微服务项目提案

一、目的

本项目旨在设计和实现一个功能全面的在线购物平台，通过采用微服务架构，实现系统的高可扩展性和灵活性。具体目标包括：

1. **微服务架构设计与集成**：通过设计微服务架构，深入理解和应用微服务的设计原则、服务划分、API设计及其开发与集成，提升系统的模块化和可维护性。
2. **常用编程框架的实践**：运用流行的编程框架（如Spring Boot、Spring Cloud等）来创建微服务和REST API，积累实际开发经验，提升开发效率和代码质量。

3. **服务组合与集成技术应用**：通过创建适配器和包装器，使用面向API的集成方式，或者结合BPMN和工作流引擎，实现服务的高效组合，满足复杂业务流程的需求。
4. **数据格式与技术应用**：在软件开发中应用XML、JSON、GPB（Google Protocol Buffers）等数据格式，掌握不同数据格式的使用场景及其优势，确保数据传输的高效性和兼容性。
5. **构建便捷的在线购物平台**：通过微服务设计的服务为中心的解决方案，打造一个用户友好、功能全面的在线购物平台，满足不同用户（客户、商家、管理员）的多样化需求，并具备良好的扩展性以适应未来业务的变化。
6. **设计模式与架构样式的应用**：在微服务和API主导的架构中，应用常见的设计模式（如单例模式、工厂模式、观察者模式等）和架构样式（如事件驱动架构、CQRS等），提升系统的健壮性和可维护性。

二、范围

本项目旨在设计和实现一个功能完善的在线购物平台，采用微服务架构以满足不同用户群体（客户、商家、管理员）的需求。项目的主要内容涵盖系统架构设计、API设计、数据传输、安全性、集成与工作流管理等方面，确保平台具备高可用性、高扩展性和高安全性。

2.1 项目包含的内容

- **微服务架构**：
  - **服务划分**：根据业务功能将系统划分为多个独立的微服务，如用户服务、商品服务、订单服务、支付服务等。
  - **服务发现与注册**：采用服务发现机制（如Eureka、Nacos）实现服务的动态注册与发现，确保服务间的通信高效稳定。
  - **负载均衡与高可用性**：通过负载均衡器（如Spring Cloud Gateway）分配请求，确保系统在高并发下的稳定性和高可用性。
  - **容错处理**：集成熔断器（如Hystrix）等机制，提高系统的容错能力，防止单个服务的故障导致整个系统的崩溃。
- **API设计**：
  - **RESTful API设计原则**：遵循RESTful风格，设计统一、简洁的API接口，确保各模块之间的高效通信。
  - **API文档与规范**：使用工具（如Swagger）生成详细的API文档，规范API的使用，方便前后端的协作开发。
  - **版本管理**：设计合理的API版本管理策略，确保API的迭代更新不会影响现有服务的稳定运行。
- **数据格式**：
  - **JSON/XML**：根据不同场景选择合适的数据格式，JSON用于前后端数据交互，XML用于需要复杂数据结构的场景。
  - **数据序列化**：应用GPB（Google Protocol Buffers）等高效序列化技术，优化数据传输效率，减少带宽占用。
- **安全性**：
  - **用户认证与授权**：实施OAuth 2.0等认证机制，确保用户身份的真实性和操作的合法性。
  - **数据加密**：对敏感数据（如用户密码、支付信息）进行加密存储和传输，防止数据泄露。
  - **安全审计与监控**：建立安全审计机制，实时监控系统安全状态，及时发现和应对安全威胁。
  - **合规性**：遵循相关法律法规（如GDPR、PCI-DSS等），确保平台的合法合规运营。

- **集成与工作流**：
  - **服务组合**：通过创建适配器和包装器，将不同的微服务进行组合，满足复杂业务需求。
  - **BPMN与工作流引擎**：集成BPMN（业务流程模型与符号）和工作流引擎（如Activiti），实现业务流程的可视化和自动化管理。
  - **API网关与消息队列**：利用API网关统一管理外部请求，采用消息队列（如RabbitMQ、Kafka）实现服务间的异步通信，提升系统的响应速度和解耦性。

