Smart Healthcare Appointment & Prescription System — Full Source & Build Guide

A Java + Spring Boot backend (with simple Thymeleaf frontend hints) for booking appointments and issuing prescriptions. This document contains a runnable project skeleton, source code for core components, and step-by-step build/run instructions.

Table of contents

---

1) Project overview

This project provides: - User registration and login with JWT-based authentication (roles: PATIENT, DOCTOR, ADMIN) - Doctor profile management - Booking appointments (patients can book doctor time slots) - Prescription creation (doctors create prescriptions tied to appointments) - Generate prescription as PDF (download endpoint)

The document focuses on a production-oriented backend implemented in Java + Spring Boot. A minimal Thymeleaf or React frontend can be added; instructions below show how to test using Postman or curl.

---

2) Tech stack & prerequisites

- Java 17 (or 11 if you prefer; repo uses Java 17 features)

- Maven (3.6+)
- Spring Boot (2.7.x recommended for compatibility with examples below)
- Optional: Docker & Docker Compose (to run as container)
- Database: H2 (embedded) by default for quick testing. Swap to MySQL/Postgres by changing properties.

Installed locally: - JDK 17 - Maven

---

3) Project structure (important files)

```
smart-healthcare/
├─ pom.xml
└─ src/
   └─ main/
      ├─ java/com/example/healthcare/
      │  ├─ HealthcareApplication.java
      │  ├─ config/
      │  │  ├─ SecurityConfig.java
      │  │  ├─ JwtAuthenticationFilter.java
      │  │  └─ JwtTokenUtil.java
      │  ├─ controller/
      │  │  ├─ AuthController.java
      │  │  ├─ AppointmentController.java
      │  │  └─ PrescriptionController.java
      │  ├─ dto/
      │  │  ├─ LoginRequest.java
      │  │  ├─ LoginResponse.java
      │  │  └─ RegisterRequest.java
      │  ├─ model/
      │  │  ├─ User.java
      │  │  ├─ Role.java
      │  │  ├─ Doctor.java
      │  │  ├─ Patient.java
      │  │  ├─ Appointment.java
      │  │  ├─ AppointmentStatus.java
      │  │  └─ Prescription.java
      │  ├─ repository/
      │  │  ├─ UserRepository.java
      │  │  ├─ DoctorRepository.java
      │  │  ├─ PatientRepository.java
      │  │  ├─ AppointmentRepository.java
      │  │  └─ PrescriptionRepository.java
      │  └─ service/
      │     ├─ UserService.java
```

```
        │     ├─ AppointmentService.java
        │     └─ PrescriptionService.java
        └─ resources/
           ├─ application.properties
           └─ data.sql (optional seed)
```

---

4) pom.xml

`pom.xml` contains the dependencies for Spring Boot web, JPA, security, JWT, PDFBox (for generating PDFs), and H2.

```
<!-- File: pom.xml -->
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>smart-healthcare</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.12</version>
        <relativePath/>
    </parent>

    <properties>
        <java.version>17</java.version>
    </properties>

    <dependencies>
        <!-- Web -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Data / JPA -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
```

```xml
        </dependency>

        <!-- Security -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>

        <!-- JWT (simple version) -->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>

        <!-- PDF generation -->
        <dependency>
            <groupId>org.apache.pdfbox</groupId>
            <artifactId>pdfbox</artifactId>
            <version>2.0.26</version>
        </dependency>

        <!-- H2 for local testing -->
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>

        <!-- MySQL (comment/uncomment if you switch to MySQL) -->
        <!--
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        -->

        <!-- Thymeleaf (optional minimal server-rendered views) -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>

        <!-- For tests -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

---

5) application.properties (important bits)

Save under `src/main/resources/application.properties`.

