.5 : Activity Diagram for Patient-Doctor Chat

- Manage Medical Records

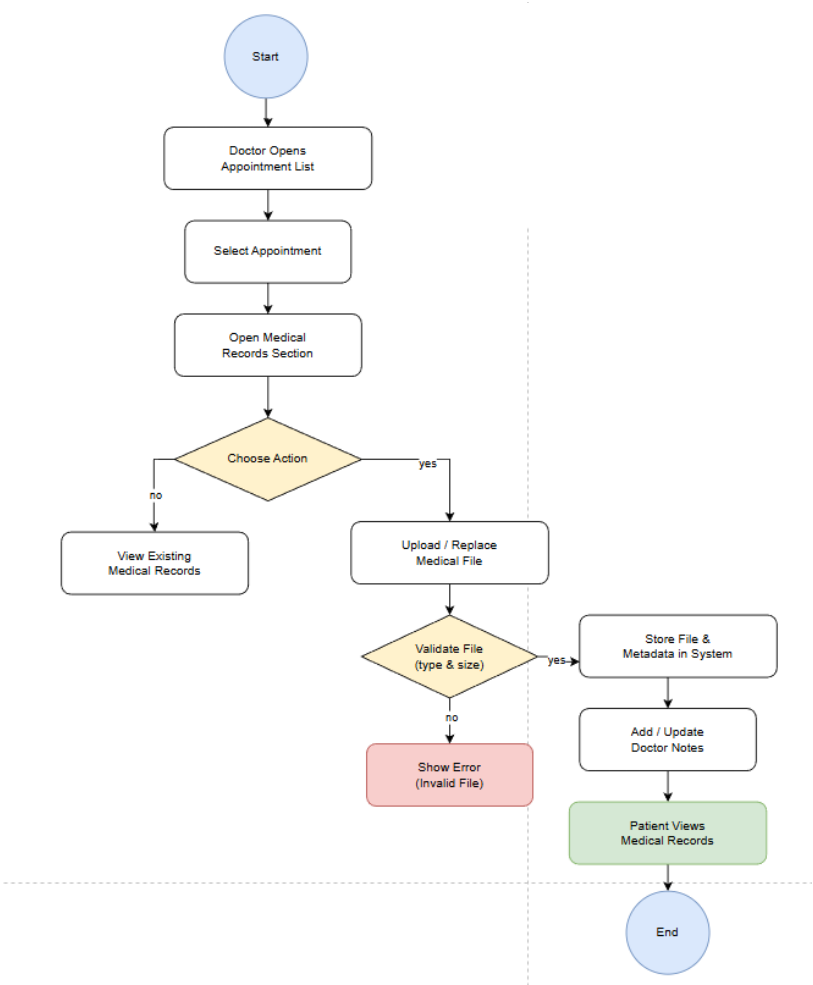


Figure 2.6 : Activity Diagram for Manage Medical Records

- Manage Schedules (Doctor)

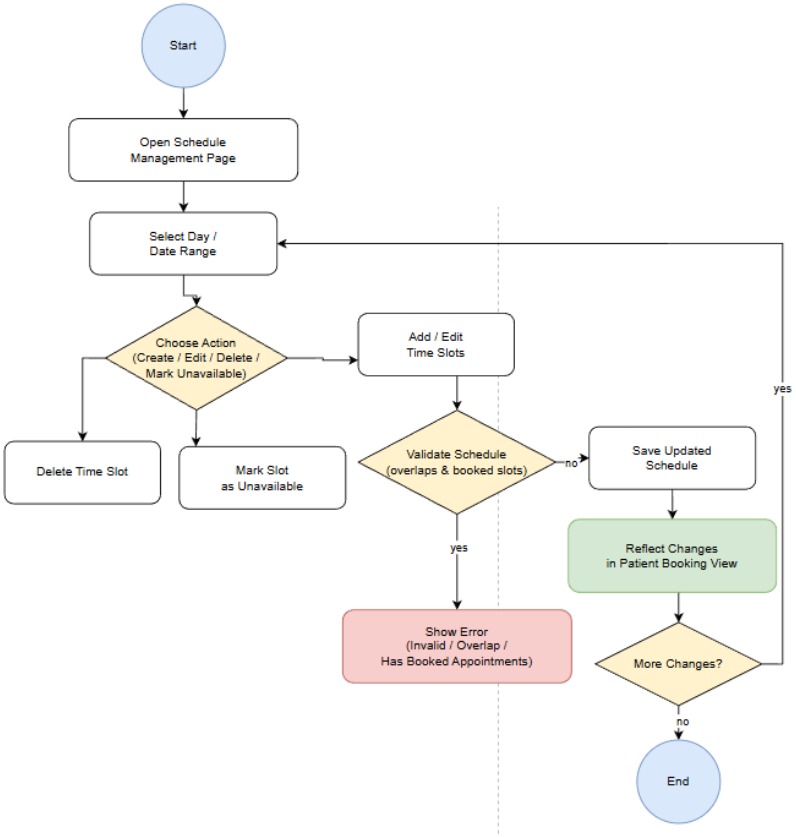


Figure 2.7 : Activity Diagram for Manage Schedules (Doctor)

