Bookstore Management System - Full Project (Pure Code)

pom.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>
    <groupId>com.bookstore</groupId>
    <artifactId>bookstore</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>Bookstore Management System</name>

    <properties>
        <java.version>17</java.version>
        <spring.boot.version>3.1.4</spring.boot.version>
        <jjwt.version>0.9.1</jjwt.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <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>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>${jjwt.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </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>
    </dependencies>

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

application.properties

```
# Server
server.port=8080

# Datasource — change values to your local DB
spring.datasource.url=jdbc:mysql://localhost:3306/bookstore?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone
spring.datasource.username=root
spring.datasource.password=yourpassword

# JPA
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

# JWT secret and expiration
jwt.secret=ReplaceThisWithASecretKeyForJWTGeneration12345
jwt.expirationMs=86400000

# Swagger (if using springdoc/openapi)
springdoc.api-docs.path=/v3/api-docs
```

BookstoreApplication.java

```
package com.bookstore;

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

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

model/Book.java

```
package com.bookstore.model;

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

@Entity
@Table(name = "books")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    private String authors;

    private String genre;

    @Column(unique = true)
    private String isbn;

    private BigDecimal price;

    @Column(length = 1000)
    private String description;

    private Integer stockQuantity;

    private String imageUrl;

    public Book() {}

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }
    public String getAuthors() { return authors; }
    public void setAuthors(String authors) { this.authors = authors; }
    public String getGenre() { return genre; }
    public void setGenre(String genre) { this.genre = genre; }
    public String getIsbn() { return isbn; }
    public void setIsbn(String isbn) { this.isbn = isbn; }
    public BigDecimal getPrice() { return price; }
    public void setPrice(BigDecimal price) { this.price = price; }
    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }
    public Integer getStockQuantity() { return stockQuantity; }
    public void setStockQuantity(Integer stockQuantity) { this.stockQuantity = stockQuantity; }
    public String getImageUrl() { return imageUrl; }
    public void setImageUrl(String imageUrl) { this.imageUrl = imageUrl; }
}
```

model/Role.java

```
package com.bookstore.model;

public enum Role {
    ROLE_CUSTOMER,
    ROLE_ADMIN
}
```

model/User.java

```
package com.bookstore.model;

import jakarta.persistence.*;
import java.util.Set;

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @Column(unique = true)
    private String email;

    private String password;

    @ElementCollection(fetch = FetchType.EAGER)
    @Enumerated(EnumType.STRING)
    @CollectionTable(name = "user_roles")
    private Set<Role> roles;

    public 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 getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
    public Set<Role> getRoles() { return roles; }
    public void setRoles(Set<Role> roles) { this.roles = roles; }
}
```

model/OrderItem.java

```
package com.bookstore.model;

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

@Entity
@Table(name = "order_items")
public class OrderItem {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    private Book book;

    private Integer quantity;

    private BigDecimal price;

    public OrderItem() {}

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public Book getBook() { return book; }
    public void setBook(Book book) { this.book = book; }
    public Integer getQuantity() { return quantity; }
    public void setQuantity(Integer quantity) { this.quantity = quantity; }
    public BigDecimal getPrice() { return price; }
    public void setPrice(BigDecimal price) { this.price = price; }
}
```

model/OrderStatus.java

```
package com.bookstore.model;

public enum OrderStatus {
    PENDING,
    SHIPPED,
    DELIVERED,
    CANCELLED
}
```

model/OrderEntity.java

```
package com.bookstore.model;

import jakarta.persistence.*;
import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.List;

@Entity
@Table(name = "orders")
public class OrderEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String customerName;

    private String customerEmail;

    @OneToMany(cascade = CascadeType.ALL)
    private List<OrderItem> items;

    private BigDecimal totalPrice;

    @Enumerated(EnumType.STRING)
    private OrderStatus status;

    private boolean paymentCompleted;

    private LocalDateTime createdAt;

    public OrderEntity() {
        this.createdAt = LocalDateTime.now();
        this.status = OrderStatus.PENDING;
    }