```
# Server
server.port=8080

# H2 (embedded DB)
spring.datasource.url=jdbc:h2:mem:healthdb;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driverClassName=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update

# JWT settings (change secret in production)
app.jwtSecret=VerySecretKeyThatYouShouldChangeToASecureRandomValue
app.jwtExpirationMs=86400000

# Show SQL for debugging
spring.jpa.show-sql=true

# Thymeleaf cache disabled in dev
spring.thymeleaf.cache=false
```

> To switch to MySQL, replace `spring.datasource.url` and add `mysql-connector-java` dependency.

---

6) Source code (core files)

All Java source files are under `src/main/java/com/example/healthcare/`.

6.1 Main application

```java
// File: HealthcareApplication.java
package com.example.healthcare;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HealthcareApplication {
    public static void main(String[] args) {
        SpringApplication.run(HealthcareApplication.class, args);
    }
}
```

6.2 Models / Entities

```java
// File: model/Role.java
package com.example.healthcare.model;

public enum Role {
    ROLE_PATIENT,
    ROLE_DOCTOR,
    ROLE_ADMIN
}
```

```java
// File: model/User.java
package com.example.healthcare.model;

import javax.persistence.*;

@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(unique = true, nullable = false)
    private String email;

    @Column(nullable = false)
    private String password; // stored as bcrypt-hash

    @Enumerated(EnumType.STRING)
    private Role role;

    // getters and setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
    public Role getRole() { return role; }
    public void setRole(Role role) { this.role = role; }
}
```

```java
// File: model/Doctor.java
package com.example.healthcare.model;

import javax.persistence.*;
import java.math.BigDecimal;

@Entity
public class Doctor {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne(fetch = FetchType.EAGER)
    private User user;

    private String specialty;

    private BigDecimal fee;

    // getters/setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public User getUser() { return user; }
    public void setUser(User user) { this.user = user; }
    public String getSpecialty() { return specialty; }
    public void setSpecialty(String specialty) { this.specialty = specialty; }
    public BigDecimal getFee() { return fee; }
    public void setFee(BigDecimal fee) { this.fee = fee; }
}
```

```java
// File: model/Patient.java
package com.example.healthcare.model;

import javax.persistence.*;

@Entity
public class Patient {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne(fetch = FetchType.EAGER)
    private User user;

    private String phone;

    // getters/setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public User getUser() { return user; }
    public void setUser(User user) { this.user = user; }
    public String getPhone() { return phone; }
    public void setPhone(String phone) { this.phone = phone; }
}
```

```java
// File: model/AppointmentStatus.java
package com.example.healthcare.model;

public enum AppointmentStatus {
    PENDING, CONFIRMED, CANCELLED, COMPLETED
}
```

```java
// File: model/Appointment.java
package com.example.healthcare.model;

import javax.persistence.*;
import java.time.LocalDateTime;

@Entity
public class Appointment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.EAGER)
    private Doctor doctor;

    @ManyToOne(fetch = FetchType.EAGER)
    private Patient patient;

    private LocalDateTime appointmentTime;

    @Enumerated(EnumType.STRING)
    private AppointmentStatus status;

    // getters/setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public Doctor getDoctor() { return doctor; }
    public void setDoctor(Doctor doctor) { this.doctor = doctor; }
    public Patient getPatient() { return patient; }
    public void setPatient(Patient patient) { this.patient = patient; }
    public LocalDateTime getAppointmentTime() { return appointmentTime; }
    public void setAppointmentTime(LocalDateTime appointmentTime) {
this.appointmentTime = appointmentTime; }
    public AppointmentStatus getStatus() { return status; }
    public void setStatus(AppointmentStatus status) { this.status = status; }
}
```

```
// File: model/Prescription.java
package com.example.healthcare.model;

import javax.persistence.*;
import java.time.LocalDateTime;

@Entity
public class Prescription {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne(fetch = FetchType.EAGER)
    private Appointment appointment;

    @Lob
    private String content; // textual prescription: medicines, notes

    private LocalDateTime createdAt;

    // getters/setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public Appointment getAppointment() { return appointment; }
    public void setAppointment(Appointment appointment) { this.appointment =
appointment; }
    public String getContent() { return content; }
    public void setContent(String content) { this.content = content; }
    public LocalDateTime getCreatedAt() { return createdAt; }
    public void setCreatedAt(LocalDateTime createdAt) { this.createdAt =
createdAt; }
}
```

6.3 Repositories

```
// File: repository/UserRepository.java
package com.example.healthcare.repository;

import com.example.healthcare.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
    boolean existsByUsername(String username);
    boolean existsByEmail(String email);
}
```