- Admin User & Appointment Management

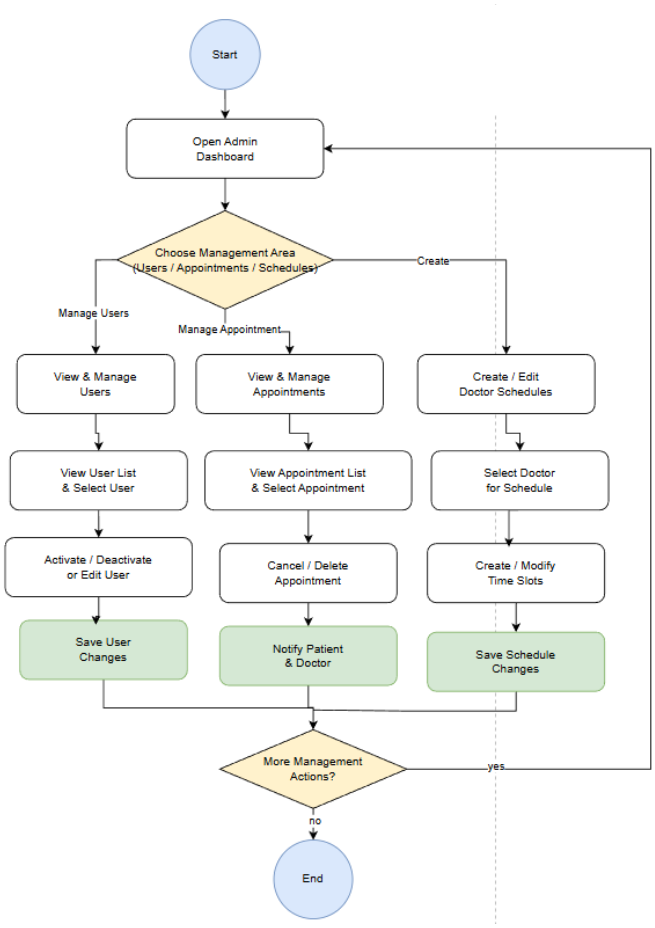


Figure 2.8 : Activity Diagram for Admin User & Appointment Management

- Notification Handling

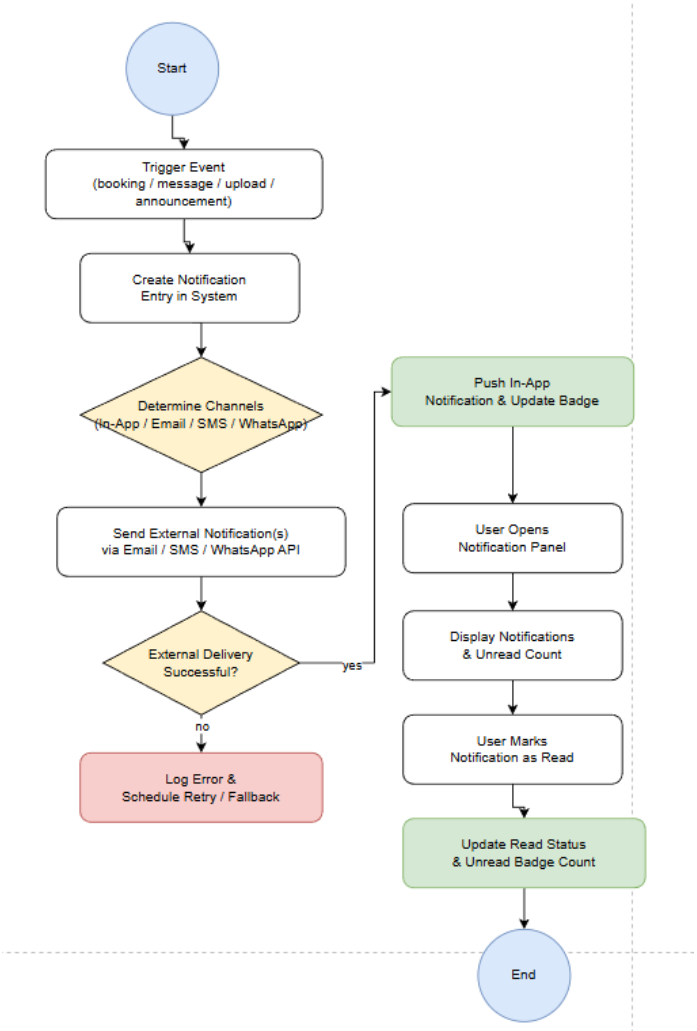


Figure 2.9 : Activity Diagram for Notification Handling

- System Security & Data Management

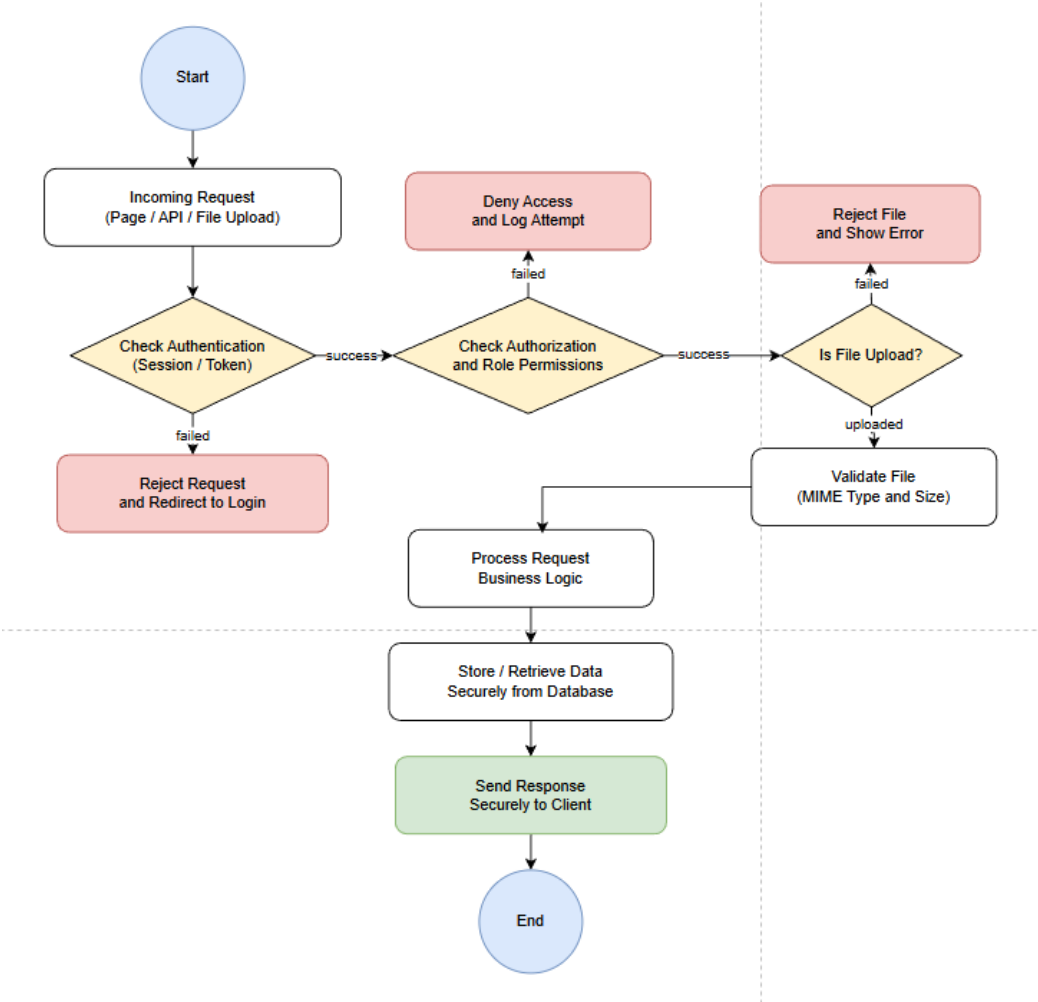


Figure 2.10 : Activity Diagram for System Security & Data Management

2.3.4 Sequence Diagram

- SD01 - User Registration

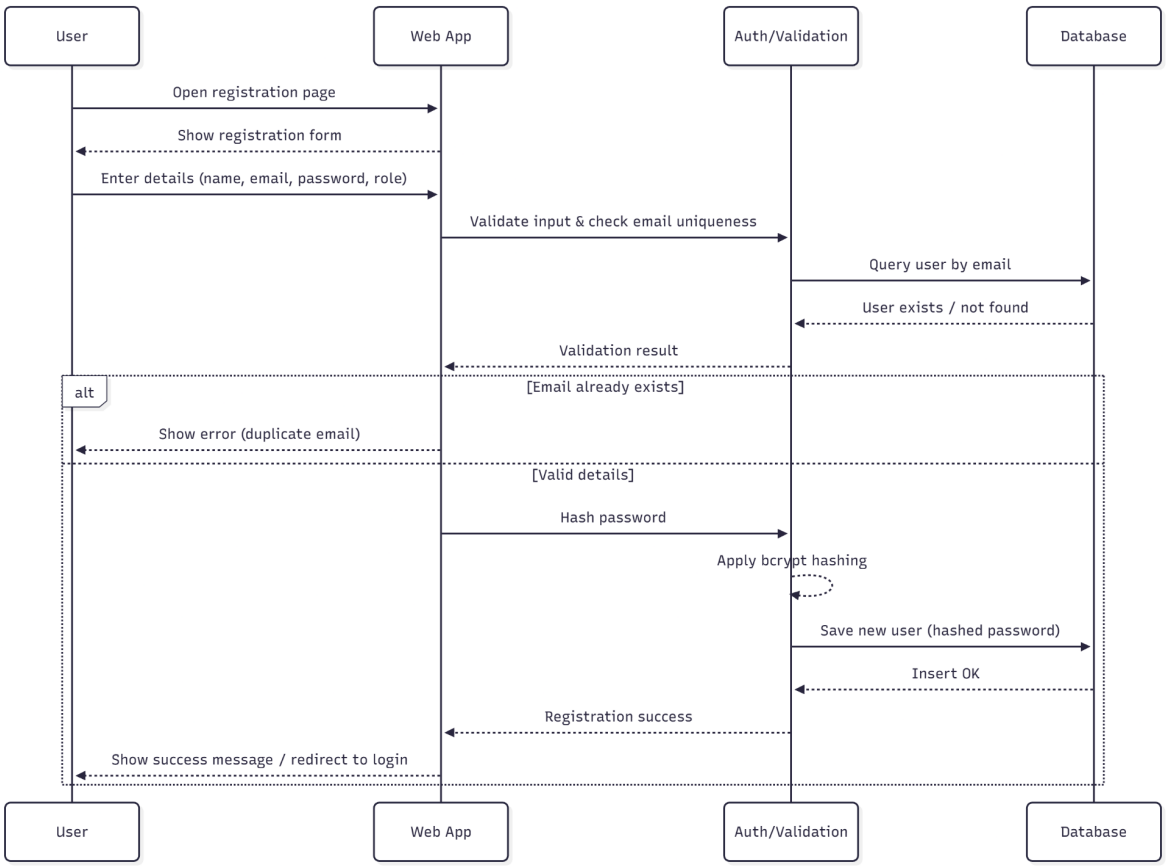


Figure 2.11 : Sequence Diagram for User Registration

- SD02 - User Login & Dashboard Access

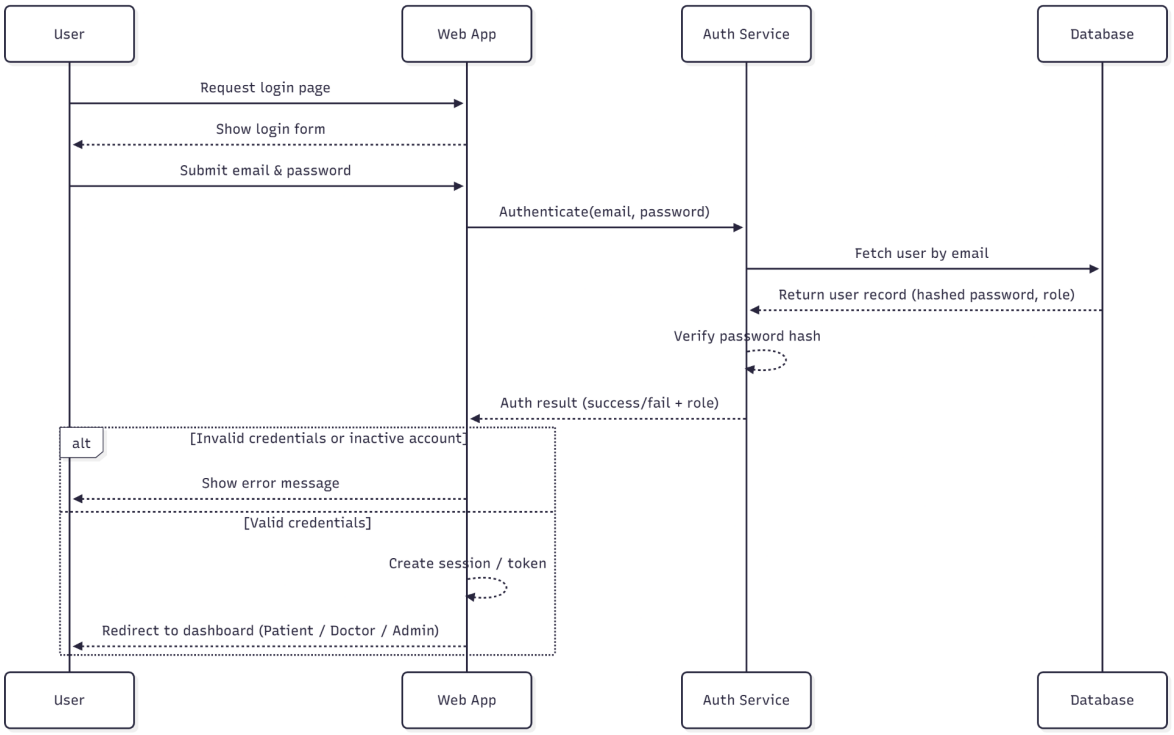


Figure 2.12 : Sequence Diagram for User Login & Dashboard Access

- SD03 - Book Appointment

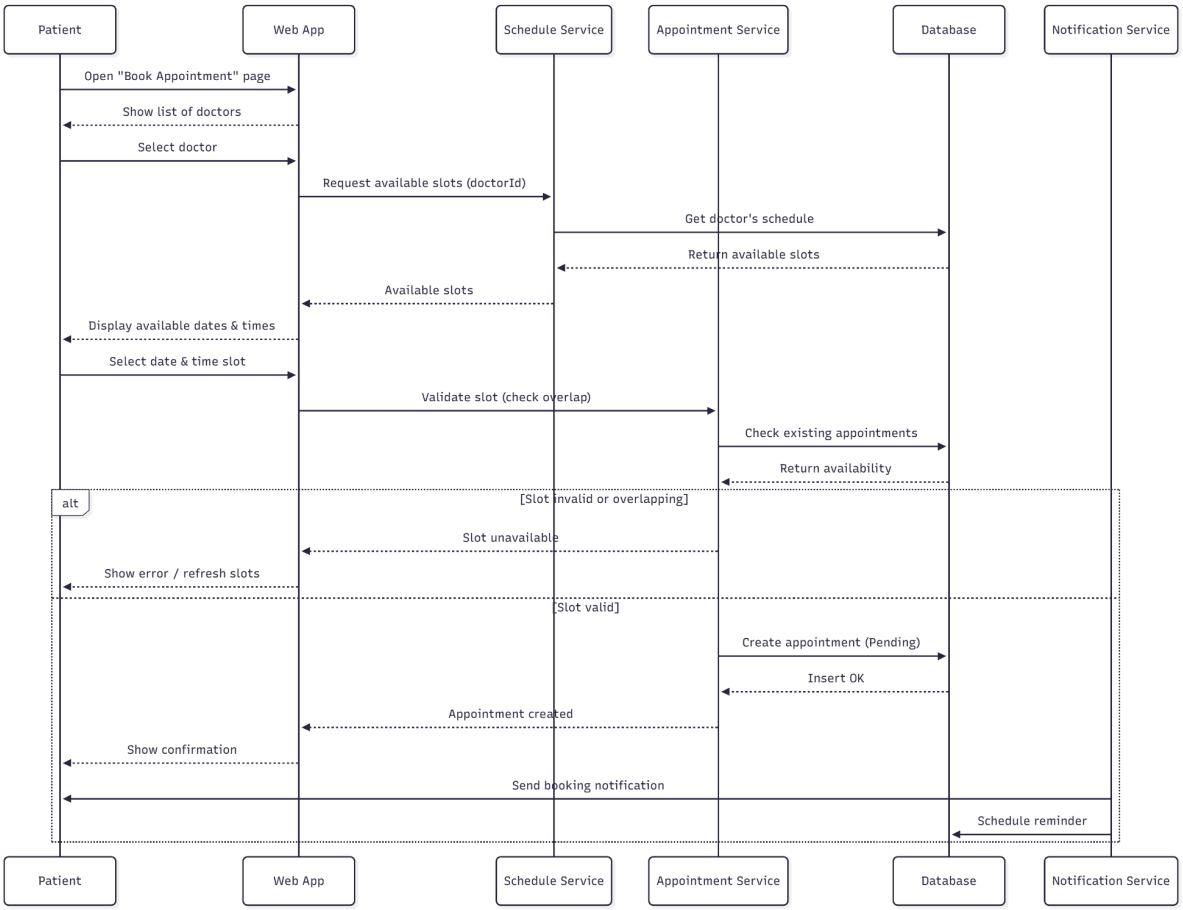


Figure 2.13 : Sequence Diagram for Book Appointment

- SD 04 - Patient–Doctor Chat

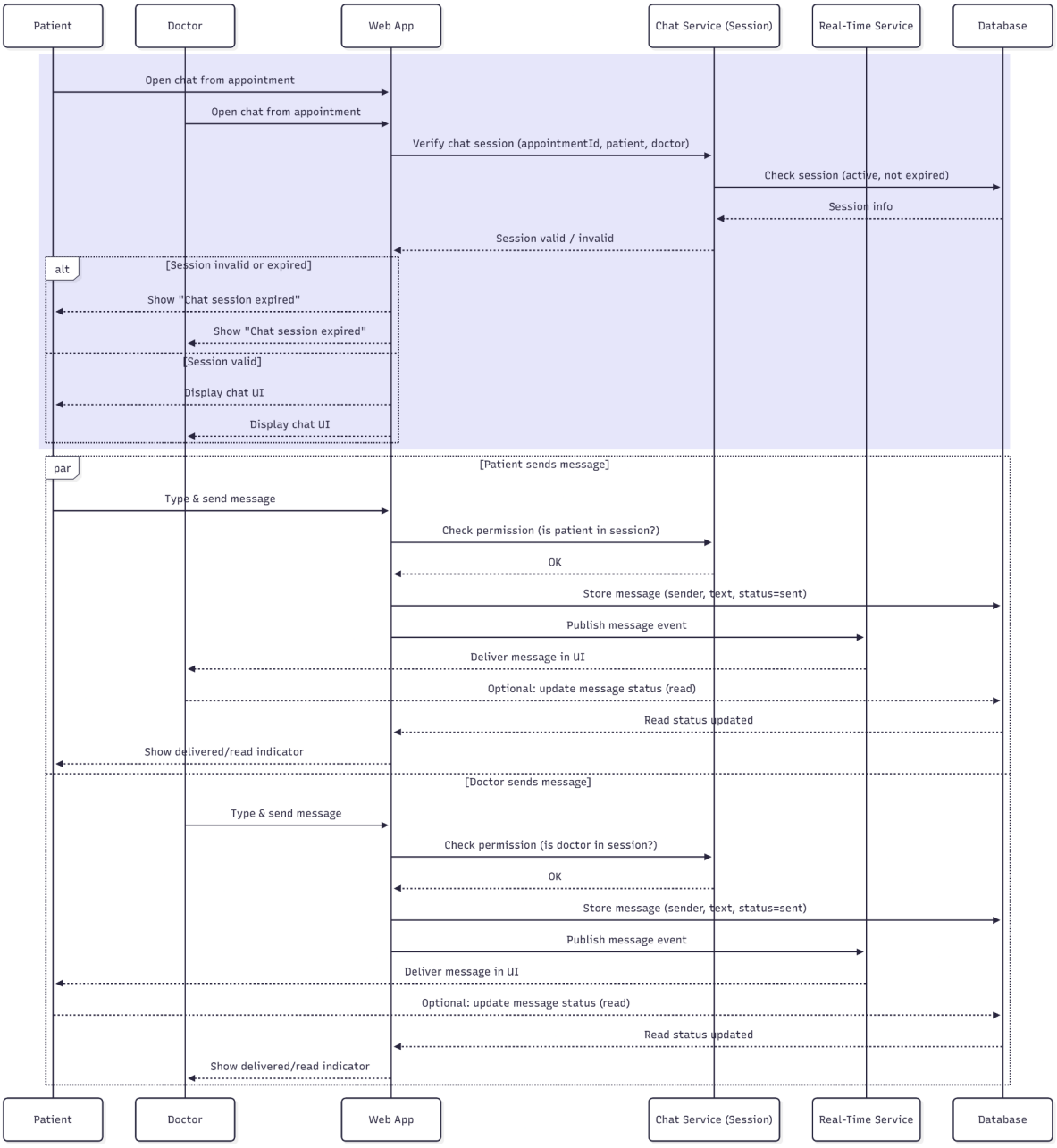


Figure 2.14 : Sequence Diagram for Patient–Doctor Chat

- SD05 - Manage Medical Records

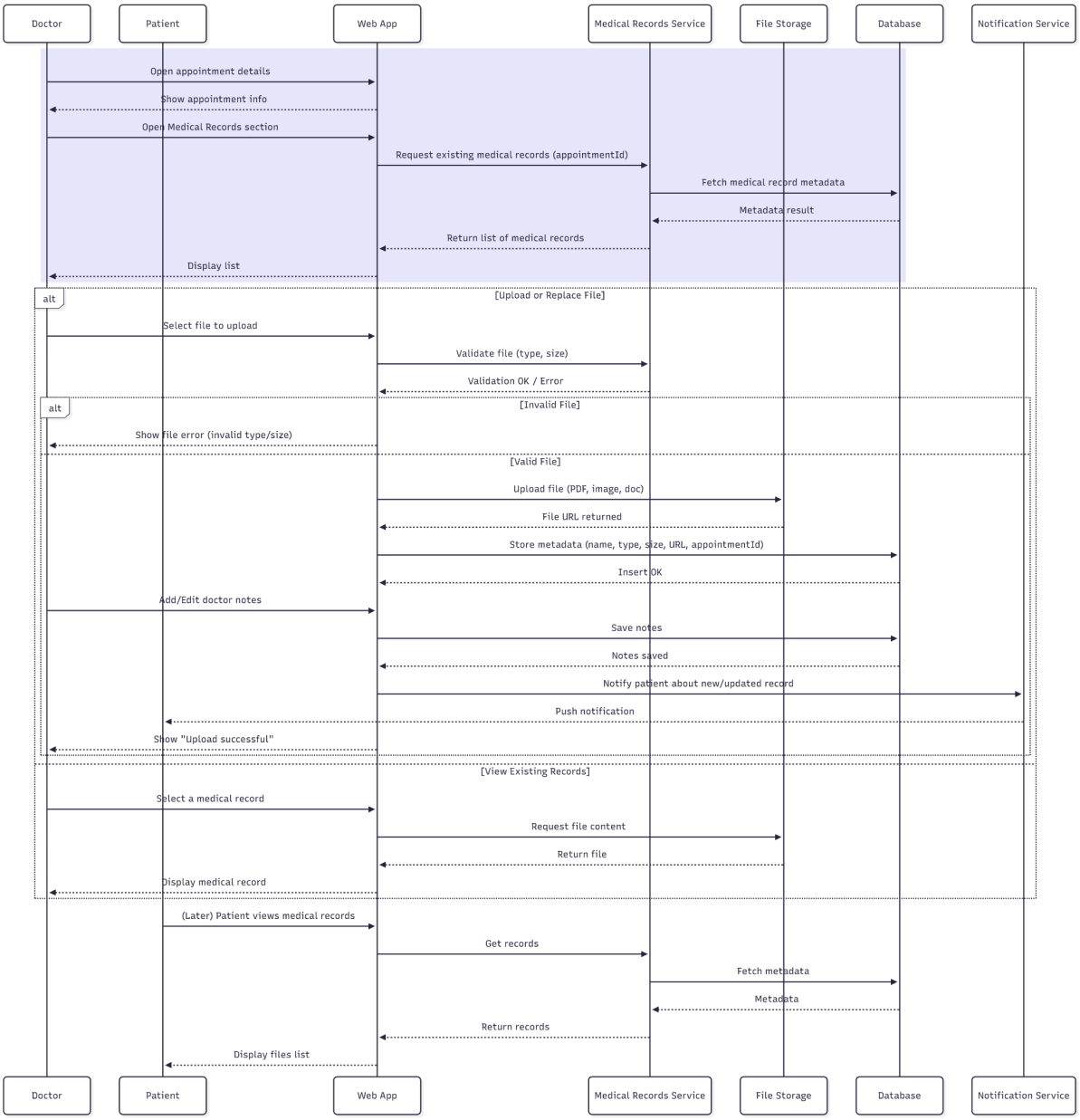


Figure 2.15 : Sequence Diagram for Manage Medical Records

- SD06 - Manage Schedules (Doctor)

