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: 12](#_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 22](#_Toc179141891)

[**SCHEMAS:** 23](#_Toc179141892)

[Patient table schema(Database): 23](#_Toc179141893)

[Patient Entity schema(Spring Boot): 23](#_Toc179141894)

[Doctor table schema(Database): 24](#_Toc179141895)

[Doctor table schema(Spring boot): 24](#_Toc179141896)

[Medication table schema(Database): 24](#_Toc179141897)

[Medication table schema(Spring boot): 25](#_Toc179141898)

[Appointment table schema(Database): 25](#_Toc179141899)

[Appointment table schema(Spring boot): 25](#_Toc179141900)

[**FOREIGN KEY REFERENCES** 26](#_Toc179141901)

[Dummy doctor data for use: 27](#_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 3.3.2
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:

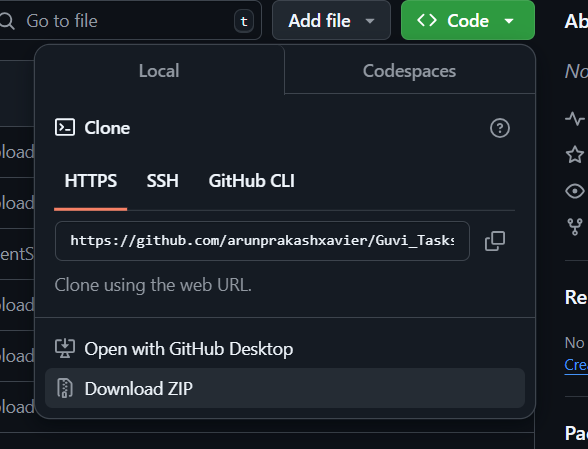
1. An IDE (like, IntelliJ)
2. JDK 17
3. Maven (Build tool)
4. MySQL along with an IDE(like MySQL Workbench)
5. A browser (like Chrome)

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

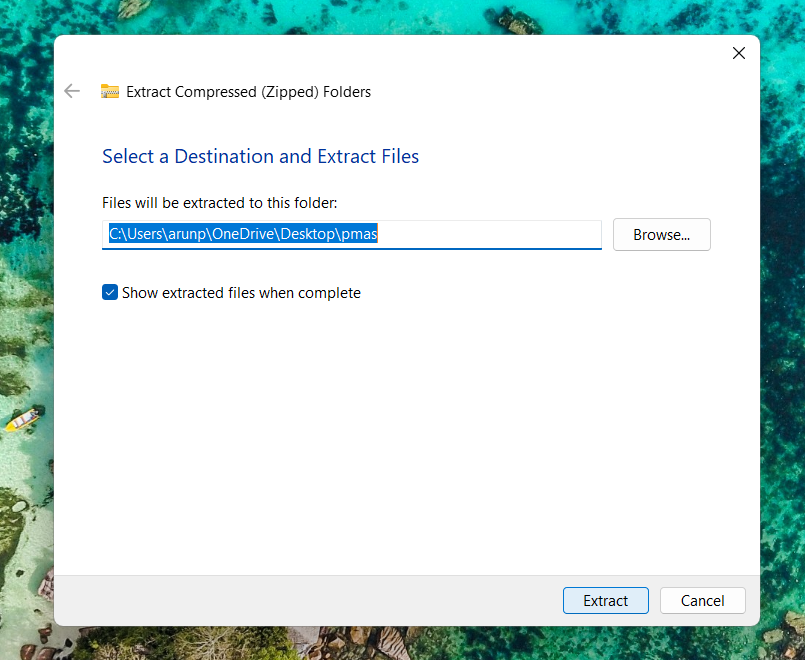
## Getting the code into your local system:

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

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



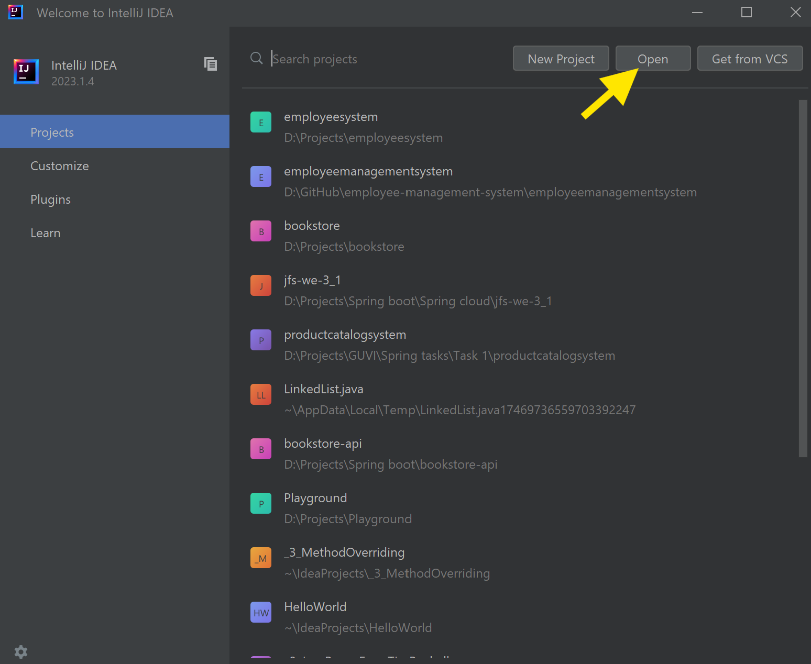
1. Then extract the downloaded zip folder into a folder as per your interest.