```
// File: repository/DoctorRepository.java
package com.example.healthcare.repository;

import com.example.healthcare.model.Doctor;
import org.springframework.data.jpa.repository.JpaRepository;

public interface DoctorRepository extends JpaRepository<Doctor, Long> {
}
```

```
// File: repository/PatientRepository.java
package com.example.healthcare.repository;

import com.example.healthcare.model.Patient;
import org.springframework.data.jpa.repository.JpaRepository;

public interface PatientRepository extends JpaRepository<Patient, Long> {
}
```

```
// File: repository/AppointmentRepository.java
package com.example.healthcare.repository;

import com.example.healthcare.model.Appointment;
import org.springframework.data.jpa.repository.JpaRepository;
import java.time.LocalDateTime;
import java.util.List;

public interface AppointmentRepository extends JpaRepository<Appointment, Long>
{
    List<Appointment> findByDoctorIdAndAppointmentTimeBetween(Long doctorId,
LocalDateTime start, LocalDateTime end);
    List<Appointment> findByPatientId(Long patientId);
}
```

```
// File: repository/PrescriptionRepository.java
package com.example.healthcare.repository;

import com.example.healthcare.model.Prescription;
import org.springframework.data.jpa.repository.JpaRepository;

public interface PrescriptionRepository extends JpaRepository<Prescription,
Long> {
}
```

6.4 DTOs (auth)

```
// File: dto/RegisterRequest.java
package com.example.healthcare.dto;

public class RegisterRequest {
    private String username;
    private String email;
    private String password;
    private String role; // PATIENT/DOCTOR

    // getters/setters
    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }
}
```

```
// File: dto/LoginRequest.java
package com.example.healthcare.dto;

public class LoginRequest {
    private String username;
    private String password;
    // getters/setters
    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
}
```

```
// File: dto/LoginResponse.java
package com.example.healthcare.dto;

public class LoginResponse {
    private String token;
    public LoginResponse(String token) { this.token = token; }
    public String getToken() { return token; }
}
```

6.5 Security (JWT utility + filter + config)

```
// File: config/JwtTokenUtil.java
package com.example.healthcare.config;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import java.util.Date;

@Component
public class JwtTokenUtil {

    @Value("${app.jwtSecret}")
    private String jwtSecret;

    @Value("${app.jwtExpirationMs}")
    private int jwtExpirationMs;

    public String generateToken(String username) {
        Date now = new Date();
        Date expiry = new Date(now.getTime() + jwtExpirationMs);
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(now)
                .setExpiration(expiry)
                .signWith(SignatureAlgorithm.HS512, jwtSecret)
                .compact();
    }

    public String getUsernameFromToken(String token) {
        Claims claims =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody();
        return claims.getSubject();
    }

    public boolean validateToken(String token) {
        try {
            Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token);
            return true;
        } catch (Exception ex) {
            return false;
        }
    }
}
```

```
// File: config/JwtAuthenticationFilter.java
package com.example.healthcare.config;

import org.springframework.beans.factory.annotation.Autowired;
```

```
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws
ServletException, IOException {
        String authHeader = request.getHeader("Authorization");
        String token = null;
        String username = null;
        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            token = authHeader.substring(7);
            try {
                username = jwtTokenUtil.getUsernameFromToken(token);
            } catch (Exception e) {
                // invalid token
            }
        }

        if (username != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails =
userDetailsService.loadUserByUsername(username);
            if (jwtTokenUtil.validateToken(token)) {
                UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities());
                authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authentication);
            }
        }

        filterChain.doFilter(request, response);
    }
}
```

```
// File: config/SecurityConfig.java
package com.example.healthcare.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuild
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    private JwtAuthenticationFilter jwtAuthenticationFilter;
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authManager(HttpSecurity http) throws
Exception {
        AuthenticationManagerBuilder authBuilder =
http.getSharedObject(AuthenticationManagerBuilder.class);

authBuilder.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
        return authBuilder.build();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .authorizeRequests()
              .antMatchers("/api/auth/**", "/h2-console/**").permitAll()
              .anyRequest().authenticated();

        // H2 console frames
        http.headers().frameOptions().disable();

        http.addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}
```

Note: We used Spring Security Java config with a `SecurityFilterChain` bean (modern style). The `UserDetailsService` is implemented below in `UserService`.

6.6 Services

```
// File: service/UserService.java
package com.example.healthcare.service;

import com.example.healthcare.model.Role;
import com.example.healthcare.model.User;
import com.example.healthcare.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.Collections;

