# **Patient Medicine and Appointment System**

This project is a full‑stack application built using Spring Boot, MongoDB, HTML, CSS, and JavaScript. The system allows patients to register, log in, book appointments with doctors (filtered by specialization), and view their medications. Doctors can log in, view their appointments (with the patient’s name mapped from doctor records), and manage (create, update, delete) medications for specific appointments.

## **Table of Contents**

1. Features
2. Project Architecture
3. Setup & Configuration
4. Running the Project
5. API Documentation
6. Data Validation Rules
7. Integration Tests
8. Swagger UI
9. Frontend Configuration
10. Deployment & Submission
11. Additional Notes

## **1. Features**

* **Patient Flow:**
  + New patients can register and log in.
  + Patients can book appointments by selecting a specialization.
  + Appointments display the doctor’s name instead of just the doctor’s ID.
  + Patients can view their appointments and, for each appointment, view a read‑only list of medications.
* **Doctor Flow:**
  + Doctors log in using their email and a default password (“Doctor@1234”).
  + Doctors see a dashboard listing all appointments assigned to them.
  + For each appointment, doctors have a “Manage Medications” button.
  + In the medication management page, doctors can add, update, or delete medications (with fields for medication name, dosage, and the number of days) for the specific appointment.
* **General:**
  + API endpoints are documented via Swagger UI.
  + The frontend API base URL is easily configurable.
  + Every screen includes Back and Logout buttons for user convenience.

## **2. Project Architecture**

* **Backend:**
  + **Framework:** Spring Boot
  + **Database:** MongoDB (Atlas or local)
  + **Dependencies:** Spring Web, Spring Data MongoDB, Spring Boot Starter Validation, springdoc-openapi-ui (for Swagger), Spring Boot Starter Test.
  + **Modules:** Patient, Doctor, Appointment, Medication.
  + **API Endpoints:** RESTful endpoints for registration, login, appointment booking, and medication management.
* **Frontend:**
  + **Technologies:** HTML, CSS, JavaScript (with Bootstrap)
  + **Pages:** Index, Patient Registration, Patient Login, Patient Dashboard, Available Appointment Slots, Patient Medications (read‑only view), Doctor Login, Doctor Dashboard, Doctor Medication Management.
  + **Configuration:** The backend API base URL is set as a constant in js/main.js for easy configuration.

## **3. Setup & Configuration**

### **Prerequisites**

* **Java Development Kit (JDK):** Java 11 or later.
* **Maven:** For building the backend project.
* **MongoDB:** Either a local instance or a MongoDB Atlas cluster.
* **IDE:** IntelliJ IDEA (or any preferred IDE).
* **Web Browser:** For testing the frontend.

### 

### 

### **Backend Setup**

**Clone the Repository:**  
 git clone https://github.com/your\_username/appointmentsystem.git

cd appointmentsystem/backend

1. **Configure MongoDB Connection:**
   * Open src/main/resources/application.properties.

Update the MongoDB URI with your credentials:  
  
spring.data.mongodb.uri=mongodb+srv://user:Priya123@clusterdeployment.kirge.mongodb.net/patient\_system?retryWrites=true&w=majority&appName=ClusterDeployment

server.port=8080

**Build the Project:**  
 mvn clean install

1. **Run the Application:**  
    mvn spring-boot:run
2. The backend will run on http://localhost:8080.

### 

### 

### **Frontend Setup**

**Clone/Copy the Frontend Folder:** Ensure that the frontend folder structure is as follows:

frontend/

├── index.html

├── patient\_register.html

├── patient\_login.html

├── patient\_dashboard.html

├── patient\_medications.html

├── available\_slots.html

├── doctor\_login.html

├── doctor\_dashboard.html

├── doctor\_medications.html

├── css/

│ └── styles.css

└── js/

└── main.js

1. **Configure API Base URL:**

In js/main.js, set the constant for your API base URL:  
 javascript  
  
 const API\_BASE\_URL = "http://localhost:8080/api";

* + Replace all occurrences of 'http://localhost:8080/api' with ${API\_BASE\_URL}.

1. **Serve the Frontend:**
   * Open the HTML files directly in your browser or use a static file server (e.g., Live Server in VS Code).

## 

## 

## **4. Running the Project**

### **Backend**

* **Start the Backend Application:** Run mvn spring-boot:run from the backend folder.

**Swagger UI:**

Access Swagger UI at: http://localhost:8080/swagger-ui/index.html

* This interface provides interactive API documentation.

### **Frontend**

* **Index Page:** Open index.html in your browser (e.g., http://127.0.0.1:5500/index.html).
* **Patient Flow:**
  + Navigate to Patient Registration or Patient Login.
  + After login, the patient is redirected to the Patient Dashboard.
  + From the dashboard, the patient can book appointments via **available\_slots.html** and view medications for each appointment via the “View Medications” button (which leads to **patient\_medications.html**).
* **Doctor Flow:**
  + Navigate to Doctor Login.
  + After login, the doctor is redirected to the Doctor Dashboard.
  + The dashboard lists appointments and includes a “Manage Medications” button for each appointment that leads to **doctor\_medications.html**.

## **5. API Documentation**

### **Patient Endpoints**

* **POST /api/patients/register**
  + **Description:** Register a new patient.

**Request Body:** json  
  
{

"name": "John Doe",

"email": "john@example.com",

"contactNumber": "1234567890",

"medicalHistory": "No allergies",

"password": "patient123"

}

* + **Response:** Returns the created patient object, including the generated id.
* **POST /api/patients/login**
  + **Description:** Patient login.