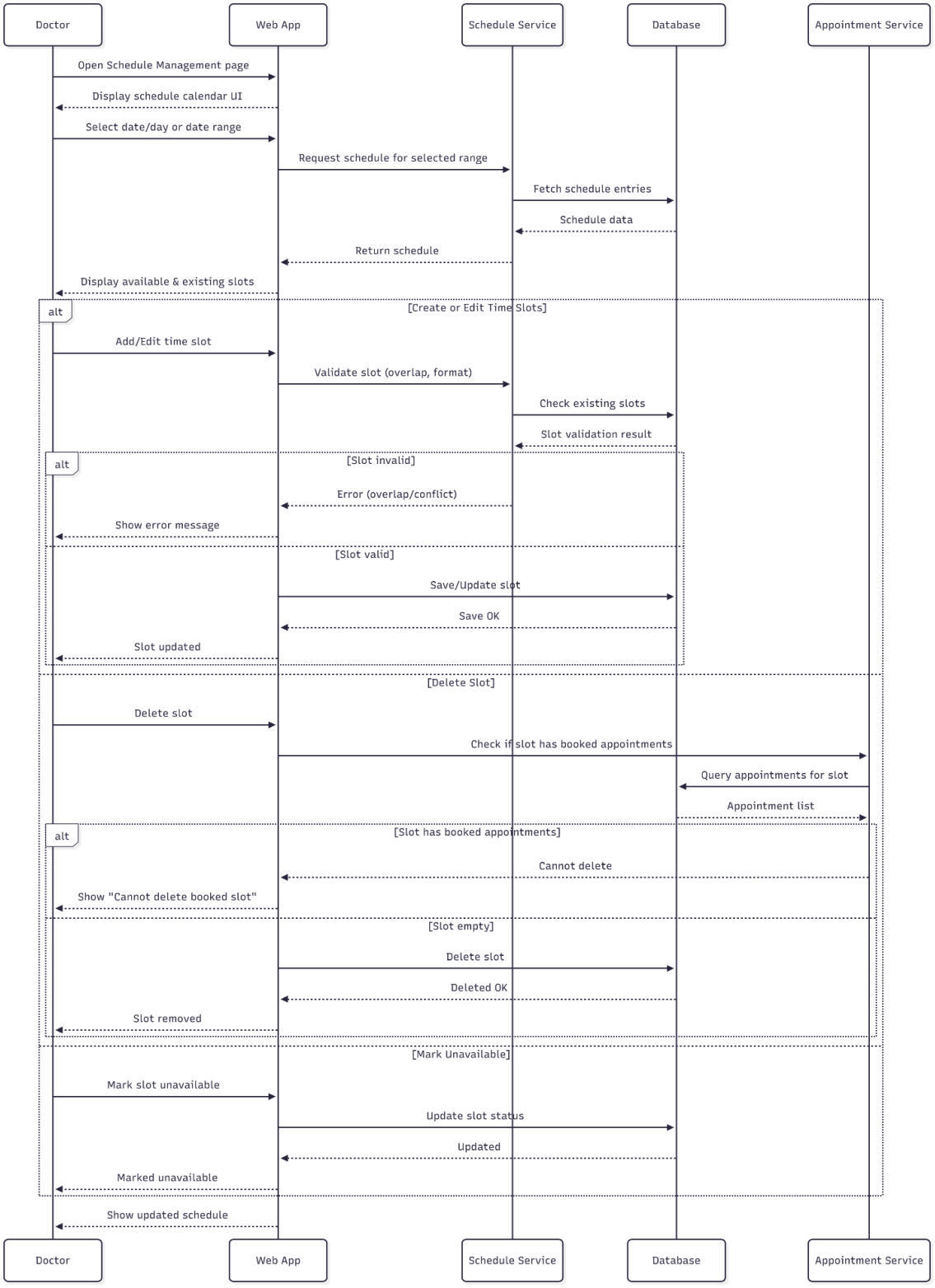


Figure 2.16 : Sequence Diagram for Manage Schedules (Doctor)