@Service
public class UserService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Transactional
    public User register(User user) {
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        if (user.getRole() == null) user.setRole(Role.ROLE_PATIENT);
        return userRepository.save(user);
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not
found: " + username));
        GrantedAuthority authority = new
SimpleGrantedAuthority(user.getRole().name());
        return new
org.springframework.security.core.userdetails.User(user.getUsername(),
user.getPassword(), Collections.singleton(authority));
    }

    public User findByUsername(String username) {
        return userRepository.findByUsername(username).orElse(null);
    }
}
```

```java
// File: service/AppointmentService.java
package com.example.healthcare.service;

import com.example.healthcare.model.*;
import com.example.healthcare.repository.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.time.LocalDateTime;
import java.util.List;

@Service
public class AppointmentService {

    @Autowired
    private AppointmentRepository appointmentRepository;

    @Autowired
    private DoctorRepository doctorRepository;

    @Autowired
    private PatientRepository patientRepository;

    public List<Appointment> findAppointmentsForDoctorBetween(Long doctorId,
LocalDateTime start, LocalDateTime end) {
        return
appointmentRepository.findByDoctorIdAndAppointmentTimeBetween(doctorId, start,
end);
    }

    @Transactional
    public Appointment bookAppointment(Long doctorId, Long patientId,
LocalDateTime when) {
        Doctor doctor = doctorRepository.findById(doctorId).orElseThrow(() ->
new IllegalArgumentException("Doctor not found"));
        Patient patient = patientRepository.findById(patientId).orElseThrow(() -
> new IllegalArgumentException("Patient not found"));

        Appointment appt = new Appointment();
        appt.setDoctor(doctor);
        appt.setPatient(patient);
        appt.setAppointmentTime(when);
        appt.setStatus(AppointmentStatus.PENDING);

        return appointmentRepository.save(appt);
    }
    public List<Appointment> findByPatientId(Long patientId) {
        return appointmentRepository.findByPatientId(patientId);
    }
}
```

```java
// File: service/PrescriptionService.java
package com.example.healthcare.service;

import com.example.healthcare.model.Prescription;
import com.example.healthcare.model.Appointment;
import com.example.healthcare.repository.PrescriptionRepository;
import com.example.healthcare.repository.AppointmentRepository;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.pdmodel.PDPage;
import org.apache.pdfbox.pdmodel.PDPageContentStream;
import org.apache.pdfbox.pdmodel.font.PDType1Font;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.time.LocalDateTime;

@Service
public class PrescriptionService {

    @Autowired
    private PrescriptionRepository prescriptionRepository;

    @Autowired
    private AppointmentRepository appointmentRepository;

    @Transactional
    public Prescription createPrescription(Long appointmentId, String content) {
        Appointment appt =
appointmentRepository.findById(appointmentId).orElseThrow(() -> new
IllegalArgumentException("Appointment not found"));
        Prescription p = new Prescription();
        p.setAppointment(appt);
        p.setContent(content);
        p.setCreatedAt(LocalDateTime.now());
        return prescriptionRepository.save(p);
    }
    public byte[] generatePrescriptionPdf(Long prescriptionId) throws
IOException {
        Prescription p =
prescriptionRepository.findById(prescriptionId).orElseThrow(() -> new
IllegalArgumentException("Prescription not found"));
        Appointment appt = p.getAppointment();

        try (PDDocument doc = new PDDocument(); ByteArrayOutputStream baos =
new ByteArrayOutputStream()) {
            PDPage page = new PDPage();
            doc.addPage(page);

            PDPageContentStream cs = new PDPageContentStream(doc, page);
            cs.beginText();
            cs.setFont(PDType1Font.HELVETICA_BOLD, 16);
            cs.newLineAtOffset(50, 700);
            cs.showText("Prescription");
            cs.newLineAtOffset(0, -25);
            cs.setFont(PDType1Font.HELVETICA, 12);
            cs.showText("Date: " + p.getCreatedAt().toLocalDate().toString());
            cs.newLineAtOffset(0, -20);

            // doctor/patient info
            String docName = appt.getDoctor().getUser().getUsername();
            String patientName = appt.getPatient().getUser().getUsername();
            cs.showText("Doctor: " + docName);
            cs.newLineAtOffset(0, -20);
            cs.showText("Patient: " + patientName);
            cs.newLineAtOffset(0, -20);
            cs.showText("Appointment: " + appt.getAppointmentTime().toString());
            cs.newLineAtOffset(0, -30);

