**软件工程报告**

**小组成员及分工**

|  |  |
| --- | --- |
| **第九组** | |
| 组长：**魏唯** | 分工：**软件体验测试和前后端接口测试，发现并修复软件bug，包括发布商品后页面进行刷新、拍卖订单无法正常取消下架(前端筛选商品类别)、卖家无法完成订单、拍卖商品无人出价时卖家订单错误显示、拍卖倒计时处理，apifox接口测试** |
| 组员：**刘传哲** | 分工：**软件前后端基础构建和衔接软件基础框架、数据库构建、内网穿透、后端包括商品发布、用户登录、订单确认、前端包括所有页面的拼接、tabbar、发布页面和个人主页用户登录** |
| 组员：**尹杰** | 分工：**前端个人主页的收藏夹、个人信息修改、历史记录、查看订单部分** |
| 组员：**陈彬镔** | 分工：**前端商品页面和确认订单页面** |
| 组员：**仇誉翔** | 分工：**后端修改个人信息、收藏夹、历史浏览记录** |
| 组员：**傅嘉浠** | 分工：**后端订单处理** |
| 组员：**周泽臣** | 分工：**后端拍卖功能、商品页面信息和图片处理** |
| 组员：**廉琦** | 分工：**后端筛选商品和搜索** |
| 组员：**冷远兴** | 分工：**前端各类商品的搜索和分类** |
| 组员：**林海华** | 分工：**前端各类商品的首页展示和个人中心展示** |

**需求分析**

**一、项目概述**

1. **项目背景**

随着校园生活的日益丰富以及消费观念的不断变化，校内师生对于物品的更新换代愈发频繁，由此产生了大量闲置的二手物品。这些闲置物品中不乏仍具有较高使用价值的商品，然而目前校内缺乏一个统一、便捷且功能完善的交易平台来促进这些物品的有效流转。一方面，传统的线下二手交易方式存在诸多不便之处。例如，信息传播范围有限，往往只能通过校内论坛、张贴传单等方式发布交易信息，导致很多有需求的潜在买家无法及时获取消息；交易时间和地点难以协调，买卖双方需要多次沟通协商，耗费大量时间和精力；而且缺乏有效的交易保障机制，容易出现交易纠纷，双方权益难以得到充分维护。另一方面，随着移动互联网的飞速发展，小程序凭借其无需安装、即用即走的便捷特性，深受广大用户喜爱，成为了各类应用服务的热门载体。在校内搭建一个基于微信小程序的二手交易平台，不仅能够充分利用师生们日常高频使用微信的习惯，方便大家随时随地进行交易操作，还能借助小程序的功能优势，有效整合校内二手资源，规范交易流程，提高交易效率，营造一个安全、可靠、活跃的校园二手交易环境。此外，考虑到校园内不同的交易需求，除了常规的二手商品买卖功能外，增加拍卖和租赁功能也极具现实意义。拍卖功能可以满足一些具有特殊价值或稀缺性物品的交易需求，通过竞拍的方式实现物品价值的最大化；租赁功能则适用于那些使用频率不高但偶尔又有需求的物品，如各类活动用品、短期学习工具等，为有需要的师生提供了一种灵活、经济的使用方式。综上所述，开发一个具备二手、拍卖、租赁功能的校内二手交易平台小程序，对于优化校园资源配置、满足师生多样化交易需求以及提升校园生活便捷性都有着重要的推动作用。

1. **目标用户**

主要面向校内全体师生，包括学生、教职工等，满足他们售卖闲置二手物品、参与物品拍卖以及租赁所需物品的需求。

**二、功能需求**

**（一）用户注册与登录**

1. **注册功能**
   * 支持使用微信登录方式注册账号。
   * 可选择填写额外的个人信息，如姓名、头像等，用于完善个人资料，增强平台交互性和可信度，但非必填项。
2. **登录功能**
   * 提供登录方式，如微信授权登录等，方便用户快速进入平台。
   * 记住登录状态功能，下次打开小程序自动登录，提升用户体验。

**（二）二手商品交易功能**

1. **商品发布**
   * 用户能够上传二手商品的基本信息，包括商品名称、详细描述、价格、商品图片。
   * 选择商品分类，便于买家按照分类筛选查找商品。
   * 填写卖家的联系电话、所在校区及具体地等信息。
2. **商品浏览与搜索**
   * 在首页或专门的二手商品展示页面，以列表形式展示已发布的二手商品，展示关键信息如商品图片、名称、价格、发布者昵称等，点击可进入商品详情页查看更详细内容。
   * 提供强大的搜索功能，用户可通过关键词、分类筛选等条件进行精准搜索，快速找到心仪的商品。
3. **商品详情页**
   * 完整展示商品的所有发布信息，包括详细描述、图片、卖家信息等。
   * “收藏” 功能，用户可将感兴趣的商品添加到收藏夹，方便后续快速查看和对比。
4. **交易流程**
   * 买家与卖家沟通达成交易意向后，可选择线下交易。
   * 交易完成后，买家和卖家可对此次交易进行评价，评价内容包括对商品质量、交易过程、对方沟通态度等方面的打分和文字描述，评价结果展示在双方的交易记录及个人主页等相关位置，供其他用户参考。

**（三）拍卖功能**

1. **拍卖商品发布**
   * 除了二手商品发布所需的基本信息外，卖家需设定起拍价、拍卖结束时间等拍卖专属信息。
   * 商品图片展示、分类选择、卖家联系信息等要求与二手商品发布一致，确保信息完整性和规范性。
2. **拍卖商品展示与参与**
   * 在专门的拍卖页面展示正在进行拍卖的商品列表，展示关键信息如商品图片、名称、起拍价、当前出价、剩余时间等，吸引用户参与竞拍。
   * 用户点击进入拍卖商品详情页后，可查看详细信息，并能实时看到当前出价情况以及出价者昵称。
   * 已登录的用户若有意向参与竞拍，可在当前出价基础上按照加价幅度要求输入出价金额进行出价操作，系统实时更新当前出价及出价者信息，确保竞拍的公正性和透明度。
3. **拍卖结束处理**
   * 当拍卖结束时间到达时，系统自动判断并确定最终中标者，向中标者和卖家告知拍卖结果。

**（四）租赁功能**

1. **租赁商品发布**
   * 发布租赁商品信息时，除常规的商品名称、描述、图片等内容外，需明确租赁价格、可租赁时长范围、是否提供配送等关键租赁相关信息。
   * 同样要填写卖家的联系信息，便于承租方沟通咨询。
2. **租赁商品浏览与预订**
   * 在租赁商品展示页面，以列表形式呈现可供租赁的商品，展示主要信息如商品图片、名称、租赁价格等，方便用户查找筛选。
   * 用户点击进入详情页后，可查看详细租赁信息，若有意向租赁，可选择租赁时长，点击按钮提交租赁申请，系统提示用户填写预计取货时间、归还时间等必要信息。
   * 出租方收到租赁申请后，可查看承租方的申请信息，并决定是否同意租赁，若同意，则双方进入租赁交易流程；若不同意，承租方可根据反馈调整申请或选择其他商品。
3. **租赁交易流程**
   * 租赁申请通过后，根据双方约定完成前期的款项支付和商品交接环节。
   * 在租赁期间，系统可提醒承租方归还时间等信息，承租方按时归还商品后，出租方确认商品无损坏等情况，完成整个租赁交易。

**（五）个人中心功能**

1. **个人信息管理**
   * 用户可查看和编辑自己的信息，如更新联系方式、更换头像等，确保个人信息的准确性和及时性。
   * 展示历史交易记录，方便用户回顾过往交易情况以及让其他用户在交易前参考信用情况。
2. **我的发布**
   * 展示用户自己发布的所有商品信息列表，可对商品进行编、下架、删除等操作，方便用户根据实际情况灵活管理自己发布的商品。
3. **我的收藏**
   * 展示用户收藏的所有商品列表，方便快速查看感兴趣的商品，并可直接从收藏列表进入商品详情页，进一步了解或发起交易操作。

**（六）后台管理功能（供系统管理员使用）**

1. **用户管理**
   * 可查看所有注册用户的基本信息列表，维护平台良好的交易秩序。
2. **商品管理**
   * 审核用户新发布的商品信息，确保信息真实、合规，对于不符合要求的商品信息可驳回并提示用户修改后重新发布。
   * 对违规商品进行下架处理，并通知发布者，保障平台交易内容的合法性和安全性。
   * 统计各类商品的发布数量、交易情，了解平台不同品类商品的热度和市场需求，以便进行针对性的运营调整。
3. **交易管理**
   * 查看所有正在进行和已完成的交易记录，依据平台规则保障双方合理权益。

**三、性能需求**

1. **响应速度**  
   小程序在各种网络环境下，页面加载时间应控制在 3 秒以内，操作响应应在 1 - 2 秒内完成，确保用户操作流畅。
2. **稳定性**  
   保证小程序长时间稳定运行，避免出现闪退、卡顿等现象，确保交易等核心功能不受影响。
3. **兼容性**  
   兼容主流的微信版本以及不同尺寸的手机屏幕，确保界面展示正常、操作无异常，为用户提供一致的使用体验，无需担心因设备差异而无法正常使用平台。

**四、安全需求**

1. **用户信息安全**  
   对用户的注册信息等进行加密存储，防止信息泄露，确保用户信息在网络传输过程中的安全性，同时遵循相关法律法规，严格保护用户隐私，未经用户许可不得将个人信息提供给第三方。

**五、界面设计需求**

1. **整体风格**  
   界面设计应简洁、清新、符合校园文化氛围，色彩搭配协调，操作按钮布局合理，方便用户快速找到所需功能，同时要注重用户体验，界面交互逻辑清晰易懂，让不同年龄段、不同操作习惯的校内师生都能轻松上手使用。
2. **页面设计**
   * **首页**：展示热门商品推荐、分类导航栏、搜索框等主要元素，引导用户浏览和查找商品。
   * **商品详情页**：以图片、文字结合的方式清晰展示商品所有信息，布局上要突出关键信息，方便用户联系卖家和了解商品详情，操作按钮放置在醒目位置，便于用户操作。
   * **个人中心页**：采用模块化设计，将个人信息管理、我的发布、我的收藏等功能模块清晰划分，方便用户查看和操作相应内容，展示内容要简洁明了，避免信息过于繁杂影响用户体验。

**六、其他需求**

1. **版本更新与维护**  
   定期对小程序进行版本更新，修复已知的漏洞、优化性能、增加新功能等，在更新前提前通知用户，确保用户知晓并能及时更新使用最新版本，同时建立完善的维护机制，保障平台在日常运行中出现问题时能够快速响应并解决。

**项目特色**

**1.功能综合性**

集二手商品交易、拍卖以及租赁三种功能于一体，全面覆盖了校内师生多样化的物品流转需求。无论是闲置的日常用品、学习资料等常规二手交易，还是具有特殊价值物品的拍卖，亦或是临时性使用物品的租赁，都能在同一个平台上便捷完成，为用户提供了一站式的交易服务体验，无需在多个不同平台间切换寻找合适的交易方式。

三种功能模块相互协同又各自独立，有着统一的操作流程规范和界面风格，便于用户快速上手并理解各功能的使用方法，在保障使用便利性的同时，最大程度地发挥不同交易方式的优势，提高校内物品资源的整体利用率。

