Medic System Documentation

Table of Contents

1. Introduction
2. Technologies Used
3. Project Structure
4. Backend and Frontend Setup
   - ❖ Development Environment
   - ❖ Dependencies
   - ❖ Configuration
   - ❖ Model Class
   - ❖ Repository Layer
   - ❖ Service Layer
   - ❖ Controller Layer
   - ❖ DTO Layer
   - ❖ Exception Layer
   - ❖ Config Layer
   - ❖ HTML Pages for Appointment, Medication, Doctor and Patient with Tailwind CSS
5. API Endpoints
6. Table Schema
7. Running the Application
8. Conclusion

Introduction

In the rapidly evolving landscape of healthcare, the need for efficient management systems has never been more critical. The Medic System Application is designed to address the complexities of patient care, medication management, and appointment scheduling within healthcare facilities. This application serves as a comprehensive solution that streamlines these processes, ultimately enhancing the quality of care provided to patients.

The primary objective of the Medic System Application is to facilitate healthcare providers in managing patient information effectively. In a typical healthcare setting, practitioners often face challenges related to the organization and accessibility of patient data. Traditional methods of record-keeping can lead to inefficiencies, errors, and delays in patient care. The Medic System Application aims to mitigate these issues by providing a centralized platform where healthcare professionals can easily access, update, and manage patient records.

One of the core features of the application is its patient management system. This functionality allows users to create, read, update, and delete patient records seamlessly. Healthcare providers can input essential information such as patient demographics, medical history, and contact details. This ensures that all relevant data is readily available, enabling practitioners to make informed decisions regarding patient care. Additionally, the application supports the tracking of medications prescribed to patients, including dosages and administration schedules. This feature is crucial for enhancing patient safety and adherence to treatment plans, as it minimizes the risk of medication errors.

Appointment scheduling is another vital aspect of the Medic System Application. The application provides tools for healthcare providers to manage patient appointments efficiently. Users can schedule, reschedule, and cancel appointments with ease, reducing wait times and improving the overall patient experience. By streamlining the appointment process, healthcare facilities can optimize their operations and ensure that patients receive timely care.

The Medic System Application is built using modern technologies that enhance its functionality and user experience. The backend is developed using Spring JPA, which provides a robust framework for data persistence and management. This allows for efficient interaction with the PostgreSQL database, ensuring that patient data is stored securely and can be retrieved quickly. The frontend is designed using Thyme leaf, a powerful templating engine that enables dynamic rendering of web pages. Coupled with Tailwind CSS, the application offers a responsive and visually appealing user interface that enhances usability across various devices.

Security is a paramount concern in healthcare applications, and the Medic System Application incorporates best practices to protect sensitive patient information. User authentication and authorization mechanisms are implemented to ensure that only authorized personnel can access and modify patient records. This safeguards against unauthorized access and helps maintain patient confidentiality.

In summary, the Medic System Application is a vital tool for healthcare providers seeking to improve their operational efficiency and enhance patient care. By offering a comprehensive solution for patient management, medication tracking, and appointment scheduling, the application addresses the challenges faced by healthcare professionals in today's fast-paced environment. With its user-friendly interface and robust backend functionality, the Medic System Application is poised to make a significant impact on the quality of healthcare delivery. As the healthcare landscape continues to evolve, applications like this will play a crucial role in ensuring that providers can deliver the best possible care to their patients.

Technologies Used

Backend: Java, Spring Boot, Spring Data JPA, Maven

Frontend: Thymeleaf, HTML, Tailwind CSS

Testing: Assertj Core, Mockito Core

Database: PostgreSQL

API: Postman Client

Development Tools:

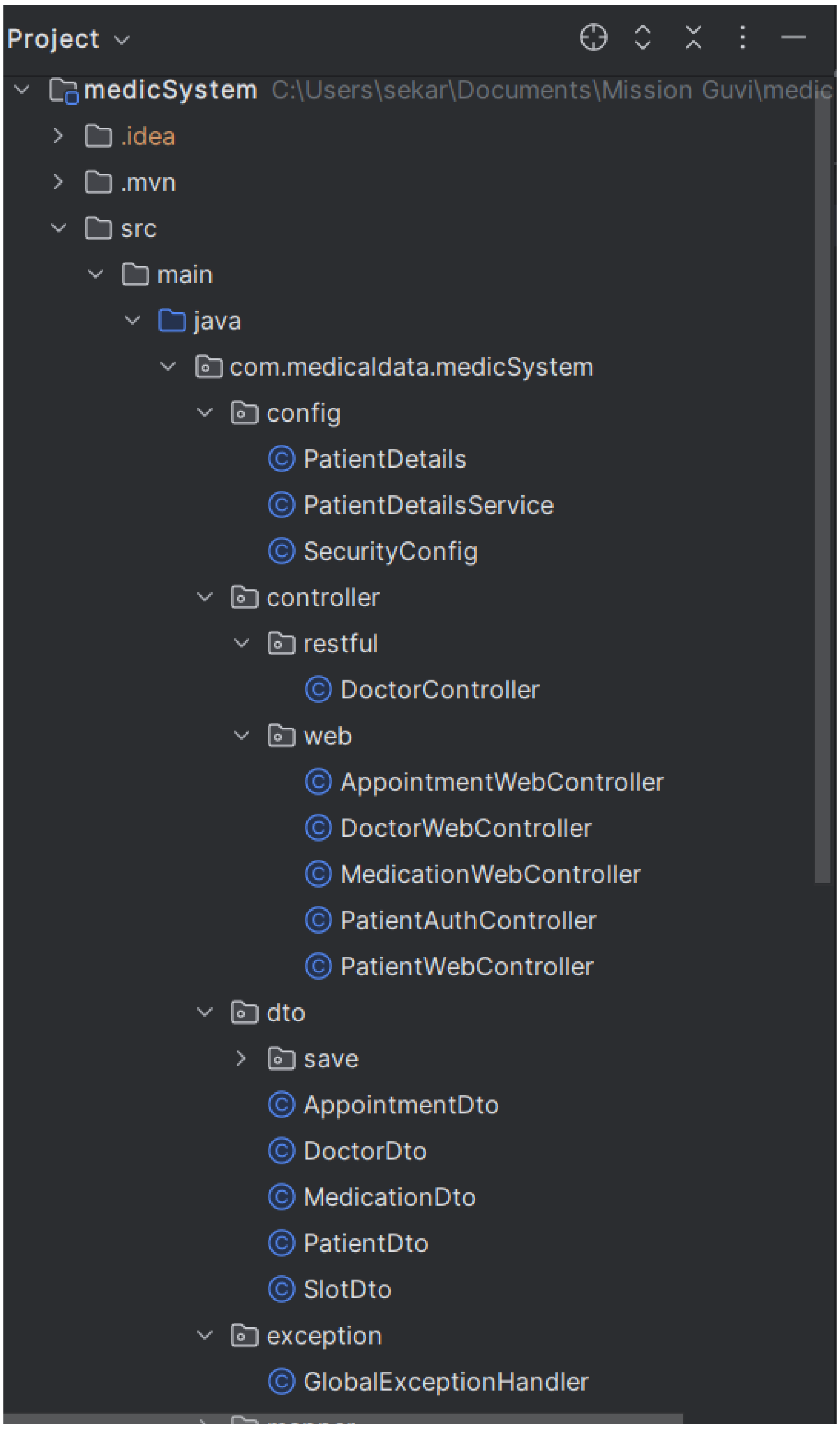
- ❖ IntelliJ IDEA Community Edition 2024.3.5
  Note: Don’t try the latest version (2025.1) and above. Because thymeleaf is not supportable for updated version. Kindly recommended to use 2024.3.5 and below version.

Project Structure

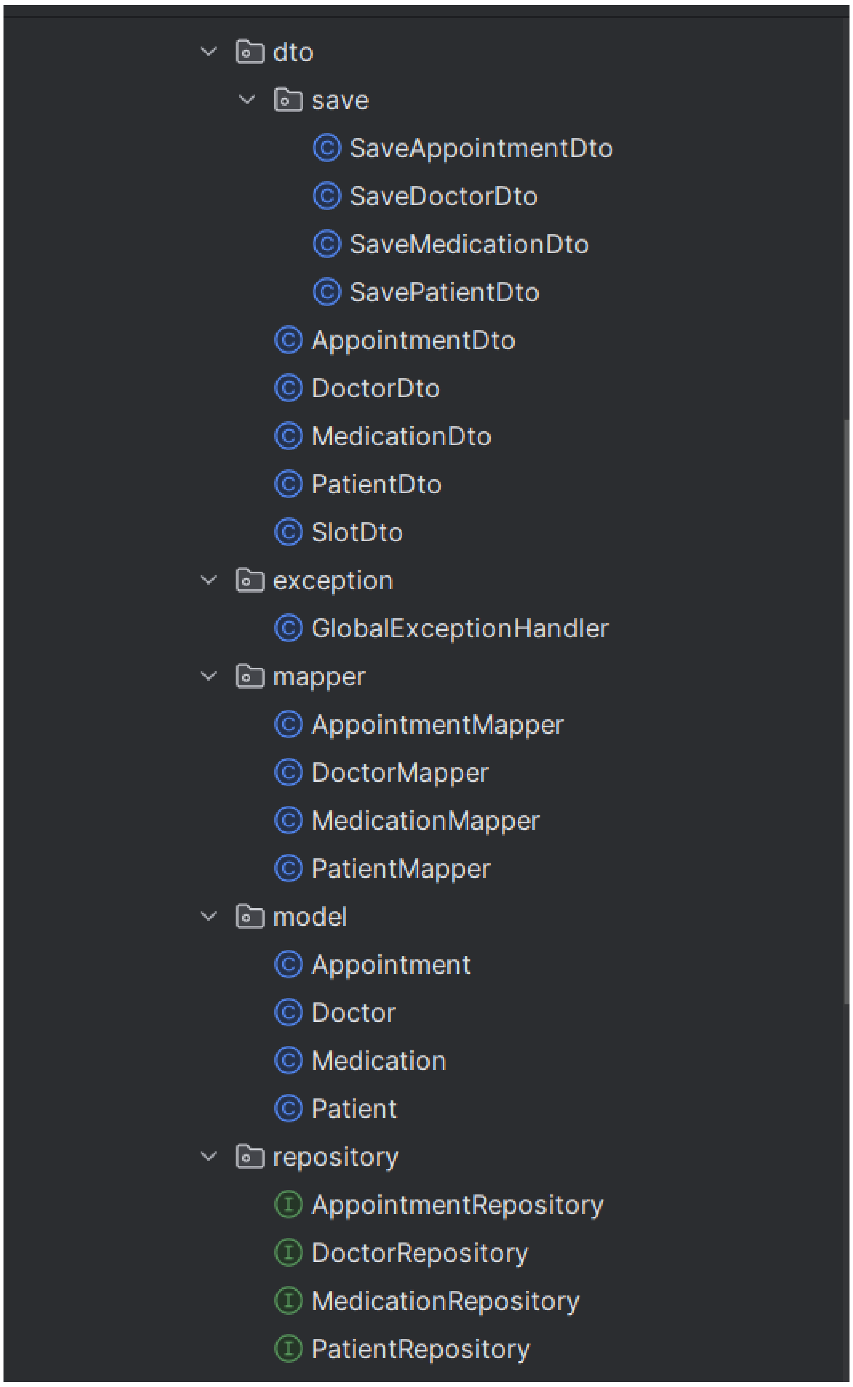
Spring Boot Structure with Dependencies

- Spring Boot Starter Web
- Spring Data JPA
- PostgreSQL driver
- Lombok
- Spring Validation
- Spring Security

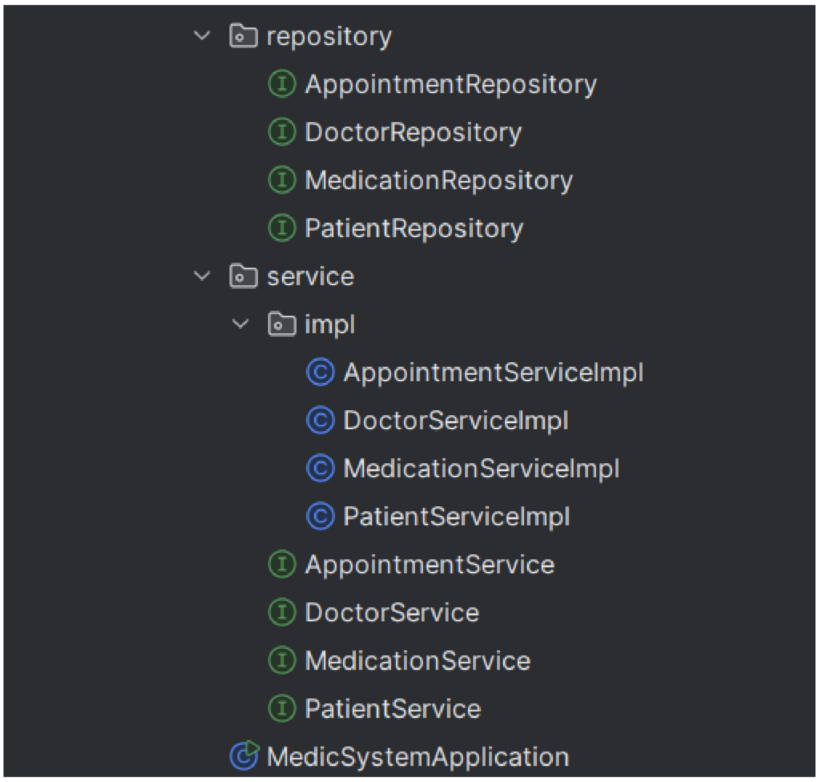
Backend Setup



```
Project
medicSystem  C:\Users\sekar\Documents\Mission Guvi\medic
  .idea
  .mvn
  src
    main
      java
        com.medicaldata.medicSystem
          config
            PatientDetails
            PatientDetailsService
            SecurityConfig
          controller
            restful
              DoctorController
            web
              AppointmentWebController
              DoctorWebController
              MedicationWebController
              PatientAuthController
              PatientWebController
          dto
            save
            AppointmentDto
            DoctorDto
            MedicationDto
            PatientDto
            SlotDto
          exception
            GlobalExceptionHandler
```

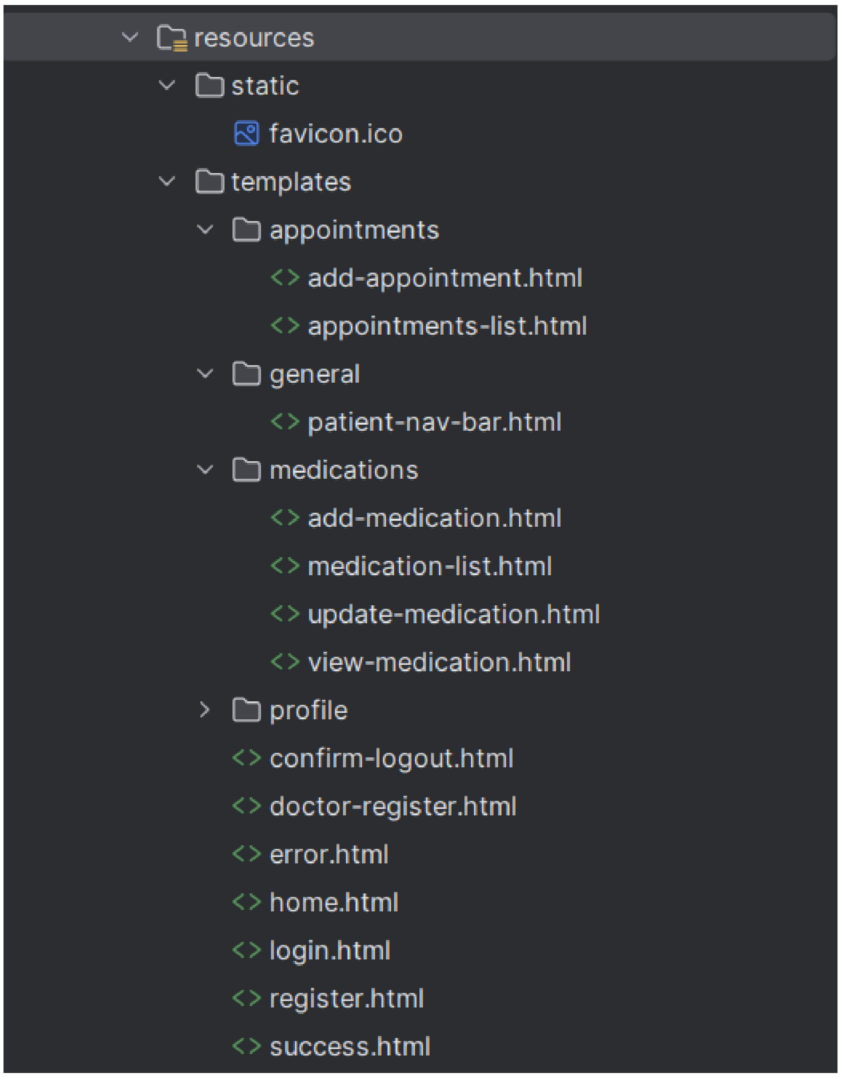


```
dto
  save
    SaveAppointmentDto
    SaveDoctorDto
    SaveMedicationDto
    SavePatientDto
  AppointmentDto
  DoctorDto
  MedicationDto
  PatientDto
  SlotDto
exception
  GlobalExceptionHandler
mapper
  AppointmentMapper
  DoctorMapper
  MedicationMapper
  PatientMapper
model
  Appointment
  Doctor
  Medication
  Patient
repository
  AppointmentRepository
  DoctorRepository
  MedicationRepository
  PatientRepository
```



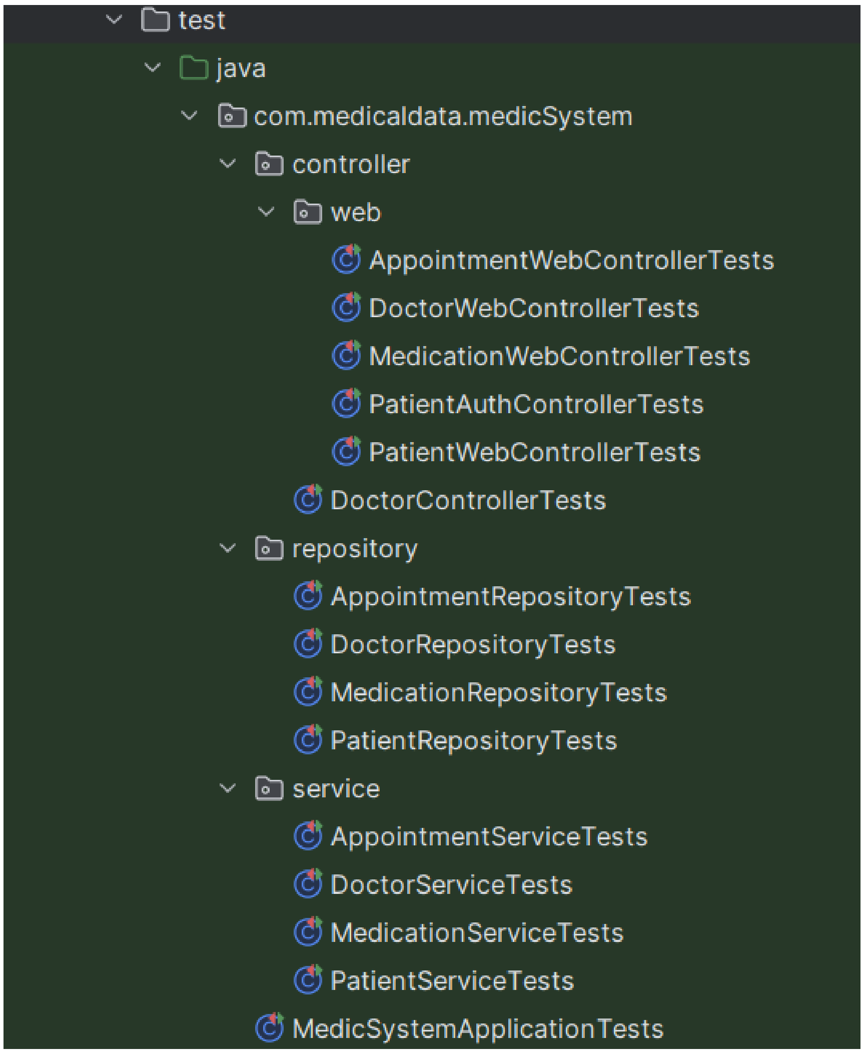
```
repository
    AppointmentRepository
    DoctorRepository
    MedicationRepository
    PatientRepository
service
    impl
        AppointmentServiceImpl
        DoctorServiceImpl
        MedicationServiceImpl
        PatientServiceImpl
    AppointmentService
    DoctorService
    MedicationService
    PatientService
MedicSystemApplication
```

Frontend Setup



```
resources
  static
    favicon.ico
  templates
    appointments
      add-appointment.html
      appointments-list.html
    general
      patient-nav-bar.html
    medications
      add-medication.html
      medication-list.html
      update-medication.html
      view-medication.html
    profile
    confirm-logout.html
    doctor-register.html
    error.html
    home.html
    login.html
    register.html
    success.html
```

Testing Setup



```
test
  java
    com.medicaldata.medicSystem
      controller
        web
          AppointmentWebControllerTests
          DoctorWebControllerTests
          MedicationWebControllerTests
          PatientAuthControllerTests
          PatientWebControllerTests
        DoctorControllerTests
      repository
        AppointmentRepositoryTests
        DoctorRepositoryTests
        MedicationRepositoryTests
        PatientRepositoryTests
      service
        AppointmentServiceTests
        DoctorServiceTests
        MedicationServiceTests
        PatientServiceTests
      MedicSystemApplicationTests
```

Core Components and Packages

Backend Contents

This section details the purpose of the key packages within src/main/java/com/medicaldata/medicSystem/.