            // prescription content (wrap rudimentary)
            String[] lines = p.getContent().split("\\n");
            for (String line : lines) {
                cs.showText(line);
                cs.newLineAtOffset(0, -15);
            }

            cs.endText();
            cs.close();

            doc.save(baos);
            return baos.toByteArray();
        }
    }
}
```

6.7 Controllers

```java
// File: controller/AuthController.java
package com.example.healthcare.controller;

import com.example.healthcare.config.JwtTokenUtil;
import com.example.healthcare.dto.LoginRequest;
import com.example.healthcare.dto.LoginResponse;
import com.example.healthcare.dto.RegisterRequest;
import com.example.healthcare.model.Role;
import com.example.healthcare.model.User;
import com.example.healthcare.repository.PatientRepository;
import com.example.healthcare.repository.DoctorRepository;
import com.example.healthcare.model.Doctor;
import com.example.healthcare.model.Patient;
import com.example.healthcare.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Autowired
    private UserService userService;

    @Autowired
    private PatientRepository patientRepository;

    @Autowired
    private DoctorRepository doctorRepository;

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest req) {
        Authentication auth = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(req.getUsername(), req.getPassword()));
        SecurityContextHolder.getContext().setAuthentication(auth);
        String token = jwtTokenUtil.generateToken(req.getUsername());
        return ResponseEntity.ok(new LoginResponse(token));
    }

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody RegisterRequest req) {
        if (userService.findByUsername(req.getUsername()) != null) {
            return ResponseEntity.badRequest().body("Username already taken");
        }

        User u = new User();
        u.setUsername(req.getUsername());
        u.setEmail(req.getEmail());
        u.setPassword(req.getPassword());
        if ("DOCTOR".equalsIgnoreCase(req.getRole()))
u.setRole(Role.ROLE_DOCTOR);
        else u.setRole(Role.ROLE_PATIENT);

        User saved = userService.register(u);

        if (saved.getRole() == Role.ROLE_PATIENT) {
            Patient p = new Patient();
            p.setUser(saved);
            patientRepository.save(p);
        } else if (saved.getRole() == Role.ROLE_DOCTOR) {
            Doctor d = new Doctor();
            d.setUser(saved);
            doctorRepository.save(d);
        }

        return ResponseEntity.ok("Registered");
    }
}
```

```
// File: controller/AppointmentController.java
package com.example.healthcare.controller;

import com.example.healthcare.model.Appointment;
import com.example.healthcare.service.AppointmentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
```

```java
import java.time.LocalDateTime;
import java.util.List;

@RestController
@RequestMapping("/api/appointments")
public class AppointmentController {

    @Autowired
    private AppointmentService appointmentService;

    @GetMapping("/doctor/{doctorId}/between")
    public ResponseEntity<?> findBetween(@PathVariable Long doctorId,
                                         @RequestParam("start")
@DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME) LocalDateTime start,
                                         @RequestParam("end")
@DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME) LocalDateTime end) {
        List<Appointment> list =
appointmentService.findAppointmentsForDoctorBetween(doctorId, start, end);
        return ResponseEntity.ok(list);
    }

    @PostMapping("/book")
    public ResponseEntity<?> book(@RequestParam Long doctorId, @RequestParam
Long patientId,
                                  @RequestParam @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE_TIME) LocalDateTime when) {
        Appointment a = appointmentService.bookAppointment(doctorId, patientId,
when);
        return ResponseEntity.ok(a);
    }

}
```

```java
// File: controller/PrescriptionController.java
package com.example.healthcare.controller;

import com.example.healthcare.model.Prescription;
import com.example.healthcare.service.PrescriptionService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.io.IOException;
```

```
@RestController
@RequestMapping("/api/prescriptions")
public class PrescriptionController {

    @Autowired
    private PrescriptionService prescriptionService;

    @PostMapping("/create")
    public ResponseEntity<?> create(@RequestParam Long appointmentId,
@RequestBody String content) {
        Prescription p = prescriptionService.createPrescription(appointmentId,
content);
        return ResponseEntity.ok(p);
    }