## 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.

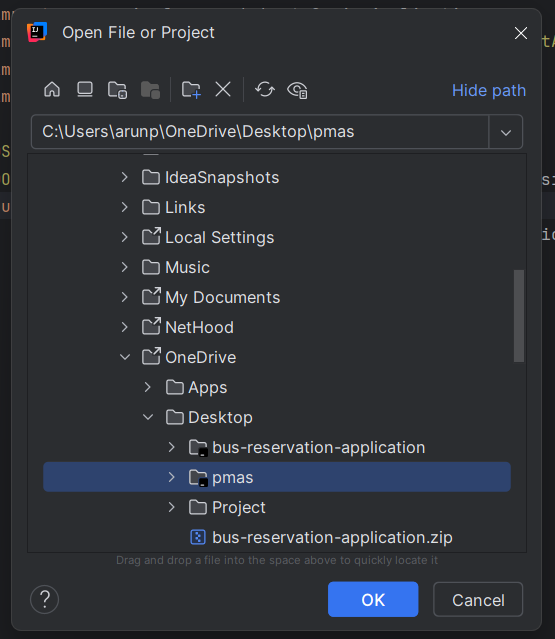
1. Open IntelliJ. Select *Open*.



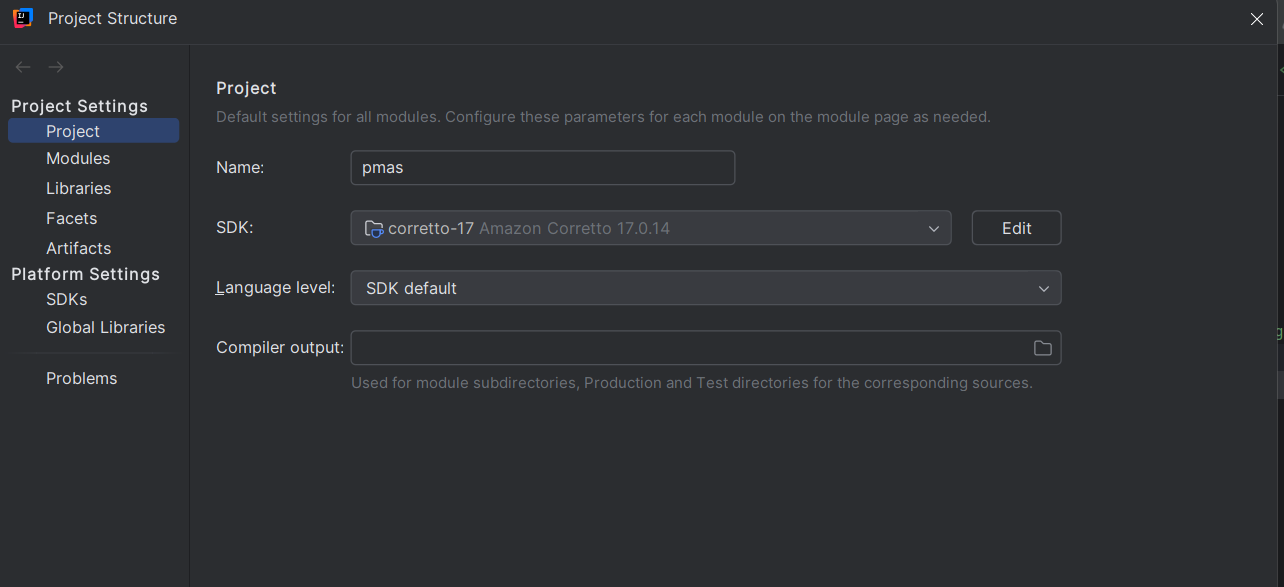
1. Browse to the path where we extracted our project and select *OK*.