1. config

* Purpose: Declares a class that provides **bean definitions** using @Bean methods, used to **centralize configuration** of dependencies and services and enables **Java-based configuration** for the Spring container.

* Key File: SecurityConfig.java

   * This class typically used to **secure the application** using Spring Security. It helps manage access to sensitive data like patient records, prescriptions, and appointments.

2.controller

* Purpose: This package contains classes responsible for handling incoming HTTP requests from clients. Controllers act as the entry point to the application's business logic.

* There are two types of controllers included in this file. One is RESTful and the other one is Web Controller.

Only one content is added in RESTful Controller (src/main/java/com/medicaldata/medicSystem/controller/restful)

* Key File: DoctorController.java

   * This class likely defines REST endpoints for operations related to adding Doctor's detail like Name, Gender, Mobile, Email, Specialty, etc..,

There are 5 contents in Web Controller

(src/main/java/com/medicaldata/medicSystem/controller/web)

*Key Files :

1. AppointmentWebController.java

* a **Spring MVC controller** that handles web requests related to appointments, such as creating, updating, viewing, or deleting appointments.

2. DoctorWebController.java

*a **controller** class that handles web requests related to "doctor" resources, such as listing doctors, adding a new doctor, updating doctor information, or deleting a doctor.

3. MedicationWebController.java

*It is part of a **Spring Boot** application that handles web requests related to **medication** data — such as retrieving, adding, updating, or deleting medication records. It is a **Spring MVC controller** that maps HTTP requests to handler methods.

4.PatientAuthController.java

*This controller handles authentication and possibly registration or login functionalities specific to "Patient" users.

5.PatientWebController.java

*This controller handles HTTP requests related to patients—such as viewing patient information, creating new patients, updating existing records, and deleting patients.

3. dto

* Purpose: This package holds Data Transfer Objects (DTOs).

Encapsulation: Hide model/internal details and expose only required fields.

Security: Prevent sending sensitive or unwanted fields (e.g., passwords, internal IDs).

Efficiency: Avoid sending large or nested entities when only a subset is needed.

Validation: Enable client-side or request-based validation on input data.

Mapping Simplification: Customize how domain models are presented to clients or APIs.

* Key Files:

1.AppointmentDto.java

* Represents appointment data sent to or received from the client, without exposing internal model details like doctor passwords, internal IDs, or database-specific configurations.

2.DoctorDto.java

*Used to **transfer doctor-related data** between layers like controller, service, and repository - especially in APIs or UI-bound operations — without exposing the full Doctor model.

3.MedicationDto.java

*Represents medication data in a clean and controlled way—typically used in **healthcare or medical system applications** to pass medication data between layers (controller ↔ service ↔ persistence) or to/from clients in an API.

4.PatientDto.java

*Used to **carry patient-related data** between layers in a medical system application (e.g., controller ⇄ service ⇄ repository), while avoiding direct exposure of the internal Patient model.

5.SlotDto.java

*Commonly used to **transfer appointment slot data** between layers (like controller, service, and frontend). It helps decouple the internal Slot model from external input/output (e.g., via REST APIs or UI forms).

In dto package, save package also included. It consists of four contents.

(src/main/java/com/medicaldata/medicSystem/dto/save)

* Key Files:

1.SaveAppointmentDto.java

* Used to **transfer appointment creation data** (like booking details) from the client/UI to the backend — especially when a patient wants to book an appointment.

2.SaveDoctorDto.java

* Used specifically when **creating or updating doctor information**—typically via a form or REST API request. It separates the **input data model** from the internal Doctor model, ensuring **data validation, encapsulation, and security**.

3.SaveMedicationDto.java

* Used to **capture and validate medication input data** when saving or updating medication records — especially via REST APIs or form submissions.

4.SavePatientDto.java

* Used to **collect and transfer patient data** when creating or updating a patient record. It's a form-specific DTO—optimized for input (especially from UI forms or API requests)—that **decouples internal model logic** from external data handling.

4. exception

* Purpose: This package is for custom exception classes. Defining custom exceptions helps in handling specific error scenarios in a structured way, improving the clarity and maintainability of error handling.

* Key File: GlobalExceptionHandler.java

* It is used in a **Java Spring Boot application** to **centrally handle exceptions** across the entire application, instead of writing repetitive try-catch logic in each controller or service. It improves code **cleanliness, maintainability, and error handling consistency**—especially useful in applications like a **medical system**, where invalid patient data, booking conflicts, or missing resources are common.

5. mapper

* Purpose: This package contains classes responsible for mapping data between different object types, specifically between DTOs and Model objects. Using mappers helps in separating the mapping logic from the core business logic. Mapper is often used for error handling in java.

* Key Files:

1.AppointmentMapper.java

* This mapper decouples internal models (entities) from external views (DTOs), simplifies repetitive mapping logic, keeps controller and service layers clean and centralizes conversion rules (e.g., date formatting, nested fields)

2.DoctorMapper.java

* This mapper helps you to keep your application **modular, clean, and decoupled**, especially when exposing only specific fields about doctors to the client (e.g., name, specialization, not sensitive/internal info like salary or credentials).

3.MedicationMapper.java

* This mapper helps **isolate data transformation logic**, keep code **clean**, and prevent exposing sensitive/internal fields.

4. PatientMapper.java

* This mapper encapsulates conversion logic between entity and DTO, keeps controller/service layers clean, prevents exposing sensitive fields in APIs and enables customized output/input formatting (e.g., full name, age calculation).

6. entity

* Purpose: This package contains JPA Model classes. These classes map to database tables and represent the data model of the application.

* Key Files:

1. Appointment.java

* This class represents a patient's appointment with a doctor. It is typically annotated with JPA/Hibernate annotations and mapped to a database table like appointments.

2. Doctor.java

* This class models a **doctor or healthcare provider** in the system. It's typically a JPA entity that represents the doctors table in the database.

3. Medication.java

*This class models a drug or prescribed medicine. It plays a central role in managing prescriptions, treatment plans, and patient medication history.

4.Patient.java

*This class models a **patient** — someone receiving medical care or services. It typically maps to a patients table in the database and is used throughout the application for managing patient data.

7. repository

Purpose: This package represents the Data Access Layer. It contains interfaces that extend Spring Data JPA repositories, providing methods for performing CRUD (Create, Read, Update, Delete) operations on the medicSystem entity without writing boilerplate code.

* Key Files:

1. AppointmentRepository.java

* A Spring Data JPA repository interface used to manage appointment data in the database.

2. DoctorRepository.java

* A Spring Data JPA interface used to manage Doctor entities in a medical system. It provides an abstraction layer for performing CRUD operations and custom queries on the doctors table.

3. MedicationRepository.java

* A Spring Data JPA interface that provides database access for the Medication entity in a medical system. It allows you to retrieve, store, and manage medications that are part of patient prescriptions.

4. PatientRepository.java

* An interface in a Java Spring Boot application that is responsible for **accessing and managing patient data in the database**. It typically extends JpaRepository, which gives it all basic CRUD operations for free.

8. service

Purpose: This package contains the business logic of the application. The service layer orchestrates operations by interacting with repositories and potentially other services.
It defines the core functionalities related to medical operations, such as patient registration, appointment booking, doctor management, and prescription handling.

* Key Files:

1.AppointmentService.java

*An interface acts as a **contract** for appointment operations, used to define **method signatures** that AppointmentServiceImpl will implement and allows for easier testing and flexibility (e.g., using mocks).

2.AppointmentServiceImpl.java

*An implementation class that implements the logic declared in AppointmentService, interacts with repositories, converts between **entities and DTOs** using mappers and applies **business rules**, such as checking time slot availability.

3.DoctorService.java

*An interface that defines what services are offered, promotes **loose coupling** and **testability** and used for dependency injection (@Autowired).

4.DoctorServiceImpl.java

*An implementation class that implements logic: fetching, saving, mapping doctors.

5.MedicationService.java

* An **interface** that defines business operations related to **medications** and promotes **clean architecture** and easy testing by abstracting implementation.

6.MedicationServiceImpl.java

* **An implementation** of MedicationService interface containing the actual logic.

7.PatientService.java

* An interface that defines the **contract** for patient-related business operations and promotes **loose coupling** and **testability**

8.PatientServiceImpl.java

*An implementation class that Implements the methods defined in PatientService.java using Custom logic and validation.

MedicSystemApplication.java

* Purpose: This is the main class that bootstraps and launches the Spring Boot application. It contains the main method.

Frontend Contents

This section details the purpose of the key packages within src/main/resources/.

Purpose: This directory is used for storing configuration files (e.g., application.properties or application.yml), static resources, and templates.

A. Static
   In this directory, I've added a favicon.ico which consists of 16x16 ICO file with 32-bit color that related to the application.

B. Templates
   In this directory, there are 15 html files available for accessing the application with a presentable view, user friendly, easy access and attractive icons, animations and effects.

   Listing the html files that contains in src/main/resources/templates

   - confirm-logout.html -> Asking for confirming to logout
   - doctor-register.html -> Registration page for doctors
   - error.html -> Error page
   - home.html -> Homepage for Patients
   - login.html ->Login page for patients
   - register.html-> Registration page for patients
   - success.html ->Success page

   Listing the html files that contains in src/main/resources/templates/appointments

   - add-appointment.html -> Booking an appointment
   - appointments-list.html -> Showing list of appointments (upcoming and completed).

   Listing the html files that contains in src/main/resources/templates/general

   - patient-nav-bar.html -> Navigation bar for all pages

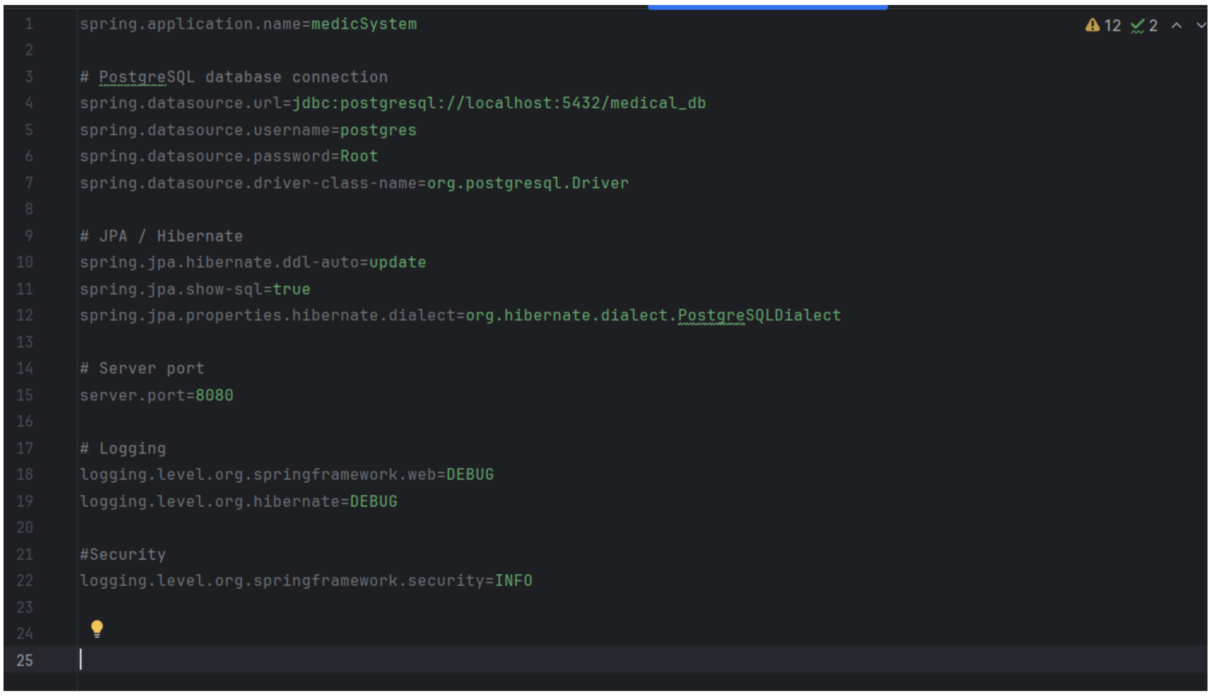
   Listing the html files that contains in src/main/resources/templates/medications

   - add-medication.html -> Adding medicine to the patient
   - medication-list.html -> List of Medicine
   - update-medication.html -> Modifying the medicine details
   - view-medication.html -> Viewing the medicine details

Listing the html files that contains in src/main/resources/templates/profile

- profile.html -> Displaying the patient profile

application.properties



```
spring.application.name=medicSystem

# PostgreSQL database connection
spring.datasource.url=jdbc:postgresql://localhost:5432/medical_db
spring.datasource.username=postgres
spring.datasource.password=Root
spring.datasource.driver-class-name=org.postgresql.Driver

# JPA / Hibernate
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

# Server port
server.port=8080

# Logging
logging.level.org.springframework.web=DEBUG
logging.level.org.hibernate=DEBUG

#Security
logging.level.org.springframework.security=INFO


```

10. Dependencies (pom.xml)

The pom.xml file (not explicitly shown but implied) located in the project root manages the project's dependencies (e.g., Spring Boot starters, database drivers, testing libraries) and build configuration using Maven.

Testing Contents

This section details the purpose of the key packages within src/test/java/com/medicaldata/medicSystem/.

1. Controller
   Purpose: To test the REST endpoints of the controller layer and verify that the system correctly handles incoming API requests and responses.

   *Key file in src/test/java/com/medicaldata/medicSystem/controller:
   DoctorControllerTests.java
   * An unit or integration test class that verifies the behavior of the DoctorController in the MedicSystem backend application. It ensures that the doctor-related REST endpoints function as expected.

   *Key files in src/test/java/com/medicaldata/medicSystem/controller/web:
   1.AppointmentWebControllerTests.java
   * A test class responsible for verifying the behavior of the AppointmentWebController in the MedicSystem web module (typically for admin or patient UI interactions).

   2.DoctorWebControllerTests.java
   * A test class specifically designed to verify the behavior of the DoctorWebController, which handles web (HTML-based) requests in the MedicSystem — typically used in traditional MVC (Thymeleaf/JSP) rather than REST APIs.

   3.MedicationWebControllerTests.java
   * A test class designed to verify the behavior of medication-related HTTP endpoints exposed by the MedicationWebController in a Java Spring Boot application.

   4.PatientAuthControllerTests.java
   * A test class designed to verify the behavior of the PatientAuthController, which typically handles patient authentication (like login, registration, and account access) in the MedicSystem.

5.PatientWebControllerTests.java
* An unit test class or integration test class that tests the functionality of the PatientWebController, which likely handles web-based patient interactions in the MedicSystem application (e.g., page views, form submissions, etc.).

2. repository
   Purpose: In tests, repositories are used to **prepare**, **validate**, and **clean up** real data in an **in-memory database** to ensure the application behaves correctly under different conditions.

   *Key files:
   1.AppointmentRepositoryTests.java
   * A test class in a Java-based MedicalSystem application used to verify that the AppointmentRepository works correctly. It ensures that appointment-related queries (e.g., by doctor, patient, or date) return the correct data from the database.

   2.DoctorRepositoryTests.java
   * A **test class** used to verify the functionality of the DoctorRepository in a **medical system application**.

   3.MedicationRepositoryTests.java
   * An unit or integration test class designed to test the behavior of the MedicationRepository.

   4.PatientRepositoryTests.java
   * A test class used to verify that the PatientRepository in a MedicalSystem application behaves as expected. It focuses on unit or integration testing of data access methods for the Patient entity.

3. Service
   Purpose: The service layer is tested to ensure that the application's business logic works correctly, independently of the database or web layer.

*Key files:
1.AppointmentServiceTests.java
* A unit test class designed to test the business logic inside the AppointmentService or AppointmentServiceImpl in a medical system application.

2.DoctorServiceTests.java
* A unit test class that verifies the correctness of the DoctorService (typically implemented by DoctorServiceImpl). It ensures that the business logic related to doctor management behaves as expected.

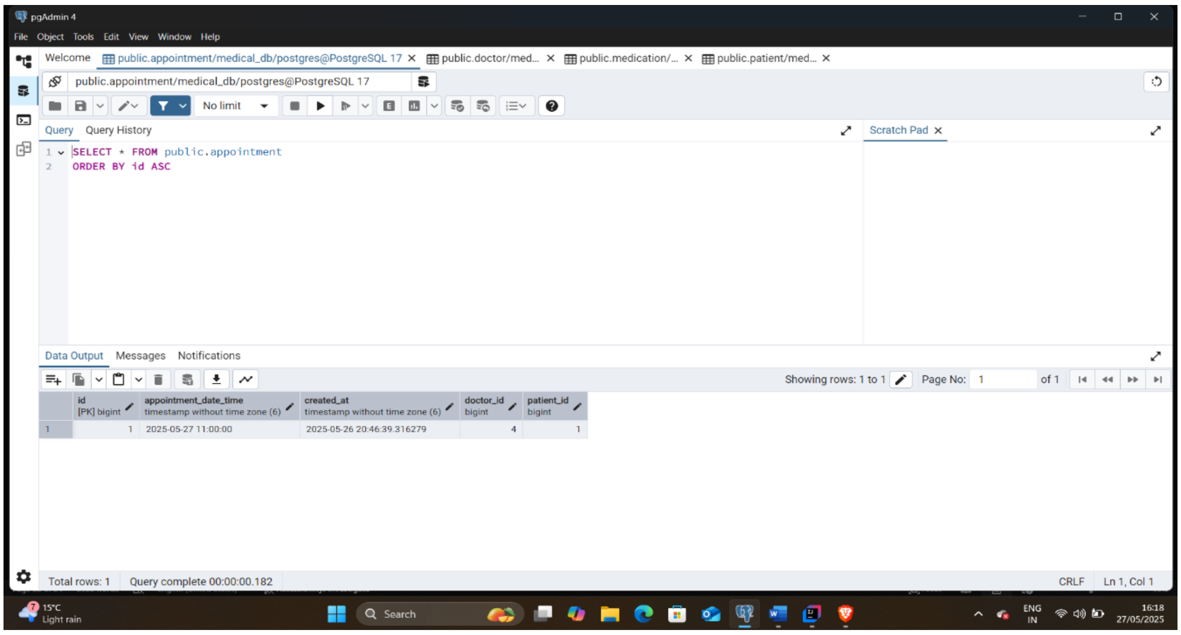
3.MedicationServiceTests.java
* A unit test class that verifies the correctness of the business logic in the MedicationService or MedicationServiceImpl class.

4.PatientServiceTests.java
* A unit test class used to verify the behavior of the PatientService (often PatientServiceImpl). It ensures that patient-related business logic works as expected, independently of other components like the database or controller.

