PATIENT MEDICINE AND APPOINTMENT SYSTEM

Contents

[Prerequisites: 2](#_Toc179141877)

[Getting the code into your local system: 3](#_Toc179141878)

[Opening the project in an IDE: 4](#_Toc179141879)

[Setting the database: 5](#_Toc179141880)

[Starting the application: 6](#_Toc179141881)

[**API DOCUMENTATION:** 9](#_Toc179141882)

[**ENDPOINTS:** 9](#_Toc179141883)

[Api Documentation For Webcontroller 9](#_Toc179141884)

[Api documentation for doctor-restcontroller: 14](#_Toc179141885)

[**DATA VALIDATION RULES** 17](#_Toc179141886)

[Patient Registration Validation Rules 17](#_Toc179141887)

[Doctor Registration Validation Rules 19](#_Toc179141888)

[Medication creation Validation rules: 20](#_Toc179141889)

[Appointment booking validation rules: 21](#_Toc179141890)

[Controller Validation Rules 21](#_Toc179141891)

[**SCHEMAS:** 22](#_Toc179141892)

[Patient table schema(Database): 22](#_Toc179141893)

[Patient Entity schema(Spring Boot): 22](#_Toc179141894)

[Doctor table schema(Database): 22](#_Toc179141895)

[Doctor table schema(Spring boot): 23](#_Toc179141896)

[Medication table schema(Database): 23](#_Toc179141897)

[Medication table schema(Spring boot): 23](#_Toc179141898)

[Appointment table schema(Database): 24](#_Toc179141899)

[Appointment table schema(Spring boot): 24](#_Toc179141900)

[**FOREIGN KEY REFERENCES** 24](#_Toc179141901)

[Dummy doctor data for use: 24](#_Toc179141902)

This documentation describes about a Patient Medicine and Appointment System(PMAS). This is a web-based healthcare application designed to facilitate patient registration, medical history management, and appointment scheduling. It allows patients to create profiles, providing key details such as personal information, medical history, and emergency contacts. Additionally, it includes user authentication for both patients and staff, ensuring secure access.

This application provides **Web interfaces(mainly)** for CRUD operations, ensuring data integrity through robust validation. The project is structured with clear separation of concerns, using DTOs, controllers, and services to handle data flow and business logic.

The following, focus on how to set-up, configure and run the same.

Moreover, the application utilizes the following key components:

1. JDK 17
2. Maven
3. Spring boot 3.4.4
4. MySQL
5. Git and Github
6. Thymeleaf template engine (for the Front-end files)
7. HTML
8. Bootstrap-CSS
9. Javascript

The complete code for the application is stored in the git hub repository: https://github.com/arunprakashxavier/PMAS

## Prerequisites:

1. An IDE (like, IntelliJ)
2. JDK 17
3. Maven (Build tool)
4. MySQL along with an IDE(like MySQL Workbench)
5. A browser (like Chrome)

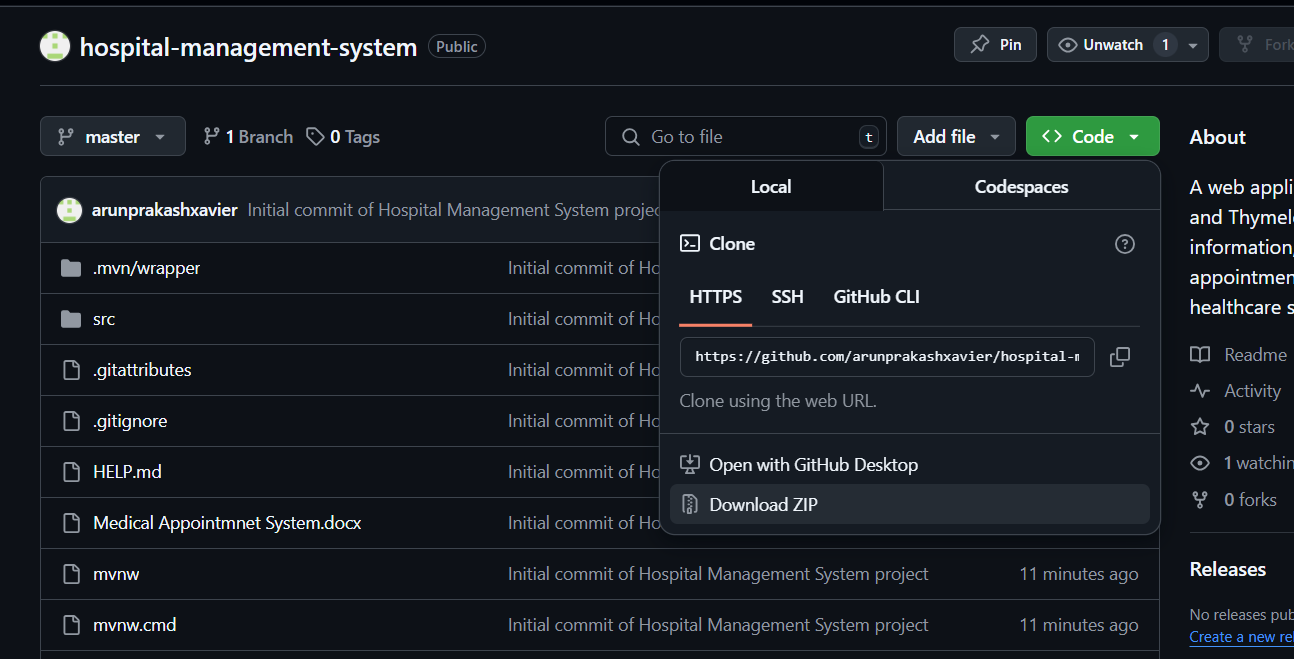
Now, let us proceed to the set-up and configuration of the project.

## 

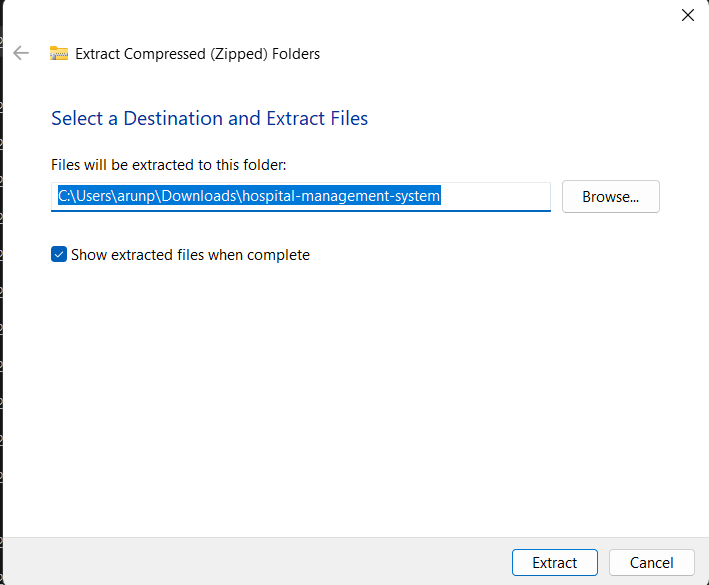
## Getting the code into your local system:

1. Get the code from the github repository <https://github.com/arunprakashxavier/hospital-management-system> to your system.