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.*

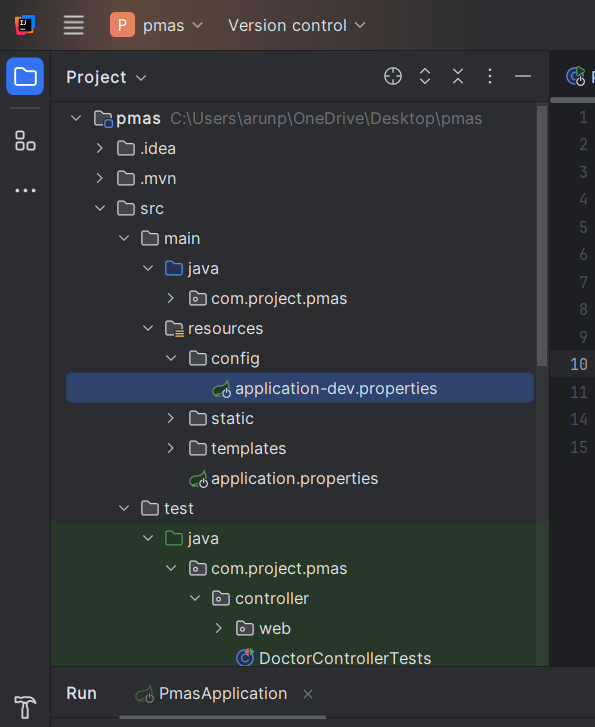


1. Go to *File -> Project Structure -> Project* and ensure that *JDK 17* is selected as the SDK



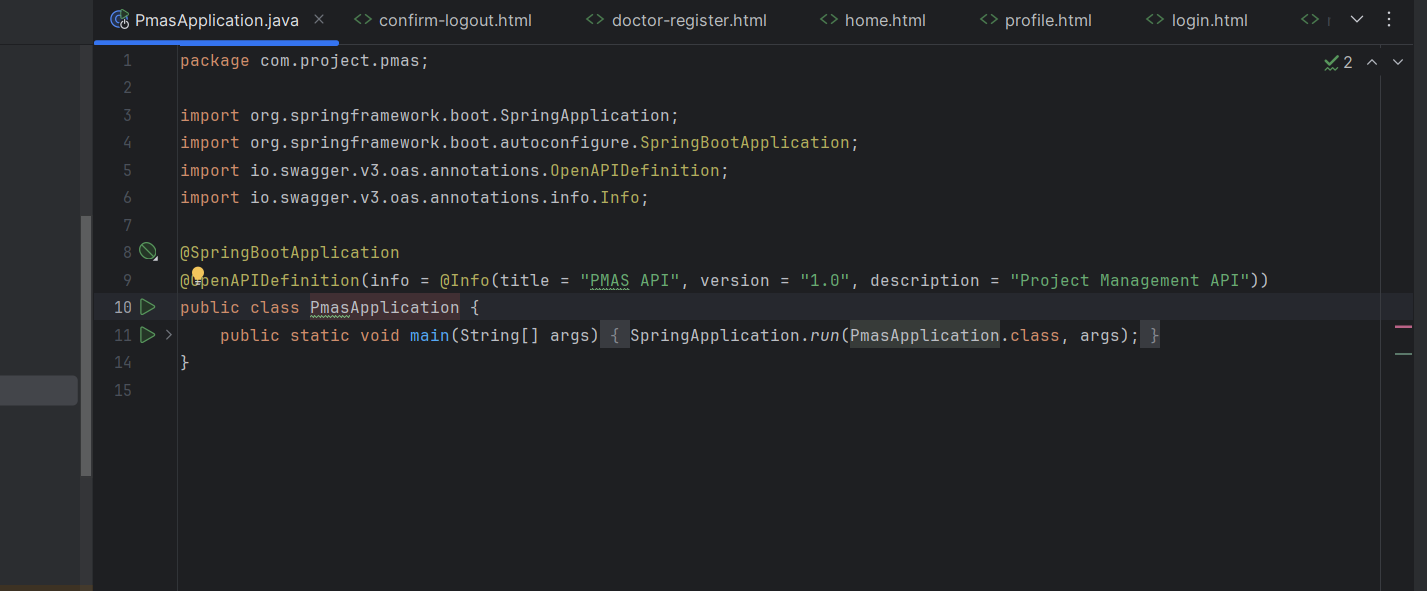
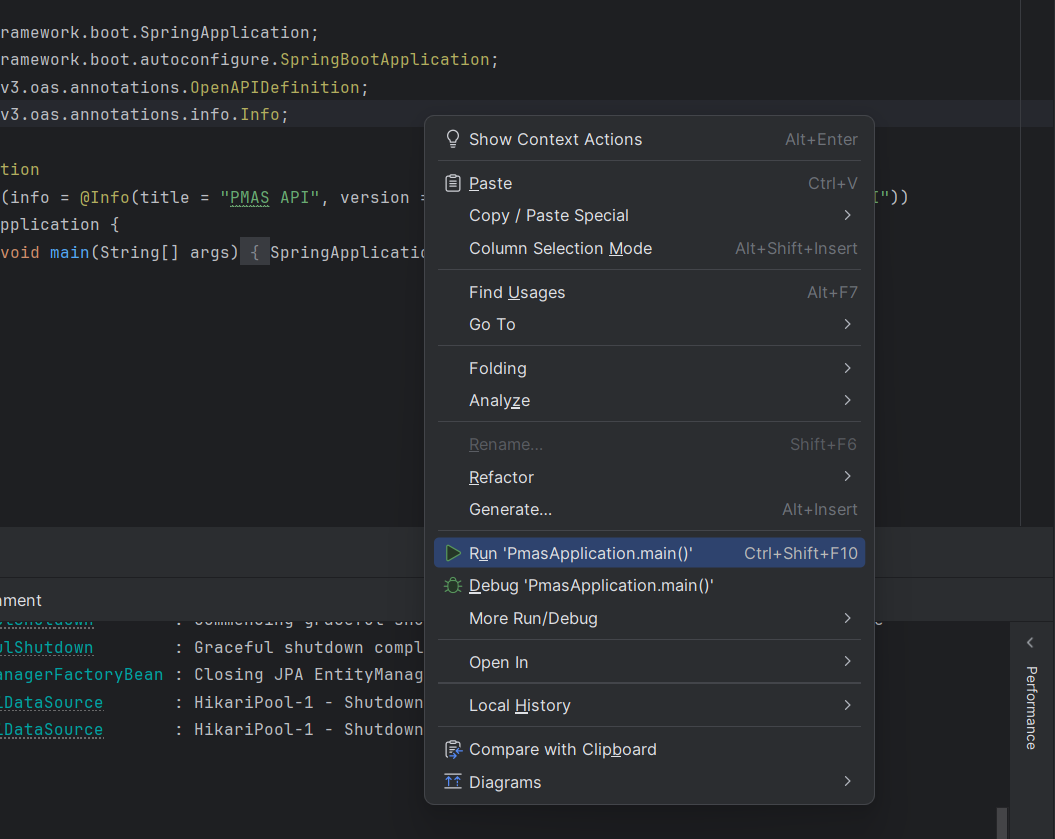
## Setting the database:

