Java实训

第 1 部分：项目概述与技术栈 (Project Overview & Technology Stack)

在开始安装任何工具之前，我们先明确这个项目的**目标**和我们将要使用的**核心技术**。

1.1 项目目标

我们将从零开始，手把手搭建一个功能完备的**全栈（前后端分离）**网上二手书交易网站。

- **核心功能**：分为“买卖模块”（用户浏览、发布、加购物车、下单）和“管理模块”（管理员查看统计、管理用户和书籍）。
- **核心体验**：实现用户认证、图片上传、全局搜索和动态导航栏。
- **视觉风格**：借鉴亚马逊的 UI 布局，实现一个专业、整洁的吸顶导航栏和卡片式商品展示。

1.2 技术栈 (Technology Stack)

我们将使用一套现代企业开发中非常流行的技术栈：

后端 (Backend)

- **核心框架 (Spring Boot 3)**：使用 Java 17 和 Spring Boot 3 作为后端的主体框架，快速构建强大的 RESTful API 接口。
- **数据库访问 (Spring Data JPA / Hibernate)**：使用 JPA (Java Persistence API) 来进行“对象关系映射”(ORM)，让我们能用 Java 对象（`Book`, `User`）来操作数据库表，而不需要手写复杂的 SQL。
- **安全与认证 (Spring Security + JWT)**：使用 Spring Security 来保护我们的 API。我们将实现基于 **JWT (JSON Web Token)** 的无状态认证，处理用户登录、注册，并通过路由守卫实现基于角色的权限控制（普通用户 vs. 管理员）。
- **数据库 (MySQL)**：作为我们的关系型数据库，用来持久化存储所有用户、书籍、订单等数据。
- **构建工具 (Maven)**：用来管理后端项目的所有依赖库（如 Spring Web, Spring Data JPA, JWT 库等）并打包项目。
- **文件处理**：实现后端 API 接收前端上传的图片文件，并将其存储在服务器的文件系统中。

前端 (Frontend)

- **核心框架 (Vue.js 3)**：使用 Vue 3 (通过 Vue CLI 创建) 作为前端的声明式、组件化的 UI 框架。
- **路由 (Vue Router 4)**：负责前端的页面跳转（如 `/`，`/login`，`/admin/dashboard`）和页面级的权限控制（导航守卫 `beforeEach`）。
- **状态管理 (Pinia)**：使用 Vue 3 官方推荐的 Pinia 库，作为全局状态管理器。我们将用它来**响应式地**管理用户的登录状态（Token 和用户信息），解决导航栏自动更新的问题。
- **HTTP 请求 (Axios)**：一个强大的、基于 Promise 的 HTTP 客户端，负责前端所有与后端 API 的数据通信（登录、获取书籍、发布书籍等）。
- **数据可视化 (Chart.js)**：我们将使用 `Chart.js` 和 `vue-chartjs` 包装器，在管理后台的数据看板上绘制柱状图，实现数据可视化。
- **CSS 样式**：我们将手写**自定义 CSS**（在 Vue 的 `<style scoped>` 块中）来实现类似亚马逊的布局、卡片和吸顶导航栏，不依赖大型 UI 库。

第 2 部分：后端项目搭建 (Spring Boot Setup)

在这一部分，我们将使用 IntelliJ IDEA 创建 Spring Boot 后端项目，配置所有必需的依赖项（`pom.xml`），并设置数据库和安全密钥（`application.properties`）。

2.1 使用 Spring Initializr 创建项目

1. 打开 **IntelliJ IDEA** (Ultimate 版)。
2. 在欢迎界面，点击 **"New Project" (新建项目)**。
3. 在左侧菜单中，选择 **"Spring Initializr"**。
4. 填写项目元数据：
    - **Name (名称)**: `secondhand-book-backend` (这是你的项目文件夹名)
    - **Location (位置)**: 选择一个你喜欢的位置（比如桌面）。
    - **Type (类型)**: 选择 **"Maven"**。
    - **Group (组织)**: `com.example` (或者你自己的域名)
    - **Artifact (构件)**: `secondhand-book-backend` (这会是 `.jar` 包的名称)
    - **Package name (包名)**: `com.example.secondhandbookbackend`
    - **Java Version (Java 版本)**: **"17"**

5. 点击 **"Next" (下一步)**。
6. **【关键】添加依赖 (Dependencies)**：在 "Dependencies" 页面，搜索并添加以下 **7 个** 依赖项：
   - `Spring Web`：用于构建 RESTful API 接口。
   - `Spring Data JPA`：用于和数据库交互 (JPA 规范)。
   - `Spring Security`：用于处理认证和权限。
   - `MySQL Driver`：数据库驱动，让 Java 能连接到 MySQL。
   - `Lombok`：一个工具，帮我们自动生成 getter, setter, 构造函数等，让代码更简洁。
   - `Java JWT (io.jsonwebtoken:jjwt-api)`：用于创建和验证 JWT (JSON Web Token)。
   - `Java JWT Impl (io.jsonwebtoken:jjwt-impl)`：JWT 的实现。
   - `Java JWT Jackson (io.jsonwebtoken:jjwt-jackson)`：用于 JWT 的 JSON 转换。
7. 点击 **"Create" (创建)**。IntelliJ IDEA 会自动创建项目并下载所有依赖。

2.2 配置 `pom.xml` (项目依赖)

IntelliJ 自动生成的 `pom.xml` 可能缺少 JWT 依赖的完整运行时包。请用下面的**完整代码**覆盖你项目根目录下的 `pom.xml` 文件。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://w
ww.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://ma
ven.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>3.4.0</version> <relativePath/> </parent>

    <groupId>com.example</groupId>
    <artifactId>secondhand-book-backend</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>secondhand-book-backend</name>
    <description>Backend for Secondhand Book Trading Website</descripti
```

```
on>

  <properties>
    <java.version>17</java.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>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </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>io.jsonwebtoken</groupId>
      <artifactId>jjwt-api</artifactId>
      <version>0.12.6</version>
    </dependency>
    <dependency>
      <groupId>io.jsonwebtoken</groupId>
      <artifactId>jjwt-impl</artifactId>
      <version>0.12.6</version>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>io.jsonwebtoken</groupId>
      <artifactId>jjwt-jackson</artifactId>
      <version>0.12.6</version>
      <scope>runtime</scope>
    </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.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>
```

2.3 配置 application.properties (项目核心配置)

这是 Spring Boot 的核心配置文件。

1. 打开 src/main/resources/application.properties 文件。
2. 用下面的**完整配置**替换掉文件的全部内容。

```
# 1. 服务器配置
# ----------------------------------------
# 配置 Spring Boot 内置 Tomcat 服务器的运行端口
# 我们用 8080 作为后端端口 (前端将使用 8081)
server.port=8080

# 2. 数据库 (DataSource) 配置
# ----------------------------------------
# MySQL 数据库的连接 URL
# a. "jdbc:mysql://localhost:3306" 是数据库服务器地址和端口
# b. "secondhand_book_db" 是我们将要创建的数据库名称
# c. "useSSL=false..." 是为了禁用 SSL 并设置时区，避免警告
spring.datasource.url=jdbc:mysql://localhost:3306/secondhand_book_db?
useSSL=false&serverTimezone=UTC&allowPublicKeyRetrieval=true
# 数据库用户名 (请根据你的 MySQL 设置修改)
spring.datasource.username=root
# 数据库密码 (请根据你的 MySQL 设置修改)
spring.datasource.password=root
# 数据库驱动程序类
```

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# 3. JPA 和 Hibernate 配置
# ----------------------------------------
# "spring.jpa.hibernate.ddl-auto" 是 JPA 最强大的功能之一
# 它告诉 Hibernate (JPA的实现) 如何处理数据库表结构
#   - "create": 启动时删除所有表，然后重建 (适合早期开发)
#   - "update": 启动时检查 Java 实体类和数据库表，只添加缺失的表或列 (适合本项目)
#   - "validate": 启动时检查，如果表结构不匹配就报错
#   - "none": 不做任何操作 (适合生产环境)
spring.jpa.hibernate.ddl-auto=update

# 在控制台打印 Hibernate 正在执行的 SQL 语句，方便调试
spring.jpa.show-sql=true

# 可选：格式化打印出来的 SQL 语句
spring.jpa.properties.hibernate.format_sql=true

# 4. JWT 安全密钥配置
# ----------------------------------------
# 这是我们用来签名 JWT Token 的【密钥】
# 它【必须】是一个 Base64 编码的、足够长的随机字符串
# 【重要】：你必须使用你自己的密钥！
# (你可以使用在线 Base64 编码工具，输入一长串随机字符来生成它)
# (这个示例密钥是我们之前调试时使用的，长度为 64 字节)
secondhandbook.app.jwtSecret=MWQ2Njk0ZGQtYjI1MS00YjJlLWI3MjMtN2NhM2I5NDVlYjQwMWQ2Njk0ZGQtYjI1MS00YjJlLWI3MjMtN2NhM2I5NDVlYjQw

# Token 过期时间 (毫秒)
# 86400000 毫秒 = 24 小时
secondhandbook.app.jwtExpirationMs=86400000

# 5. 文件上传配置
# ----------------------------------------
# 定义一个相对路径，用于存储用户上传的封面图片
```

```
# "my-uploads" 会在项目根目录下 (与 src 同级)
file.upload-dir=my-uploads
```

2.4 在 Navicat 中创建数据库

1. 打开 **Navicat** (或 MySQL Workbench)。
2. 连接到你的本地 MySQL 服务器（使用 `root` 和你设置的密码）。
3. 在连接上右键，选择 **"新建数据库..." (New Database...)**。
4. **数据库名称 (Database Name)**: `secondhand_book_db`
5. **字符集 (Charset)**: 选择 `utf8mb4` (支持表情符号等)
6. **排序规则 (Collation)**: 选择 `utf8mb4_unicode_ci`
7. 点击 **"确定" (OK)**。

章节总结：

在完成这一部分后，你已经有了一个可以**运行**的 Spring Boot 后端项目。它已经配置好了所有依赖（`pom.xml`）和数据库连接（`application.properties`），并且在数据库中也有了一个空的 `secondhand_book_db`。

第 3 部分：后端 - 数据库模型 (Entities)

在这一部分，我们将创建 Java 类来“映射”到数据库表。得益于 Spring Data JPA (Hibernate)，我们只需要写 Java 代码，Hibernate 就会自动帮我们在 MySQL 里创建对应的表。

所有这些文件都必须放在 `com.example.secondhandbookbackend.model` 包下。

3.1 创建 `model` 包

1. 在 IntelliJ IDEA 左侧的项目结构中，找到 `src/main/java/com/example/secondhandbookbackend`。
2. **右键点击**它 → **New** → **Package**。
3. 输入 `model` 并按回车。

3.2 用户与角色模型

我们需要两个实体来管理用户：`User` 和 `Role`。

3.2.1 `ERole.java` (枚举)

- **作用**：定义系统中仅有的两种角色类型。

```
package com.example.secondhandbookbackend.model;

// 枚举 (Enum) 是一种特殊的 Java 类，用于定义一组常量。
public enum ERole {
    ROLE_USER,  // 普通用户
    ROLE_ADMIN  // 管理员
}
```

3.2.2 `Role.java` (角色实体)

- **作用**：对应数据库中的 `roles` 表。

```
package com.example.secondhandbookbackend.model;

import jakarta.persistence.*; // JPA 标准注解
import lombok.Getter;
import lombok.Setter;

@Entity // 告诉 JPA 这是一个实体类，需要映射到数据库表
@Table(name = "roles") // 指定数据库中的表名为 "roles"
@Getter // Lombok 注解：自动生成所有字段的 get 方法
@Setter // Lombok 注解：自动生成所有字段的 set 方法
public class Role {
    @Id // 标记这个字段为主键
    @GeneratedValue(strategy = GenerationType.IDENTITY) // 主键生成策略：由数据库自动递增 (AUTO_INCREMENT)
    private Integer id;

    @Enumerated(EnumType.STRING) // 告诉 JPA 把枚举以字符串形式 ("ROLE_USER") 存入数据库，而不是数字 (0)
    @Column(length = 20, unique = true) // 设置列属性：最大长度 20，值必须唯一
    private ERole name;

    // JPA 需要一个无参构造函数
    public Role() {
    }
```

```
    public Role(ERole name) {
        this.name = name;
    }
}
```

3.2.3 `User.java` (用户实体)

- **作用**：对应 `users` 表，存储用户的登录信息。

```
package com.example.secondhandbookbackend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "users",
      uniqueConstraints = { // 定义唯一约束，防止重复注册
          @UniqueConstraint(columnNames = "username"),
          @UniqueConstraint(columnNames = "email")
      })
@Getter
@Setter
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false) // 不能为空
    private String username;

    @Column(nullable = false)
    private String email;

    @Column(nullable = false)
    private String password; // 存储加密后的哈希密码
```

```
    // 定义多对多关系：一个用户可以有多个角色，一个角色也可以赋予多个用户
    // fetch = FetchType.EAGER: 加载用户时，立即顺便加载他的角色信息
    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(  name = "user_roles", // 中间表的名称
                joinColumns = @JoinColumn(name = "user_id"), // 中间表中指向 users 表的外键
                inverseJoinColumns = @JoinColumn(name = "role_id")) // 中间表中指向 roles 表的外键
    private Set<Role> roles = new HashSet<>();

    public User() {
    }

    // 一个方便的构造函数，用于注册时创建用户
    public User(String username, String email, String password) {
        this.username = username;
        this.email = email;
        this.password = password;
    }
}
```

3.3 书籍与购物车模型

3.3.1 `EBookStatus.java` (枚举)

- **作用**：定义书籍的三种状态。

```
package com.example.secondhandbookbackend.model;

public enum EBookStatus {
    FOR_SALE, // 在售：所有人可见，可购买
    LOCKED,   // 已锁定：已被某人加入购物车，其他人不可见
    SOLD      // 已售出：已被购买，作为历史记录存在
}
```

3.3.2 `Book.java` (书籍实体)