2.2 项目不包含的内容

- **线下商家管理和实体店铺运营**：本项目专注于线上平台的开发，不涉及线下商家的管理及实体店铺的运营。
- **第三方支付平台的具体实施细节**：仅支持与第三方支付平台的接口对接，不涵盖支付平台的内部实现细节。
- **高级数据分析和机器学习功能**：不包括复杂的数据分析和机器学习功能，这些可以作为后续迭代的考虑范围。

三、项目内容

3.1 微服务架构

采用微服务架构，将系统划分为多个独立的服务，每个服务负责特定的业务功能。这种架构能够提高系统的灵活性和可扩展性，便于团队协作开发和独立部署。主要涉及的微服务包括：

- **用户服务**：负责用户的注册、登录、个人信息管理等功能。
- **商品服务**：管理商品的展示、分类、库存等信息。
- **订单服务**：处理用户的订单创建、取消、查看等操作。
- **购物车服务**：管理用户的购物车，支持添加、删除、修改商品。
- **支付服务**：处理安全支付相关功能，支持多种支付方式接口对接。
- **评论服务**：管理用户对商品的评论与评分。
- **商家服务**：负责商家的入驻、商铺信息管理、商品管理等。
- **管理员服务**：提供用户管理、商品审核、订单监控等功能。

3.2 API设计

系统各个微服务之间通过RESTful API进行通信，遵循统一的设计规范，确保接口的一致性和易用性。主要设计原则包括：

- **资源导向**：以资源为中心，使用HTTP方法（GET、POST、PUT、DELETE）对应不同的操作。
- **状态无关**：每个请求包含完成操作所需的所有信息，服务器不保存客户端的状态。
- **统一返回格式**：所有API返回统一的响应格式（如JSON），包含状态码、消息和数据。
- **错误处理**：设计合理的错误码和错误信息，便于前端进行错误处理和提示。

3.3 数据格式

采用多种数据格式进行数据传输，以满足不同的需求：

- **JSON**：主要用于前后端的数据交互，因其轻量级和易读性高，广泛应用于Web应用。
- **XML**：用于需要复杂数据结构和严格格式的场景，如与某些外部系统的集成。
- **GPB（Google Protocol Buffers）**：用于服务间高效的数据传输，提升通信效率，减少带宽占用。

3.4 安全性

系统的安全性设计涵盖以下几个方面：

- **身份认证与授权**：采用JWT（JSON Web Token）或OAuth 2.0机制，实现用户的身份认证和权限控制。
- **数据加密**：对敏感数据进行加密存储（如AES、RSA）和传输（如TLS/SSL），确保数据的机密性和完整性。
- **防护措施**：防范常见的Web安全威胁，如SQL注入、跨站脚本（XSS）、跨站请求伪造（CSRF）等。
- **安全审计**：记录用户操作日志和系统事件日志，定期进行安全审计，及时发现和处理安全漏洞。
- **合规性遵循**：确保平台符合相关法律法规和行业标准，保障用户隐私和数据安全。

3.5 集成与工作流

通过集成适配器和包装器，实现不同服务之间的组合和协作，满足复杂业务流程的需求。具体包括：

- **服务适配器**：将不同服务的接口进行适配，确保它们能够无缝协作。
- **包装器**：对服务进行封装，提供统一的调用接口，简化服务的使用和维护。
- **BPMN与工作流引擎**：集成BPMN工具和工作流引擎，实现业务流程的可视化设计和自动化执行，提高业务处理的效率和准确性。
- **API网关与消息队列**：通过API网关统一管理外部请求，使用消息队列实现服务间的异步通信，提升系统的响应速度和解耦性。