**2.校园针对性**

精准定位于校内师生群体，在功能设计和运营模式上充分考虑校园环境及用户特点。例如，在商品发布时可选择填写所在校区及具体地址，方便买卖双方根据距离选择线下交易地点，尤其适合校园内近距离的面对面交付。

界面风格贴合校园文化氛围，营造出亲切、清新的视觉感受，操作流程简洁明了，符合师生日常使用习惯，降低了用户的学习成本，让不同专业、不同年龄段的校内人员都能轻松参与到交易活动中来，打造专属校园的二手交易生态。

**3.便捷性与即时性**

基于微信小程序开发，借助微信庞大的用户基础和便捷的使用方式，用户无需额外下载安装应用程序，通过微信扫一扫或搜索即可快速进入平台，真正实现了即用即走，随时随地都能进行商品浏览、发布、交易等操作。

无论是商品信息的发布还是交易沟通环节，都设计得尽可能简洁高效。例如，提供一键发布功能简化商品上架流程，大大缩短了交易周期，提高了物品流转速度。

**系统安全**

**用户认证与授权安全**

**1.多因素身份认证**

结合微信授权登录，方便用户快速进入平台。对于关键操作（如发布商品、进行交易、修改重要个人信息等），可设置二次验证机制，确保是用户本人操作，防止账号被盗用引发的安全风险。

**2.权限管理**

针对不同角色（普通用户、管理员）设置清晰明确的权限体系。普通用户可以进行商品发布、浏览、交易等常规操作，但对于涉及其他用户隐私信息（如查看他人完整联系方式需对方同意等）、平台管理功能等则无权限访问；管理员拥有对用户、商品、交易等全方位的管理权限，能够进行审核、调解纠纷、处理违规等操作，但同样需遵循严格的操作流程和权限限制，防止权限滥用。

**数据传输安全**

**1.加密协议应用**

在用户与平台之间的数据交互过程中，全面采用安全的加密协议，如 HTTPS（Hypertext Transfer Protocol Secure），对传输的数据进行加密处理，确保用户的登录信息、商品信息、交易数据等在网络传输过程中不会被窃取或篡改。无论是在校园 Wi-Fi 环境还是移动网络环境下，保障数据传输的保密性、完整性和真实性，让用户放心使用平台进行各类操作。

**2.数据完整性校验**

对于重要的数据交互，如交易订单信息、商品关键属性修改等，在接收端进行数据完整性校验，通过校验和、数字签名等技术手段，验证接收到的数据是否与发送端一致，防止数据在传输过程中出现丢失、篡改等情况，确保平台各业务流程基于准确、完整的数据进行运转，避免因数据问题引发的交易纠纷或系统故障。

**数据存储安全**

**1.数据加密存储**

对用户的敏感信息进行加密存储，确保即使数据库发生数据泄露事件，攻击者也无法直接获取到明文信息，最大程度保护用户隐私。同时，对于商品信息、交易记录等重要业务数据，也进行适当的加密处理或混淆存储，增加数据的安全性。

**2.数据库安全管理**

选用安全可靠的数据库管理系统，并定期进行安全漏洞扫描和修复，及时更新数据库的安全补丁，防止因数据库软件本身的漏洞被黑客攻击。同时，配置严格的数据库访问权限，限制只有授权的服务器和管理员账号能够访问数据库，并且对数据库的操作（如查询、插入、更新、删除等）进行详细的日志记录，便于事后审计和追踪异常操作。

实施数据备份策略，定期对整个数据库进行全量备份，并根据业务需求设置增量备份的时间间隔，将备份数据存储在异地的安全存储介质上（如云端存储或离线的物理存储设备），以防止因本地硬件故障、自然灾害、人为误操作等原因导致的数据丢失，确保在出现问题时能够快速恢复数据，保障平台业务的连续性。

**平台运行安全**

**1.安全漏洞管理**

定期开展系统安全自查工作，包括对小程序前端代码、后端服务器以及相关接口的安全扫描，及时发现可能存在的安全漏洞（如 SQL 注入漏洞、跨站脚本攻击漏洞等）。

**2.服务器安全防护**

选用高安全性、高可靠性的服务器来部署平台，配置防火墙、入侵检测系统（IDS）、入侵防御系统（IPS）等网络安全设备，对服务器的网络访问进行严格管控，只允许合法的端口和协议通过，拦截非法的网络请求，防止外部黑客的入侵攻击。同时，对服务器进行定期的系统更新和安全加固，关闭不必要的服务和端口，降低服务器的安全风险，保障平台稳定运行。

**3.应急响应机制**

制定完善的应急响应预案，明确在遇到突发安全事件时的应对流程和责任分工。设立应急响应团队，确保在事件发生后能够快速响应，采取有效的措施进行止损、恢复和调查，最大程度降低安全事件对平台和用户造成的损失。