- **作用**：对应 `books` 表，这是我们最核心的商品数据。

```
package com.example.secondhandbookbackend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

@Entity
@Table(name = "books")
@Getter
@Setter
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    private String author;

    @Column(nullable = false)
    private Double price;

    @Lob // Large Object，告诉数据库这个字段可能存储大量文本 (在 MySQL 中映射为 TEXT 类型)
    private String description;

    // 存储封面图片的 URL 地址 (例如: "http://localhost:8080/images/xxx.jpg")
    private String imageUrl;

    @Enumerated(EnumType.STRING)
    private EBookStatus status;

    // 多对一关系：多本书可以属于同一个卖家 (User)
    // fetch = FetchType.LAZY: 性能优化，默认不加载卖家信息，只有需要时才查数据库
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "seller_id", nullable = false)
```

```
    private User seller;

    public Book() {
    }
}
```

3.3.3 CartItem.java (购物车项实体)

- **作用**：对应 cart_items 表，连接用户和他们想买的书。

```
package com.example.secondhandbookbackend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

@Entity
@Table(name = "cart_items")
@Getter
@Setter
public class CartItem {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // 一对一：因为一本二手书是唯一的，它同时只能在一个人的购物车里
    @OneToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "book_id", nullable = false)
    private Book book;

    // 多对一：一个用户可以有多个购物车项
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    public CartItem() {
    }

    public CartItem(Book book, User user) {
```

```
        this.book = book;
        this.user = user;
    }
}
```

3.4 订单模型

3.4.1 `Order.java` (主订单实体)

- **作用**：对应 `orders` 表，记录一次购买行为的总览。

```
package com.example.secondhandbookbackend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.time.LocalDateTime;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "orders") // "order" 是 SQL 关键字，所以表名用复数 "order
s"
@Getter
@Setter
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "buyer_id", nullable = false)
    private User buyer; // 买家

    @Column(nullable = false)
    private LocalDateTime orderDate; // 下单时间

    @Column(nullable = false)
```

```
    private Double totalPrice; // 订单总金额

    // 一对多：一个订单包含多个订单项
    // cascade = CascadeType.ALL: 对订单的操作（如保存、删除）会级联到所有订单项
    @OneToMany(mappedBy = "order", cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    private Set<OrderItem> items = new HashSet<>();

    public Order() {
        this.orderDate = LocalDateTime.now(); // 默认下单时间为当前时间
    }
}
```

3.4.2 `OrderItem.java` (订单项实体)

- **作用**：对应 `order_items` 表，记录订单里具体买了哪些书，以及当时的价格。

```
package com.example.secondhandbookbackend.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

@Entity
@Table(name = "order_items")
@Getter
@Setter
public class OrderItem {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "order_id", nullable = false)
    private Order order; // 所属的主订单

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "book_id", nullable = false)
```

```
    private Book book; // 购买的书籍

    @Column(nullable = false)
    private Double priceAtPurchase; // 记录购买时的价格 (防止未来书价变动影响历史订单)

    public OrderItem() {
    }
}
```

阶段性验证 (非常重要！)

1. 确保以上 **8 个文件** 都已正确创建在 `model` 包下。
2. **运行项目**：点击 IntelliJ IDEA 的运行按钮 (▶)。
3. **观察日志**：你应该能看到一系列 `Hibernate: create table ...` 的日志。
4. **检查数据库**：打开 **Navicat**，刷新 `secondhand_book_db` 数据库。你应该能看到 **6 张表**：`books`，`cart_items`，`orders`，`order_items`，`roles`，`users`，以及一张自动生成的中间表 `user_roles`。

请完成验证，如果数据库表成功创建，我们就进入【第 4 部分：后端 - 核心安全】。

第 4 部分：后端 - 核心安全 (Spring Security & JWT)

在这一部分，我们将实现一套基于 JWT (JSON Web Token) 的无状态认证系统。

- **目标**：让用户能注册、登录，获取一个“令牌”(Token)。之后用户访问“需要登录”的接口（如加购物车）时，必须带上这个令牌。
- **核心组件**：
    1. `Repository`：用于从数据库查找用户。
    2. `UserDetails` 服务：告诉 Spring Security 如何理解我们的 `User` 实体。
    3. `JwtUtils`：负责生成和验证令牌的工具。
    4. `AuthTokenFilter`：每一笔请求的“安检员”。
    5. `WebSecurityConfig`：安全系统的总指挥（配置放行规则、CORS等）。

4.1 创建 Repositories (数据仓库)

我们需要先创建用于操作数据库的接口。

1. 在 `com.example.secondhandbookbackend` 下新建包 `repository`。
2. 创建以下 3 个接口文件。

4.1.1 UserRepository.java

```
package com.example.secondhandbookbackend.repository;

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

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    // Spring Data JPA 会自动根据方法名生成 SQL 查询
    Optional<User> findByUsername(String username); // 用于登录时查找用户
    Boolean existsByUsername(String username); // 用于注册时检查用户名是否重复
    Boolean existsByEmail(String email); // 用于注册时检查邮箱是否重复
}
```

4.1.2 RoleRepository.java

```
package com.example.secondhandbookbackend.repository;

import com.example.secondhandbookbackend.model.ERole;
import com.example.secondhandbookbackend.model.Role;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;

@Repository
public interface RoleRepository extends JpaRepository<Role, Integer> {
    Optional<Role> findByName(ERole name); // 用于注册时根据枚举查找角色实体
}
```

(顺便把其他需要的 Repository 也建好)

4.1.3 BookRepository.java

```
package com.example.secondhandbookbackend.repository;

import com.example.secondhandbookbackend.model.Book;
import com.example.secondhandbookbackend.model.EBookStatus;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;

@Repository
public interface BookRepository extends JpaRepository<Book, Long> {
    // 查找特定状态的书籍 (例如：只查 FOR_SALE 的)
    List<Book> findByStatus(EBookStatus status);

    // 模糊搜索：同时在标题和作者中查找，且忽略大小写
    List<Book> findByStatusAndTitleContainingIgnoreCaseOrStatusAndAuth
orContainingIgnoreCase(
            EBookStatus status1, String title, EBookStatus status2, String autho
r);
}
```

4.2 实现 UserDetails 服务

Spring Security 不认识我们的 `User` 实体，它只认识自己的 `UserDetails` 接口。我们需要做一个“适配器”。

1. 在 `com.example.secondhandbookbackend` 下新建包 `security`。
2. 创建以下 2 个类。

4.2.1 UserDetailsImpl.java (适配器)

```
package com.example.secondhandbookbackend.security;

import com.example.secondhandbookbackend.model.User;
import com.fasterxml.jackson.annotation.JsonIgnore;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthorit
```

```
y;
import org.springframework.security.core.userdetails.UserDetails;

import java.io.Serial;
import java.util.Collection;
import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;

// 这个类把我们的 User 对象“包装”成 Spring Security 能看懂的 UserDetails 对象
public class UserDetailsImpl implements UserDetails {
    @Serial
    private static final long serialVersionUID = 1L;

    private final Long id;
    private final String username;
    private final String email;
    @JsonIgnore // 确保密码不会被序列化到 JSON 中返回给前端
    private final String password;
    private final Collection<? extends GrantedAuthority> authorities;

    public UserDetailsImpl(Long id, String username, String email, String pas
sword,
                           Collection<? extends GrantedAuthority> authorities) {
        this.id = id;
        this.username = username;
        this.email = email;
        this.password = password;
        this.authorities = authorities;
    }

    // 静态工厂方法：将数据库实体 User 转换为 UserDetailsImpl
    public static UserDetailsImpl build(User user) {
        List<GrantedAuthority> authorities = user.getRoles().stream()
                .map(role -> new SimpleGrantedAuthority(role.getName().name
()))
                .collect(Collectors.toList());
```

```
        return new UserDetailsImpl(
                user.getId(),
                user.getUsername(),
                user.getEmail(),
                user.getPassword(),
                authorities);
    }

    @Override public Collection<? extends GrantedAuthority> getAuthorities
() { return authorities; }
    public Long getId() { return id; }
    public String getEmail() { return email; }
    @Override public String getPassword() { return password; }
    @Override public String getUsername() { return username; }
    @Override public boolean isAccountNonExpired() { return true; }
    @Override public boolean isAccountNonLocked() { return true; }
    @Override public boolean isCredentialsNonExpired() { return true; }
    @Override public boolean isEnabled() { return true; }
    @Override public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        UserDetailsImpl user = (UserDetailsImpl) o;
        return Objects.equals(id, user.id);
    }
}
```

4.2.2 `UserDetailsServiceImpl.java` (加载服务)

```
package com.example.secondhandbookbackend.security;

import com.example.secondhandbookbackend.model.User;
import com.example.secondhandbookbackend.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.UsernameNotFoundE
xception;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

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

    @Override
    @Transactional
    // Spring Security 在登录时会调用这个方法来查找用户
    public UserDetails loadUserByUsername(String username) throws Usern
ameNotFoundException {
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("User Not F
ound with username: " + username));

        return UserDetailsImpl.build(user);
    }
}
```

4.3 JWT 核心工具类 (`JwtUtils.java`)

这是整个安全系统的核心引擎，负责生成和验证加密令牌。我们使用**最终修复版**，确保线程安全和密钥一致性。

4.3.1 `JwtUtils.java` (放在 `security` 包中)

```
package com.example.secondhandbookbackend.security;

import io.jsonwebtoken.*;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
```

```
import org.springframework.stereotype.Component;
import javax.crypto.SecretKey;
import java.util.Date;

@Component
public class JwtUtils {
    private static final Logger logger = LoggerFactory.getLogger(JwtUtils.cla
ss);

    @Value("${secondhandbook.app.jwtSecret}")
    private String jwtSecret; // 从 application.properties 读取密钥

    @Value("${secondhandbook.app.jwtExpirationMs}")
    private int jwtExpirationMs; // 从 application.properties 读取过期时间

    // 帮助方法：将 Base64 字符串转为密钥对象
    private SecretKey key() {
        return Keys.hmacShaKeyFor(Decoders.BASE64.decode(jwtSecret));
    }

    // 1. 生成 Token (用户登录成功后调用)
    public String generateJwtToken(Authentication authentication) {
        UserDetailsImpl userPrincipal = (UserDetailsImpl) authentication.getPri
ncipal();
        return Jwts.builder()
                .subject((userPrincipal.getUsername()))
                .issuedAt(new Date())
                .expiration(new Date((new Date()).getTime() + jwtExpirationMs))
                .signWith(key(), Jwts.SIG.HS256) // 使用 HS256 算法签名
                .compact();
    }

    // 2. 从 Token 中解析用户名 (过滤器验证时调用)
    public String getUserNameFromJwtToken(String token) {
        return Jwts.parser().verifyWith(key()).build()
                .parseSignedClaims(token).getPayload().getSubject();
    }
```

```
    // 3. 验证 Token 是否有效
    public boolean validateJwtToken(String authToken) {
        try {
            Jwts.parser().verifyWith(key()).build().parseSignedClaims(authToke
n);
            return true;
        } catch (MalformedJwtException e) {
            logger.error("Invalid JWT token: {}", e.getMessage());
        } catch (ExpiredJwtException e) {
            logger.error("JWT token is expired: {}", e.getMessage());
        } catch (UnsupportedJwtException e) {
            logger.error("JWT token is unsupported: {}", e.getMessage());
        } catch (IllegalArgumentException e) {
            logger.error("JWT claims string is empty: {}", e.getMessage());
        } catch (io.jsonwebtoken.security.SignatureException e) {
             logger.error("Invalid JWT signature: {}", e.getMessage());
        }
        return false;
    }
}
```

4.4 认证过滤器与入口

4.4.1 AuthTokenFilter.java (安检员)

- **作用**：拦截每一个请求，检查它是否携带了有效的 JWT 令牌。

```
package com.example.secondhandbookbackend.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.authentication.UsernamePasswordAu
thenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
```

```
import org.springframework.security.web.authentication.WebAuthenticatio
nDetailsSource;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;
import java.io.IOException;

public class AuthTokenFilter extends OncePerRequestFilter {
    private final JwtUtils jwtUtils;
    private final UserDetailsServiceImpl userDetailsService;
    private static final Logger logger = LoggerFactory.getLogger(AuthToken
Filter.class);

    public AuthTokenFilter(JwtUtils jwtUtils, UserDetailsServiceImpl userDet
ailsService) {
        this.jwtUtils = jwtUtils;
        this.userDetailsService = userDetailsService;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletR
esponse response, FilterChain filterChain)
            throws ServletException, IOException {
        try {
            String jwt = parseJwt(request);
            if (jwt != null && jwtUtils.validateJwtToken(jwt)) {
                String username = jwtUtils.getUserNameFromJwtToken(jwt);
                UserDetails userDetails = userDetailsService.loadUserByUserna
me(username);

                // 创建认证对象并放入 SecurityContext，表示“已登录”
                UsernamePasswordAuthenticationToken authentication =
                        new UsernamePasswordAuthenticationToken(userDetails, nu
ll, userDetails.getAuthorities());
                authentication.setDetails(new WebAuthenticationDetailsSource().
buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authenticat
ion);
            }
```

```
        } catch (Exception e) {
            logger.error("Cannot set user authentication: {}", e.getMessage());
        }
        filterChain.doFilter(request, response); // 继续处理请求
    }

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

4.4.2 AuthEntryPointJwt.java (门卫)

- **作用**：当未登录用户试图访问受保护接口时，返回 401 错误。

```
package com.example.secondhandbookbackend.security;

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;
import java.io.IOException;

@Component
public class AuthEntryPointJwt implements AuthenticationEntryPoint {
    private static final Logger logger = LoggerFactory.getLogger(AuthEntryP
ointJwt.class);

    @Override
    public void commence(HttpServletRequest request, HttpServletRespons
```