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getCustomerName() { return customerName; }
    public void setCustomerName(String customerName) { this.customerName = customerName; }
    public String getCustomerEmail() { return customerEmail; }
    public void setCustomerEmail(String customerEmail) { this.customerEmail = customerEmail; }
    public List<OrderItem> getItems() { return items; }
    public void setItems(List<OrderItem> items) { this.items = items; }
    public BigDecimal getTotalPrice() { return totalPrice; }
    public void setTotalPrice(BigDecimal totalPrice) { this.totalPrice = totalPrice; }
    public OrderStatus getStatus() { return status; }
    public void setStatus(OrderStatus status) { this.status = status; }
    public boolean isPaymentCompleted() { return paymentCompleted; }
    public void setPaymentCompleted(boolean paymentCompleted) { this.paymentCompleted = paymentCompleted; }
    public LocalDateTime getCreatedAt() { return createdAt; }
    public void setCreatedAt(LocalDateTime createdAt) { this.createdAt = createdAt; }
}
```

repository/BookRepository.java

```
package com.bookstore.repository;

import com.bookstore.model.Book;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {
    Page<Book> findByTitleContainingIgnoreCaseOrAuthorsContainingIgnoreCase(String title, String authors, Pageable p
}
```

repository/UserRepository.java

```
package com.bookstore.repository;

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

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);
}
```

repository/OrderRepository.java

```
package com.bookstore.repository;

import com.bookstore.model.OrderEntity;
import org.springframework.data.jpa.repository.JpaRepository;

public interface OrderRepository extends JpaRepository<OrderEntity, Long> {
}
```

repository/OrderItemRepository.java

```
package com.bookstore.repository;

import com.bookstore.model.OrderItem;
import org.springframework.data.jpa.repository.JpaRepository;

public interface OrderItemRepository extends JpaRepository<OrderItem, Long> {
}
```

security/JwtUtil.java

```
package com.bookstore.security;

import io.jsonwebtoken.*;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import java.util.Date;
import java.util.function.Function;

@Component
public class JwtUtil {

    @Value("${jwt.secret}")
    private String secret;

    @Value("${jwt.expirationMs}")
    private long jwtExpirationMs;

    public String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + jwtExpirationMs))
                .signWith(SignatureAlgorithm.HS256, secret)
                .compact();
    }

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
        return claimsResolver.apply(claims);
    }

    public boolean validateToken(String token, String username) {
        final String extractedUser = extractUsername(token);
        return (extractedUser.equals(username) && !extractExpiration(token).before(new Date()));
    }
}
```

security/JwtAuthenticationFilter.java

```
package com.bookstore.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;
import com.bookstore.service.CustomUserDetailsService;
import java.io.IOException;

public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtUtil jwtUtil;
    private final CustomUserDetailsService userDetailsService;

    public JwtAuthenticationFilter(JwtUtil jwtUtil, CustomUserDetailsService userDetailsService) {
        this.jwtUtil = jwtUtil;
        this.userDetailsService = userDetailsService;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {
        try {
            String jwt = parseJwt(request);
            if (jwt != null) {
                String username = jwtUtil.extractUsername(jwt);
                var userDetails = userDetailsService.loadUserByUsername(username);
                if (jwtUtil.validateToken(jwt, userDetails.getUsername())) {
                    UsernamePasswordAuthenticationToken authentication =
                            new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities()
                    authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                    SecurityContextHolder.getContext().setAuthentication(authentication);
                }
            }
        } catch (Exception ex) {
            // ignore or log
        }
        filterChain.doFilter(request, response);
    }

    private String parseJwt(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");
        if (StringUtils.hasText(headerAuth) && headerAuth.startsWith("Bearer ")) {
            return headerAuth.substring(7);
        }
        return null;
    }
}
```

service/CustomUserDetailsService.java

