Patient

|

1, Login -> Patient Dashboard , Book Appointment

Patient Dashboard -> List of past Appointments and Medical History

Book Appointment -> Appointment (List of Doctors)-> Select a doctor (Post doctor id , patient id ,timeslot) -> (GET Availability Phase)->Confirm Booking(Enum status)

3, Availability->Post doctor id-> get availability

Doctor

|

1, Login -> Doctor Dashboard->Manage Availability, Consultation Records

Manage Availability -> Add availability in table

Consultation Record -> List of Appointment -> Get Patient Details -> add medical prescription

Entities

• User

o User ID

o Name

o Role (Doctor/Patient)

o Email

o Phone

Appointment

o AppointmentID

o PatientID

o DoctorID

o TimeSlot

o Status (Booked, Cancelled, Completed)

Consultation

o ConsultationID

o AppointmentID

o Notes

o Prescription

Availability

o DoctorID

o Date

o TimeSlots

Tables and Relationships

• User

o Primary Key: UserID

• Appointment

o Primary Key: AppointmentID

o Foreign Keys: PatientID, DoctorID

• Consultation

o Primary Key: ConsultationID

o Foreign Key: AppointmentID

• Availability

o Primary Key: DoctorID

User Interface Design

7.1 Wireframes

• Patient Dashboard: View appointments and medical history.

• Doctor Dashboard: Manage availability and consultation records.

• Appointment Page: Search and book appointments.

Links:

Patient

List of Appointments Past -> (Patient Id , Appointment Id)

List of Medical History -> (PatientId -> Appointment Id -> Consultation Id)

Doctor

List of Appointment ->(Doctor Id , Appointment Id)

Manage Availability -> (Doctor Id , Availabiltity ID)

Project: Healthcare Appointment Management System

1. Introduction

The purpose of this document is to provide a detailed Low-Level Design (LLD) for the Healthcare

Appointment Management System. This system facilitates efficient appointment scheduling, doctor patient management, and consultation records. It employs a REST API-based backend architecture and

uses Angular or React for the frontend.

This design is compatible with both Java (Spring Boot) and .NET (ASP.NET Core) frameworks.

2. Module Overview

The project consists of the following modules:

2.1 User Management

Handles registration, authentication, and profile management of doctors and patients.

2.2 Appointment Scheduling

Allows patients to book, modify, or cancel appointments.

2.3 Consultation Records

Manages patient consultation notes, prescriptions, and medical history.

2.4 Doctor Availability

Enables doctors to manage their schedules and availability.

2.5 Notifications and Alerts

Notifies patients and doctors about appointment reminders and changes.

3. Architecture Overview

3.1 Architectural Style

• Frontend: Angular or React

• Backend: REST API-based architecture

• Database: Relational Database (MySQL/PostgreSQL/SQL Server)

3.2 Component Interaction

• The frontend interacts with the REST API for all operations.

• The backend handles business logic and communicates with the database.

• Notifications are sent via email, SMS, or the frontend.

4. Module-Wise Design

4.1 User Management Module

4.1.1 Features

• Register as a doctor or patient.

• Login and manage profiles.

4.1.2 Data Flow

1. Users interact with the frontend to register/login.

2. Frontend sends user data to the REST API.

3. Backend authenticates users and interacts with the database.

4. Responses are sent back to the frontend for display.

4.1.3 Entities

• User

o UserID

o Name

o Role (Doctor/Patient)

o Email

o Phone

4.2 Appointment Scheduling Module

4.2.1 Features

• Book, update, or cancel appointments.

• View doctor availability and select time slots.

4.2.2 Data Flow

1. Patients initiate appointment requests via the frontend.

2. Frontend communicates with the backend API.

3. Backend validates requests and updates the database.

4. Confirmation is sent back to the patient.

4.2.3 Entities

• Appointment

o AppointmentID

o PatientID

o DoctorID

o TimeSlot

o Status (Booked, Cancelled, Completed)

4.3 Consultation Records Module

4.3.1 Features

• Store consultation notes and prescriptions.

• View medical history.

4.3.2 Data Flow

1. Doctors record consultation details via the frontend.

2. Backend saves the records to the database.

3. Patients can access their medical history.

4.3.3 Entities

• Consultation

o ConsultationID

o AppointmentID

o Notes

o Prescription

4.4 Doctor Availability Module

4.4.1 Features

• Manage availability and time slots.

• Block off unavailable days.

4.4.2 Data Flow

1. Doctors update their availability via the frontend.

2. Backend updates the database.

3. Availability is displayed on the appointment scheduling module.

4.4.3 Entities

• Availability

o DoctorID

o Date

o TimeSlots

4.5 Notifications and Alerts Module

4.5.1 Features

• Send reminders for upcoming appointments.

• Notify patients about cancellations or reschedules.

4.5.2 Data Flow

1. Backend generates notifications based on triggers (e.g., upcoming appointment).

2. Notifications are sent via email/SMS or displayed in the frontend.

5. Deployment Strategy

5.1 Local Deployment

• Frontend: Local Angular/React servers.

• Backend: REST API deployed using Spring Boot/ASP.NET Core.

• Database: Local database instance for development.

6. Database Design

6.1 Tables and Relationships

• User

o Primary Key: UserID

• Appointment

o Primary Key: AppointmentID

o Foreign Keys: PatientID, DoctorID

• Consultation

o Primary Key: ConsultationID

o Foreign Key: AppointmentID

• Availability

o Primary Key: DoctorID

7. User Interface Design

7.1 Wireframes

• Patient Dashboard: View appointments and medical history.

• Doctor Dashboard: Manage availability and consultation records.

• Appointment Page: Search and book appointments.

8. Non-Functional Requirements

8.1 Performance

• System must support up to 50 concurrent users in the development phase.

8.2 Scalability

• Easily scalable for production environments.

8.3 Security

• Ensure secure login and data encryption.

8.4 Usability

• The user interface must be responsive and accessible.

9. Assumptions and Constraints

9.1 Assumptions

• The system will operate in a local environment during development.

9.2 Constraints

• No cloud integration is planned for the initial phase.