**Request Body:** json  
  
{

"email": "john@example.com",

"password": "patient123"

}

* + **Response:** Returns the patient object if credentials are valid.
* **GET /api/patients/{id}**
  + **Description:** Retrieve a patient by ID.

### **Doctor Endpoints**

* **POST /api/doctors/register**
  + **Description:** Register a new doctor.

**Request Body:** json  
{

"name": "Dr. Smith",

"specialization": "General Medicine",

"availableSlots": ["2025-05-01T09:00:00", "2025-05-01T10:00:00"],

"email": "dr.smith@example.com",

"password": "Doctor@1234"

}

* + **Response:** Returns the created doctor object.
* **POST /api/doctors/login**
  + **Description:** Doctor login.

**Request Body:** json  
{

"email": "dr.smith@example.com",

"password": "Doctor@1234"

}

* + **Response:** Returns the doctor object if credentials are valid.
* **GET /api/doctors**
  + **Description:** Retrieve all doctors.
* **GET /api/doctors/{doctorId}/appointments**
  + **Description:** Retrieve appointments for a specific doctor.

### **Appointment Endpoints**

* **POST /api/appointments**
  + **Description:** Book a new appointment.

**Request Body:** json  
{

"patientId": "patient\_id\_here",

"doctorId": "doctor\_id\_here",

"appointmentTime": "2025-05-01T09:00:00"

}

* + **Response:** Returns the created appointment object.
* **GET /api/appointments/patient/{patientId}**
  + **Description:** Get appointments for a patient.
* **GET /api/appointments/doctor/{doctorId}**
  + **Description:** Get appointments for a doctor.
* **PUT /api/appointments/{id}/diagnosis**
  + **Description:** Update diagnosis and doctor comments for an appointment.

**Request Body:** json  
{

"diagnosis": "Flu",

"doctorComments": "Prescribe rest and hydration"

}

### **Medication Endpoints**

* **POST /api/medications**
  + **Description:** Add a new medication.

**Request Body:** json  
{

"patientId": "patient\_id\_here",

"appointmentId": "appointment\_id\_here",

"medicationName": "Paracetamol",

"dosage": "500mg",

"days": 5

}

* + **Response:** Returns the created medication object.
* **PUT /api/medications/{id}**
  + **Description:** Update an existing medication.
* **DELETE /api/medications/{id}**
  + **Description:** Delete a medication.
* **GET /api/medications/patient/{patientId}**
  + **Description:** Retrieve medications for a patient.
* **GET /api/medications/appointment/{appointmentId}**
  + **Description:** Retrieve medications for a specific appointment.

## **6. Data Validation Rules**

* **Patient Registration:**
  + name: Must not be blank.
  + email: Must be a valid email address and not blank.
  + contactNumber: Must not be blank.
  + password: Must not be blank.
* **Doctor Registration:**
  + name: Must not be blank.
  + specialization: Must be one of the allowed specializations (e.g., "Orthopedics", "General Medicine", "Oncology", "Gynecology", "Pediatrics").
  + email: Must be valid and not blank.
  + password: Must not be blank (default is "Doctor@1234").
  + availableSlots: Must be an array of ISO 8601 formatted date-time strings.
* **Appointment Booking:**
  + patientId: Must not be blank.
  + doctorId: Must not be blank.
  + appointmentTime: Must be a valid date-time string.
* **Medication Management:**
  + medicationName: Must not be blank.
  + dosage: Must not be blank.
  + days: Must be a positive integer.

## **7. Integration Tests**

The integration tests are located in src/test/java/com/example/appointmentsystem/ApplicationIntegrationTest.java and cover:

* Patient registration and login.
* Doctor registration and login.
* Appointment booking.
* Medication CRUD operations.

### **Running the Tests**

From the backend folder, run:

mvn test

This command runs all integration tests and outputs the results in the console.

## **8. Swagger UI**

After starting the backend, Swagger UI is available at:

http://localhost:8080/swagger-ui/index.html

This interface provides interactive documentation of all API endpoints, including request/response formats.

## **9. Frontend Configuration**

In the js/main.js file, the backend API URL is configurable via a constant:

javascript

const API\_BASE\_URL = "http://localhost:8080/api";

Replace all fetch calls with ${API\_BASE\_URL}. For example:

javascript

fetch(`${API\_BASE\_URL}/doctors`)

This allows you to easily change the backend URL in one place.

## **10. Deployment & Submission**

1. **Backend:**
   * Build and run the backend using Maven.

Package the backend as an executable JAR:  
  
mvn clean package

* + Push your backend code to GitHub.

1. **Frontend:**
   * Host your static frontend files using GitHub Pages, Netlify, or a similar service.
   * Push your frontend code to a separate GitHub repository.

## **11. Additional Notes**

* **Back and Logout Buttons:** Every page includes Back and Logout buttons for user convenience.
* **Doctor Name Mapping:** The Patient Dashboard loads a mapping of doctor IDs to names so that appointments display the doctor’s name.
* **Testing:** Ensure you have sample data (doctors, patients, appointments, medications) in MongoDB for testing all flows.

## 

## 

## 

## **How to Run the Project**

### **Backend**

1. Navigate to the backend folder.

Run: mvn spring-boot:run

1. Verify Swagger UI at: http://localhost:8080/swagger-ui/index.html

### **Frontend**

1. Open index.html in your browser.
2. Use the navigation links to log in as a patient or doctor and follow the flows.