四、项目主要功能

4.1 购物车服务

购物车服务是商城系统的基础部分，主要包括对购物车的增删改查，以及与库存和订单的联动。可以进一步拆分为：

- **购物车管理服务**：
  - **添加商品至购物车**：用户选择商品规格后，将商品加入购物车。此功能需要验证商品的基本信息，如是否上架、库存量是否足够等。对不同用户类型（登录用户/未登录用户），需要建立不同的购物车管理策略，支持跨设备或浏览器的购物车合并。
  - **移除商品**：用户可以手动将商品从购物车中删除，该功能需要提供对单个或批量商品的移除支持。
  - **修改商品数量**：用户可以在购物车中调整商品的数量，该功能需要在修改时验证库存是否充足，以及应用相关的价格调整（如批量折扣）。

    - **查看购物车**：用户可以查看购物车的所有商品及其信息（如名称、图片、单价、数量、小计等），包括商品的实时状态（如库存不足、下架等）。
- **库存检查服务**：
    - **实时库存验证**：当用户添加商品到购物车或修改商品数量时，系统应进行实时的库存检查，确保商品库存量能够满足用户需求。在用户浏览购物车页面时，定期（或基于事件驱动）更新商品的库存状态，并在库存不足时给予提示或自动调整数量。
    - **库存状态更新**：当库存发生变化（如其他用户下单、商家补货等），需要通过异步事件通知购物车服务更新商品状态。可以借助消息队列或订阅发布模式来触发库存状态的更新。
    - **库存锁定或预扣**：如果系统采用库存预扣策略，当用户将商品加入购物车后，需要对商品的库存进行一定时间的锁定或预扣，以保证库存的准确性和用户体验。
- **优惠计算服务**：
    - **促销活动应用**：根据商城的促销策略（如满减、折扣、买一送一等），动态计算购物车中商品的优惠价格。需要支持多种促销活动的叠加或优先级逻辑。
    - **优惠券应用**：允许用户在购物车中选择和应用优惠券，并在优惠券符合条件时自动计算折扣。优惠券的计算逻辑需要考虑各种优惠券类型（如满减券、折扣券、现金券等）的限制和适用条件。
    - **动态总价计算**：购物车中的商品、优惠和折扣会实时影响购物车的总价，系统需要提供一个实时计算的机制，确保用户能够随时看到最新的应付金额。
- **用户会话服务**：
    - **用户购物车同步**：对于登录用户，当用户在不同设备或浏览器中使用购物车时，系统需要支持跨设备同步。
    - **未登录用户的购物车持久化**：对于未登录用户，可以使用浏览器的`localStorage`或`sessionStorage`保存购物车数据。或者，通过匿名用户ID（如基于Cookie生成的唯一ID）将购物车数据存储在数据库中。登录后将未登录购物车数据与登录账户进行合并。
    - **购物车数据合并**：当未登录用户登录后，系统需要将之前的匿名购物车数据与登录用户的已有购物车数据进行合并。处理合并逻辑时，需要注意重复商品的数量处理策略。
- **购物车状态管理和通知服务**：
    - **库存状态变化通知**：当某个商品库存不足或补货时，购物车模块需要能够及时收到消息，更新用户的购物车状态。
    - **优惠活动的动态更新**：如果系统中的促销活动发生变化（如活动结束、优惠券失效），需要实时通知购物车模块进行刷新。

4.2 智能推荐服务

智能推荐服务是提升用户体验的重要模块，用于根据用户行为提供个性化的商品推荐。可以进一步细分为：

- **用户行为采集服务**：负责收集用户的各种行为数据，如浏览、点击、购买、搜索和收藏等，为推荐算法提供输入数据。
- **推荐算法服务**：根据用户的行为数据，利用协同过滤、内容推荐、或深度学习等算法生成推荐商品列表。
- **推荐结果缓存服务**：缓存生成的推荐结果，以减少实时计算压力，提高推荐的响应速度。
- **推荐效果评估服务**：通过A/B测试等手段对不同推荐策略的效果进行测试和评估，以优化推荐算法。