- SD07 - Admin User & Appointment Management

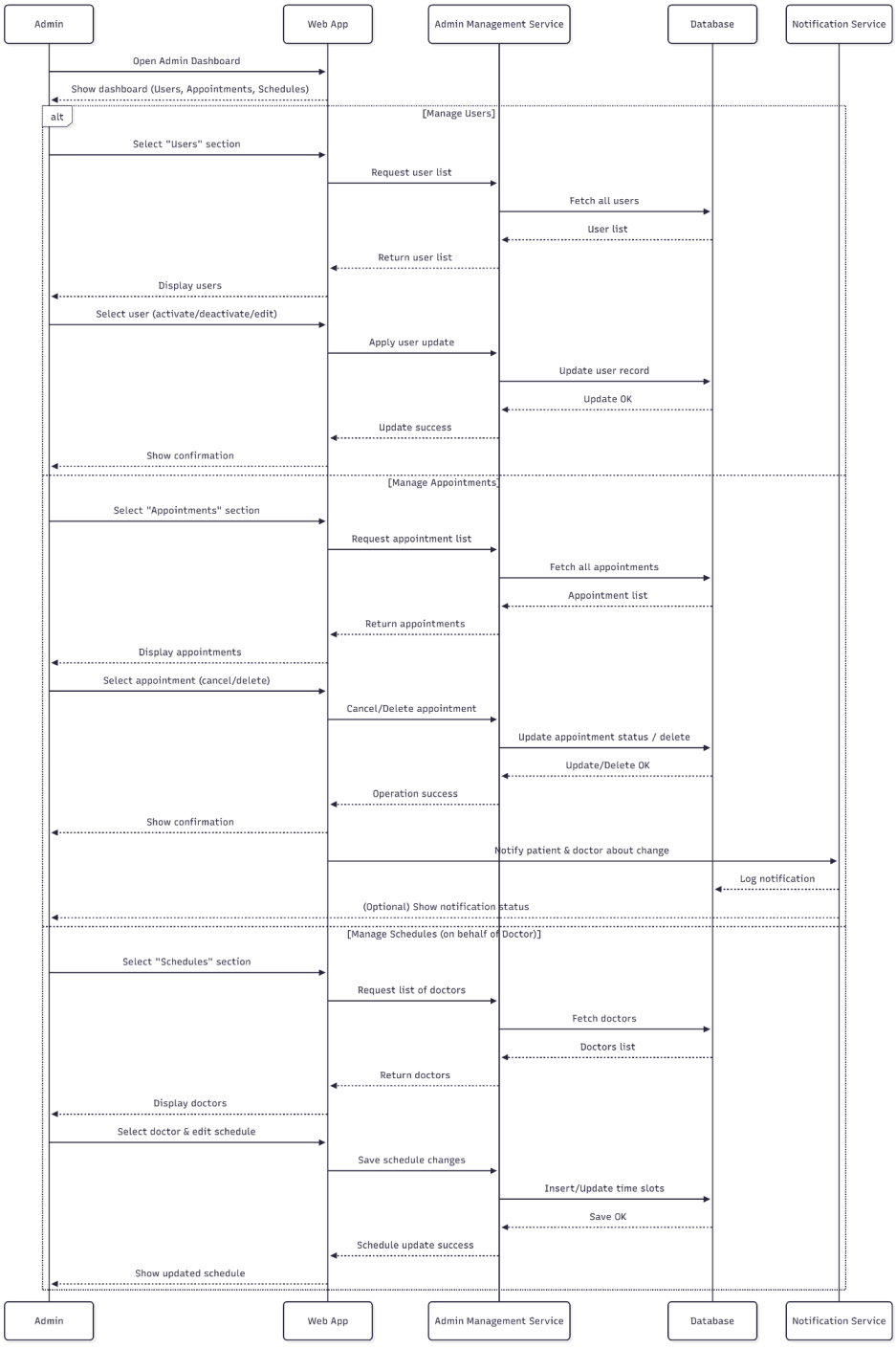


Figure 2.17 : Sequence Diagram for Admin User & Appointment Management

- SD08 - Notification Handling

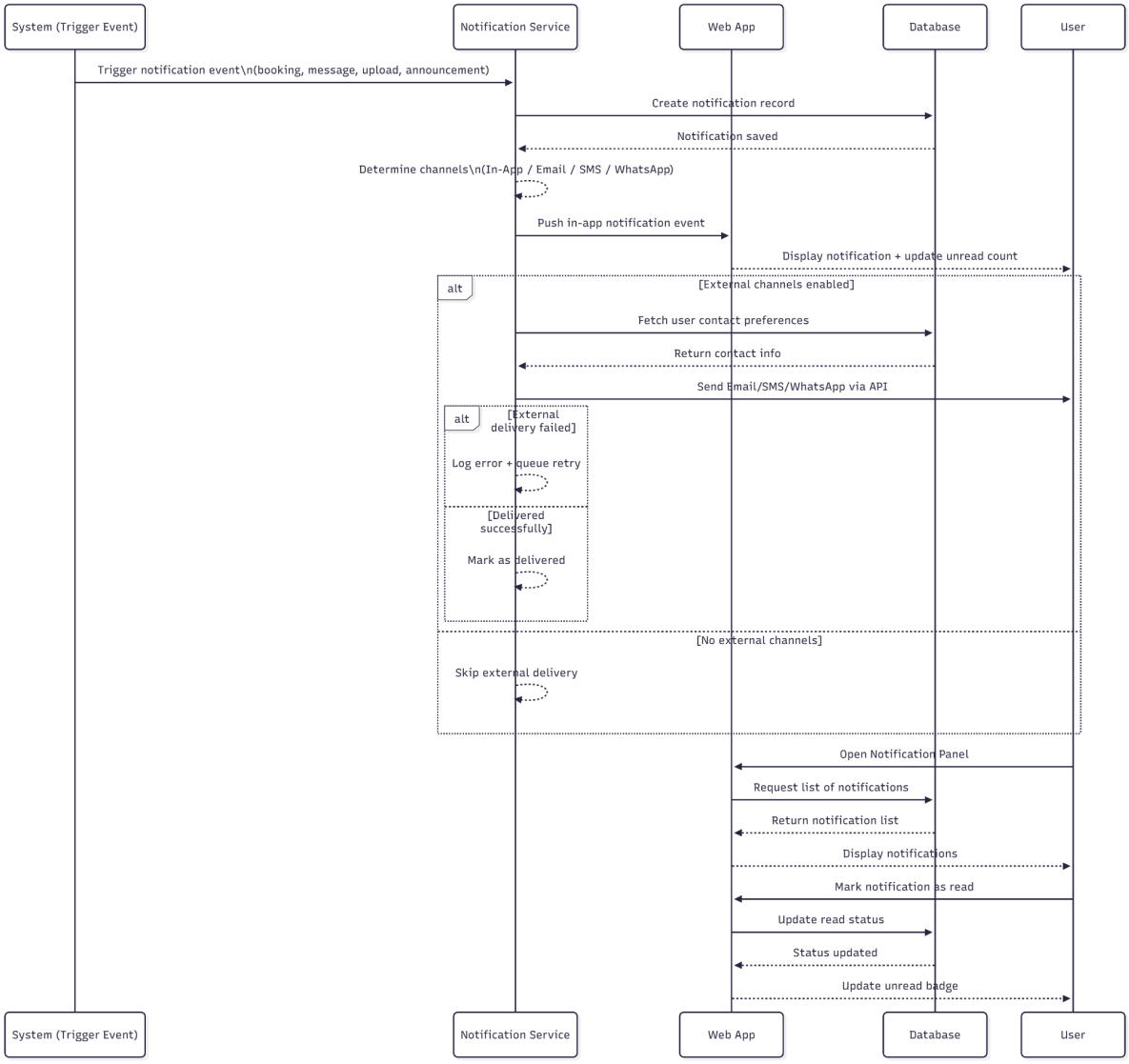


Figure 2.18 : Sequence Diagram for Notification Handling

- SD09 - System Security & Data Management

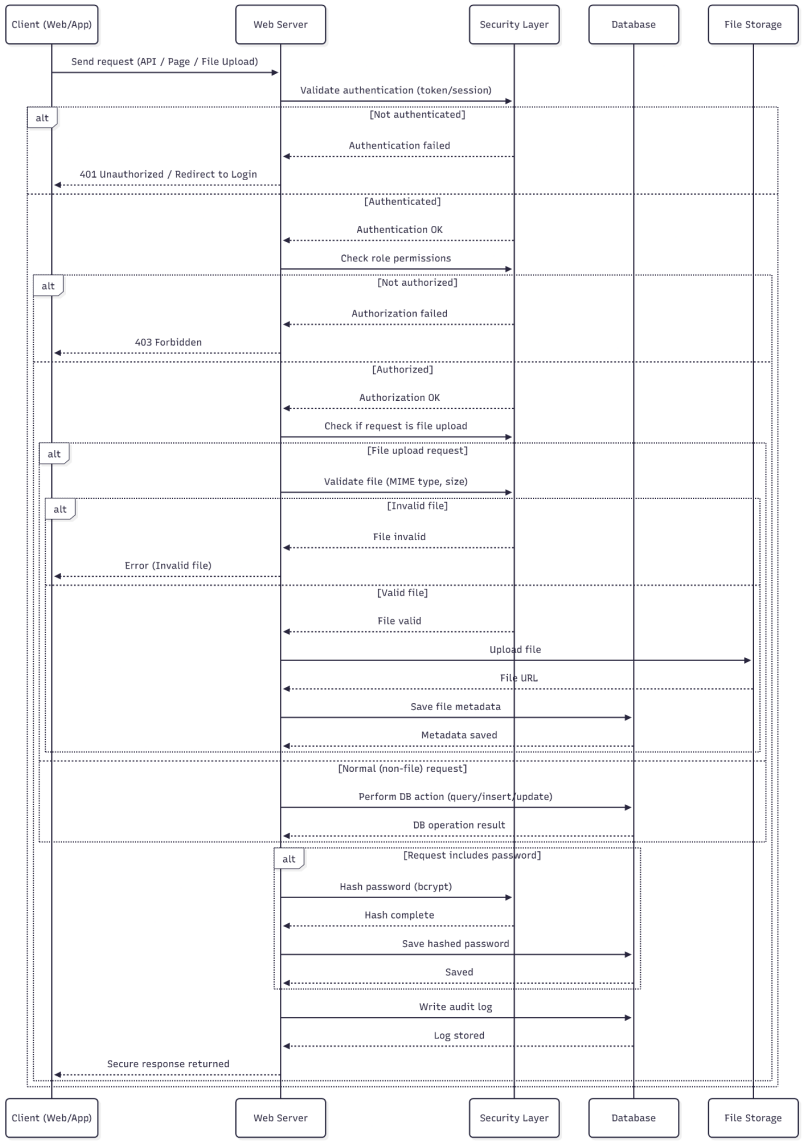


Figure 2.19 : Sequence Diagram for System Security & Data Management

2.3.5 ER Diagram

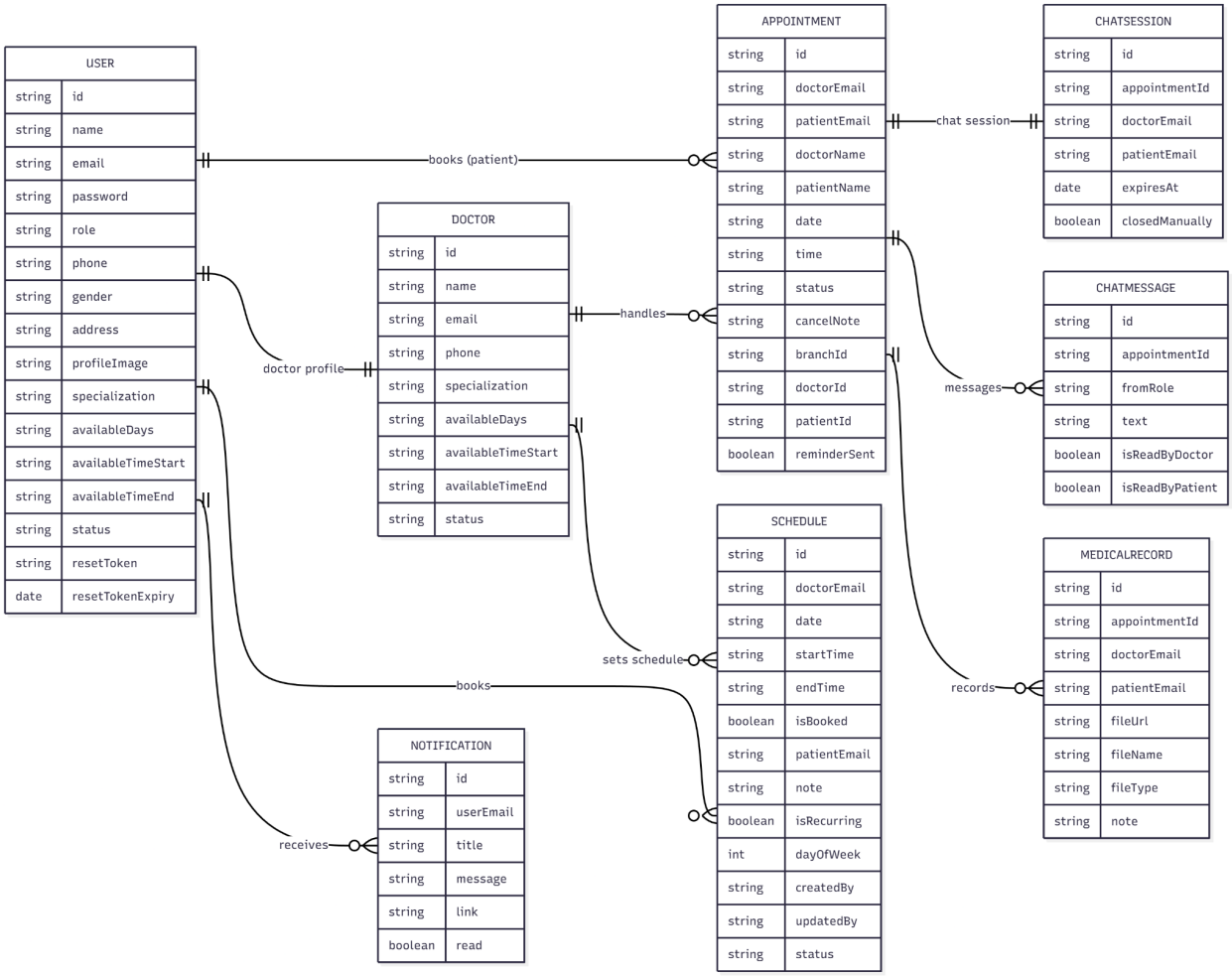


Figure 2.20 : ER Diagram

Chapter 3 : Software Testing

1.1 Testing Features

This section outlines the major features of the Doctor Appointment System that were tested to verify compliance with the functional requirements. Testing focused on ensuring that the workflows for Patients, Doctors, and Administrators operate correctly and reliably across all modules of the system.

Table 3.1 : Testing Features Table

Feature Area	Specific Functionality Tested
User Authentication	User Registration, User Login, Role-Based Access Control, Session Handling, Unauthorized Access Prevention
Appointment Booking (Patient)	View Doctors, View Available Time Slots, Select Slot, Book Appointment, Conflict/Overlap Validation, Appointment Confirmation
Doctor Schedule Management	Create Time Slots, Edit Time Slots, Delete Time Slots, Mark Slot as Unavailable, Overlap Validation, Reflect Changes in Patient View
Real-Time Chat (Patient–Doctor)	Open Chat Session, Verify Session Validity, Send & Receive Messages, Real-Time Delivery, Read/Unread Status Update
Medical Record Management (Doctor & Patient)	Upload Medical Files, File Type & Size Validation, Replace Existing Records, View Medical Records, Metadata Storage
Admin User Management	View User List, Activate/Deactivate User, Edit User Details, Prevent Unauthorized Admin Access

Admin Appointment Management	View All Appointments, Cancel/Delete Appointment, Update Appointment Status, Modify Schedule on Behalf of Doctor
Notification System	In-App Notifications, Real-Time Alerts, Unread Count Update, External Channel Triggers (Email/SMS/WhatsApp), Retry Mechanism
System Security & Data Integrity	Authentication Token Validation, Role-Based Permissions, File Type Restrictions, Secure Data Storage, Error Handling for Protected Routes

1.2 Testing Strategies

- **Unit Testing:**

  Ensured individual components and backend functions (such as appointment validation, schedule generation, and authentication utilities) operated correctly in isolation.

- **Integration Testing:**

  Verified interactions between modules such as booking → notifications, schedule → availability updates, and chat session → real-time messaging service.

- **Manual E2E Testing:**

  Conducted end-to-end testing of fulfill workflows for Patient, Doctor, and Admin roles, together with registration, login, booking appointments, chatting, managing schedules, uploading medical records, and receipt notifications.

1.3System Testing

System Testing was performed to validate that the Doctor Appointment System works as a complete, integrated application.

The goal was to ensure end-to-end functionality across all user roles—Patient, Doctor, and Admin—while verifying that business rules, workflows, and data handling operate correctly under realistic conditions.

Table 3.2 : System Testing

Test Case ID	Feature	Test Steps	Expected Result	Status
TC-01	**(Positive) User Registration**	1. Open registration page. 2. Fill out form with valid name, email, password, and role. 3. Click **Register**.	User account is created, stored in database with hashed password, and redirected to login page.	**Pass**
TC-02	**(Negative) User Registration – Email Exists**	1. Enter an email already registered. 2. Click **Register**.	System shows error: **"Email already in use."**	**Pass**
TC-03	**(Positive) User Login**	1. Enter valid email and password. 2. Click **Log In**.	User is authenticated and redirected to correct dashboard (Patient / Doctor / Admin).	**Pass**
TC-04	**(Negative) User Login – Invalid Password**	1. Enter valid email and incorrect password. 2. Click **Log In**.	System displays **"Invalid credentials"**message.	**Pass**
TC-05	**(Positive) View Doctors**	1. Log in as Patient. 2. Navigate to **Find Doctors** page.	Doctor list loads with names, specialization, and availability.	**Pass**

TC-06	**(Positive) View Available Time Slots**	1. Log in as Patient. 2. Select any doctor.	System displays available dates and time slots without duplicates.	**Pass**
TC-07	**(Positive) Book Appointment**	1. Log in as Patient. 2. Select doctor → choose date & time. 3. Click Book Appointment.	Appointment is created, slot marked booked, and confirmation notification sent.	**Pass**
TC-08	**(Negative) Overlapping Booking**	1. Try booking a slot that was recently booked by another user.	System prevents booking and refreshes available slots.	**Pass**
TC-09	**(Positive) Patient–Doctor Chat**	1. Log in → open active appointment chat. 2. Send a message.	Message delivered in real-time, visible to both users with read/unread status.	**Pass**
TC-10	**(Positive) Upload Medical Record**	1. Log in as Doctor. 2. Open appointment → upload valid file.	File accepted, metadata stored, patient receives notification.	**Pass**
TC-11	**(Positive) View Medical Records**	1. Log in as Patient. 2. Open appointment details.	System displays correct list of uploaded medical records.	**Pass**
TC-12	**(Positive) Manage Schedule (Doctor)**	1. Log in as Doctor. 2. Create / Edit / Delete time slots.	Schedule updates saved and reflected in patient booking view.	**Pass**

TC-13	**(Positive) Admin User Management**	1. Log in as Admin. 2. Activate or deactivate user.	User status updated successfully.	**Pass**
TC-14	**(Positive) Admin Appointment Update**	1. Log in as Admin. 2. Cancel or delete appointment.	Appointment updated and notifications sent.	**Pass**
TC-15	**(Positive) Notification Handling**	Trigger booking/chat/upload actions.	In-app notifications displayed; unread count updates; external channels triggered if enabled.	**Pass**
TC-16	**(Positive) System Security**	1. Attempt accessing protected routes without authentication.	System blocks access and redirects to login.	**Pass**

Chapter 4 : Deployment and Maintenance

4.1 Deployment Overview

Table 4.1 : Deployment Overview

Aspect	Details
Hosting Platform	Vercel (Next.js deployment with auto-builds)
Database	MongoDB Atlas (cloud-hosted NoSQL)
Real-Time Services	Pusher (chat + notification events)
Environment Variables	Secure credentials stored in .env (DB URI, Auth secrets, Pusher keys)
Deployment Type	Continuous Deployment via GitHub → Vercel

4.2 Deployment Steps

Table 4.2 : Deployment Steps

Step	Description
1. Configure Environment	Added MongoDB, NextAuth, Pusher keys in .env.
2. Build Application	Ran production build and validated API routes.
3. Deploy to Vercel	Connected GitHub repo, enabled auto-deploys.

4. Setup Database	Configured MongoDB Atlas cluster and collections.
5. Connect Real-Time Services	Added Pusher app keys and tested live chat & notifications.
6. Final Testing	Verified booking, chat, schedule, uploads in live environment.

4.3 Maintenance Overview

Table 4.3 : Maintenance Overview

Maintenance Type	Activities
Corrective	Bug fixes, API error correction, UI fixes.
Adaptive	Updating dependencies, adjusting to API changes (Pusher/Vercel).
Perfective	UI/UX improvements, performance optimization.
Preventive	Regular backup checks, security patching, log monitoring.

4.4 Monitoring & Backup

Table 4.4 : Monitoring & Backup

Area	Tools / Activities
Performance Monitoring	Vercel Analytics, MongoDB Atlas Performance Dashboard
Error Tracking	Server logs, API logs, Vercel function logs

Backup & Recovery	Automated daily backups via MongoDB Atlas
Real-Time Monitoring	Pusher Dashboard for message delivery status

4.5 Security Maintenance

Table 4.5 : Security Maintenance

Security Task	Description
Access Control Review	Ensure roles (Admin/Patient/Doctor) are correctly enforced.
API Key Rotation	Regenerate Pusher / DB credentials if needed.
HTTPS Enforcement	All communications secured using SSL.
File Validation	Validate uploaded medical files (type & size).

Chapter 5 : User Manual

This user manual provides clear, step-by-step instructions for installing, configuring, and running the Smart Doctor Appointment System in a local development environment. The guide is intended for developers, testers, and technical users who want to set up the full project (frontend + backend + database + real-time services) on their own machine.

The manual covers environment setup, project installation, API configuration, and database connectivity.

5.1 Environment Setup

5.1.1 Install Visual Studio Code

1. Visit the official Visual Studio Code website:
   https://code.visualstudio.com/download
2. Select the installer suitable for your operating system (Windows, macOS, Linux).
3. Download and complete the installation following the on-screen instructions.

