```
package com.hms.model;

import jakarta.persistence.*;
import java.time.LocalDate;

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

    private LocalDate date;
    private String timeSlot;

    @ManyToOne
    private Doctor doctor;

    @ManyToOne
    private Patient patient;

    @Enumerated(EnumType.STRING)
    private AppointmentStatus status;

    public Appointment() {}

    public Appointment(LocalDate date, String timeSlot, Doctor doctor, Patient patient, AppointmentStatus status) {
        this.date = date;
        this.timeSlot = timeSlot;
        this.doctor = doctor;
        this.patient = patient;
        this.status = status;
    }

        public Long getId() {
                return id;
        }

        public void setId(Long id) {
                this.id = id;
        }

        public LocalDate getDate() {
                return date;
        }

        public void setDate(LocalDate date) {
                this.date = date;
        }

        public String getTimeSlot() {
                return timeSlot;
        }

        public void setTimeSlot(String timeSlot) {
                this.timeSlot = timeSlot;
        }

        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 AppointmentStatus getStatus() {
                return status;
        }

        public void setStatus(AppointmentStatus status) {
                this.status = status;
        }

}




package com.hms.model;

public enum AppointmentStatus {
    BOOKED,
    CANCELLED
}




package com.hms.model;
```

```
import jakarta.persistence.*;
import java.time.LocalDate;

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

    private Double amount;
    private LocalDate billDate;

    @OneToOne
    private Appointment appointment;

    public Bill() {}

    public Bill(Double amount, LocalDate billDate, Appointment appointment) {
        this.amount = amount;
        this.billDate = billDate;
        this.appointment = appointment;
    }

        public Long getId() {
                return id;
        }

        public void setId(Long id) {
                this.id = id;
        }

        public Double getAmount() {
                return amount;
        }

        public void setAmount(Double amount) {
                this.amount = amount;
        }

        public LocalDate getBillDate() {
                return billDate;
        }

        public void setBillDate(LocalDate billDate) {
                this.billDate = billDate;
        }

        public Appointment getAppointment() {
                return appointment;
        }
        public void setAppointment(Appointment appointment) {
                this.appointment = appointment;
        }

}





package com.hms.model;

import jakarta.persistence.*;

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

    private String name;
    private String specialization;

    @OneToOne
    private User user;

    public Doctor() {}

    public Doctor(String name, String specialization, User user) {
        this.name = name;
        this.specialization = specialization;
        this.user = user;
    }

        public Long getId() {
                return id;
        }

        public void setId(Long id) {
                this.id = id;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }
        public String getSpecialization() {
                return specialization;
        }

        public void setSpecialization(String specialization) {
                this.specialization = specialization;
        }

        public User getUser() {
                return user;
        }

        public void setUser(User user) {
                this.user = user;
        }

}





package com.hms.model;

import jakarta.persistence.*;

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

    private String name;
    private String contact;

    @OneToOne
    private User user;

    public Patient() {}

    public Patient(String name, String contact, User user) {
        this.name = name;
        this.contact = contact;
        this.user = user;
    }

        public Long getId() {
                return id;
        }
        public void setId(Long id) {
                this.id = id;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public String getContact() {
                return contact;
        }

        public void setContact(String contact) {
                this.contact = contact;
        }

        public User getUser() {
                return user;
        }

        public void setUser(User user) {
                this.user = user;
        }

}
```

```
package com.hms.model;

public enum Role {
    ADMIN,
    DOCTOR,
    PATIENT
}
```

```
package com.hms.model;

import jakarta.persistence.*;

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

    private String username;
    private String password;

    @Enumerated(EnumType.STRING)
    private Role role;

    public User() {}

    public User(String username, String password, Role role) {
        this.username = username;
        this.password = password;
        this.role = role;
    }

        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 getPassword() {
                return password;
        }

        public void setPassword(String password) {
                this.password = password;
        }

        public Role getRole() {
                return role;
        }

        public void setRole(Role role) {
                this.role = role;
        }

}


==================================================================================


package com.hms.dto;

import java.time.LocalDate;

public class AppointmentRequest {
    private Long doctorId;
    private Long patientId;
    private LocalDate date;
    private String timeSlot;

    public AppointmentRequest() {}

    public AppointmentRequest(Long doctorId, Long patientId, LocalDate date, String timeSlot) {
        this.doctorId = doctorId;
        this.patientId = patientId;
        this.date = date;
        this.timeSlot = timeSlot;
    }

        public Long getDoctorId() {
                return doctorId;
        }

        public void setDoctorId(Long doctorId) {
                this.doctorId = doctorId;
        }

        public Long getPatientId() {
                return patientId;
        }

        public void setPatientId(Long patientId) {
                this.patientId = patientId;
        }

        public LocalDate getDate() {
                return date;
        }
        public void setDate(LocalDate date) {
                this.date = date;
        }

        public String getTimeSlot() {
                return timeSlot;
        }

        public void setTimeSlot(String timeSlot) {
                this.timeSlot = timeSlot;
        }

}




package com.hms.dto;

public class BillRequest {
    private Long appointmentId;
    private Double amount;

    public BillRequest() {}

    public BillRequest(Long appointmentId, Double amount) {
        this.appointmentId = appointmentId;
        this.amount = amount;
    }

        public Long getAppointmentId() {
                return appointmentId;
        }

        public void setAppointmentId(Long appointmentId) {
                this.appointmentId = appointmentId;
        }

        public Double getAmount() {
                return amount;
        }

        public void setAmount(Double amount) {
                this.amount = amount;
        }

}
```

```
package com.hms.dto;

public class RegisterDoctorRequest {
    private String username;
    private String password;
    private String name;
    private String specialization;

    public RegisterDoctorRequest() {}

    public RegisterDoctorRequest(String username, String password, String name, String specialization) {
        this.username = username;
        this.password = password;
        this.name = name;
        this.specialization = specialization;
    }

        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;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public String getSpecialization() {
                return specialization;
        }

        public void setSpecialization(String specialization) {
                this.specialization = specialization;
        }
```

```
}







package com.hms.dto;

public class RegisterPatientRequest {
    private String username;
    private String password;
    private String name;
    private String contact;

    public RegisterPatientRequest() {}

    public RegisterPatientRequest(String username, String password, String name, String
contact) {
        this.username = username;
        this.password = password;
        this.name = name;
        this.contact = contact;
    }

        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;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }
        public String getContact() {
                return contact;
        }

        public void setContact(String contact) {
                this.contact = contact;
        }

}





=======================================================




package com.hms.config;

import io.swagger.v3.oas.models.info.Contact;
import io.swagger.v3.oas.models.info.License;
import io.swagger.v3.oas.models.info.Info;
import io.swagger.v3.oas.models.OpenAPI;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SwaggerConfig {

    @Bean
    public OpenAPI hmsOpenAPI() {
        return new OpenAPI()
            .info(new Info()
                .title("Hospital Management System API")
                .description("Simple, role-based REST APIs for Admin, Doctor, Patient, Appointments
and Billing")
                .version("1.0.0")
                .contact(new Contact()
                    .name("HMS Support")
                    .email("support@example.com"))
                .license(new License()
                    .name("MIT")
                    .url("https://opensource.org/licenses/MIT"))
            );
    }
}
```

```
package com.hms.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.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.Customizer;
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.config.http.SessionCreationPolicy;

import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.util.List;

@Configuration
public class SecurityConfig {

    // Will be provided in Section 7 as UserDetailsServiceImpl
    @Autowired
    private UserDetailsService userDetailsService;

    @Bean
    public PasswordEncoder passwordEncoder() {
        // BCrypt for hashing passwords
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setPasswordEncoder(passwordEncoder());
        provider.setUserDetailsService(userDetailsService);
        return provider;
    }
    // Expose AuthenticationManager for AuthController (authenticationManager.authenticate(...))
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            // REST APIs: disable CSRF and use stateless sessions
            .csrf(csrf -> csrf.disable())
            .cors(Customizer.withDefaults())
            .sessionManagement(sm -> sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

            // Authorization rules
            .authorizeHttpRequests(auth -> auth
                // Public endpoints
                .requestMatchers(
                    "/api/auth/**",                   // login
                    "/api/user/register-patient",     // patient self-registration
                    "/v3/api-docs/**",                // OpenAPI JSON
                    "/swagger-ui/**",                 // Swagger UI resources
                    "/swagger-ui.html"                // Swagger UI entry
                ).permitAll()

                // Admin-only endpoints
                .requestMatchers("/api/admin/**").hasRole("ADMIN")

                // Everything else needs authentication
                .anyRequest().authenticated()
            )

            // Simple for Postman/testing
            .httpBasic(Customizer.withDefaults());

        return http.build();
    }

    /**
     * CORS for local dev (VS Code Live Server and typical local ports)
     * Adjust or tighten for prod as needed.
     */
    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration cfg = new CorsConfiguration();
        cfg.setAllowedOrigins(List.of(
            "http://127.0.0.1:5500",
            "http://localhost:5500",
            "http://localhost:3000",
            "http://localhost"
        ));
        cfg.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS"));
        cfg.setAllowedHeaders(List.of("Authorization", "Content-Type"));
        cfg.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", cfg);
        return source;
    }
}




=================================================================


package com.hms.exception;

import java.time.LocalDateTime;

public class ApiError {
    private LocalDateTime timestamp;
    private int status;
    private String error;
    private String message;

    public ApiError(int status, String error, String message) {
        this.timestamp = LocalDateTime.now();
        this.status = status;
        this.error = error;
        this.message = message;
    }

        public LocalDateTime getTimestamp() {
                return timestamp;
        }

        public int getStatus() {
                return status;
        }

        public String getError() {
                return error;
        }

        public String getMessage() {
                return message;
        }

}
```

```
package com.hms.exception;

public class BadRequestException extends RuntimeException {
    public BadRequestException(String message) {
        super(message);
    }
}
```

```
package com.hms.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ApiError> handleNotFound(ResourceNotFoundException ex) {
        ApiError error = new ApiError(HttpStatus.NOT_FOUND.value(), "Not Found",
ex.getMessage());
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(error);
    }

    @ExceptionHandler(BadRequestException.class)
    public ResponseEntity<ApiError> handleBadRequest(BadRequestException ex) {
        ApiError error = new ApiError(HttpStatus.BAD_REQUEST.value(), "Bad Request",
ex.getMessage());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(error);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ApiError> handleGeneric(Exception ex) {
        ApiError error = new ApiError(HttpStatus.INTERNAL_SERVER_ERROR.value(), "Error",
ex.getMessage());
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(error);
    }
}
```

```
package com.hms.exception;

public class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException(String message) {
        super(message);
    }
} this is my half project
```

Perfect 👍 Samiksha — you've completed **half of your Hospital Management System (HMS)** project quite solidly!

Here's a quick **review and roadmap** of what you've already built and what's left to finish, based on your provided code 👇

✅ What's Already Done (Completed Modules)

🏗️ 1. Model Layer

You've correctly created JPA entities with relationships:

- **User**, **Doctor**, **Patient**, **Appointment**, **Bill**
- Enum classes for **Role** and **AppointmentStatus**
  ✅ Uses `@Entity`, `@OneToOne`, `@ManyToOne`, and `@Enumerated` properly.
  ✅ Perfectly structured for relational mapping.