```
e response, AuthenticationException authException)
            throws IOException, ServletException {
        logger.error("Unauthorized error: {}", authException.getMessage());
        // 返回 401 未授权状态码
        response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Error:
Unauthorized");
    }
}
```

4.5 Web 安全总配置 (WebSecurityConfig.java)

这是将以上所有组件组装在一起的地方。我们使用**最终修复版**，包含了关键的 CORS 和 OPTIONS 请求放行配置。

1. 在 `com.example.secondhandbookbackend` 下新建包 `config`。
2. 创建 `WebSecurityConfig.java`。

```
package com.example.secondhandbookbackend.config;

import com.example.secondhandbookbackend.security.AuthEntryPointJwt;
import com.example.secondhandbookbackend.security.AuthTokenFilter;
import com.example.secondhandbookbackend.security.JwtUtils;
import com.example.secondhandbookbackend.security.UserDetailsService
Impl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationManage
r;
import org.springframework.security.authentication.dao.DaoAuthentication
Provider;
import org.springframework.security.config.annotation.authentication.confi
guration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.method.configurati
on.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpS
ecurity;
import org.springframework.security.config.annotation.web.configuration.E
```

```
nableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncod
er;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswo
rdAuthenticationFilter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import java.util.Arrays;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity // 启用 @PreAuthorize 注解
public class WebSecurityConfig {

    @Autowired UserDetailsServiceImpl userDetailsService;
    @Autowired private AuthEntryPointJwt unauthorizedHandler;
    @Autowired private JwtUtils jwtUtils;

    @Bean
    public AuthTokenFilter authenticationJwtTokenFilter() {
        return new AuthTokenFilter(jwtUtils, userDetailsService);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(); // 使用 BCrypt 强哈希加密密码
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProv
ider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());
```

```
        return authProvider;
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationCon
figuration authConfig) throws Exception {
        return authConfig.getAuthenticationManager();
    }

    // --- 全局 CORS 配置 (允许前端跨域访问) ---
    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(Arrays.asList("http://localhost:808
1")); // 前端地址
        configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PU
T", "DELETE", "OPTIONS"));
        configuration.setAllowedHeaders(Arrays.asList("Authorization", "Cach
e-Control", "Content-Type"));
        configuration.setAllowCredentials(true);
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfig
urationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }

    // --- 安全过滤器链配置 ---
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
{
        http.cors(cors -> cors.configurationSource(corsConfigurationSource
())) // 应用 CORS
            .csrf(csrf -> csrf.disable()) // 禁用 CSRF (JWT 不需要)
            .exceptionHandling(exception -> exception.authenticationEntryPoint
(unauthorizedHandler))
            .sessionManagement(session -> session.sessionCreationPolicy(Ses
sionCreationPolicy.STATELESS)) // 无状态
            .authorizeHttpRequests(authz -> authz
```

```
                .requestMatchers(HttpMethod.OPTIONS, "/**").permitAll() // 【关
键】放行所有预检请求
                .requestMatchers("/api/auth/**").permitAll() // 放行登录注册
                .requestMatchers(HttpMethod.GET, "/api/books", "/api/books/**",
"/api/books/search").permitAll() // 放行书籍浏览和搜索
                .requestMatchers("/images/**").permitAll() // 放行图片访问
                .anyRequest().authenticated() // 其他所有请求都需要认证
            );

        http.authenticationProvider(authenticationProvider());
        http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswo
rdAuthenticationFilter.class);

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

阶段性验证：

1. 确保以上所有文件都已正确创建。
2. **【关键】初始化数据库数据**：打开 Navicat，在 `roles` 表中**手动添加**两行数据：
   - id: `1`, name: `ROLE_USER`
   - id: `2`, name: `ROLE_ADMIN`
   - *(如果缺少这一步，后续注册会失败)*
3. **尝试运行后端**：它应该能成功启动，没有红色错误。

请确认这一部分（核心安全）已完成并理解，然后我们进入【第 5 部分：后端 - API 接口】。

第 5 部分：后端 - API 接口 (Controllers & DTOs)

在这一部分，我们将实现业务逻辑的核心。Controller 是前端与后端交互的桥梁。为了让数据传输更规范，我们还需要创建一系列 **DTO (Data Transfer Object)** 对象。

所有 DTO 文件放在 `com.example.secondhandbookbackend.dto` 包下。所有 Controller 文件放在 `com.example.secondhandbookbackend.controller` 包下。

5.1 数据传输对象 (DTOs)

DTO 就像是“信封”，用来在前后端之间传递特定的数据。

5.1.1 认证相关 DTOs

- `SignupRequest.java` (注册请求)

```
package com.example.secondhandbookbackend.dto;
import jakarta.validation.constraints.*;
import lombok.Getter;
import lombok.Setter;
import java.util.Set;

@Getter @Setter
public class SignupRequest {
    @NotBlank @Size(min = 3, max = 20)
    private String username;
    @NotBlank @Email
    private String email;
    @NotBlank @Size(min = 6, max = 40)
    private String password;
    private Set<String> role; // 允许前端请求特定角色（通常只用 "user"）
}
```

`LoginRequest.java` (登录请求)

```
package com.example.secondhandbookbackend.dto;
import jakarta.validation.constraints.NotBlank;
import lombok.Getter;
import lombok.Setter;

@Getter @Setter
public class LoginRequest {
    @NotBlank private String username;
    @NotBlank private String password;
}
```

`JwtResponse.java` (登录成功响应)

```
package com.example.secondhandbookbackend.dto;
import lombok.Getter;
import lombok.Setter;
import java.util.List;

@Getter @Setter
public class JwtResponse {
    private String token;
    private String type = "Bearer";
    private Long id;
    private String username;
    private String email;
    private List<String> roles;

    public JwtResponse(String token, Long id, String username, String email,
List<String> roles) {
        this.token = token;
        this.id = id;
        this.username = username;
        this.email = email;
        this.roles = roles;
    }
}
```

5.1.2 业务相关 DTOs

- `UserDTO.java` (用于管理后台显示用户信息，不包含密码)

```
package com.example.secondhandbookbackend.dto;
import lombok.Getter;
import lombok.Setter;
import java.util.Set;

@Getter @Setter
public class UserDTO {
    private Long id;
```

```
    private String username;
    private String email;
    private Set<String> roles;
}
```

BookDTO.java (用于书籍信息的接收和发送)

```
package com.example.secondhandbookbackend.dto;
import com.example.secondhandbookbackend.model.EBookStatus;
import lombok.Getter;
import lombok.Setter;

@Getter @Setter
public class BookDTO {
    private Long id;
    private String title;
    private String author;
    private Double price;
    private String description;
    private EBookStatus status;
    private String imageUrl; // 封面图片 URL
    private String sellerUsername; // 只包含卖家用户名，而不是整个用户对象
}
```

DashboardStatsDTO.java (数据看板统计)

```
package com.example.secondhandbookbackend.dto;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.Setter;

@Getter @Setter @AllArgsConstructor
public class DashboardStatsDTO {
    private long totalUsers;
    private long totalBooks;
```

```
    private long totalOrders;
}
```

5.1.3 订单相关 DTOs (解决 JSON 循环引用问题的关键)

- OrderItemDTO.java

```
package com.example.secondhandbookbackend.dto;
import com.example.secondhandbookbackend.model.Book;
import com.example.secondhandbookbackend.model.OrderItem;
import lombok.Getter;
import lombok.Setter;

@Getter @Setter
public class OrderItemDTO {
    private Long id;
    private Double priceAtPurchase;
    private Long bookId;
    private String bookTitle;
    private String bookImageUrl;

    // 静态工厂方法：从实体转换为 DTO
    public static OrderItemDTO fromEntity(OrderItem item) {
        OrderItemDTO dto = new OrderItemDTO();
        dto.setId(item.getId());
        dto.setPriceAtPurchase(item.getPriceAtPurchase());
        Book book = item.getBook();
        if (book != null) {
            dto.setBookId(book.getId());
            dto.setBookTitle(book.getTitle());
            dto.setBookImageUrl(book.getImageUrl());
        }
        return dto;
    }
}
```

OrderDTO.java

```
package com.example.secondhandbookbackend.dto;
import com.example.secondhandbookbackend.model.Order;
import lombok.Getter;
import lombok.Setter;
import java.time.LocalDateTime;
import java.util.Set;
import java.util.stream.Collectors;

@Getter @Setter
public class OrderDTO {
    private Long id;
    private LocalDateTime orderDate;
    private Double totalPrice;
    private Set<OrderItemDTO> items;

    public static OrderDTO fromEntity(Order order) {
        OrderDTO dto = new OrderDTO();
        dto.setId(order.getId());
        dto.setOrderDate(order.getOrderDate());
        dto.setTotalPrice(order.getTotalPrice());
        dto.setItems(order.getItems().stream().map(OrderItemDTO::fromEntit
y).collect(Collectors.toSet()));
        return dto;
    }
}
```

5.2 API 接口 (Controllers)

请确保这些文件都位于 `src/main/java/com/example/secondhandbookbackend/controller` 目录下。

1. AuthController.java (认证)

```
package com.example.secondhandbookbackend.controller;

import com.example.secondhandbookbackend.dto.JwtResponse;
import com.example.secondhandbookbackend.dto.LoginRequest;
import com.example.secondhandbookbackend.dto.SignupRequest;
import com.example.secondhandbookbackend.model.ERole;
```

```
import com.example.secondhandbookbackend.model.Role;
import com.example.secondhandbookbackend.model.User;
import com.example.secondhandbookbackend.repository.RoleRepository;
import com.example.secondhandbookbackend.repository.UserRepository;
import com.example.secondhandbookbackend.security.JwtUtils;
import com.example.secondhandbookbackend.security.UserDetailsImpl;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManage
r;
import org.springframework.security.authentication.UsernamePasswordAu
thenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.*;

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

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/auth")
public class AuthController {
    @Autowired
    AuthenticationManager authenticationManager;

    @Autowired
    UserRepository userRepository;

    @Autowired
    RoleRepository roleRepository;

    @Autowired
    PasswordEncoder encoder;
```

```
    @Autowired
    JwtUtils jwtUtils;

    @PostMapping("/signin")
    public ResponseEntity<?> authenticateUser(@Valid @RequestBody Logi
nRequest loginRequest) {
        Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(loginRequest.getUs
ername(), loginRequest.getPassword()));

        SecurityContextHolder.getContext().setAuthentication(authentication);
        String jwt = jwtUtils.generateJwtToken(authentication);

        UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrin
cipal();
        List<String> roles = userDetails.getAuthorities().stream()
                .map(item -> item.getAuthority())
                .collect(Collectors.toList());

        return ResponseEntity.ok(new JwtResponse(jwt,
                userDetails.getId(),
                userDetails.getUsername(),
                userDetails.getEmail(),
                roles));
    }

    @PostMapping("/signup")
    public ResponseEntity<?> registerUser(@Valid @RequestBody SignupRe
quest signUpRequest) {
        if (userRepository.existsByUsername(signUpRequest.getUsername()))
{
            return ResponseEntity
                    .badRequest()
                    .body("Error: Username is already taken!");
        }

        if (userRepository.existsByEmail(signUpRequest.getEmail())) {
```

```
            return ResponseEntity
                    .badRequest()
                    .body("Error: Email is already in use!");
        }

        // Create new user's account
        User user = new User(signUpRequest.getUsername(),
                signUpRequest.getEmail(),
                encoder.encode(signUpRequest.getPassword()));

        Set<String> strRoles = signUpRequest.getRole();
        Set<Role> roles = new HashSet<>();

        if (strRoles == null) {
            Role userRole = roleRepository.findByName(ERole.ROLE_USER)
                    .orElseThrow(() -> new RuntimeException("Error: Role is not fou
nd."));
            roles.add(userRole);
        } else {
            strRoles.forEach(role -> {
                switch (role) {
                    case "admin":
                        Role adminRole = roleRepository.findByName(ERole.ROLE_A
DMIN)
                                .orElseThrow(() -> new RuntimeException("Error: Role is
not found."));
                        roles.add(adminRole);
                        break;
                    default:
                        Role userRole = roleRepository.findByName(ERole.ROLE_USE
R)
                                .orElseThrow(() -> new RuntimeException("Error: Role is
not found."));
                        roles.add(userRole);
                }
            });
        }
```

```
        user.setRoles(roles);
        userRepository.save(user);

        return ResponseEntity.ok("User registered successfully!");
    }
}
```

2. `BookController.java` (书籍)

```
package com.example.secondhandbookbackend.controller;

import com.example.secondhandbookbackend.dto.BookDTO;
import com.example.secondhandbookbackend.model.Book;
import com.example.secondhandbookbackend.model.EBookStatus;
import com.example.secondhandbookbackend.model.User;
import com.example.secondhandbookbackend.repository.BookRepository;
import com.example.secondhandbookbackend.repository.UserRepository;
import com.example.secondhandbookbackend.security.UserDetailsImpl;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.*;

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

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/books")
public class BookController {

    @Autowired
```

```
    BookRepository bookRepository;

    @Autowired
    UserRepository userRepository;

    @GetMapping
    public ResponseEntity<List<BookDTO>> getAllBooks() {
        List<Book> books = bookRepository.findByStatus(EBookStatus.FOR_S
ALE);
        List<BookDTO> bookDTOs = books.stream()
                .map(this::convertToDto)
                .collect(Collectors.toList());

        return ResponseEntity.ok(bookDTOs);
    }

    @GetMapping("/search")
    public ResponseEntity<List<BookDTO>> searchBooks(@RequestParam
(name = "q") String query) {
        List<Book> books = bookRepository.findByStatusAndTitleContainingIg
noreCaseOrStatusAndAuthorContainingIgnoreCase(
                EBookStatus.FOR_SALE, query, EBookStatus.FOR_SALE, query
        );

        List<BookDTO> bookDTOs = books.stream()
                .map(this::convertToDto)
                .collect(Collectors.toList());

        return ResponseEntity.ok(bookDTOs);
    }

    @GetMapping("/{bookId}")
    public ResponseEntity<?> getBookById(@PathVariable Long bookId) {
        Optional<Book> bookOpt = bookRepository.findById(bookId);
        if (bookOpt.isPresent()) {
            return ResponseEntity.ok(convertToDto(bookOpt.get()));
        } else {
            return ResponseEntity.notFound().build();
```

```
        }
    }

    @PostMapping
    @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
    public ResponseEntity<?> publishBook(@Valid @RequestBody BookDTO
bookDTO) {
        User currentUser = getCurrentUser();

        Book book = new Book();
        book.setTitle(bookDTO.getTitle());
        book.setAuthor(bookDTO.getAuthor());
        book.setDescription(bookDTO.getDescription());
        book.setPrice(bookDTO.getPrice());
        book.setStatus(EBookStatus.FOR_SALE);
        book.setSeller(currentUser);
        book.setImageUrl(bookDTO.getImageUrl());

        bookRepository.save(book);

        return ResponseEntity.ok("Book published successfully!");
    }

    private BookDTO convertToDto(Book book) {
        BookDTO bookDTO = new BookDTO();
        bookDTO.setId(book.getId());
        bookDTO.setTitle(book.getTitle());
        bookDTO.setAuthor(book.getAuthor());
        bookDTO.setDescription(book.getDescription());
        bookDTO.setPrice(book.getPrice());
        bookDTO.setStatus(book.getStatus());
        bookDTO.setImageUrl(book.getImageUrl());
        if (book.getSeller() != null) {
            bookDTO.setSellerUsername(book.getSeller().getUsername());
        }
        return bookDTO;
    }
```

```
    private User getCurrentUser() {
        Authentication authentication = SecurityContextHolder.getContext().g
etAuthentication();
        UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrin
cipal();
        return userRepository.findById(userDetails.getId())
                .orElseThrow(() -> new RuntimeException("Error: User not foun
d."));
    }
}
```