```
package com.bookstore.service;

import com.bookstore.model.User;
import com.bookstore.repository.UserRepository;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.*;
import org.springframework.stereotype.Service;
import java.util.Set;
import java.util.stream.Collectors;

@Service
public class CustomUserDetailsService implements UserDetailsService {

    private final UserRepository userRepository;

    public CustomUserDetailsService(UserRepository repo) {
        this.userRepository = repo;
    }

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(email)
                .orElseThrow(() -> new UsernameNotFoundException("User not found with email: " + email));
        Set<GrantedAuthority> authorities = user.getRoles().stream()
                .map(role -> new SimpleGrantedAuthority(role.name()))
                .collect(Collectors.toSet());

        return new org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(), authoriti
    }
}
```

security/SecurityConfig.java

```
package com.bookstore.security;

import com.bookstore.service.CustomUserDetailsService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.*;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.*;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;

@Configuration
@EnableMethodSecurity
public class SecurityConfig {

    private final JwtUtil jwtUtil;
    private final CustomUserDetailsService userDetailsService;

    public SecurityConfig(JwtUtil jwtUtil, CustomUserDetailsService userDetailsService) {
        this.jwtUtil = jwtUtil;
        this.userDetailsService = userDetailsService;
    }

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

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

    @Bean
    public JwtAuthenticationFilter jwtAuthenticationFilter() {
        return new JwtAuthenticationFilter(jwtUtil, userDetailsService);
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http = http.csrf().disable()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and();

        http.authorizeHttpRequests()
                .requestMatchers("/api/auth/**", "/v3/api-docs/**", "/swagger-ui/**", "/swagger-ui.html").permitAll
                .requestMatchers("/api/books/**").permitAll()
                .requestMatchers("/api/orders/**").authenticated()
                .anyRequest().authenticated();

        http.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);

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

dto/AuthRequest.java

```
package com.bookstore.dto;

public class AuthRequest {
    private String email;
    private String password;

    public AuthRequest() {}

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
}
```

dto/AuthResponse.java

```
package com.bookstore.dto;

public class AuthResponse {
    private String token;

    public AuthResponse() {}

    public AuthResponse(String token) { this.token = token; }

    public String getToken() { return token; }
    public void setToken(String token) { this.token = token; }
}
```

dto/RegisterRequest.java

```
package com.bookstore.dto;

import java.util.Set;

public class RegisterRequest {
    private String name;
    private String email;
    private String password;
    private Set<String> roles;

    public RegisterRequest() {}

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
    public Set<String> getRoles() { return roles; }
    public void setRoles(Set<String> roles) { this.roles = roles; }
}
```

service/BookService.java

```
package com.bookstore.service;

import com.bookstore.model.Book;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;

public interface BookService {
    Page<Book> getAllBooks(String search, Pageable pageable);
    Book getBookById(Long id);
    Book addBook(Book book);
    Book updateBook(Long id, Book book);
    void deleteBook(Long id);
}
```

service/impl/BookServiceImpl.java

```
package com.bookstore.service.impl;

import com.bookstore.model.Book;
import com.bookstore.repository.BookRepository;
import com.bookstore.service.BookService;
import org.springframework.data.domain.*;
import org.springframework.stereotype.Service;
import java.util.Optional;

@Service
public class BookServiceImpl implements BookService {

    private final BookRepository repo;

    public BookServiceImpl(BookRepository repo) { this.repo = repo; }

    @Override
    public Page<Book> getAllBooks(String search, Pageable pageable) {
        if (search == null || search.isBlank()) {
            return repo.findAll(pageable);
        }
        return repo.findByTitleContainingIgnoreCaseOrAuthorsContainingIgnoreCase(search, search, pageable);
    }

    @Override
    public Book getBookById(Long id) {
        return repo.findById(id).orElseThrow(() -> new RuntimeException("Book not found"));
    }

    @Override
    public Book addBook(Book book) {
        return repo.save(book);
    }

    @Override
    public Book updateBook(Long id, Book book) {
        Book existing = getBookById(id);
        existing.setTitle(book.getTitle());
        existing.setAuthors(book.getAuthors());
        existing.setDescription(book.getDescription());
        existing.setGenre(book.getGenre());
        existing.setIsbn(book.getIsbn());
        existing.setPrice(book.getPrice());
        existing.setStockQuantity(book.getStockQuantity());
        existing.setImageUrl(book.getImageUrl());
        return repo.save(existing);
    }

    @Override
    public void deleteBook(Long id) {
        repo.deleteById(id);
    }
}
```