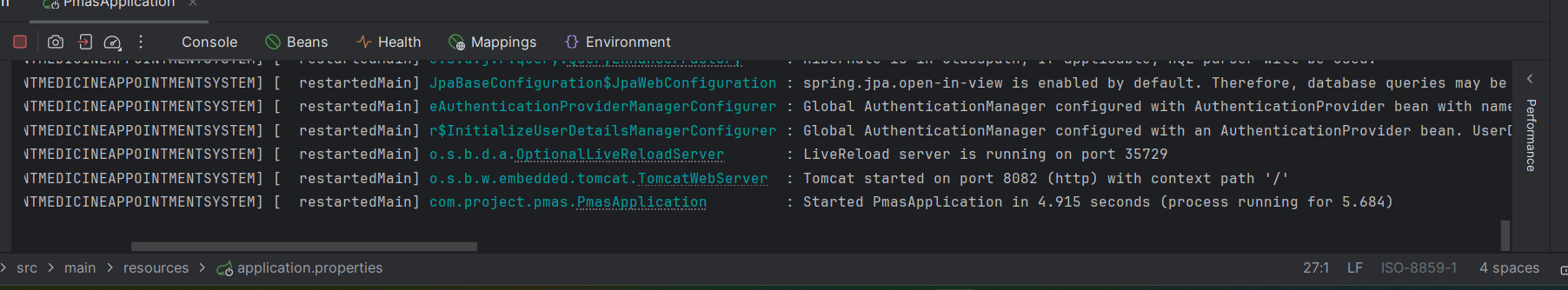
The application is created having MySQL as its database for storing its data. The main configurations of the application are written inside the application-dev.properties file inside

***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:

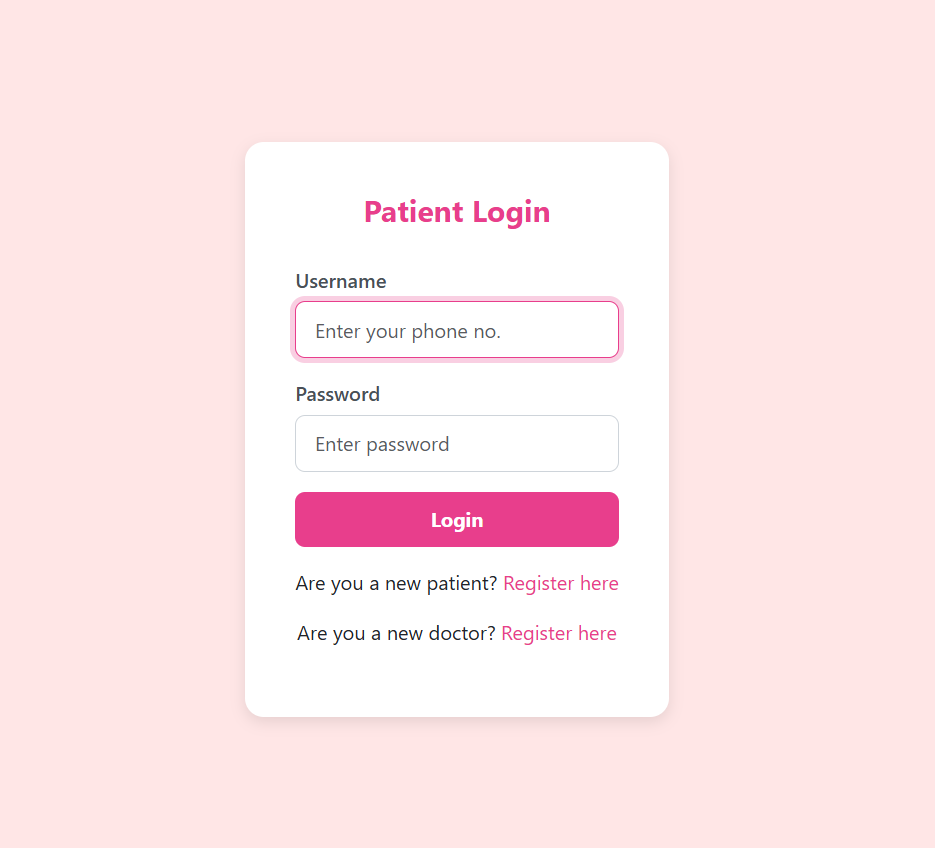
1. Before starting the application, ensure MySQL service is up and running.
2. Go to *pmas -> src -> main -> java -> com.project.pmas*
3. Right-click inside the *PatientmedicineappointmentsystemApplication.java* file and click RunPmasApplication.main()
4. On clicking Run, the application will start and you will see the below screen:

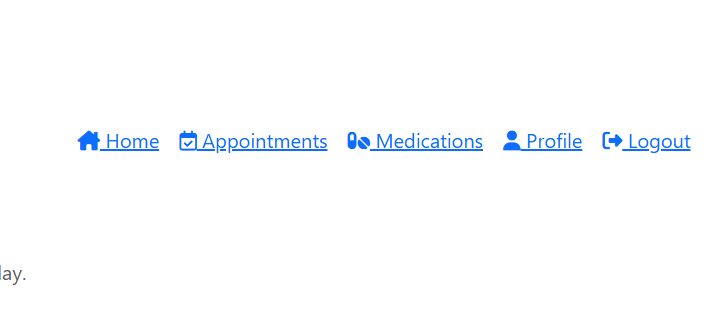


If the database name in the *application-dev.properties* and in the MySQL database match, Hibernate will automatically create a table as required and the system will get started.