Table Schema

1) Appointment Table

Column	Type	Description
id	long	Appointment unique identifier
patient_Id	bigint	ID Number of Patient
doctor_Id	bigint	ID Number of Doctor
appointmentDateTime	timestamp without timezone	Appointment date and time decided by patient
createdAt	timestamp without timezone	Appointment created date and time by patient



```
SELECT * FROM public.appointment
ORDER BY id ASC
```

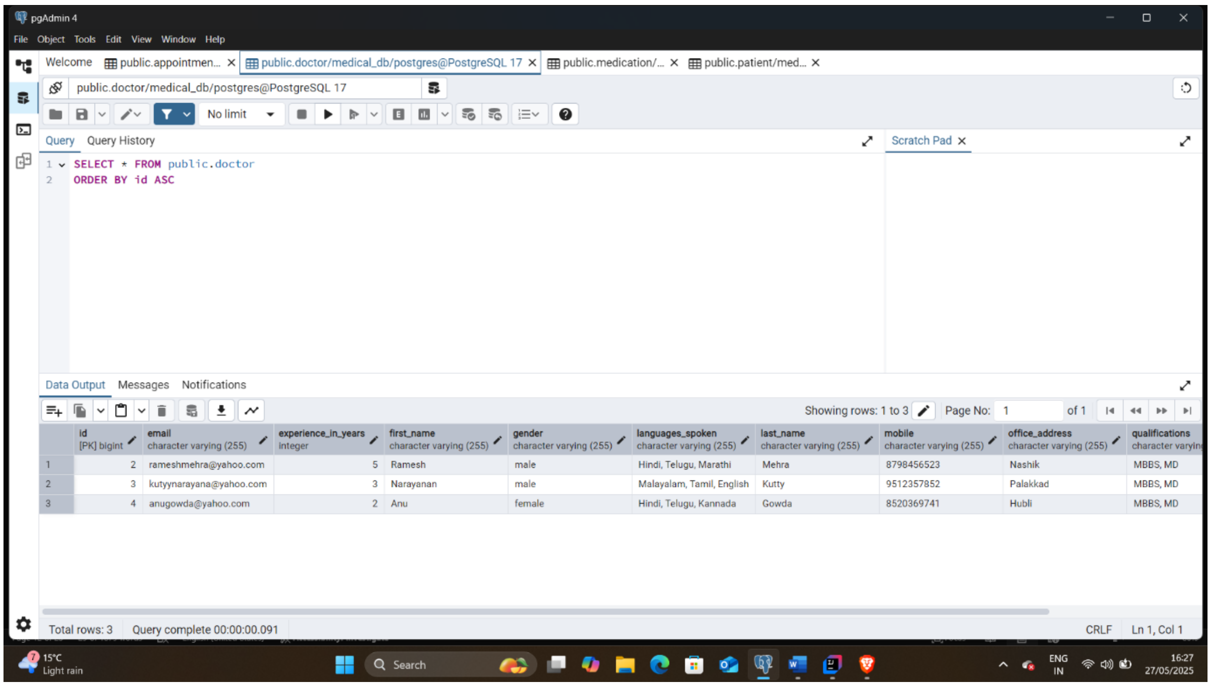
Showing rows: 1 to 1 Page No: 1 of 1

	id [PK] bigint	appointment_date_time timestamp without time zone (6)	created_at timestamp without time zone (6)	doctor_id bigint	patient_id bigint
1	1	2025-05-27 11:00:00	2025-05-26 20:46:39.316279	4	1

Total rows: 1 Query complete 00:00:00.182 CRLF Ln 1, Col 1

2) Doctor Table

Column	Type	Description
Id	Long	Doctor unique identifier
firstName	String	First Name of Doctor
lastName	String	Last Name of Doctor
gender	String	Gender of Doctor
mobile	String	Contact number of doctors
email	String	Mail ID of doctors
speciality	String	Specialty of doctors
experienceInYears	int	Number of years' experience for doctor
qualifications	String	Qualifications of a doctor
languagesSpoken	String	Languages spoken by doctor
officeAddress	String	Location of doctors



```
SELECT * FROM public.doctor
ORDER BY id ASC
```

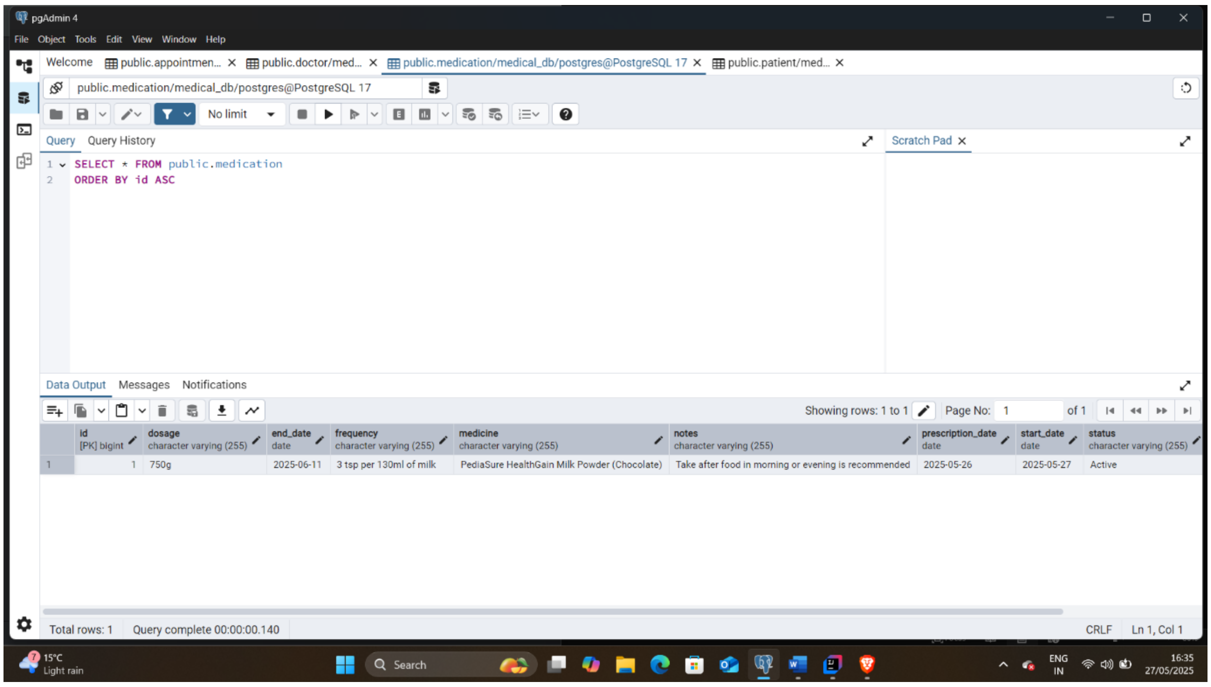
Showing rows: 1 to 3

	id [PK] bigint	email character varying (255)	experience_in_years integer	first_name character varying (255)	gender character varying (255)	languages_spoken character varying (255)	last_name character varying (255)	mobile character varying (255)	office_address character varying (255)	qualifications character varying
1	2	rameshmehra@yahoo.com	5	Ramesh	male	Hindi, Telugu, Marathi	Mehra	8798456523	Nashik	MBBS, MD
2	3	kutyynarayana@yahoo.com	3	Narayanan	male	Malayalam, Tamil, English	Kutty	9512357852	Palakkad	MBBS, MD
3	4	anugowda@yahoo.com	2	Anu	female	Hindi, Telugu, Kannada	Gowda	8520369741	Hubli	MBBS, MD

Total rows: 3 Query complete 00:00:00.091

3) Medication Table

Column	Type	Description
Id	Long	Medication unique identifier
patient_id	bigint	ID Number of Patient
medcine	String	Name of the medicine
dosage	String	Dosage for the medicine
frequency	String	Frequency for taking medicines
status	String	Status of medicines whether active, completed or hold.
startDate	LocalDate	Medication starting date
endDate	LocalDate	Medication ending date
prescriptionDate	LocalDate	Date of prescription given by doctor
updatedDate	LocalDate	Updated date for the prescription
notes	String	Notes to convey to the patients



```
SELECT * FROM public.medication
ORDER BY id ASC
```

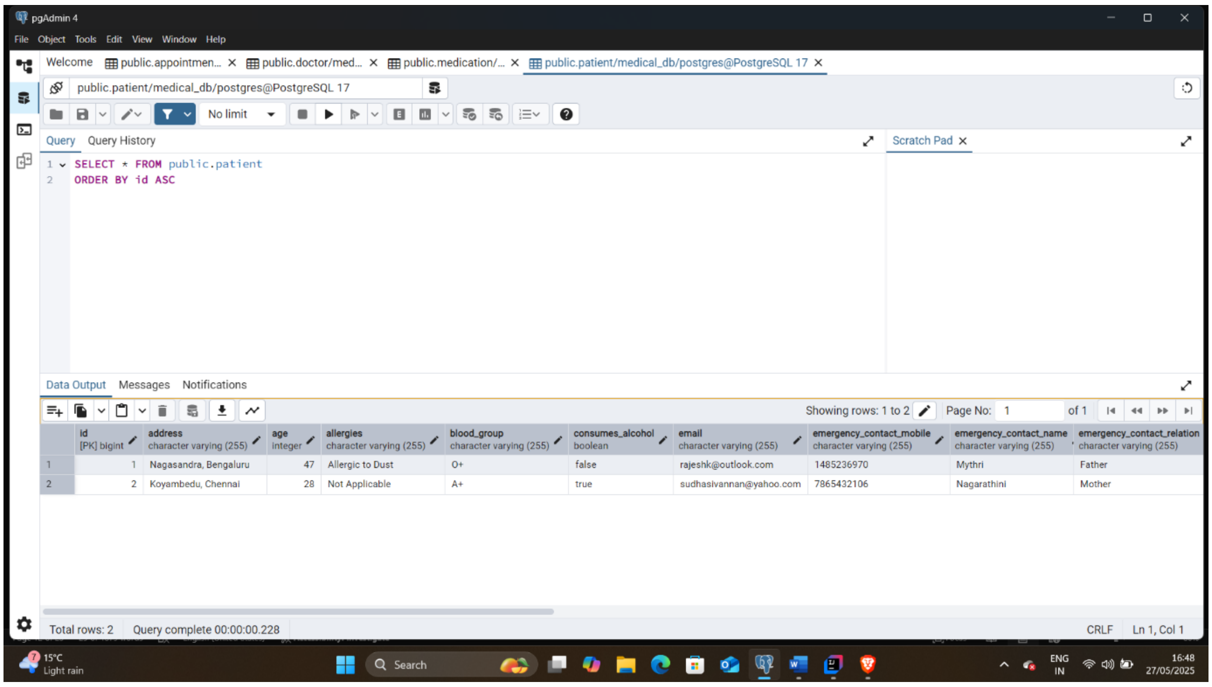
Showing rows: 1 to 1

	id [PK] bigint	dosage character varying (255)	end_date date	frequency character varying (255)	medicine character varying (255)	notes character varying (255)	prescription_date date	start_date date	status character varying (255)
1	1	750g	2025-06-11	3 tsp per 130ml of milk	PediaSure HealthGain Milk Powder (Chocolate)	Take after food in morning or evening is recommended	2025-05-26	2025-05-27	Active

Total rows: 1 Query complete 00:00:00.140

4) Patient Table

Column	Type	Description
Id	Long	Patient unique identifier
firstName	String	First Name of patient
lastName	String	Last Name of patient
email	String	Email of patient
mobile	String	Contact number of patients
password	String	Password of patient to login
address	String	Location of patient
age	int	Age of patient
gender	String	Gender of patient
bloodGroup	String	Blood Group of patients
emergencyContactName	String	Emergency contact name of patient's relatives
emergencyContactMobile	String	Emergency contact number of patient's relatives
emergencyContactRelation	String	Emergency contact relationship of patient's relatives
previousDiagnoses	String	Previous diagnoses of patient
surgeries	String	Any surgeries happened for patients
allergies	String	Any allergies happening to patients
vaccinationHistory	String	Any vaccinations happened for patients
isSmoker	boolean	Is patient a smoker?
consumesAlcohol	boolean	Is patient consumes alcohol?



```sql
SELECT * FROM public.patient
ORDER BY id ASC
```

	id [PK] bigint	address character varying (255)	age integer	allergies character varying (255)	blood_group character varying (255)	consumes_alcohol boolean	email character varying (255)	emergency_contact_mobile character varying (255)	emergency_contact_name character varying (255)	emergency_contact_relation character varying (255)
1	1	Nagasandra, Bengaluru	47	Allergic to Dust	O+	false	rajeshk@outlook.com	1485236970	Mythri	Father
2	2	Koyambedu, Chennai	28	Not Applicable	A+	true	sudhasivannan@yahoo.com	7865432106	Nagarathini	Mother

Total rows: 2 Query complete 00:00:00.228 CRLF Ln 1, Col 1

API Endpoints

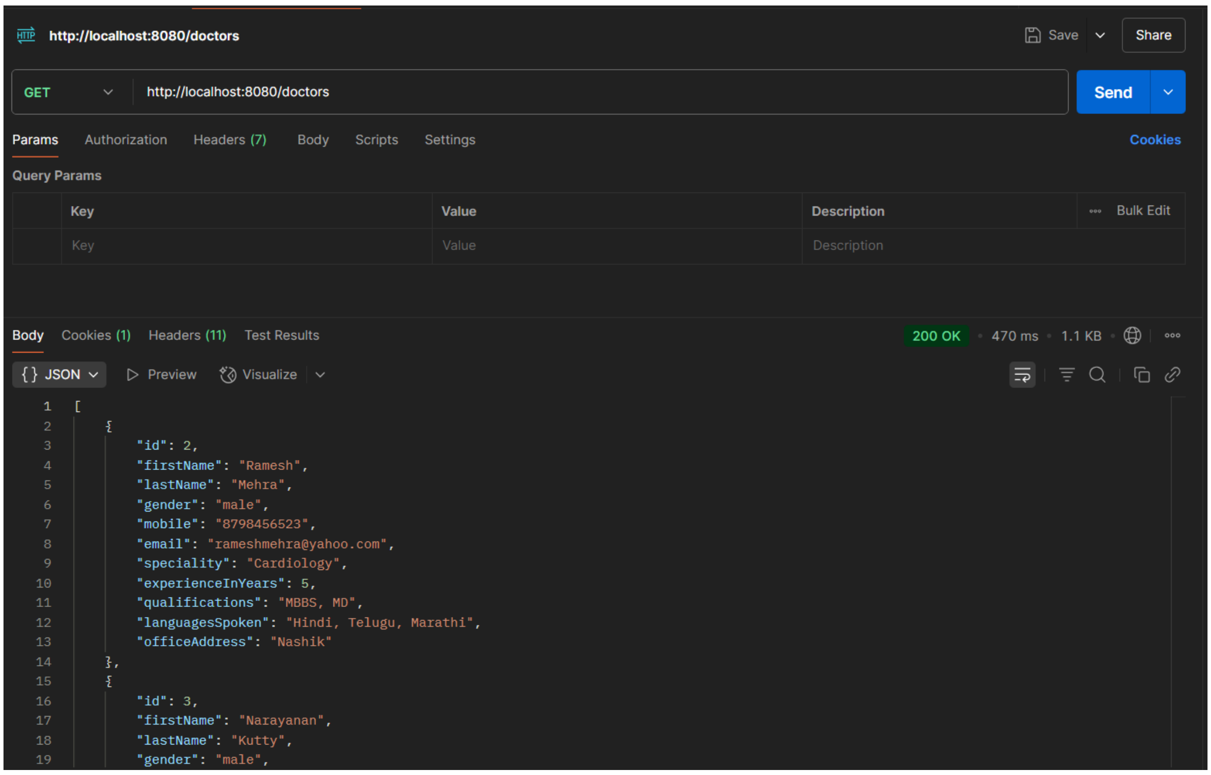
DoctorController.java

1.Get all Doctors

Method: ‘GET’

Endpoint: ‘/doctors’

Description: Gives the list of the doctors.



GET http://localhost:8080/doctors

Key	Value	Description
Key	Value	Description

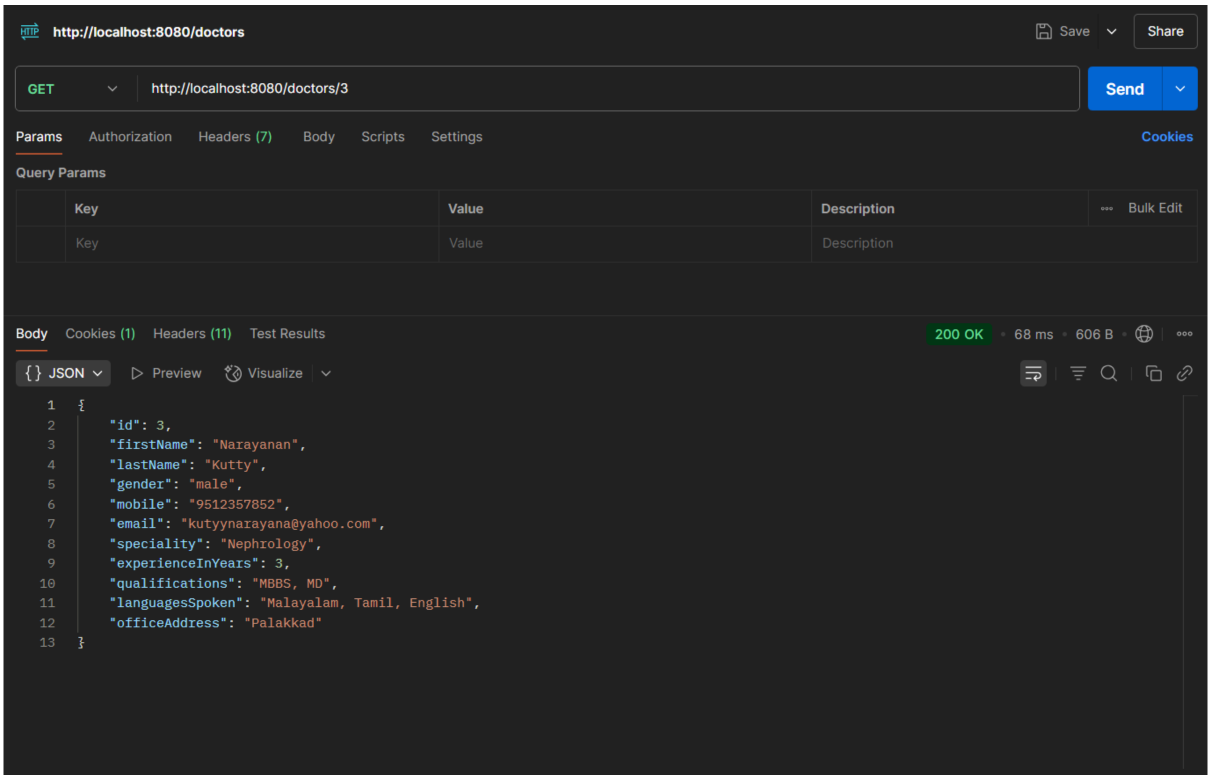
```
[
    {
        "id": 2,
        "firstName": "Ramesh",
        "lastName": "Mehra",
        "gender": "male",
        "mobile": "8798456523",
        "email": "rameshmehra@yahoo.com",
        "speciality": "Cardiology",
        "experienceInYears": 5,
        "qualifications": "MBBS, MD",
        "languagesSpoken": "Hindi, Telugu, Marathi",
        "officeAddress": "Nashik"
    },
    {
        "id": 3,
        "firstName": "Narayanan",
        "lastName": "Kutty",
        "gender": "male",
```

2.Get doctor by ID

Method: 'GET'

Endpoint: '/doctors/{id}'

Description: Gives the particular doctor detail by ID.



http://localhost:8080/doctors/3

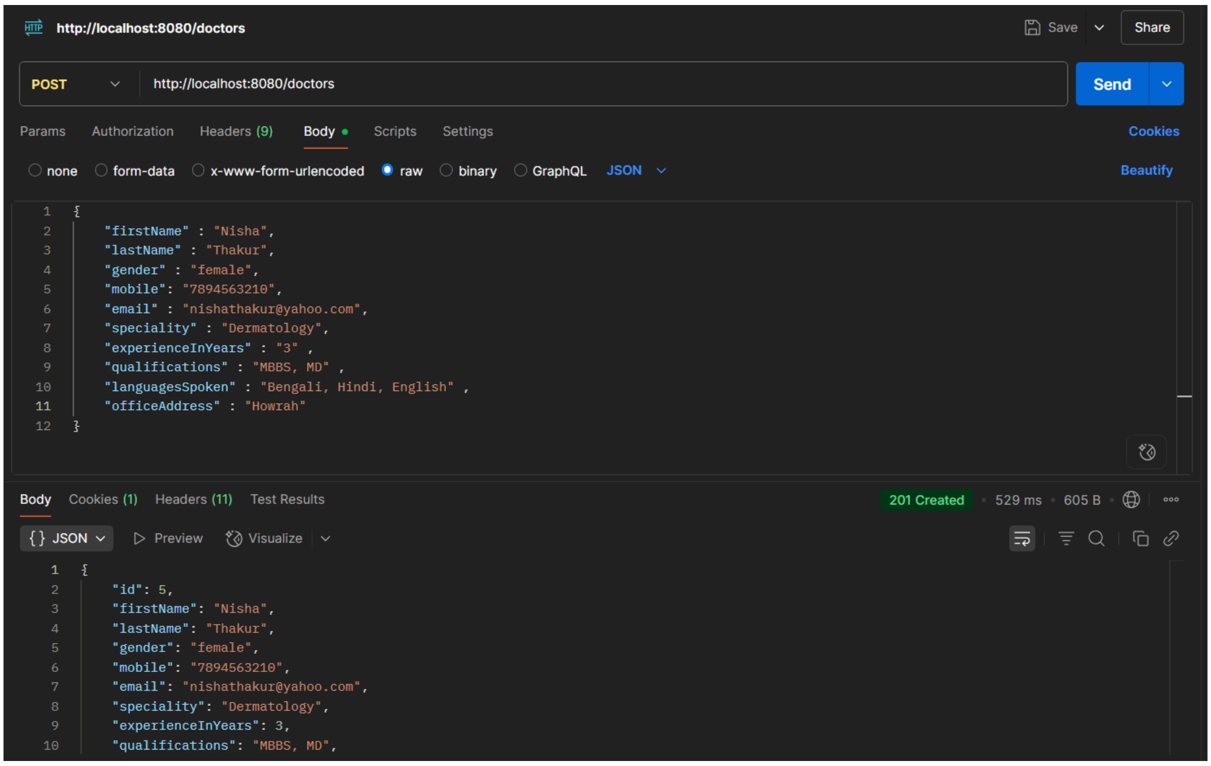
```
{
    "id": 3,
    "firstName": "Narayanan",
    "lastName": "Kutty",
    "gender": "male",
    "mobile": "9512357852",
    "email": "kutyynarayana@yahoo.com",
    "speciality": "Nephrology",
    "experienceInYears": 3,
    "qualifications": "MBBS, MD",
    "languagesSpoken": "Malayalam, Tamil, English",
    "officeAddress": "Palakkad"
}
```

3.Add a single doctor

Method: 'POST'

Endpoint: '/doctors'

Description: Create a new doctor record.