5.1.2 Install Node.js

1. Open the Node.js download page:
   https://nodejs.org/en/download
2. Download the LTS version recommended for most users.
3. Run the installer and complete the setup.
4. Confirm installation by running in terminal:

```
bash
node -v
npm -v
```

5.1.3 Install MongoDB (Optional for Local DB)

If you plan to use local MongoDB instead of MongoDB Atlas:

1. Go to: https://www.mongodb.com/try/download/community
2. Choose Latest Version.

3. Download the installer and run it.
4. Select Complete Setup.
5. Finish installation and start the MongoDB service.

(If using MongoDB Atlas, skip this step.)

5.2 Setup Frontend & Backend

5.2.1 Pull Project from GitHub

bash
https://github.com/jubail65/doctor-appointment-system

5.2.2 Install Dependencies

Open terminals in both folders:

bash
npm install

5.2.3 Create Environment Files (.env.local)

Create a .env.local file inside the project root.

Example .env.local for Doctor Appointment System:

bash
MONGODB_URI="mongodb+srv://doctor_admin:mypassword123@cluster0.xlfmgus.mongodb.net/doctor-appointment" NEXTAUTH_SECRET="N09lNKOwjw1/9GAMtcqgwnKpbcB30UAWvoMkZvcanOo=" NEXTAUTH_URL="http://localhost:3000"

```
NEXT_PUBLIC_PUSHER_KEY=1d5dbb3b7c050ab6e2f3
NEXT_PUBLIC_PUSHER_CLUSTER=ap2
PUSHER_APP_ID=2070563
PUSHER_KEY=1d5dbb3b7c050ab6e2f3
PUSHER_SECRET=5484e26c97a4b2cec64a
PUSHER_CLUSTER=ap2
PUSHER_USE_TLS=true
```

Required values include:

- MongoDB Atlas connection string
- NextAuth secret
- Pusher real-time API keys
- Application URL

5.2.4 Start the Development Server

```
bash
npm run dev
```

This automatically starts API routes

```
bash
http://localhost:3000
```

5.2.5 Troubleshooting Common Issues

Frontend not loading or API showing errors?

1. Open the API configuration file (example):

```
bash
doctor-appointment-system/src/lib/axios.js   (or api.js)
```

2. Check the API base URL:

```
bash
baseURL: "http://localhost:3000/api"
```

3. Update the URL if using separate backend.
4. Save changes (Ctrl + S) and restart:

```
bash
npm run dev
```

5.3 Connect to MongoDB Atlas (Remote Database)

5.3.1 Open MongoDB Atlas

1. Visit: https://www.mongodb.com/cloud/atlas
2. Log in or create an account.

5.3.2 Create Cluster

1. Click Build a Database.
2. Choose the free M0 cluster.
3. Select region and create cluster.

5.3.3 Whitelist IP & Get Connection URL

1. Go to Network Access → Add IP: 0.0.0.0/0 (developer mode).
2. Go to Database → Connect.
3. Choose Connect Using Connection String.
4. Copy the connection URL:

```
bash
mongodb+srv://<username>:<password>@cluster.mongodb.net/?retryWrites=true&w=majority
```

