一、 软件架构设计方案

1.1 系统功能概述

在线商城系统主要面向普通用户、商家以及管理员三类角色，提供完整的电子商务功能。用户可以完成注册登录、个人资料维护、商品浏览与搜索、购物车管理、订单创建与支付等操作；商家可以提交入驻申请、管理店铺信息和商品库存；管理员可以查看平台统计数据、审核商家申请、管理订单等。系统支持商品分类浏览、商品详情查看、购物车增删改、订单支付（二维码支付）、订单状态更新（确认收货）等电商核心功能。此外，系统提供用户头像和商家 Logo 上传功能，以及后台统计分析等功能。该系统实现了一个前后端分离的电子商城平台，具有用户端、商家端和管理员端的功能模块。

1.2 架构概览

系统采用典型的分层架构和 MVC 设计思想，将应用分为表现层、业务逻辑层和数据访问层。后端基于 Spring Boot 框架实现，使用 Spring MVC 来处理 HTTP 请求，并使用 MyBatis-Plus 进行数据访问。前端采用 Vue.js 单页面应用与后端交互，双方通过 RESTful API 通信。总体架构如下：

- **客户端（前端）**：使用 Vue.js 开发的 Web 界面，负责用户交互和数据展示，通过 HTTP 请求调用后端 API。
- **控制层（Controller）**：后端 Spring Boot 控制器接收客户端请求，进行身份认证（例如通过 JWT 验证令牌）和参数校验，然后委托业务逻辑层处理请求。
- **业务逻辑层（Service）**：处理具体的业务规则和流程，通常以 Service 类形式存在。该层调用数据访问层获取或更新实体数据，并封装业务结果。
- **数据访问层（Model/Repository）**：包括实体类（Entity）、数据传输对象（DTO）以及 MyBatis 的 Mapper 接口。实体类定义了数据库表映射和业务数据模型，Mapper 接口负责与数据库交互（增删改查），DTO 用于前后端数据传输。
- **数据库和静态资源**：数据库用于持久化存储用户、商品、订单等信息；系统还维护静态文件存储（如商品图片、商家 Logo 等）在服务器的 uploads 目录下。

系统采用前后端分离架构，前端和后端职责明确，且整体设计遵循 MVC 模式：数据模型（Model）层、视图（View）层和控制（Controller）层各司其职，实现功能模块化和高内聚、低耦合的结构。

1.3 模块划分

Model 层

Model 层负责系统的业务数据模型和数据访问，主要包括实体类（Entity）、数据访问接口（Mapper）和数据传输对象（DTO）。

- **实体类（Entity）**：如 User、Product、Order、Cart、Category、Seller、ProductReview 等，使用 MyBatis-Plus 注解映射到数据库表，表示系统核心数据结构。这些实体封装了业务数据字段和关系，例如用户信息、商品详情、订单明细等。
- **数据访问（Mapper）**：对应每个实体有 Mapper 接口（如 UserMapper、ProductMapper 等），定义数据库操作方法。MyBatis-Plus 自动实现 CRUD 功能，Service 层通过 Mapper 与数据库交互。
- **数据传输对象（DTO/VO）**：为了前后端传输和参数封装，系统使用 DTO（如 ProductDTO、OrderQueryDTO、SellerApplyDTO 等）和 VO（如统一返回结果 Result、分页结果 PageResult）。DTO 负责封装控制器接收的请求参数或返回的数据结构，VO 用于封装返回给客户端的统一响应格式。
- **业务逻辑（Service）**：虽然 Service 层不属于传统 MVC 三层中的“Model”部分，但它承载了业务规则，在 MVC 架构中常与 Model 层相辅相成。Service 层接口（如 ProductService、OrderService 等）定义业务功能，具体实现类（*ServiceImpl）包含交易流程、数据校验等逻辑。Service 调用 Mapper 完成数据持久化。

View 层

View 层负责与用户交互的用户界面部分。系统使用 Vue.js 构建前端单页面应用（SPA），主要特点包括：

- **Vue 组件和页面**：位于 frontend/src/views 目录下，每个页面对应一个 Vue 组件，例如登录页、商品列表页、购物车页、订单页、商家申请页等。组件负责展示数据和处理用户事件。
- **前端路由和状态**：使用 Vue Router 管理路由，实现页面导航；可能使用 Pinia 或 Vuex 进行状态管理，维护登录状态、购物车状态等。
- **API 调用**：位于 frontend/src/api 的模块封装了调用后端接口的功能，例如 auth.ts 处理登录注册，product.ts 处理商品相关 API，order.ts 处理订单 API 等。组件通过这些 API 模块发送 HTTP 请求，与后端通信。
- **静态资源**：前端项目包含用于界面展示的静态资源，如图片、样式文件等。后端 uploads 目录中的商品图片和商家 Logo 会被前端引用或展示。

视图层不直接访问数据库，只通过调用后端的 Controller 提供的 RESTful 接口来获取或提交数据。通过 Vue 组件、模板和绑定语法，视图层将 Model 层返回的数据渲染成用户可见的页面。