service/OrderService.java

```
package com.bookstore.service;

import com.bookstore.model.OrderEntity;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;

public interface OrderService {
    OrderEntity placeOrder(OrderEntity order);
    OrderEntity getOrderById(Long id);
    Page<OrderEntity> getAllOrders(Pageable pageable);
    OrderEntity updateOrderStatus(Long id, String status);
}
```

service/impl/OrderServiceImpl.java

```
package com.bookstore.service.impl;

import com.bookstore.model.*;
import com.bookstore.repository.*;
import com.bookstore.service.OrderService;
import org.springframework.data.domain.*;
import org.springframework.stereotype.Service;
import java.math.BigDecimal;
import java.util.List;

@Service
public class OrderServiceImpl implements OrderService {

    private final OrderRepository orderRepo;
    private final BookRepository bookRepo;

    public OrderServiceImpl(OrderRepository orderRepo, BookRepository bookRepo) {
        this.orderRepo = orderRepo;
        this.bookRepo = bookRepo;
    }

    @Override
    public OrderEntity placeOrder(OrderEntity order) {
        BigDecimal total = BigDecimal.ZERO;
        List<OrderItem> items = order.getItems();
        for (OrderItem item : items) {
            Book book = bookRepo.findById(item.getBook().getId())
                    .orElseThrow(() -> new RuntimeException("Book not found: " + item.getBook().getId()));
            if (book.getStockQuantity() < item.getQuantity()) {
                throw new RuntimeException("Insufficient stock for book: " + book.getTitle());
            }
            item.setPrice(book.getPrice());
            book.setStockQuantity(book.getStockQuantity() - item.getQuantity());
            bookRepo.save(book);
            total = total.add(book.getPrice().multiply(java.math.BigDecimal.valueOf(item.getQuantity())));
        }
        order.setTotalPrice(total);
        order.setStatus(OrderStatus.PENDING);
        order.setPaymentCompleted(false);
        return orderRepo.save(order);
    }

    @Override
    public OrderEntity getOrderById(Long id) {
        return orderRepo.findById(id).orElseThrow(() -> new RuntimeException("Order not found"));
    }

    @Override
    public Page<OrderEntity> getAllOrders(Pageable pageable) {
        return orderRepo.findAll(pageable);
    }

    @Override
    public OrderEntity updateOrderStatus(Long id, String status) {
        OrderEntity order = getOrderById(id);
        order.setStatus(OrderStatus.valueOf(status));
        return orderRepo.save(order);
    }
}
```

controller/BookController.java

```
package com.bookstore.controller;

import com.bookstore.model.Book;
import com.bookstore.service.BookService;
import org.springframework.data.domain.*;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/books")
public class BookController {

    private final BookService service;

    public BookController(BookService service) { this.service = service; }

    @GetMapping
    public ResponseEntity<Page<Book>> getAllBooks(
            @RequestParam(value = "search", required = false) String search,
            @RequestParam(value = "page", defaultValue = "0") int page,
            @RequestParam(value = "size", defaultValue = "10") int size) {
        Page<Book> books = service.getAllBooks(search, PageRequest.of(page, size));
        return ResponseEntity.ok(books);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Book> getBook(@PathVariable Long id) {
        return ResponseEntity.ok(service.getBookById(id));
    }

    @PostMapping
    public ResponseEntity<Book> addBook(@RequestBody Book book) {
        return ResponseEntity.status(201).body(service.addBook(book));
    }

    @PutMapping("/{id}")
    public ResponseEntity<Book> updateBook(@PathVariable Long id, @RequestBody Book book) {
        return ResponseEntity.ok(service.updateBook(id, book));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> deleteBook(@PathVariable Long id) {
        service.deleteBook(id);
        return ResponseEntity.noContent().build();
    }
}
```