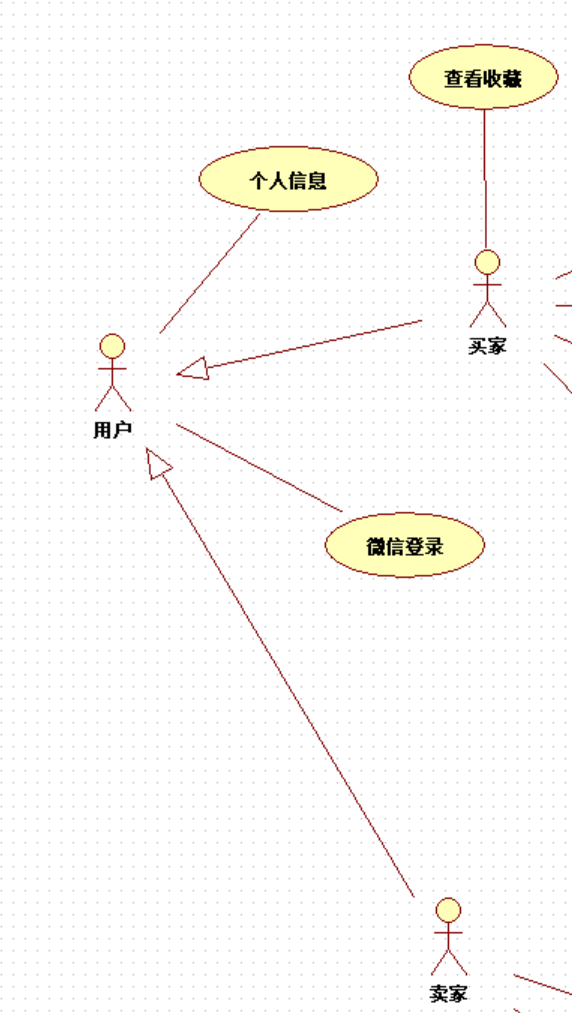
**系统分析**

**1.UML图构建**

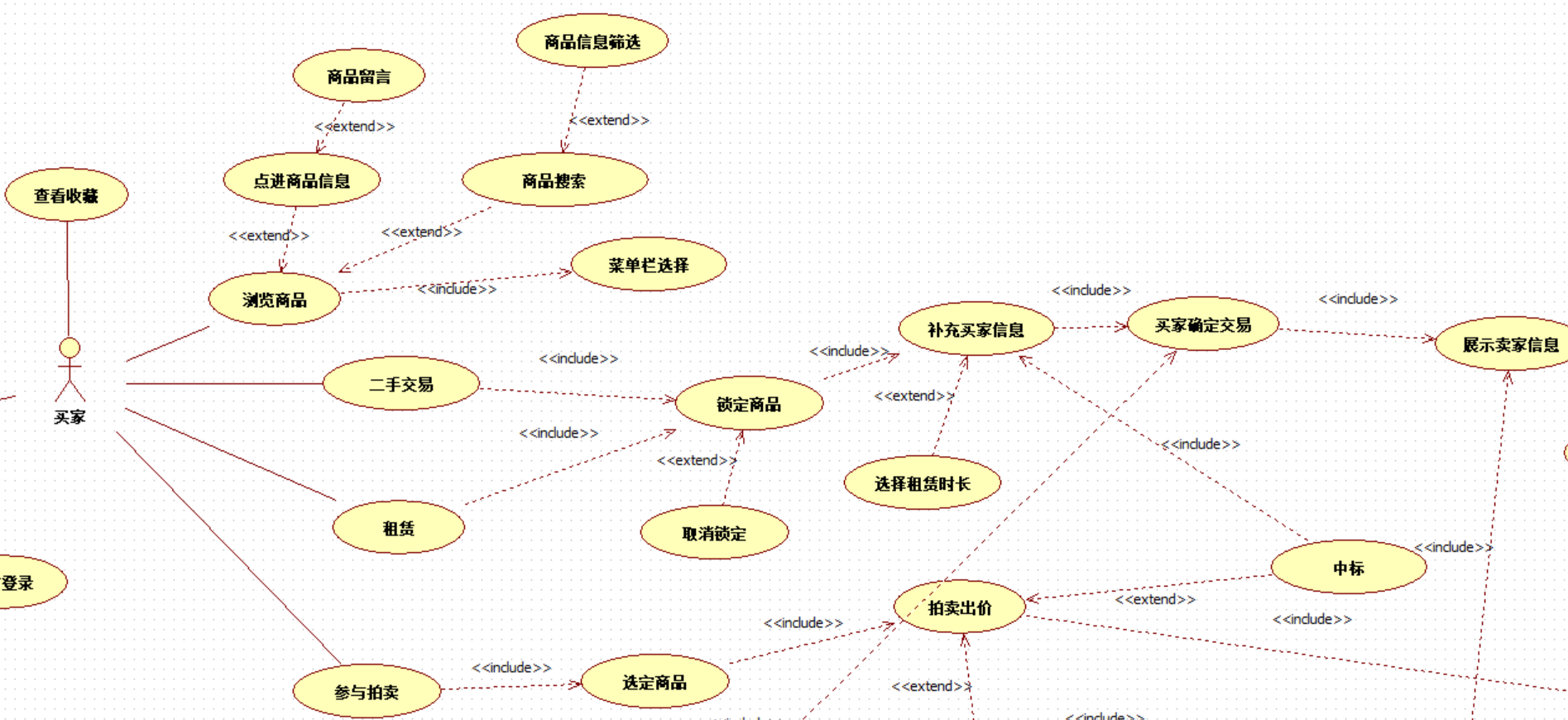
**（1）用例图**

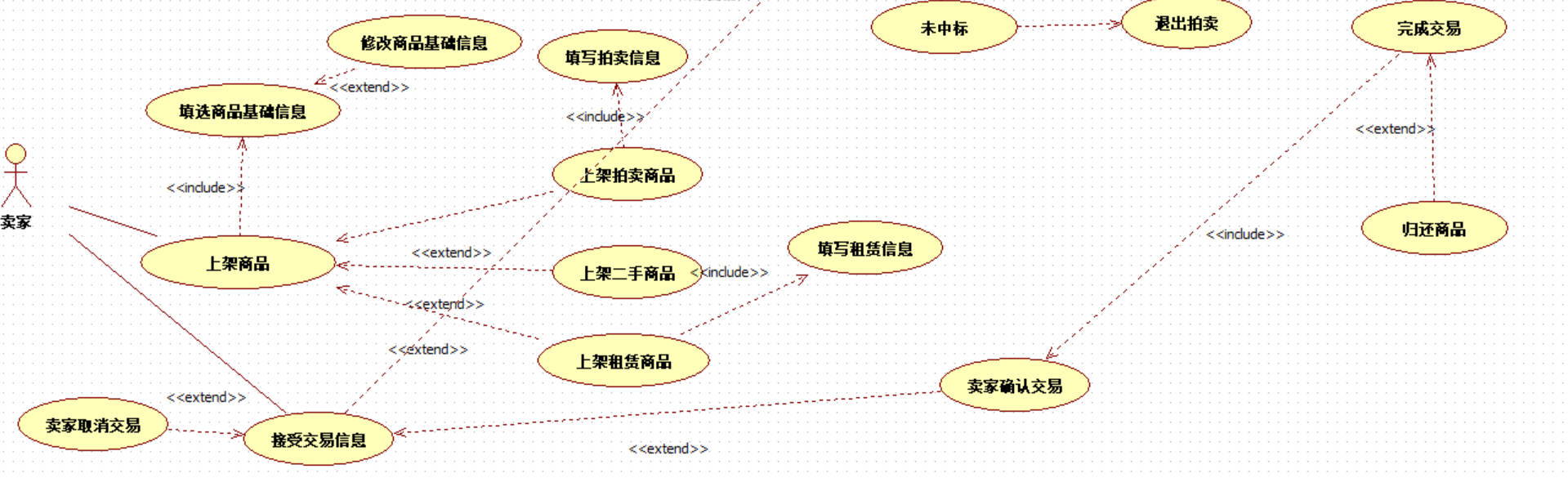
我们的项目包含**系统管理员、用户、卖家和卖家**等子系统，因此下面展示它们的用例图。