📦 2. DTO Layer

You've cleanly separated input data using DTOs:

- `RegisterDoctorRequest`, `RegisterPatientRequest`, `AppointmentRequest`, `BillRequest`
  ✅ Makes API endpoints cleaner and more secure.

⚙️ 3. Configuration Layer

- `SwaggerConfig`: For API documentation (✅ nice addition)
- `SecurityConfig`: For authentication and authorization (✅ roles handled properly)

🚨 4. Exception Handling

- Centralized **`GlobalExceptionHandler`** with custom exceptions like **`BadRequestException`** and **`ResourceNotFoundException`**.
  ✅ This will make API error responses more meaningful.

🚧 What's Left (Remaining Modules)

Here's what you should add next — we'll build this step-by-step:

1️⃣ Repository Layer

You'll need interfaces for database operations:

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

public interface DoctorRepository extends JpaRepository<Doctor, Long> {}
public interface PatientRepository extends JpaRepository<Patient, Long> {}
public interface AppointmentRepository extends JpaRepository<Appointment, Long> {}
public interface BillRepository extends JpaRepository<Bill, Long> {}
```

2️⃣ Service Layer

Here we'll implement business logic for:

- Registering doctors and patients
- Booking and cancelling appointments
- Generating bills
- Fetching details by admin/doctor/patient

We'll make services like:

- **`UserService`**,
- **`DoctorService`**,
- **`PatientService`**,
- **`AppointmentService`**,
- **`BillService`**.

3️⃣ Controller Layer

You'll expose REST endpoints for:

- **`/api/auth`** → login
- **`/api/user`** → registration
- **`/api/admin`** → admin management

- `/api/doctor` → doctor operations
- `/api/patient` → patient operations
- `/api/appointment` and `/api/bill` → appointment and billing management

4️⃣ Security - UserDetailsServiceImpl

This connects your `User` entity to Spring Security authentication:

```java
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired
    private UserRepository userRepository;

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

5️⃣ Authentication Controller

Handles user login (using `AuthenticationManager`) and registration for patients.

6️⃣ Database Configuration

Make sure your `application.properties` or `application.yml` includes:

```properties
spring.datasource.url=jdbc:mysql://localhost:3306/hospital_db
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
springdoc.swagger-ui.path=/swagger-ui.html
```

🎯 Next Step Suggestion

Let’s proceed **module by module** from here.
Since your **Model, DTO, Config, and Exception layers** are done, the logical next step is:

👉 **Step 1:** Create the **Repository Layer**.

Would you like me to give you the full ready-to-use **repository package code (com.hms.repository)** next?

It’ll be short and clean — all interfaces using Spring Data JPA.

```
package com.hms.controller;

import com.hms.dto.RegisterDoctorRequest;
import com.hms.model.Doctor;
import com.hms.service.DoctorService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/admin")
public class AdminController {

    @Autowired
    private DoctorService doctorService;

    @PostMapping("/register-doctor")
    public ResponseEntity<Doctor> registerDoctor(@RequestBody RegisterDoctorRequest
request) {
        Doctor doctor = doctorService.registerDoctor(request);
        return ResponseEntity.ok(doctor);
    }
}







package com.hms.controller;

import com.hms.dto.AppointmentRequest;
import com.hms.model.Appointment;
import com.hms.service.AppointmentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

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

    @Autowired
    private AppointmentService appointmentService;
    @PostMapping("/book")
    public ResponseEntity<Appointment> bookAppointment(@RequestBody AppointmentRequest request) {
        Appointment appointment = appointmentService.bookAppointment(request);
        return ResponseEntity.ok(appointment);
    }
}
```

```
package com.hms.controller;

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.web.bind.annotation.*;

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

    @Autowired
    private AuthenticationManager authenticationManager;

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestParam String username, @RequestParam String password) {
        Authentication auth = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(username, password));
        if (auth.isAuthenticated()) {
            return ResponseEntity.ok("Login successful for user: " + username);
        } else {
            return ResponseEntity.status(401).body("Invalid credentials");
        }
    }
}
```

```
package com.hms.controller;

import com.hms.dto.BillRequest;
import com.hms.model.Bill;
import com.hms.service.BillService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/bill")
public class BillController {

    @Autowired
    private BillService billService;

    @PostMapping("/generate")
    public ResponseEntity<Bill> generateBill(@RequestBody BillRequest request) {
        Bill bill = billService.generateBill(request);
        return ResponseEntity.ok(bill);
    }
}
```

```
package com.hms.controller;

import com.hms.model.Appointment;
import com.hms.service.AppointmentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/doctor")
public class DoctorController {

    @Autowired
    private AppointmentService appointmentService;

    @GetMapping("/appointments/{doctorId}")
    public List<Appointment> getAppointments(@PathVariable Long doctorId) {
        return appointmentService.getAppointmentsByDoctor(doctorId);
    }
}
```

```
package com.hms.controller;

import com.hms.dto.RegisterPatientRequest;
import com.hms.model.Patient;
import com.hms.service.PatientService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/user")
public class UserController {

    @Autowired
    private PatientService patientService;

    @PostMapping("/register-patient")
    public ResponseEntity<Patient> registerPatient(@RequestBody RegisterPatientRequest
request) {
        Patient patient = patientService.registerPatient(request);
        return ResponseEntity.ok(patient);
    }
}



==========================================================



package com.hms.security;

import com.hms.model.User;
import com.hms.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
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.core.authority.SimpleGrantedAuthority;
import org.springframework.stereotype.Service;

import java.util.Collections;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

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




============================================================








package com.hms.repository;

import com.hms.model.Appointment;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface AppointmentRepository extends JpaRepository<Appointment, Long> {
    List<Appointment> findByDoctorId(Long doctorId);
    List<Appointment> findByPatientId(Long patientId);
}






package com.hms.repository;

import com.hms.model.Bill;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BillRepository extends JpaRepository<Bill, Long> {
}




package com.hms.repository;

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

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




package com.hms.repository;

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

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



package com.hms.repository;

import com.hms.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);
}


============================================================




package com.hms.service;

import com.hms.dto.AppointmentRequest;
import com.hms.model.*;
import com.hms.repository.AppointmentRepository;
import com.hms.repository.DoctorRepository;
import com.hms.repository.PatientRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class AppointmentService {
    @Autowired
    private AppointmentRepository appointmentRepository;

    @Autowired
    private DoctorRepository doctorRepository;

    @Autowired
    private PatientRepository patientRepository;

    public Appointment bookAppointment(AppointmentRequest request) {
        Doctor doctor = doctorRepository.findById(request.getDoctorId())
                .orElseThrow(() -> new RuntimeException("Doctor not found"));
        Patient patient = patientRepository.findById(request.getPatientId())
                .orElseThrow(() -> new RuntimeException("Patient not found"));

        // Check if slot already booked
        List<Appointment> existingAppointments =
appointmentRepository.findByDoctorId(doctor.getId());
        boolean slotTaken = existingAppointments.stream()
                .anyMatch(a -> a.getDate().equals(request.getDate()) &&
a.getTimeSlot().equals(request.getTimeSlot()));
        if (slotTaken) {
            throw new RuntimeException("Slot already booked");
        }

        Appointment appointment = new Appointment(request.getDate(), request.getTimeSlot(),
doctor, patient, AppointmentStatus.BOOKED);
        return appointmentRepository.save(appointment);
    }

    public List<Appointment> getAppointmentsByDoctor(Long doctorId) {
        return appointmentRepository.findByDoctorId(doctorId);
    }

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




package com.hms.service;

import com.hms.dto.BillRequest;
import com.hms.model.Appointment;
import com.hms.model.Bill;
import com.hms.repository.AppointmentRepository;
import com.hms.repository.BillRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDate;

@Service
```

```
public class BillService {
    @Autowired
    private BillRepository billRepository;

    @Autowired
    private AppointmentRepository appointmentRepository;

    public Bill generateBill(BillRequest request) {
        Appointment appointment = appointmentRepository.findById(request.getAppointmentId())
                .orElseThrow(() -> new RuntimeException("Appointment not found"));
        Bill bill = new Bill(request.getAmount(), LocalDate.now(), appointment);
        return billRepository.save(bill);
    }
}
```

```
package com.hms.service;

import com.hms.model.Role;
import com.hms.model.User;
import com.hms.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;

@Component
public class DataInitializer implements CommandLineRunner {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Override
    public void run(String... args) {
        if (userRepository.findByUsername("admin").isEmpty()) {
            User admin = new User("admin", passwordEncoder.encode("admin123"), Role.ADMIN);
            userRepository.save(admin);
            System.out.println("Default ADMIN created: username=admin, password=admin123");
        }
    }
}
```

```
package com.hms.service;

import com.hms.dto.RegisterDoctorRequest;
import com.hms.model.Doctor;
import com.hms.model.Role;
import com.hms.model.User;
import com.hms.repository.DoctorRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class DoctorService {
    @Autowired
    private DoctorRepository doctorRepository;

    @Autowired
    private UserService userService;

    public Doctor registerDoctor(RegisterDoctorRequest request) {
        User user = userService.registerUser(request.getUsername(), request.getPassword(),
Role.DOCTOR);
        Doctor doctor = new Doctor(request.getName(), request.getSpecialization(), user);
        return doctorRepository.save(doctor);
    }
}
```

```
package com.hms.service;

import com.hms.dto.RegisterPatientRequest;
import com.hms.model.Patient;
import com.hms.model.Role;
import com.hms.model.User;
import com.hms.repository.PatientRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class PatientService {
    @Autowired
    private PatientRepository patientRepository;

    @Autowired
    private UserService userService;

    public Patient registerPatient(RegisterPatientRequest request) {
        User user = userService.registerUser(request.getUsername(), request.getPassword(),
Role.PATIENT);
        Patient patient = new Patient(request.getName(), request.getContact(), user);
        return patientRepository.save(patient);
    }
}





package com.hms.service;

import com.hms.model.Role;
import com.hms.model.User;
import com.hms.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    public User registerUser(String username, String password, Role role) {
        if (userRepository.findByUsername(username).isPresent()) {
            throw new RuntimeException("Username already exists");
        }
        User user = new User(username, passwordEncoder.encode(password), role);
        return userRepository.save(user);
    }

    public Optional<User> findByUsername(String username) {
        return userRepository.findByUsername(username);
    }
}




==========================================================
package com.hms;

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

@SpringBootApplication
public class HmsProjectApplication {

        public static void main(String[] args) {
                SpringApplication.run(HmsProjectApplication.class, args);
        }

}
==========================================================





spring.application.name=HMSProject
spring.datasource.url=jdbc:mysql://localhost:3306/hms?useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

server.port=9090

==========================================================


<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>3.5.6</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.hms</groupId>
        <artifactId>HMSProject</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>HMSProject</name>
```

```
	<description>Hospital Management System Project</description>
	<url/>
	<licenses>
		<license/>
	</licenses>
	<developers>
		<developer/>
	</developers>
	<scm>
		<connection/>
		<developerConnection/>
		<tag/>
		<url/>
	</scm>
	<properties>
		<java.version>17</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-security</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
			<optional>true</optional>
		</dependency>
		<dependency>
			<groupId>com.mysql</groupId>
			<artifactId>mysql-connector-j</artifactId>
			<scope>runtime</scope>
		</dependency>
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<optional>true</optional>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
```

```
            </dependency>
            <dependency>
                <groupId>org.springframework.security</groupId>
                <artifactId>spring-security-test</artifactId>
                <scope>test</scope>
            </dependency>
            <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
            <dependency>
                <groupId>org.springdoc</groupId>
                <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
                <version>2.8.13</version>
            </dependency>
            <!-- Swagger / OpenAPI for Spring Boot 3+ -->
    <dependency>
      <groupId>org.springdoc</groupId>
      <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
      <version>2.6.0</version>
    </dependency>
        </dependencies>

        <build>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <configuration>
                        <annotationProcessorPaths>
                            <path>
                                <groupId>org.projectlombok</groupId>
                                <artifactId>lombok</artifactId>
                            </path>
                        </annotationProcessorPaths>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                    <configuration>
                        <excludes>
                            <exclude>
                                <groupId>org.projectlombok</groupId>
                                <artifactId>lombok</artifactId>
                            </exclude>
                        </excludes>
                    </configuration>
                </plugin>
            </plugins>
        </build>
```

</project>
....this is full backend project

Nice — this is a very solid backend! 👍 You already have a complete, working structure: entities, DTOs, repositories, services, controllers, security, Swagger, initialization and config. Nice job wiring everything together.