3. CartController.java (购物车)

```
package com.example.secondhandbookbackend.controller;

import com.example.secondhandbookbackend.dto.BookDTO;
import com.example.secondhandbookbackend.model.Book;
import com.example.secondhandbookbackend.model.CartItem;
import com.example.secondhandbookbackend.model.EBookStatus;
import com.example.secondhandbookbackend.model.User;
import com.example.secondhandbookbackend.repository.BookRepository;
import com.example.secondhandbookbackend.repository.CartItemReposito
ry;
import com.example.secondhandbookbackend.repository.UserRepository;
import com.example.secondhandbookbackend.security.UserDetailsImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.*;

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

```
@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/cart")
@PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
public class CartController {

    @Autowired
    CartItemRepository cartItemRepository;

    @Autowired
    BookRepository bookRepository;

    @Autowired
    UserRepository userRepository;

    @GetMapping
    public ResponseEntity<List<BookDTO>> getMyCart() {
        User currentUser = getCurrentUser();
        List<CartItem> cartItems = cartItemRepository.findByUser(currentUse
r);

        List<BookDTO> booksInCart = cartItems.stream()
                .map(cartItem -> convertToDto(cartItem.getBook()))
                .collect(Collectors.toList());

        return ResponseEntity.ok(booksInCart);
    }

    @PostMapping("/add/{bookId}")
    public ResponseEntity<?> addBookToCart(@PathVariable Long bookId) {
        User currentUser = getCurrentUser();

        Optional<Book> bookOpt = bookRepository.findById(bookId);
        if (!bookOpt.isPresent()) {
            return ResponseEntity.badRequest().body("Error: Book not found!");
        }
        Book book = bookOpt.get();
```

```
        if (book.getStatus() != EBookStatus.FOR_SALE) {
            return ResponseEntity.badRequest().body("Error: Book is not availa
ble for sale!");
        }

        if (cartItemRepository.findByBookId(bookId).isPresent()) {
            return ResponseEntity.badRequest().body("Error: Book is already in
someone else's cart!");
        }

        book.setStatus(EBookStatus.LOCKED);
        bookRepository.save(book);

        CartItem cartItem = new CartItem(book, currentUser);
        cartItemRepository.save(cartItem);

        return ResponseEntity.ok("Book added to cart successfully!");
    }

    @DeleteMapping("/remove/{bookId}")
    public ResponseEntity<?> removeBookFromCart(@PathVariable Long bo
okId) {
        User currentUser = getCurrentUser();

        Optional<CartItem> cartItemOpt = cartItemRepository.findByBookIdAn
dUser(bookId, currentUser);
        if (!cartItemOpt.isPresent()) {
            return ResponseEntity.badRequest().body("Error: Book not found in
your cart!");
        }
        CartItem cartItem = cartItemOpt.get();

        Book book = cartItem.getBook();
        book.setStatus(EBookStatus.FOR_SALE);
        bookRepository.save(book);

        cartItemRepository.delete(cartItem);
```

```
        return ResponseEntity.ok("Book removed from cart successfully!");
    }

    private User getCurrentUser() {
        Authentication authentication = SecurityContextHolder.getContext().g
etAuthentication();
        UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrin
cipal();
        return userRepository.findById(userDetails.getId())
                .orElseThrow(() -> new RuntimeException("Error: User not foun
d."));
    }

    private BookDTO convertToDto(Book book) {
        BookDTO bookDTO = new BookDTO();
        bookDTO.setId(book.getId());
        bookDTO.setTitle(book.getTitle());
        bookDTO.setAuthor(book.getAuthor());
        bookDTO.setDescription(book.getDescription());
        bookDTO.setPrice(book.getPrice());
        bookDTO.setStatus(book.getStatus());
        bookDTO.setImageUrl(book.getImageUrl());
        if (book.getSeller() != null) {
            bookDTO.setSellerUsername(book.getSeller().getUsername());
        }
        return bookDTO;
    }
}
```

4. `OrderController.java` (订单)

```
package com.example.secondhandbookbackend.controller;

import com.example.secondhandbookbackend.dto.OrderDTO;
import com.example.secondhandbookbackend.model.*;
import com.example.secondhandbookbackend.repository.CartItemReposito
```

```
ry;
import com.example.secondhandbookbackend.repository.OrderRepository;
import com.example.secondhandbookbackend.repository.UserRepository;
import com.example.secondhandbookbackend.security.UserDetailsImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.*;

import java.time.LocalDateTime;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/orders")
@PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
public class OrderController {

    @Autowired
    private OrderRepository orderRepository;

    @Autowired
    private CartItemRepository cartItemRepository;

    @Autowired
    private UserRepository userRepository;

    private User getCurrentUser() {
        Authentication authentication = SecurityContextHolder.getContext().g
etAuthentication();
        UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrin
cipal();
```

```
        return userRepository.findById(userDetails.getId())
                .orElseThrow(() -> new RuntimeException("Error: User not foun
d."));
    }

    @PostMapping("/checkout")
    @Transactional
    public ResponseEntity<?> checkout() {
        User currentUser = getCurrentUser();

        List<CartItem> cartItems = cartItemRepository.findByUser(currentUse
r);

        if (cartItems.isEmpty()) {
            return ResponseEntity.badRequest().body("Error: Your cart is empt
y.");
        }

        Order order = new Order();
        order.setBuyer(currentUser);
        order.setOrderDate(LocalDateTime.now());

        Set<OrderItem> orderItems = new HashSet<>();
        double totalPrice = 0.0;

        for (CartItem cartItem : cartItems) {
            Book book = cartItem.getBook();

            if (book.getStatus() != EBookStatus.LOCKED) {
                throw new RuntimeException("Error: Book " + book.getTitle() + " i
s no longer available.");
            }

            OrderItem orderItem = new OrderItem();
            orderItem.setOrder(order);
            orderItem.setBook(book);
            orderItem.setPriceAtPurchase(book.getPrice());
```

```
            orderItems.add(orderItem);
            totalPrice += book.getPrice();

            book.setStatus(EBookStatus.SOLD);
            cartItemRepository.delete(cartItem);
        }

        order.setTotalPrice(totalPrice);
        order.setItems(orderItems);

        orderRepository.save(order);

        return ResponseEntity.ok("Checkout successful! Order created.");
    }

    @GetMapping
    public ResponseEntity<List<OrderDTO>> getMyOrders() {
        User currentUser = getCurrentUser();
        List<Order> orders = orderRepository.findByBuyerOrderByOrderDateD
esc(currentUser);

        List<OrderDTO> orderDTOs = orders.stream()
                .map(OrderDTO::fromEntity)
                .collect(Collectors.toList());

        return ResponseEntity.ok(orderDTOs);
    }
}
```

5. `AdminController.java` (管理员)

```
package com.example.secondhandbookbackend.controller;

import com.example.secondhandbookbackend.dto.BookDTO;
import com.example.secondhandbookbackend.dto.DashboardStatsDTO;
import com.example.secondhandbookbackend.dto.UserDTO;
```

```
import com.example.secondhandbookbackend.model.Book;
import com.example.secondhandbookbackend.model.User;
import com.example.secondhandbookbackend.repository.BookRepository;
import com.example.secondhandbookbackend.repository.OrderRepository;
import com.example.secondhandbookbackend.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

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

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/admin")
@PreAuthorize("hasRole('ADMIN')")
public class AdminController {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private BookRepository bookRepository;

    @Autowired
    private OrderRepository orderRepository;

    @GetMapping("/users")
    public ResponseEntity<List<UserDTO>> getAllUsers() {
        List<User> users = userRepository.findAll();
        List<UserDTO> userDTOs = users.stream()
                .map(this::convertToUserDto)
                .collect(Collectors.toList());
        return ResponseEntity.ok(userDTOs);
    }
```

```
    @DeleteMapping("/users/{id}")
    public ResponseEntity<?> deleteUser(@PathVariable Long id) {
        if (!userRepository.existsById(id)) {
            return ResponseEntity.notFound().build();
        }
        userRepository.deleteById(id);
        return ResponseEntity.ok("User deleted successfully!");
    }

    @GetMapping("/books")
    public ResponseEntity<List<BookDTO>> getAllBooksAdmin() {
        List<Book> books = bookRepository.findAll();
        List<BookDTO> bookDTOs = books.stream()
                .map(this::convertToBookDto)
                .collect(Collectors.toList());
        return ResponseEntity.ok(bookDTOs);
    }

    @DeleteMapping("/books/{id}")
    public ResponseEntity<?> deleteBook(@PathVariable Long id) {
        if (!bookRepository.existsById(id)) {
            return ResponseEntity.notFound().build();
        }
        // (Simple deletion, will fail if book is in an order due to FK constraints)
        bookRepository.deleteById(id);
        return ResponseEntity.ok("Book deleted successfully!");
    }

    @GetMapping("/stats/dashboard")
    public ResponseEntity<DashboardStatsDTO> getDashboardStats() {
        long totalUsers = userRepository.count();
        long totalBooks = bookRepository.count();
        long totalOrders = orderRepository.count();

        DashboardStatsDTO stats = new DashboardStatsDTO(totalUsers, total
Books, totalOrders);
        return ResponseEntity.ok(stats);
    }
```

```
    @GetMapping("/test")
    public ResponseEntity<String> testAdminAccess() {
        return ResponseEntity.ok("Admin content successfully accessed!");
    }

    private UserDTO convertToUserDto(User user) {
        UserDTO userDTO = new UserDTO();
        userDTO.setId(user.getId());
        userDTO.setUsername(user.getUsername());
        userDTO.setEmail(user.getEmail());
        Set<String> roles = user.getRoles().stream()
                .map(role -> role.getName().name())
                .collect(Collectors.toSet());
        userDTO.setRoles(roles);
        return userDTO;
    }

    private BookDTO convertToBookDto(Book book) {
        BookDTO bookDTO = new BookDTO();
        bookDTO.setId(book.getId());
        bookDTO.setTitle(book.getTitle());
        bookDTO.setAuthor(book.getAuthor());
        bookDTO.setDescription(book.getDescription());
        bookDTO.setPrice(book.getPrice());
        bookDTO.setStatus(book.getStatus());
        bookDTO.setImageUrl(book.getImageUrl());
        if (book.getSeller() != null) {
            bookDTO.setSellerUsername(book.getSeller().getUsername());
        }
        return bookDTO;
    }
}
```

6. `FileController.java` (文件上传)

```
package com.example.secondhandbookbackend.controller;

import com.example.secondhandbookbackend.service.FileStorageService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.support.ServletUriComponentsBui
lder;

import java.util.Map;

@RestController
@RequestMapping("/api")
@CrossOrigin(origins = "*", maxAge = 3600)
public class FileController {

    @Autowired
    private FileStorageService fileStorageService;

    @PostMapping("/upload")
    @PreAuthorize("hasRole('USER') or hasRole('ADMIN')")
    public ResponseEntity<?> uploadFile(@RequestParam("file") MultipartFil
e file) {
        String fileName = fileStorageService.storeFile(file);

        String fileDownloadUri = ServletUriComponentsBuilder.fromCurrentCo
ntextPath()
                .path("/images/")
                .path(fileName)
                .toUriString();

        return ResponseEntity.ok(Map.of("url", fileDownloadUri));
    }
}
```

章节总结：

这一部分是后端工作量最大的一块。完成后，你的后端就不仅仅是一个空壳，而是一个拥有完整业务逻辑的智能系统了。

请确认你对这些 API 的设计和实现逻辑（特别是 DTO 的作用）有了清晰的理解。准备好后，我们将进入激动人心的【第 6 部分：前端项目搭建】！

第 6 部分：前端项目搭建 (Frontend Setup)

这一部分的目标是创建一个干净的 Vue 3 项目，并安装所有必要的第三方库。

6.1 创建 Vue 3 项目

(你已经完成过这一步，这里是教程回顾)

我们使用 Vue CLI (或 `create-vue`) 来初始化项目。

```
# 在终端运行 (确保在你想存放项目的目录下)
vue create secondhand-book-frontend
```

- **选择配置**：Manually select features (手动选择)
- **勾选**：`Babel`, `Router`, `Linter / Formatter`
- **Vue 版本**：`3.x`
- **路由模式**：Use history mode? → `Y` (是)

6.2 安装核心依赖

我们需要安装用于 HTTP 请求、状态管理和图表的库。