**用户用例图：**

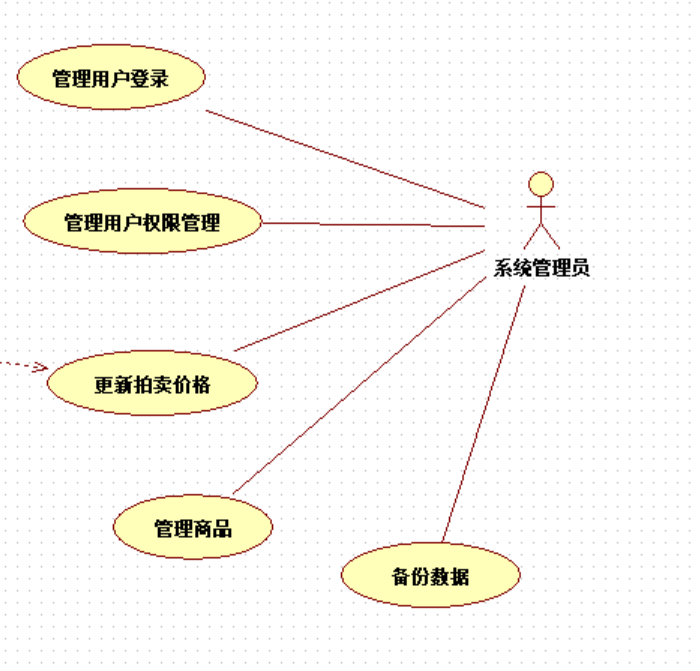
s

**买家用例图：**

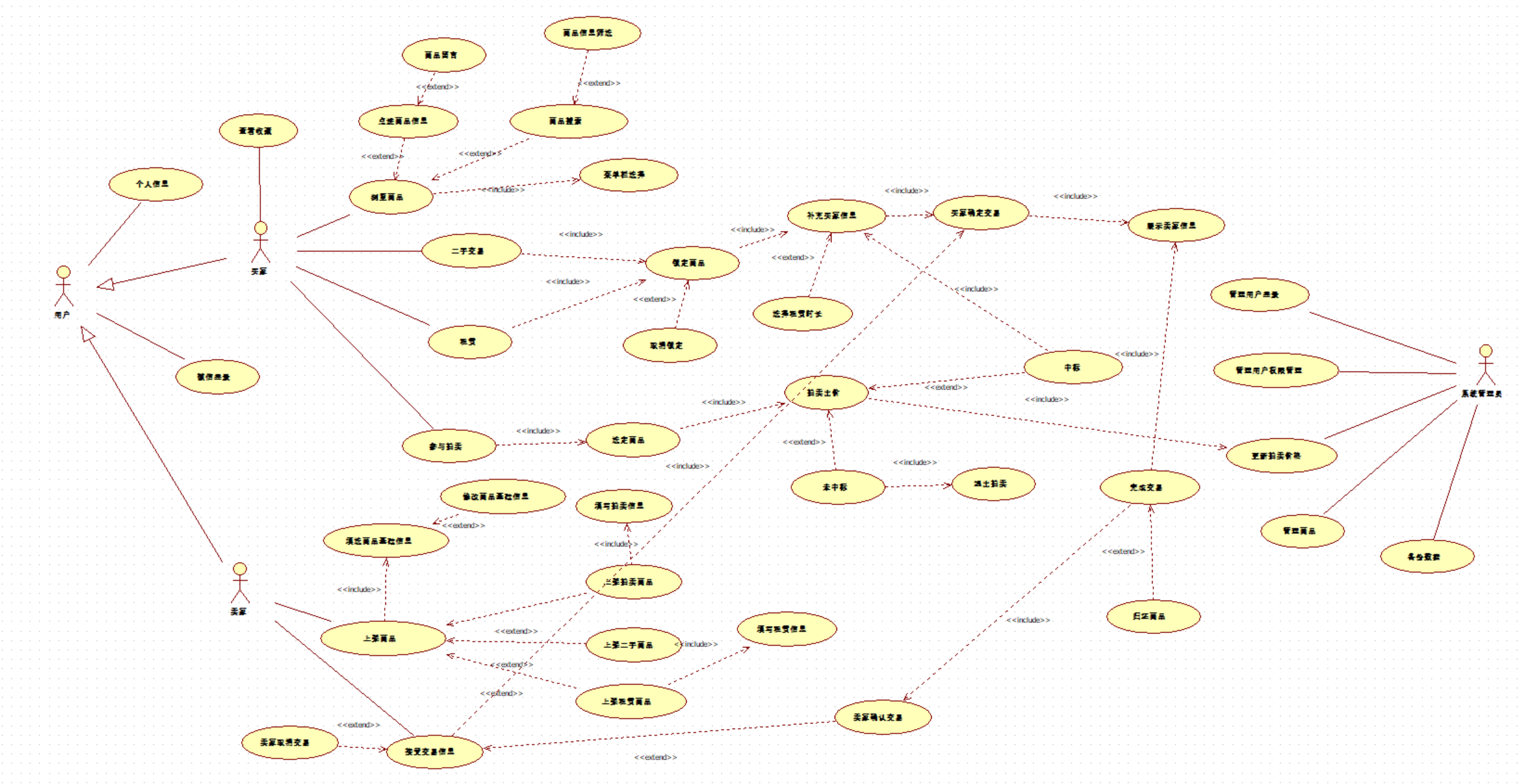


**卖家用例图：**  


**系统管理员用例图：**



**系统总体用例图：**



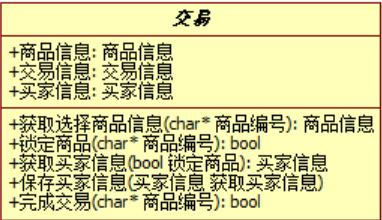
**（2）类图**

我们项目一共包含13个类图，分别是**商品信息、交易、买家信息、交易信息、拍卖、租赁、商品、拍卖类物品、租赁类物品、用户、卖家、买家和个人信息主页**。

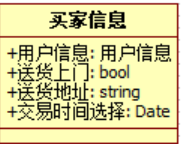
**商品信息：**



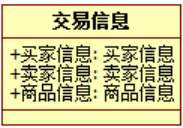
**交易：**



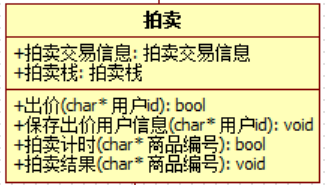
**买家信息：**



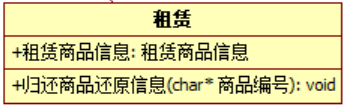
**交易信息：**



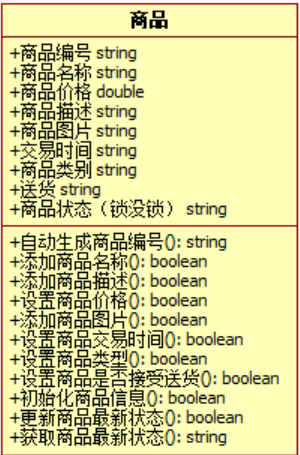
**拍卖：**



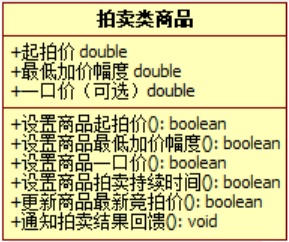
**租赁：**



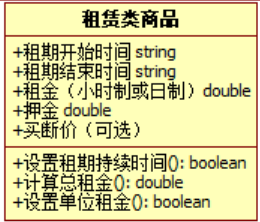
**商品：**



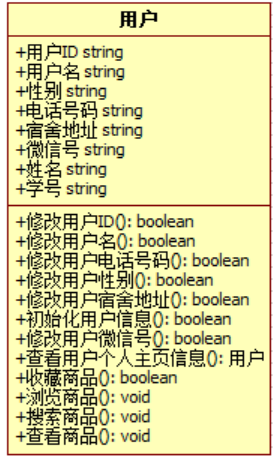
**拍卖类商品：**



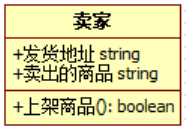
**租赁类商品：**



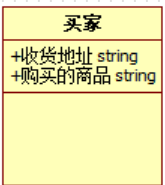
**用户：**



**卖家：**



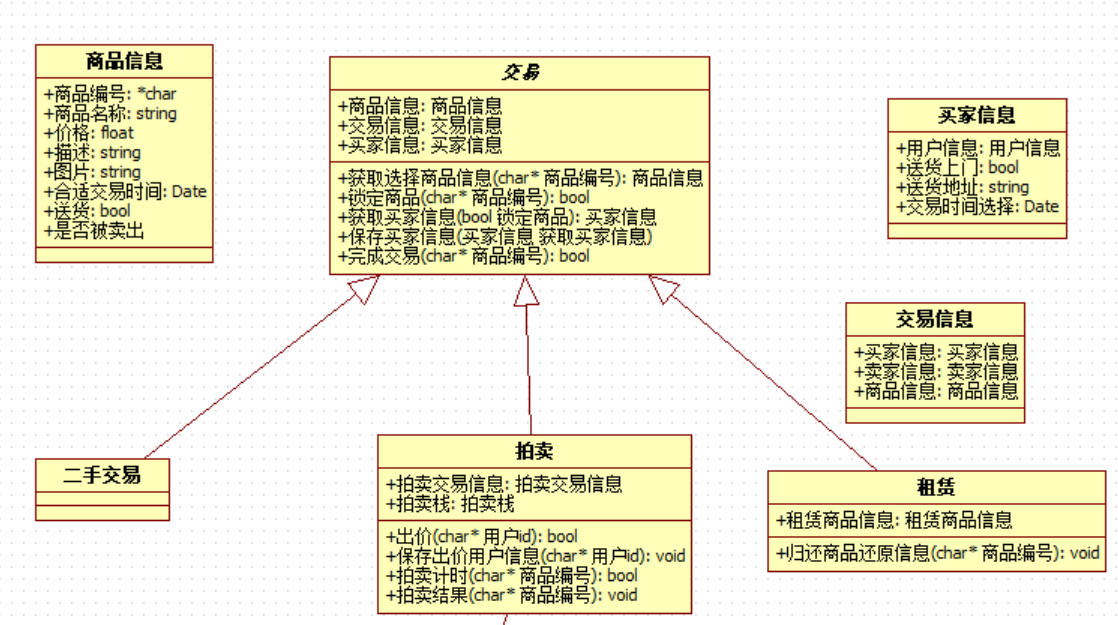
**买家：**



**个人信息主页：**



**项目中各类的关系如下所示：**



**2.系统架构模型**

**（1）表现层**

**作用**：作为用户与系统交互的接口，负责展示信息给用户以及接收用户的操作请求，呈现出友好、易用的界面。

**包含内容**：

**用户界面（UI）组件**：涵盖了各种页面，如商品列表页、商品详情页、交易订单页、个人信息主页等。例如在商品列表页，会展示众多商品的关键信息（名称、图片、价格等），方便用户浏览查找感兴趣的商品；个人信息主页则呈现用户的基本资料、购买历史、收藏列表等个性化内容。

**交互逻辑处理**：处理用户在界面上的各种操作，像点击查看商品详情、提交订单、添加收藏、修改个人信息等行为对应的事件响应。例如，用户点击 “购买” 按钮后，会触发相应的交互逻辑将购买请求传递给下一层进行处理。

**（2）应用层**

**作用**：协调业务逻辑，接收来自表现层的请求，调用领域层的业务逻辑进行处理，并将处理结果返回给表现层，起到承上启下的桥梁作用。

**包含内容**：

**商品服务**：负责处理与商品相关的各种业务操作，比如根据用户的筛选条件查询商品信息（调用商品类相关方法获取数据）、发布新商品（涉及商品信息类实例的创建和初始化，以及关联到对应的卖家）、更新商品详情等。

**交易服务**：管理整个交易流程，包括创建交易（调用交易类启动交易流程）、推进交易状态（依据不同交易类型如购买、拍卖、租赁分别调用对应的处理逻辑来更新状态）、完成交易（确保所有交易环节完成后更新相关商品和用户记录）等功能，协调买家、卖家以及商品之间的交互。

**买家服务**：围绕买家相关业务展开，像管理买家信息（调用买家信息类方法实现添加购买记录、更新联系信息等操作）、处理买家的收藏和浏览历史行为（添加或删除收藏、记录浏览记录等），为买家在系统中的各种活动提供支持。

**拍卖服务**：专门针对拍卖业务逻辑，启动拍卖活动（通过拍卖类启动方法）、处理竞拍操作（验证并记录出价等）、结束拍卖并确定结果（调用拍卖类相应方法处理后续流程）等，确保拍卖流程的顺利进行。

**租赁服务**：处理租赁相关业务，从创建租赁协议（调用租赁类方法开启租赁流程）、计算租赁费用（根据规则算出准确费用并更新记录）到管理租赁状态（检查是否到期、处理续租或归还商品等情况）等一系列操作。

**事务管理**：负责协调多个业务操作的原子性，确保在复杂业务场景下（如一个交易涉及多个步骤，包括支付、更新商品状态、记录交易记录等）要么所有操作都成功执行，要么全部回滚，保证数据的一致性。例如，在创建订单并支付的过程中，如果支付失败，要确保订单相关的其他数据修改也不会生效。

**（3）领域层**

**作用**：是整个系统的核心，包含了业务领域中的各种实体、值对象以及它们之间的业务逻辑和规则，体现了系统对业务的核心理解和处理能力。

**包含内容**：

**商品类**：作为基础的商品实体，定义了商品的通用属性和基本方，是其他具体商品类型的父类，体现了商品在系统中的核心概念和通用操作。

**买家类**：对应买家角色相关信息，包含与买家身份标识、联系信息、购买历史、收藏列表等相关的属性以及对应的操作方法，与系统中买家相关的业务逻辑紧密相关，例如管理自身信息、参与交易等行为都是围绕买家这个实体展开。

**卖家类**：类似买家类，聚焦于卖家的相关属和操作，在商品发布、交易过程中作为另一方主体参与，比如发布商品时关联到卖家信息，交易成功后资金流向卖家等场景涉及卖家实体的处理。

**交易类**：定义了交易的通用属和处理交易从发起、推进到完成整个流程的方法，是不同类型交易的抽象，体现了交易这个核心业务概念在系统中的基本逻辑。

**拍卖类**：针对拍卖这种特殊交易形式，有专属的属性和方，用于处理拍卖业务特有的操作和规则，比如竞拍过程中的出价验证、结果判定等。

**租赁类**：围绕租赁业务构建，具备租赁相关属性和操作方，管理租赁业务全流程的各项事务。

**值对象**：

**商品信息**：用于封装商品的描述性信息，如名称、价格、描述、图片、类别、状态等属性，它是对商品具体特征的一种聚合呈现，在系统中多处用于展示、传递和比较商品的相关详情，与商品实体紧密关联但更侧重于信息的承载和传递。

**交易信息**：聚焦于每一笔具体交易订单的详细情况，包含买家、卖家、商品、交易价格、交易状态、支付方式、配送方式等属性，是交易在订单层面的详细数据体现，方便在系统各环节对具体订单进行管理、查询和展示。

**领域服务**：

**商品验证服务**：负责验证商品相关信息的合法性，例如检查商品价格是否在合理范围、商品描述是否符合格式要求等，在商品发布、更新等操作时发挥作用，确保进入系统的商品数据是符合业务规则的。

**交易验证服务**：针对交易流程中的关键环节进行验证，比如验证买家是否有足够资金进行购买（、拍卖出价是否符合规则、租赁周期设置是否合理等，保障交易的合规性和有效性。

**用户验证服务**：对用户相关操作进行验证，比如买家注册时验证输入的手机号是否合法、邮箱格式是否正确，以及在登录时验证账号密码是否匹配等，维护用户信息的准确性和系统的安全性。

**（4）基础设施层**

**作用**：为上层提供技术支撑，负责与外部系统的交互以及数据的持久化存储等基础功能，是系统运行的底层保障。

**包含内容**：

**数据库访问层**：

**数据库操作接口**：定义了针对不同实体类的数据访问方法，例如商品 DAO 提供了对商品的增删改查操作接口，交易 DAO 负责处理与交易相关的数据操作等，实现了领域层实体与数据库表之间的映射和交互，隐藏了具体数据库的实现细节，方便上层调用。

**数据库驱动及连接管理**：负责加载相应的数据库驱动，建立和管理与数据库的连接，确保系统能够稳定、高效地与数据库进行通信，同时处理连接的创建、关闭、复用等操作，优化数据库资源的利用。

**3.设计模式**

**（1）单例模式**

**应用场景**：

在基础设施层中，针对数据库连接管理，确保整个项目里只有一个数据库连接实例，无论是商品信息的持久化存储、交易记录的写入，还是用户相关数据的更新等操作，都复用这同一个数据库连接，避免频繁创建和销毁连接带来的资源损耗。

**优势**：

节省数据库连接资源和系统内存资源，保证在不同业务模块使用缓存或数据库时数据的一致性，让整个项目的数据交互更稳定、高效。

**（2）工厂模式**

**应用场景**：

在领域层创建商品对象时，依据商品类型（普通商品、拍卖类商品、租赁类商品等），利用 “商品工厂” 类按照对应规则来创建不同的商品实例。例如，当业务逻辑中需要发布一款新的拍卖商品时，“商品工厂” 根据传入的拍卖商品相关参数创建出带有起拍价、拍卖结束时间等专属属性的拍卖类商品实例。

在应用层处理交易业务时，通过 “交易工厂” 类根据交易类型（购买、拍卖、租赁）分别创建对应的交易实例，并初始化如购买交易的订单详情、拍卖交易的起拍价与竞拍者列表、租赁交易的租赁周期等特定属性，清晰且规范地开启不同类型的交易流程。

**优势**：

将复杂的商品和交易对象创建过程从具体业务逻辑中分离出来，使得业务代码专注于使用对象进行后续操作，降低代码耦合度。后续若新增特殊商品类型或交易类型，只需在相应工厂类里添加创建逻辑，便于扩展项目功能。

**（3）观察者模式**

**应用场景**：

在领域层中，当商品的状态发生改变（比如商品从 “在售” 变为 “已售出”，或者拍卖商品的拍卖结束且确定中标者后状态更新等情况），商品类作为被观察的主体，会通知已注册的观察者，像关联的卖家服务模块（用于提示卖家发货、处理交易款项等后续事宜）以及买家服务模块（告知买家交易结果等），确保各相关方及时知晓商品状态变化并做出相应处理。

对于交易流程，每一次交易状态更新（从 “已下单” 到 “已支付”，再到 “已发货” 等不同阶段），交易类作为主题通知与之关联的观察者，例如支付服务模块根据 “已支付” 状态进行支付结果确认与记录，物流服务模块在 “已发货” 状态时安排实际发货操作等，保证整个交易流程各环节有序衔接。

**优势**：

实现业务模块之间的松耦合，各模块只关心自己感兴趣的状态变化事件，方便独立开发、测试与维护。同时，当项目新增一些需要关注商品或交易状态变化的功能模块时，只需简单注册为观察者即可轻松融入现有业务逻辑，扩展性强。

**（4）外观模式**

**应用场景**：

在应用层处理复杂交易流程时，“交易外观” 类将商品验证、交易创建、支付处理、物流安排等多个分散步骤整合起来，对外呈现一个简单清晰的接口用于完成整个交易过程，让表现层等其他模块调用交易功能时无需了解底层复杂的业务逻辑和模块交互情况。

**优势**：

对外简化了接口调用，隐藏了底层复杂的实现细节，无论是与外部系统交互还是内部复杂业务流程的整合，都大大降低了项目整体的复杂度，提高了代码的可读性和可维护性，便于后续对支付、物流接口以及交易流程等方面进行修改和功能扩展。

**（5）装饰器模式**

**应用场景**：

在应用层涉及用户操作时，根据不同业务场景下的权限需求，像部分高级功能仅限特定等级用户使用或者重要操作需要二次验证等情况，利用不同的权限装饰器类（如 “管理员权限装饰器”“二次验证装饰器” 等）对用户服务进行功能增强，在执行相关操作时动态添加额外的验证逻辑，保障系统的安全性和权限管理的灵活性。

**优势**：

可以灵活地在运行时为商品或用户服务添加新功能，无需修改它们原有的类结构，符合开闭原则。这样方便根据业务变化随时调整促销策略、权限验证等功能，且各个装饰器类职责明确，使得代码结构更清晰，易于理解、维护和测试。