    @GetMapping(value = "/{id}/pdf")
    public ResponseEntity<byte[]> downloadPdf(@PathVariable Long id) throws
IOException {
        byte[] pdf = prescriptionService.generatePrescriptionPdf(id);
        return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=prescription_" + id + ".pdf")
                .contentType(MediaType.APPLICATION_PDF)
                .body(pdf);
    }
}
```

---

7) How to build & run (local)

Steps (quick): 1. Make sure JDK 17 and Maven are installed. 2. Create the project folder with the file tree above (or copy files into an IDE like IntelliJ). 3. In project root run: `mvn clean package`. 4. Run: - via Maven: `mvn spring-boot:run` - or run jar: `java -jar target/smart-healthcare-0.0.1-SNAPSHOT.jar` 5. The application starts on `http://localhost:8080`.

H2 console: `http://localhost:8080/h2-console` (JDBC URL `jdbc:h2:mem:healthdb`) — useful to inspect data.

Test flow (Postman/curl): 1. Register a doctor:

```
curl -X POST http://localhost:8080/api/auth/register -H "Content-Type:
application/json" -d
'{"username":"drjones","email":"dr@x.com","password":"pass123","role":"DOCTOR"}'
```

2. Register a patient:

```
curl -X POST http://localhost:8080/api/auth/register -H "Content-Type: 
application/json" -d 
'{"username":"alice","email":"alice@x.com","password":"pass123","role":"PATIENT"}'
```

3. Login patient to get token:

```
curl -X POST http://localhost:8080/api/auth/login -H "Content-Type: application/
json" -d '{"username":"alice","password":"pass123"}'
```

The response body contains JWT token. 4. Book appointment (use token in Authorization header `Bearer <token>` if endpoints are secured by roles in your extension). For now the `book` endpoint uses patientId and doctorId directly.

```
curl -X POST "http://localhost:8080/api/appointments/book?
doctorId=1&patientId=2&when=2025-09-20T10:30:00"
```

5. Create prescription (as doctor) then download PDF:

```
curl -X POST "http://localhost:8080/api/prescriptions/create?appointmentId=1" -
H "Content-Type: text/plain" -d "Take 1 pill X twice daily for 5 days."
curl -X GET http://localhost:8080/api/prescriptions/1/pdf --output 
prescription_1.pdf
```

---

8) Dockerfile (optional)

Create `Dockerfile` to build container image.

```
FROM eclipse-temurin:17-jdk-jammy
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

Build:

```
mvn package -DskipTests
docker build -t smart-healthcare:latest .
```

Run:

```
docker run -p 8080:8080 smart-healthcare:latest
```

---

9) How to test with Postman (sample collection)

- POST `/api/auth/register` with JSON body `{username,email,password,role}`
- POST `/api/auth/login` with JSON body `{username,password}` -> get `token`
- POST `/api/appointments/book?doctorId=1&patientId=2&when=2025-09-20T10:30:00`
- POST `/api/prescriptions/create?appointmentId=1` with raw text body
- GET `/api/prescriptions/1/pdf` download

  Tip: Add header `Authorization: Bearer <token>` for endpoints you secure. In the example above, some endpoints are permitted to simplify testing — harden them before production.

---

10) Production & extension ideas (to make project resume-grade)

To increase the project's value on your resume, implement these extras: - Use a frontend (React) with a polished UI, appointment calendar, and live validation. - Replace H2 with MySQL/Postgres and add Flyway migrations. - Add role-based authorization on endpoints (doctors can only create prescriptions for their appointments). - Integrate payment gateway for paid consultations (Stripe/PayPal). - Add email/SMS notifications using Spring Mail / Twilio. - Add appointment scheduling logic & conflict checks. - Add unit/integration tests (JUnit + Mockito + SpringBootTest). - Containerize and deploy to AWS ECS/EKS or Heroku. Use CI (GitHub Actions) to run tests and build images. - Add analytics dashboard (charts) using charting library.

Final notes

- This document provides a runnable backend skeleton. Some parts (like production-grade validation, user input sanitation, role authorization, and advanced scheduling logic) are intentionally concise to keep the template readable. Ask me and I will expand any file or add a React frontend and provide a downloadable zip/jar.

If you'd like, I can now: - generate a downloadable ZIP of the project files, - create a React frontend scaffold that connects to these APIs, - or convert the example to use MySQL and Flyway and provide Docker Compose for a full-stack dev environment.

Tell me which of those you'd like and I will produce it next.