```
cd secondhand-book-frontend
npm install axios pinia chart.js vue-chartjs
```

- `axios`: 前端用来向后端发请求的“邮递员”。
- `pinia`: Vue 3 官方推荐的状态管理库，用来存储全局数据（如用户信息、是否登录）。
- `chart.js` & `vue-chartjs`: 用于在管理后台绘制统计图表。

6.3 项目清理

为了有一个干净的开始，我们需要删除 Vue CLI 默认生成的一些示例文件：

- 删除 `src/components/HelloWorld.vue`
- 清空 `src/views/HomeView.vue` 和 `src/App.vue` 的默认内容（我们稍后会重写）。

第 7 部分：前端 - 状态管理与路由 (Core Infrastructure)

7.1 初始化插件 (`src/main.js`)

这是 Vue 应用的入口文件。我们需要在这里安装并启用 Pinia（状态管理）和 Vue Router（路由）。

完整代码 (`src/main.js`)：

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'
import { createPinia } from 'pinia' // 1. 导入 Pinia

const app = createApp(App)

app.use(createPinia()) // 2. 安装 Pinia 实例
app.use(router)        // 3. 安装 Router 实例
app.mount('#app')      // 4. 挂载应用
```

7.2 全局身份认证仓库 (`src/store/auth.js`)

这是我们用来管理用户登录状态的核心。它使用 Pinia 来创建一个响应式的“仓库”，并自动将状态同步到浏览器的 `localStorage` 中，以防止刷新页面后登录状态丢失。

你需要先创建 `src/store` 文件夹。

完整代码 (`src/store/auth.js`)：

```
import { defineStore } from 'pinia';

export const useAuthStore = defineStore('auth', {

  // 1. 定义状态 (State)
  //    初始化时尝试从 localStorage 读取，以便在页面刷新后恢复登录状态
  state: () => ({
    token: localStorage.getItem('token') || null,
    user: JSON.parse(localStorage.getItem('user')) || null,
```

```
  }),

  // 2. 定义 Getters (类似计算属性)
  getters: {
    // 判断是否已登录
    isLoggedIn: (state) => !!state.token,
    // 获取当前用户名
    currentUserUsername: (state) => (state.user ? state.user.username : null),
    // 判断是否是管理员
    isAdmin: (state) => (state.user && state.user.roles ? state.user.roles.includ
es('ROLE_ADMIN') : false),
    // 获取 Token 字符串
    authToken: (state) => state.token,
  },

  // 3. 定义 Actions (方法)
  actions: {
    // 登录成功时调用，保存状态到 Pinia 和 localStorage
    loginSuccess(loginData) {
      this.token = loginData.token;

      const user = {
        id: loginData.id,
        username: loginData.username,
        email: loginData.email,
        roles: loginData.roles
      };
      this.user = user;

      localStorage.setItem('token', loginData.token);
      localStorage.setItem('user', JSON.stringify(user));
    },

    // 退出登录时调用，清除所有状态
    logout() {
      this.token = null;
      this.user = null;
      localStorage.removeItem('token');
```

```
      localStorage.removeItem('user');
    }
  },
});
```

7.3 路由配置与权限守卫 (src/router/index.js)

这个文件定义了网站的所有页面地址，并配置了“路由守卫”来保护需要登录或管理员权限才能访问的页面。

重要提示：这份代码引用了所有我们将要创建的页面组件。如果你还没有创建这些文件，Vue 可能会报错。你可以先创建这些空文件，或者在下一章创建完所有页面后再回头确认这个文件。

完整代码 (src/router/index.js)：

```
import { createRouter, createWebHistory } from 'vue-router'
import { useAuthStore } from '@/store/auth'; // 导入我们刚创建的 store

// --- 导入所有页面组件 ---
import HomeView from '../views/HomeView.vue'
import LoginView from '../views/LoginView.vue';
import RegisterView from '../views/RegisterView.vue';
import PublishView from '../views/PublishView.vue';
import BookDetailView from '../views/BookDetailView.vue';
import CartView from '../views/CartView.vue';
import OrderHistoryView from '../views/OrderHistoryView.vue';
// 导入管理员相关组件
import AdminLayout from '../layouts/AdminLayout.vue';
import AdminDashboard from '../views/admin/AdminDashboard.vue';
import AdminUserManagement from '../views/admin/AdminUserManagement.vue';
import AdminBookManagement from '../views/admin/AdminBookManagement.vue';

const routes = [
  // --- 公开页面 ---
  {
    path: '/',
    name: 'home',
```

```
    component: HomeView
  },
  {
    path: '/login',
    name: 'login',
    component: LoginView
  },
  {
    path: '/register',
    name: 'register',
    component: RegisterView
  },
  {
    path: '/book/:id', // 书籍详情页 (动态路由)
    name: 'book-detail',
    component: BookDetailView
  },

  // --- 需要登录的页面 (requiresAuth: true) ---
  {
    path: '/publish',
    name: 'publish',
    component: PublishView,
    meta: { requiresAuth: true }
  },
  {
    path: '/cart',
    name: 'cart',
    component: CartView,
    meta: { requiresAuth: true }
  },
  {
    path: '/orders',
    name: 'orders',
    component: OrderHistoryView,
    meta: { requiresAuth: true }
  },
```

```
  // --- 管理员后台 (requiresAuth: true, requiresAdmin: true) ---
  {
    path: '/admin',
    component: AdminLayout, // 使用专门的布局组件
    meta: { requiresAuth: true, requiresAdmin: true },
    children: [
      {
        path: '', // 默认重定向到仪表盘
        redirect: '/admin/dashboard'
      },
      {
        path: 'dashboard',
        name: 'admin-dashboard',
        component: AdminDashboard
      },
      {
        path: 'users',
        name: 'admin-users',
        component: AdminUserManagement
      },
      {
        path: 'books',
        name: 'admin-books',
        component: AdminBookManagement
      }
    ]
  },

  // (可选) 关于页面
  {
    path: '/about',
    name: 'about',
    // 路由懒加载示例
    component: () => import(/* webpackChunkName: "about" */ '../views/Ab
outView.vue')
  }
]
```

```
const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes
})

// --- 全局前置路由守卫 ---
router.beforeEach((to, from, next) => {
  const authStore = useAuthStore();
  const loggedIn = authStore.isLoggedIn;
  const isAdmin = authStore.isAdmin;

  // 1. 检查目标路由是否需要登录
  if (to.matched.some(record => record.meta.requiresAuth)) {
    if (!loggedIn) {
      // 未登录，重定向到登录页
      next({ name: 'login' });
    } else {
      // 已登录，检查是否需要管理员权限
      if (to.matched.some(record => record.meta.requiresAdmin) && !isAdmin)
{
        // 权限不足，提示并重定向回首页
        alert('您没有权限访问此页面！');
        next({ name: 'home' });
      } else {
        // 权限满足，放行
        next();
      }
    }
  } else {
    // 不需要登录的页面，直接放行
    next();
  }
});

export default router
```

好的，第 7 部分的基础设施已经就位。现在我们来构建用户直接交互的核心页面。

这是**【第 8 部分：前端 - 核心布局与页面】的完整代码**。

请**逐个**打开这些文件，用下面的代码**完全替换**掉它们原有的内容。这些代码已经包含了我们之前所有的修复（Pinia 集成、401 强制刷新、ESLint 修复等）。

第 8 部分：前端 - 核心布局与页面 (Core UI & Pages)

1. 主布局 `src/App.vue`

(包含吸顶导航栏、全局搜索框、根据登录状态变化的链接)

```
<template>
  <header class="main-header">
    <div class="header-left">
      <router-link to="/" class="logo">二手书店</router-link>
      <router-link to="/publish" class="nav-link">发布书籍</router-link>
    </div>

    <div class="header-middle">
      <form @submit.prevent="handleSearch" class="search-form">
        <input type="text" v-model="searchQuery" placeholder="搜索书名或
作者...">
        <button type="submit">搜索</button>
      </form>
    </div>

    <div class="header-right">
      <span v-if="!isLoggedIn" class="user-links">
        <router-link to="/login">登录</router-link> |
        <router-link to="/register">注册</router-link>
      </span>
      <span v-else class="user-links">
        <span>Hi, {{ currentUserUsername }}</span> |
        <router-link to="/orders">我的订单</router-link> |
        <router-link to="/cart">购物车</router-link> |
        <router-link v-if="isAdmin" to="/admin">管理后台</router-link> |
        <a href="#" @click.prevent="logout">退出</a>
      </span>
    </div>
  </header>
```

```
  <div class="main-content">
    <router-view/>
  </div>
</template>

<script>
import { useAuthStore } from '@/store/auth';
import { mapState } from 'pinia';

export default {
  data() {
    return {
      searchQuery: '',
    };
  },
  computed: {
    ...mapState(useAuthStore, ['isLoggedIn', 'currentUserUsername', 'isAdmi
n'])
  },
  methods: {
    logout() {
      const authStore = useAuthStore();
      authStore.logout();
      this.$router.push('/').then(() => {
        window.location.reload(); // 强制刷新以重置状态
      });
    },
    handleSearch() {
      if (!this.searchQuery.trim()) {
        this.$router.push({ path: '/' });
      } else {
        this.$router.push({ path: '/', query: { q: this.searchQuery } });
      }
    }
  }
}
</script>
```

```
<style>
/* 全局样式 */
body {
  margin: 0;
  font-family: Avenir, Helvetica, Arial, sans-serif;
  color: #2c3e50;
  background-color: #f5f7fa;
}
a {
  text-decoration: none;
  color: inherit;
}

/* 导航栏样式 */
.main-header {
  background-color: #232f3e;
  color: white;
  padding: 10px 20px;
  display: flex;
  align-items: center;
  justify-content: space-between;
  gap: 20px;
  position: sticky;
  top: 0;
  z-index: 1000;
  height: 60px;
  box-shadow: 0 2px 5px rgba(0,0,0,0.2);
  box-sizing: border-box;
}
.header-left {
  display: flex;
  align-items: center;
  gap: 20px;
}
.logo {
  font-size: 1.5em;
  font-weight: bold;
  color: #febd69;
```

```
}
.nav-link {
  color: white;
  font-weight: bold;
}
.nav-link:hover {
  text-decoration: underline;
}
.header-middle {
  flex-grow: 1;
  max-width: 600px;
}
.search-form {
  display: flex;
  width: 100%;
}
.search-form input {
  flex-grow: 1;
  padding: 8px 10px;
  border: none;
  border-radius: 4px 0 0 4px;
  outline: none;
}
.search-form button {
  padding: 8px 15px;
  border: none;
  background-color: #febd69;
  color: #111;
  cursor: pointer;
  border-radius: 0 4px 4px 0;
}
.search-form button:hover {
  background-color: #f3a847;
}
.header-right {
  display: flex;
  align-items: center;
}
```

```
.user-links {
  display: flex;
  align-items: center;
  gap: 15px;
  font-size: 0.9em;
  color: #ccc;
}
.user-links a {
  color: white;
}
.user-links a:hover {
  text-decoration: underline;
}
.user-links span {
  color: #febd69;
  font-weight: bold;
}
</style>
```

2. 首页 src/views/HomeView.vue

(响应全局搜索，展示书籍网格)

```
<template>
  <div class="home">
    <h2 v-if="currentQuery">"{{ currentQuery }}" 的搜索结果</h2>
    <h2 v-else>热门二手书</h2>

    <div v-if="loading" class="loading">正在加载书籍...</div>
    <div v-else-if="error" class="error-message">{{ error }}</div>

    <div v-else-if="books.length === 0" class="empty-state">
      <p>
        <span v-if="currentQuery">没有找到相关书籍。</span>
        <span v-else>暂时还没有人发布书籍。</span>
      </p>
      <button v-if="currentQuery" @click="clearSearch" class="clear-btn">
返回所有书籍</button>
```

```
    </div>

    <div v-else class="book-grid">
      <div v-for="book in books" :key="book.id" class="book-card">
        <router-link :to="{ name: 'book-detail', params: { id: book.id } }" class
="card-link">
          <div class="image-container">
            <img v-if="book.imageUrl" :src="book.imageUrl" :alt="book.title"
class="book-cover">
            <div v-else class="book-cover placeholder">无封面</div>
          </div>
          <div class="book-info">
            <h3 :title="book.title">{{ book.title }}</h3>
            <p class="author">{{ book.author || '未知作者' }}</p>
            <p class="price">￥{{ book.price.toFixed(2) }}</p>
          </div>
        </router-link>
        <button class="add-to-cart-btn" @click="addToCart(book.id)">加入购
物车</button>
      </div>
    </div>
  </div>
</template>

<script>
import axios from 'axios';
import { useAuthStore } from '@/store/auth';

export default {
  name: 'HomeView',
  data() {
    return {
      books: [],
      loading: true,
      error: null,
    };
  },
  computed: {
```

```
  currentQuery() {
    return this.$route.query.q || '';
  }
},
watch: {
  currentQuery(newQuery) {
    this.loadBooks(newQuery);
  }
},
mounted() {
  this.loadBooks(this.currentQuery);
},
methods: {
  loadBooks(query) {
    this.loading = true;
    this.error = null;
    const url = query
      ? `http://localhost:8080/api/books/search`
      : 'http://localhost:8080/api/books';
    const params = query ? { q: query } : {};

    axios.get(url, { params })
      .then(response => {
        this.books = response.data;
        this.loading = false;
      })
      .catch(error => {
        console.error("加载书籍失败:", error);
        this.error = '无法加载书籍列表，请稍后再试。';
        this.loading = false;
      });
  },
  clearSearch() {
    this.$router.push({ path: '/' });
  },
  addToCart(bookId) {
    const authStore = useAuthStore();
    if (!authStore.authToken) {
```

```
        this.$router.push('/login');
        return;
      }
      axios.post(`http://localhost:8080/api/cart/add/${bookId}`, {}, {
        headers: { 'Authorization': `Bearer ${authStore.authToken}` }
      })
      .then(() => { alert('成功加入购物车！'); })
      .catch(error => {
        if (error.response?.status === 401) {
          authStore.logout();
          window.location.reload();
        } else {
          alert(`加入失败: ${error.response?.data?.message || '未知错误'}`);
        }
      });
    }
  }
}
</script>

<style scoped>
.home { max-width: 1200px; margin: 0 auto; padding: 20px; }
h2 { margin-bottom: 20px; color: #333; }
.loading, .error-message, .empty-state { text-align: center; padding: 40px;
color: #666; }
.error-message { color: #d9534f; }
.clear-btn { padding: 8px 16px; background: #eee; border: 1px solid #ccc; b
order-radius: 4px; cursor: pointer; }
.book-grid { display: grid; grid-template-columns: repeat(auto-fill, minmax
(180px, 1fr)); gap: 20px; }
.book-card { background: #fff; border: 1px solid #e1e8ed; border-radius: 8p
x; padding: 15px; display: flex; flex-direction: column; transition: box-shado
w 0.2s; }
.book-card:hover { box-shadow: 0 5px 15px rgba(0,0,0,0.1); }
.card-link { flex-grow: 1; display: flex; flex-direction: column; }
.image-container { height: 200px; display: flex; align-items: center; justify-c
ontent: center; margin-bottom: 15px; }
.book-cover { max-height: 100%; max-width: 100%; object-fit: contain; }
```

```
.book-cover.placeholder { height: 100%; width: 100%; background: #eee; d
isplay: flex; align-items: center; justify-content: center; color: #999; }
.book-info h3 { font-size: 1em; margin: 0 0 5px; white-space: nowrap; overf
low: hidden; text-overflow: ellipsis; }
.book-info .author { color: #555; font-size: 0.9em; margin: 0 0 10px; }
.book-info .price { color: #b12704; font-weight: bold; font-size: 1.2em; marg
in: 0 0 10px; }
.add-to-cart-btn { background: #ffd814; border: 1px solid #fcd200; border-
radius: 20px; padding: 8px; cursor: pointer; width: 100%; }
.add-to-cart-btn:hover { background: #f7ca00; }
</style>
```