One of the methods you can do is download the file as a zip folder, shown below:



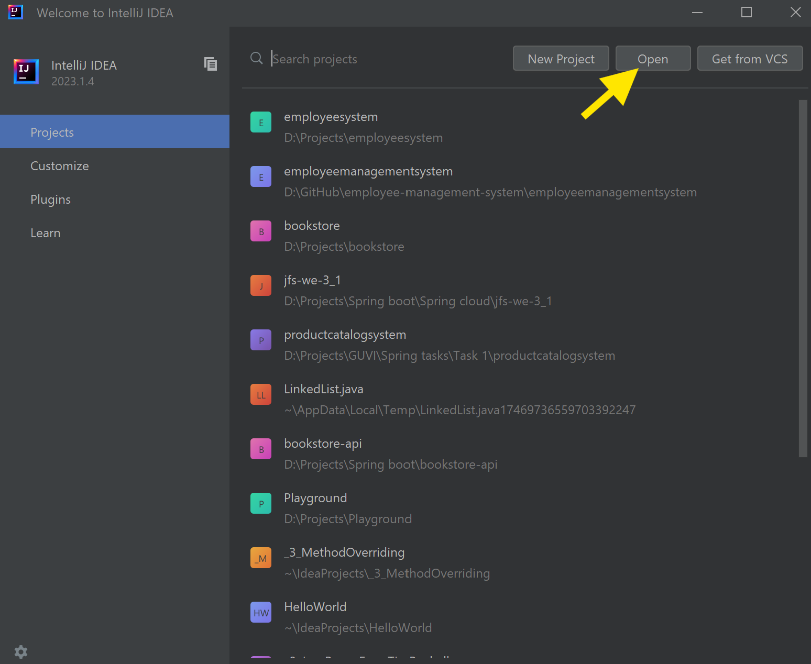
1. Then extract the downloaded zip folder into a folder as per your interest.



## Opening the project in an IDE:

In this manual an my project, I will be using IntelliJ as my preferred IDE. Whereas using Eclipse or any other IDE shouldn’t be a problem as the procedure is almost the same in any IDE.

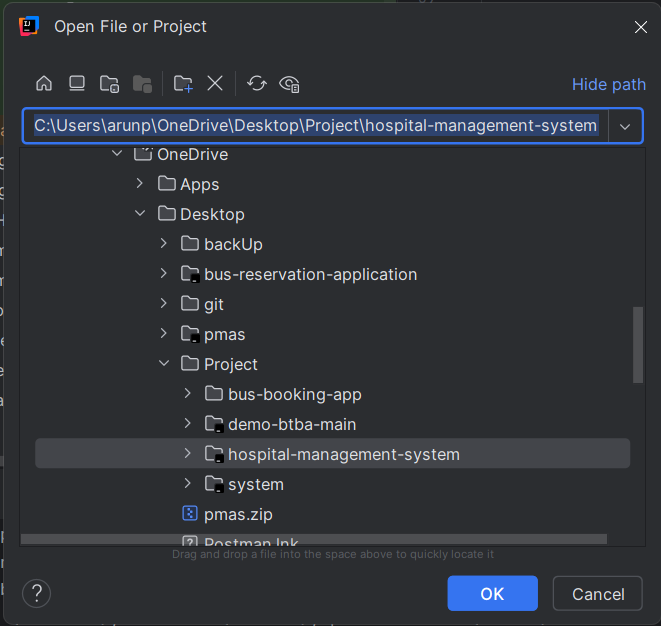
1. pen IntelliJ. Select *Open*.



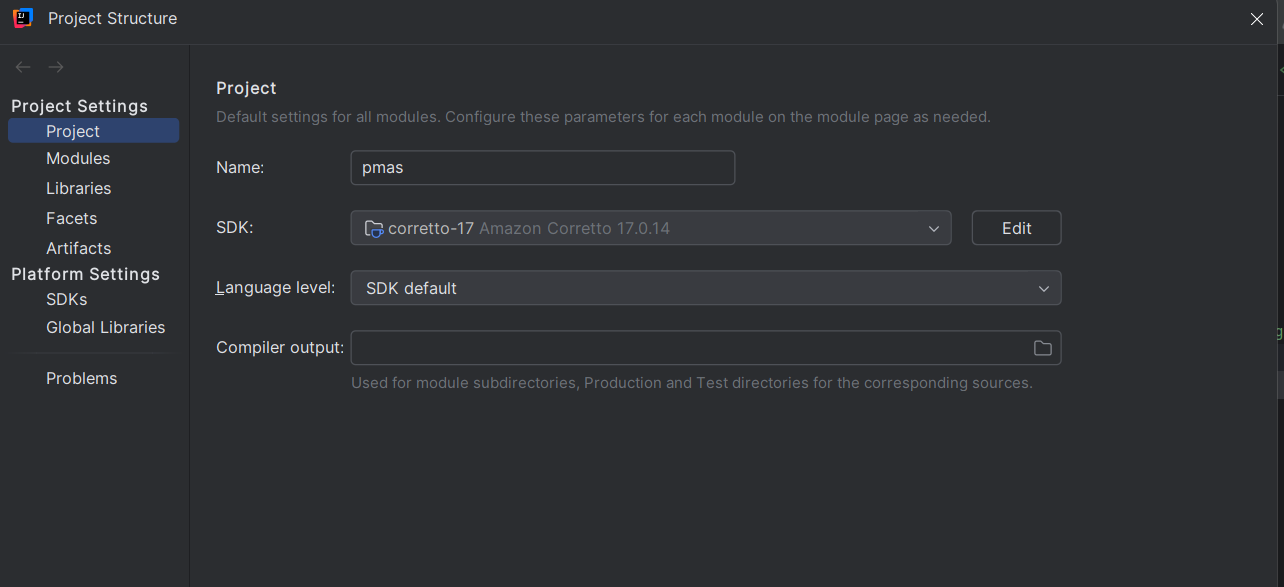
1. Browse to the path where we extracted our project and select *OK*.

Wait for a few seconds as the project will automatically load. During this time, the maven dependencies must also load on its own. You can also manually do this by clicking