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>/web/patients/login](http://localhost:%3cport-number%3e/web/patients/login). For port 8082: [http://localhost:8082/web/patients/login](http://localhost:8082/api/employees/home) . This should render the login page that after logging in will lead to the homepage of the application.

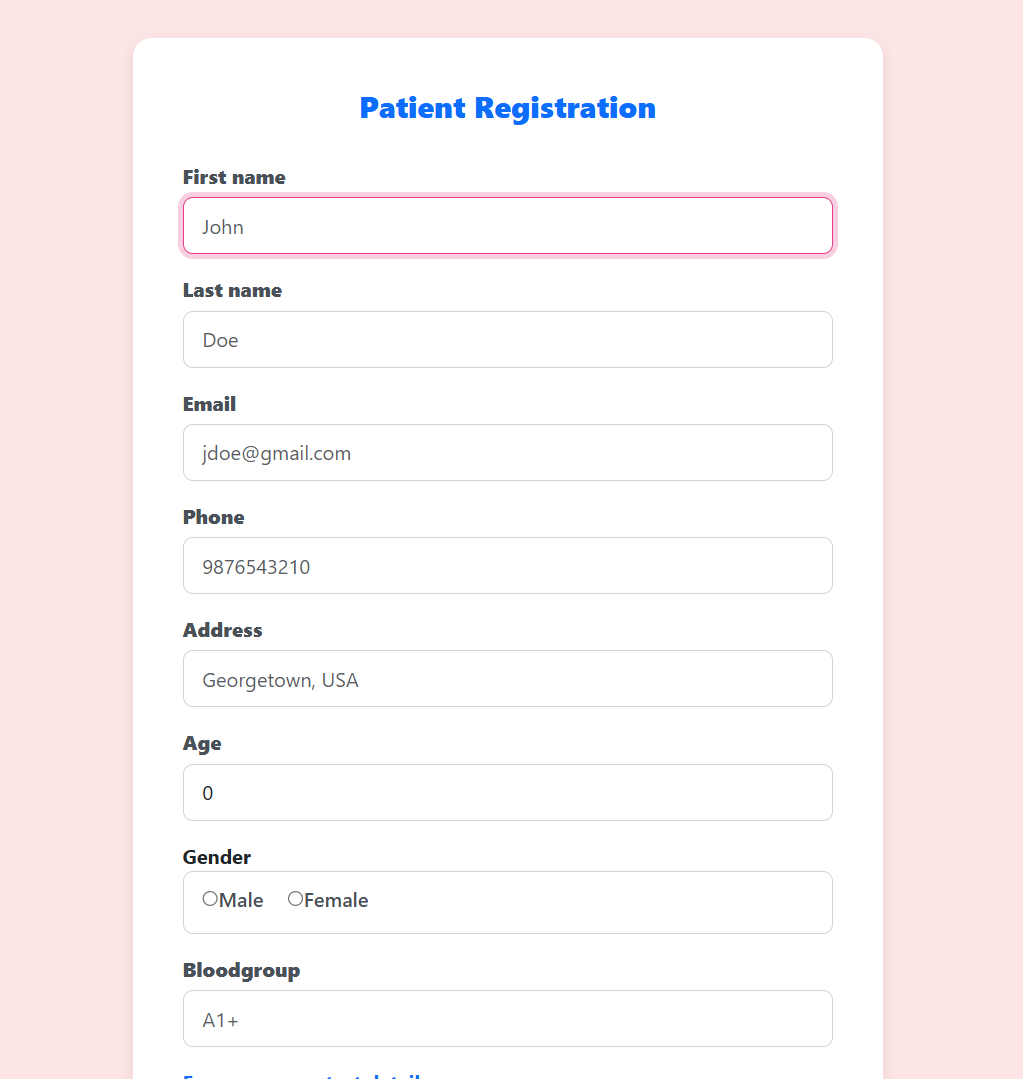
This is how the login page will look like:



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 login Page**

* **Endpoint:** /web/patients/login
* **Method:** GET
* **Description:** Renders the login page.
* **Response:** HTML page with form for accepting login credentials.

1. **Render registration page**

* **Endpoint:** /web/patients/register
* **Method:** GET
* **Description:** Renders a page with form to collect new patient’s details.
* **Response:** HTML page displaying the form for collecting new patient details.

1. **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.
* **Parameter:** Patient object of type SavePatientDto class with the patient details to be saved.
* **Response:** Redirection to the same page with a success/error message based on the outcome.

#### 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 registration page**

* **Endpoint:** /web/doctors/register
* **Method:** GET
* **Description:** Renders a page with form to collect new doctor’s details.
* **Response:** HTML page displaying the form for collecting new doctor details.

1. **Save the new doctor to database**

* **Endpoint:** /web/doctors/saveDoctor
* **Method:** POST
* **Description:** Makes a post request to save the doctor details into the database.
* **Parameter:** Doctor object of type SaveDoctorDto.
* **Response:** Redirection to the same page with a success/error message based on the outcome.