3. 登录页 src/views/LoginView.vue

```
<template>
  <div class="auth-container">
    <div class="auth-box">
      <h2>登录</h2>
      <form @submit.prevent="handleLogin">
        <div class="form-group">
          <label>用户名</label>
          <input type="text" v-model="username" required>
        </div>
        <div class="form-group">
          <label>密码</label>
          <input type="password" v-model="password" required>
        </div>
        <button type="submit" :disabled="loading" class="auth-btn">
          {{ loading ? '登录中...' : '登录' }}
        </button>
        <p v-if="errorMessage" class="error-message">{{ errorMessage }}</
p>
      </form>
      <div class="auth-footer">
        新用户？ <router-link to="/register">创建您的账户</router-link>
      </div>
    </div>
```

```
  </div>
</template>

<script>
import axios from 'axios';
import { useAuthStore } from '@/store/auth';

export default {
  name: 'LoginView',
  data() {
    return { username: '', password: '', loading: false, errorMessage: '' };
  },
  methods: {
    handleLogin() {
      this.loading = true;
      this.errorMessage = '';
      const authStore = useAuthStore();

      axios.post('http://localhost:8080/api/auth/signin', {
        username: this.username, password: this.password
      })
      .then(response => {
        authStore.loginSuccess(response.data);
        this.$router.push('/');
      })
      .catch(error => {
        this.loading = false;
        this.errorMessage = (error.response && error.response.status === 40
1)
          ? '用户名或密码错误'
          : '登录失败，请稍后重试';
      });
    }
  }
}
</script>

<style scoped>
```

```
/* 复用样式，可提取到全局 CSS */
.auth-container { display: flex; justify-content: center; align-items: center; min-height: calc(100vh - 60px); }
.auth-box { width: 100%; max-width: 350px; padding: 25px; border: 1px solid #ddd; border-radius: 4px; background: #fff; }
h2 { margin-top: 0; margin-bottom: 20px; }
.form-group { margin-bottom: 15px; text-align: left; }
.form-group label { display: block; margin-bottom: 5px; font-weight: bold; font-size: 0.9em; }
.form-group input { width: 100%; padding: 10px; border: 1px solid #ccc; border-radius: 3px; box-sizing: border-box; }
.form-group input:focus { border-color: #e77600; box-shadow: 0 0 3px 2px rgba(228,121,17,.5); outline: none; }
.auth-btn { width: 100%; padding: 10px; background: #f0c14b; border: 1px solid #a88734; border-radius: 3px; cursor: pointer; }
.auth-btn:hover { background: #ddb347; }
.error-message { color: #d9534f; font-size: 0.9em; margin-top: 15px; }
.auth-footer { margin-top: 20px; font-size: 0.9em; color: #555; text-align: center; }
.auth-footer a { color: #0066c0; }
</style>
```

4. 注册页 src/views/RegisterView.vue

```
<template>
  <div class="auth-container">
    <div class="auth-box">
      <h2>创建账户</h2>
      <form @submit.prevent="handleRegister">
        <div class="form-group">
          <label>用户名</label>
          <input type="text" v-model="username" required minlength="3">
        </div>
        <div class="form-group">
          <label>电子邮箱</label>
          <input type="email" v-model="email" required>
        </div>
```

```
      <div class="form-group">
        <label>密码</label>
        <input type="password" v-model="password" required minlength
="6" placeholder="至少 6 位">
      </div>
      <button type="submit" :disabled="loading" class="auth-btn">
        {{ loading ? '注册中...' : '注册' }}
      </button>
      <p v-if="successMessage" class="success-message">{{ successMe
ssage }}</p>
      <p v-if="errorMessage" class="error-message">{{ errorMessage }}</
p>
    </form>
    <div class="auth-footer">
      已有账户？ <router-link to="/login">登录</router-link>
    </div>
  </div>
 </div>
</template>

<script>
import axios from 'axios';

export default {
 name: 'RegisterView',
 data() {
  return { username: '', email: '', password: '', loading: false, errorMessage:
'', successMessage: '' };
 },
 methods: {
  handleRegister() {
   this.loading = true; this.errorMessage = ''; this.successMessage = '';
   axios.post('http://localhost:8080/api/auth/signup', {
     username: this.username, email: this.email, password: this.password, r
ole: ["user"]
   })
   .then(() => {
     this.loading = false;
```

```
        this.successMessage = '注册成功！请登录。';
      })
      .catch(error => {
        this.loading = false;
        this.errorMessage = error.response?.data || '注册失败，请稍后重试。';
      });
    }
  }
}
</script>

<style scoped>
/* 复用 LoginView 样式，略 */
.auth-container { display: flex; justify-content: center; align-items: center;
min-height: calc(100vh - 60px); }
.auth-box { width: 100%; max-width: 350px; padding: 25px; border: 1px soli
d #ddd; border-radius: 4px; background: #fff; }
h2 { margin-top: 0; margin-bottom: 20px; }
.form-group { margin-bottom: 15px; text-align: left; }
.form-group label { display: block; margin-bottom: 5px; font-weight: bold; f
ont-size: 0.9em; }
.form-group input { width: 100%; padding: 10px; border: 1px solid #ccc; bor
der-radius: 3px; box-sizing: border-box; }
.auth-btn { width: 100%; padding: 10px; background: #f0c14b; border: 1px s
olid #a88734; border-radius: 3px; cursor: pointer; }
.error-message { color: #d9534f; font-size: 0.9em; margin-top: 15px; }
.success-message { color: green; font-size: 0.9em; margin-top: 15px; }
.auth-footer { margin-top: 20px; font-size: 0.9em; color: #555; text-align: c
enter; }
.auth-footer a { color: #0066c0; }
</style>
```

5. 发布页 src/views/PublishView.vue

(支持图片上传)

```
<template>
  <div class="publish-container">
```

```
    <h2>发布新书</h2>
    <form @submit.prevent="handlePublish">
      <div class="form-group">
        <label>书名 *</label>
        <input type="text" v-model="book.title" required>
      </div>
      <div class="form-row">
        <div class="form-group half">
          <label>作者</label>
          <input type="text" v-model="book.author">
        </div>
        <div class="form-group half">
          <label>价格 (元) *</label>
          <input type="number" v-model.number="book.price" required min
="0" step="0.01">
        </div>
      </div>
      <div class="form-group">
        <label>描述</label>
        <textarea v-model="book.description" rows="5"></textarea>
      </div>

      <div class="form-group">
        <label>封面图片</label>
        <div class="upload-area">
          <input type="file" ref="fileInput" @change="handleImageUpload" a
ccept="image/*" style="display: none">
          <button type="button" @click="$refs.fileInput.click()" class="upload
-btn">选择图片</button>
          <span v-if="uploading" class="upload-status">上传中...</span>
        </div>
        <div v-if="uploadError" class="error-message">{{ uploadError }}</div
>
        <div v-if="book.imageUrl" class="image-preview">
          <img :src="book.imageUrl" alt="Preview">
        </div>
      </div>
```

```
    <div v-if="errorMessage" class="error-message">{{ errorMessage }}
</div>
    <div v-if="successMessage" class="success-message">{{ successMe
ssage }}</div>

    <button type="submit" :disabled="loading || uploading" class="submit-
btn">
      {{ loading ? '发布中...' : '确认发布' }}
    </button>
   </form>
  </div>
</template>

<script>
import axios from 'axios';
import { useAuthStore } from '@/store/auth';

export default {
  name: 'PublishView',
  data() {
    return {
      book: { title: '', author: '', price: null, description: '', imageUrl: '' },
      loading: false, uploading: false, errorMessage: '', successMessage: '', u
ploadError: ''
    };
  },
  methods: {
    handleImageUpload(event) {
      const file = event.target.files[0];
      if (!file) return;
      if (file.size > 5 * 1024 * 1024) { this.uploadError = '图片不能超过 5MB'; r
eturn; }

      this.uploading = true; this.uploadError = ''; this.book.imageUrl = '';
      const formData = new FormData();
      formData.append('file', file);

      const authStore = useAuthStore();
```

```
      axios.post('http://localhost:8080/api/upload', formData, {
        headers: { 'Authorization': `Bearer ${authStore.authToken}`, 'Content-T
ype': 'multipart/form-data' }
      })
      .then(res => { this.book.imageUrl = res.data.url; this.uploading = false; })
      .catch(err => {
        this.uploading = false;
        if (err.response?.status === 401) { authStore.logout(); window.locatio
n.reload(); }
        else { this.uploadError = '上传失败，请重试'; }
      });
    },
    handlePublish() {
      if (!this.book.imageUrl) { this.errorMessage = '请上传封面图片'; return; }
      this.loading = true; this.errorMessage = ''; this.successMessage = '';

      const authStore = useAuthStore();
      axios.post('http://localhost:8080/api/books', this.book, {
        headers: { 'Authorization': `Bearer ${authStore.authToken}` }
      })
      .then(() => {
        this.loading = false; this.successMessage = '发布成功！';
        this.book = { title: '', author: '', price: null, description: '', imageUrl: '' };
      })
      .catch(err => {
        this.loading = false;
        if (err.response?.status === 401) { authStore.logout(); window.locatio
n.reload(); }
        else { this.errorMessage = err.response?.data?.message || '发布失败';
}
      });
    }
  }
}
</script>

<style scoped>
.publish-container { max-width: 600px; margin: 30px auto; padding: 30px;
```

```
background: #fff; border: 1px solid #ddd; border-radius: 8px; }
h2 { text-align: center; margin-bottom: 25px; }
.form-group { margin-bottom: 20px; text-align: left; }
.form-row { display: flex; gap: 20px; }
.form-group.half { flex: 1; }
label { display: block; margin-bottom: 8px; font-weight: bold; color: #333; }
input[type="text"], input[type="number"], textarea { width: 100%; padding:
10px; border: 1px solid #ccc; border-radius: 4px; box-sizing: border-box; }
.upload-area { display: flex; align-items: center; gap: 10px; }
.upload-btn { padding: 8px 15px; background: #f0f0f0; border: 1px solid #c
cc; border-radius: 4px; cursor: pointer; }
.image-preview { margin-top: 10px; padding: 5px; border: 1px solid #eee; di
splay: inline-block; }
.image-preview img { max-width: 150px; max-height: 150px; display: block;
}
.submit-btn { width: 100%; padding: 12px; background: #f0c14b; border: 1p
x solid #a88734; border-radius: 4px; cursor: pointer; font-size: 1.1em; }
.error-message { color: #d9534f; text-align: center; margin: 10px 0; }
.success-message { color: green; text-align: center; margin: 10px 0; }
</style>
```