*maven -> Reload all maven projects.*

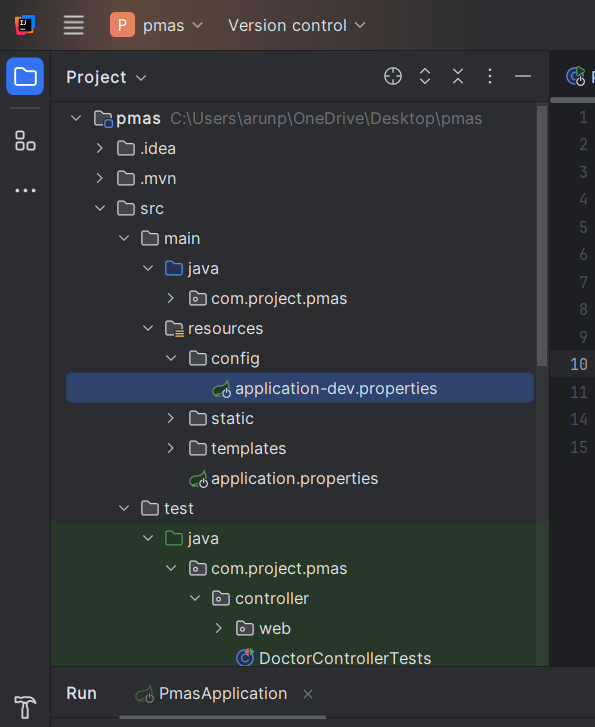


1. Go to *File -> Project Structure -> Project* and ensure that *JDK 17* is selected as the SDK



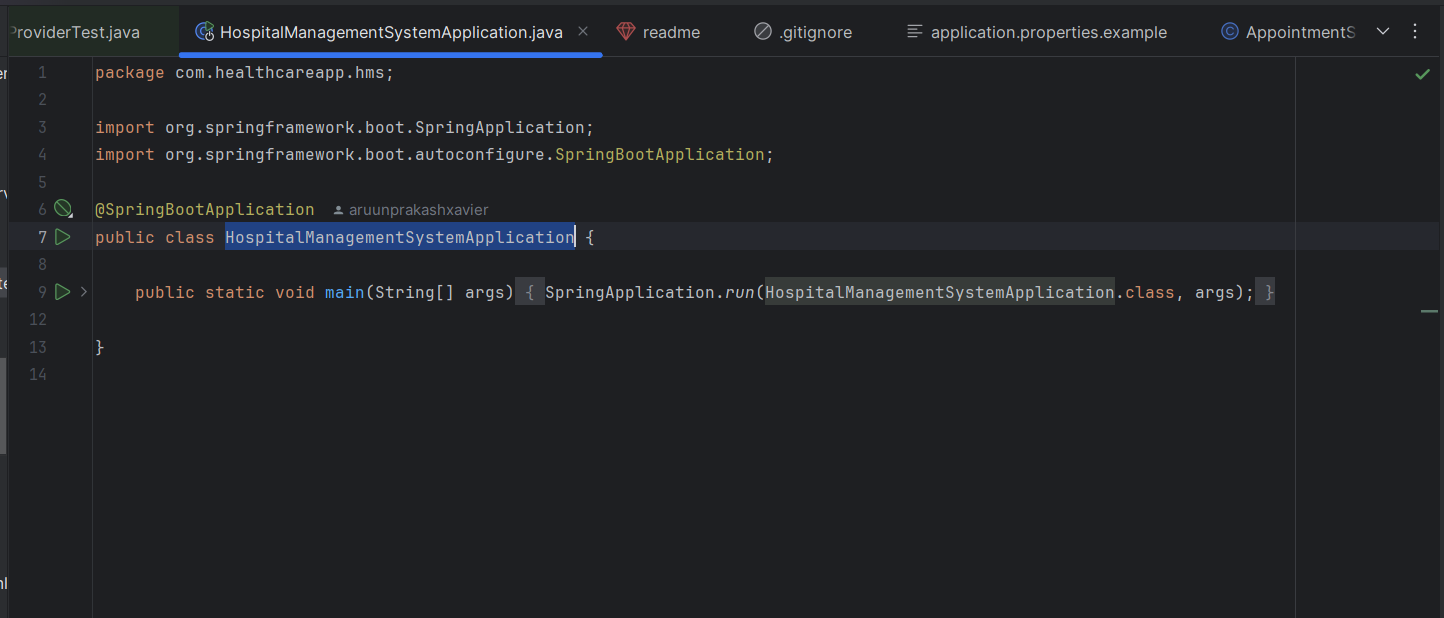
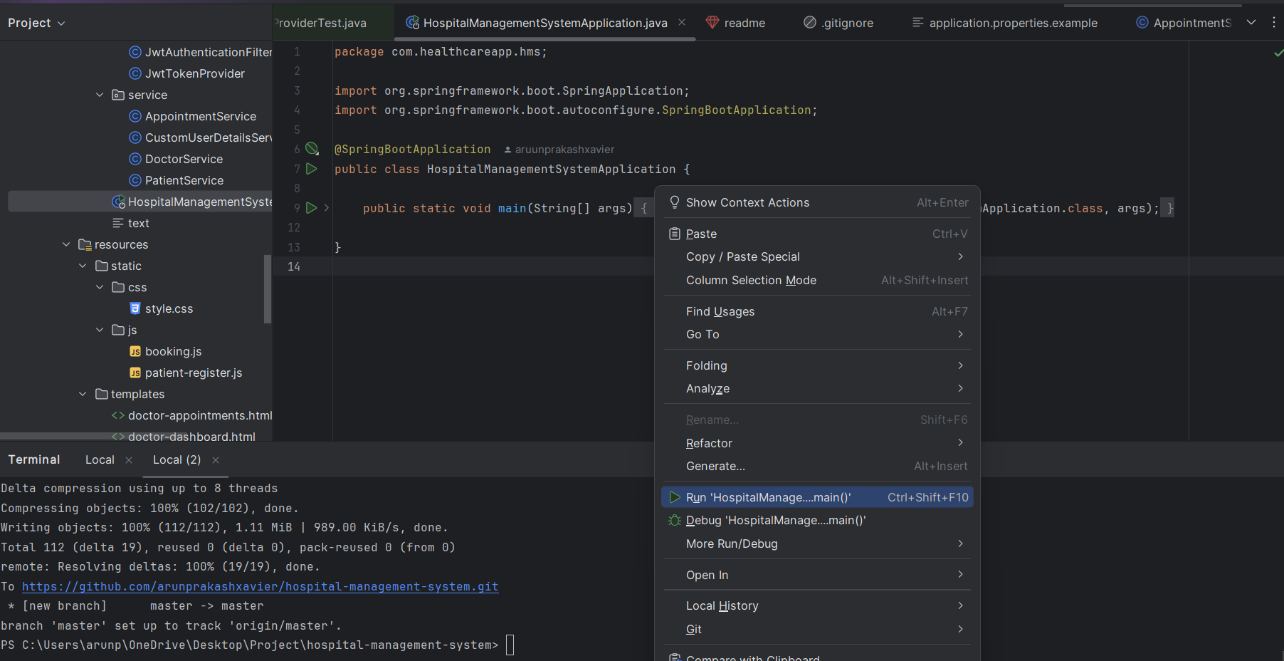
## Setting the database:

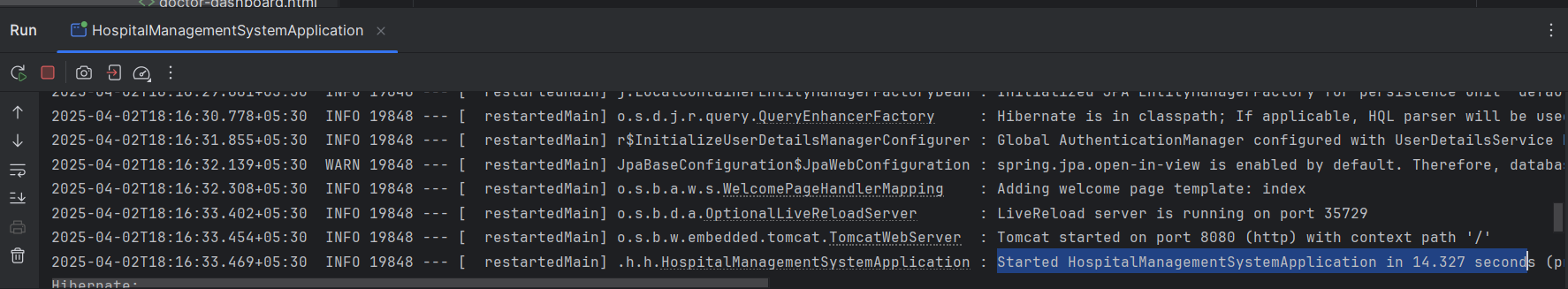
The application is created having MySQL as its database for storing its data. The main configurations of the application are written inside the application-dev.properties file inside

***src->main->resources->config->application-dev.properties***

* As shown, change the *url, username* and *password* properties as per your system configurations.
* Create a database in MySQL for this application.
* **Do ensure that the database name and the name present in the url (in this case: pmas\_db) are the exact same.** Any difference in the name will cause error while running the application.