POST http://localhost:8080/doctors

```
{
    "firstName" : "Nisha",
    "lastName" : "Thakur",
    "gender" : "female",
    "mobile": "7894563210",
    "email" : "nishathakur@yahoo.com",
    "speciality" : "Dermatology",
    "experienceInYears" : "3" ,
    "qualifications" : "MBBS, MD" ,
    "languagesSpoken" : "Bengali, Hindi, English" ,
    "officeAddress" : "Howrah"
}
```

Body 201 Created 529 ms 605 B

JSON

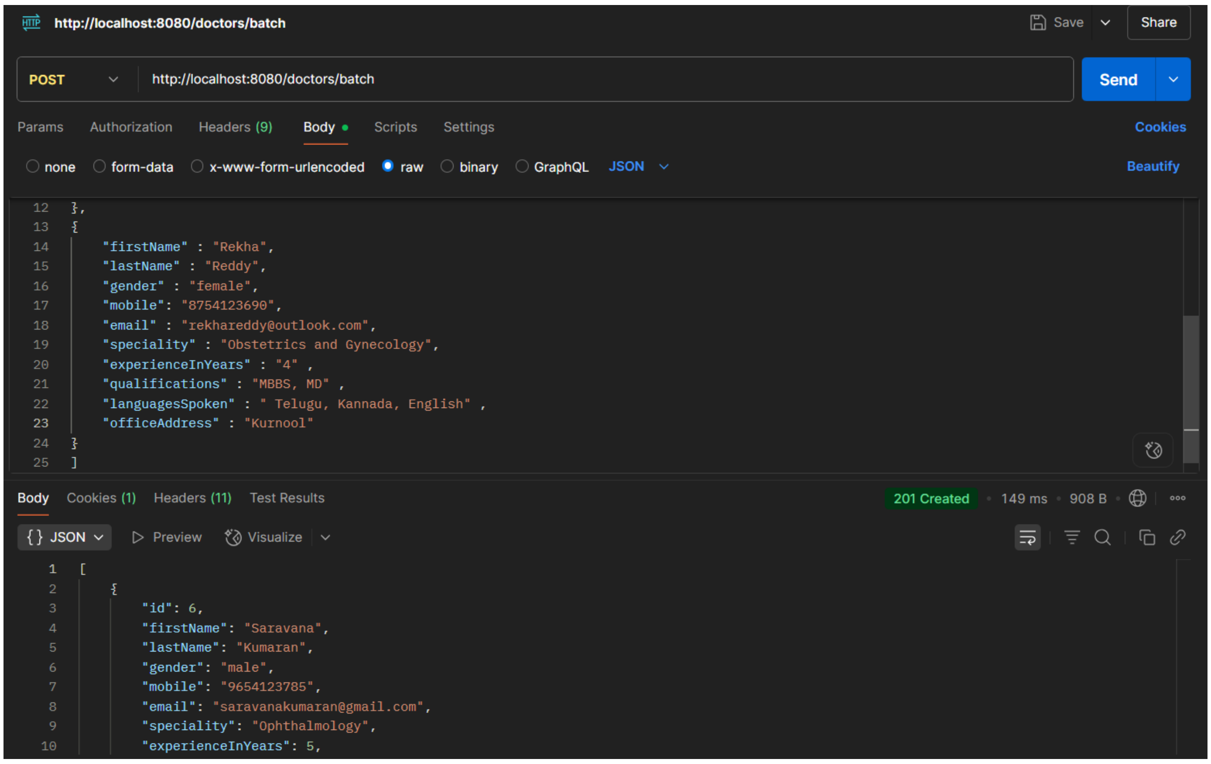
```
{
    "id": 5,
    "firstName": "Nisha",
    "lastName": "Thakur",
    "gender": "female",
    "mobile": "7894563210",
    "email": "nishathakur@yahoo.com",
    "speciality": "Dermatology",
    "experienceInYears": 3,
    "qualifications": "MBBS, MD",
```

4.Add multiple doctors in a single attempt

Method: ‘POST’

Endpoint: ‘/doctors/batch’

Description: Create multiple doctors records in a single time.



```
},
{
    "firstName" : "Rekha",
    "lastName" : "Reddy",
    "gender" : "female",
    "mobile": "8754123690",
    "email" : "rekhareddy@outlook.com",
    "speciality" : "Obstetrics and Gynecology",
    "experienceInYears" : "4" ,
    "qualifications" : "MBBS, MD" ,
    "languagesSpoken" : " Telugu, Kannada, English" ,
    "officeAddress" : "Kurnool"
}
]
```

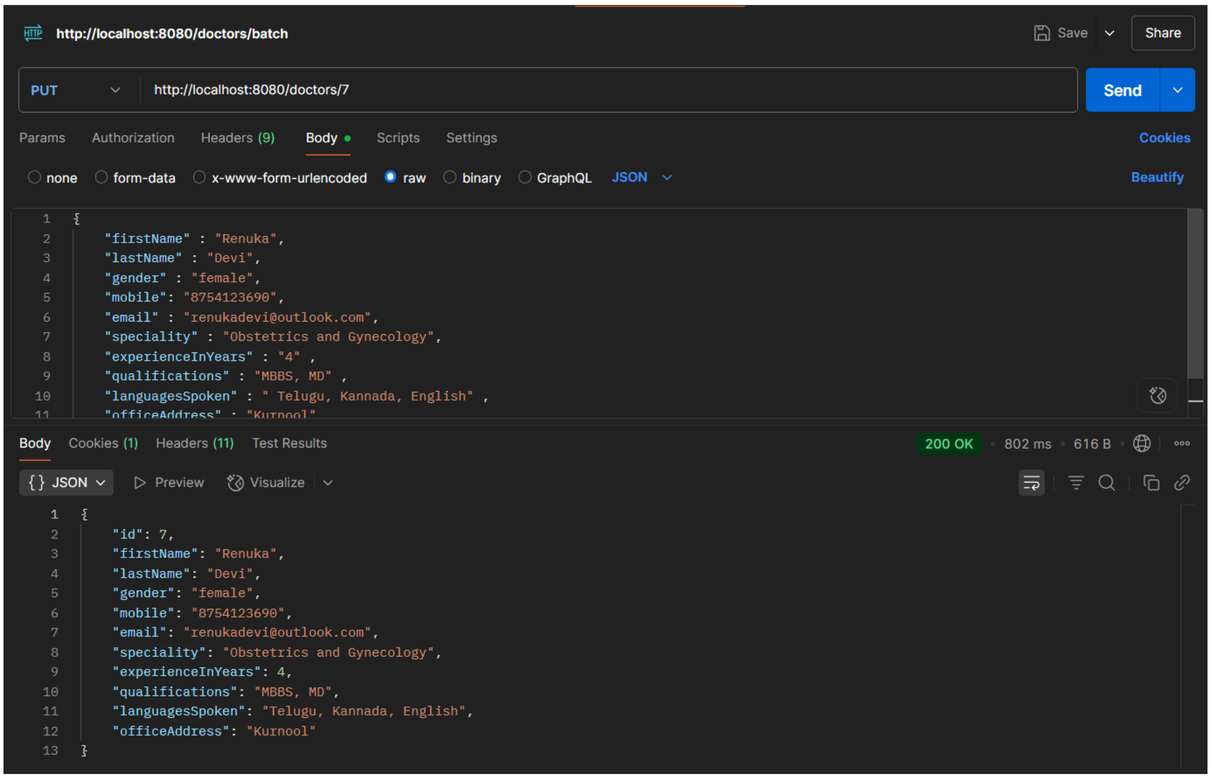
```
[
    {
        "id": 6,
        "firstName": "Saravana",
        "lastName": "Kumaran",
        "gender": "male",
        "mobile": "9654123785",
        "email": "saravanakumaran@gmail.com",
        "speciality": "Ophthalmology",
        "experienceInYears": 5,
```

5.Update a doctor record

Method: 'PUT'

Endpoint: '/doctors/{id}'

Description: Update the existing doctor record.



PUT http://localhost:8080/doctors/7

```
{
    "firstName" : "Renuka",
    "lastName" : "Devi",
    "gender" : "female",
    "mobile": "8754123690",
    "email" : "renukadevi@outlook.com",
    "speciality" : "Obstetrics and Gynecology",
    "experienceInYears" : "4" ,
    "qualifications" : "MBBS, MD" ,
    "languagesSpoken" : " Telugu, Kannada, English" ,
    "officeAddress" : "Kurnool"
```

200 OK

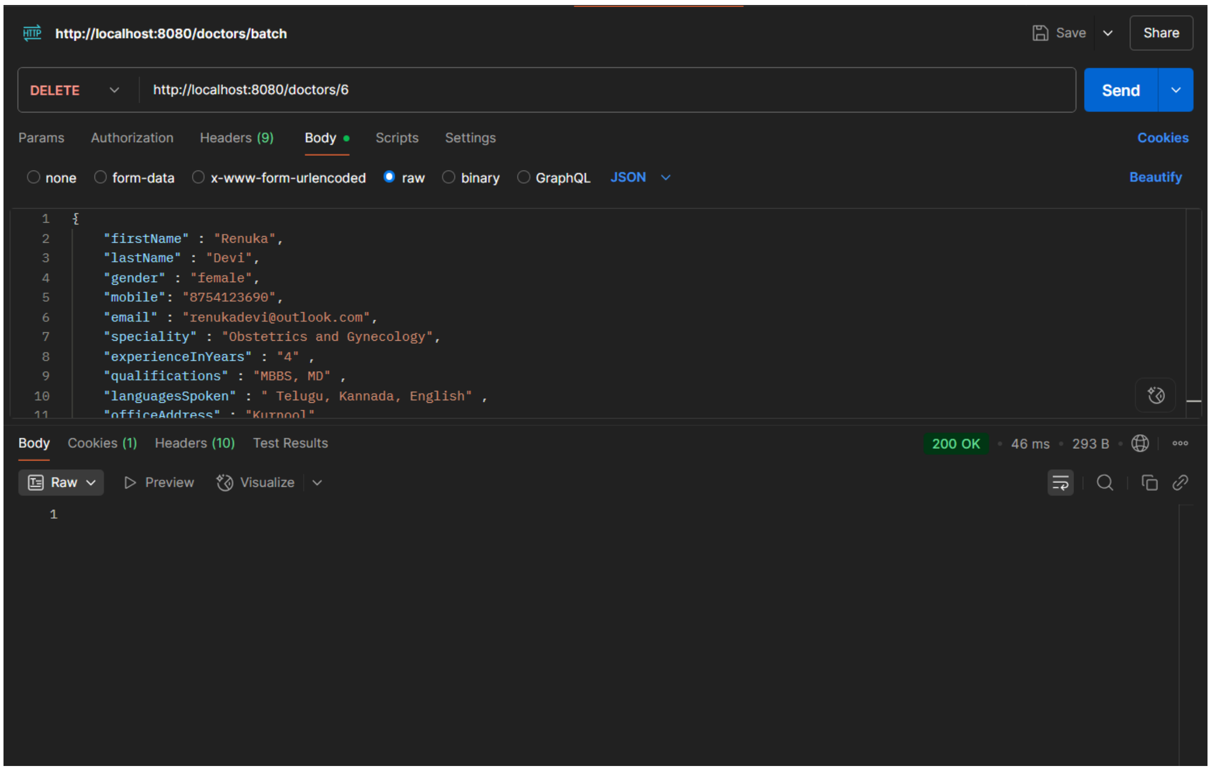
```
{
    "id": 7,
    "firstName": "Renuka",
    "lastName": "Devi",
    "gender": "female",
    "mobile": "8754123690",
    "email": "renukadevi@outlook.com",
    "speciality": "Obstetrics and Gynecology",
    "experienceInYears": 4,
    "qualifications": "MBBS, MD",
    "languagesSpoken": "Telugu, Kannada, English",
    "officeAddress": "Kurnool"
}
```

6.Delete a doctor record

Method: 'PUT'

Endpoint: '/doctors/{id}'

Description: Delete a doctor record.



```
DELETE http://localhost:8080/doctors/6
```

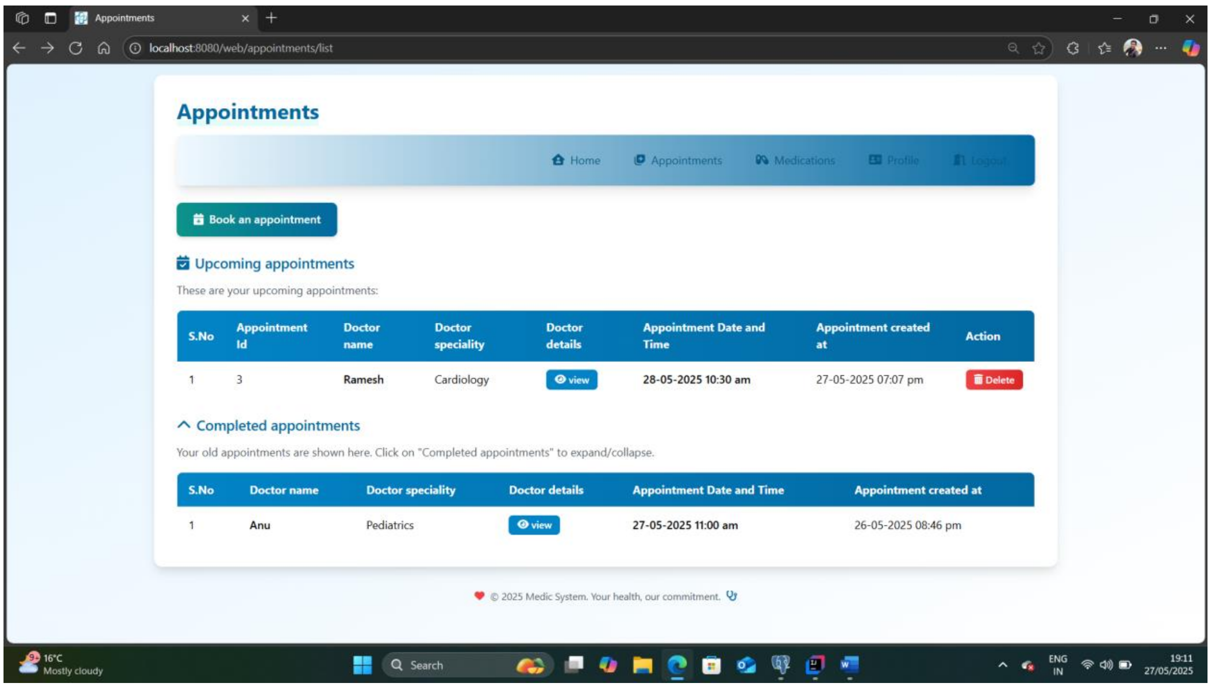
```
{
    "firstName" : "Renuka",
    "lastName" : "Devi",
    "gender" : "female",
    "mobile": "8754123690",
    "email" : "renukadevi@outlook.com",
    "speciality" : "Obstetrics and Gynecology",
    "experienceInYears" : "4" ,
    "qualifications" : "MBBS, MD" ,
    "languagesSpoken" : " Telugu, Kannada, English" ,
    "officeAddress" : "Kurnool"
```

Web Endpoints

AppointmentWebController.java

1. **Getting the appointment table**

   - **Endpoint:** /web/appointments/list
   - **Method:** GET
   - **Description:** Gets all the scheduled appointments of the patient.



Appointments

Upcoming appointments

These are your upcoming appointments:

S.No	Appointment Id	Doctor name	Doctor speciality	Doctor details	Appointment Date and Time	Appointment created at	Action
1	3	Ramesh	Cardiology	view	28-05-2025 10:30 am	27-05-2025 07:07 pm	Delete

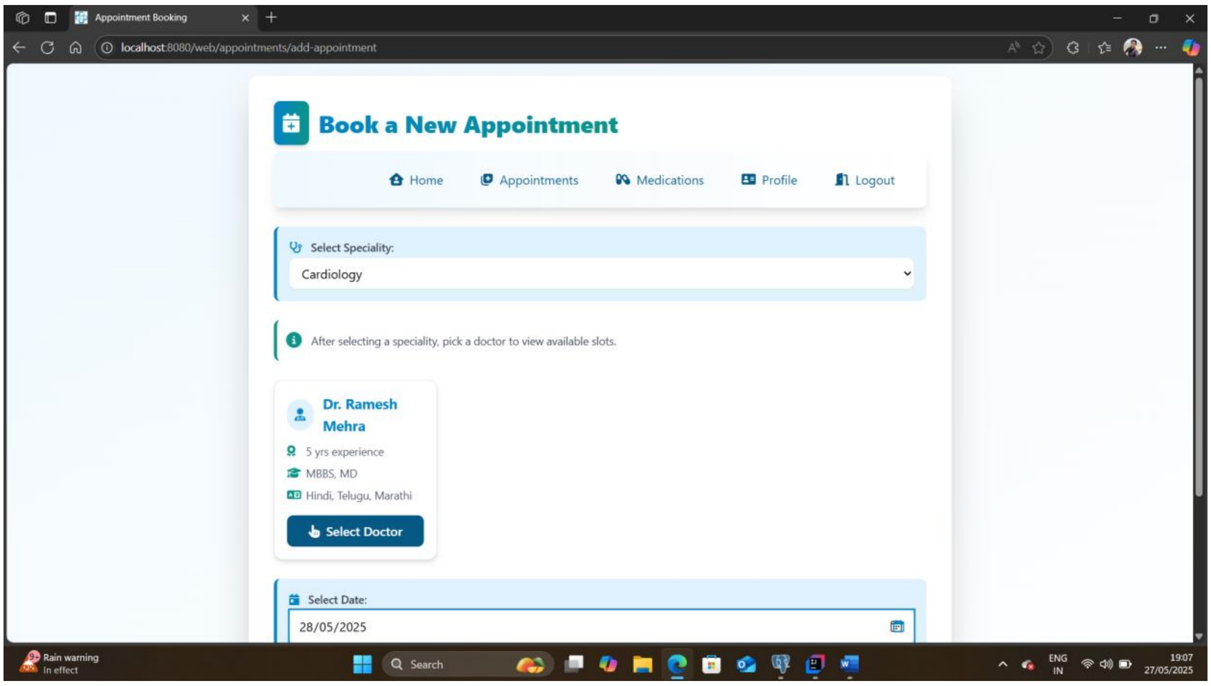
Completed appointments

Your old appointments are shown here. Click on "Completed appointments" to expand/collapse.

S.No	Doctor name	Doctor speciality	Doctor details	Appointment Date and Time	Appointment created at
1	Anu	Pediatrics	view	27-05-2025 11:00 am	26-05-2025 08:46 pm

© 2025 Medic System. Your health, our commitment.