Below I'll do three things in this reply:

1. Quick **high-level review** (what's good + risks / bugs I spot).
2. **Concrete fixes & improvements** you can copy-paste (pom fix, validation, safer JPA mappings, a cancel-appointment flow, bill retrieval endpoints, better controller responses, transactional safety).
3. **Suggested next steps & enhancements** (tests, DTO responses, docs, frontend hints, deployment).

---

1) Quick review — what's good & issues to fix

What's good

- Clean separation: models, DTOs, services, controllers, repositories. ✅
- Security wired (PasswordEncoder, UserDetailsService, AuthenticationManager). ✅
- Centralized exception handling (`GlobalExceptionHandler`). ✅
- Swagger/OpenAPI added. ✅
- Data initializer to create admin user — handy. ✅

Potential issues / improvements to address

- **Validation missing** on DTOs (`@Valid`, `@NotBlank`, etc.). This will allow bad data into services.
- **Duplicate dependency** for `springdoc-openapi-starter-webmvc-ui` in `pom.xml` (two versions). Remove duplicate to avoid startup issues.
- **Controller responses** return entity directly; better to return DTOs and proper status codes (`201 Created` for create).
- **Exception types**: using `RuntimeException` in services is okay for a quick app, but better to throw custom exceptions (`ResourceNotFoundException`, `BadRequestException`) so handlers produce consistent API errors.
- **Transaction safety**: operations that write multiple tables should be `@Transactional` where appropriate.
- **JPA fetch/cascade defaults**: your relations are unconfigured (defaults may be fine), but consider `fetch = LAZY` on `@ManyToOne` to avoid n+1 issues and add cascade only where you want it.
- **Appointment slot check**: you check all appointments for a doctor and then stream; okay, but you might want a repository query to check existence which is more efficient.
- **Passwords**: you are hashing passwords — good. Make sure you never return password field in responses (use DTOs).

- **Swagger & Security**: you permit `/swagger-ui/**` but sometimes path is `/swagger-ui/index.html`; ensure swagger path matches springdoc version.
- **OpenAPI dependency mismatch**: two versions in pom — fix.
- **API design**: use plural resources and consistent naming (`/api/appointments`, `/api/bills`, `/api/admin/doctors`). Right now you have mixed namespaces (ok, but consider tidying).
- **Testing**: add unit & integration tests (spring-test).
- **DTO Responses & mapping**: avoid returning entity directly (exposes JPA proxies, relations, passwords). Use response DTOs or map with ModelMapper or manual mapping.

---

2) Concrete fixes & code snippets

Below are safe, practical changes you can apply immediately.

A. Fix `pom.xml` — remove duplicate springdoc dependency

Replace the two springdoc entries with a single dependency (keep the version you want). Example snippet:

```xml
<!-- remove the duplicate entry and keep one -->
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.8.13</version> <!-- choose one version only -->
</dependency>
```

B. Add validation annotations to DTOs & use `@Valid` in controllers

Example: update `RegisterPatientRequest` and `RegisterDoctorRequest`:

```java
package com.hms.dto;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;

public class RegisterPatientRequest {
    @NotBlank
    private String username;

    @NotBlank
    @Size(min = 6)
    private String password;

    @NotBlank
    private String name;

    @NotBlank
    private String contact;

    // constructors, getters, setters...
}
```

Then update controller methods to use **`@Valid`**:

```java
@PostMapping("/register-patient")
public ResponseEntity<Patient> registerPatient(@Valid @RequestBody RegisterPatientRequest
request) {
    Patient patient = patientService.registerPatient(request);
    return ResponseEntity.status(HttpStatus.CREATED).body(patient);
}
```

Make sure you have **`spring-boot-starter-validation`** (you already do).

C. Use DTO Response objects (avoid returning entities directly)

Create a **`PatientResponse`** DTO so you don't return password or lazy proxies:

```java
package com.hms.dto;

public class PatientResponse {
    private Long id;
    private String name;
    private String contact;
    private String username;
    // getters/setters, constructors
}
```

Map entity -> response in the service or controller before returning.

Example mapping inside **`PatientService`**:

```java
public PatientResponse registerPatient(RegisterPatientRequest request) {
    User user = userService.registerUser(request.getUsername(), request.getPassword(),
Role.PATIENT);
    Patient patient = new Patient(request.getName(), request.getContact(), user);
    Patient saved = patientRepository.save(patient);
    return new PatientResponse(saved.getId(), saved.getName(), saved.getContact(),
saved.getUser().getUsername());
}
```

Then controller returns **`ResponseEntity<PatientResponse>`**.

D. Make appointment slot check efficient (repository method)

Add method to **`AppointmentRepository`**:

```java
boolean existsByDoctorIdAndDateAndTimeSlot(Long doctorId, LocalDate date, String timeSlot);
```

Then in **`AppointmentService.bookAppointment`**:

```
java

if (appointmentRepository.existsByDoctorIdAndDateAndTimeSlot(request.getDoctorId(),
request.getDate(), request.getTimeSlot())) {
    throw new BadRequestException("Slot already booked");
}
```

This avoids fetching all appointments.

E. Add cancel appointment flow

Add service method:

```
java

import org.springframework.transaction.annotation.Transactional;

@Service
public class AppointmentService {
    // ... existing autowired repos

    @Transactional
    public Appointment cancelAppointment(Long appointmentId) {
        Appointment appointment = appointmentRepository.findById(appointmentId)
            .orElseThrow(() -> new ResourceNotFoundException("Appointment not found"));
        appointment.setStatus(AppointmentStatus.CANCELLED);
        return appointmentRepository.save(appointment);
    }
}
```

Controller endpoint:

```
java

@DeleteMapping("/{id}")
public ResponseEntity<Void> cancelAppointment(@PathVariable Long id) {
    appointmentService.cancelAppointment(id);
    return ResponseEntity.noContent().build();
}
```

(Using **`DELETE /api/appointments/{id}`** for cancel — or **`POST /api/appointments/{id}/cancel`** if you prefer semantics.)

F. Add bill retrieval endpoints

`BillService`:

```
java

public List<Bill> getBillsByPatient(Long patientId) {
    return billRepository.findByAppointmentPatientId(patientId);
}
```

Add repository query in **`BillRepository`**:

```java
List<Bill> findByAppointmentPatientId(Long patientId);
```

Controller:

```java
@GetMapping("/by-patient/{patientId}")
public ResponseEntity<List<BillResponse>> getBillsByPatient(@PathVariable Long patientId) {
    List<Bill> bills = billService.getBillsByPatient(patientId);
    List<BillResponse> responses = bills.stream()
        .map(b -> new BillResponse(b.getId(), b.getAmount(), b.getBillDate(),
b.getAppointment().getId()))
        .toList();
    return ResponseEntity.ok(responses);
}
```

G. Use `fetch = FetchType.LAZY` on `@ManyToOne` to avoid eager fetching surprises

In **`Appointment`** entity:

```java
@ManyToOne(fetch = FetchType.LAZY)
private Doctor doctor;

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

And set **`@OneToOne(fetch = FetchType.LAZY)`** where appropriate for **`Doctor.user`** and **`Patient.user`** if you don't always need user data.

If you change fetch type, be careful when serializing (use DTOs) or **`@JsonIgnore`** cycles.

H. Avoid returning `password` — mark as `@JsonIgnore` or use DTOs

In **`User`** entity (if you ever return it), add:

```java
import com.fasterxml.jackson.annotation.JsonProperty;

@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
private String password;
```

This prevents password from being serialized in API responses.

I. Replace generic `RuntimeException` with your custom exceptions

Where you do **`throw new RuntimeException("...")`**, prefer:

```java
throw new ResourceNotFoundException("Patient not found with id: " + request.getPatientId());
```

This integrates with your `GlobalExceptionHandler` and returns consistent structured errors.

J. Small SecurityConfig improvements

- Permit swagger UI resources in security config (already done but double-check path for your springdoc version).
- Consider using JWT for stateless auth in production (instead of httpBasic). For now httpBasic is fine for a backend demo.

---

3) Suggested next steps / roadmap

Short-term (next 1–2 coding sessions)

- Add the **response DTOs** and switch controllers to return DTOs.
- Add **validation** to DTOs with `@Valid`.
- Add the **cancel appointment** endpoint (snippet above).
- Add **bill listing** endpoints and repository queries.
- Remove duplicate dependency in `pom.xml`.

Medium-term

- Add **unit tests** (JUnit + Mockito) for services and `@SpringBootTest` integration tests for controllers.
- Add **API documentation** Examples and security scheme for Swagger (bearer token or basic).
- Add **logging** (SLF4J + structured logs).
- Add **CI** (GitHub Actions) and DB migration tool (Flyway/Liquibase).
- Add **Dockerfile** and `docker-compose` for MySQL + app for easy local dev.

Long-term / Production

- Replace `httpBasic` with **JWT** or OAuth2.
- Harden CORS and allowed origins.
- Add rate-limiting and monitoring (Prometheus + Grafana).
- Use role-based method security `@PreAuthorize` for controller methods.

Helpful check list you can copy for PRs / TODOs

- ☐ Remove duplicate springdoc dependency in `pom.xml`.
- ☐ Add validation annotations to DTOs and `@Valid` to controller params.

- ☐ Create Response DTOs for User/Patient/Doctor/Appointment/Bill.
- ☐ Ensure `password` is never serialized.
- ☐ Change appointment existence check to repository method (efficient).
- ☐ Add cancel appointment endpoint + transactional safety.
- ☐ Add bill lookup endpoints.
- ☐ Add unit/integration tests.
- ☐ Add README with endpoints and Postman collection.

If you'd like, I can do one of these **right now** (I'll implement it in full and give copy-paste-ready code):

- Option A: **Fix `pom.xml` + add DTO validations + update controllers to use `@Valid`** (quick).
- Option B: **Add cancel-appointment endpoint + repository method + tests** (useful).
- Option C: **Create Response DTOs and update Patient & Doctor endpoints to return safe DTOs instead of Entities**.
- Option D: **Create a Postman collection + README template** for your repo.

Tell me which of A/B/C/D you want immediately and I'll produce the exact files/patches you can paste into your project. (Or I can do a combination — pick what matters most.)

what changes you gave give me full code of it overall in backend simple

Great — I'll give you a **simple, ready-to-copy backend** with the improvements I suggested. I kept the structure close to your original project but made these concrete changes:

- Fixed `pom.xml` (removed duplicate dependency).
- Added validation annotations to request DTOs and used `@Valid` in controllers.
- Created **response DTOs** so we don't return entities (and never expose passwords).
- Added efficient repo method `existsByDoctorIdAndDateAndTimeSlot(...)`.
- Made `@ManyToOne` and `@OneToOne` use `FetchType.LAZY`.
- Marked `User.password` as write-only so it never gets serialized.
- Replaced generic `RuntimeException` uses with `ResourceNotFoundException` / `BadRequestException`.
- Added **cancel appointment** endpoint and service method.
- Added bills-by-patient retrieval.
- Minor controller response improvements (`201 Created`, `204 No Content`, etc.)

Below are the updated/added files you can paste into your project. I only included files that changed or were added — everything else in your project remains the same.

If you want, I can also prepare a diff/patch or a ZIP, but for now paste these into your codebase.

---

1) `pom.xml` (fixed — keep single springdoc dependency)

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.5.6</version>
    <relativePath/>
  </parent>

  <groupId>com.hms</groupId>
  <artifactId>HMSProject</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>HMSProject</name>

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

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

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

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

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>

    <dependency>
      <groupId>com.mysql</groupId>
      <artifactId>mysql-connector-j</artifactId>
      <scope>runtime</scope>
    </dependency>

    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springdoc</groupId>
      <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
      <version>2.8.13</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <annotationProcessorPaths>
            <path>
              <groupId>org.projectlombok</groupId>
              <artifactId>lombok</artifactId>
            </path>
          </annotationProcessorPaths>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <excludes>
            <exclude>
              <groupId>org.projectlombok</groupId>
              <artifactId>lombok</artifactId>
            </exclude>
          </excludes>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

---

2) Entities (small changes: fetch types, password write-only)

`com.hms.model.User`

```java
package com.hms.model;

import jakarta.persistence.*;
import com.fasterxml.jackson.annotation.JsonProperty;

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

    private String username;

    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private String password;

    @Enumerated(EnumType.STRING)
    private Role role;
    public User() {}

    public User(String username, String password, Role role) {
        this.username = username;
        this.password = password;
        this.role = role;
    }

    // getters & 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 getPassword() { return password; }
        public void setPassword(String password) { this.password = password; }
        public Role getRole() { return role; }
        public void setRole(Role role) { this.role = role; }
}
```

`com.hms.model.Appointment` (LAZY)

```java
package com.hms.model;

import jakarta.persistence.*;
import java.time.LocalDate;

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

    private LocalDate date;
    private String timeSlot;

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

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

    @Enumerated(EnumType.STRING)
    private AppointmentStatus status;

    public Appointment() {}

    public Appointment(LocalDate date, String timeSlot, Doctor doctor, Patient patient,
AppointmentStatus status) {
        this.date = date;
        this.timeSlot = timeSlot;
        this.doctor = doctor;
        this.patient = patient;
        this.status = status;
    }

    // getters & setters
        public Long getId() { return id; }
        public void setId(Long id) { this.id = id; }
        public LocalDate getDate() { return date; }
        public void setDate(LocalDate date) { this.date = date; }
        public String getTimeSlot() { return timeSlot; }
        public void setTimeSlot(String timeSlot) { this.timeSlot = timeSlot; }
        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 AppointmentStatus getStatus() { return status; }
        public void setStatus(AppointmentStatus status) { this.status = status; }
}
```

`com.hms.model.Doctor` (user LAZY)

```java
package com.hms.model;

import jakarta.persistence.*;

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

    private String name;
    private String specialization;

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

    public Doctor() {}

    public Doctor(String name, String specialization, User user) {
        this.name = name;
        this.specialization = specialization;
        this.user = user;
    }

    // getters & setters
        public Long getId() { return id; }
        public void setId(Long id) { this.id = id; }
        public String getName() { return name; }
        public void setName(String name) { this.name = name; }
        public String getSpecialization() { return specialization; }
        public void setSpecialization(String specialization) { this.specialization =
specialization; }
        public User getUser() { return user; }
        public void setUser(User user) { this.user = user; }
}
```

`com.hms.model.Patient` (user LAZY)

```java
package com.hms.model;

import jakarta.persistence.*;

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

    private String name;
    private String contact;

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

    public Patient() {}

    public Patient(String name, String contact, User user) {
        this.name = name;
        this.contact = contact;
        this.user = user;
    }

    // getters & setters
        public Long getId() { return id; }
        public void setId(Long id) { this.id = id; }
        public String getName() { return name; }
        public void setName(String name) { this.name = name; }
        public String getContact() { return contact; }
        public void setContact(String contact) { this.contact = contact; }
        public User getUser() { return user; }
        public void setUser(User user) { this.user = user; }
}
```

`com.hms.model.Bill` (unchanged)

(keep same as you had)

---

3) Repositories (small addition)

`com.hms.repository.AppointmentRepository`

```java
package com.hms.repository;

import com.hms.model.Appointment;
import org.springframework.data.jpa.repository.JpaRepository;
import java.time.LocalDate;
import java.util.List;

public interface AppointmentRepository extends JpaRepository<Appointment, Long> {
    List<Appointment> findByDoctorId(Long doctorId);
    List<Appointment> findByPatientId(Long patientId);

    boolean existsByDoctorIdAndDateAndTimeSlot(Long doctorId, LocalDate date, String timeSlot);
}
```

`com.hms.repository.BillRepository` (added custom query)

```java
package com.hms.repository;

import com.hms.model.Bill;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface BillRepository extends JpaRepository<Bill, Long> {
    List<Bill> findByAppointmentPatientId(Long patientId);
}
```

Other repository interfaces unchanged: **`UserRepository`**, **`DoctorRepository`**, **`PatientRepository`**.

---

4) DTOs (request + response) with validation

`com.hms.dto.RegisterPatientRequest`

```java
package com.hms.dto;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;

public class RegisterPatientRequest {
    @NotBlank
    private String username;

    @NotBlank
    @Size(min = 6)
    private String password;

    @NotBlank
    private String name;

    @NotBlank
    private String contact;

    public RegisterPatientRequest() {}

    public RegisterPatientRequest(String username, String password, String name, String contact)
{
        this.username = username;
        this.password = password;
        this.name = name;
        this.contact = contact;
    }

    // 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; }
        public String getName() { return name; }
        public void setName(String name) { this.name = name; }
        public String getContact() { return contact; }
        public void setContact(String contact) { this.contact = contact; }
}
```

`com.hms.dto.RegisterDoctorRequest`

```java
package com.hms.dto;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;

public class RegisterDoctorRequest {
    @NotBlank
    private String username;

    @NotBlank
    @Size(min = 6)
    private String password;

    @NotBlank
    private String name;

    @NotBlank
    private String specialization;

    public RegisterDoctorRequest() {}

    public RegisterDoctorRequest(String username, String password, String name, String
specialization) {
        this.username = username;
        this.password = password;
        this.name = name;
        this.specialization = specialization;
    }

    // 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; }
        public String getName() { return name; }
        public void setName(String name) { this.name = name; }
        public String getSpecialization() { return specialization; }
        public void setSpecialization(String specialization) { this.specialization =
specialization; }
}
```

`com.hms.dto.AppointmentRequest` (add validation)

```java
package com.hms.dto;

import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.NotBlank;
import java.time.LocalDate;

public class AppointmentRequest {
    @NotNull
    private Long doctorId;
    @NotNull
    private Long patientId;
    @NotNull
    private LocalDate date;
    @NotBlank
    private String timeSlot;

    public AppointmentRequest() {}

    public AppointmentRequest(Long doctorId, Long patientId, LocalDate date, String timeSlot) {
        this.doctorId = doctorId;
        this.patientId = patientId;
        this.date = date;
        this.timeSlot = timeSlot;
    }

    // getters & setters
        public Long getDoctorId() { return doctorId; }
        public void setDoctorId(Long doctorId) { this.doctorId = doctorId; }
        public Long getPatientId() { return patientId; }
        public void setPatientId(Long patientId) { this.patientId = patientId; }
        public LocalDate getDate() { return date; }
        public void setDate(LocalDate date) { this.date = date; }
        public String getTimeSlot() { return timeSlot; }
        public void setTimeSlot(String timeSlot) { this.timeSlot = timeSlot; }
}
```

`com.hms.dto.BillRequest` (add validation)

```java
package com.hms.dto;

import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Positive;

public class BillRequest {
    @NotNull
    private Long appointmentId;

    @NotNull
    @Positive
    private Double amount;

    public BillRequest() {}
    public BillRequest(Long appointmentId, Double amount) {
        this.appointmentId = appointmentId;
        this.amount = amount;
    }

    // getters & setters
        public Long getAppointmentId() { return appointmentId; }
        public void setAppointmentId(Long appointmentId) { this.appointmentId = appointmentId; }
        public Double getAmount() { return amount; }
        public void setAmount(Double amount) { this.amount = amount; }
}
```

Response DTOs: `PatientResponse`, `DoctorResponse`, `AppointmentResponse`, `BillResponse`

`PatientResponse`:

```java
package com.hms.dto;

public class PatientResponse {
    private Long id;
    private String name;
    private String contact;
    private String username;

    public PatientResponse() {}
    public PatientResponse(Long id, String name, String contact, String username) {
        this.id = id; this.name = name; this.contact = contact; this.username = username;
    }

    // getters & setters
        public Long getId() { return id; }
        public void setId(Long id) { this.id = id; }
        public String getName() { return name; }
        public void setName(String name) { this.name = name; }
        public String getContact() { return contact; }
        public void setContact(String contact) { this.contact = contact; }
        public String getUsername() { return username; }
        public void setUsername(String username) { this.username = username; }
}
```

`DoctorResponse`:

```java
package com.hms.dto;

public class DoctorResponse {
    private Long id;
    private String name;
    private String specialization;
    private String username;

    public DoctorResponse() {}
    public DoctorResponse(Long id, String name, String specialization, String username) {
        this.id = id; this.name = name; this.specialization = specialization; this.username =
username;
    }