## Starting the application:

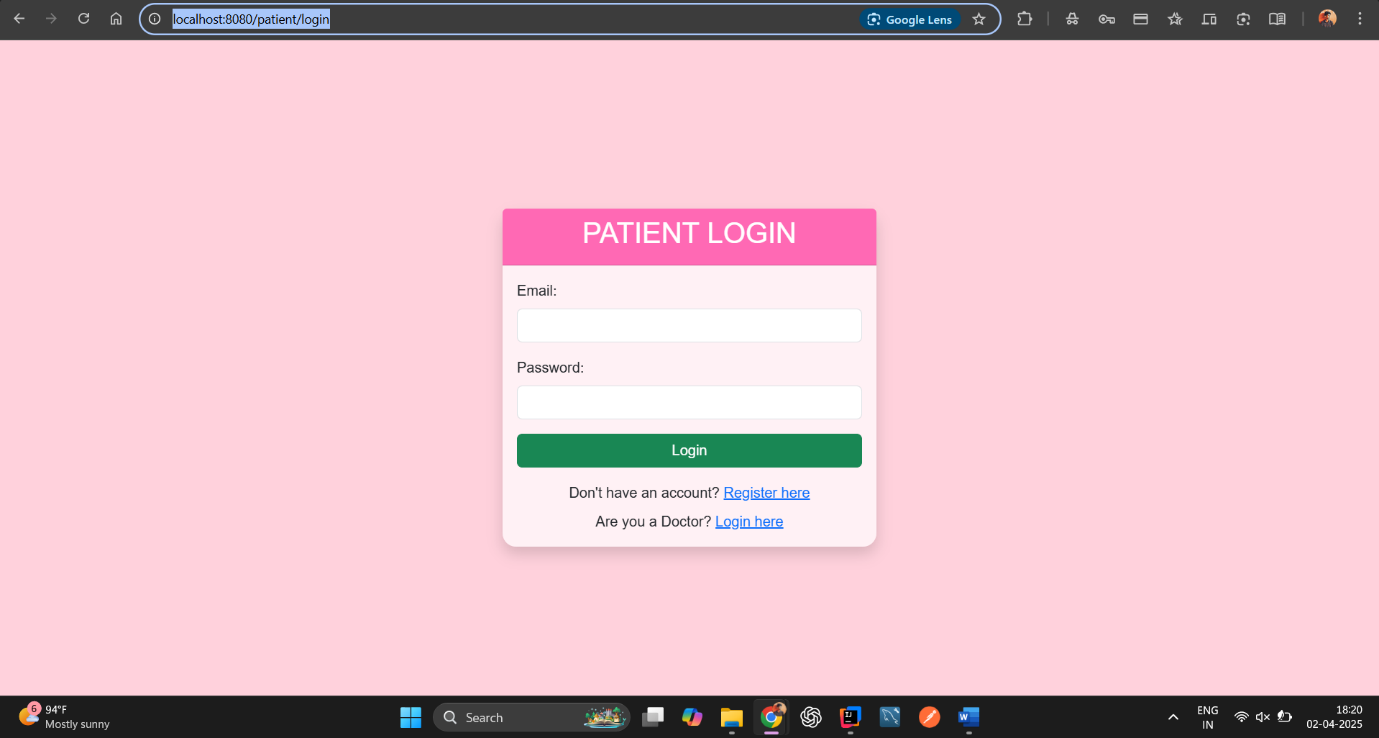
1. Before starting the application, ensure MySQL service is up and running.
2. Go to *hospital-management-system-> src -> main -> java -> com.healthcareapp.hms*
3. 
4. Right-click inside the *HospitalManagementSystemApplicatiom.java f*ile and click HospitalManagementSystemApplication.main()
5. **
6. On clicking Run, the application will start and you will see the below screen:



If the database name in the *application-dev.properties* and in the MySQL database match, Hibernate will automatically create a table as required and the system will get started.

1. In *application-dev.properties*, *8082* is the port number that is configured for this application. But ensure that the no other service is using that port and it is correctly mentioned in *server.port* in *application-dev.properties*.
2. Now, go to your browser and hit [http://localhost:<port-number>/patients/login](http://localhost:%3cport-number%3e/patients/login). For port 8080: <http://localhost:8080/patient/login> . This should render the login page that after logging in will lead to the homepage of the application.

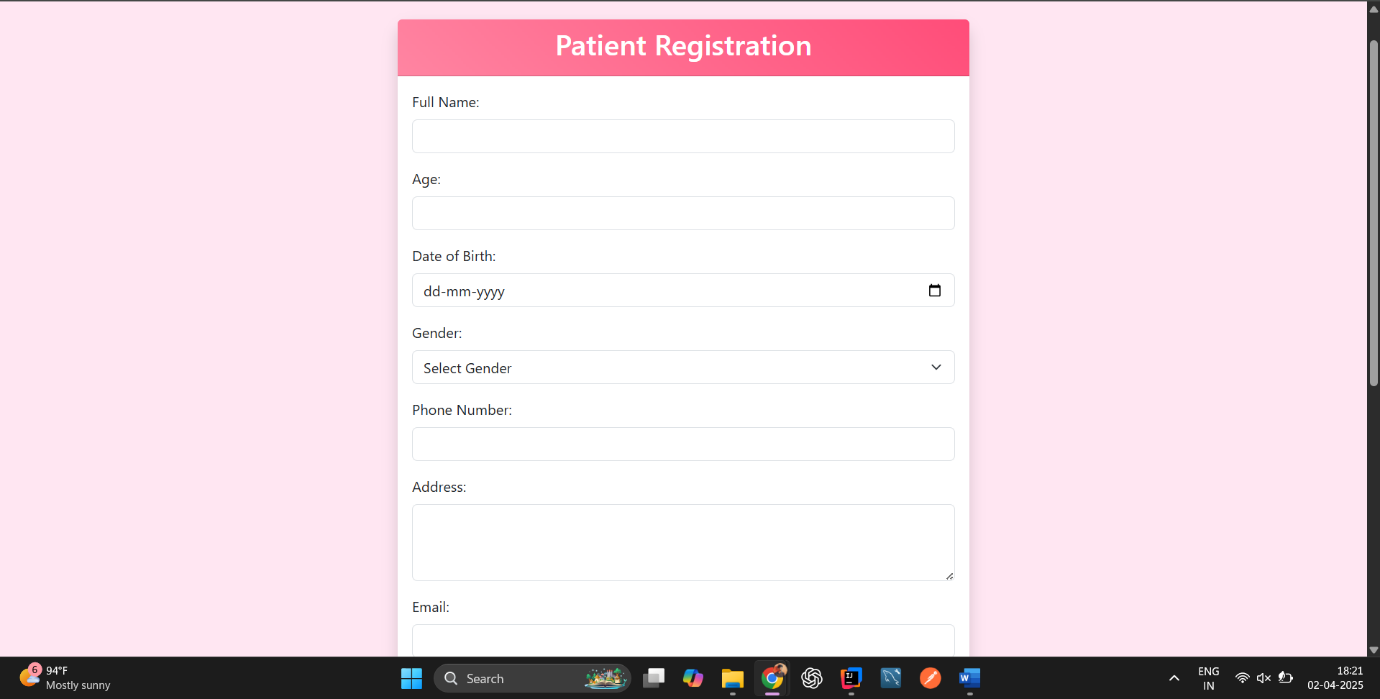
This is how the login page will look like:



Once logged in, you will be able to see the home page of the application:

From here, you can communicate with the application through the browser for Appointment booking and Medication management.

Also, at the login page there are links provided to allow new registration of a patient as well as a doctor.

Patient registration page:  


Similarly there is a registration page for Doctors as well.