6. 详情页 src/views/BookDetailView.vue

```
<template>
  <div class="detail-container">
    <div v-if="loading" class="loading">加载中...</div>
    <div v-else-if="error" class="error">{{ error }}</div>
    <div v-else-if="book" class="book-detail">
      <div class="img-col">
        <img :src="book.imageUrl || ''" class="main-img" alt="Cover">
      </div>
      <div class="info-col">
        <h1>{{ book.title }}</h1>
        <p class="author">作者: {{ book.author || '未知' }}</p>
        <p class="seller">卖家: {{ book.sellerUsername }}</p>
        <hr>
        <div class="price-block">
```

```
        <span class="price-label">价格:</span>
        <span class="price">￥{{ book.price.toFixed(2) }}</span>
      </div>
      <div class="status-block">
        <span>状态: </span>
        <span :class="['status-badge', book.status]">{{ formatStatus(book.s
tatus) }}</span>
      </div>

      <button v-if="book.status === 'FOR_SALE'" @click="addToCart" clas
s="buy-btn">加入购物车</button>
      <button v-else disabled class="buy-btn disabled">不可购买</button>

      <div class="desc-block">
        <h3>书籍描述</h3>
        <p>{{ book.description || '暂无描述' }}</p>
      </div>
    </div>
  </div>
 </div>
</template>

<script>
import axios from 'axios';
import { useAuthStore } from '@/store/auth';

export default {
 name: 'BookDetailView',
 data() { return { book: null, loading: true, error: null }; },
 mounted() { this.fetchBook(); },
 methods: {
  fetchBook() {
   this.loading = true;
   axios.get(`http://localhost:8080/api/books/${this.$route.params.id}`)
    .then(res => { this.book = res.data; this.loading = false; })
    .catch(err => { this.error = '加载失败'; this.loading = false; });
  },
  formatStatus(s) { return { FOR_SALE: '在售', LOCKED: '交易中', SOLD: '已
```

```
售出' }[s] || s; },
    addToCart() {
      const auth = useAuthStore();
      if (!auth.authToken) return this.$router.push('/login');
      axios.post(`http://localhost:8080/api/cart/add/${this.book.id}`, {}, {
        headers: { 'Authorization': `Bearer ${auth.authToken}` }
      })
      .then(() => { alert('已加入购物车'); this.book.status = 'LOCKED'; })
      .catch(err => {
        if (err.response?.status === 401) { auth.logout(); window.location.reloa
d(); }
        else { alert(err.response?.data || '加入失败'); }
      });
    }
  }
}
</script>

<style scoped>
.detail-container { max-width: 1000px; margin: 30px auto; padding: 20px; b
ackground: #fff; border-radius: 8px; box-shadow: 0 2px 10px rgba(0,0,0,0.0
5); }
.book-detail { display: flex; gap: 40px; }
.img-col { flex: 0 0 350px; }
.main-img { width: 100%; border: 1px solid #eee; border-radius: 4px; }
.info-col { flex: 1; text-align: left; }
h1 { margin-top: 0; color: #333; }
.author, .seller { color: #555; margin: 5px 0; }
hr { margin: 20px 0; border-top: 1px solid #eee; }
.price { color: #b12704; font-size: 1.5em; font-weight: bold; margin-left: 10p
x; }
.status-badge { display: inline-block; padding: 3px 8px; border-radius: 12p
x; font-size: 0.9em; margin-left: 10px; color: white; }
.status-badge.FOR_SALE { background: #42b983; }
.status-badge.LOCKED { background: #f0ad4e; }
.status-badge.SOLD { background: #d9534f; }
.buy-btn { display: block; width: 200px; padding: 12px; margin: 25px 0; bac
kground: #ffd814; border: 1px solid #fcd200; border-radius: 20px; cursor: p
```

```
ointer; font-size: 1.1em; }
.buy-btn.disabled { background: #eee; border-color: #ddd; color: #999; cur
sor: not-allowed; }
.desc-block { margin-top: 30px; }
.desc-block h3 { font-size: 1.2em; border-bottom: 2px solid #eee; padding-
bottom: 10px; margin-bottom: 15px; }
.desc-block p { line-height: 1.6; color: #333; }
</style>
```

7. 购物车 src/views/CartView.vue

```
<template>
  <div class="cart-container">
    <h2>购物车 ({{ cartItems.length }})</h2>
    <div v-if="loading" class="loading">加载中...</div>
    <div v-else-if="cartItems.length === 0" class="empty-cart">
      购物车是空的，<router-link to="/">去选购吧！</router-link>
    </div>
    <div v-else class="cart-content">
      <div class="cart-list">
        <div v-for="item in cartItems" :key="item.id" class="cart-item">
          <img :src="item.imageUrl" class="item-img" alt="Book">
          <div class="item-info">
            <router-link :to="'/book/' + item.id" class="item-title">{{ item.title }}
</router-link>
            <p class="item-author">{{ item.author }}</p>
            <p class="item-seller">卖家: {{ item.sellerUsername }}</p>
            <button @click="removeItem(item.id)" class="delete-link">删除</b
utton>
          </div>
          <div class="item-price">￥{{ item.price.toFixed(2) }}</div>
        </div>
      </div>
      <div class="cart-sidebar">
        <div class="summary-box">
          <h3>订单摘要</h3>
          <div class="summary-row total">
```

```
          <span>总计:</span>
          <span class="total-amount">￥{{ totalPrice.toFixed(2) }}</span>
        </div>
        <button @click="checkout" class="checkout-btn">去结算</button>
      </div>
    </div>
  </div>
</template>

<script>
import axios from 'axios';
import { useAuthStore } from '@/store/auth';

export default {
  name: 'CartView',
  data() { return { cartItems: [], loading: true }; },
  computed: {
    totalPrice() { return this.cartItems.reduce((sum, item) => sum + item.pric
e, 0); }
  },
  mounted() { this.loadCart(); },
  methods: {
    loadCart() {
      const auth = useAuthStore();
      if (!auth.authToken) return;
      this.loading = true;
      axios.get('http://localhost:8080/api/cart', { headers: { 'Authorization': `B
earer ${auth.authToken}` } })
        .then(res => { this.cartItems = res.data; this.loading = false; })
        .catch(err => this.handleError(err));
    },
    removeItem(id) {
      const auth = useAuthStore();
      axios.delete(`http://localhost:8080/api/cart/remove/${id}`, { headers: {
'Authorization': `Bearer ${auth.authToken}` } })
        .then(() => this.loadCart())
        .catch(err => this.handleError(err));
```

```
    },
    checkout() {
      const auth = useAuthStore();
      if (!confirm(`确认支付 ￥${this.totalPrice.toFixed(2)}?`)) return;
      axios.post('http://localhost:8080/api/orders/checkout', {}, { headers: {
'Authorization': `Bearer ${auth.authToken}` } })
        .then(() => { alert('支付成功！'); this.$router.push('/orders'); })
        .catch(err => this.handleError(err));
    },
    handleError(err) {
      this.loading = false;
      if (err.response?.status === 401) {
        useAuthStore().logout();
        window.location.reload();
      } else {
        alert(err.response?.data?.message || '操作失败');
      }
    }
  }
}
</script>

<style scoped>
.cart-container { max-width: 1100px; margin: 30px auto; padding: 20px; }
h2 { text-align: left; margin-bottom: 20px; }
.cart-content { display: flex; gap: 20px; align-items: flex-start; }
.cart-list { flex: 1; background: #fff; padding: 20px; border: 1px solid #ddd;
border-radius: 4px; }
.cart-item { display: flex; padding: 15px 0; border-bottom: 1px solid #eee; }
.item-img { width: 100px; height: 120px; object-fit: contain; margin-right: 20
px; }
.item-info { flex: 1; text-align: left; }
.item-title { font-size: 1.2em; color: #0066c0; font-weight: bold; }
.item-author, .item-seller { color: #555; font-size: 0.9em; margin: 5px 0; }
.delete-link { background: none; border: none; color: #0066c0; cursor: poin
ter; padding: 0; font-size: 0.9em; }
.delete-link:hover { text-decoration: underline; }
.item-price { font-size: 1.2em; font-weight: bold; color: #B12704; text-align:
```

```
right; min-width: 100px; }
.cart-sidebar { width: 300px; }
.summary-box { background: #f3f3f3; padding: 20px; border-radius: 4px; b
order: 1px solid #ddd; }
.summary-row.total { display: flex; justify-content: space-between; font-siz
e: 1.3em; font-weight: bold; margin: 20px 0; }
.total-amount { color: #B12704; }
.checkout-btn { width: 100%; padding: 10px; background: #ffd814; border:
1px solid #fcd200; border-radius: 8px; font-size: 1.1em; cursor: pointer; }
.empty-cart { padding: 50px; text-align: center; font-size: 1.2em; backgrou
nd: #fff; border: 1px solid #ddd; }
</style>
```

8. 订单页 src/views/OrderHistoryView.vue

```
<template>
  <div class="orders-container">
    <h2>我的订单</h2>
    <div v-if="loading">加载中...</div>
    <div v-else-if="orders.length === 0" class="empty-orders">您还没有订
单。</div>
    <div v-else class="order-list">
      <div v-for="order in orders" :key="order.id" class="order-card">
        <div class="order-header">
          <div class="header-col">
            <span class="label">下单时间</span>
            <span>{{ formatDate(order.orderDate) }}</span>
          </div>
          <div class="header-col">
            <span class="label">总计</span>
            <span>￥{{ order.totalPrice.toFixed(2) }}</span>
          </div>
          <div class="header-col order-id">
            <span class="label">订单号 # {{ order.id }}</span>
          </div>
        </div>
        <div class="order-body">
```

```
        <div v-for="item in order.items" :key="item.id" class="order-item">
          <img :src="item.bookImageUrl" class="item-img" alt="Book">
          <div class="item-info">
            <router-link :to="'/book/' + item.bookId" class="item-link">{{ item.
bookTitle }}</router-link>
            <div class="item-price">￥{{ item.priceAtPurchase.toFixed(2) }}</
div>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>

<script>
import axios from 'axios';
import { useAuthStore } from '@/store/auth';

export default {
  name: 'OrderHistoryView',
  data() { return { orders: [], loading: true }; },
  mounted() {
    const auth = useAuthStore();
    if (!auth.authToken) return;
    axios.get('http://localhost:8080/api/orders', { headers: { 'Authorization': `
Bearer ${auth.authToken}` } })
      .then(res => { this.orders = res.data; this.loading = false; })
      .catch(err => {
        this.loading = false;
        if (err.response?.status === 401) { auth.logout(); window.location.reloa
d(); }
      });
  },
  methods: {
    formatDate(dateStr) { return new Date(dateStr).toLocaleString(); }
  }
}
```

```
</script>

<style scoped>
.orders-container { max-width: 900px; margin: 30px auto; padding: 20px; }
h2 { text-align: left; margin-bottom: 20px; }
.order-card { border: 1px solid #ddd; border-radius: 8px; margin-bottom: 20
px; overflow: hidden; background: #fff; }
.order-header { background: #f6f6f6; padding: 15px; display: flex; gap: 40p
x; color: #555; font-size: 0.9em; border-bottom: 1px solid #ddd; }
.header-col { display: flex; flex-direction: column; }
.header-col .label { font-size: 0.85em; text-transform: uppercase; margin-b
ottom: 5px; }
.order-id { margin-left: auto; text-align: right; }
.order-body { padding: 20px; }
.order-item { display: flex; margin-bottom: 15px; }
.item-img { width: 60px; height: 80px; object-fit: contain; margin-right: 20p
x; }
.item-info { display: flex; flex-direction: column; justify-content: center; }
.item-link { color: #0066c0; font-weight: bold; text-decoration: none; }
.item-link:hover { text-decoration: underline; show: inline-block; }
.item-price { color: #B12704; font-weight: bold; margin-top: 5px; }
.empty-orders { padding: 40px; text-align: center; background: #fff; borde
r: 1px solid #ddd; border-radius: 8px; }
</style>
```

第 9 部分：前端 - 管理后台 (Admin Modules)

这些文件都位于 `src` 目录下。

1. 后台布局 `src/layouts/AdminLayout.vue`

```
<template>
 <div class="admin-layout">
  <nav class="admin-sidebar">
   <h3>管理菜单</h3>
   <ul>
    <li><router-link to="/admin/dashboard">数据看板</router-link></li>
    <li><router-link to="/admin/users">用户管理</router-link></li>
    <li><router-link to="/admin/books">书籍管理</router-link></li>
```

```
        <li><router-link to="/">返回前台首页</router-link></li>
      </ul>
    </nav>
    <main class="admin-content">
      <router-view />
    </main>
  </div>
</template>

<script>
export default {
  name: 'AdminLayout'
}
</script>

<style scoped>
.admin-layout { display: flex; min-height: calc(100vh - 60px); }
.admin-sidebar { width: 200px; background-color: #2c3e50; color: white; p
adding: 20px; flex-shrink: 0; }
.admin-sidebar h3 { margin-top: 0; color: #febd69; border-bottom: 1px soli
d #34495e; padding-bottom: 10px; }
.admin-sidebar ul { list-style: none; padding: 0; margin: 0; }
.admin-sidebar li { margin-bottom: 5px; }
.admin-sidebar a { color: #b8c7d9; text-decoration: none; display: block; p
adding: 10px; border-radius: 4px; transition: all 0.3s; }
.admin-sidebar a:hover, .admin-sidebar a.router-link-active { background-c
olor: #34495e; color: white; }
.admin-content { flex-grow: 1; padding: 30px; background-color: #f5f7fa; o
verflow-y: auto; }
</style>
```

2. 数据看板 src/views/admin/AdminDashboard.vue

(修复了 .then 和 catch)

```
<template>
  <div class="dashboard">
    <h2>数据看板</h2>
```

```
    <div v-if="loading">正在加载统计数据...</div>
    <div v-else-if="error" class="error-message">{{ error }}</div>
    <div v-else>
      <div class="stats-grid">
        <div class="stat-card">
          <h3>总用户数</h3>
          <p>{{ stats.totalUsers }}</p>
        </div>
        <div class="stat-card">
          <h3>总书籍数</h3>
          <p>{{ stats.totalBooks }}</p>
        </div>
        <div class="stat-card">
          <h3>总订单数</h3>
          <p>{{ stats.totalOrders }}</p>
        </div>
      </div>
      <div class="chart-container" v-if="chartDataReady">
        <h3>统计概览</h3>
        <Bar :data="chartData" :options="chartOptions" />
      </div>
    </div>
  </div>
</template>

<script>
import axios from 'axios';
import { useAuthStore } from '@/store/auth';
import { Bar } from 'vue-chartjs';
import { Chart as ChartJS, Title, Tooltip, Legend, BarElement, CategoryScal
e, LinearScale } from 'chart.js';
ChartJS.register(Title, Tooltip, Legend, BarElement, CategoryScale, LinearS
cale);

export default {
  name: 'AdminDashboard',
  components: { Bar },
  data() {
```

```
    return {
      stats: null, loading: true, error: null, chartDataReady: false,
      chartData: { labels: ['用户数', '书籍数', '订单数'], datasets: [{ label: '总量',
backgroundColor: ['#42A5F5', '#66BB6A', '#FFA726'], data: [] }] },
      chartOptions: { responsive: true, maintainAspectRatio: false, plugins: { l
egend: { display: false } }, scales: { y: { beginAtZero: true, ticks: { precision:
0 } } } }
    };
  },
  mounted() { this.fetchStats(); },
  methods: {
    fetchStats() {
      this.loading = true; this.error = null; this.chartDataReady = false;
      const auth = useAuthStore();
      if (!auth.authToken) return;

      axios.get('http://localhost:8080/api/admin/stats/dashboard', { headers:
{ 'Authorization': `Bearer ${auth.authToken}` } })
      .then(res => {
        this.stats = res.data;
        this.chartData.datasets[0].data = [this.stats.totalUsers, this.stats.totalB
ooks, this.stats.totalOrders];
        this.chartDataReady = true;
        this.loading = false;
      })
      .catch(err => {
        this.loading = false;
        if (err.response?.status === 401) { auth.logout(); window.location.reloa
d(); }
        else { this.error = '无法加载统计数据'; }
      });
    }
  }
}
</script>

<style scoped>
.dashboard h2 { text-align: left; margin-bottom: 20px; }
```

```
.stats-grid { display: grid; grid-template-columns: repeat(auto-fill, minmax
(200px, 1fr)); gap: 20px; }
.stat-card { background: #fff; padding: 20px; border-radius: 8px; text-align:
center; box-shadow: 0 2px 4px rgba(0,0,0,0.05); }
.stat-card h3 { margin: 0; color: #555; font-size: 1.1em; }
.stat-card p { font-size: 2.5em; font-weight: bold; color: #2c3e50; margin: 1
0px 0 0; }
.chart-container { margin-top: 30px; padding: 20px; background: #fff; bord
er-radius: 8px; height: 400px; box-shadow: 0 2px 4px rgba(0,0,0,0.05); }
.chart-container h3 { text-align: center; margin-top: 0; }
.error-message { color: red; }
</style>
```