5.4 Smart Doctor Appointment System Interface

5.4.1 Login User - All User

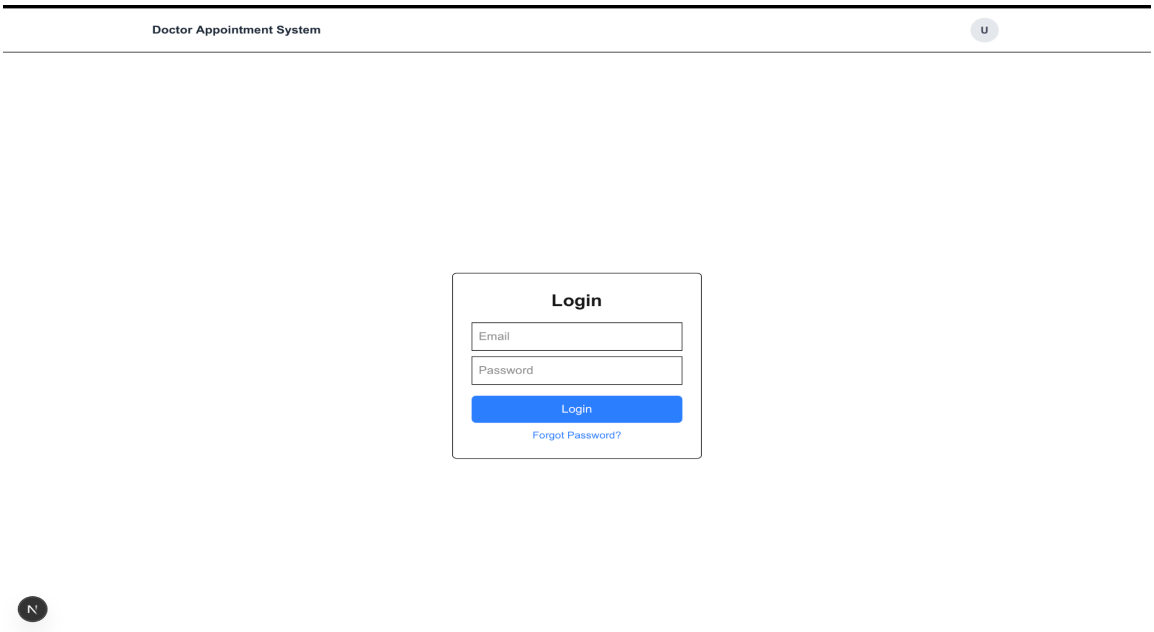


Figure 5.1 : Login User

5.4.2 Register User - All User

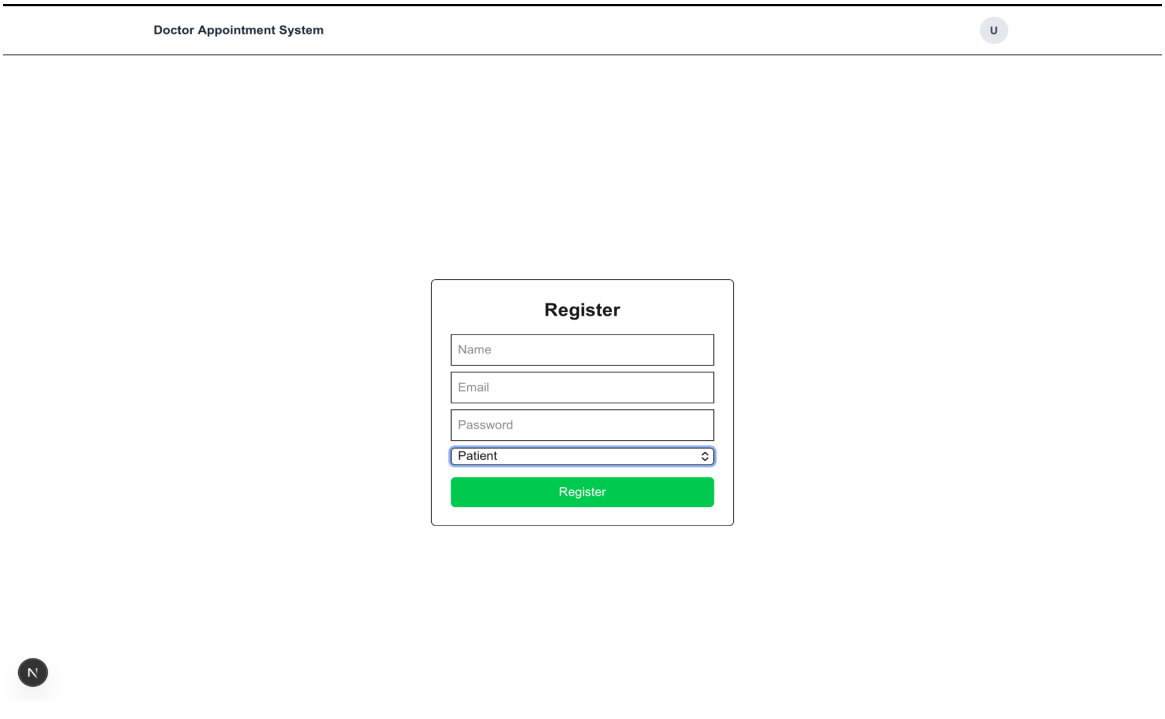


Figure 5.2 : Register User

5.4.3 Dashboard - Admin

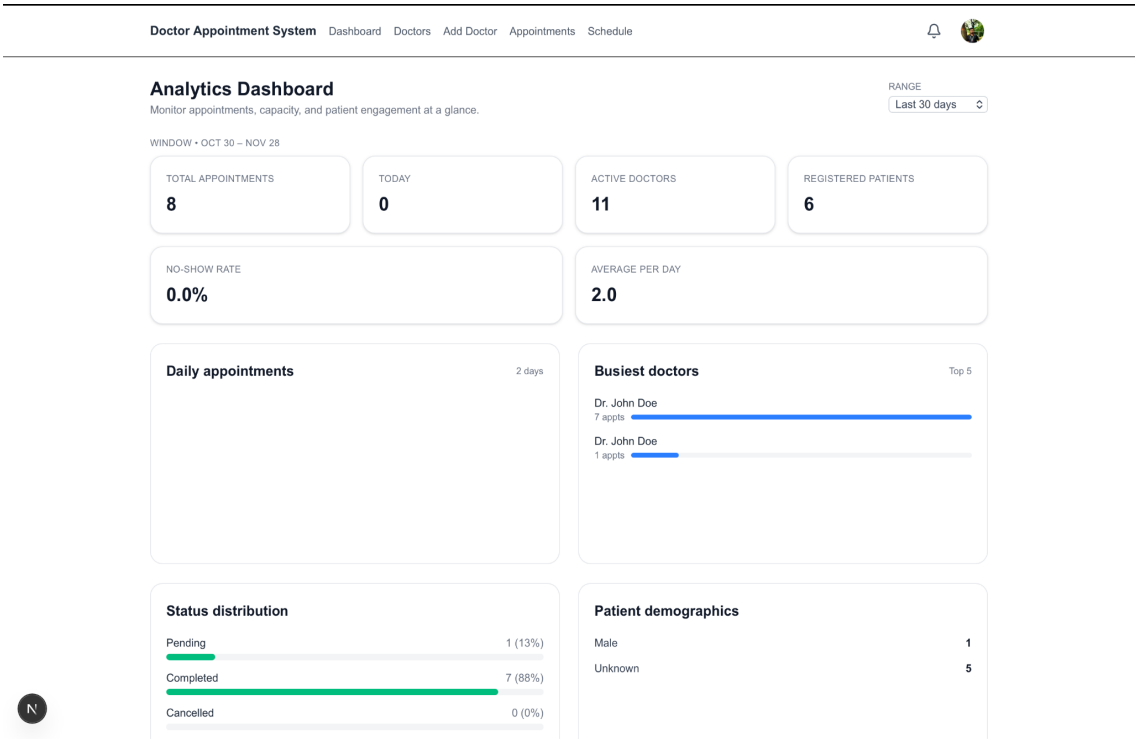


Figure 5.3 : Dashboard – Admin

5.4.4 Doctors List - Admin

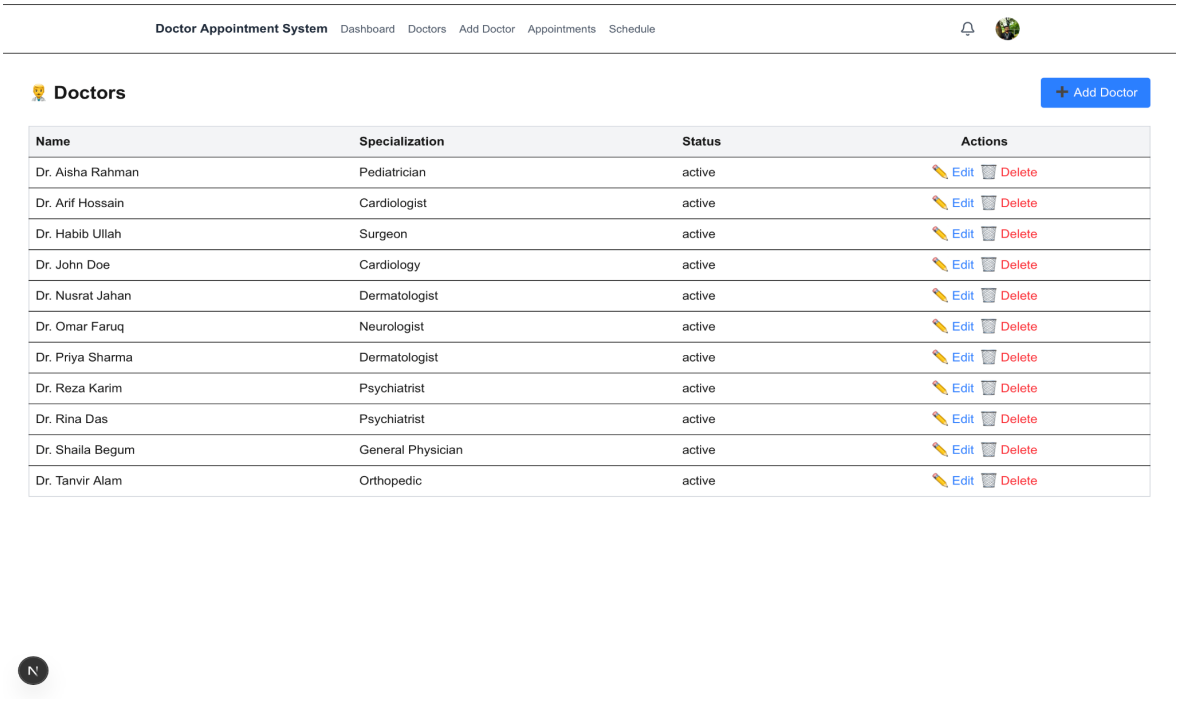


Figure 5.4 : Doctors List – Admin

5.4.5 Add New Doctors - Admin

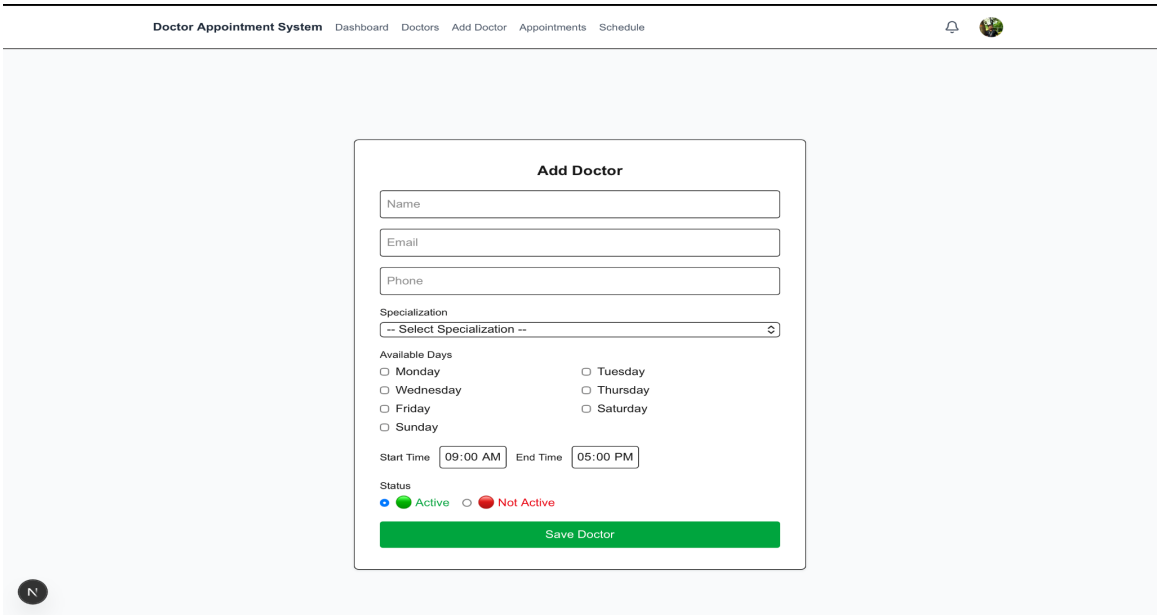


Figure 5.5 : Add New Doctors - Admin

5.4.6 All Appointments List - Admin

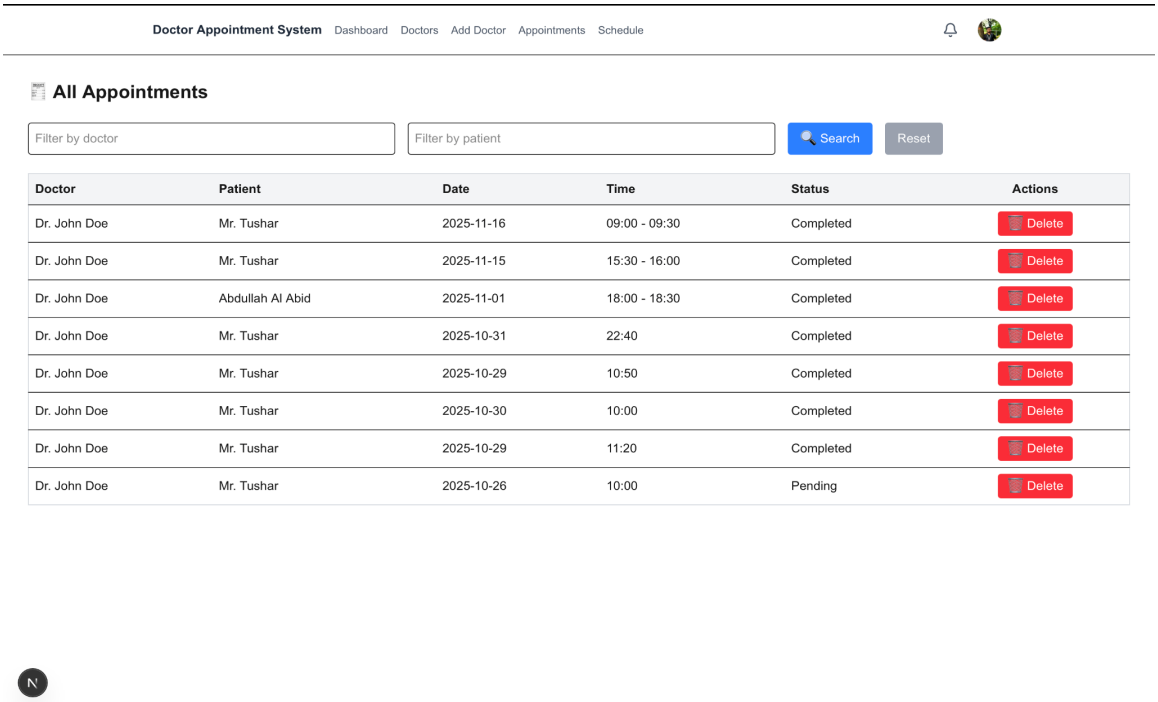


Figure 5.6 : All Appointments List - Admin

5.4.7 Manage Doctor Schedules - Admin

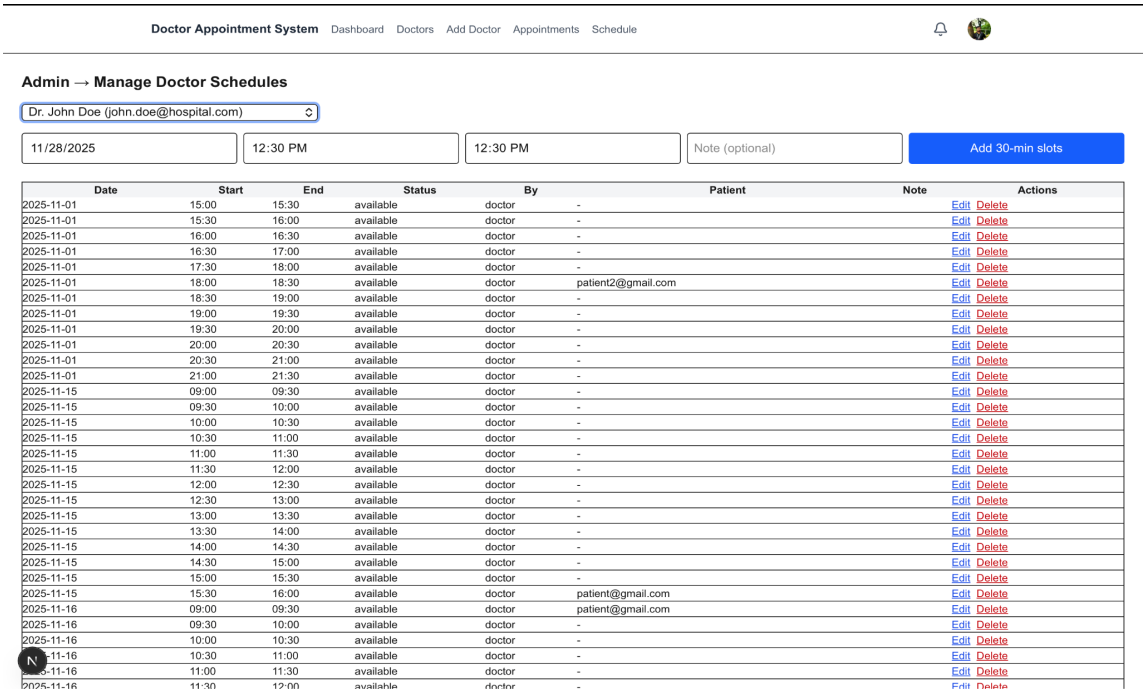


Figure 5.7 : Manage Doctor Schedules - Admin

5.4.8 Profile Manage - Admin

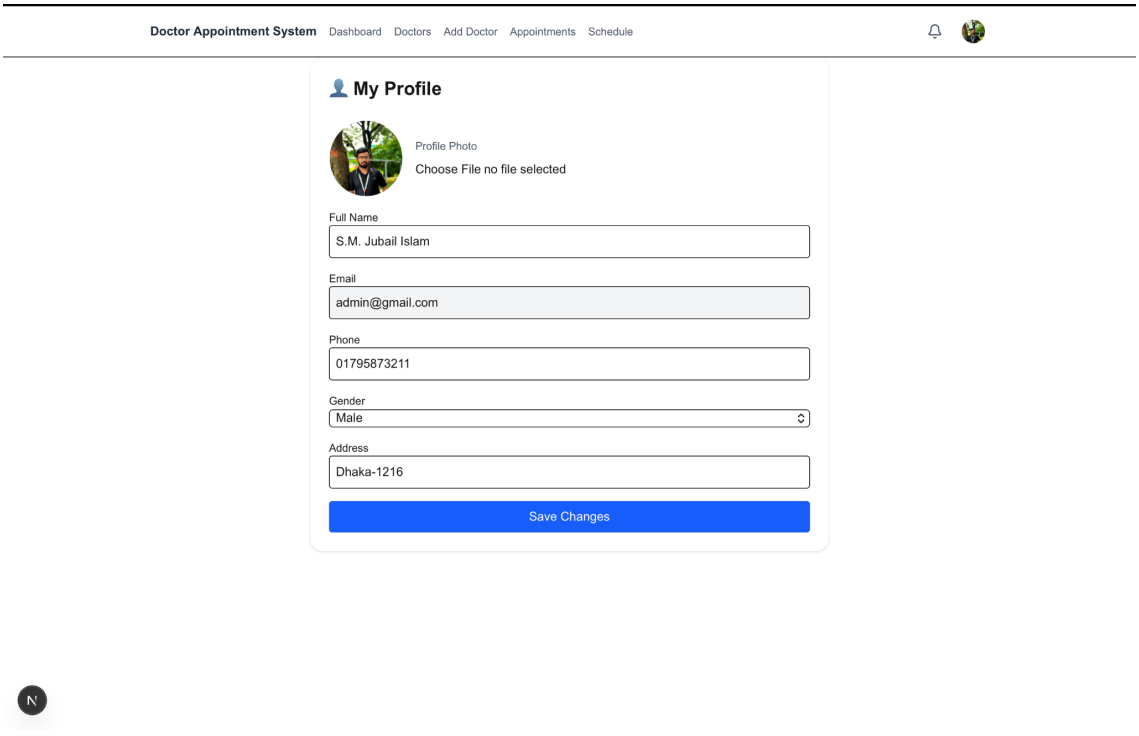


Figure 5.8 : Profile Manage - Admin

5.4.9 Dashboard - Doctor

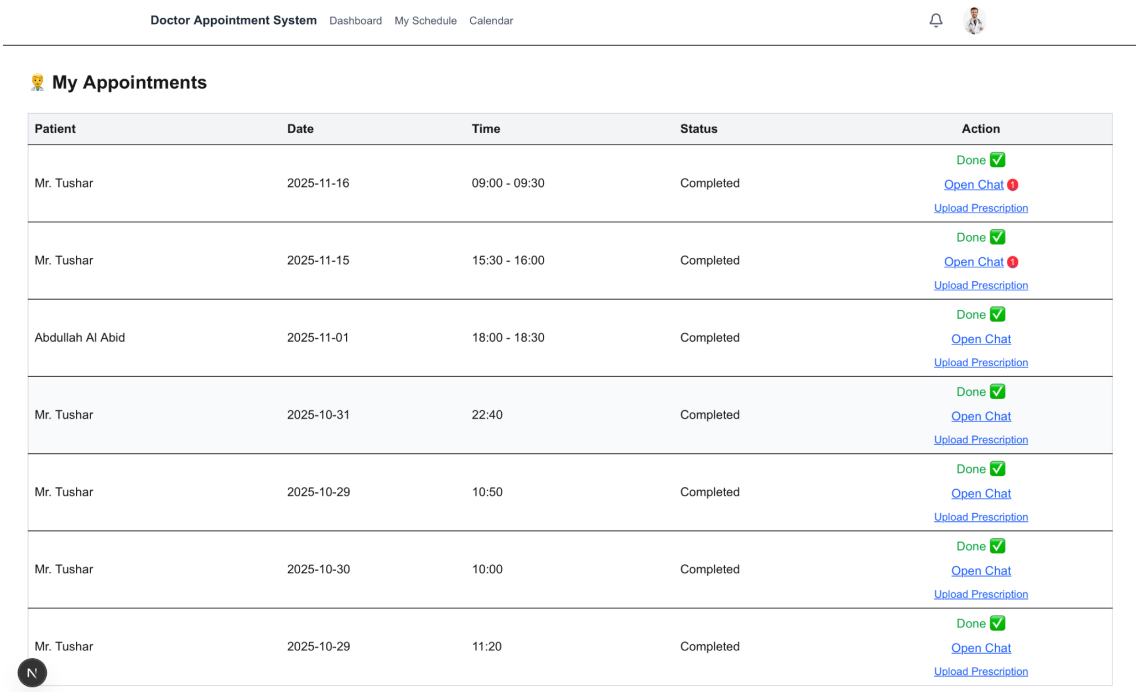


Figure 5.9 : Dashboard - Doctor

5.4.10 Chat with Patient - Doctor

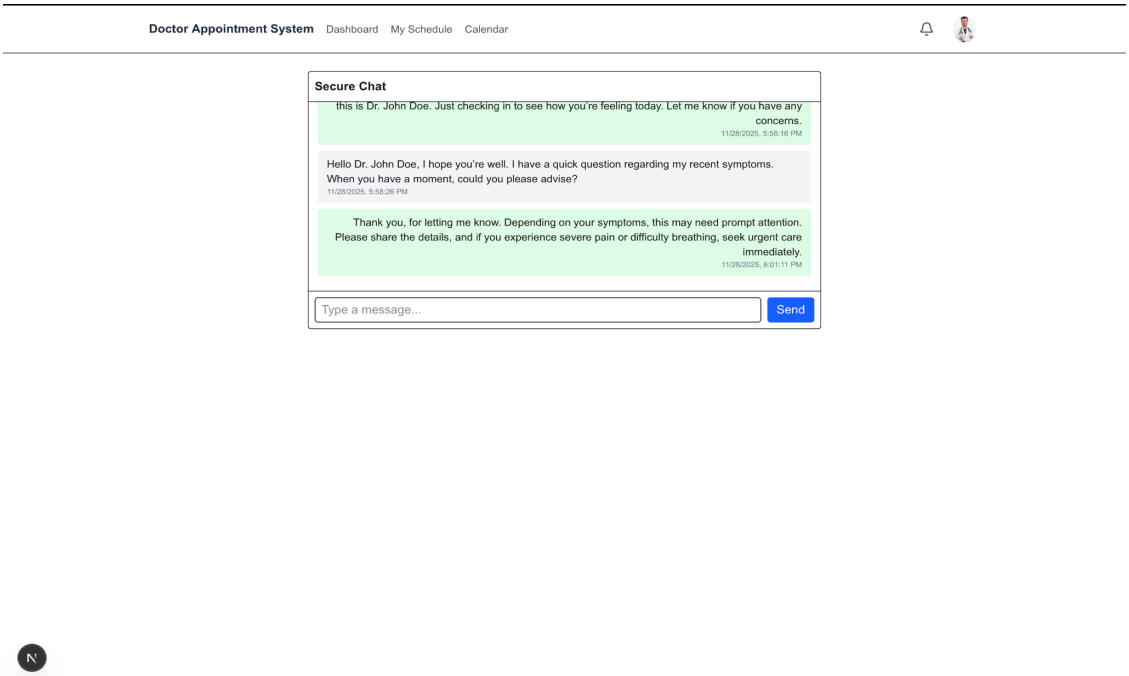


Figure 5.10 : Chat with Patient - Doctor

5.4.11 Upload Prescriptions / Reports - Doctor

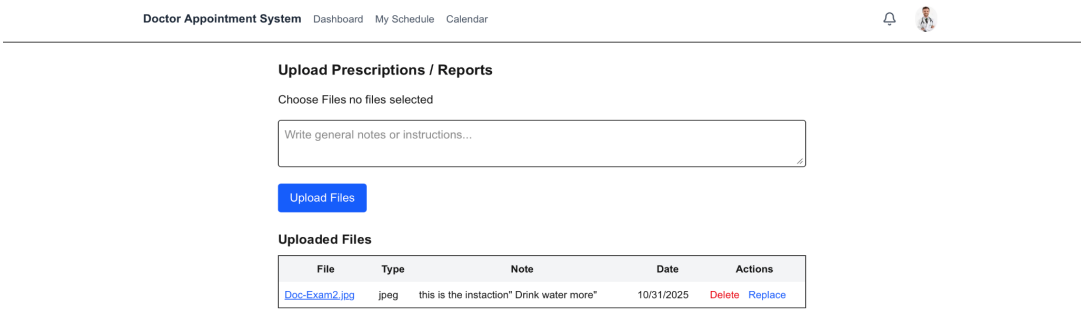


Figure 5.11 : Upload Prescriptions / Reports - Doctor

5.4.12 Manage Schedule - Doctor

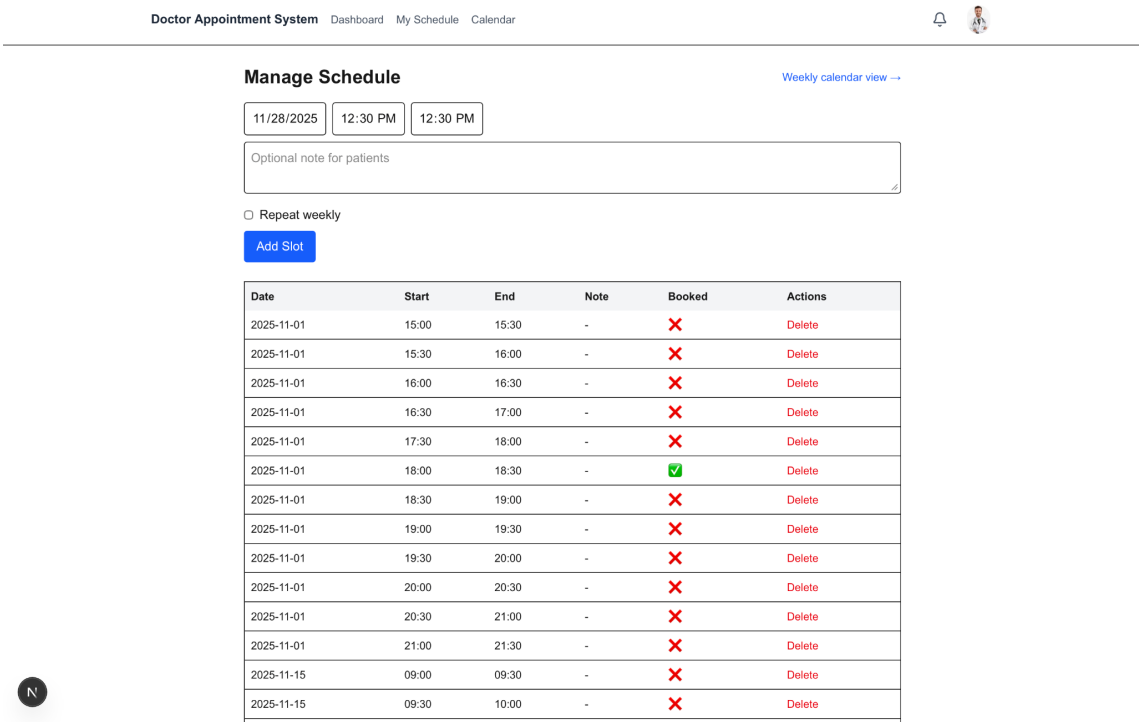


Figure 5.12 : Manage Schedule - Doctor

5.4.13 Manage Calendar View - Doctor

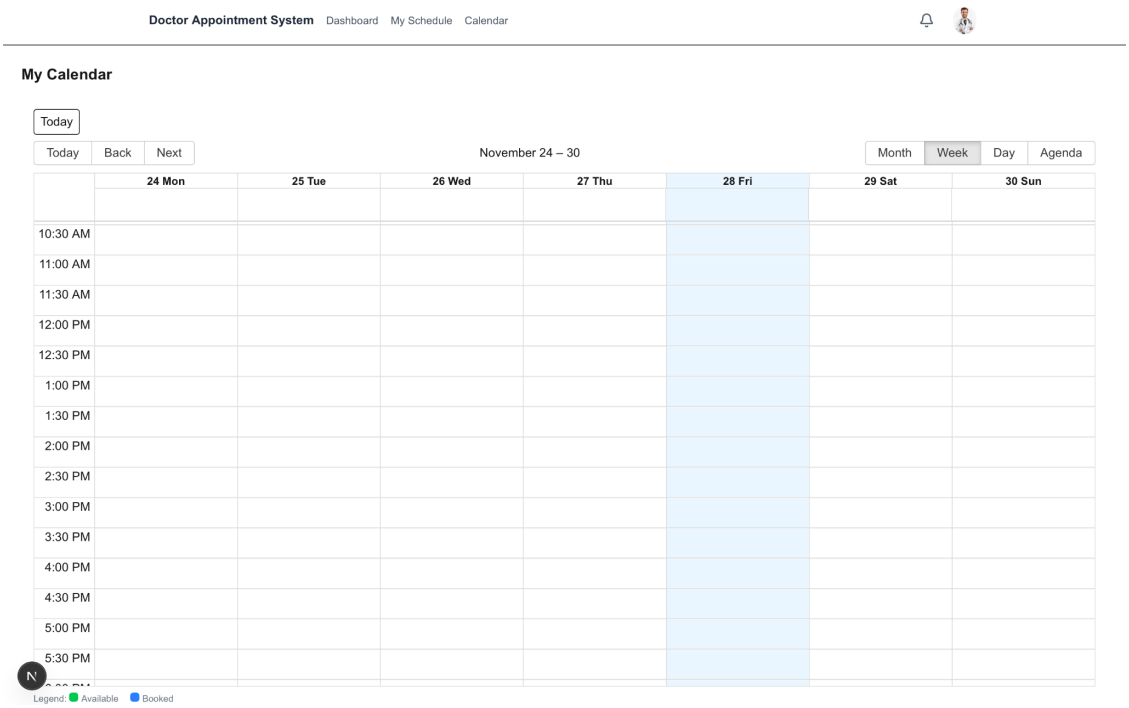


Figure 5.13 : Manage Calendar View - Doctor

5.4.14 Manage Profile Manage - Doctor

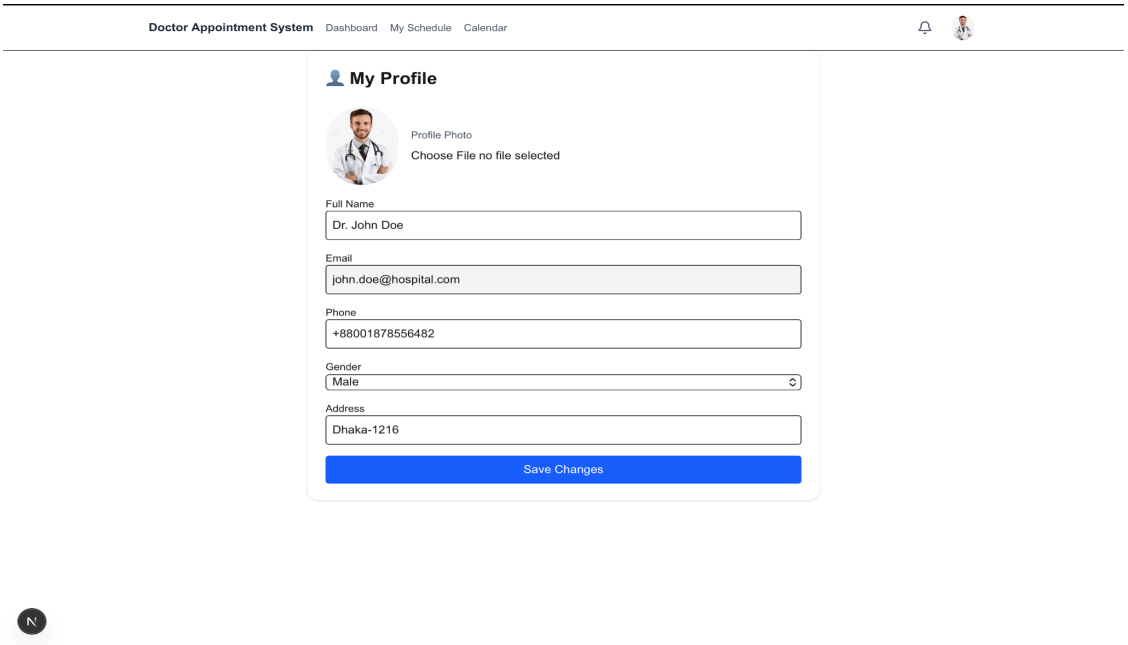


Figure 5.14 : Profile Manage – Doctor

5.4.15 Dashboard - Patient

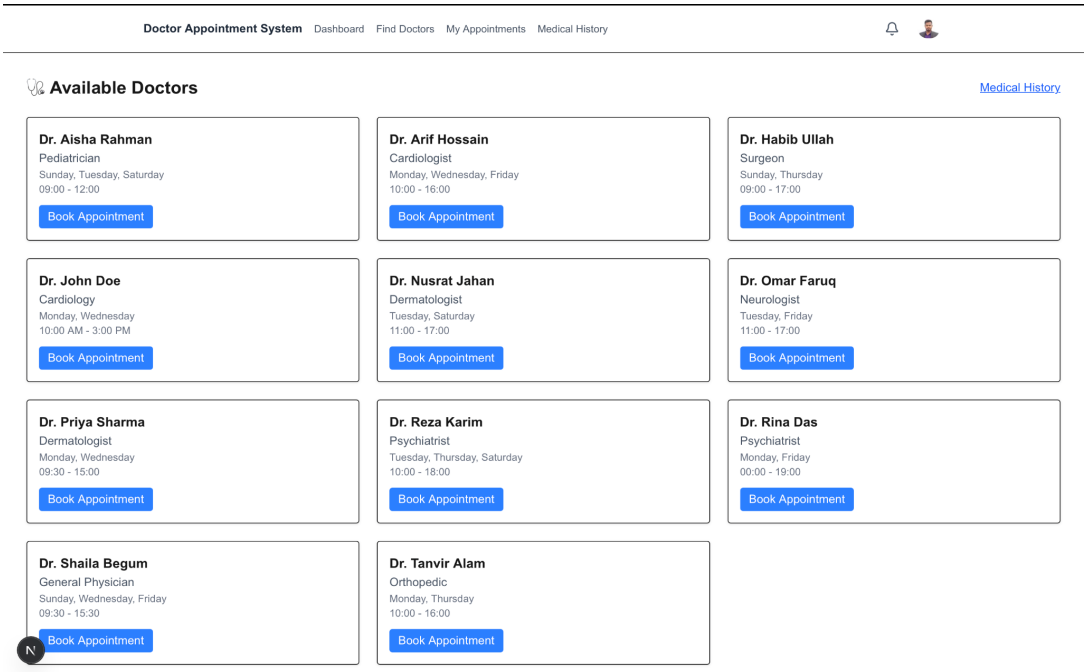


Figure 5.15 : Dashboard – Patient

5.4.16 Dashboard - Patient

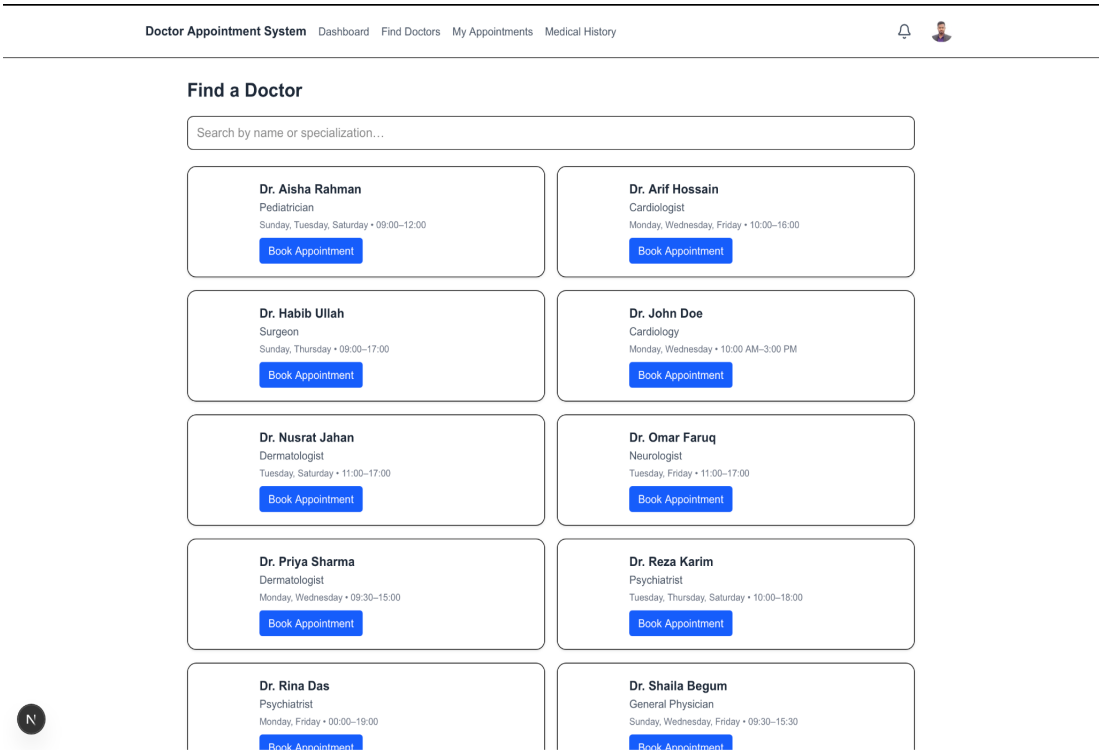


Figure 5.16 : Find a Doctor - Patient

5.4.17 Dashboard - PatientMy Appointments - Patient

Figure 5.17 : My Appointments – Patient

5.4.18 My Medical History- Patient

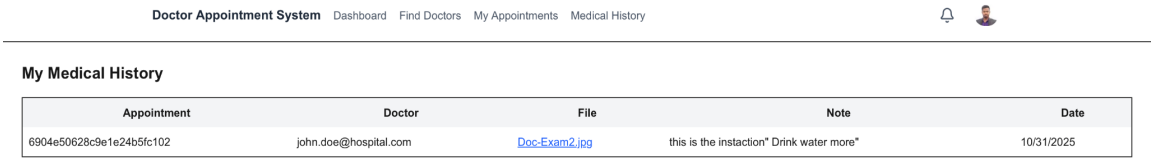


Figure 5.18 : My Medical History- Patient

5.4.19 Profile - Patient

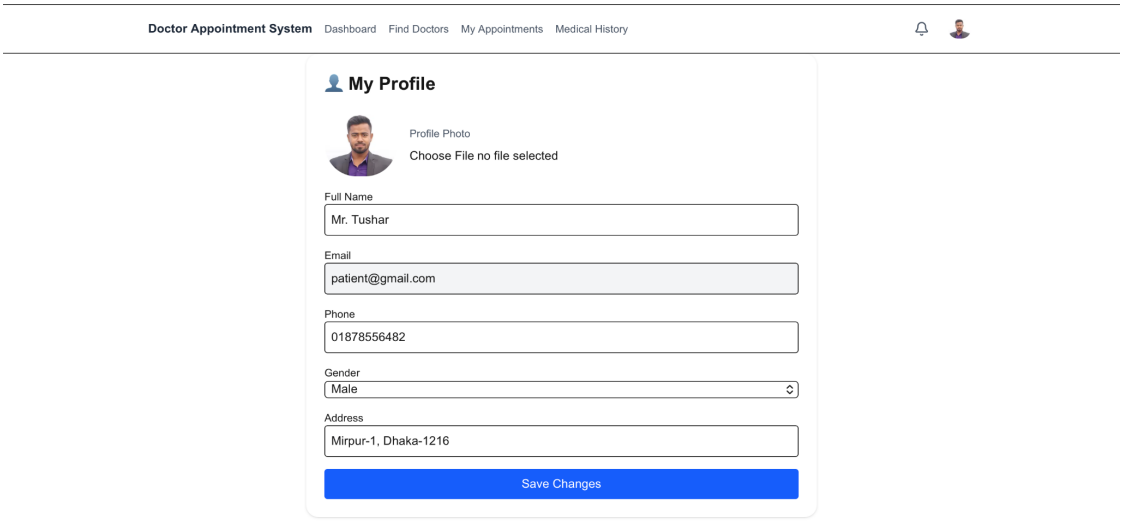


Figure 5.19 : Profile – Patient

5.4.20 Notifications - All User

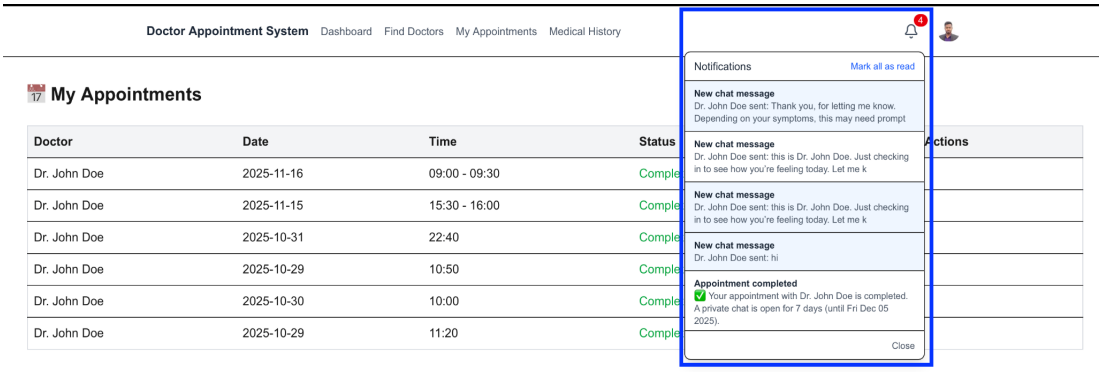


Figure 5.20 : Notifications - All User

Chapter 6 : Project Summary

6.1 Introduction

The Smart Doctor Appointment System was developed to modernize the traditional healthcare appointment workflow by providing an accessible, efficient, and fully online platform for patients, doctors, and administrators. The system enables users to manage appointments, schedules, communication, and medical records through a secure and intuitive web interface. Built with modern web technologies such as Next.js, MongoDB Atlas, NextAuth, and Pusher, the system demonstrates the application of structured software engineering practices, including requirement analysis, design modeling, implementation, and testing.

This chapter summarizes the overall outcomes of the project, its limitations, scope, future enhancement opportunities, and concluding observations.

6.2 Project Limitations

Although the system performs effectively within its designed scope, several limitations were identified during development:

- The system currently supports web-based access only; no native Android or iOS application is included.
- Real-time communication is limited to text-based chat only, without support for audio or video consultation.
- The medical record upload feature accepts only basic file types (PDF, images), with no in-built virus scanning or advanced verification.
- Integration with external hospital management systems, e-prescription modules, or national health databases is not implemented.
- Notification channels (SMS/Email/WhatsApp) depend on external API services, which may require paid plans for production use.
- The UI/UX is designed for functional efficiency but does not include advanced accessibility features such as screen-reader optimization.

These limitations present opportunities for improvement in future versions.