4.3 退换货服务

退换货服务涉及用户体验的关键环节，需要清晰的流程和自动化的管理。可以进一步细分为：

- **退换货申请管理服务**：
  - **退换货申请服务**：负责接收用户的退货或换货申请，并根据商城政策进行初步验证（如退货期限、商品状态等）。
  - **审核服务**：自动化处理大部分退换货申请的初步审核，同时支持复杂申请的人工审核介入。
- **退款/换货处理服务**：
  - **退款处理服务**：管理退款的具体流程，与财务系统对接，确保退款的准确性和及时性。
  - **换货处理服务**：处理换货订单的库存管理、物流安排和新商品的发货。
- **物流追踪服务**：
  - **退货物流追踪服务**：追踪退货商品的物流状态，实时更新系统中的退货状态，并通知用户。
  - **换货物流追踪服务**：追踪换货商品的发货状态，确保用户和商家能够了解商品的运输情况。
- **客服支持服务**：
  - **自动客服服务**：自动回复和解决一些常见的退换货问题，如退货政策咨询、物流信息查询等。
  - **人工客服服务**：处理复杂的退换货问题或纠纷，确保用户能够获得及时的帮助。

4.4 安全支付服务

安全支付服务涉及资金的安全流动和管理，需要保证交易的安全和稳定性。这里主要调用外部支付API，可以进一步细分为：

- **订单管理功能服务**
  - **订单生成服务**：在用户提交购物车中的商品后，系统会生成订单，并保存订单的所有信息（如商品、总价、折扣、用户信息、支付信息等）。
  - **订单状态管理服务**：在订单生成、支付中、支付成功、支付失败、取消等状态下，系统能够对订单进行状态的更新和管理。
  - **订单与支付关联服务**：为订单和支付过程建立唯一的关联，以保证资金的准确性和安全性。
- **支付网关服务**
  - **多渠道支付服务**：支持多种第三方支付平台（如支付宝、微信支付、银联等），为用户提供多样化的支付选择。
  - **支付请求处理服务**：处理与第三方支付平台的通信过程，确保支付请求的安全性和稳定性。
- **支付状态回调服务**
  - **支付结果接收服务**：第三方支付平台在支付完成后，会向商城系统发送支付状态的回调通知。系统需要处理这些回调，更新订单的支付状态。
  - **回调验证服务**：系统需要验证回调的真实性，避免虚假通知的风险。验证方式通常包括签名验证、订单号和金额的匹配等。
- **支付异常处理服务**
  - **支付失败处理服务**：如果支付未成功（如网络中断、用户取消等），系统需要提供重试和取消支付的选项。
  - **超时处理和退款服务**：如果支付超时或用户申请退款，系统需要与第三方支付平台对接，发起退款请求，并更新订单状态。

4.5 其他面向客户的服务

- **用户注册、登录、个人信息管理功能**：
  - **注册与登录**：支持用户通过邮箱、手机号等多种方式注册和登录，提供密码重置和多因素认证选项。
  - **个人信息管理**：用户可以查看和编辑个人信息，包括昵称、头像、联系方式、地址等。
- **展示商品信息供用户浏览**：
  - **商品分类展示**：按类别展示商品，用户可以根据类别快速找到感兴趣的商品。
  - **关键词搜索与筛选**：支持关键词搜索、价格区间筛选、品牌筛选等多种筛选条件，帮助用户精准查找商品。
- **评论与评分功能**：
  - **商品评论**：用户在购买商品后，可以对商品进行评论，分享使用体验。
  - **评分系统**：用户可以对商品进行评分，帮助其他用户做出购买决策。
- **取消和查看订单功能**：
  - **取消订单**：在订单未发货前，用户可以取消订单。
  - **查看订单**：用户可以查看历史订单记录，包括订单状态、商品详情、支付信息等。