2. **Render form to get the new appointment details**
    - **Endpoint:** /web/appointments/add-appointment
    - **Method:** GET
    - **Description:** A method to get new appointment details through a form.


localhost:8080/web/appointments/add-appointment

Book a New Appointment

Select Speciality: Cardiology

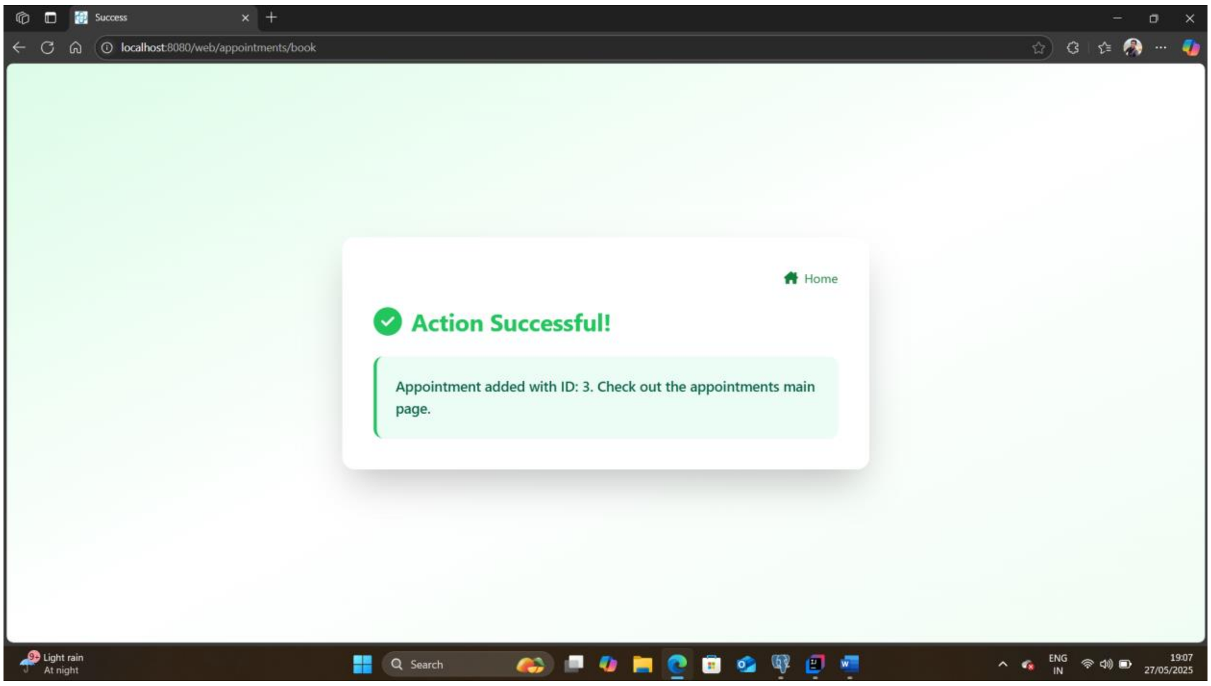
After selecting a speciality, pick a doctor to view available slots.

Dr. Ramesh Mehra
5 yrs experience
MBBS, MD
Hindi, Telugu, Marathi

Select Date: 28/05/2025

3. **Book an appointment**

- **Endpoint:** /web/appointments/book
- **Method:** POST
- **Description:** A method to book appointment once all the appointment details are submitted through the form.

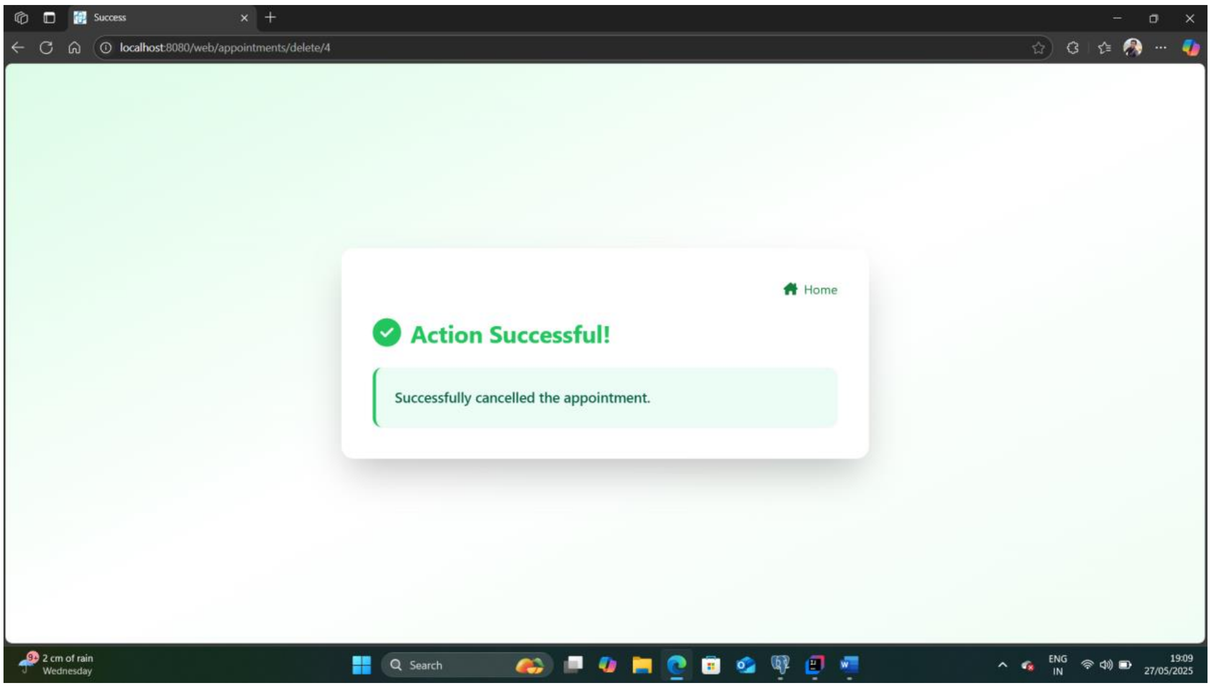


Action Successful!

Appointment added with ID: 3. Check out the appointments main page.

4. **Delete an appointment**

- **Endpoint:** /web/ appointments/delete/{id}
- **Method:** GET
- **Description:** Deletes the appointment object upon clicking delete from the webpage.

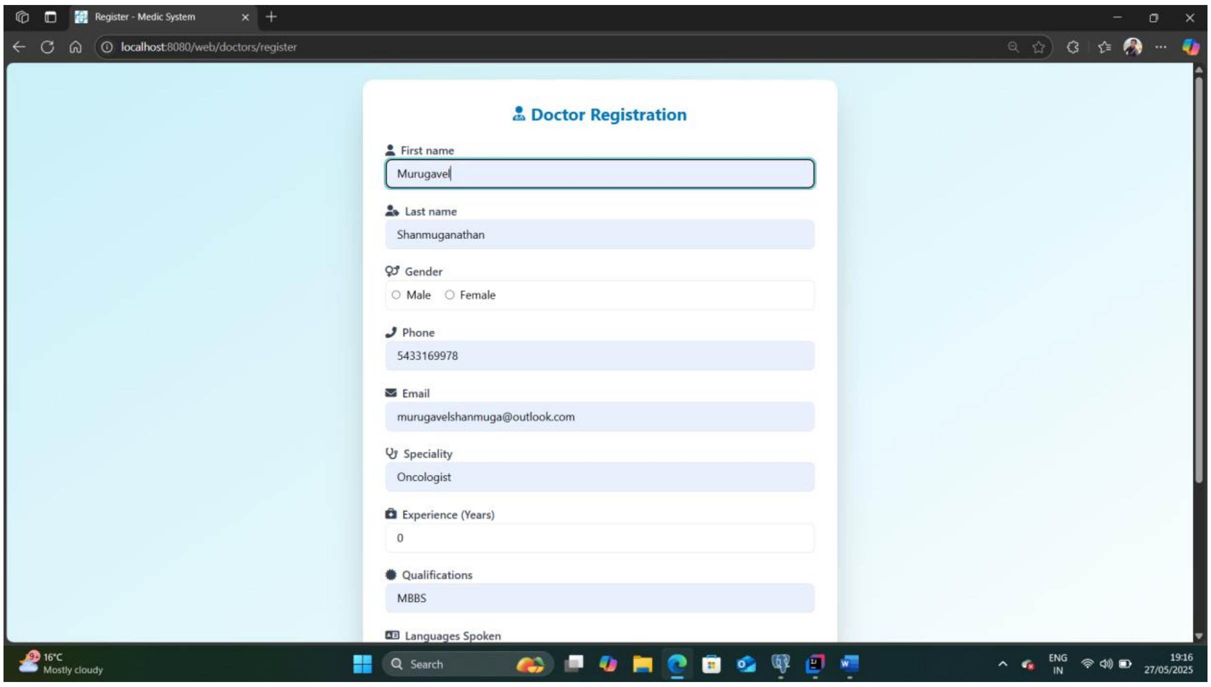


Action Successful!

Successfully cancelled the appointment.

<u>Doctorwebcontroller.java</u>

1. **Render registration page**
   - **Endpoint: /web/doctors/register**
   - **Method: GET**
   - **Description: Renders a page with form to collect new doctor's details.**



Register - Medic System

localhost:8080/web/doctors/register

Doctor Registration

First name: Murugavel

Last name: Shanmuganathan

Gender: Male Female

Phone: 5433169978

Email: murugavelshanmuga@outlook.com

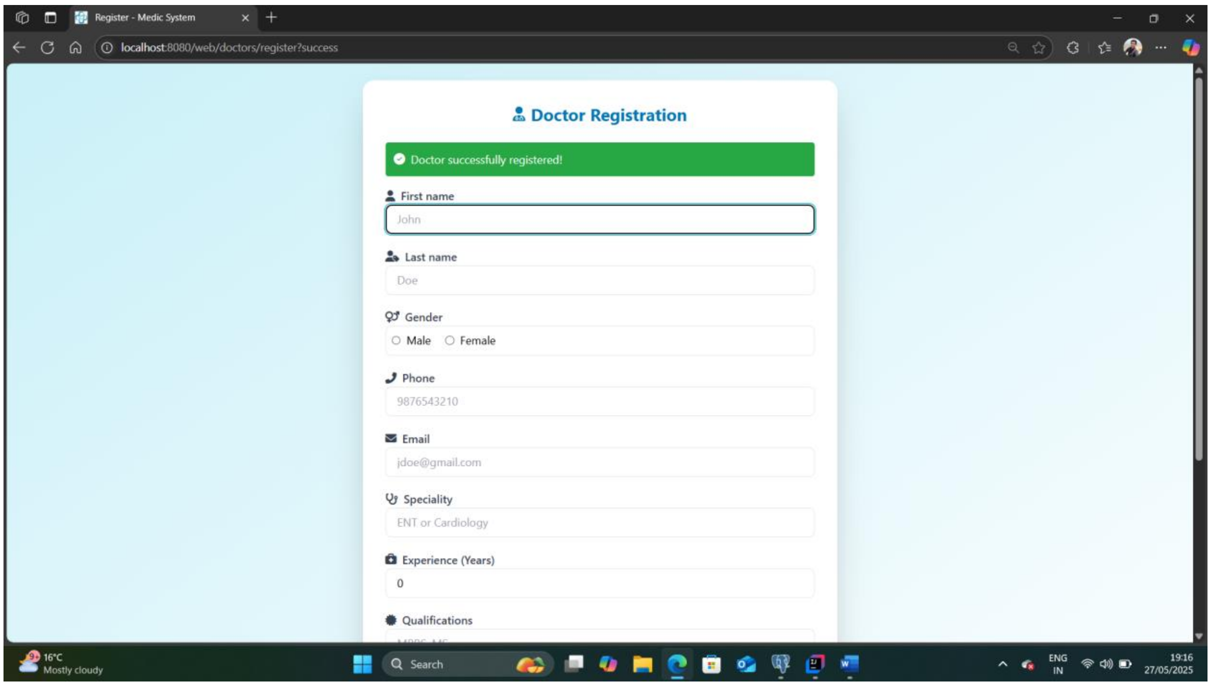
Speciality: Oncologist

Experience (Years): 0

Qualifications: MBBS

Languages Spoken

16°C Mostly cloudy · Search · ENG IN · 19:16 27/05/2025



Register - Medic System

localhost:8080/web/doctors/register?success

Doctor Registration

Doctor successfully registered!

First name: John

Last name: Doe

Gender: Male Female

Phone: 9876543210

Email: jdoe@gmail.com

Speciality: ENT or Cardiology

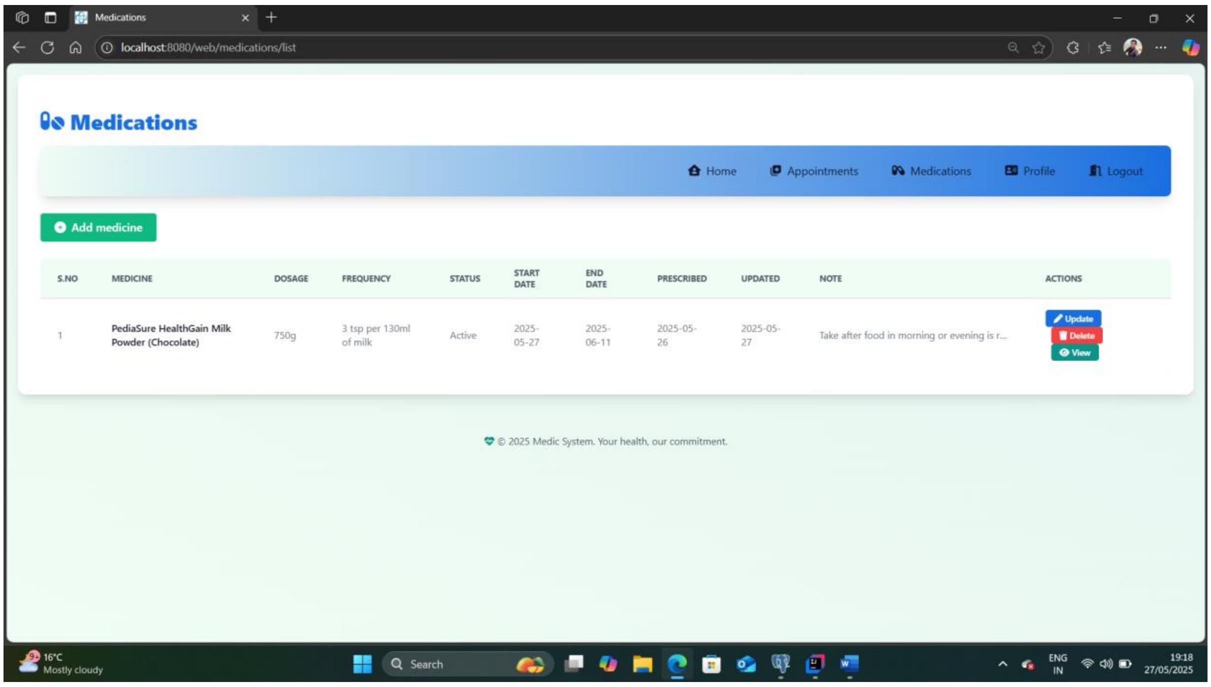
Experience (Years): 0

Qualifications

16°C Mostly cloudy · Search · ENG IN · 19:16 27/05/2025

MedicationWebController.java

1. **Getting the medication table**
   - **Endpoint:** /web/medications/list
   - **Method:** GET
   - **Description:** Gets all the available medication of the user.



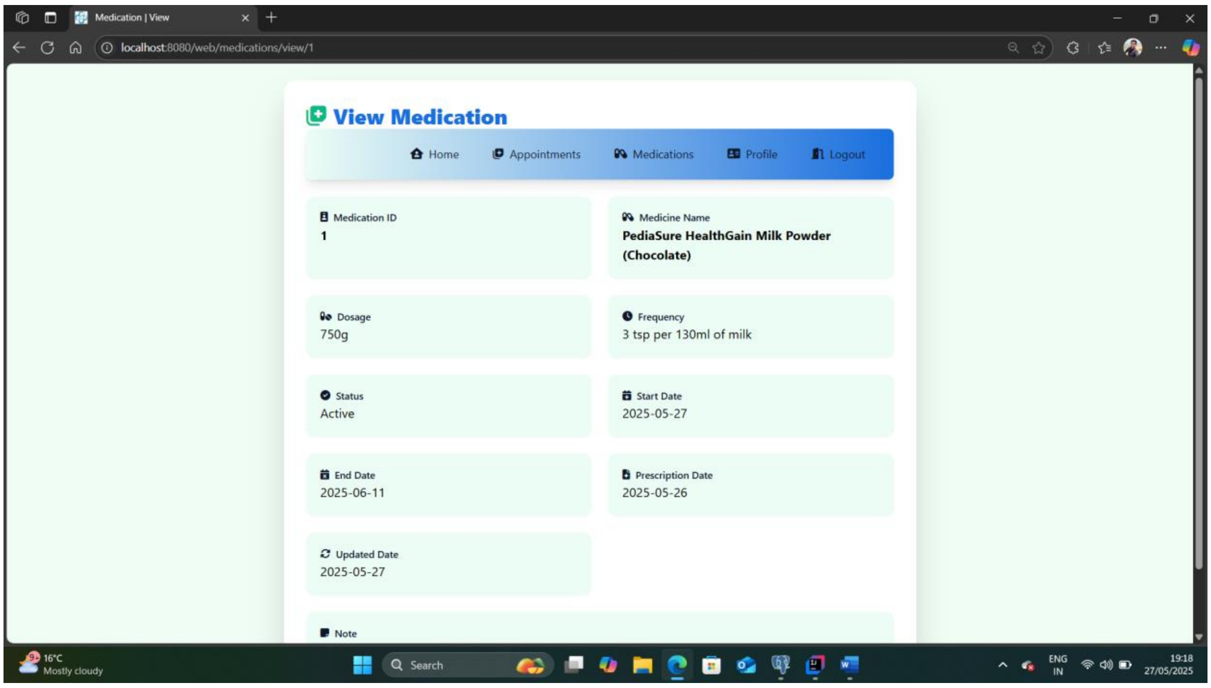
Medications

S.NO	MEDICINE	DOSAGE	FREQUENCY	STATUS	START DATE	END DATE	PRESCRIBED	UPDATED	NOTE	ACTIONS
1	PediaSure HealthGain Milk Powder (Chocolate)	750g	3 tsp per 130ml of milk	Active	2025-05-27	2025-06-11	2025-05-26	2025-05-27	Take after food in morning or evening is r...	Update Delete View

© 2025 Medic System. Your health, our commitment.

2. Viewing a particular medication

- **Endpoint:** /web/medications/view/{id}
- **Method:** GET
- **Description:** To view a particular medication in detail, like when it was added or updated.



Medication ID
1

Medicine Name
PediaSure HealthGain Milk Powder (Chocolate)

Dosage
750g

Frequency
3 tsp per 130ml of milk

Status
Active

Start Date
2025-05-27

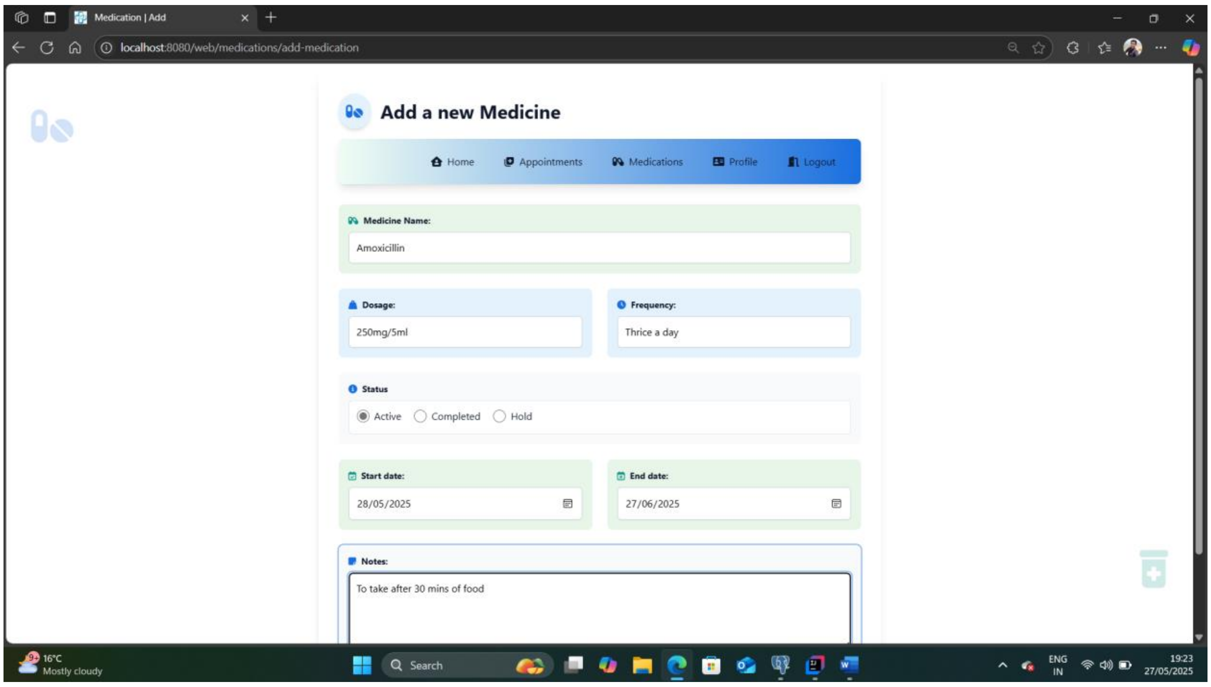
End Date
2025-06-11

Prescription Date
2025-05-26

Updated Date
2025-05-27

Note

3. **Render form to get the new medication details**
   - **Endpoint:** /web/medications/add-medication
   - **Method:** GET
   - **Description:** Renders the HTML form page for adding a new medication.