    // getters & setters
```

```
        public Long getId() { return id; }
        public void setId(Long id) { this.id = id; }
        public String getName() { return name; }
        public void setName(String name) { this.name = name; }
        public String getSpecialization() { return specialization; }
        public void setSpecialization(String specialization) { this.specialization =
specialization; }
        public String getUsername() { return username; }
        public void setUsername(String username) { this.username = username; }
}
```

`AppointmentResponse`:

```java
package com.hms.dto;

import java.time.LocalDate;

public class AppointmentResponse {
    private Long id;
    private LocalDate date;
    private String timeSlot;
    private Long doctorId;
    private String doctorName;
    private Long patientId;
    private String patientName;
    private String status;

    public AppointmentResponse() {}

    public AppointmentResponse(Long id, LocalDate date, String timeSlot,
                               Long doctorId, String doctorName,
                               Long patientId, String patientName,
                               String status) {
        this.id = id; this.date = date; this.timeSlot = timeSlot;
        this.doctorId = doctorId; this.doctorName = doctorName;
        this.patientId = patientId; this.patientName = patientName;
        this.status = status;
    }

    // getters & setters
        public Long getId() { return id; }
        public void setId(Long id) { this.id = id; }
        public LocalDate getDate() { return date; }
        public void setDate(LocalDate date) { this.date = date; }
        public String getTimeSlot() { return timeSlot; }
        public void setTimeSlot(String timeSlot) { this.timeSlot = timeSlot; }
        public Long getDoctorId() { return doctorId; }
        public void setDoctorId(Long doctorId) { this.doctorId = doctorId; }
        public String getDoctorName() { return doctorName; }
        public void setDoctorName(String doctorName) { this.doctorName = doctorName; }
        public Long getPatientId() { return patientId; }
        public void setPatientId(Long patientId) { this.patientId = patientId; }
        public String getPatientName() { return patientName; }
        public void setPatientName(String patientName) { this.patientName = patientName; }
        public String getStatus() { return status; }
        public void setStatus(String status) { this.status = status; }
}
```

`BillResponse`:

```java
package com.hms.dto;

import java.time.LocalDate;

public class BillResponse {
    private Long id;
    private Double amount;
    private LocalDate billDate;
    private Long appointmentId;

    public BillResponse() {}
    public BillResponse(Long id, Double amount, LocalDate billDate, Long appointmentId) {
        this.id = id; this.amount = amount; this.billDate = billDate; this.appointmentId =
appointmentId;
    }

    // getters & setters
        public Long getId() { return id; }
        public void setId(Long id) { this.id = id; }
        public Double getAmount() { return amount; }
        public void setAmount(Double amount) { this.amount = amount; }
        public LocalDate getBillDate() { return billDate; }
        public void setBillDate(LocalDate billDate) { this.billDate = billDate; }
        public Long getAppointmentId() { return appointmentId; }
        public void setAppointmentId(Long appointmentId) { this.appointmentId = appointmentId; }
}
```

---

5) Services (updated to use repository methods, use custom exceptions, return response DTOs where needed)

`com.hms.service.UserService` (unchanged behavior, but throws Runtime -> keep as-is)

```java
package com.hms.service;

import com.hms.model.Role;
import com.hms.model.User;
import com.hms.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import java.util.Optional;

@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    public User registerUser(String username, String password, Role role) {
        if (userRepository.findByUsername(username).isPresent()) {
            throw new RuntimeException("Username already exists");
        }
        User user = new User(username, passwordEncoder.encode(password), role);
        return userRepository.save(user);
    }

    public Optional<User> findByUsername(String username) {
        return userRepository.findByUsername(username);
    }
}
```

> Note: you may later replace the `RuntimeException` with `BadRequestException`; I kept it to avoid changing multiple call-sites — if you want, I can change it too.

`com.hms.service.DoctorService` (returns DoctorResponse)

```java
package com.hms.service;

import com.hms.dto.DoctorResponse;
import com.hms.dto.RegisterDoctorRequest;
import com.hms.model.Doctor;
import com.hms.model.Role;
import com.hms.model.User;
import com.hms.repository.DoctorRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class DoctorService {
    @Autowired
    private DoctorRepository doctorRepository;

    @Autowired
    private UserService userService;

    public DoctorResponse registerDoctor(RegisterDoctorRequest request) {
        User user = userService.registerUser(request.getUsername(), request.getPassword(), Role.DOCTOR);
        Doctor doctor = new Doctor(request.getName(), request.getSpecialization(), user);
        Doctor saved = doctorRepository.save(doctor);
        return new DoctorResponse(saved.getId(), saved.getName(), saved.getSpecialization(), saved.getUser().getUsername());
    }
}
```

`com.hms.service.PatientService` (returns PatientResponse)

```java
package com.hms.service;

import com.hms.dto.PatientResponse;
import com.hms.dto.RegisterPatientRequest;
import com.hms.model.Patient;
import com.hms.model.Role;
import com.hms.model.User;
import com.hms.repository.PatientRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class PatientService {
    @Autowired
    private PatientRepository patientRepository;

    @Autowired
    private UserService userService;

    public PatientResponse registerPatient(RegisterPatientRequest request) {
        User user = userService.registerUser(request.getUsername(), request.getPassword(), Role.PATIENT);
        Patient patient = new Patient(request.getName(), request.getContact(), user);
        Patient saved = patientRepository.save(patient);
        return new PatientResponse(saved.getId(), saved.getName(), saved.getContact(), saved.getUser().getUsername());
    }
}
```

`com.hms.service.AppointmentService` (efficient slot check, cancel method, returns AppointmentResponse)

```java
package com.hms.service;

import com.hms.dto.AppointmentRequest;
import com.hms.dto.AppointmentResponse;
import com.hms.exception.BadRequestException;
import com.hms.exception.ResourceNotFoundException;
import com.hms.model.*;
import com.hms.repository.AppointmentRepository;
import com.hms.repository.DoctorRepository;
import com.hms.repository.PatientRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.stream.Collectors;

@Service
public class AppointmentService {
    @Autowired
    private AppointmentRepository appointmentRepository;

    @Autowired
    private DoctorRepository doctorRepository;

    @Autowired
    private PatientRepository patientRepository;

    public AppointmentResponse bookAppointment(AppointmentRequest request) {
        Doctor doctor = doctorRepository.findById(request.getDoctorId())
                .orElseThrow(() -> new ResourceNotFoundException("Doctor not found"));
        Patient patient = patientRepository.findById(request.getPatientId())
                .orElseThrow(() -> new ResourceNotFoundException("Patient not found"));

        if (appointmentRepository.existsByDoctorIdAndDateAndTimeSlot(doctor.getId(),
request.getDate(), request.getTimeSlot())) {
            throw new BadRequestException("Slot already booked");
        }

        Appointment appointment = new Appointment(request.getDate(), request.getTimeSlot(),
doctor, patient, AppointmentStatus.BOOKED);
        Appointment saved = appointmentRepository.save(appointment);
        return mapToResponse(saved);
    }

    public List<AppointmentResponse> getAppointmentsByDoctor(Long doctorId) {
        List<Appointment> list = appointmentRepository.findByDoctorId(doctorId);
        return list.stream().map(this::mapToResponse).collect(Collectors.toList());
    }

    public List<AppointmentResponse> getAppointmentsByPatient(Long patientId) {
        List<Appointment> list = appointmentRepository.findByPatientId(patientId);
        return list.stream().map(this::mapToResponse).collect(Collectors.toList());
    }

    @Transactional
    public void cancelAppointment(Long appointmentId) {
        Appointment appointment = appointmentRepository.findById(appointmentId)
                .orElseThrow(() -> new ResourceNotFoundException("Appointment not found"));
        if (appointment.getStatus() == AppointmentStatus.CANCELLED) {
            throw new BadRequestException("Appointment already cancelled");
        }
        appointment.setStatus(AppointmentStatus.CANCELLED);
        appointmentRepository.save(appointment);
    }

    private AppointmentResponse mapToResponse(Appointment a) {
        String doctorName = a.getDoctor() != null ? a.getDoctor().getName() : null;
        String patientName = a.getPatient() != null ? a.getPatient().getName() : null;
        Long doctorId = a.getDoctor() != null ? a.getDoctor().getId() : null;
        Long patientId = a.getPatient() != null ? a.getPatient().getId() : null;
        return new AppointmentResponse(a.getId(), a.getDate(), a.getTimeSlot(),
                doctorId, doctorName, patientId, patientName, a.getStatus().name());
    }
}
```

`com.hms.service.BillService` (generate + getByPatient)

```java
java


package com.hms.service;

import com.hms.dto.BillRequest;
import com.hms.dto.BillResponse;
import com.hms.exception.ResourceNotFoundException;
import com.hms.model.Appointment;
import com.hms.model.Bill;
import com.hms.repository.AppointmentRepository;
import com.hms.repository.BillRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDate;
import java.util.List;
import java.util.stream.Collectors;

@Service
public class BillService {
    @Autowired
    private BillRepository billRepository;

    @Autowired
    private AppointmentRepository appointmentRepository;

    public BillResponse generateBill(BillRequest request) {
        Appointment appointment = appointmentRepository.findById(request.getAppointmentId())
                .orElseThrow(() -> new ResourceNotFoundException("Appointment not found"));
        Bill bill = new Bill(request.getAmount(), LocalDate.now(), appointment);
        Bill saved = billRepository.save(bill);
        return new BillResponse(saved.getId(), saved.getAmount(), saved.getBillDate(),
saved.getAppointment().getId());
    }

    public List<BillResponse> getBillsByPatient(Long patientId) {
        List<Bill> bills = billRepository.findByAppointmentPatientId(patientId);
        return bills.stream()
                .map(b -> new BillResponse(b.getId(), b.getAmount(), b.getBillDate(),
b.getAppointment().getId()))
                .collect(Collectors.toList());
    }
}
```

---

6) Controllers (use @Valid and return response DTOs)

`com.hms.controller.UserController` (register patient)

```java
java


package com.hms.controller;

import com.hms.dto.PatientResponse;
import com.hms.dto.RegisterPatientRequest;
import com.hms.service.PatientService;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/user")
public class UserController {

    @Autowired
    private PatientService patientService;

    @PostMapping("/register-patient")
    public ResponseEntity<PatientResponse> registerPatient(@Valid @RequestBody
RegisterPatientRequest request) {
        PatientResponse patient = patientService.registerPatient(request);
        return ResponseEntity.status(HttpStatus.CREATED).body(patient);
    }
}
```

`com.hms.controller.AdminController` (register doctor)

```java
package com.hms.controller;

import com.hms.dto.DoctorResponse;
import com.hms.dto.RegisterDoctorRequest;
import com.hms.service.DoctorService;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/admin")
public class AdminController {

    @Autowired
    private DoctorService doctorService;

    @PostMapping("/register-doctor")
    public ResponseEntity<DoctorResponse> registerDoctor(@Valid @RequestBody
RegisterDoctorRequest request) {
        DoctorResponse doctor = doctorService.registerDoctor(request);
        return ResponseEntity.status(HttpStatus.CREATED).body(doctor);
    }
}
```

`com.hms.controller.AppointmentController` (book + cancel + list by patient)