4.6 面向商家的服务

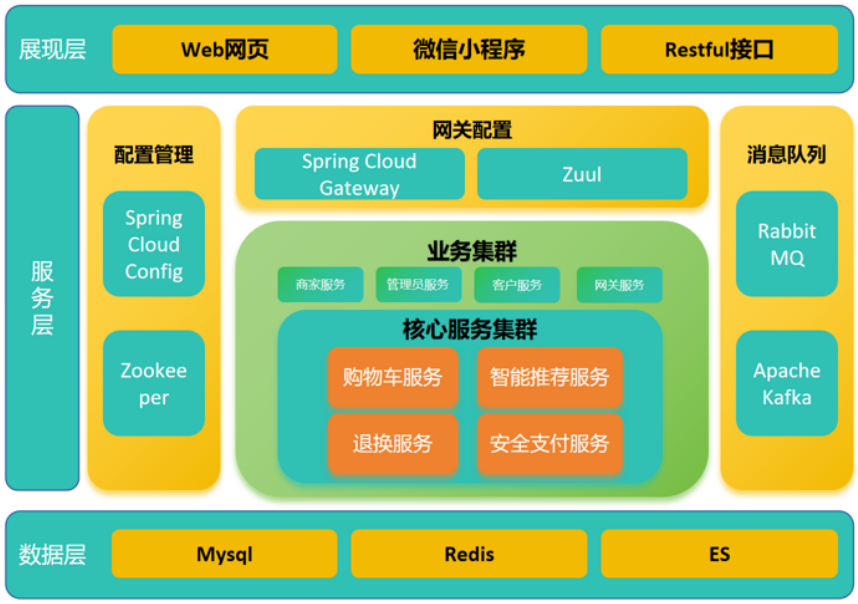
- **商家入驻与商铺信息管理功能**：
  - **商家注册与认证**：支持商家注册入驻，提供资质认证和审核流程，确保平台商品的质量和合法性。
  - **商铺信息管理**：商家可以管理自己的商铺信息，包括店铺名称、简介、联系方式、营业时间等。
- **更新商品信息**：
  - **商品上架、下架**：商家可以根据需要上架或下架商品，灵活管理商品的销售状态。
  - **修改商品数量**：实时更新商品的库存数量，避免超卖或缺货情况。
  - **编辑商品详情**：商家可以编辑商品的名称、描述、价格、图片等详细信息，保持商品信息的准确性和吸引力。
- **查看订单与处理订单发货、退货功能**：
  - **查看订单管理**：商家可以查看收到的订单，了解订单的详细信息。
  - **发货处理**：商家在确认订单后，可以进行发货操作，更新订单状态。
  - **退货处理**：处理用户的退货申请，更新库存和订单状态，确保售后服务的及时性和有效性。
- **回复用户评论功能**：
  - **评论回复**：商家可以对用户的评论进行回复，提升用户互动和满意度，改善客户服务体验。

4.7 面向管理员的服务

- **用户管理功能**：
  - **查看用户**：管理员可以查看平台所有注册用户的信息，进行管理和维护。
  - **审核商家**：对新入驻的商家进行审核，确保商家的资质和信誉，维护平台的质量标准。
- **商品管理功能**：

    - **审核新上架商品**：管理员审核商家新上架的商品，确保商品符合平台的质量和合规要求。
    - **下架违规商品**：及时发现并下架违规商品，保护用户权益和平台声誉。
- **订单监控功能**：
    - **管理订单状态**：监控所有订单的状态，确保订单流程的顺畅。
    - **处理异常订单**：针对异常订单（如支付失败、发货延迟等）进行处理，保障用户的购物体验。
    - **数据统计**：对订单数据进行统计分析，生成报表，辅助决策和优化平台运营。

五、初步逻辑架构



六、候选开发技术

6.1 后端开发框架

- **Spring Boot**：
    - **简介**：基于Spring框架的快速开发平台，简化了Spring应用的配置和部署。
    - **优势**：易上手、自动配置、丰富的生态系统，适合构建独立、生产级别的微服务应用。
    - **应用**：用于开发各个微服务，如用户服务、商品服务、订单服务等。
- **Spring Cloud**：
    - **简介**：基于Spring Boot的微服务架构解决方案，集成了各种微服务功能组件。
    - **优势**：提供服务发现、配置管理、断路器、网关等功能，支持自动装配，简化微服务开发。
    - **主要组件**：
        - **Eureka**：服务发现与注册。
        - **Nacos**：配置管理和服务发现。
        - **OpenFeign**：声明式HTTP客户端，简化服务间的通信。
        - **Spring Cloud Gateway**：API网关，处理路由、过滤、负载均衡等。
- **Dubbo**：
    - **简介**：高性能的Java RPC框架，主要用于构建分布式服务架构。