**Do checkout the video present in the repository for a project walk-through.**

# **API DOCUMENTATION:**

## **ENDPOINTS:**

### Api Documentation For Webcontroller

**Overview:**

The WebControllers handle web-based requests for managing the patient appointments, medications and showing their profile as well.

Also, it is to be noted that the endpoints in **PATIENTAUTHCONTROLLER** and **DOCTORWEBCONTROLLER** classes are the only ones that can be accessed without any authentication.

#### PATIENTAUTHCONTROLLER.CLASS

**1. Render Patient Login Page**

* **Endpoint:** /patient/login
* **Method:** GET
* **Controller:** PageController.java
* **Description**: Renders the patient login page (patient-login.html). Passes optional error and logout parameters to the model for displaying messages.
* **Response:** HTML page with a form for accepting login credentials (email/password). Form submits via POST to /perform\_patient\_login (handled by Spring Security).

**2. Render Patient Registration Page**

* **Endpoint:** /patient/register
* **Method:** GET
* **Controller:** PageController.java
* **Description:** Renders the patient registration page (patient-register.html). Adds an empty PatientRegistrationDTO object to the model for form binding (though currently handled by JS).
* **Response:** HTML page displaying the form for collecting new patient details.

**3. Register New Patient (API)**

* **Endpoint:** /api/auth/patient/register
* **Method:** POST
* **Controller:** AuthController.java
* **Description:** Handles the API request to register a new patient. This endpoint is typically called asynchronously via JavaScript (patient-register.js) after the user submits the form onthe /patient/register page. It validates the input, checks for duplicates, encodes the password, and saves the patient details to the database via PatientService.
* **Parameter:** Expects JSON data in the request body matching the PatientRegistrationDTO class (@RequestBody). Input validation (@Valid) is performed.
* **Response:** Returns a JSON response. On success, typically returns 201 Created status with a success message and basic user info (AuthResponseDTO)

#### PATIENTWEBCONTROLLER.CLASS

1. **Render home Page**

* **Endpoint:** /web/patients/home
* **Method:** GET
* **Description:** Renders the home page.
* **Response:** HTML page with directing links to appointment, medication management and profile viewing sections.

1. **Render profile page**

* **Endpoint:** /web/patients/profile
* **Method:** GET
* **Description:** Renders the patient’s profile page.
* **Response:** HTML page displaying the patient details.

1. **Render logout confirmation page**

* **Endpoint:** /web/patients/confirm-logout
* **Method:** GET
* **Description:** For confirmation to logout.
* **Response:** An HTML page which asks for a confirmation if the user wants to logout or return home.

#### DOCTORWEBCONTROLLER.CLASS

1. **Render Patient Dashboard Page (Patient's "Home")**

* **Endpoint:** /patient/dashboard
* **Method:** GET
* **Controller:** PageController.java
* **Security:** Requires authenticated user with ROLE\_PATIENT (via @PreAuthorize).
* **Description:** Renders the main dashboard page (patient-dashboard.html) for a logged-in patient after successful login.
* **Response:** HTML page typically displaying a welcome message and containing links/cards directing to "My Appointments", "Book Appointment", "My Medications", and "My Profile" sections. *(Data for dashboard widgets like appointment counts is marked as TODO).*

1. **Render Patient Profile Page**

* **Endpoint:** /patient/profile
* **Method:** GET
* **Controller:** PageController.java
* **Security**: Requires authenticated user with ROLE\_PATIENT (via @PreAuthorize).
* Description: Fetches the full profile details (PatientProfileDTO) for the currently logged-in patient using PatientService and renders the patient's profile page (patient-profile.html).
* **Response:** HTML page displaying the patient's details (Name, Email, Address, Guardian Info, etc.). Includes buttons to trigger modals for editing the profile and changing the password.

1. **Logout Process (No Confirmation Page)**

* **Endpoint:** /logout
* **Method:** Typically POST (handled by Spring Security Filter)
* **Configuration:** SecurityConfig.java (.logout(...) section)
* **Description:** There isn't a dedicated *confirmation page*. Instead, a request to /logout (usually triggered by submitting a simple form via a "Logout" button/link) is intercepted by Spring Security. It invalidates the user's session, clears security context, potentially deletes cookies (JSESSIONID), and then redirects the user.
* **Response:** A redirect (HTTP 302) to the configured logoutSuccessUrl (currently / with ?logout=true appended, i.e., /?logout=true). The landing page (/) or login page (/patient/login) can optionally check for the logout=true parameter to display a "You have been logged out" message.

#### MEDICATIONWEBCONTROLLER.CLASS

1. **Render Patient's Medication List Page**
   * **Endpoint:** /patient/medications
   * **Method:** GET
   * **Controller**: PageController.java
   * **Security:** Requires authenticated user with ROLE\_PATIENT (via @PreAuthorize).
   * **Description:** Renders the HTML page (patient-medications.html) displaying the list of all medications prescribed to the currently logged-in patient across all their appointments. It fetches the data (List<MedicationDTO>) via AppointmentService.
   * **Response:** HTML page showing the list of the patient's prescribed medications (Name, Dosage, Frequency, Doctor, Date Prescribed, etc.).
2. **Viewing a Particular Medication (by ID)**
   * **Endpoint:** *N/A*
   * **Description:** This feature (viewing a single medication record by its specific ID via a dedicated web page or API) is not currently implemented in the project. Medications are viewed either as a full list for the patient or as a list associated with a specific appointment (via API).
3. **Assign Medication(s) to Appointment (API)**
   * **Endpoint:** /api/appointments/{appointmentId}/medications
   * **Method:** POST
   * **Controller:** AppointmentController.java
   * **Security**: Requires authenticated user with ROLE\_DOCTOR or ROLE\_ADMIN (checked via @PreAuthorize), plus service-level checks for ownership and appointment status (COMPLETED).
   * **Description:** API endpoint used to assign/prescribe one or more medications to a specific *completed* appointment. This is typically called via JavaScript from the Doctor's UI (the modal after clicking "Complete").
   * **Parameter:** Expects the appointmentId in the path and a JSON array of MedicationInputDTO objects in the request body (@RequestBody List<MedicationInputDTO>).
   * **Response:** Returns 201 Created with a JSON array of the created MedicationDTO objects on success, or an error response (e.g., 400, 403, 404) on failure.

#### APPOINTMENTWEBCONTROLLER.CLASS

### **1. Render Patient's Appointment List Page**

### **Endpoint:** /patient/appointments

### **Method:** GET

### **Controller:** PageController.java

### **Security**: Requires ROLE\_PATIENT.

### **Description**: Renders the HTML page displaying all appointments for the currently logged-in patient. Fetches data via AppointmentService.

### **Response:** HTML page (patient-appointments.html) showing past and upcoming appointments in a table, potentially with action buttons (like Cancel).

### ***(Note: A similar endpoint /doctor/appointments exists for doctors).***

### **2. Get Doctors by Specialization (API)**

### **Endpoint:** /api/doctors/specialization/{specializationName}

### **Method:** GET

### **Controller:** DoctorController.java

### **Security:** Requires any authenticated user (isAuthenticated()).

### **Description:** API endpoint called via JavaScript (from the booking page) to retrieve doctors matching a specific specialization.

### **Parameter:**

### specializationName: (Path Variable) The specialization string.

### **Response:** JSON response containing a list of doctors (List<DoctorDTO>) matching the criteria.

### **3. Render Appointment Booking Page**

### **Endpoint:** /patient/book-appointment

### **Method:** GET

### **Controller:** PageController.java

### **Security**: Requires ROLE\_PATIENT.

### **Description:** Renders the HTML page (patient-book-appointment.html) where a patient can select specialization, doctor, date, and time slot to book a new appointment. This page uses JavaScript to call APIs (#2 and #4 below).