**数据库设计**

数据库设计旨在创建一个能够高效存储、组织和管理数据的模型，以满足特定应用系统在数据处理、查询、更新以及数据完整性、一致性等方面的需求。合理的数据库设计有助于提升系统的性能、可扩展性、数据安全性，同时方便用户对数据进行操作，为整个应用系统的稳定运行和功能实现提供坚实的数据基础。

**1.用户需求与业务规则**

在这个项目当中主要包含三种角色，分别为**普通用户、商品卖家和系统管理员**。他们对应的需求如下：

**（1）普通用户角色**

* **注册与登录需求**：
  + 希望能便捷地注册成为系统用户，可选择通过微信授权登录来创建账号，但期望关键标识信息具有唯一性，避免重复注册问题。
  + 登录后系统能准确识别身份，快速进入个人操作界面，无缝衔接后续的各种功能操作，比如浏览商品、管理个人收藏等。
* **商品浏览与筛选需求**：
  + 能够在系统界面清晰、直观地浏览众多商品信息，包括查看商品名称、价格、详细描述、展示图片等关键内容，以便全面了解商品详情。
  + 期望具备灵活的商品筛选功能，可依据商品类、商品状态、是否提供上门服务以及租赁周期等多种条件进行筛选，快速定位到符合自身需求和兴趣的商品。
* **商品交易需求**：
  + 当确定要购买或租赁某商品时，能顺利发起交易流程，准确填写相关交易信息，清晰知晓订单的成交价格、交易状态的实时更新情况。
  + 在交易过程中，希望得到系统对交易流程的合理引导，确保每一步操作符合规范，比如下单、支付、确认收货等环节都能顺利进行，并且在出现问题时能方便地操作订单状态变更，同时保障自身权益。
* **商品收藏需求**：
  + 希望可以方便地将感兴趣但暂时不打算交易的商品添加到收藏列表中，系统能够快速响应收藏操作，收藏后在个人收藏页面能清晰展示所收藏商品的关键信，方便后续对比查看。
  + 要求系统能保证同一商品不会被重复收藏，维持收藏列表的准确性和整洁性，便于管理自己感兴趣的商品集合。
* **浏览历史查看需求**：
  + 期望系统自动记录自己浏览过的商品情况，形成浏览历史记录，并且能按照浏览时间顺序清晰展示浏览过的商品详情，方便回顾之前查看过的商品信息。

**（2）商品卖家角色（同时也是普通用户，但具备商品发布与管理相关额外需求）**

* **商品发布需求**：
  + 能够方便地发布商品信息到系统中，在发布时准确填写各项必填内容，如商品类别、名称、价格、描述、联系电话、卖家地址等，同时可以根据商品实际情况选择填写是否提供上门服务、租赁周期等可选信息，确保商品信息完整且能准确传达给潜在买家。
  + 希望商品发布后能快速上架成功，进入可被其他用户浏览和交易的状态，并能及时获取商品发布的反馈信息。
* **商品管理需求**：
  + 有权对自己已经发布的商品信息进行更新修改，例如调整价格、修改商品描述等，以便跟踪商品信息变化轨迹，同时期望修改操作能及时生效，在其他用户浏览商品时看到最新的信息。
  + 根据业务情况，对商品状态进行合理管理，比如在特定条件下能将商品设置为 “锁定” 状态，限制其他操作，待相应业务环节结束（如拍卖结束）后，再根据结果将商品状态变更为合适的状态（如 “交易成功”“交易取消” 等），保障商品交易流程的有序性。

**（3）系统管理员角色（通常负责系统整体的运维与管理等工作）**

* **用户管理需求**：
  + 能够查看系统内所有用户的注册信息，对用户账号进行管理，例如在出现违规行为或异常情况时，有权冻结、解封用户账号，确保系统用户环境的健康和合规。
  + 可以对用户的关键信息修改记录进行审核（如果有相应审核机制设定），保证用户信息变更符合系统要求和相关政策法规，维护用户数据的安全性和准确性。
* **商品管理需求**：
  + 对系统内所有商品信息进行整体把控，能查看商品的发布、交易、状态变更等情况，对于不符合规定（如违规商品描述、虚假价格等）的商品，有权进行下架处理，保障商品信息的真实性和合法性，维护良好的交易环境。
  + 可以统计分析商品相关的数据，如各类商品的发布数量、不同状态商品的占比等，以便根据数据反馈来优化系统功能或调整业务策略，促进系统健康运营。
* **订单管理需求**：
  + 能够查看系统内所有订单的详细情况，包括订单的生成、交易状态变化、成交价等关键信息，在出现交易纠纷等特殊情况时，介入协调处理，保障交易双方的合法权益，维护正常的交易秩序。
  + 可以基于订单数据进行一些深度分析，比如统计不同时间段的订单量、各类商品的交易活跃度等，为系统运营和业务拓展提供数据支持。
* **系统维护与监控需求**：
  + 定期对数据库进行备份操作，防止数据丢失，确保在遇到硬件故障、软件错误等意外情况时，系统数据能够快速恢复，保障系统的可用性和数据安全性。
  + 监控数据库的性能指标（如查询响应时间、存储空间使用情况等），根据性能数据及时调整数据库的配置（如优化索引、清理冗余数据等），保证系统在高并发访问、大数据量存储等情况下依然能够稳定高效运行，为用户提供良好的使用体验。

根据上面三种角色的需求，我们给出项目的**业务规则**如下所示：

**（1）用户相关业务规则**

**注册规则**：保证注册用户在系统内唯一，且要验证输入信息格式。

**登录规则**：通过微信验证登录，关联用户相关数据。

**信息修改规则**：登录后可改个人信息，修改后要在各业务场景体现新信息。

**（2）商品相关业务规则**

**发布规则**：必填商品类别、名称、价格等基本信息，选填服务、租赁周期等，发布后初始状态为 “上架中”。

**更新规则**：可更新已发布商品信息，更新时自动记录更新时间，特殊状态下可能限制更新。

**状态流转规则**：商品状态依业务场景变化，如拍卖时锁定，结束后按结果变更状态，需遵循设定条件。

**（3）订单相关业务规则**

**生成规则**：买家满足条件后生成订单，关联商品，记录买家、成交价等，初始状态为 “交易中”。

**状态变更规则**：按交易流程推进变更订单状态，如收货后变为 “已收货” 等，要依流程操作并记录时间节点。

**租赁订单规则**：明确租期，依租赁周期和时长算费用，租期结束涉及商品归还等操作。

**（4）收藏相关业务规则**

**收藏操作规则**：用户可收藏商品，系统记录关联信息与时间，同一用户不能重复收藏同一商品。

**取消收藏规则**：用户可取消收藏，系统实时更新收藏列表。

**（5）浏览历史相关业务规则**

**记录规则**：用户浏览商品时系统自动记录关联信息，用于分析浏览行为等。

**数据清理规则**：定期清理过期浏览历史数据，如保留近几个月数据，保障空间合理利用。

**2.数据库表格设计**

数据库表格设计是数据库设计过程中的一个关键环节，它是指根据具体的业务需求和数据处理要求，对数据库中将要存储的数据进行合理的组织、规划与结构化描述的过程。具体来说，就是确定数据库中各个表的名称、表内包含的字段（列）及其相应的数据类型、长度、约束条件等属性，同时还要明确各表之间通过外键等方式建立起来的关联关系，以此构建出一个能够有效存储、管理和操作数据的逻辑模型，方便后续数据库的实施以及应用系统基于这些数据开展各类业务功能。

下面将详细展示本项目中所包含的所有表格设计属性。

（1）**User（用户）表**主要用于存储系统中所有用户的基本信息，是整个系统用户管理的核心基础。它通过user\_id这个唯一主键来精准区分不同用户，其他字段如 openid、wechat\_id、email等从不同角度进一步完善用户的身份标识以及提供联系沟通等相关信息，方便在后续的各种业务操作中（比如商品交易时关联买家或卖家、发送通知等）准确识别和定位用户，同时也支持用户个性化信息（如头像、地址等）的存储与管理，为构建用户相关的各类功能和服务提供数据支撑。

|  |  |
| --- | --- |
| **字段名** | **说明** |
| user\_id | 用户 ID，自动生成 |
| openid | 微信用户的 openid，唯一 |
| wechat\_id | 微信号，唯一且可选 |
| username | 用户名 |
| email | 邮箱，可选 |
| phone | 联系电话，可选 |
| address | 地址，可选 |
| profile\_picture | 头像，可选 |

（2）**Product（商品）表**的核心功能在于存储系统内全部商品的详细信息，是整个商品交易业务的数据核心所在。它以product\_id作为唯一标识，涵盖了商品从基本分类（通过category1和category2体现）、关键属性（如名称、价格、描述等）到交易相关特性（如是否提供上门服务、租赁周期等）以及状态变化（商品状态、拍卖结束时间等）和创建更新时间记录等多方面内容，并且通过user\_id外键关联到User表确定商品所属用户，以此为基础来支持商品的展示、交易、状态管理等一系列操作，为买家浏览选购商品以及卖家发布管理商品提供了全面的数据依据，保障商品交易流程能够顺利进行。

|  |  |
| --- | --- |
| **字段名** | **说明** |
| product\_id | 商品编号，自动生成 |
| user\_id | 外键关联到User表，表示商品所属用户 |
| category1 | 商品类别 1，从Category1枚举类中取值 |
| category2 | 商品类别 2，从Category2枚举类中取值 |
| name | 商品名称 |
| price | 商品价格 |
| description | 商品描述 |
| phone | 联系电话 |
| address | 卖家地址 |
| images | 商品图片（存储图片路径的列表） |
| provide\_service | 是否提供上门服务 |
| rental\_period | 租赁周期（天/月），从RentalPeriod枚举类中取值 |
| created\_at | 商品创建时间 |
| updated\_at | 商品更新时间 |
| product\_status | 商品状态，从ProductStatus枚举类中取值 |
| auction\_end\_time | 拍卖结束时间（可为空） |

（3）**Order（订单）表**主要负责记录商品交易订单的详细情况，是跟踪商品交易流程、把控交易状态以及开展交易数据统计分析的关键数据表。通过 order\_id 唯一标识每个订单，借助 buyer\_openid 和 product\_id 分别关联到 User 表和 Product 表，将订单与具体的买家和商品对应起来，同时记录像订单成交价、创建时间、交易状态以及在租赁订单场景下的租期等关键信息，以此完整呈现订单从生成到完成整个过程中的各项要素，方便系统根据不同的交易状态执行相应操作、买卖双方查看订单进展以及后续基于订单数据进行各类业务分析和管理。

|  |  |
| --- | --- |
| **字段名** | **说明** |
| order\_id | 订单编号，自动生成 |
| buyer\_openid | 买家 openid |
| product\_id | 外键关联到Product表，表示订单对应的商品 |
| created\_at | 订单创建时间 |
| transaction\_price | 订单成交价 |
| order\_status | 交易状态，从OrderStatus枚举类中取值 |
| rental\_period | 租期，可为空 |