```java
package com.hms.controller;

import com.hms.dto.AppointmentRequest;
import com.hms.dto.AppointmentResponse;
import com.hms.service.AppointmentService;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import java.util.List;

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

    @Autowired
    private AppointmentService appointmentService;

    @PostMapping("/book")
    public ResponseEntity<AppointmentResponse> bookAppointment(@Valid @RequestBody
AppointmentRequest request) {
        AppointmentResponse appointment = appointmentService.bookAppointment(request);
        return ResponseEntity.status(HttpStatus.CREATED).body(appointment);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> cancelAppointment(@PathVariable Long id) {
        appointmentService.cancelAppointment(id);
        return ResponseEntity.noContent().build();
    }

    @GetMapping("/by-patient/{patientId}")
    public ResponseEntity<List<AppointmentResponse>> getByPatient(@PathVariable Long patientId)
{
        List<AppointmentResponse> list = appointmentService.getAppointmentsByPatient(patientId);
        return ResponseEntity.ok(list);
    }
}
```

`com.hms.controller.DoctorController` (appointments by doctor)

```java
package com.hms.controller;

import com.hms.dto.AppointmentResponse;
import com.hms.service.AppointmentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/doctor")
public class DoctorController {

    @Autowired
    private AppointmentService appointmentService;

    @GetMapping("/appointments/{doctorId}")
    public List<AppointmentResponse> getAppointments(@PathVariable Long doctorId) {
        return appointmentService.getAppointmentsByDoctor(doctorId);
    }
}
```

`com.hms.controller.BillController` (generate + getByPatient)

```java
package com.hms.controller;

import com.hms.dto.BillRequest;
import com.hms.dto.BillResponse;
import com.hms.service.BillService;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/bill")
public class BillController {
    @Autowired
    private BillService billService;

    @PostMapping("/generate")
    public ResponseEntity<BillResponse> generateBill(@Valid @RequestBody BillRequest request) {
        BillResponse bill = billService.generateBill(request);
        return ResponseEntity.status(HttpStatus.CREATED).body(bill);
    }

    @GetMapping("/by-patient/{patientId}")
    public ResponseEntity<List<BillResponse>> getBillsByPatient(@PathVariable Long patientId) {
        List<BillResponse> list = billService.getBillsByPatient(patientId);
        return ResponseEntity.ok(list);
    }
}
```

`com.hms.controller.AuthController` (unchanged; basic auth)

```java
package com.hms.controller;

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.web.bind.annotation.*;

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

    @Autowired
    private AuthenticationManager authenticationManager;

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestParam String username, @RequestParam String
password) {
        Authentication auth = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(username, password));
        if (auth.isAuthenticated()) {
            return ResponseEntity.ok("Login successful for user: " + username);
        } else {
            return ResponseEntity.status(401).body("Invalid credentials");
        }
    }
}
```

---

7) Exceptions (already present but re-showing for clarity)

`com.hms.exception.BadRequestException`

```java
package com.hms.exception;

public class BadRequestException extends RuntimeException {
    public BadRequestException(String message) { super(message); }
}
```

`com.hms.exception.ResourceNotFoundException`

```java
package com.hms.exception;

public class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException(String message) { super(message); }
}
```

`com.hms.exception.ApiError` and `GlobalExceptionHandler` — keep existing `GlobalExceptionHandler` you already had; it will handle these exceptions correctly.

---

8) DataInitializer (unchanged)

You already had `DataInitializer` creating admin — keep it.

---

9) SecurityConfig (minor note)

Your `SecurityConfig` is fine for a demo. Keep `httpBasic()` for now. If you move to JWT later, I can add that.

---

10) `application.properties` (unchanged — for quick local run)

```properties
spring.application.name=HMSProject
spring.datasource.url=jdbc:mysql://localhost:3306/hms?useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

server.port=9090
```

How to test quickly (Postman / curl examples)

1. Start app: `mvn spring-boot:run`
2. Create doctor (admin path; httpBasic with admin credentials created by DataInitializer if using `admin:admin123`):

```bash
POST http://localhost:9090/api/admin/register-doctor
Body (JSON):
{
  "username": "doc1",
  "password": "password123",
  "name": "Dr. A",
  "specialization": "ENT"
}
```

3. Register patient:

```bash
POST http://localhost:9090/api/user/register-patient
Body:
{
  "username": "pat1",
  "password": "password123",
  "name": "Patient A",
  "contact": "9999999999"
}
```

4. Book appointment:

```bash
POST http://localhost:9090/api/appointments/book
Body:
{
  "doctorId": 1,
  "patientId": 2,
  "date": "2025-10-20",
  "timeSlot": "10:00-10:30"
}
```

5. Cancel appointment:

```bash
DELETE http://localhost:9090/api/appointments/1
```

6. Generate bill:

```bash
POST http://localhost:9090/api/bill/generate
Body:
{
  "appointmentId": 1,
  "amount": 500.0
}
```

7. Get bills by patient:

```bash
GET http://localhost:9090/api/bill/by-patient/2
```

Final notes & next offer

- I kept logic simple and readable. I returned response DTOs so your API responses won't accidentally expose passwords or JPA internals.
- If you'd like, I can:
  1. Convert the remaining services/controllers to use DTOs in the same style (if any endpoints still return entities).

2. Create a Postman collection or README with endpoint docs.
3. Add unit/integration tests for `AppointmentService` and `BillService`.

Which of those should I do next? Or do you want me to paste any missing file from your original project updated to the new structure?

ya give everything same style and also logger slf4J

Great — I updated your backend so **every controller and service follows the same DTO-style responses** and added **SLF4J logging** (using `org.slf4j.Logger` + `LoggerFactory`) across the codebase. I also replaced the last remaining `RuntimeException` use in `UserService` with `BadRequestException` and made other small logging-friendly improvements (no behavioural changes beyond what we discussed earlier).

Below are the updated files — copy-paste each into your project (same package paths as before). I focused on the main app classes you need to run and test: controllers, services, security, initializer, exception handler, and user-details service — all now use SLF4J logging and consistent DTO responses.

---

1) `com.hms.controller.AdminController`

```java
package com.hms.controller;

import com.hms.dto.DoctorResponse;
import com.hms.dto.RegisterDoctorRequest;
import com.hms.service.DoctorService;
import jakarta.validation.Valid;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/admin")
public class AdminController {
    private static final Logger log = LoggerFactory.getLogger(AdminController.class);

    @Autowired
    private DoctorService doctorService;

    @PostMapping("/register-doctor")
    public ResponseEntity<DoctorResponse> registerDoctor(@Valid @RequestBody
RegisterDoctorRequest request) {
        log.info("Admin requested doctor registration: username={}", request.getUsername());
        DoctorResponse doctor = doctorService.registerDoctor(request);
        log.info("Doctor registered: id={} username={}", doctor.getId(), doctor.getUsername());
        return ResponseEntity.status(HttpStatus.CREATED).body(doctor);
    }
}
```

---

2) `com.hms.controller.UserController`

```java
package com.hms.controller;

import com.hms.dto.PatientResponse;
import com.hms.dto.RegisterPatientRequest;
import com.hms.service.PatientService;
import jakarta.validation.Valid;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/user")
public class UserController {
    private static final Logger log = LoggerFactory.getLogger(UserController.class);

    @Autowired
    private PatientService patientService;

    @PostMapping("/register-patient")
    public ResponseEntity<PatientResponse> registerPatient(@Valid @RequestBody
RegisterPatientRequest request) {
        log.info("Register patient request received: username={}", request.getUsername());
        PatientResponse patient = patientService.registerPatient(request);
        log.info("Patient registered: id={} username={}", patient.getId(),
patient.getUsername());
        return ResponseEntity.status(HttpStatus.CREATED).body(patient);
    }
}
```

---

3) `com.hms.controller.AppointmentController`

```java
package com.hms.controller;

import com.hms.dto.AppointmentRequest;
import com.hms.dto.AppointmentResponse;
import com.hms.service.AppointmentService;
import jakarta.validation.Valid;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/appointments")
public class AppointmentController {
    private static final Logger log = LoggerFactory.getLogger(AppointmentController.class);

    @Autowired
    private AppointmentService appointmentService;

    @PostMapping("/book")
    public ResponseEntity<AppointmentResponse> bookAppointment(@Valid @RequestBody
AppointmentRequest request) {
        log.info("Booking appointment: doctorId={}, patientId={}, date={}, slot={}",
                request.getDoctorId(), request.getPatientId(), request.getDate(),
request.getTimeSlot());
        AppointmentResponse appointment = appointmentService.bookAppointment(request);
        log.info("Appointment booked: id={} doctorId={} patientId={}", appointment.getId(),
appointment.getDoctorId(), appointment.getPatientId());
        return ResponseEntity.status(HttpStatus.CREATED).body(appointment);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> cancelAppointment(@PathVariable Long id) {
        log.info("Cancel appointment requested: id={}", id);
        appointmentService.cancelAppointment(id);
        log.info("Appointment cancelled: id={}", id);
        return ResponseEntity.noContent().build();
    }

    @GetMapping("/by-patient/{patientId}")
    public ResponseEntity<List<AppointmentResponse>> getByPatient(@PathVariable Long patientId)
{
        log.info("Get appointments by patient: patientId={}", patientId);
        List<AppointmentResponse> list = appointmentService.getAppointmentsByPatient(patientId);
        return ResponseEntity.ok(list);
    }
}
```

---

4) `com.hms.controller.DoctorController`

java

```
package com.hms.controller;

import com.hms.dto.AppointmentResponse;
import com.hms.service.AppointmentService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/doctor")
public class DoctorController {
    private static final Logger log = LoggerFactory.getLogger(DoctorController.class);

    @Autowired
    private AppointmentService appointmentService;

    @GetMapping("/appointments/{doctorId}")
    public List<AppointmentResponse> getAppointments(@PathVariable Long doctorId) {
        log.info("Fetching appointments for doctorId={}", doctorId);
        return appointmentService.getAppointmentsByDoctor(doctorId);
    }
}
```

---

5) `com.hms.controller.BillController`

java

```
package com.hms.controller;

import com.hms.dto.BillRequest;
import com.hms.dto.BillResponse;
import com.hms.service.BillService;
import jakarta.validation.Valid;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/bill")
public class BillController {
    private static final Logger log = LoggerFactory.getLogger(BillController.class);

    @Autowired
    private BillService billService;

    @PostMapping("/generate")
    public ResponseEntity<BillResponse> generateBill(@Valid @RequestBody BillRequest request) {
        log.info("Generating bill for appointmentId={} amount={}", request.getAppointmentId(),
request.getAmount());
        BillResponse bill = billService.generateBill(request);
        log.info("Bill generated: id={} appointmentId={}", bill.getId(),
bill.getAppointmentId());
        return ResponseEntity.status(HttpStatus.CREATED).body(bill);
    }

    @GetMapping("/by-patient/{patientId}")
    public ResponseEntity<List<BillResponse>> getBillsByPatient(@PathVariable Long patientId) {
        log.info("Fetch bills for patientId={}", patientId);
        List<BillResponse> list = billService.getBillsByPatient(patientId);
        return ResponseEntity.ok(list);
    }
}
```

---

6) `com.hms.controller.AuthController` (basic auth; logging only)