### **Response:** HTML page with form elements for booking details.

### **4. Get Available Doctor Slots (API)**

### **Endpoint:** /api/doctors/{doctorId}/available-slots

### **Method:** GET

### **Controller**: DoctorController.java

### **Security:** Requires any authenticated user (isAuthenticated()).

### **Description:** API endpoint called via JavaScript (from the booking page) to get available 30-minute time slots for a specific doctor on a given date.

### **Parameters:**

### **doctorId:** (Path Variable) The ID of the selected doctor.

### **date:** (Request Parameter, format YYYY-MM-DD) The date for which to check slots.

### **Response:** JSON response containing a list of available slots (List<AvailableSlotDTO>), each with a start and end time.

### **5. Book an Appointment (API)**

### **Endpoint:** /api/appointments/book

### **Method:** POST

### **Controller:** AppointmentController.java

### **Security:** Requires ROLE\_PATIENT.

### **Description:** API endpoint called via JavaScript (from the booking page) to submit and finalize the appointment booking after the user has selected all details. Creates an appointment with 'PENDING' status.

### **Parameter:** Expects JSON data in the request body matching AppointmentBookingDTO (containing doctorId, requestedDateTime, reason). The patient ID is taken from the logged-in user context.

### **Response:** JSON response (AppointmentDTO) representing the newly created appointment with PENDING status on success (Status 201 Created), or a JSON error object on failure.

### **6. Cancel an Appointment (API)**

### **Endpoint:** /api/appointments/{appointmentId}/cancel

### **Method:** PUT

### **Controller:** AppointmentController.java

### **Security:** Requires ROLE\_PATIENT or ROLE\_DOCTOR or ROLE\_ADMIN (Service layer validates ownership).

### **Description:** API endpoint called via JavaScript (from buttons on appointment list pages) to cancel a PENDING or SCHEDULED appointment.

### **Parameter:**

### appointmentId: (Path Variable) The ID of the appointment to cancel.

### **Response:** JSON response (AppointmentDTO) representing the appointment with its status updated to CANCELLED on success, or a JSON error object on failure.

### Api documentation for doctor-restcontroller:

**Overview:**

The DoctorRestController provides ways to perform CRUD operations through RESTful endpoints. The intial purpose of this controller was to provide a way to populate the database with multiple doctors. And, it extended with complete REST functionalities.

1. **Get All Doctors**

* **Endpoint:** / api/admin/doctors/register
* **Method:** GET
* **Description:** A RESTful method to get all the doctors from the database
* **Request Parameters:** None
* **Response:**
  + **200 OK:** Returns a list of DoctorDTO objects.
  + **204 No Content:** Returns an empty list if no doctors are found.
* **Sample Response(Json):**

[

{

"id": 1,

"Name": "Selva",

"gender": "Male",

"mobile": "9999999909",

"email": "ashok@hspt.com",

"speciality": "General Medicine",

"experienceInYears": 3,

"qualifications": "MBBS, MS",

},

{

"id": 2,

"Name": “kira",

"gender": "Male",

"mobile": "9999999910",

"email": "raj@hspt.com",

"speciality": "ENT",

"experienceInYears": 3,

"qualifications": "MBBS, ENT",

}

]

1. **Add a New Doctor**

* **Endpoint:** /api/admin/doctors/register
* **Method:** POST
* **Description:** A RESTful method to add a new doctor into the database.
* **Request Body:**
  + **Content-Type:** json – SaveDoctorDto object
  + **Sample Body(Json):**

{

    "firstName": "Yuvan",

    "lastName": "P",

    "gender": "Male",

    "mobile": "9988776655",

    "email": "jsmith@gmail.com",

    "speciality": "Dermatology",

    "experienceInYears": 5,

    "qualifications": "MBBS",

    "languagesSpoken": "Tamil, English, Kannada",

    "officeAddress": "Chennai"

}

* **Response:**
  + **201 Created:** Returns the newly created DoctorDTO object.
  + **400 Bad Request:** If a user with provided mobile number already exists.
* **Sample Response(Json):**

{

    "id": 4,

    "Name": "Yuvana",

    "gender": "Male",

    "mobile": "9988776655",

    "email": "jsmith@gmail.com",

    "speciality": "Dermatology",

    "experienceInYears": 5,

    "qualifications": "MBBS",

}

1. **Get Doctor by ID**

* **Endpoint:** /doctors/get/{id}
* **Method:** GET
* **Description:** A RESTful method to get a doctor from the database through its id.
* **Path Parameters:**
  + id (Long): The ID of the doctor to retrieve.
* **Response:**
  + **200 OK:** Returns the DoctorDTO object of the requested doctor.
  + **404 Not Found:** If the doctor with the specified ID does not exist.
* **Sample Response(json):**

{

    "id": 3,

    "Name": "Fzsil",

    "gender": "Female",

    "mobile": "6666666692",

    "email": "farah@hspt.com",

    "speciality": "Pathology",

    "experienceInYears": 4,

    "qualifications": "MBBS, DNB Pathology"

}

1. **Update a doctor**

* **Endpoint:** /doctors/update/{id}
* **Method:** PUT
* **Description:** A RESTful method to update a doctor in the database
* **Path Parameters:**
  + **Id:** Id of the doctor to be updated
* **Request Body:**
  + **Content-Type:** json – SaveDoctorDto object
  + **Sample Body(Json):**

{

    “Name": "Yuvan",

    "gender": "Male",

    "mobile": "9988776655",

    "email": "jsmith@gmail.com",

    "speciality": "Dermatology",

    "experienceInYears": 5,

    "qualifications": "MBBS",

}

* **Response:**
  + **200 OK:** Returns the updated EmployeeDTO object.
* **Sample Response(json):**

{

    "id": 4,

    "Name": "Yuvan",

    "gender": "Male",

    "mobile": "9988776655",

    "email": "jsmith@gmail.com",

    "speciality": "Dermatology",

    "experienceInYears": 5,

    "qualifications": "MBBS",

}

1. **Delete a doctor**

* **Endpoint:** /doctors/delete/{id}
* **Method:** DELETE
* **Description:** A RESTful method to delete a doctor from the database
* **Path Parameters:**
  + id (Long): The ID of the doctor to delete.
* **Response:**
  + **200 OK:** Returns a message indicating whether the deletion was successful.
* **Sample Response(json):**
  + "Doctor with ID - 1 removed from the database."

# **DATA VALIDATION RULES**

In the application, data validation is implemented to ensure that the data provided by users conforms to the expected format and constraints. Here are the key validation rules:

## Patient Registration Validation Rules

For registering new patients, the **register.html** form and the **SavePatientDto class** are used. The validation rules used for these ensure that the necessary fields are provided and are in the correct format.