    - **优势**：支持负载均衡、服务治理、高可用性，适合大型分布式系统。
    - **应用**：适用于对性能要求高、服务调用频繁的场景。
- **MybatisPlus**：
    - **简介**：MyBatis的增强版，简化了数据库操作。
    - **优势**：提供CRUD代码生成、逻辑删除、分页查询等功能，提升开发效率。
    - **应用**：用于各微服务的数据持久层，简化数据库交互。

6.2 前端开发框架

- **Vue.js**：
    - **简介**：渐进式JavaScript框架，适用于构建用户界面。
    - **优势**：易学易用、性能出色、支持组件化开发，适合构建复杂的单页应用（SPA）。
    - **应用**：用于开发平台的前端界面，包括用户端、商家端和管理员端。
- **Element Plus**：
    - **简介**：基于Vue 3的UI组件库，提供丰富的高质量组件。
    - **优势**：设计精美、易于定制，简化前端开发过程。
    - **应用**：用于构建平台的UI界面，提升用户体验和开发效率。

6.3 数据库

- **MySQL**：
    - **简介**：开源的关系型数据库，广泛应用于Web应用。
    - **优势**：稳定性高、性能优越、支持复杂查询和事务处理。
    - **应用**：用于存储结构化数据，如用户信息、商品信息、订单信息等。
- **Redis**：
    - **简介**：开源的内存数据存储系统，支持多种数据结构。
    - **优势**：高性能、支持缓存、消息队列、分布式锁等功能，提升系统性能。
    - **应用**：用于缓存热点数据、加速数据访问、实现会话管理等。

6.4 中间件

6.4.1 消息队列

- **RabbitMQ**：
    - **简介**：高性能的异步通信组件，基于AMQP协议。
    - **优势**：支持复杂消息路由、高可靠性，适用于任务队列、消息广播等场景。
    - **应用**：用于服务间的异步通信，如订单处理、库存更新等。
- **Kafka**：
    - **简介**：高吞吐量的分布式流处理平台，设计用于处理大量数据流。
    - **优势**：高可扩展性、低延迟，适合实时数据处理和日志聚合。
    - **应用**：用于实时数据流处理、日志收集与分析等。

6.4.2 配置管理

- **Spring Cloud Config**：
  - **简介**：集中管理微服务的配置，支持版本控制和动态更新。
  - **优势**：通过Git等存储配置文件，实现配置的统一管理和版本控制。
  - **应用**：用于管理各微服务的配置文件，确保配置的一致性和可维护性。
- **Zookeeper**：
  - **简介**：开源的分布式协调服务，主要用于管理分布式系统中的配置、命名、同步和组服务。
  - **优势**：高可靠性、强一致性，适用于分布式系统的协调和管理。
  - **应用**：用于服务发现、分布式锁管理等。

6.4.3 网关配置

- **Spring Cloud Gateway**：
  - **简介**：基于Spring生态的API网关，支持动态路由、过滤、负载均衡和安全认证。
  - **优势**：高度可定制、易于集成，适合构建现代化的API网关。
  - **应用**：用于统一管理外部请求，处理路由转发、权限校验、限流等。
- **Zuul**：
  - **简介**：Netflix提供的API网关，支持动态路由和请求过滤。
  - **优势**：集成简单、功能强大，适用于服务的统一入口管理。
  - **应用**：作为备用选项，用于构建API网关，实现请求的路由和过滤。

6.5 平台

6.5.1 云平台

- **阿里云**：
  - **简介**：提供全面的云计算服务，包括计算、存储、数据库、网络安全等。
  - **优势**：稳定可靠、服务丰富、全球化布局，适合构建和部署大规模应用。
  - **应用**：用于部署微服务、数据库、缓存、消息队列等基础设施，确保平台的高可用性和可扩展性。