（4）**Favorite（收藏）表**用于存储用户收藏商品的记录信息，旨在体现用户对感兴趣商品的收藏行为，构建起用户与商品之间多对多的收藏关系。以自动生成的id作为主键来唯一区分每条收藏记录，通过user\_id和product\_id外键分别关联到User表和Product表，确保收藏记录与具体的用户和商品相对应，并且借助联合唯一约束避免同一用户重复收藏同一商品的情况出现，使收藏数据更具合理性和准确性，方便用户后续快速查看自己收藏的商品以及系统基于收藏数据进行相关的个性化推荐等功能拓展。

| **字段名** | **说明** |
| --- | --- |
| id | 主键 |
| user\_id | 外键关联到User表，表示收藏的用户 |
| product\_id | 外键关联到Product表，表示被收藏的商品 |
| created\_at | 收藏时间 |

（5）**History（浏览历史）表**的作用是记录用户浏览商品的历史信息，它能够精准地捕捉用户在系统内浏览不同商品的行为轨迹。以自动生成的id作为主键便于对每条浏览历史记录进行管理，通过user\_id和product\_id外键分别关联到User表和Product表，将浏览历史与具体的用户和商品相关联，确保数据的准确性和关联性，再结合viewed\_at字段记录浏览时间，不仅方便用户回顾之前查看过的商品，还能为系统后续分析用户浏览行为模式、实现个性化推荐等功能提供重要的数据基础，助力提升用户体验以及系统的智能化服务水平。

|  |  |
| --- | --- |
| **字段名** | **说明** |
| id | 主键 |
| user\_id | 外键关联到User表，表示浏览历史对应的用户 |
| product\_id | 外键关联到Product表，表示被浏览的商品 |
| viewed\_at | 记录浏览时间 |

**小组项目测试与问题解决记录**

下面是我们小组对于项目成品的测试记录，记录了我们在测试当中所遇到的问题，并且对于在测试当中所遇到的问题进行彻底的解决。

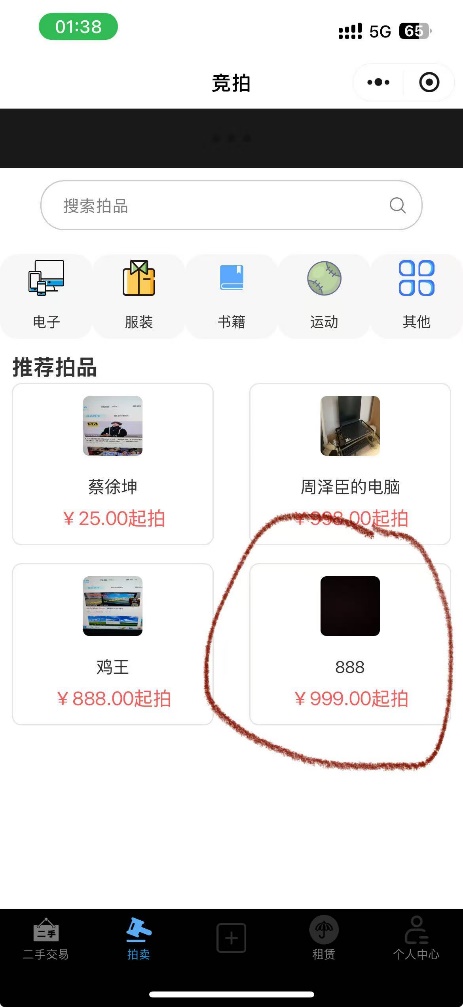
1. 买家确认收货后卖家无法点击完成交易√
2. 卖家的拍卖还在进行中（还没到截止时间），买家出了更高的价格后，在“个人中心”的“查看订单”里就可以收货了，而且在拍卖界面仍然显示着已收货的拍卖商品，再次合理出价会显示“该商品尚未拍卖”。



1. 这个问题不大——搜索时，会对简体中文与繁体中文进行区分。（比如：“电脑”跟“電腦”搜出来的东西不一样）
2. 在发布拍卖商品时，如果输入了非法的拍卖截止时间，点击发布后再将其转换成二手交易与租赁时，点击发布商品后依旧显示拍卖时间错误。



1. 卖家上架拍卖商品后，在“个人中心”的“查看订单”里取消订单后，在拍卖的主页面还能看到该商品，再次合理出价会显示“该商品尚未拍卖”。（拍卖商品下架不了）



1. 上架商品之后，多点几次发布商品会上架多个相同的商品，而且发布成功之后文本还会存在，只有重新进入小程序才会消失。我觉得可以在点击发布商品成功之后清除文本和图片内容。



1. 好像拍卖的下拉刷新不会自己回弹，和其他两个不一样



1. 我是卖家，却在查看订单的时候，显示我拍卖的物品的取消订单和确认收货。

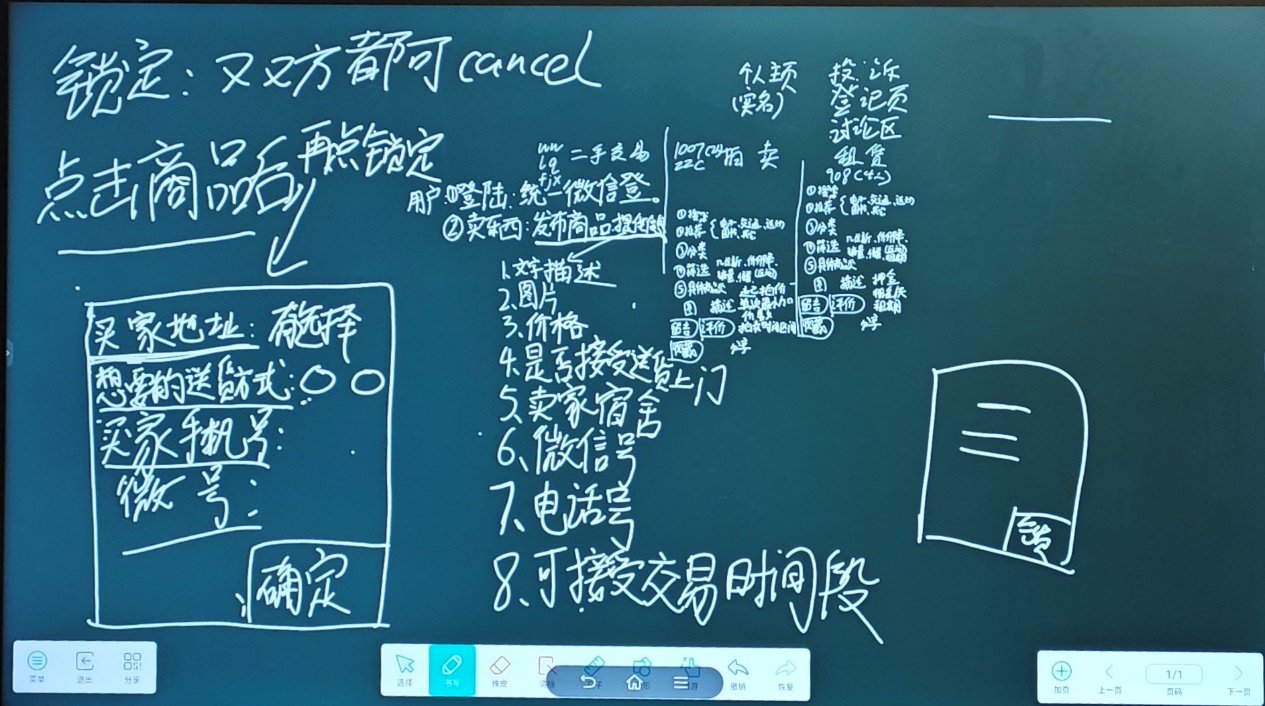


**小组开会内容记录**

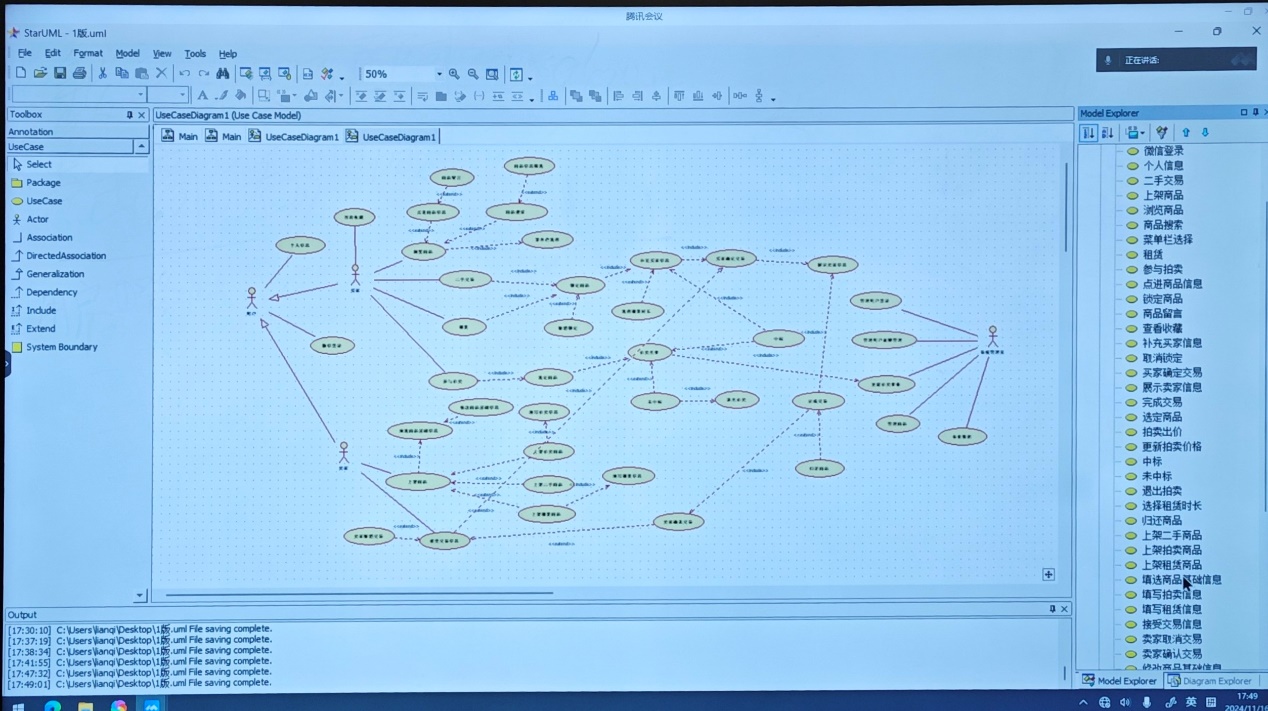
我们小组一共开了三次组会，每一次都会针对阶段性的问题进行总结，并且检查组员之间的进度问题。当然还有线上会议，不过线上会议并没有记录。下面展示了我们线下开会的一些截图，以及内容概要。