Controller 层

Controller 层位于后端，主要由标注为@RestController 或@Controller 的类组成，负责接收和处理客户端（前端）发起的 HTTP 请求，以及向前端返回数据。主要特点包括：

- **请求路由**：系统中各控制器类（如 AuthController、UserController、ProductController、CartController、OrderController、SellerController、AdminController 等）通过@RequestMapping 或 @GetMapping/@PostMapping 等注解定义 RESTful 接口的 URL 路径和请求方法。例如，/auth/login 用于用户登录，/products 用于获取商品列表，/cart/add 用于添加购物车项等。
- **功能职责**：各控制器根据功能模块区分职责：
    - AuthController：处理用户认证相关请求，如注册、登录、检查 Token 有效性等，通常调用 UserService 和安全组件完成 JWT 令牌的生成与验证。
    - UserController：处理用户信息管理，如获取和更新个人资料、上传头像等功能。
    - ProductController：提供商品相关接口，如查询商品列表、查看商品详情、商品上架下架等（商家权限）。
    - CategoryController：提供商品分类接口，返回分类列表供前端展示。
    - CartController：处理购物车操作，如添加商品到购物车、查看购物车列表、删除购物项等功能。
    - OrderController：管理订单流程，包括创建订单、支付订单（二维码支付）、查询订单状态、确认收货等。
    - SellerController：商家相关功能，如提交入驻申请、管理店铺和商品库存、上传商家支付二维码等。
    - AdminController：后台管理接口，如查看平台概览统计、审核商家入驻申请、管理订单等管理员功能。
- **业务协同**：控制器接收到请求后，一般完成参数校验和身份验证（如通过拦截器验证 JWT），然后调用对应的 Service 方法执行业务逻辑。处理结果返回时，通常将数据封装在统一的响应对象（如 Result）中，返回给前端。
- **过滤器与拦截器**：系统配置了安全组件（如 JwtAuthenticationFilter 和 SecurityConfig），在请求达到 Controller 前进行身份认证和权限校验；日志或请求拦截（如 RequestLoggingInterceptor）也可在控制器层前后执行辅助功能。

Controller 层将前端请求映射到后端业务，承担着业务流程的触发作用，是系统连接前端与后端服务的关键枢纽。

1.4 组件交互流程

以下以“用户创建订单并支付”这一流程为例，说明各组件之间的交互流程：

1. **前端页面操作**：用户在浏览器中选择商品并提交购物车，进入结算页面，然后点击“确认支付”按钮。前端 Vue 应用收集购物信息，调用订单接口（例如发送 POST /orders）请求创建订单。
2. **请求到达控制器**：该请求首先经过 Spring Security 配置的 JWT 验证过滤器，如验证通过，则由 OrderController 的对应方法接收。控制器方法解析请求参数（可能是 CreateOrderDTO），并获取当前用户身份信息。
3. **业务逻辑处理**：OrderController 调用 OrderService 的创建订单方法。OrderService 首先校验库存和用户信息，然后调用 CartMapper 查询购物车数据、OrderMapper 将订单数据持久化到数据库，同时生成订单号。
4. **持久化数据**：OrderService 使用 Mapper 接口向数据库写入订单表和订单明细表（包括订单项 OrderItem），同时可能更新商品库存和清空购物车等操作。上述操作通过 MyBatis-Plus 提供的 CRUD 功能完成，事务性保证数据一致。
5. **返回结果**：创建订单成功后，Service 返回订单信息或订单号给控制器。OrderController 将结果封装到统一响应体（Result 对象）中并返回给前端。
6. **前端发起支付**：前端收到订单号后，显示支付二维码页面，并调用支付 API（如/order/pay）。控制器接收支付请求后，调用业务层生成或校验支付二维码（可能调用第三方支付服务），并返回二维码图片信息给前端。
7. **支付成功确认**：用户扫码支付后，前端会调用后端支付成功回调接口。OrderController 确认支付状态，调用 OrderService 修改订单状态为“已支付”。
8. **确认收货**：当用户收到商品后，前端调用/order/confirm 接口，OrderController 将请求转交给 OrderService，服务层更新订单状态为“已完成”。

以上流程中，各层的协同关系清晰可见：前端视图层与用户交互并调用后端接口，控制器层接收请求并调用业务层逻辑，业务层操作数据访问层与数据库交互，最终结果返回给前端。整个过程遵循 MVC 分层，各层职责明确，且通过 RESTful 接口和统一的数据传输格式完成模块之间的通信。

1.5 总结

在线商城系统采用 Spring Boot + MyBatis-Plus + Vue 的技术栈，实现了前后端分离的 MVC 架构。系统架构将表现层（Vue 视图）、控制层（SpringMVC 控制器）和模型层（实体、服务、数据访问）有机地分离，各模块功能清晰：控制器处理请求并协调业务逻辑，服务层实现核心业务规则，模型层负责数据表示和存取。这样的设计提高了系统的可维护性和可扩展性，前端界面和后端服务解耦，便于并行开发和独立测试。通过对模型、视图、控制器的明确划分，技

术团队可以快速定位各自职责区，并对系统进行功能扩展或优化，从而保证了整体架构的清晰性和稳定性。