6.3 Project Scopes

The primary scope of this system focuses on improving appointment workflow and communication within a healthcare environment. Key features included within the project scope:

- Secure user authentication for Patients, Doctors, and Admins.
- Viewing doctor profiles, availability, and booking appointments.
- Doctor schedule management with real-time slot updates.
- Patient–Doctor real-time chat functionality.
- Uploading and viewing medical records.
- System-wide notification management.
- Administrative tools for managing users, appointments, and schedules.
- Cloud-based deployment and database integration.

The system is intended for clinics, private chambers, or small healthcare organizations seeking a reliable digital appointment solution.

6.4 Future Works

Several improvement can significantly increase the system's quantity:

- Mobile App Development: Android and iOS applications for improved accessibility.
- Video Consultation Module: Enabling online telemedicine visits via WebRTC or integrated API services.
- AI-Powered Appointment Suggestions: Intelligent recommendations for doctors based on symptoms or patient history.
- Advanced Medical Record System: Including diagnosis templates, prescription generation, and doctor notes.
- Integration with Government Health Systems: Synchronizing patient data with national health databases (e.g., vaccination or ID verification).

- Automated Chatbot Support: Assisting patients with booking, FAQs, and symptom-based routing.
- Multi-language Support: Allowing users to choose regional languages.
- Enhanced Analytics Dashboard: Providing insights for doctors and admins (appointment trends, patient reports, workload metrics).

6.5 Conclusion

The Smart Doctor Appointment System successfully demonstrates how modern web technologies can be applied to overcome the inefficiencies of manual healthcare scheduling. Through secure authentication, streamlined appointment booking, real-time communication, and digital medical record management, the system enhances both patient convenience and doctor productivity. The project reflects well-defined software engineering principles, comprehensive testing, and structured implementation.

While there are limitations and opportunities for improvement, the system provides a strong foundation for future expansion into a full-scale digital healthcare platform capable of supporting advanced telemedicine and clinical workflow automation.

References

BrainBox IDE – Official Deployment. (2024). BrainBox IDE Original. https://brain-box-ide-original.vercel.app/

Firebase. (2024). Firebase Authentication documentation. https://firebase.google.com/docs/auth

GitHub. (2024). GitHub Docs: Version control and repository management. https://docs.github.com/

International Organization for Standardization. (2018). ISO/IEC/IEEE 29148:2018 — Systems and software engineering — Life cycle processes — Requirements engineering. https://www.iso.org/standard/72057.html

MDN Web Docs. (2024). Web APIs, JavaScript, HTML, and CSS documentation. https://developer.mozilla.org/

MongoDB. (2024). MongoDB Atlas documentation. https://www.mongodb.com/docs/

Next.js. (2024). Next.js documentation. https://nextjs.org/docs

Node.js. (2024). Node.js runtime documentation. https://nodejs.org/en/docs

OpenAI. (2024). Security best practices for web applications. https://openai.com/policies

OWASP Foundation. (2023). OWASP Top 10 — Web application security risks. https://owasp.org/www-project-top-ten/

Pusher. (2024). Pusher real-time messaging documentation. https://pusher.com/docs/

React. (2024). React developer documentation. https://react.dev/

Sommerville, I. (2016). Software engineering (10th ed.). Pearson.

Tailwind CSS. (2024). Tailwind CSS documentation. https://tailwindcss.com/docs

Ahamed, S. (2024). BrainBox IDE – Original Version [Source code]. GitHub. https://github.com/shamim36/BrainBox-IDE-Original

Ullah, M. (2024). SmartPOS Restaurant Management System – Backend [Source code]. GitHub. https://github.com/monir-ullah/SmartPOS-Restaurant-Management-System-Backend