**第一次小组会议：**主要讨论了我们项目的主题内容以及需求，还有绘制用例图，这是项目的开端。

下面是我们当时罗列的项目需求分析：

****

下面是我们当时绘制的用例图：

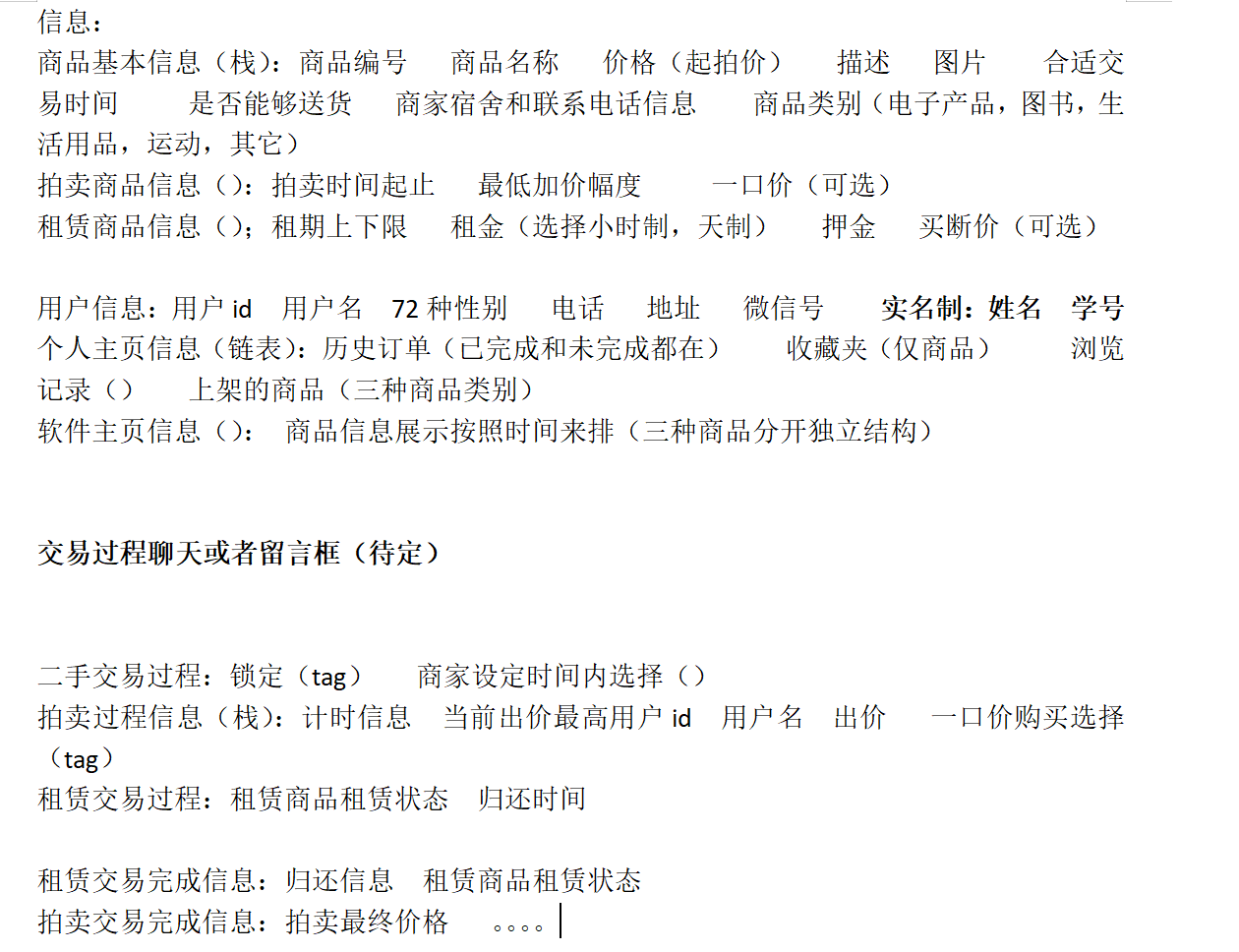


下面是我们当时开会所拍的合影照片：



**第二次小组会议：**主要制定我们小组项目所有的类图，还有确定我们项目的其他很多细节，这时候已经开始编写前后端代码了。

下面是我们小组当时的一些类图草稿记录。

****

**第三次小组会议：**也是最后一次会议，我们已经完成了前后端项目，然后现场进行前后端接口拼接测试，同时分配一些其他后续收尾任务。

下面是我们当时的拍照合影：



**系统实现**

## 配置文件示例

### API端口配置及分析

|  |
| --- |
| URL configuration for myproject project. The `urlpatterns` list routes URLs to views. For more information please see:  https://docs.djangoproject.com/en/4.2/topics/http/urls/ Examples: Function views  1. Add an import: from my\_app import views  2. Add a URL to urlpatterns: path('', views.home, name='home') Class-based views  1. Add an import: from other\_app.views import Home  2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home') Including another URLconf  1. Import the include() function: from django.urls import include, path  2. Add a URL to urlpatterns: path('blog/', include('blog.urls')) """ from django.conf import settings from django.conf.urls.static import static from django.urls import path  # 导入 方法模块 from myapp import publish, rentalgoods, login, createorder, uploadimages, secondhandgoods, downloadimages, \  auctiongoods from myapp import selectgoods, secondhandsearch, rentalsearch, changeuserdata, getuserinfo,selectcategory,auctionsearch,deleteproduct,getorder,cancelorder,checkauctionstatus,createrentalorder,finishorder,getgoods,selectauction from myapp.history import addtohistory, clearhistory, gethistory from myapp.favorites import addtofavorites, getfavorites, removefromfavorites from myapp import changeorder,bid,getsellerorder urlpatterns = ([ path('api/products/', publish.publish\_product, name='publish\_product'), path('api/login/', login.wechat\_login, name='wechat\_login'), path('api/createorder/', createorder.create\_order, name='create\_order'),  path('api/uploadimages/',uploadimages.upload\_images, name='upload\_images'), path('api/secondhandgoods/', secondhandgoods.secondhand\_goods, name='secondhand\_goods'), path('api/rentalgoods/', rentalgoods.rental\_goods, name='rental\_goods'), path('api/downloadimages/<str:image\_name>/', downloadimages.download\_image, name='download\_image'), path('api/selectgoods/', selectgoods.select\_goods, name='select\_goods'), path('api/secondhandsearch/', secondhandsearch.secondhand\_search, name='secondhand\_search'), path('api/rentalsearch/', rentalsearch.rental\_search, name='rental\_search'), path('api/changeuserdata/',changeuserdata.change\_user\_data, name='change\_user\_data'), path('api/addtohistory/', addtohistory.add\_to\_history, name='add\_to\_history'), path('api/addtofavorites/', addtofavorites.add\_to\_favorites, name='add\_to\_favorites'), path('api/clearhistory/', clearhistory.clear\_history, name='clear\_history'), path('api/gethistory/', gethistory.get\_history, name='get\_history'), path('api/getfavorites/', getfavorites.get\_favorites, name='get\_favorites'), path('api/getuserinfo/', getuserinfo.get\_user\_info, name='get\_user\_info'), path('api/removefromfavorites/', removefromfavorites.remove\_from\_favorites, name='remove\_from\_favorites'), path('api/auctiongoods/', auctiongoods.auction\_goods, name='auction\_goods'), path('api/selectcategory/', selectcategory.select\_category, name='select\_category'), path('api/auctionsearch/', auctionsearch.auction\_search, name='auction\_search'), path('api/deleteproduct/', deleteproduct.delete\_product, name='delete\_product'), path('api/getorder/',getorder.get\_user\_orders, name='get\_user\_orders'), path('api/cancelorder/', cancelorder.cancel\_order, name='cancel\_order'), path('api/checkauctionstatus/',checkauctionstatus.check\_auction\_status, name='check\_auction\_status'), path('api/createrentalorder/', createrentalorder.create\_rental\_order, name='create\_rental\_order'), path('api/finishorder/',finishorder.finish\_order, name='finish\_order'), path('api/getgoods/',getgoods.get\_goods, name='get\_goods'), path('api/selectauction/', selectauction.select\_auction, name='select\_auction'), path('api/changeorder/',changeorder.change\_order, name='change\_order'), path('api/getsellerorder/', getsellerorder.get\_seller\_orders, name='get\_seller\_orders'), path('api/bid/',bid.bid, name='bid'),  ]  + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)) |

这段代码是Django项目中的URL配置，用于将URL路径与对应的视图函数或类关联起来。每个URL路径都映射到一个具体的视图函数，在视图中处理业务逻辑并返回响应

## 功能实现及代码解释

### 查询拍卖商品并返回它们的相关信息

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt from myapp.models import Product, Category1, ProductStatus  @csrf\_exempt def auction\_goods(request):  if request.method == 'GET':  try:  *# 查询所有 category1 为 'AUCTION' 且商品状态为 'LIST' 的商品* auction\_products = Product.objects.filter(  category1=Category1.AUCTION,  product\_status=ProductStatus.LIST *# 只返回状态为 'LIST' (上架中) 的商品* )   product\_list = []  for product in auction\_products:  *# 获取商品的图片路径列表* image\_urls = []  for image\_path in product.images:  *# 去掉 /media/ 前缀* image\_url = image\_path.replace('/media/', '')  image\_urls.append(image\_url)    *# 创建一个商品字典，并加入修改后的图片路径* product\_data = {  'product\_id': product.product\_id,  'name': product.name,  'price': str(product.price), *# 将价格转换为字符串，避免浮动精度问题* 'description': product.description,  'address': product.address,  'phone': product.phone,  'images': image\_urls *# 存储去掉 /media/ 前缀后的图片路径* }  product\_list.append(product\_data)   *# 返回所有符合条件的商品数据* return JsonResponse({'products': product\_list}, status=200)   except Exception as e:  return JsonResponse({'error': str(e)}, status=500)   *# 如果不是 GET 请求，返回方法不允许* return JsonResponse({'error': 'Method not allowed'}, status=405) |

### 基于关键词的商品过滤和展示功能

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import json from myapp.models import Product, Category1, ProductStatus  @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def auction\_search(request):  if request.method == 'POST':  try:  *# 获取前端传来的关键词* body = json.loads(request.body) *# 解析请求体* keyword = body.get('keyword', '').strip() *# 获取关键词并去除空格* if not keyword:  return JsonResponse({'error': 'Keyword is required'}, status=400)   *# 筛选出 category1 为 'AUCTION' 且商品状态为 'LIST' 的商品* products = Product.objects.filter(  category1=Category1.AUCTION,  product\_status=ProductStatus.LIST *# 只查询上架中状态的商品* )   *# 再进行商品名称的模糊匹配* products = products.filter(name\_\_icontains=keyword) *# `icontains` 用于不区分大小写的模糊匹配   # 将查询结果转换为字典列表* product\_list = []  for product in products:  *# 获取商品的图片路径列表* image\_urls = []  for image\_path in product.images:  *# 去掉 /media/ 前缀* image\_url = image\_path.replace('/media/', '')  image\_urls.append(image\_url)   *# 创建一个商品字典，并加入修改后的图片路径* product\_data = {  'product\_id': product.product\_id,  'name': product.name,  'price': str(product.price), *# 将价格转换为字符串，避免浮动精度问题* 'description': product.description,  'address': product.address,  'phone': product.phone,  'category1': product.category1,  'category2': product.category2,  'images': image\_urls *# 存储去掉 /media/ 前缀后的图片路径* }  product\_list.append(product\_data)   *# 返回匹配的商品列表* return JsonResponse({'products': product\_list}, status=200)   except json.JSONDecodeError:  return JsonResponse({'error': 'Invalid JSON data'}, status=400)  except Exception as e:  return JsonResponse({'error': str(e)}, status=500)   else:  return JsonResponse({'error': 'Invalid HTTP method'}, status=405) |