Medication | Add

localhost:8080/web/medications/add-medication

Add a new Medicine

Home | Appointments | Medications | Profile | Logout

Medicine Name: Amoxicillin

Dosage: 250mg/5ml

Frequency: Thrice a day

Status: [✓ Active] Completed Hold

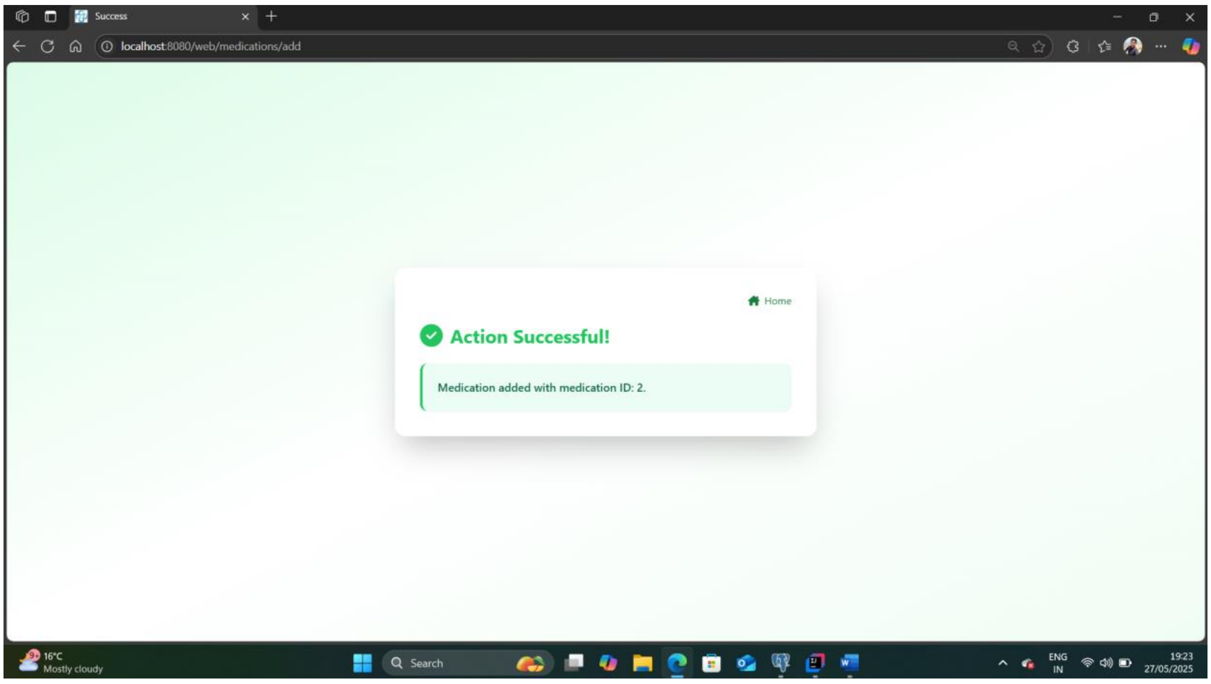
Start date: 28/05/2025

End date: 27/06/2025

Notes: To take after 30 mins of food

4. **Save the new medication to the database**

- **Endpoint:** /web/medications/add
- **Method:** POST
- **Description:** Makes a post request to save the medication details into the database.

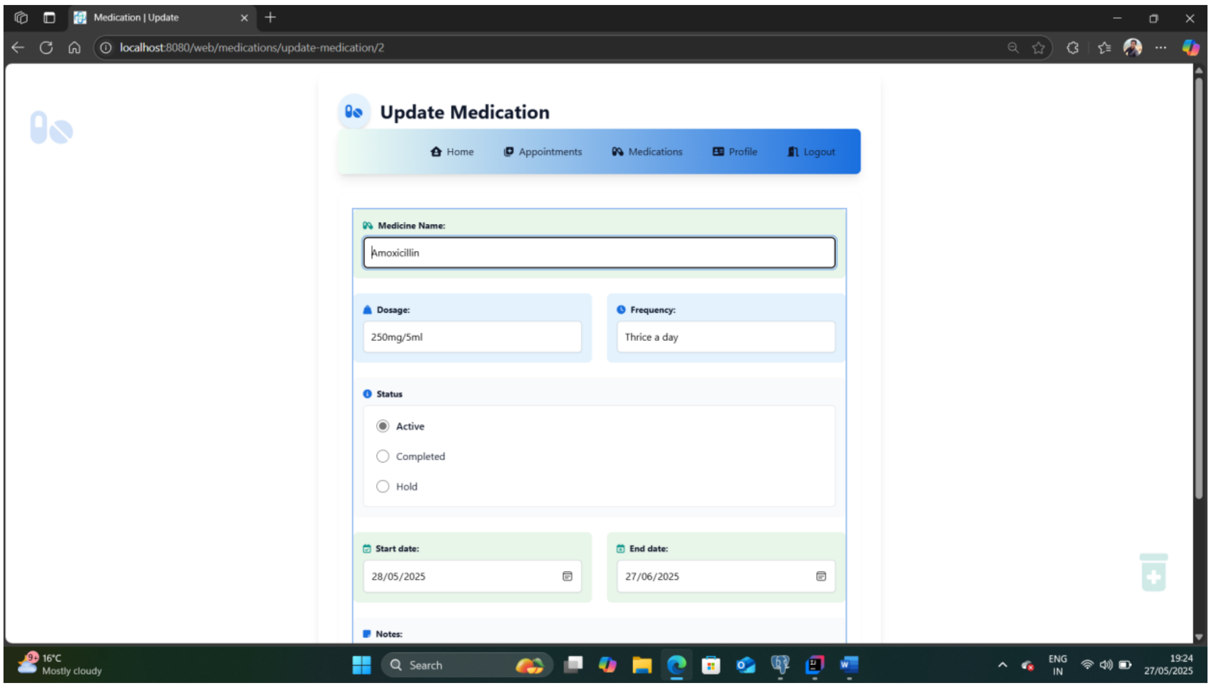


Action Successful!

Medication added with medication ID: 2.

5. **Update a medication with its id**

- **Endpoint:** /web/medications/update-medication/{id}
- **Method:** GET
- **Description:** Collects required data through the webpage for update.



Update Medication

Medicine Name: Amoxicillin

Dosage: 250mg/5ml

Frequency: Thrice a day

Status
- [✓ Active]
- Completed
- Hold

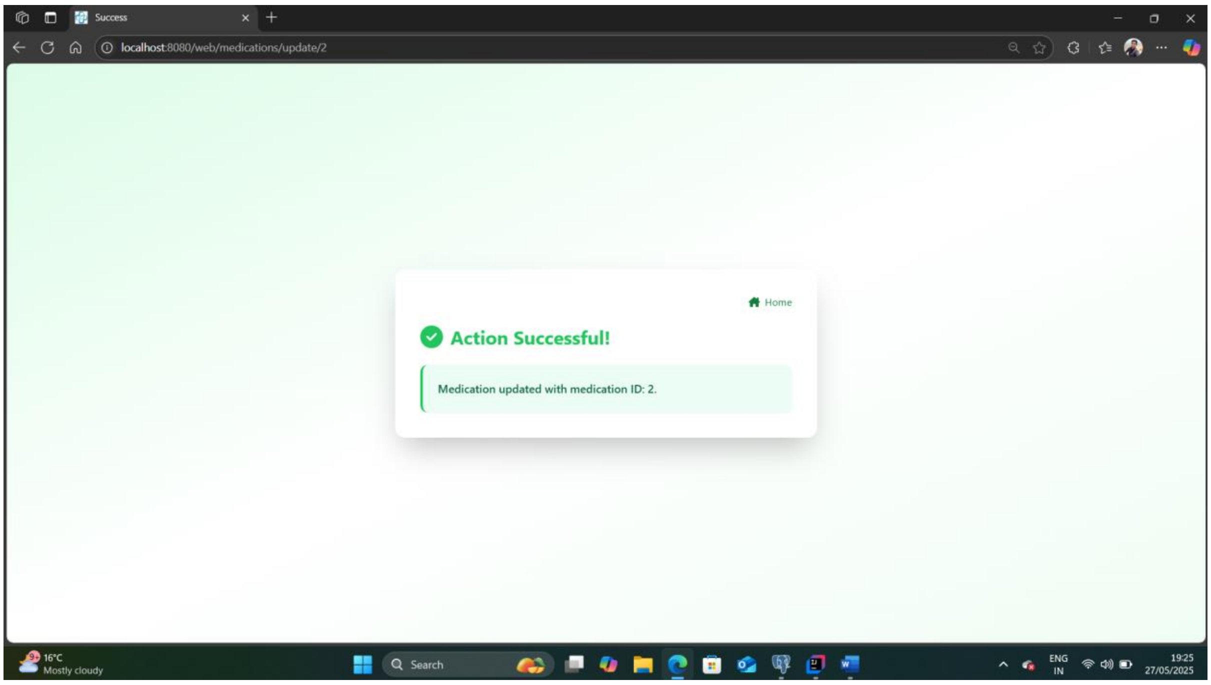
Start date: 28/05/2025

End date: 27/06/2025

Notes:

6. Update existing medication

- **Endpoint:** /web/medications/update/{id}
- **Method:** POST
- **Description:** Makes a post request to update the doctor details in the database.

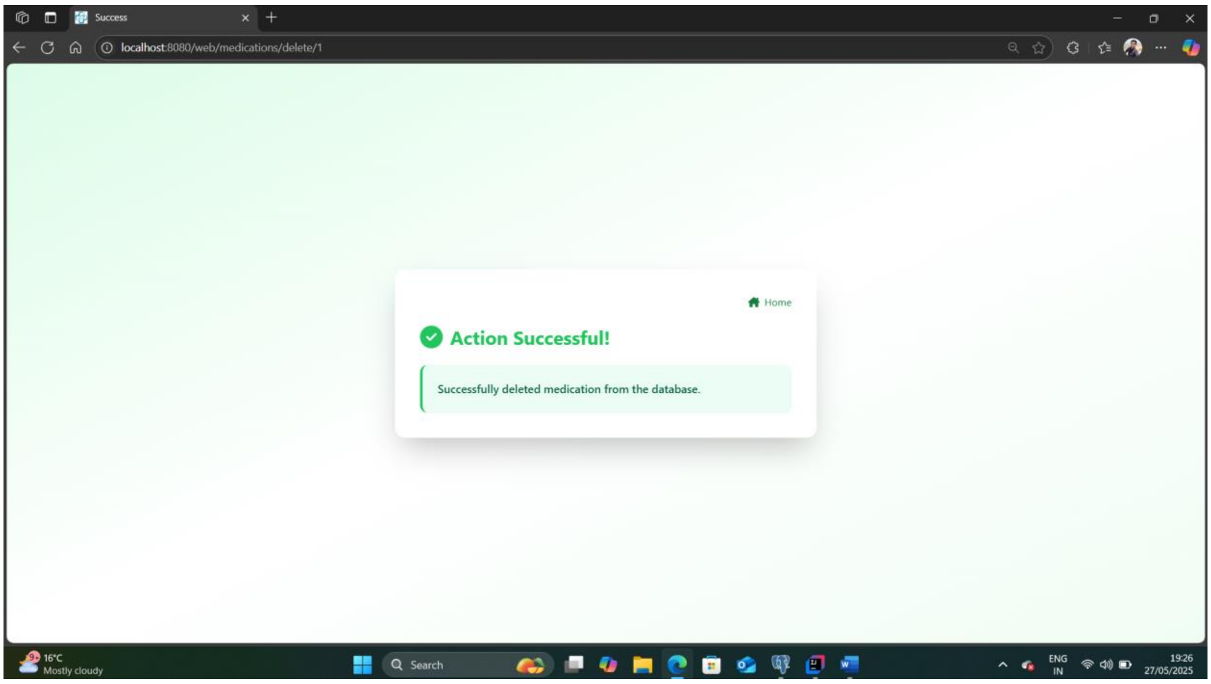


Action Successful!

Medication updated with medication ID: 2.

7. **Delete a medication**

- **Endpoint:** /web/ medications/delete/{id}
- **Method:** GET
- **Description:** Deletes selected medication from the medication list.

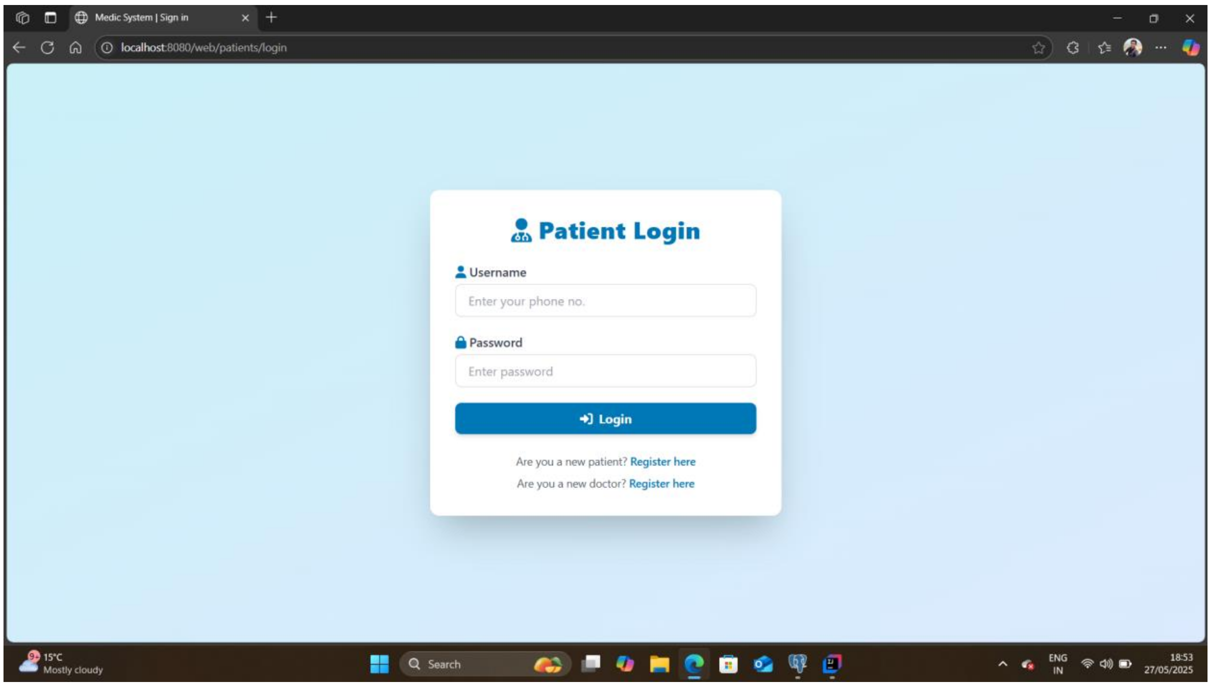


Action Successful!

Successfully deleted medication from the database.

PatientAuthController.java

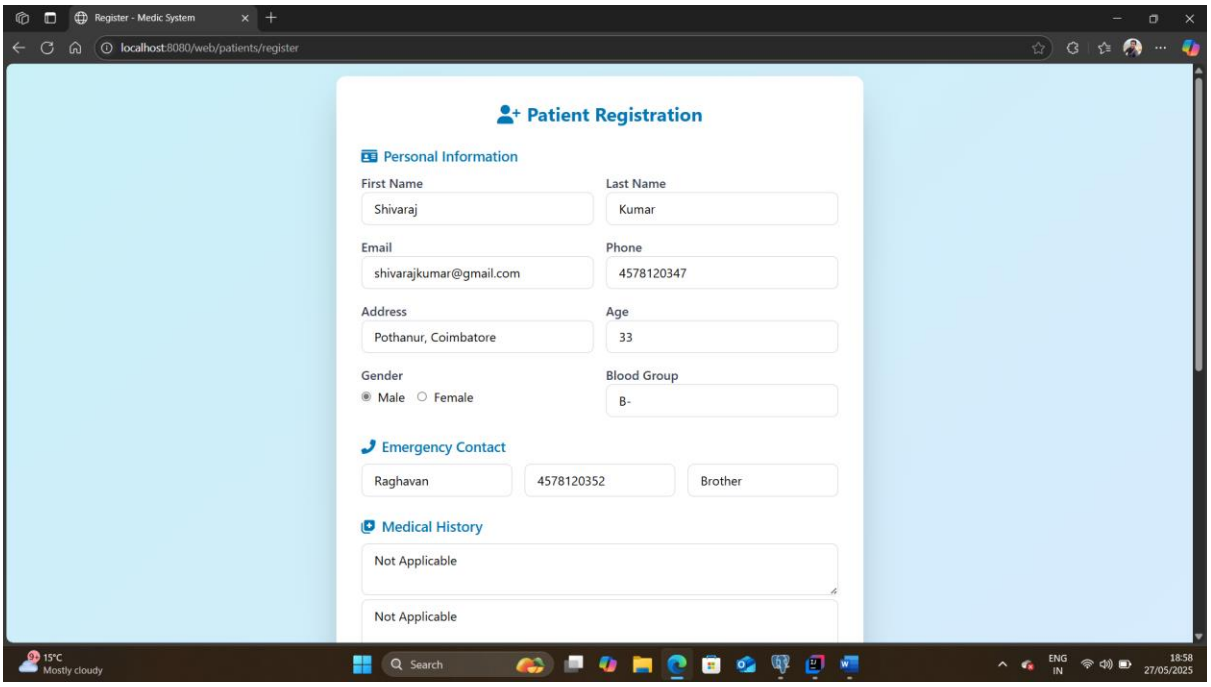
1. **Render login Page**
   - **Endpoint:** /web/patients/login
   - **Method:** GET
   - **Description:** Renders the login page.



Are you a new patient? Register here
Are you a new doctor? Register here

2. Render registration page

- **Endpoint:** /web/patients/register
- **Method:** GET
- **Description:** Renders a page with form to collect new patient's details.



Patient Registration

Personal Information

First Name: Shivaraj	Last Name: Kumar
Email: shivarajkumar@gmail.com	Phone: 4578120347
Address: Pothanur, Coimbatore	Age: 33
Gender: [✓ Male] Female	Blood Group: B-

Emergency Contact

Raghavan	4578120352	Brother

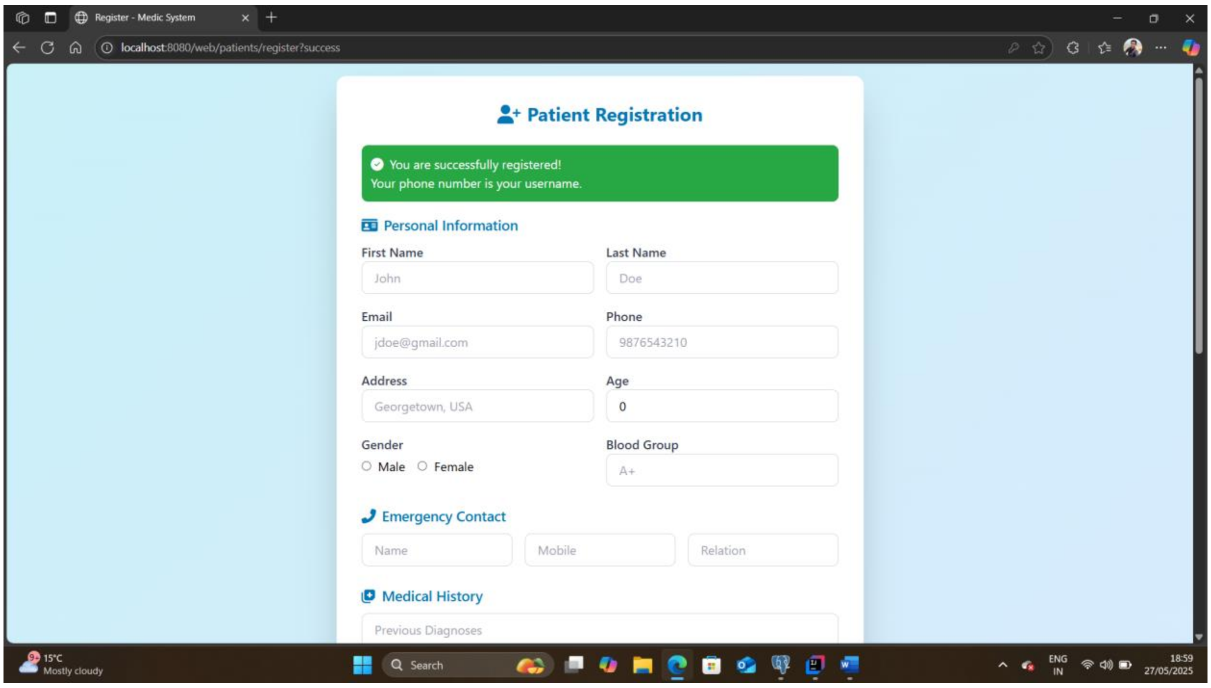
Medical History

Not Applicable

Not Applicable

3. Save the new patient to database

- **Endpoint:** /web/patients/savePatient
- **Method:** POST
- **Description:** Makes a post request to save the patient details into the database.