Let us look at the fields and the validation rules:

* **First Name**
  + **Required:** Yes
  + **Validation:** Must not be blank. It can have letters and spaces only.
  + **Constraints:**
    - @NotBlank(message = "First name of a patient cannot be blank.")
    - @Pattern(regexp = "^[A-Za-z ]\*$", message = "Name can only contain letters and spaces")
* **Last Name**
  + **Required:** Yes
  + **Validation:** Must not be blank. It can have letters and spaces only.
  + **Constraints:**
    - @NotBlank(message = "Last name of a patient cannot be blank.")
    - @Pattern(regexp = "^[A-Za-z ]\*$", message = "Name can only contain letters and spaces")
* **Email**
  + **Required:** Yes
  + **Validation:** Must be a well-formed email address.
  + **Constraints:**
    - @Email(message = "Enter a valid email address.")
    - @NotBlank(message = "Email cannot be blank.")
  + **Format Example:** example@domain.com
* **Mobile Number**
  + **Required:** Yes
  + **Validation:** Must be numeric and exactly 10 digits long.
  + **Constraints:**
    - @Pattern(regexp = "^\\d{10}$", message = "Mobile number must be numeric and exactly 10 digits long.")
* **Password**
  + **Required**: Yes
  + **Validation:** Must contain at least 8 characters, one uppercase letter, one lowercase letter, one number, and one special character (if provided).
  + **Constraints:**
    - @Pattern(regexp = "^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d)(?=.\*[!@#$%^&\*(),.?\":{}|<>])[A-Za-z\\d!@#$%^&\*(),.?\":{}|<>]{8,}$", message = "Password must contain at least 8 characters, at least one capital letter, at least one small, at least one number, and at least one special character.")
* **Address**
  + **Required:** Yes
  + **Validation:** Must not be null or empty.
  + **Constraints:**
    - @NotBlank(message = "Address of a patient cannot be blank.")
* **Age**
  + **Required:** Yes(From Frontend)
  + **Validation:** No specific validation rules provided. Will get only numbers from the webpage.
* **Gender**
  + **Required:** Yes(From Frontend)
  + **Validation:** No specific validation rules provided. Will return a string based on the radio type input selected from the webpage.
* **Emergency Contact Name**
  + **Required:** Yes
  + **Validation:** Must not be null or empty. Can have only letters and spaces.
  + **Constraints:**
    - @NotBlank(message = "Name of the emergency contact cannot be blank.")
    - @Pattern(regexp = "^[A-Za-z ]\*$", message = "Name of the emergency contact can only contain letters and spaces")
* **Emergency Contact Mobile**
  + **Required:** Yes
  + **Validation:** Must be numeric and exactly 10 digits long.
  + **Constraints:**
    - @Pattern(regexp = "^\\d{10}$", message = "Emergency contact Mobile number must be numeric and exactly 10 digits long.")
* **Emergency Contact Relation**
  + **Required:** Yes
  + **Validation:** Must not be null or empty. Can have only letters and spaces.
  + **Constraints:**
    - @NotBlank(message = "Emergency contact relationship cannot be blank.")
    - @Pattern(regexp = "^[A-Za-z ]\*$", message = "Relationship of the emergency contact can only contain letters and spaces".

## Doctor Registration Validation Rules

For registering new doctors, the **doctor-register.html** form and the **SaveDoctorDto class** are used. The validation rules used for these ensure that the necessary fields are provided and are in the correct format.

Let us look at the fields and the validation rules:

* **Name**
  + **Required:** Yes
  + **Validation**: Must not be null or empty. Can have only letters and spaces
  + **Constraints:**
    - @NotBlank(message = "First name of a doctor cannot be blank.")
    - @Pattern(regexp = "^[A-Za-z ]\*$", message = "Name can only contain letters and spaces")
* **Gender**
  + **Required:** Yes
  + **Validation:** Must not be null or empty. Will return a string based on the radio type input selected from the webpage.
  + **Constraints:**
    - @NotBlank(message = "Gender cannot be blank.")
* **Mobile Number**
  + **Required:** Yes
  + **Validation:** Must be numeric and exactly 10 digits long.
  + **Constraints:**
    - @NotBlank(message = "Mobile number is a mandatory field")
    - @Pattern(regexp = "^\\d{10}$", message = "Mobile number should be 10 digits long.")
* **Email**
  + **Required:** Yes(From front-end)
  + **Validation:** Must be a well-formed email address.
* **Speciality**
  + **Required:** Yes
  + **Validation:** Must not be null or empty. Can have only letters and spaces
  + **Constraints:**
    - @NotBlank(message = "Doctor speciality cannot be blank.")
    - @Pattern(regexp = "^[A-Za-z ]\*$", message = "Name can only contain letters and spaces")
* **Experience in Years**
  + **Required:** Yes(From front-end)
  + **Validation:** No specific validation rules provided in the backend. Will receive numbers only from the frontend.
* **Qualifications**
  + **Required:** Yes(From front-end)
  + **Validation:** No specific validation rules provided in the backend.

## Medication creation Validation rules:

For creating new medications, the **add-medication.html** and the **SaveMedicationDto** class provide necessary validations:

* **Patient ID**
  + **Required**: Yes
  + **Validation**: Must be a positive numeric value.
  + **Constraints**:
    - @Pattern(regexp = "^\\d+$", message = "Enter a valid patient id. Expecting a positive number.")
* **Medicine**
  + **Required**: Yes
  + **Validation**: Must not be null or empty.
  + **Constraints**:
    - @NotBlank(message = "Medicine should not be blank.")
* **Dosage**
  + **Required**: Yes(From Frontend)
  + **Validation**: No specific validation rules provided in the backend.
* **Frequency**
  + **Required**: Yes
  + **Validation**: Must not be null or empty.
  + **Constraints**:
    - @NotBlank(message = "In frequency field, enter how much intake is required per day.")
* **Status**
  + **Required**: Yes
  + **Validation**: Must not be null or empty.
  + **Constraints**:
    - @NotBlank(message = "Medicine current status is mandatory.")
* **Start Date**
  + **Required**: Yes(From frontend)
  + **Validation**: Must be in the format <yyyy-MM-dd>.
  + **Constraints**:
    - @Pattern(regexp = "^\\d{4}-\\d{2}-\\d{2}$", message = "Enter medication start date in <yyyy-MM-dd> format.")
* **End Date**
  + **Required**: Yes(From frontend)
  + **Validation**: Must be in the format <yyyy-MM-dd>.
  + **Constraints**:
    - @Pattern(regexp = "^\\d{4}-\\d{2}-\\d{2}$", message = "Enter medication end date in <yyyy-MM-dd> format.")
* **Notes**
  + **Required**: Yes(From frontend)

## Appointment booking validation rules:

For booking an appointment, the **add-appointment.html** file and the **SaveAppointmentDto** class provide the required validation rules that are listed below:

* **Patient ID**
  + **Required**: Yes
  + **Validation**: Must not be a null value.
  + **Constraints**:
    - @NotNull(message = "Patient id cannot be null")
* **Doctor ID**
  + **Required**: Yes
  + **Validation**: Must not be a null value.
  + **Constraints**:
    - @NotNull(message = "Doctor id cannot be null")
* **appointmentDateTime**
  + **Required**: Yes
  + **Validation**: Must be in the format “<yyyy-MM-dd>T<HH:mm:ss>”
  + **Constraints**:
    - @Pattern(regexp = "^\\d{4}-\\d{2}-\\d{2}T\\d{2}:\\d{2}:\\d{2}$", message = "Enter value in <yyyy-MM-dd>T<HH:mm:ss> format")

## Controller Validation Rules

The **@Controller** and **@RestController** validated Controller classes use validation annotations to enforce the below rules on the incoming requests:

* **Request Body Validation:**
  + The @Valid annotation is used in the controller methods to enforce validation on the DTO objects being passed in the request body. If validation fails, the request will not proceed, and an appropriate error message will be returned.
* **Path Variables:**
  + id (used in methods like getDoctorById, updateDoctor, deleteAppointment, etc.) must be a valid Long and is validated through standard Spring mechanisms to ensure it matches the required type.