controller/AuthController.java

```
package com.bookstore.controller;

import com.bookstore.dto.*;
import com.bookstore.model.*;
import com.bookstore.repository.UserRepository;
import com.bookstore.security.JwtUtil;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.*;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.web.bind.annotation.*;
import java.util.Set;
import java.util.stream.Collectors;

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

    private final AuthenticationManager authManager;
    private final UserRepository userRepository;
    private final BCryptPasswordEncoder passwordEncoder;
    private final JwtUtil jwtUtil;

    public AuthController(AuthenticationManager authManager,
                          UserRepository userRepository,
                          BCryptPasswordEncoder passwordEncoder,
                          JwtUtil jwtUtil) {
        this.authManager = authManager;
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
        this.jwtUtil = jwtUtil;
    }

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody RegisterRequest req) {
        if (userRepository.findByEmail(req.getEmail()).isPresent()) {
            return ResponseEntity.badRequest().body("Email already in use");
        }
        User user = new User();
        user.setName(req.getName());
        user.setEmail(req.getEmail());
        user.setPassword(passwordEncoder.encode(req.getPassword()));
        Set<Role> roles = req.getRoles().stream()
                .map(r -> r.equalsIgnoreCase("admin") ? Role.ROLE_ADMIN : Role.ROLE_CUSTOMER)
                .collect(Collectors.toSet());
        user.setRoles(roles);
        userRepository.save(user);
        return ResponseEntity.status(201).body("User registered");
    }

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody AuthRequest req) {
        try {
            Authentication authentication = authManager.authenticate(
                    new UsernamePasswordAuthenticationToken(req.getEmail(), req.getPassword()));
            String token = jwtUtil.generateToken(req.getEmail());
            return ResponseEntity.ok(new AuthResponse(token));
        } catch (BadCredentialsException ex) {
            return ResponseEntity.status(401).body("Invalid credentials");
        }
    }
}
```

controller/OrderController.java

```
package com.bookstore.controller;

import com.bookstore.model.OrderEntity;
import com.bookstore.service.OrderService;
import org.springframework.data.domain.*;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/orders")
public class OrderController {

    private final OrderService service;

    public OrderController(OrderService service) { this.service = service; }

    @PostMapping
    @PreAuthorize("hasAuthority('ROLE_CUSTOMER') or hasAuthority('ROLE_ADMIN')")
    public ResponseEntity<OrderEntity> placeOrder(@RequestBody OrderEntity order) {
        return ResponseEntity.status(201).body(service.placeOrder(order));
    }

    @GetMapping("/{id}")
    public ResponseEntity<OrderEntity> getOrder(@PathVariable Long id) {
        return ResponseEntity.ok(service.getOrderById(id));
    }

    @GetMapping
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public ResponseEntity<Page<OrderEntity>> getAllOrders(
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "10") int size) {
        return ResponseEntity.ok(service.getAllOrders(PageRequest.of(page, size)));
    }

    @PutMapping("/{id}/status")
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public ResponseEntity<OrderEntity> updateStatus(@PathVariable Long id, @RequestParam String status) {
        return ResponseEntity.ok(service.updateOrderStatus(id, status));
    }
}
```

data.sql

```
INSERT INTO books (title, authors, genre, isbn, price, description, stock_quantity, image_url) VALUES
('The Great Gatsby', 'F. Scott Fitzgerald', 'Classic', '9780743273565', 499.00, 'A classic novel.', 20, ''),
('1984', 'George Orwell', 'Dystopian', '9780451524935', 399.00, 'Dystopian novel.', 15, '');
```