Patient Registration

You are successfully registered!
Your phone number is your username.

Personal Information

First Name: John	Last Name: Doe
Email: jdoe@gmail.com	Phone: 9876543210
Address: Georgetown, USA	Age: 0
Gender: Male Female	Blood Group: A+

Emergency Contact

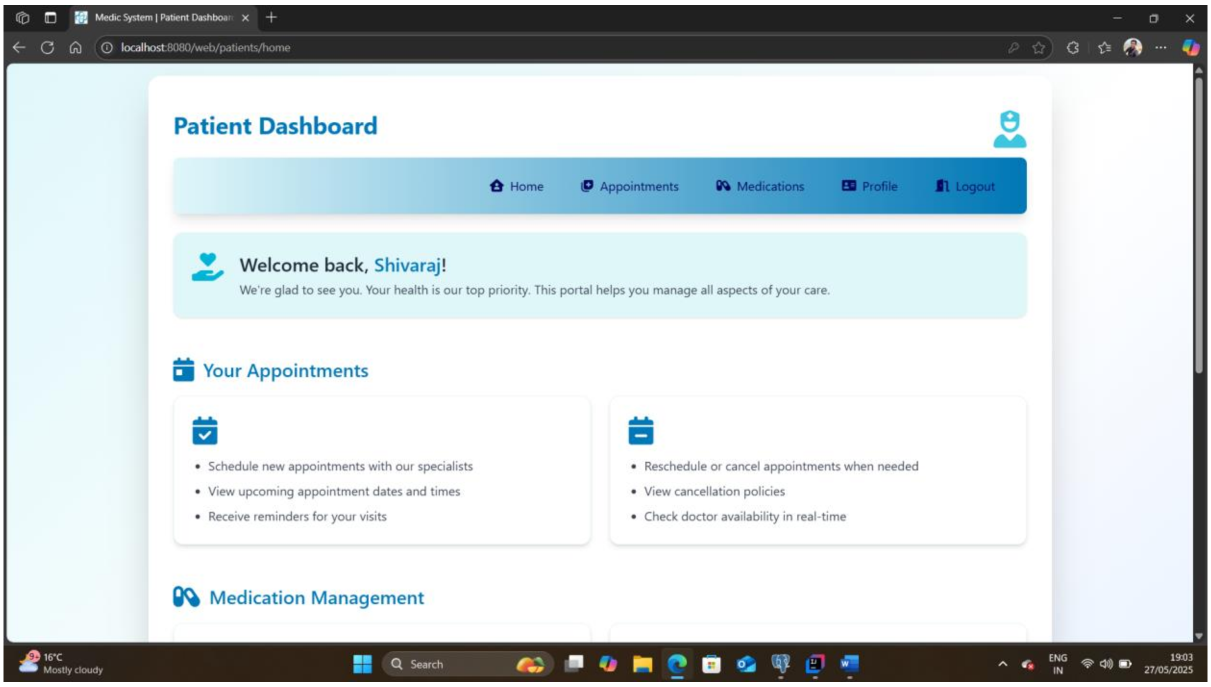
Name Mobile Relation

Medical History

Previous Diagnoses

PatientWebController.java

1. **Render home Page**
   - **Endpoint:** /web/patients/home
   - **Method:** GET
   - **Description:** Renders the home page.


Medic System | Patient Dashboard
localhost:8080/web/patients/home
Home
Appointments
Medications
Profile
Logout

Patient Dashboard

Welcome back, Shivaraj!

We're glad to see you. Your health is our top priority. This portal helps you manage all aspects of your care.

Your Appointments

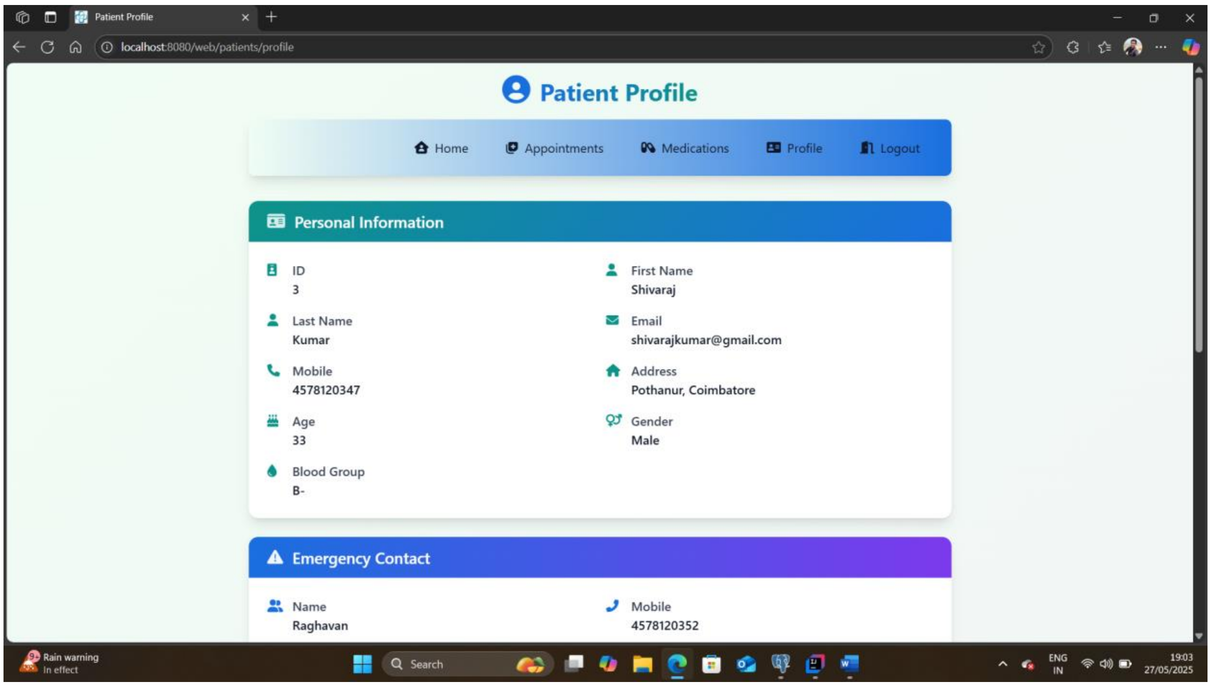
- Schedule new appointments with our specialists
- View upcoming appointment dates and times
- Receive reminders for your visits

- Reschedule or cancel appointments when needed
- View cancellation policies
- Check doctor availability in real-time

Medication Management

2. Render profile page

- **Endpoint:** /web/patients/profile
- **Method:** GET
- **Description:** Renders the patient's profile page.



Patient Profile

Personal Information

ID
3

First Name
Shivaraj

Last Name
Kumar

Email
shivarajkumar@gmail.com

Mobile
4578120347

Address
Pothanur, Coimbatore

Age
33

Gender
Male

Blood Group
B-

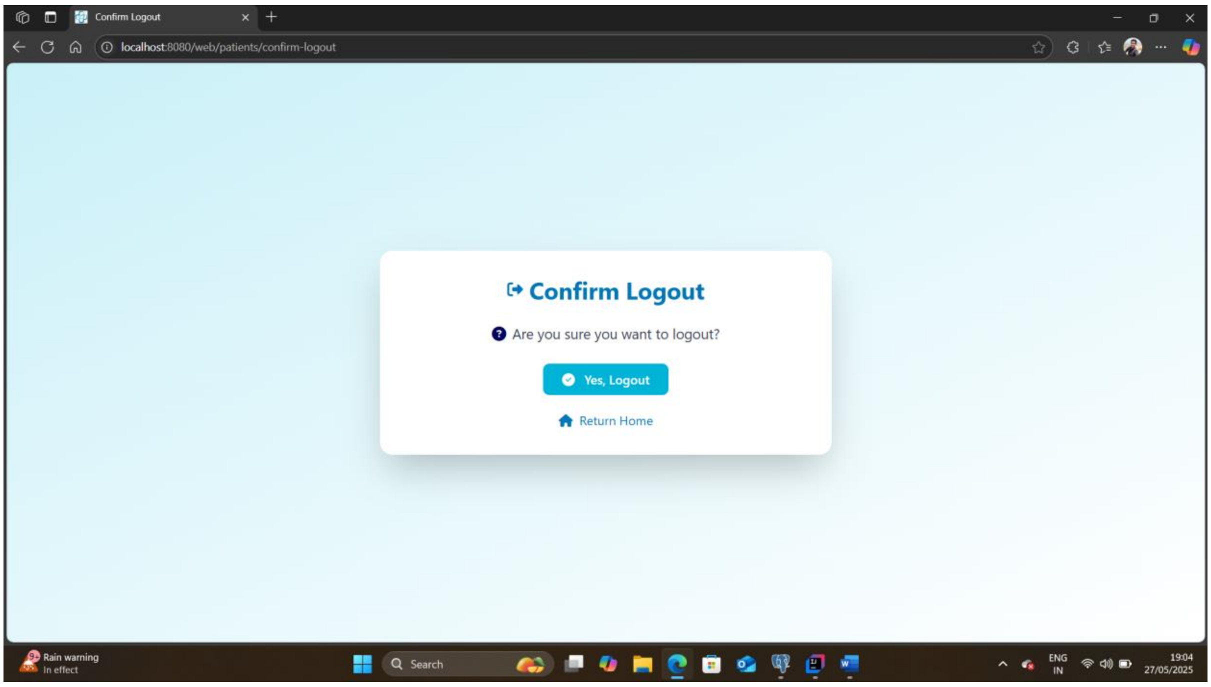
Emergency Contact

Name
Raghavan

Mobile
4578120352

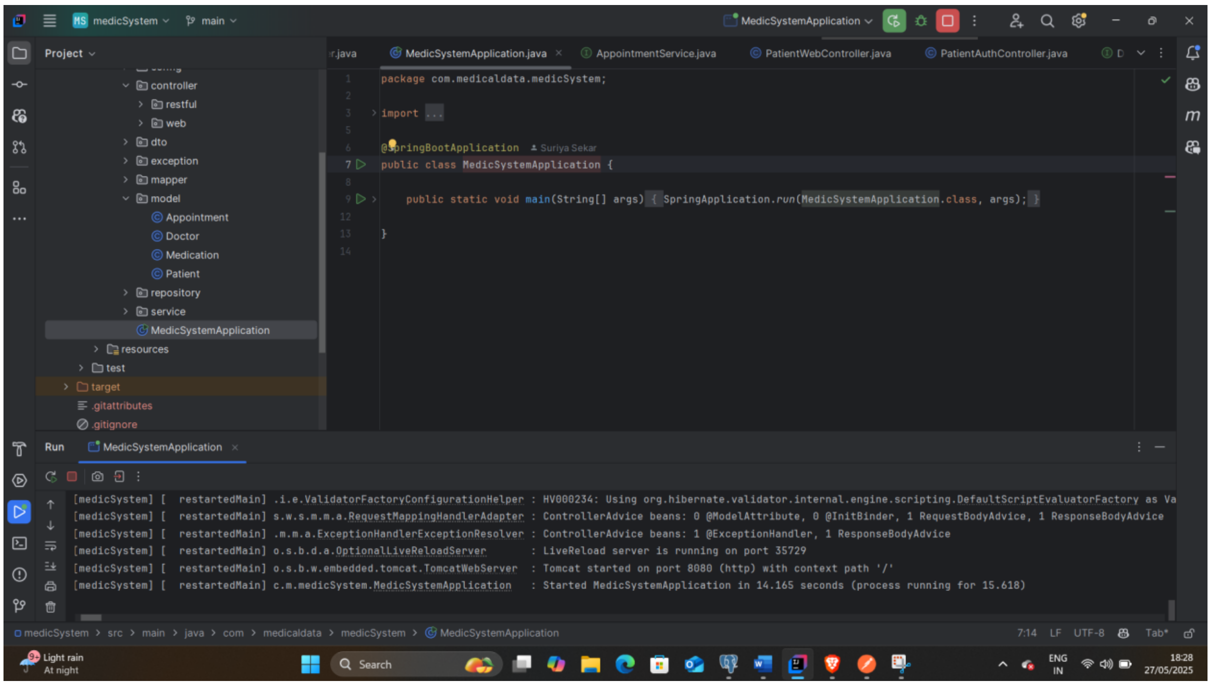
3. **Render logout confirmation page**

- **Endpoint:** /web/patients/confirm-logout
- **Method:** GET
- **Description:** For confirmation to logout.



Running the Application

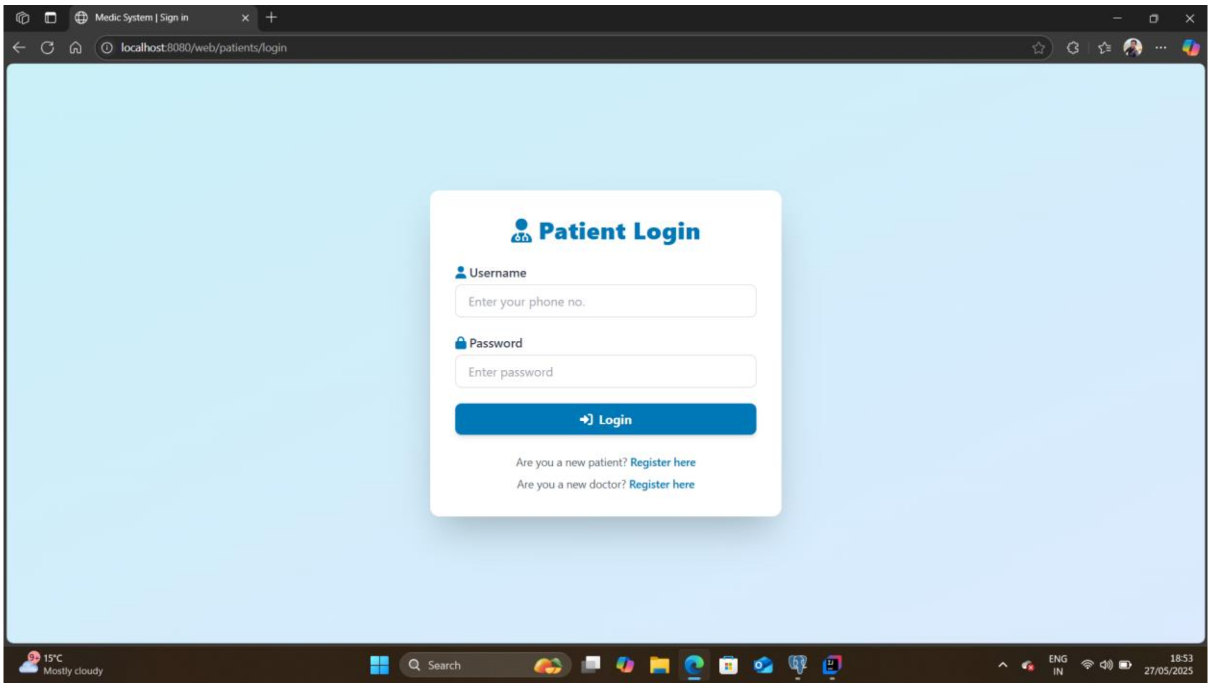
Initialization from Intellij IDEA



```
package com.medicaldata.medicSystem;

import ...

@SpringBootApplication
public class MedicSystemApplication {

    public static void main(String[] args) { SpringApplication.run(MedicSystemApplication.class, args); }

}
```

```
[medicSystem] [  restartedMain] .i.e.ValidatorFactoryConfigurationHelper : HV000234: Using org.hibernate.validator.internal.engine.scripting.DefaultScriptEvaluatorFactory as Va
[medicSystem] [  restartedMain] s.w.s.m.m.a.RequestMappingHandlerAdapter : ControllerAdvice beans: 0 @ModelAttribute, 0 @InitBinder, 1 RequestBodyAdvice, 1 ResponseBodyAdvice
[medicSystem] [  restartedMain] .m.m.a.ExceptionHandlerExceptionResolver : ControllerAdvice beans: 1 @ExceptionHandler, 1 ResponseBodyAdvice
[medicSystem] [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
[medicSystem] [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
[medicSystem] [  restartedMain] c.m.medicSystem.MedicSystemApplication   : Started MedicSystemApplication in 14.165 seconds (process running for 15.618)
```

Login Page



Patient Login

Username

Enter your phone no.

Password

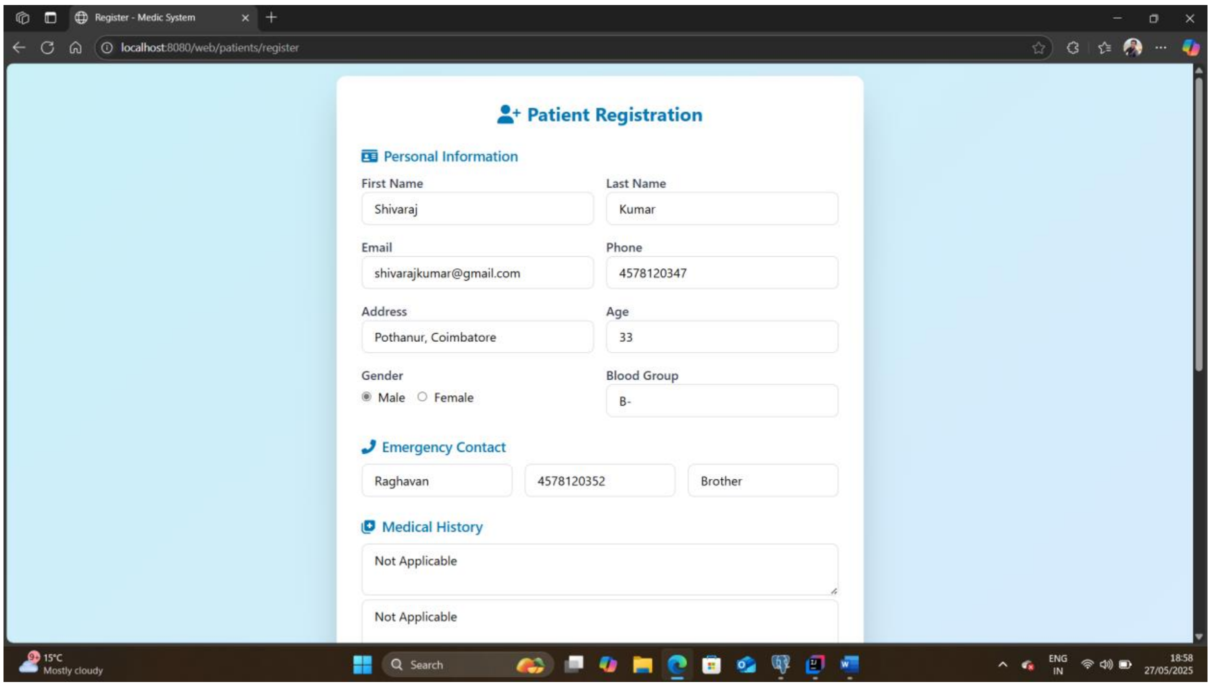
Enter password

Login

Are you a new patient? Register here

Are you a new doctor? Register here

Register Page


Register - Medic System

localhost:8080/web/patients/register

Patient Registration

Personal Information

First Name: Shivaraj	Last Name: Kumar
Email: shivarajkumar@gmail.com	Phone: 4578120347
Address: Pothanur, Coimbatore	Age: 33
Gender: [● Male] ○ Female	Blood Group: B-

Emergency Contact

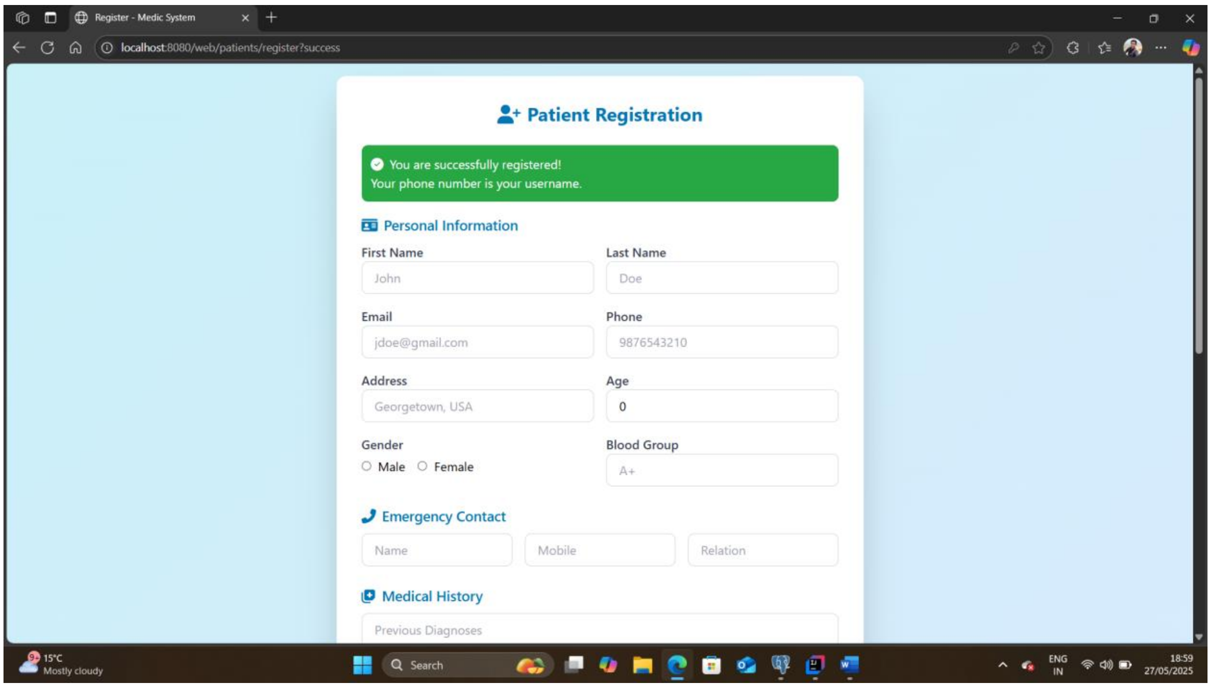
Raghavan | 4578120352 | Brother

Medical History

Not Applicable

Not Applicable

Successful Registration Page


Register - Medic System

localhost:8080/web/patients/register?success

Patient Registration

You are successfully registered!
Your phone number is your username.