#### MEDICATIONWEBCONTROLLER.CLASS

1. **Getting the medication table**

* **Endpoint:** /web/medications/list
* **Method:** GET
* **Description:** Gets all the available medication of the user.
* **Response:** An HTML view showing the list of medications added.

1. **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.
* **Parameter:** Medication id.
* **Response:** An HTML view rendering the particular medication.

1. **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.
* **Response:** An HTML page with a form..

1. **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.
* **Parameter:** A Medication object of type SaveMedicationDto.
* **Response:** Redirection to the success/error page with a message based on the outcome.

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

* **Endpoint:** /web/medications/update-medication/{id}
* **Method:** GET
* **Description:** Collects required data through the webpage for update.
* **Parameter:** Medication id.
* **Response:** Renders the HTML form for collecting data to update the medication.

1. **Update existing medication**

* **Endpoint:** /web/medications/update/{id}
* **Method:** POST
* **Description:** Makes a post request to update the doctor details in the database.
* **Parameters:**
  + **Id:** Medication id,
  + **prescriptionDate:** The date in which the medication was created for the first time.
  + **saveMedicationDto:** Updated Medication data
* **Response:** Redirection to the success/error page with a message based on the outcome.

1. **Delete a medication**

* **Endpoint:** /web/ medications/delete/{id}
* **Method:** GET
* **Description:** Deletes selected medication from the medication list.
* **Parameter:**
  + **Id:** Medication id.
* **Response:** Redirection to the success/error page with a message based on the outcome.

#### APPOINTMENTWEBCONTROLLER.CLASS

1. **Getting the appointment table**

* **Endpoint:** /web/appointments/list
* **Method:** GET
* **Description:** Gets all the scheduled appointments of the patient.
* **Response:** An HTML view rendering the upcoming and completed appointment tables.

1. **Filter doctors on a particular speciality**

* **Endpoint:** /web/appointments/doctors
* **Method:** POST
* **Description:** A method to get all the doctors of a particular speciality.
* **Parameter:**
  + **Speciality:** A string to filter out doctors based on their speciality.
* **Response:** A JSON response with a doctors list of the required speciality.

1. **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.
* **Response:** An HTML view asking for details for a new appointment.

1. **Fetch appointment slots of a doctor on a particular date**

* **Endpoint:** /web/appointments/checkSlots
* **Method:** POST
* **Description:** A method to get all the slots of the given doctor on the give date.
* **Parameter:**
  + **doctorId:** Doctor id
  + **date:** The date on which the appointments are to be fetched.
* **Response:** A JSON response having the slot data.

1. **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.
* **Parameters:**
  + **saveAppointmentDto:** Appointment object with details.
* **Response:** A success or an error page based on the outcome.

1. **Delete an appointment**

* **Endpoint:** /web/ appointments/delete/{id}
* **Method:** GET
* **Description:** Deletes the appointment object upon clicking delete from the webpage.
* **Parameter:**
  + **Id:** Appointment id
* **Response:** Redirection to a success or an error page based on the outcome.

### 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:** /doctors/getAll
* **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,

"firstName": "Selva",

"lastName": "M",

"gender": "Male",

"mobile": "9999999909",

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

"speciality": "General Medicine",

"experienceInYears": 3,

"qualifications": "MBBS, MS",

"languagesSpoken": "Tamil, English, Hindi",

"officeAddress": "Chennai"

},

{

"id": 2,

"firstName": “kira",

"lastName": "Kumar",

"gender": "Male",

"mobile": "9999999910",

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

"speciality": "ENT",

"experienceInYears": 3,

"qualifications": "MBBS, ENT",

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

"officeAddress": "Chennai"

}

]

1. **Add a New Doctor**

* **Endpoint:** /doctors/add
* **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,

    "firstName": "Yuvana",

    "lastName": "P",

    "gender": "Male",

    "mobile": "9988776655",

    "email": "jsmith@gmail.com",

    "speciality": "Dermatology",

    "experienceInYears": 5,

    "qualifications": "MBBS",

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

    "officeAddress": "Chennai"

}

1. **Add Multiple Doctors**

* **Endpoint:** /doctors/addAll
* **Method:** POST
* **Description:** A RESTful method to add multiple doctors into the database
* **Request Body:**
  + **Content-Type:** json – SaveDoctorDto List object
  + **Sample Body(Json):**

[

{

    "firstName": "Yugi",

    "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 List of DoctorDTO objects.
* **Sample Response(Json):**

[

{

    "id": 4,

    "firstName": "Yuvan",

    "lastName": "P",

    "gender": "Male",

    "mobile": "9988776655",

    "email": "jsmith@gmail.com",

    "speciality": "Dermatology",

    "experienceInYears": 5,

    "qualifications": "MBBS",

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

    "officeAddress": "Chennai"

}

]

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,

    "firstName": "Fzsil",

    "lastName": "U",

    "gender": "Female",

    "mobile": "6666666692",

    "email": "farah@hspt.com",

    "speciality": "Pathology",

    "experienceInYears": 4,

    "qualifications": "MBBS, DNB Pathology",

    "languagesSpoken": "Urdu, English",

    "officeAddress": "Kolkata"