```java
package com.hms.controller;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
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.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
public class AuthController {
    private static final Logger log = LoggerFactory.getLogger(AuthController.class);

    @Autowired
    private AuthenticationManager authenticationManager;

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestParam String username, @RequestParam String
password) {
        log.info("Login attempt for username={}", username);
        Authentication auth = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(username, password));
        if (auth.isAuthenticated()) {
            log.info("Login successful: username={}", username);
            return ResponseEntity.ok("Login successful for user: " + username);
        } else {
            log.warn("Login failed: username={}", username);
            return ResponseEntity.status(401).body("Invalid credentials");
        }
    }
}
```

7) `com.hms.service.UserService` (uses BadRequestException, logs actions)

```java
java


package com.hms.service;

import com.hms.exception.BadRequestException;
import com.hms.model.Role;
import com.hms.model.User;
import com.hms.repository.UserRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import java.util.Optional;

@Service
public class UserService {
    private static final Logger log = LoggerFactory.getLogger(UserService.class);

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    public User registerUser(String username, String password, Role role) {
        log.debug("Registering user: username={} role={}", username, role);
        if (userRepository.findByUsername(username).isPresent()) {
            log.warn("Username already exists: {}", username);
            throw new BadRequestException("Username already exists");
        }
        User user = new User(username, passwordEncoder.encode(password), role);
        User saved = userRepository.save(user);
        log.info("User registered: id={} username={} role={}", saved.getId(),
saved.getUsername(), saved.getRole());
        return saved;
    }

    public Optional<User> findByUsername(String username) {
        return userRepository.findByUsername(username);
    }
}
```

---

8) `com.hms.service.DoctorService` (returns DoctorResponse; logs)

```java
java


package com.hms.service;

import com.hms.dto.DoctorResponse;
import com.hms.dto.RegisterDoctorRequest;
import com.hms.model.Doctor;
import com.hms.model.Role;
import com.hms.model.User;
import com.hms.repository.DoctorRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
```

```
public class DoctorService {
    private static final Logger log = LoggerFactory.getLogger(DoctorService.class);

    @Autowired
    private DoctorRepository doctorRepository;

    @Autowired
    private UserService userService;

    public DoctorResponse registerDoctor(RegisterDoctorRequest request) {
        log.debug("Register doctor request: username={} name={}", request.getUsername(),
request.getName());
        User user = userService.registerUser(request.getUsername(), request.getPassword(),
Role.DOCTOR);
        Doctor doctor = new Doctor(request.getName(), request.getSpecialization(), user);
        Doctor saved = doctorRepository.save(doctor);
        log.info("Doctor saved: id={} username={}", saved.getId(),
saved.getUser().getUsername());
        return new DoctorResponse(saved.getId(), saved.getName(), saved.getSpecialization(),
saved.getUser().getUsername());
    }
}
```

---

9) `com.hms.service.PatientService` (returns PatientResponse; logs)

```java
package com.hms.service;

import com.hms.dto.PatientResponse;
import com.hms.dto.RegisterPatientRequest;
import com.hms.model.Patient;
import com.hms.model.Role;
import com.hms.model.User;
import com.hms.repository.PatientRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class PatientService {
    private static final Logger log = LoggerFactory.getLogger(PatientService.class);

    @Autowired
    private PatientRepository patientRepository;

    @Autowired
    private UserService userService;

    public PatientResponse registerPatient(RegisterPatientRequest request) {
        log.debug("Register patient request: username={} name={}", request.getUsername(),
request.getName());
        User user = userService.registerUser(request.getUsername(), request.getPassword(),
Role.PATIENT);
        Patient patient = new Patient(request.getName(), request.getContact(), user);
        Patient saved = patientRepository.save(patient);
        log.info("Patient saved: id={} username={}", saved.getId(),
saved.getUser().getUsername());
        return new PatientResponse(saved.getId(), saved.getName(), saved.getContact(),
saved.getUser().getUsername());
    }
}
```

---

10) `com.hms.service.AppointmentService` (already had logging earlier, but here's final with logger)

```java
package com.hms.service;

import com.hms.dto.AppointmentRequest;
import com.hms.dto.AppointmentResponse;
import com.hms.exception.BadRequestException;
import com.hms.exception.ResourceNotFoundException;
import com.hms.model.*;
import com.hms.repository.AppointmentRepository;
import com.hms.repository.DoctorRepository;
import com.hms.repository.PatientRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.stream.Collectors;

@Service
public class AppointmentService {
    private static final Logger log = LoggerFactory.getLogger(AppointmentService.class);

    @Autowired
    private AppointmentRepository appointmentRepository;

    @Autowired
    private DoctorRepository doctorRepository;

    @Autowired
    private PatientRepository patientRepository;

    public AppointmentResponse bookAppointment(AppointmentRequest request) {
        log.debug("Attempt booking: doctorId={} patientId={} date={} slot={}",
                request.getDoctorId(), request.getPatientId(), request.getDate(),
request.getTimeSlot());

        Doctor doctor = doctorRepository.findById(request.getDoctorId())
                .orElseThrow(() -> new ResourceNotFoundException("Doctor not found"));
        Patient patient = patientRepository.findById(request.getPatientId())
                .orElseThrow(() -> new ResourceNotFoundException("Patient not found"));

        if (appointmentRepository.existsByDoctorIdAndDateAndTimeSlot(doctor.getId(),
request.getDate(), request.getTimeSlot())) {
            log.warn("Slot already booked: doctorId={} date={} slot={}", doctor.getId(),
request.getDate(), request.getTimeSlot());
            throw new BadRequestException("Slot already booked");
        }

        Appointment appointment = new Appointment(request.getDate(), request.getTimeSlot(),
doctor, patient, AppointmentStatus.BOOKED);
        Appointment saved = appointmentRepository.save(appointment);
        log.info("Appointment created: id={} doctorId={} patientId={}", saved.getId(),
doctor.getId(), patient.getId());
        return mapToResponse(saved);
    }

    public List<AppointmentResponse> getAppointmentsByDoctor(Long doctorId) {
        log.debug("Retrieve appointments by doctor: {}", doctorId);
        List<Appointment> list = appointmentRepository.findByDoctorId(doctorId);
        return list.stream().map(this::mapToResponse).collect(Collectors.toList());
    }

    public List<AppointmentResponse> getAppointmentsByPatient(Long patientId) {
        log.debug("Retrieve appointments by patient: {}", patientId);
        List<Appointment> list = appointmentRepository.findByPatientId(patientId);
        return list.stream().map(this::mapToResponse).collect(Collectors.toList());
    }

    @Transactional
    public void cancelAppointment(Long appointmentId) {
        log.debug("Cancelling appointment: {}", appointmentId);
        Appointment appointment = appointmentRepository.findById(appointmentId)
                .orElseThrow(() -> new ResourceNotFoundException("Appointment not found"));
        if (appointment.getStatus() == AppointmentStatus.CANCELLED) {
            log.warn("Attempt to cancel already cancelled appointment: {}", appointmentId);
            throw new BadRequestException("Appointment already cancelled");
        }
        appointment.setStatus(AppointmentStatus.CANCELLED);
        appointmentRepository.save(appointment);
        log.info("Appointment cancelled: {}", appointmentId);
    }

    private AppointmentResponse mapToResponse(Appointment a) {
        String doctorName = a.getDoctor() != null ? a.getDoctor().getName() : null;
        String patientName = a.getPatient() != null ? a.getPatient().getName() : null;
        Long doctorId = a.getDoctor() != null ? a.getDoctor().getId() : null;
        Long patientId = a.getPatient() != null ? a.getPatient().getId() : null;
        return new AppointmentResponse(a.getId(), a.getDate(), a.getTimeSlot(),
                doctorId, doctorName, patientId, patientName, a.getStatus().name());
    }
}
```

---

11) `com.hms.service.BillService` (with logger)

```
java

package com.hms.service;

import com.hms.dto.BillRequest;
import com.hms.dto.BillResponse;
import com.hms.exception.ResourceNotFoundException;
import com.hms.model.Appointment;
import com.hms.model.Bill;
import com.hms.repository.AppointmentRepository;
import com.hms.repository.BillRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDate;
import java.util.List;
import java.util.stream.Collectors;

@Service
public class BillService {
    private static final Logger log = LoggerFactory.getLogger(BillService.class);

    @Autowired
    private BillRepository billRepository;

    @Autowired
    private AppointmentRepository appointmentRepository;

    public BillResponse generateBill(BillRequest request) {
        log.debug("Generating bill for appointmentId={} amount={}", request.getAppointmentId(),
request.getAmount());
        Appointment appointment = appointmentRepository.findById(request.getAppointmentId())
                .orElseThrow(() -> new ResourceNotFoundException("Appointment not found"));
        Bill bill = new Bill(request.getAmount(), LocalDate.now(), appointment);
        Bill saved = billRepository.save(bill);
        log.info("Bill saved: id={} appointmentId={}", saved.getId(), appointment.getId());
        return new BillResponse(saved.getId(), saved.getAmount(), saved.getBillDate(),
saved.getAppointment().getId());
    }
    public List<BillResponse> getBillsByPatient(Long patientId) {
        log.debug("Fetching bills for patientId={}", patientId);
        List<Bill> bills = billRepository.findByAppointmentPatientId(patientId);
        return bills.stream()
                .map(b -> new BillResponse(b.getId(), b.getAmount(), b.getBillDate(),
b.getAppointment().getId()))
                .collect(Collectors.toList());
    }
}
```

---

12) `com.hms.security.UserDetailsServiceImpl` (logging)

```java
package com.hms.security;

import com.hms.model.User;
import com.hms.repository.UserRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
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.core.authority.SimpleGrantedAuthority;
import org.springframework.stereotype.Service;

import java.util.Collections;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    private static final Logger log = LoggerFactory.getLogger(UserDetailsServiceImpl.class);

    @Autowired
    private UserRepository userRepository;

    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        log.debug("Loading user by username={}", username);
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> {
                    log.warn("User not found: {}", username);
                    return new UsernameNotFoundException("User not found: " + username);
                });

        return new org.springframework.security.core.userdetails.User(
                user.getUsername(),
                user.getPassword(),
                Collections.singleton(new SimpleGrantedAuthority("ROLE_" +
user.getRole().name()))
        );
    }
}
```

---

13) `com.hms.service.DataInitializer` (logging)

```java
package com.hms.service;

import com.hms.model.Role;
import com.hms.model.User;
import com.hms.repository.UserRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;

@Component
public class DataInitializer implements CommandLineRunner {
    private static final Logger log = LoggerFactory.getLogger(DataInitializer.class);

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Override
    public void run(String... args) {
        if (userRepository.findByUsername("admin").isEmpty()) {
            User admin = new User("admin", passwordEncoder.encode("admin123"), Role.ADMIN);
            userRepository.save(admin);
            log.info("Default ADMIN created: username=admin password=admin123");
        } else {
            log.info("Admin user already exists; skipping initializer.");
        }
    }
}
```

---

14) `com.hms.exception.GlobalExceptionHandler` (log exceptions)

