# MediTrack System – Backend Documentation

## 1. Overview

The **Prescription Management System** backend is a Spring Boot–based application that provides a secure, efficient, and scalable REST API for managing prescriptions. It ensures that only authenticated users can access and manipulate data, while supporting full CRUD operations, prescription history tracking, and day-wise reporting.

The backend is powered by **Spring Boot 3.5.5**, **Spring Security**, and **Spring Data JPA**, with **H2** as the development database. The design follows a **layered architecture**, making it maintainable and extensible for future enhancements.

## 2. Features

The backend fully supports the following:

- **Authentication & Authorization**
    - Implemented using **Spring Security**.
    - Users and roles stored in H2 database.
    - Passwords hashed with **BCrypt**.
    - No anonymous access – unauthenticated users are redirected to login.
- **Prescription Management (CRUD)**

    - Create, read, update, and delete prescriptions.
    - Mandatory fields validated: date, name, age (range), gender.
    - Optional fields supported: diagnosis, medicines, next visit date.

- **Prescription Filtering by Date Range**
    - Retrieve prescriptions for any date range (default: current month).
- **Day-Wise Prescription Reports**
    - Generates prescription counts per day using a custom JPA query.
- **Error Handling & Validation**
    - Spring Validation annotations (@NotNull, @NotBlank, @Min, @Max).
    - Global exception handler ensures consistent JSON error responses.
- **REST API (JSON)**
    - All data exposed as REST endpoints for frontend integration (e.g., React).

## 3. Extra Backend Features

In addition to core requirements, the backend provides:

- **Prescription History Entity** – A separate table and API for tracking history.
- **User Seeder** – Seeds an admin account into the database on startup.
- **CORS Configuration** – Enables cross-origin requests from http://localhost:5173.
- **Batch Deletion** – Delete prescriptions in bulk by date range.
- **Optimized Reporting** – Uses JPQL queries for efficient day-wise counts.

## 4. System Architecture

The backend follows a standard layered architecture:

1. **Controller Layer** – Handles REST API endpoints.
2. **Service Layer** – Implements business logic.
3. **Repository Layer** – Database operations via Spring Data JPA.
4. **Model Layer** – Entities representing database tables.
5. **Security Layer** – Authentication, authorization, password hashing.
6. **Exception Handling** – Centralized error handling.
7. **Database Layer** – H2 in-memory DB (configurable to MySQL/PostgreSQL).

## 5. Project Structure

```
src/main/java/com/prescription/prescription_backend/
│
├── config/                 # Security & CORS configs
│   ├── GlobalCorsConfig.java
│   └── SecurityConfig.java
│
├── controller/             # REST controllers
│   ├── AuthController.java
│   ├── PrescriptionController.java
│   └── HistoryController.java
│
├── dto/                    # Data Transfer Objects
│   └── DayWiseCount.java
│
├── exception/              # Exception handling
│   └── GlobalExceptionHandler.java
│
├── model/                  # Entities
│   ├── User.java
│   ├── Prescription.java
│   └── History.java
│
├── repository/             # JPA Repositories
│   ├── UserRepository.java
│   ├── PrescriptionRepository.java
│   └── HistoryRepository.java
│
├── seeding/                # Data seeders
│   ├── UserSeeder.java
│   ├── DataSeeder.java
│   └── HistoryDataSeeder.java
│
└── service/                # Business services
    ├── PrescriptionService.java
    └── HistoryService.java
```

# 6. Dependencies

The backend uses the following dependencies (from pom.xml):

- **Spring Boot Starter Web**
    - For building REST APIs.
- **Spring Boot Starter Data JPA**
    - ORM and database interactions with Hibernate.
- **Spring Boot Starter Security**
    - User authentication, authorization, and session management.
- **Spring Boot Starter Validation**
    - Validation annotations (@NotBlank, @Valid, etc.) for request data.
- **H2 Database**
    - In-memory database for development/testing.
- **Spring Boot Starter Test**
    - JUnit, Mockito, and other testing utilities.
- **Spring Security**
    - For secure endpoints.
- **Lombok**
    - Reduces boilerplate with annotations like @Getter, @Setter, @Builder.
- **Thymeleaf + Thymeleaf Spring Security Extras** *(optional, not critical)*
    - Template engine integration (not used in core APIs, but included).

# 7. API Endpoints

### Authentication

- POST /api/v1/auth/login – Authenticate user with username and password.

### Prescription

- GET /api/v1/prescription – Get all prescriptions
- GET /api/v1/prescription/{id} – Get by ID
- GET /api/v1/prescription/by-name?name={name} – Get by patient name
- GET /api/v1/prescription/by-gender?gender={gender} – Get by gender
- GET /api/v1/prescription/by-date?start={date}&end={date} – Get by date range
- GET /api/v1/prescription/daywise-report?start={date}&end={date} – Day-wise report
- POST /api/v1/prescription – Create prescription
- PUT /api/v1/prescription/{id} – Update prescription
- DELETE /api/v1/prescription/{id} – Delete prescription
- DELETE /api/v1/prescription/by-date?start={date}&end={date} – Delete by date range

### History

- Same structure as prescriptions, but operates on the **History** entity.

# 8. Database Design

- **User** – id, username, password, roles
- **Prescription** – id, date, patientName, age, gender, diagnosis, medicines, nextVisitDate
- **History** – Same fields as Prescription, stored separately for archival purposes

## 9. Future Enhancements

- Swagger/OpenAPI integration for API documentation.
- Email notifications for next-visit reminders.
- Role-based access control (admin vs. regular user).
- MySQL/PostgreSQL support for production deployments.
- Enhanced analytics: patient-wise stats, monthly summaries.
- Audit logging for CRUD events.

## 10. Conclusion

The backend is **secure, feature-complete, and ready for frontend integration**. It supports authentication, full prescription CRUD, reporting, validation, and history tracking. With its modular structure, it is easy to extend with additional features like Swagger docs, role-based access, and production-ready databases.