}

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):**

{

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

{

    "id": 4,

    "firstName": "Yuvan",

    "lastName": "P",

    "gender": "Male",

    "mobile": "9988776655",

    "email": "jsmith@gmail.com",

    "speciality": "Dermatology",

    "experienceInYears": 5,

    "qualifications": "MBBS",

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

    "officeAddress": "Chennai"

}

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.
* **Blood Group**
  + **Required:** Yes(From Frontend)
* **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".
* **Previous Diagnoses**
  + **Required:** Yes
  + **Validation:** Must not be null or empty.
  + **Constraints:**
    - @NotBlank(message = "Previous diagnoses cannot be blank.")
* **Surgeries**
  + **Required:** Yes
  + **Validation:** Must not be null or empty.
  + **Constraints:**
    - @NotBlank(message = "Surgeries cannot be blank.")
* **Allergies**
  + **Required:** Yes
  + **Validation:** Must not be null or empty.
  + **Constraints:**
    - @NotBlank(message = "Allergies cannot be blank.")
* **Vaccination History**
  + **Required:** Yes
  + **Validation:** Must not be null or empty.
  + **Constraints:**
    - @NotBlank(message = "Vaccination history cannot be blank.")
* **Smoker:**
  + **Required:** Yes(From Frontend)
  + **Validation:** No specific validation rules provided in the backend. Will alot values based on the radio type input at the webpage.
* **Consumes Alcohol:**
  + **Required:** Yes(From Frontend)
  + **Validation:** No specific validation rules provided in the backend. Will alot values based on the radio type input at the webpage.

## 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:

* **First 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 ]\*$", message = "Name can only contain letters and spaces")
* **Last Name**
  + **Required:** Yes
  + **Validation:** Must not be null or empty. Can have only letters and spaces
  + **Constraints:**
    - @NotBlank(message = "Last name of a doctor cannot be blank.")
    - @Pattern(regexp = "^[A-Za-z ]\*$", 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.
* **Languages Spoken**
  + **Required:** Yes(From front-end)
  + **Validation:** No specific validation rules provided in the backend.
* **Office Address**
  + **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 |
| first\_name | varchar(255) | DEFAULT NULL |
| last\_name | varchar(255) | DEFAULT NULL |
| mobile | varchar(255) | DEFAULT NULL |
| password | varchar(255) | DEFAULT NULL |
| age | int | NOT NULL |
| allergies | varchar(255) | DEFAULT NULL |
| blood\_group | varchar(255) | DEFAULT NULL |
| consumes\_alcohol | bit(1) | NOT 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 |
| is\_smoker | bit(1) | NOT NULL |
| previous\_diagnoses | varchar(255) | DEFAULT NULL |
| surgeries | varchar(255) | DEFAULT NULL |
| vaccination\_history | varchar(255) | DEFAULT NULL |

### Patient Entity schema(Spring Boot):

|  |  |
| --- | --- |
| **Column Name** | **Datatype** |
| id | Long |
| address | String |
| email | String |
| first\_name | String |
| last\_name | String |
| mobile | String |
| password | String |
| age | int |
| allergies | String |
| blood\_group | String |
| consumes\_alcohol | boolean |
| emergency\_contact\_mobile | String |
| emergency\_contact\_name | String |
| emergency\_contact\_relation | String |
| gender | String |
| is\_smoker | boolean |
| previous\_diagnoses | String |
| surgeries | String |
| vaccination\_history | String |

### Doctor table schema(Database):

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** | **Remarks** |
| id | bigint | PRIMARY KEY, NOT NULL, AUTO\_INCREMENT |
| speciality | varchar(255) | DEFAULT NULL |
| first\_name | varchar(255) | DEFAULT NULL |
| last\_name | varchar(255) | DEFAULT NULL |
| email | varchar(255) | DEFAULT NULL |
| experience\_in\_years | int | NOT NULL |
| gender | varchar(255) | DEFAULT NULL |
| languages\_spoken | varchar(255) | DEFAULT NULL |
| mobile | varchar(255) | DEFAULT NULL |
| office\_address | varchar(255) | DEFAULT NULL |
| qualifications | varchar(255) | DEFAULT NULL |

### Doctor table schema(Spring boot):

|  |  |
| --- | --- |
| **Column Name** | **Datatype** |
| id | Long |
| speciality | String |
| first\_name | String |
| last\_name | String |
| email | String |
| experience\_in\_years | int |
| gender | String |
| languages\_spoken | String |
| mobile | String |
| office\_address | 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 |
| start\_date | date | DEFAULT NULL |
| patient\_id | bigint | NOT NULL, Foreign Key(References `id` from patient table.) |
| dosage | varchar(255) | DEFAULT NULL |
| prescription\_date | date | DEFAULT NULL |
| status | varchar(255) | DEFAULT NULL |
| updated\_date | date | DEFAULT NULL |

### Medication table schema(Spring boot):

|  |  |
| --- | --- |
| **Column Name** | **Datatype** |
| id | Long |
| end\_date | LocalDate |
| 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)

[

{

"firstName": "Rajesh",

"lastName": "M",

"gender": "Male",

"mobile": "9999999912",

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

"speciality": "Pathology",

"experienceInYears": 5,

"qualifications": "MBBS, MD Pathology",

"languagesSpoken": "Tamil, English, Hindi",

"officeAddress": "Chennai"