6.5.2 容器化

- **Docker**：
  - **简介**：开源的容器化平台，允许开发者打包应用及其依赖到一个可移植的容器中。
  - **优势**：简化应用的部署和管理，提升开发与运维效率。
  - **应用**：用于构建和部署各微服务的容器，确保环境一致性和快速部署。

6.6 监控与日志

- **ELK Stack**：
  - **Elasticsearch**：
    - **简介**：分布式搜索和分析引擎，负责存储和检索日志数据。
    - **优势**：支持快速查询和实时分析，适合大规模日志数据的处理。

        - **应用**：用于存储和索引系统日志、应用日志，支持快速检索和分析。
    - **Logstash**：
        - **简介**：数据收集和处理管道，可以从多种来源收集日志，进行过滤、转换，并将数据发送到Elasticsearch。
        - **优势**：高度可扩展、支持多种输入和输出插件，适用于复杂的数据处理需求。
        - **应用**：用于收集和处理各微服务的日志数据，统一传输到Elasticsearch中。
    - **Kibana**：
        - **简介**：数据可视化工具，用于创建动态仪表板，展示存储在Elasticsearch中的日志数据。
        - **优势**：直观的可视化界面、强大的查询和分析功能，便于监控和故障排查。
        - **应用**：用于实时监控系统性能、分析日志数据，帮助运维团队及时发现和解决问题。

6.7 版本控制

- **Git**：
    - **简介**：分布式版本控制系统，支持团队协作开发。
    - **优势**：高效的分支管理、强大的合并功能，适合大规模团队协作。
    - **应用**：用于管理项目代码，支持多人协作开发，确保代码的版本控制和历史追溯。

七、项目开发进度

7.1 项目总体规划（7周）

1. **需求分析与设计阶段**（第1周）
    1. 收集和分析项目需求
    2. 制定详细的项目计划和时间表
    3. 设计系统的总体架构和各模块的功能
2. **系统架构设计与后端开发阶段**（第2-3周）
    1. 设计微服务架构，确定服务划分和职责
    2. 实现基础服务（如用户服务、商品服务、订单服务等）
    3. 编写和调试服务间的API接口
3. **前端开发阶段**（第4-5周）
    1. 设计和开发前端界面
    2. 实现前后端集成，通过RESTful API实现功能交互
    3. 进行前端的功能调试与优化
4. **集成与测试阶段**（第6周）
    1. 完成系统集成
    2. 进行单元测试、集成测试、系统测试，确保功能完整性和稳定性
5. **部署与上线阶段**（第7周）
    1. 部署到云平台，进行性能优化
    2. 完成最终的系统测试与验收，正式上线

7.2 任务分配

- **后端开发**（2人）：专注于微服务开发，使用Spring Boot和Spring Cloud，负责用户、订单、商品等核心服务的实现。
- **前端开发**（2人）：使用Vue.js和Element Plus，开发用户端和管理端的界面，确保UI友好、响应迅速。
- **架构与集成**（1人）：负责系统架构设计和服务集成，进行API设计、消息队列、数据库管理和服务组合等工作。

7.3 时间安排

周数	主要任务	负责人
第1周	需求分析与架构设计	全团队协作
第2周	开发用户服务、商品服务、订单服务（后端基础开发）	后端开发人员（2人）
第3周	完成后端服务，初步搭建前端框架	后端开发人员 + 前端开发人员
第4周	实现购物车、支付等前端功能，集成后端服务	前端开发人员（2人）
第5周	完善前后端功能，确保服务正常运行	前端 + 后端协作
第6周	进行集成测试，完成系统优化与Bug修复	全团队协作
第7周	部署到云平台，进行系统测试、验收并上线	架构与集成负责人

八、团队成员

2153393 胡峻玮

2153495 钟承哲

2154284 杨骏昊

2154343 邹涵

2250821 郭平伟