```java
java


package com.hms.exception;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice
public class GlobalExceptionHandler {
    private static final Logger log = LoggerFactory.getLogger(GlobalExceptionHandler.class);

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ApiError> handleNotFound(ResourceNotFoundException ex) {
        log.warn("Resource not found: {}", ex.getMessage());
        ApiError error = new ApiError(HttpStatus.NOT_FOUND.value(), "Not Found",
ex.getMessage());
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(error);
    }

    @ExceptionHandler(BadRequestException.class)
    public ResponseEntity<ApiError> handleBadRequest(BadRequestException ex) {
        log.warn("Bad request: {}", ex.getMessage());
        ApiError error = new ApiError(HttpStatus.BAD_REQUEST.value(), "Bad Request",
ex.getMessage());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(error);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ApiError> handleGeneric(Exception ex) {
        log.error("Internal server error: ", ex);
        ApiError error = new ApiError(HttpStatus.INTERNAL_SERVER_ERROR.value(), "Error",
ex.getMessage());
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(error);
    }
}
```

---

15) `com.hms.config.SecurityConfig` (add logger)

```java
package com.hms.config;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
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.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.authentication.AuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfig
uration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.Customizer;
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.config.http.SessionCreationPolicy;

import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.util.List;

@Configuration
public class SecurityConfig {
    private static final Logger log = LoggerFactory.getLogger(SecurityConfig.class);

    @Autowired
    private UserDetailsService userDetailsService;

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationProvider authenticationProvider() {
        log.debug("Configuring DaoAuthenticationProvider");
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setPasswordEncoder(passwordEncoder());
        provider.setUserDetailsService(userDetailsService);
        return provider;
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config)
throws Exception {
        log.debug("Providing AuthenticationManager");
        return config.getAuthenticationManager();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        log.debug("Building security filter chain");
        http
            .csrf(csrf -> csrf.disable())
            .cors(Customizer.withDefaults())
            .sessionManagement(sm -> sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(
                    "/api/auth/**",
                    "/api/user/register-patient",
                    "/v3/api-docs/**",
                    "/swagger-ui/**",
                    "/swagger-ui.html"
                ).permitAll()
                .requestMatchers("/api/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated()
            )
            .httpBasic(Customizer.withDefaults());

        return http.build();
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration cfg = new CorsConfiguration();
        cfg.setAllowedOrigins(List.of(
            "http://127.0.0.1:5500",
            "http://localhost:5500",
            "http://localhost:3000",
            "http://localhost"
        ));
        cfg.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS"));
        cfg.setAllowedHeaders(List.of("Authorization", "Content-Type"));
        cfg.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", cfg);
        return source;
    }
}
```

Notes & Next Steps

- All classes now use SLF4J logging. If you prefer Lombok **`@Slf4j`** instead, I can switch them — but I used explicit **`LoggerFactory`** to avoid Lombok annotation dependency issues.
- I replaced the remaining **`RuntimeException`** in **`UserService`** with **`BadRequestException`** to keep the exception handling consistent.
- Make sure your DTOs and other classes (from previous message) are present — controllers expect the DTOs we defined earlier.
- Start the app and watch the logs; you'll see useful debug/info/warn/error messages. Example:

  ```
  arduino

  mvn spring-boot:run
  ```

- If you want, I can:
  - Convert logs to structured logging (MDC) for correlation IDs.
  - Add a logging configuration file (**`logback-spring.xml`**) to control log levels and formatting.
  - Replace HTTP Basic with JWT authentication (recommended for production).

Would you like me to (pick one):

1. Provide a **`logback-spring.xml`** with sensible defaults (console + file)?

2. Swap to Lombok `@Slf4j` everywhere to reduce boilerplate?
3. Add JWT-based auth scaffolding?

Tell me which and I'll add it immediately.

Provide a logback-spring.xml with sensible defaults (console + file)? is it necessary

Short answer: not strictly **necessary** — Spring Boot runs fine without a custom `logback-spring.xml` (it uses sensible defaults).
But adding one gives you **clear control** over log formats, file rotation, separate files for app vs. errors, and nicer console output for development. It's highly recommended for any non-trivial project (local dev + production).

Below I give a ready-to-drop `logback-spring.xml` with sensible defaults: console (human-friendly), rolling file with size+time rollover, separate `error` file, and easy log-level overrides via Spring properties. Put this file in `src/main/resources/logback-spring.xml`.

Why add it?

- Persist logs to file for debugging post-crash / server logs.
- Rotate logs (avoid disk fill).
- Different formats for console (readable) and file (structured).
- Control log levels for packages without recompiling.
- Useful for production/CI troubleshooting.

Drop-in `logback-spring.xml`

xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="true" scanPeriod="30 seconds">

    <!-- Allow overriding via application.properties / env -->
    <property name="LOG_PATH" value="${LOG_PATH:-./logs}" />
    <property name="LOG_FILE" value="${LOG_FILE:-app.log}" />
    <property name="MAX_FILE_SIZE" value="${LOG_MAX_FILE_SIZE:-10MB}" />
    <property name="MAX_HISTORY" value="${LOG_MAX_HISTORY:-30}" />
    <property name="TOTAL_SIZE_CAP" value="${LOG_TOTAL_SIZE_CAP:-1GB}" />

    <!-- Console pattern (developer-friendly) -->
    <springProperty scope="context" name="PID" source="PID" />
    <property name="CONSOLE_PATTERN" value="%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level
%logger{36} - %msg%n" />
    <property name="FILE_PATTERN" value="%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level [%X{requestId}]
%logger{36} - %msg%n" />

    <!-- Console appender -->
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>${CONSOLE_PATTERN}</pattern>
            <charset>UTF-8</charset>
        </encoder>
    </appender>

    <!-- Rolling file appender -->
    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${LOG_PATH}/${LOG_FILE}</file>

        <rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
            <!-- daily rollover + size cap per file -->
            <fileNamePattern>${LOG_PATH}/archived/app-%d{yyyy-MM-dd}.%i.log.gz</fileNamePattern>
            <maxFileSize>${MAX_FILE_SIZE}</maxFileSize>
            <maxHistory>${MAX_HISTORY}</maxHistory>
            <totalSizeCap>${TOTAL_SIZE_CAP}</totalSizeCap>
        </rollingPolicy>

        <encoder>
            <pattern>${FILE_PATTERN}</pattern>
            <charset>UTF-8</charset>
        </encoder>
    </appender>

    <!-- Separate appender for ERRORs (optional) -->
    <appender name="ERROR_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${LOG_PATH}/error.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${LOG_PATH}/archived/error-%d{yyyy-MM-dd}.log.gz</fileNamePattern>
            <maxHistory>${MAX_HISTORY}</maxHistory>
            <totalSizeCap>${TOTAL_SIZE_CAP}</totalSizeCap>
        </rollingPolicy>
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>ERROR</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
        <encoder>
            <pattern>${FILE_PATTERN}</pattern>
            <charset>UTF-8</charset>
        </encoder>
    </appender>

    <!-- Example: asynchronous wrapper (improves perf) -->
    <appender name="ASYNC_FILE" class="ch.qos.logback.classic.AsyncAppender">
        <appender-ref ref="FILE" />
        <appender-ref ref="ERROR_FILE" />
        <discardingThreshold>0</discardingThreshold>
    </appender>

    <!-- Root logger -->
    <root level="${LOG_LEVEL_ROOT:-INFO}">
        <appender-ref ref="CONSOLE" />
        <appender-ref ref="ASYNC_FILE" />
    </root>

    <!-- Package-specific overrides (tweak for noisy libs) -->
    <logger name="org.springframework" level="${LOG_LEVEL_SPRING:-INFO}" />
    <logger name="org.hibernate" level="${LOG_LEVEL_HIBERNATE:-WARN}" />
    <logger name="com.hms" level="${LOG_LEVEL_APP:-DEBUG}" />

</configuration>
```

How to use / configure

1. Save as `src/main/resources/logback-spring.xml`. Spring Boot will pick it up automatically.
2. Configure runtime overrides with environment variables or `application.properties`. Examples:

In `application.properties`:

```properties
LOG_PATH=./logs
LOG_FILE=hms-app.log
LOG_MAX_FILE_SIZE=20MB
LOG_MAX_HISTORY=14
LOG_TOTAL_SIZE_CAP=2GB

# Optional: adjust root/app package levels
LOG_LEVEL_ROOT=INFO
LOG_LEVEL_APP=DEBUG
LOG_LEVEL_HIBERNATE=WARN
```

Or with environment variables:

```bash
export LOG_PATH=/var/log/hms
export LOG_FILE=hms.log
```

3. After starting your app (`mvn spring-boot:run`), logs will appear on console and be written to `./logs/app.log`. Rotated files are placed in `./logs/archived/`.

Tips & best practices

- Ensure the `LOG_PATH` directory is writeable by the process (in prod use `/var/log/yourapp` or a mounted volume).
- Keep `MAX_FILE_SIZE` and `MAX_HISTORY` tuned so logs don't fill disk. `10MB–100MB` per file and `14–30` days history are common.
- Use `requestId` MDC in controllers/services to correlate logs across requests. If you want, I can show how to add a filter that sets `MDC.put("requestId", ...)` for each HTTP request.
- In containerized deployments, some teams prefer only console logs (stdout/err) and let the container engine capture/rotate logs. If you run in Docker/Kubernetes, you can omit file appenders and rely on centralized logging (ELK/Fluentd). The file appender is most useful for VM or bare-metal servers or where you want local persisted logs.

If you want, I can:

- Provide a small `logback-spring.xml` variant **for containerized environments** (console-only, JSON-friendly).
- Add example code to set a requestId in MDC for correlated logs.
- Add a `logback-spring.xml` that sends logs to a remote syslog or ELK stack (filebeat/Logstash) — needs endpoint details.

Which of those would you like next?

i will not add it

That’s perfectly fine, Samiksha 👍

You **don’t have to add `logback-spring.xml`** — Spring Boot already uses a **built-in Logback configuration** by default that:
✅ Prints logs neatly to the console (with timestamps and log levels)
✅ Supports log levels like **`INFO`**, **`WARN`**, **`ERROR`**, etc.
✅ Works automatically with **`@Slf4j`** (no extra setup needed)

So you can safely keep your project **simple** and just use logging like this anywhere:

```java
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

@Slf4j
@Service
public class DoctorService {
    public void example() {
        log.info("Doctor registration started...");
        log.warn("This is a warning example");
        log.error("Something went wrong!");
    }
}
```

You’ll see clean log output in your console like:

```yaml
2025-10-16T20:03:15.423  INFO 12345 --- [main] c.h.s.DoctorService : Doctor registration
started...
```

So ✅ you can skip the **`logback-spring.xml`** file completely — your backend is already production-ready for a local setup.

Would you like me to now show how to **add proper `@Slf4j` logging in all service and controller classes** (with messages for actions like doctor registration, patient booking, etc.)? That would make your logs meaningful and interview-ready.