},

{

"firstName": "Sita",

"lastName": "R",

"gender": "Female",

"mobile": "8888888811",

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

"speciality": "Pathology",

"experienceInYears": 8,

"qualifications": "MBBS, MD Pathology",

"languagesSpoken": "Hindi, English, Gujarati",

"officeAddress": "Mumbai"

},

{

"firstName": "Ankit",

"lastName": "T",

"gender": "Male",

"mobile": "7777777701",

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

"speciality": "Pathology",

"experienceInYears": 6,

"qualifications": "MBBS, MD Pathology",

"languagesSpoken": "English, Hindi",

"officeAddress": "Delhi"

},

{

"firstName": "Farah",

"lastName": "U",

"gender": "Female",

"mobile": "6666666692",

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

"speciality": "Pathology",

"experienceInYears": 4,

"qualifications": "MBBS, DNB Pathology",

"languagesSpoken": "Urdu, English",

"officeAddress": "Kolkata"

},

{

"firstName": "Arjun",

"lastName": "K",

"gender": "Male",

"mobile": "5555555583",

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

"speciality": "Orthopedics",

"experienceInYears": 10,

"qualifications": "MBBS, MS Orthopedics",

"languagesSpoken": "Telugu, English",

"officeAddress": "Hyderabad"

},

{

"firstName": "Priya",

"lastName": "S",

"gender": "Female",

"mobile": "4444444474",

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

"speciality": "Orthopedics",

"experienceInYears": 6,

"qualifications": "MBBS, MD Orthopedics",

"languagesSpoken": "Marathi, English",

"officeAddress": "Pune"

},

{

"firstName": "Vinay",

"lastName": "T",

"gender": "Male",

"mobile": "3333333365",

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

"speciality": "Dermatology",

"experienceInYears": 4,

"qualifications": "MBBS, MD Dermatology",

"languagesSpoken": "Kannada, English, Hindi",

"officeAddress": "Bangalore"

},

{

"firstName": "Neha",

"lastName": "P",

"gender": "Female",

"mobile": "2222222256",

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

"speciality": "Dermatology",

"experienceInYears": 5,

"qualifications": "MBBS, DNB Dermatology",

"languagesSpoken": "Hindi, Punjabi, English",

"officeAddress": "Delhi"

},

{

"firstName": "Rahul",

"lastName": "J",

"gender": "Male",

"mobile": "1111111147",

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

"speciality": "Cardiology",

"experienceInYears": 7,

"qualifications": "MBBS, MD Cardiology",

"languagesSpoken": "Hindi, English",

"officeAddress": "Ahmedabad"

},

{

"firstName": "Kiran",

"lastName": "A",

"gender": "Female",

"mobile": "0000000038",

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

"speciality": "Cardiology",

"experienceInYears": 9,

"qualifications": "MBBS, DNB Cardiology",

"languagesSpoken": "Telugu, English",

"officeAddress": "Kochi"

},

{

"firstName": "Mehul",

"lastName": "B",

"gender": "Male",

"mobile": "9999999929",

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

"speciality": "Cardiology",

"experienceInYears": 5,

"qualifications": "MBBS, MD Cardiology",

"languagesSpoken": "Gujarati, English",

"officeAddress": "Vadodara"

},

{

"firstName": "Anil",

"lastName": "D",

"gender": "Male",

"mobile": "8888888810",

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

"speciality": "Neurology",

"experienceInYears": 6,

"qualifications": "MBBS, MD Neurology",

"languagesSpoken": "Malayalam, English",

"officeAddress": "Thiruvananthapuram"

},

{

"firstName": "Sneha",

"lastName": "C",

"gender": "Female",

"mobile": "7777777703",

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

"speciality": "Neurology",

"experienceInYears": 4,

"qualifications": "MBBS, DNB Neurology",

"languagesSpoken": "Tamil, English",

"officeAddress": "Coimbatore"

},

{

"firstName": "Nikhil",

"lastName": "E",

"gender": "Male",

"mobile": "6666666694",

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

"speciality": "Neurology",

"experienceInYears": 3,

"qualifications": "MBBS, MD Neurology",

"languagesSpoken": "Hindi, English",

"officeAddress": "Delhi"

},

{

"firstName": "Tina",

"lastName": "F",

"gender": "Female",

"mobile": "5555555585",

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

"speciality": "Neurology",

"experienceInYears": 8,

"qualifications": "MBBS, DNB Neurology",

"languagesSpoken": "English, Kannada",

"officeAddress": "Bangalore"

},

{

"firstName": "Amit",

"lastName": "H",

"gender": "Male",

"mobile": "4444444478",

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

"speciality": "General Medicine",

"experienceInYears": 9,

"qualifications": "MBBS, MD General Medicine",

"languagesSpoken": "Hindi, English",

"officeAddress": "Delhi"

},

{

"firstName": "Sonal",

"lastName": "I",

"gender": "Female",

"mobile": "3333333360",

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

"speciality": "General Medicine",

"experienceInYears": 4,

"qualifications": "MBBS, DNB General Medicine",

"languagesSpoken": "English, Marathi",

"officeAddress": "Mumbai"

},

{

"firstName": "Ravi",

"lastName": "K",

"gender": "Male",

"mobile": "1111111142",

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

"speciality": "General Medicine",

"experienceInYears": 3,

"qualifications": "MBBS, DNB General Medicine",

"languagesSpoken": "Telugu, English",

"officeAddress": "Hyderabad"

}

]