# **SCHEMAS:**

### Patient table schema(Database):

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** | **Remarks** |
| id | bigint | PRIMARY KEY, NOT NULL, AUTO\_INCREMENT |
| address | varchar(255) | DEFAULT NULL |
| email | varchar(255) | DEFAULT NULL |
| name | varchar(255) | DEFAULT NULL |
| mobile | varchar(255) | DEFAULT NULL |
| password | varchar(255) | DEFAULT NULL |
| age | int | NOT NULL |
| allergies | varchar(255) | DEFAULT NULL |
| emergency\_contact\_mobile | varchar(255) | DEFAULT NULL |
| emergency\_contact\_name | varchar(255) | DEFAULT NULL |
| emergency\_contact\_relation | varchar(255) | DEFAULT NULL |
| gender | varchar(255) | DEFAULT NULL |

### Patient Entity schema(Spring Boot):

|  |  |
| --- | --- |
| **Column Name** | **Datatype** |
| id | Long |
| address | String |
| email | String |
| name | String |
| mobile | String |
| password | String |
| age | int |
| emergency\_contact\_mobile | String |
| emergency\_contact\_name | String |
| emergency\_contact\_relation | String |
| gender | String |

### Doctor table schema(Database):

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** | **Remarks** |
| id | bigint | PRIMARY KEY, NOT NULL, AUTO\_INCREMENT |
| speciality | varchar(255) | DEFAULT NULL |
| name | varchar(255) | DEFAULT NULL |
|  |  |  |
| email | varchar(255) | DEFAULT NULL |
| experience\_in\_years | int | NOT NULL |
| gender | varchar(255) | DEFAULT NULL |
| qualifications | varchar(255) | DEFAULT NULL |

### Doctor table schema(Spring boot):

|  |  |
| --- | --- |
| **Column Name** | **Datatype** |
| id | Long |
| speciality | String |
| name | String |
| email | String |
| experience\_in\_years | int |
| gender | String |
| mobile | String |
| qualifications | String |
|  |  |

### Medication table schema(Database):

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** | **Remarks** |
| id | bigint | PRIMARY KEY, NOT NULL, AUTO\_INCREMENT |
| end\_date | date | DEFAULT NULL |
| frequency | varchar(255) | DEFAULT NULL |
| medicine | varchar(255) | DEFAULT NULL |
| notes | varchar(255) | DEFAULT NULL |
| patient\_id | bigint | NOT NULL, Foreign Key(References `id` from patient table.) |
| dosage | varchar(255) | DEFAULT NULL |
| prescription\_date | date | DEFAULT NULL |
| updated\_date | date | DEFAULT NULL |

### Medication table schema(Spring boot):

|  |  |
| --- | --- |
| **Column Name** | **Datatype** |
| id | Long |
| frequency | String |
| medicine | String |
| notes | String |
| start\_date | LocalDate |
| patient\_id | LocalDate |
| dosage | String |
| prescription\_date | LocalDate |
| status | String |
| updated\_date | LocalDate |

### Appointment table schema(Database):

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** | **Remarks** |
| id | bigint | PRIMARY KEY, NOT NULL, AUTO\_INCREMENT |
| appointment\_date\_time | datetime | NOT NULL |
| created\_at | datetime | NOT NULL, DEFAULT CURRENT\_TIMESTAMP |
| doctor\_id | bigint | NOT NULL, Foreign Key(References `id` from doctor table.) |
| patient\_id | bigint | NOT NULL, Foreign Key(References `id` from patient table.) |

### Appointment table schema(Spring boot):

|  |  |
| --- | --- |
| **Column Name** | **Datatype** |
| id | Long |
| appointment\_date\_time | LocalDateTime |
| created\_at | LocalDateTime |
| doctor\_id | Long |
| patient\_id | Long |

Dummy doctor data for use: (Will be required for appointment booking)

[

{

  "name": "Dr. Ben Carter",

  "age": 42,

  "qualification": "MBBS, MS (Ortho)",

  "specialization": "Orthopedics",

  "phoneNumber": "9876543211",

  "yearsOfExperience": 15,

  "email": "ben.carter@hospital.org",

  "password": "passwordB456"

}

{

  "name": "Dr. Chloe Davis",

  "age": 35,

  "qualification": "MD, Pediatrics",

  "specialization": "Pediatrics",

  "phoneNumber": "8765432102",

  "yearsOfExperience": 8,

  "email": "dr.chloe.d@clinic.net",

  "password": "passwordC789"

}

{

  "name": "Dr. David Evans",

  "age": 58,

  "qualification": "MBBS, DM (Neurology)",

  "specialization": "Neurology",

  "phoneNumber": "7654321093",

  "yearsOfExperience": 28,

  "email": "david@dr.com",

  "password": "123456789"

}

{

  "name": "Dr. Emily Foster",

  "age": 39,

  "qualification": "MD, Dermatology",

  "specialization": "Dermatology",

  "phoneNumber": "6543210984",

  "yearsOfExperience": 12,

  "email": "efoster.derm@clinic.org",

  "password": "passwordE212"

}

{

  "name": "Dr. Emily Foster",

  "age": 39,

  "qualification": "MD, Dermatology",

  "specialization": "Dermatology",

  "phoneNumber": "6543210984",

  "yearsOfExperience": 12,

  "email": "efoster.derm@clinic.org",

  "password": "passwordE212"

}

{

  "name": "Dr. Fatima Iqbal",

  "age": 45,

  "qualification": "MD, General Medicine",

  "specialization": "General Physician",

  "phoneNumber": "4321098766",

  "yearsOfExperience": 18,

  "email": "dr.iqbal.gp@medcenter.net",

  "password": "passwordG434"

}

{

  "name": "Dr. Hannah Jones",

  "age": 40,

  "qualification": "MBBS, DNB (ENT)",

  "specialization": "ENT",

  "phoneNumber": "9123456780",

  "yearsOfExperience": 12,

  "email": "hannah.jones.ent@clinic.in",

  "password": "passwordH545"

}

{

  "name": "Dr. Ian Kumar",

  "age": 55,

  "qualification": "MD (Pulmonology)",

  "specialization": "Pulmonology",

  "phoneNumber": "9234567891",

  "yearsOfExperience": 25,

  "email": "ian.kumar@respiratorycare.org",

  "password": "passwordI656"

}

{

  "name": "Dr. Jasmine Lee",

  "age": 36,

  "qualification": "MBBS, MS (Ophthalmology)",

  "specialization": "Ophthalmology",

  "phoneNumber": "9345678902",

  "yearsOfExperience": 9,

  "email": "dr.jlee.eyes@clinic.com",

  "password": "passwordJ767"

}

{

  "name": "Dr. Kevin Miller",

  "age": 61,

  "qualification": "MD, DM (Oncology)",

  "specialization": "Oncology",

  "phoneNumber": "9456789013",

  "yearsOfExperience": 30,

  "email": "kevin.miller.onco@hospital.org",

  "password": "passwordK878"

}

{

  "name": "Dr. Linda Nelson",

  "age": 47,

  "qualification": "MBBS, MD (Psychiatry)",

  "specialization": "Psychiatry",

  "phoneNumber": "9567890124",

  "yearsOfExperience": 19,

  "email": "l.nelson.psy@mentalhealth.in",

  "password": "passwordL989"

}

{

  "name": "Dr. Mohan Patel",

  "age": 50,

  "qualification": "MD, Diabetology",

  "specialization": "Diabetology",

  "phoneNumber": "9678901235",

  "yearsOfExperience": 22,

  "email": "mohan.patel@diabetesclinic.com",

  "password": "123456789"

}

]

**admin login**

http://localhost:8080/api/auth/admin/login

{

"email": "admin@healthcare.com",

"password": "adminpassword"

}  
  
  
  
  
  
  
  
  
  
Thank you Guvi team for all of your support  
 regards,

ARUN PRAKASH M