Personal Information

First Name: John	Last Name: Doe
Email: jdoe@gmail.com	Phone: 9876543210
Address: Georgetown, USA	Age: 0
Gender: ○ Male ○ Female	Blood Group: A+

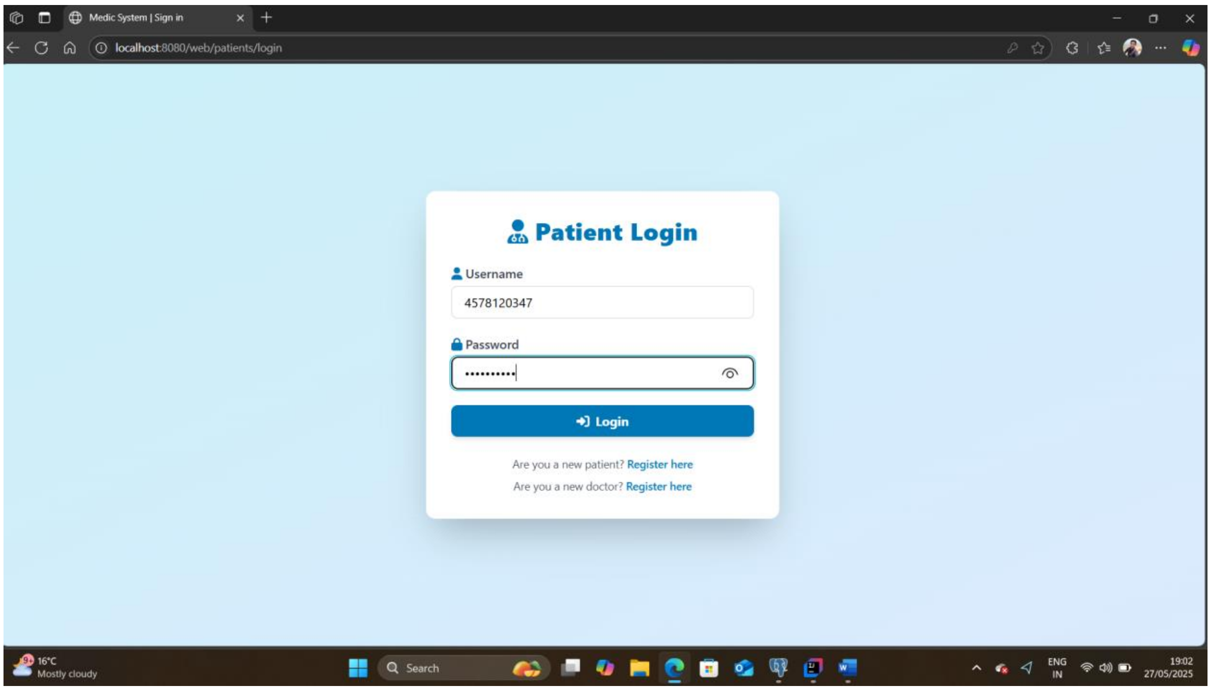
Emergency Contact

Name | Mobile | Relation

Medical History

Previous Diagnoses

After registration, login page



Patient Login

Username

4578120347

Password

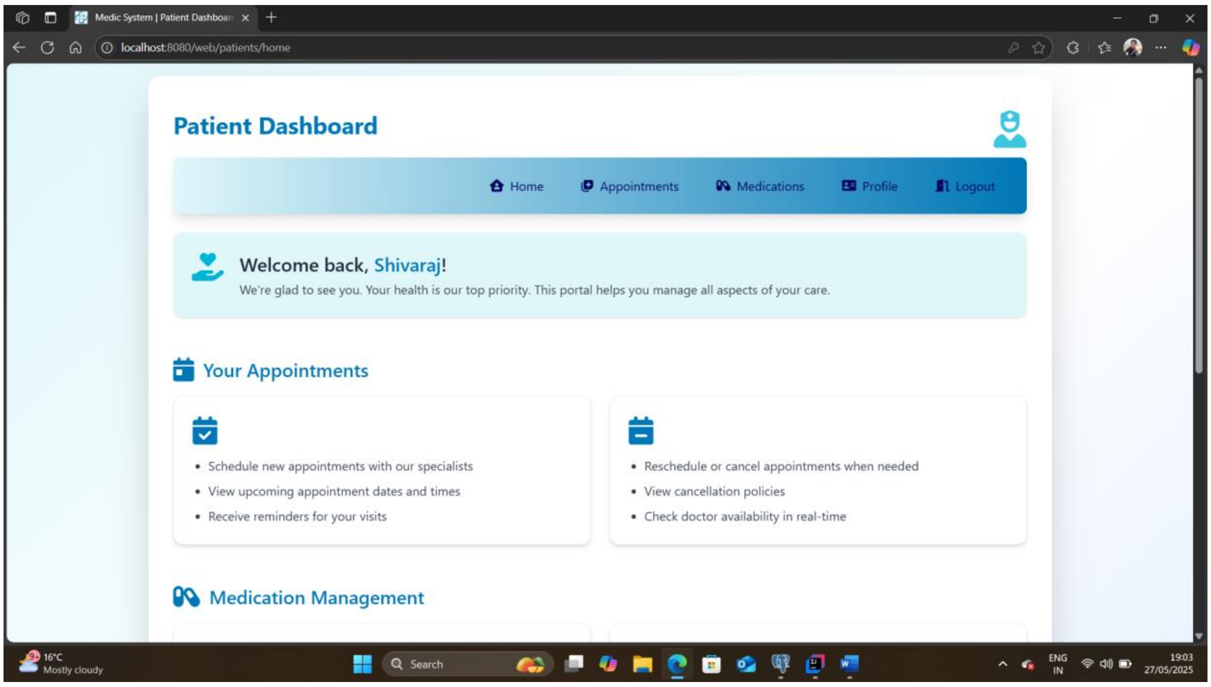
••••••••••

Login

Are you a new patient? Register here

Are you a new doctor? Register here

Home Page


Medic System | Patient Dashboard
localhost:8080/web/patients/home
Home
Appointments
Medications
Profile
Logout

Patient Dashboard

Welcome back, Shivaraj!

We're glad to see you. Your health is our top priority. This portal helps you manage all aspects of your care.

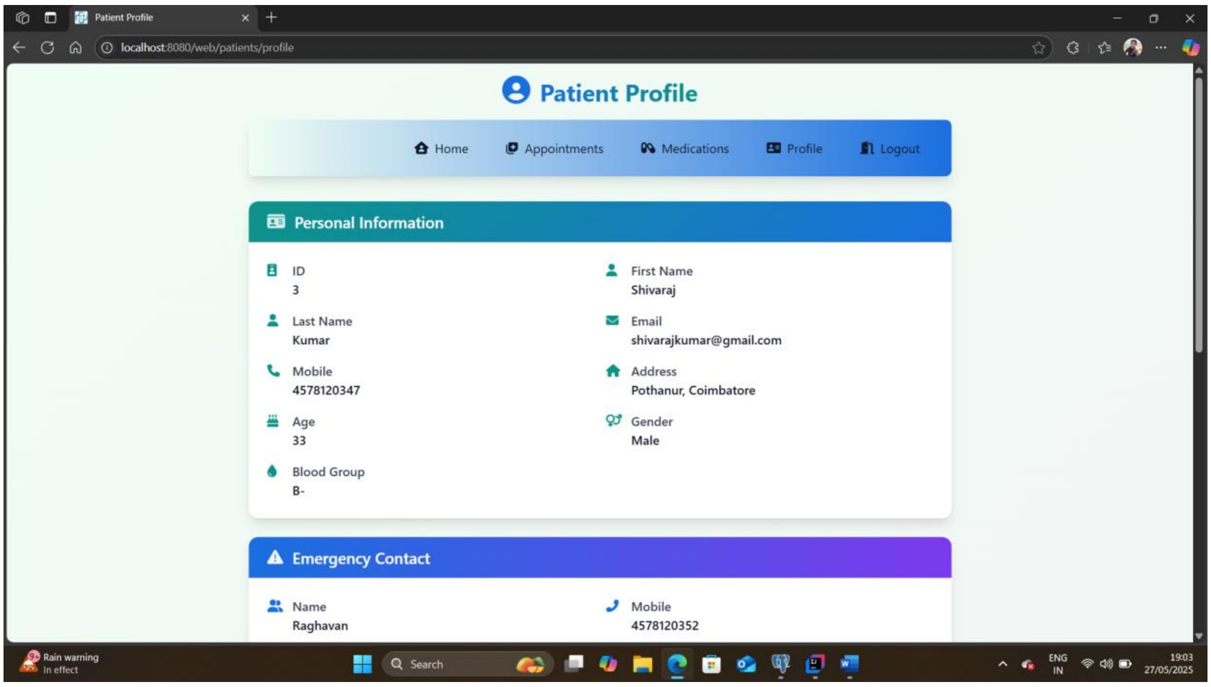
Your Appointments

- Schedule new appointments with our specialists
- View upcoming appointment dates and times
- Receive reminders for your visits

- Reschedule or cancel appointments when needed
- View cancellation policies
- Check doctor availability in real-time

Medication Management

Profile Page


Patient Profile

localhost:8080/web/patients/profile

Patient Profile

Home Appointments Medications Profile Logout

Personal Information

ID
3

First Name
Shivaraj

Last Name
Kumar

Email
shivarajkumar@gmail.com

Mobile
4578120347

Address
Pothanur, Coimbatore

Age
33

Gender
Male

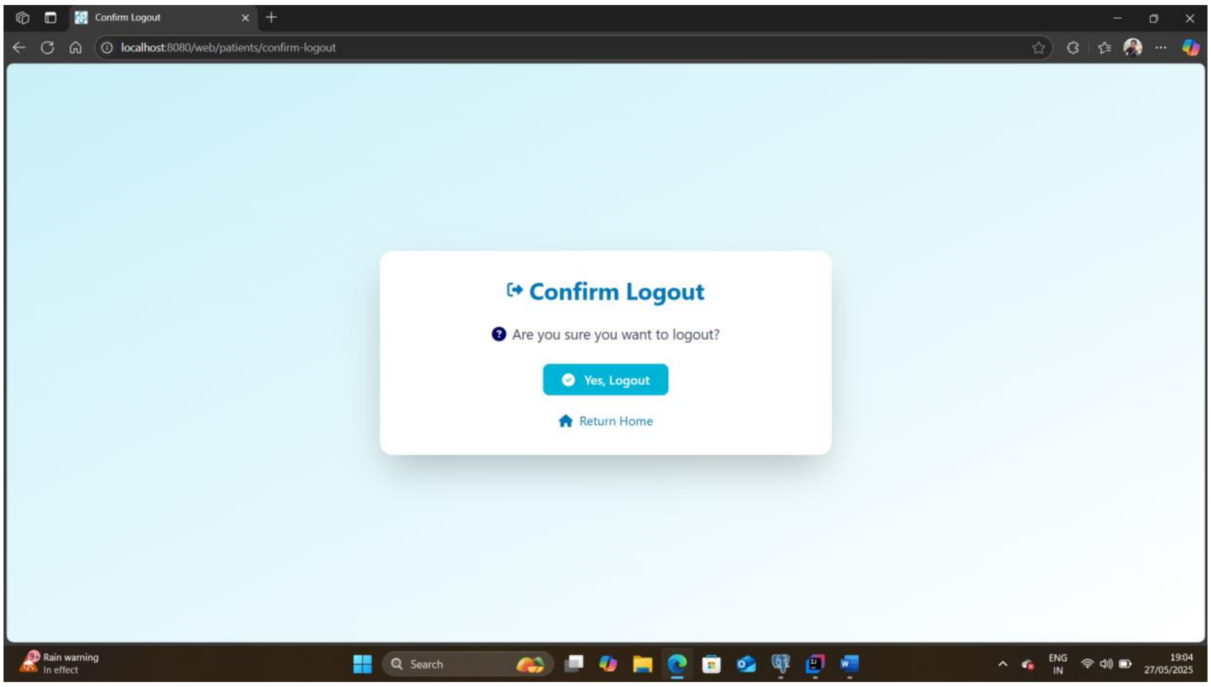
Blood Group
B-

Emergency Contact

Name
Raghavan

Mobile
4578120352

Confirm Logout Page


Confirm Logout

localhost:8080/web/patients/confirm-logout

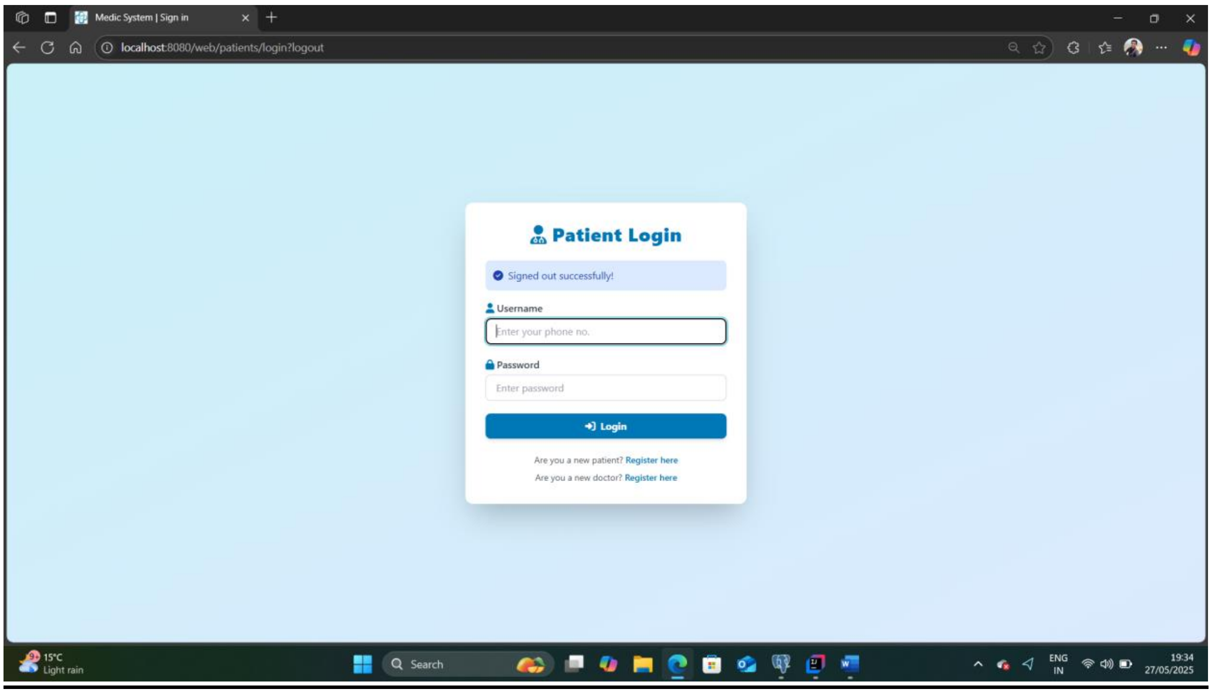
Confirm Logout

Are you sure you want to logout?

Yes, Logout

Return Home

Sign Out Success Page


Medic System | Sign in
localhost:8080/web/patients/login?logout

Signed out successfully!

Are you a new patient? Register here

Are you a new doctor? Register here

Conclusion

The Medic System Application represents a significant advancement in the management of healthcare processes, addressing the critical needs of patient care, medication tracking, and appointment scheduling. As the healthcare industry continues to evolve, the demand for efficient, reliable, and user-friendly management systems has become increasingly apparent. This application not only meets these demands but also sets a benchmark for future developments in healthcare technology.

Throughout this documentation, we have explored the various features and functionalities of the Medic System Application. The patient management system is at the heart of the application, allowing healthcare providers to maintain accurate and up-to-date patient records. This feature is essential for ensuring that practitioners have immediate access to vital information, which can significantly impact the quality of care delivered. By streamlining the process of managing patient data, the application reduces the likelihood of errors and enhances the overall efficiency of healthcare operations.

Medication management is another critical component of the Medic System Application. The ability to track prescribed medications, dosages, and administration schedules is vital for patient safety. This feature not only helps healthcare providers monitor adherence to treatment plans but also minimizes the risk of medication errors, which can have serious consequences for patient health. By providing a centralized platform for medication management, the application empowers healthcare professionals to make informed decisions and improve patient outcomes.

Appointment scheduling is a crucial aspect of healthcare delivery, and the Medic System Application excels in this area. The application simplifies the process of scheduling, rescheduling, and canceling appointments, thereby reducing wait times and enhancing the patient experience. Efficient appointment management is essential for optimizing healthcare facility operations, ensuring that resources are utilized effectively, and that patients receive timely care. By addressing these challenges, the application contributes to a more organized and responsive healthcare environment.

The technological foundation of the Medic System Application further enhances its capabilities. Built on robust frameworks such as Spring JPA and Thymeleaf, the application ensures efficient data management and dynamic content rendering. The use of PostgreSQL as the database management system guarantees secure and reliable storage of sensitive patient information. Additionally, the incorporation of Tailwind CSS provides a modern and responsive user interface, making the application accessible across various devices and improving user experience.

Security remains a paramount concern in healthcare applications, and the Medic System Application prioritizes the protection of sensitive data. By implementing user authentication and authorization mechanisms, the application safeguards against unauthorized access, ensuring that patient confidentiality is maintained. This focus on security not only complies with regulatory requirements but also fosters trust among patients and healthcare providers.

In conclusion, the Medic System Application is a comprehensive solution that addresses the multifaceted challenges faced by healthcare providers in managing patient care. By integrating essential features such as patient management, medication tracking, and appointment scheduling, the application enhances operational efficiency and improves the quality of care delivered to patients. As healthcare continues to advance, the Medic System Application stands as a testament to the potential of technology in transforming healthcare delivery. It not only meets the current needs of healthcare providers but also lays the groundwork for future innovations in the field. As we look ahead, the ongoing development and enhancement of such applications will be crucial in ensuring that healthcare systems can adapt to the everchanging landscape and continue to provide high-quality care to patients.