3. 用户管理 src/views/admin/AdminUserManagement.vue

(修复了 .then, catch, ESLint)

```
<template>
  <div class="user-management">
    <h2>用户管理</h2>
    <div v-if="loading">加载中...</div>
    <div v-else-if="error" class="error">{{ error }}</div>
    <table v-else>
      <thead><tr><th>ID</th><th>用户名</th><th>邮箱</th><th>角色</th>
<th>操作</th></tr></thead>
      <tbody>
        <tr v-for="user in users" :key="user.id">
          <td>{{ user.id }}</td><td>{{ user.username }}</td><td>{{ user.email
}}</td><td>{{ user.roles.join(', ') }}</td>
          <td><button @click="deleteUser(user.id, user.username)" class="del
-btn">删除</button></td>
        </tr>
      </tbody>
    </table>
  </div>
</template>

<script>
```

```
import axios from 'axios';
import { useAuthStore } from '@/store/auth';

export default {
  name: 'AdminUserManagement',
  data() { return { users: [], loading: true, error: null }; },
  mounted() { this.fetchUsers(); },
  methods: {
    fetchUsers() {
      this.loading = true;
      const auth = useAuthStore();
      axios.get('http://localhost:8080/api/admin/users', { headers: { 'Authoriz
ation': `Bearer ${auth.authToken}` } })
      .then(res ⇒ { this.users = res.data; this.loading = false; })
      .catch(err ⇒ {
        this.loading = false;
        if (err.response?.status === 401) { auth.logout(); window.location.relo
ad(); }
        else { this.error = '加载失败'; }
      });
    },
    deleteUser(id, name) {
      const auth = useAuthStore();
      if (auth.user?.id === id) { alert("不能删除自己！"); return; }
      if (!confirm(`确定删除用户 ${name}?`)) return;
      axios.delete(`http://localhost:8080/api/admin/users/${id}`, { headers: {
'Authorization': `Bearer ${auth.authToken}` } })
      .then(() ⇒ {
        /* eslint-disable no-unused-vars */
        alert('删除成功');
        this.users = this.users.filter(u ⇒ u.id !== id);
        /* eslint-enable no-unused-vars */
      })
      .catch(err ⇒ {
        if (err.response?.status === 401) { auth.logout(); window.location.relo
ad(); }
        else { alert('删除失败'); }
      });
```

```
    }
  }
}
</script>

<style scoped>
.user-management { background: #fff; padding: 20px; border-radius: 8px;
box-shadow: 0 2px 4px rgba(0,0,0,0.05); }
h2 { margin-top: 0; margin-bottom: 20px; }
table { width: 100%; border-collapse: collapse; }
th, td { text-align: left; padding: 12px; border-bottom: 1px solid #eee; }
th { background-color: #f8f9fa; color: #2c3e50; }
.del-btn { padding: 6px 12px; background: #dc3545; color: white; border: n
one; border-radius: 4px; cursor: pointer; }
.del-btn:hover { background: #c82333; }
.error { color: #dc3545; }
</style>
```

4. 书籍管理 src/views/admin/AdminBookManagement.vue

(修复了 .then, catch, ESLint)

```
<template>
  <div class="book-management">
    <h2>书籍管理</h2>
    <div v-if="loading">加载中...</div>
    <div v-else-if="error" class="error">{{ error }}</div>
    <table v-else>
      <thead><tr><th>ID</th><th>书名</th><th>作者</th><th>价格</th><th
>状态</th><th>卖家</th><th>操作</th></tr></thead>
      <tbody>
        <tr v-for="book in books" :key="book.id">
          <td>{{ book.id }}</td><td>{{ book.title }}</td><td>{{ book.author }}
</td><td>￥{{ book.price }}</td>
          <td><span :class="['status', book.status]">{{ book.status }}</span>
</td>
          <td>{{ book.sellerUsername }}</td>
          <td><button @click="deleteBook(book.id, book.title)" class="del-bt
```

```
n">删除</button></td>
        </tr>
      </tbody>
    </table>
  </div>
</template>

<script>
import axios from 'axios';
import { useAuthStore } from '@/store/auth';

export default {
  name: 'AdminBookManagement',
  data() { return { books: [], loading: true, error: null }; },
  mounted() { this.fetchBooks(); },
  methods: {
    fetchBooks() {
      this.loading = true;
      const auth = useAuthStore();
      axios.get('http://localhost:8080/api/admin/books', { headers: { 'Authoriz
ation': `Bearer ${auth.authToken}` } })
      .then(res => { this.books = res.data; this.loading = false; })
      .catch(err => {
        this.loading = false;
        if (err.response?.status === 401) { auth.logout(); window.location.relo
ad(); }
        else { this.error = '加载失败'; }
      });
    },
    deleteBook(id, title) {
      const auth = useAuthStore();
      if (!confirm(`确定删除书籍 ${title}?`)) return;
      axios.delete(`http://localhost:8080/api/admin/books/${id}`, { headers: {
'Authorization': `Bearer ${auth.authToken}` } })
      .then(() => {
        /* eslint-disable no-unused-vars */
        alert('删除成功');
        this.books = this.books.filter(b => b.id !== id);
```

```
        /* eslint-enable no-unused-vars */
      })
      .catch(err => {
        if (err.response?.status === 401) { auth.logout(); window.location.reload(); }
        else { alert(err.response?.data?.message || '删除失败(可能已被订单引用)'); }
      });
    }
  }
}
</script>

<style scoped>
.book-management { background: #fff; padding: 20px; border-radius: 8px; box-shadow: 0 2px 4px rgba(0,0,0,0.05); }
h2 { margin-top: 0; margin-bottom: 20px; }
table { width: 100%; border-collapse: collapse; }
th, td { text-align: left; padding: 12px; border-bottom: 1px solid #eee; }
th { background-color: #f8f9fa; color: #2c3e50; }
.status { font-weight: bold; font-size: 0.9em; }
.status.FOR_SALE { color: #42b983; }
.status.LOCKED { color: #f0ad4e; }
.status.SOLD { color: #d9534f; }
.del-btn { padding: 6px 12px; background: #dc3545; color: white; border: none; border-radius: 4px; cursor: pointer; }
.del-btn:hover { background: #c82333; }
.error { color: #dc3545; }
</style>
```

总结

现在，你手里已经有了**所有 15 个核心前端文件**的**最终、完整、无 Bug 版本**。

只要你把它们正确地放入对应的文件夹，你的前端项目就是一个**功能完备、代码整洁、且具备良好错误处理机制**的现代化 Vue 3 应用了！

项目：自助装机(攒机)配置系统

1. 技术栈升级与对比

在开始需求分析前，先明确核心变化。

技术点	二手书项目 (旧)	攒机项目 (新)	备注
核心框架	Spring Boot 3	Spring Boot 3	不变
ORM 框架	**Spring Data JPA**	**MyBatis** (推荐使用 **MyBatis-Plus**)	**核心变化**。需要手写 SQL (MyBatis) 或使用 MP 提供的便捷方法。
数据库	MySQL	MySQL	不变
安全框架	Spring Security + JWT	Spring Security + JWT	可以复用之前的代码
前端框架	Vue 3 + Pinia + Axios	Vue 3 + Pinia + Axios	不变
UI 组件库	(手写 CSS)	**推荐引入 Element Plus**	攒机系统表单和表格较多，引入 UI 库能极大提高效率。

> 建议：强烈建议使用 MyBatis-Plus。它在 MyBatis 的基础上只做增强不做改变，提供了类似 JPA 的“无 SQL”基础增删改查能力，同时又能让你随时手写复杂 SQL，是目前国内最流行的方案。

---

2. 需求分析 (Requirements Analysis)

我们将系统分为两个大角色：**普通用户**和**管理员**。

2.1 核心业务概念

在二手书系统中，核心是“书”和“订单”。
在攒机系统中，核心是：

1. **硬件 (Hardware)**：CPU、显卡、主板、内存、硬盘、机箱、电源、散热器等。
2. **分类 (Category)**：硬件必须有明确分类（不能把 CPU 插到内存槽上）。
3. **配置单 (Build/Configuration)**：用户创建的一个“装机方案”，它包含了一组特定的硬件列表。

2.2 功能模块划分

🟢 普通用户模块 (前端商城)

1. **用户认证**：注册、登录、退出 (复用之前逻辑)。
2. **硬件浏览**：
    - 查看所有硬件。
    - **按分类筛选**（核心功能：只看 CPU，只看显卡等）。
    - 搜索硬件（按名称）。
    - 查看硬件详情（参数、价格）。
3. **自助攒机 (核心功能)**：
    - **创建新配置单**：开始一个新的装机方案。
    - **选择硬件**：在配置单的特定“槽位”上选择硬件（例如：在 CPU 槽位上点击“选择”，弹窗列出所有 CPU 供选择）。
    - **实时计算总价**：选择硬件后，自动计算当前配置单的总价格。
    - **(进阶) 简单兼容性检查**：(可选) 例如，提示用户“选择了 Intel CPU 必须搭配 Intel 主板”。
4. **我的配置单**：
    - 保存配置单：将当前选择的硬件组合保存到数据库。
    - 查看我的配置单列表。
    - 查看/修改/删除某一特定的配置单。
5. **(可选) 分享/评论**：可以将自己的配置单公开分享给其他人看。

🔵 管理员模块 (后台管理)

1. **数据看板**：总用户数、总硬件数、总配置单数。
2. **分类管理**：管理硬件的分类（如：添加“水冷散热器”分类）。
3. **硬件管理**：
    - 硬件列表（带分页、搜索、筛选）。
    - **发布/编辑硬件**：录入硬件名称、分类、价格、详细参数、封面图片。
    - 上下架硬件。
4. **用户管理**：查看、删除用户。
5. **配置单管理**：查看所有用户创建的配置单（用于了解热门搭配）。

3. 页面规划 (Page Planning)

根据需求，我们需要创建以下前端页面。

3.1 公共/用户页面 (Front Office)

页面文件 (`src/views/...`)	路由路径	功能描述
`HomeView.vue`	`/`	**首页**。展示热门硬件、推荐配置单。
`LoginView.vue`	`/login`	登录页 (复用)。
`RegisterView.vue`	`/register`	注册页 (复用)。
`HardwareListView.vue`	`/hardware`	**硬件库**。左侧是分类列表(CPU/GPU...)，右侧是硬件卡片网格。带搜索和排序功能。
`HardwareDetailView.vue`	`/hardware/:id`	**硬件详情页**。展示硬件的大图、详细参数规格、价格。
`BuildPCView.vue`	`/build/new` 或 `/build/:id/edit`	**攒机工作室 (核心)**。一个大表单，列出所有必要的配件槽位（CPU槽、主板槽...），每个槽位有“选择”按钮。
`MyBuildsView.vue`	`/my-builds`	**我的配置单列表**。列出用户保存的所有装机方案。
`BuildDetailView.vue`	`/build/:id`	**配置单详情展示页**。以漂亮的清单形式展示某个配置单的所有配件和总价，可供分享。

3.2 管理员后台页面 (Back Office)

页面文件 (`src/views/admin/...`)	路由路径	功能描述
`AdminDashboard.vue`	`/admin/dashboard`	**数据看板**。统计图表。
`AdminCategory.vue`	`/admin/categories`	**分类管理**。添加/修改硬件分类（如 CPU, 显卡, 内存）。
`AdminHardwareList.vue`	`/admin/hardware`	**硬件列表管理**。表格展示，带搜索、分页、编辑/删除按钮。
`AdminHardwareEdit.vue`	`/admin/hardware/edit/:id` (或复用弹窗)	**发布/编辑硬件**。复杂的表单，需要选择分类、上传图片、填写参数。
`AdminUserList.vue`	`/admin/users`	**用户管理**。

4. 数据库设计初探 (Database Preview)

为了让你对用 MyBatis 操作什么有个底，这里简单列出核心表结构的设计思路：

1. `users` (用户表): id, username, password, role...
2. `categories` (硬件分类表): id, name (例如: "CPU", "显卡")
3. `hardware` (硬件表): id, name, price, description, image_url, **category_id (外键)**
4. `builds` (配置单主表): id, user_id, name (配置单名字，如“我的吃鸡电脑”), total_price, create_time
5. `build_items` (配置单详情表 - 核心): id, **build_id (外键)**, **hardware_id (外键)**

总结

这个新项目比二手书项目逻辑更复杂一点（特别是一对多的配置单关系），而且引入 **MyBatis** 会让你体验到完全不同的后端开发模式（你需要更多地关注 SQL 如何编写，或者如何使用 MyBatis-Plus 的 Wrapper 来构建查询）。