Ullah, M. (2024). SmartPOS Restaurant Management System – Frontend [Source code]. GitHub. https://github.com/monir-ullah/SmartPOS-Restaurant-Management-System-Fontend

Short Biography

Name: S.M. Jubail Islam

Department: Software Engineering

University: Daffodil International University

Program: Bachelor of Science in Software Engineering (BSc in SWE)

Batch: 38th

Student ID: 221-35-1048

Biography:

S.M. Jubail Islam is an undergraduate student at Daffodil International University pursuing a degree in Software Engineering. With a strong interest in software development and web technologies, he has focused on developing innovative solutions that integrate technology into real-world applications.

The *Smart Doctor Appointment System* project reflects his enthusiasm for improving healthcare management through automation and digital innovation. S.M. Jubail Islam is passionate about full-stack development, database management, and creating scalable software solutions that enhance accessibility and efficiency across sectors.

Plagiarism Report

221-35-1048

ORIGINALITY REPORT

21%	13%	3%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	dspace.daffodilvarsity.edu.bd:8080 Internet Source	9%
2	Submitted to Higher Education Commission Pakistan Student Paper	1%
3	Submitted to Daffodil International University Student Paper	1%
4	Submitted to University of Wales Institute, Cardiff Student Paper	1%
5	Yaşar, Eminullah. "A Run-Time Environment for Execution of State Machines", Dokuz Eylul Universitesi (Turkey), 2024 Publication	1%
6	library.binus.ac.id Internet Source	<1%
7	www.coursehero.com Internet Source	<1%
8	Submitted to DU Faculty Members Student Paper	<1%
9	Submitted to IPMC Circle (NCC) Student Paper	<1%
10	link.springer.com Internet Source	<1%

11	Submitted to Universiti Tun Hussein Onn Malaysia Student Paper	<1 %
12	uwe-repository.worktribe.com Internet Source	<1 %
13	Submitted to American Public University System Student Paper	<1 %
14	Submitted to UC, Irvine Student Paper	<1 %
15	Submitted to University of Carthage Student Paper	<1 %
16	Submitted to Universiti Teknologi Malaysia Student Paper	<1 %
17	Submitted to University Tun Hussein Onn Malaysia Student Paper	<1 %
18	Submitted to Concordia University Student Paper	<1 %
19	Gunes, Funda. "Variable Selection via Confidence Regions", Proquest, 20111108 Publication	<1 %
20	Submitted to University of Essex Student Paper	<1 %
21	Submitted to University of London External System Student Paper	<1 %
22	Submitted to International College of Management Student Paper	<1 %
23	ijrpr.com	

	Internet Source	<1 %
24	Submitted to King's Own Institute Student Paper	<1 %
25	Submitted to Laureate Higher Education Group Student Paper	<1 %
26	Submitted to Heriot-Watt University Student Paper	<1 %
27	Submitted to University of Strathclyde Student Paper	<1 %
28	Submitted to Kingston University Student Paper	<1 %
29	Submitted to University of Greenwich Student Paper	<1 %
30	a100.gov.bc.ca Internet Source	<1 %
31	Submitted to Kaunas University of Technology Student Paper	<1 %
32	pure.tue.nl Internet Source	<1 %
33	Submitted to Kuwait University Student Paper	<1 %
34	Submitted to Victorian Institute of Technology Student Paper	<1 %
35	www.nj.gov Internet Source	<1 %
36	Submitted to University of Teesside Student Paper	<1 %

37	Submitted to Emirates Aviation College, Aerospace & Academic Studies Student Paper	<1 %
38	Submitted to University of Nicosia Student Paper	<1 %
39	ir.jkuat.ac.ke Internet Source	<1 %
40	medium.com Internet Source	<1 %
41	www.javatpoint.com Internet Source	<1 %
42	www.marketresearchstore.com Internet Source	<1 %
43	Submitted to National College of Ireland Student Paper	<1 %
44	pdfcoffee.com Internet Source	<1 %
45	repository.upi.edu Internet Source	<1 %
46	www.educba.com Internet Source	<1 %
47	en.wikipedia.org Internet Source	<1 %
48	Phenikaa University Publication	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

Library Clearance

Account Clearance