### 基于商品类别和状态筛选出当前上架的租赁商品

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt from myapp.models import Product, Category1, ProductStatus  @csrf\_exempt def rental\_goods(request):  if request.method == 'GET':  try:  *# 查询所有 category1 为 'RENTAL' 且商品状态为 'LIST' 的商品* rental\_products = Product.objects.filter(  category1=Category1.RENTAL,  product\_status=ProductStatus.LIST *# 只返回状态为 'LIST' (上架中) 的商品* )   product\_list = []  for product in rental\_products:  *# 获取商品的图片路径列表* image\_urls = []  for image\_path in product.images:  *# 去掉 /media/ 前缀* image\_url = image\_path.replace('/media/', '')  image\_urls.append(image\_url)   *# 创建一个商品字典，并加入修改后的图片路径* product\_data = {  'product\_id': product.product\_id,  'name': product.name,  'price': str(product.price), *# 将价格转换为字符串，避免浮动精度问题* 'description': product.description,  'address': product.address,  'phone': product.phone,  'images': image\_urls, *# 存储去掉 /media/ 前缀后的图片路径* 'rental\_period': product.rental\_period,  }  product\_list.append(product\_data)   *# 返回所有符合条件的商品数据* return JsonResponse({'products': product\_list}, status=200)   except Exception as e:  return JsonResponse({'error': str(e)}, status=500)   *# 如果不是 GET 请求，返回方法不允许* return JsonResponse({'error': 'Method not allowed'}, status=405) |

### 关键词搜索租赁商品

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import json from myapp.models import Product, Category1, ProductStatus  @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def rental\_search(request):  if request.method == 'POST':  try:  *# 获取前端传来的关键词* body = json.loads(request.body) *# 解析请求体* keyword = body.get('keyword', '').strip() *# 获取关键词并去除空格* if not keyword:  return JsonResponse({'error': 'Keyword is required'}, status=400)   *# 筛选出 category1 为 '租赁' 且商品状态为 'LIST' 的商品* products = Product.objects.filter(  category1=Category1.RENTAL,  product\_status=ProductStatus.LIST *# 只查询上架中状态的商品* )   *# 再进行商品名称的模糊匹配* products = products.filter(name\_\_icontains=keyword) *# `icontains` 用于不区分大小写的模糊匹配   # 将查询结果转换为字典列表* product\_list = []  for product in products:  *# 获取商品的图片路径列表* image\_urls = []  for image\_path in product.images:  *# 去掉 /media/ 前缀* image\_url = image\_path.replace('/media/', '')  image\_urls.append(image\_url)   *# 创建一个商品字典，并加入修改后的图片路径* product\_data = {  'product\_id': product.product\_id,  'name': product.name,  'price': str(product.price), *# 将价格转换为字符串，避免浮动精度问题* 'description': product.description,  'address': product.address,  'phone': product.phone,  'category1': product.category1,  'category2': product.category2,  'rental\_period': product.rental\_period, *# 添加租赁周期字段* 'images': image\_urls *# 存储去掉 /media/ 前缀后的图片路径* }  product\_list.append(product\_data)   *# 返回匹配的商品列表* return JsonResponse({'products': product\_list}, status=200)   except json.JSONDecodeError:  return JsonResponse({'error': 'Invalid JSON data'}, status=400)  except Exception as e:  return JsonResponse({'error': str(e)}, status=500)   else:  return JsonResponse({'error': 'Invalid HTTP method'}, status=405) |

### 查询并返回所有二手商品

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import json from myapp.models import Product, Category1, ProductStatus  @csrf\_exempt def secondhand\_goods(request):  if request.method == 'GET':  try:  *# 查询所有 category1 为 'SECOND\_HAND' 且商品状态为 'LIST' 的商品* secondhand\_products = Product.objects.filter(  category1=Category1.SECOND\_HAND,  product\_status=ProductStatus.LIST *# 确保返回的商品是上架中状态* )   *# 将查询结果转换为字典列表，并去掉 images 中的 /media/ 前缀* product\_list = []  for product in secondhand\_products:  *# 获取商品的图片路径列表* image\_urls = []  for image\_path in product.images:  *# 去掉 /media/ 前缀* image\_url = image\_path.replace('/media/', '')  image\_urls.append(image\_url)   *# 创建一个商品字典，并加入修改后的图片路径* product\_data = {  'product\_id': product.product\_id,  'name': product.name,  'price': str(product.price), *# 将价格转换为字符串，避免浮动精度问题* 'description': product.description,  'address': product.address,  'phone': product.phone,  'images': image\_urls *# 存储去掉 /media/ 前缀后的图片路径* }  product\_list.append(product\_data)   return JsonResponse({'products': product\_list}, status=200)   except Exception as e:  return JsonResponse({'error': str(e)}, status=500)   *# 如果不是 GET 请求，返回方法不允许* return JsonResponse({'error': 'Method not allowed'}, status=405) |

### 关键词搜索二手商品

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import json from myapp.models import Product, Category1, ProductStatus  @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def secondhand\_search(request):  if request.method == 'POST':  try:  *# 获取前端传来的关键词* body = json.loads(request.body) *# 解析请求体* keyword = body.get('keyword', '').strip() *# 获取关键词并去除空格* if not keyword:  return JsonResponse({'error': 'Keyword is required'}, status=400)   *# 筛选出 category1 为 '二手交易' 且商品状态为 'LIST' 的商品* products = Product.objects.filter(  category1=Category1.SECOND\_HAND,  product\_status=ProductStatus.LIST *# 只查询上架中状态的商品* )   *# 再进行商品名称的模糊匹配* products = products.filter(name\_\_icontains=keyword) *# `icontains` 用于不区分大小写的模糊匹配   # 将查询结果转换为字典列表* product\_list = []  for product in products:  *# 获取商品的图片路径列表* image\_urls = []  for image\_path in product.images:  *# 去掉 /media/ 前缀* image\_url = image\_path.replace('/media/', '')  image\_urls.append(image\_url)   *# 创建一个商品字典，并加入修改后的图片路径* product\_data = {  'product\_id': product.product\_id,  'name': product.name,  'price': str(product.price), *# 将价格转换为字符串，避免浮动精度问题* 'description': product.description,  'address': product.address,  'phone': product.phone,  'category1': product.category1,  'category2': product.category2,  'images': image\_urls *# 存储去掉 /media/ 前缀后的图片路径* }  product\_list.append(product\_data)   *# 返回匹配的商品列表* return JsonResponse({'products': product\_list}, status=200)   except json.JSONDecodeError:  return JsonResponse({'error': 'Invalid JSON data'}, status=400)  except Exception as e:  return JsonResponse({'error': str(e)}, status=500)   else:  return JsonResponse({'error': 'Invalid HTTP method'}, status=405) |

### 商品分类筛选二手商品

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt from myapp.models import Product, ProductStatus import json  @csrf\_exempt def select\_category(request):  if request.method == 'POST':  try:  *# 解析前端传入的json数据* body = json.loads(request.body)  category1 = body.get('category1')  category2 = body.get('category2')   if not category1 or not category2:  return JsonResponse({"message": "category1 and category2 are required."}, status=400)   *# 根据传入的category1和category2筛选商品* products = Product.objects.filter(category1=category1, category2=category2, product\_status=ProductStatus.LIST)   product\_list = []  for product in products:  image\_urls = []  for image\_path in product.images:  *# 去掉 /media/ 前缀* image\_url = image\_path.replace('/media/', '')  image\_urls.append(image\_url)   product\_info = {  'product\_id': product.product\_id,  'name': product.name,  'price': str(product.price), *# 转换为字符串以避免精度丢失* 'description': product.description,  'phone': product.phone,  'address': product.address,  'images': image\_urls,  'provide\_service': product.provide\_service,  'rental\_period': product.rental\_period,  'created\_at': product.created\_at.isoformat(),  'updated\_at': product.updated\_at.isoformat(),  'product\_status': product.product\_status,  }  product\_list.append(product\_info)   *# 返回筛选后的商品信息* return JsonResponse({"products": product\_list}, status=200)   except Exception as e:  *# 发生异常时返回错误信息* return JsonResponse({"error": str(e)}, status=500) |

### 一个商品添加到用户的收藏夹

|  |
| --- |
| import json from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt from myapp.models import User, Product, Favorite  @csrf\_exempt def add\_to\_favorites(request):  *"""  将商品添加到用户的收藏夹  """* if request.method == 'POST':  try:  data = json.loads(request.body)  openid = data.get('openid')  product\_id = data.get('product\_id')   *# 查找用户和商品* user = User.objects.get(openid=openid)  product = Product.objects.get(product\_id=product\_id)   *# 检查用户是否已经收藏了这个商品* if Favorite.objects.filter(user=user, product=product).exists():  return JsonResponse({'success': False, 'message': '该商品已在收藏夹中'}, status=400)   *# 添加到收藏夹* Favorite.objects.create(user=user, product=product)   return JsonResponse({'success': True, 'message': '商品已添加到收藏夹'}, status=200)   except User.DoesNotExist:  return JsonResponse({'success': False, 'message': '用户不存在'}, status=404)  except Product.DoesNotExist:  return JsonResponse({'success': False, 'message': '商品不存在'}, status=404)  except Exception as e:  return JsonResponse({'success': False, 'message': f'发生错误: {str(e)}'}, status=500)   return JsonResponse({'success': False, 'message': '仅支持 POST 请求'}, status=400) |

### 获取指定用户的所有收藏商品

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import json import logging from myapp.models import User, Favorite, Product  logger = logging.getLogger(\_\_name\_\_)  @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def get\_favorites(request):  if request.method == 'POST':  try:  body = json.loads(request.body) *# 解析请求体* openid = body.get('openid') *# 获取用户的 openid* logger.debug(f'OpenID: {openid}')  if not openid:  return JsonResponse({'error': 'openid is required'}, status=400)     *# 查找用户是否存在* user = User.objects.get(openid=openid)   *# 获取用户收藏的商品* favorites = Favorite.objects.filter(user=user)   *# 如果没有收藏商品，返回空列表* if not favorites:  return JsonResponse({'message': 'No favorites found'}, status=200)   *# 构造商品数据* favorite\_products = []  for favorite in favorites:  product = favorite.product *# 获取收藏的商品* product\_data = {  'product\_id': product.product\_id,  'name': product.name,  'price': str(product.price), *# 将价格转换为字符串* 'description': product.description,  'phone': product.phone,  'address': product.address,  'images': product.images, *# 商品图片* 'provide\_service': product.provide\_service,  'rental\_period': product.rental\_period,  }  favorite\_products.append(product\_data)   *# 返回收藏夹商品的 JSON 响应* return JsonResponse({'favorites': favorite\_products}, safe=False, status=200)   except User.DoesNotExist:  *# 如果找不到用户，返回 404 错误* logger.error(f'User with openid {openid} does not exist')  return JsonResponse({'error': 'User not found'}, status=404)  except json.JSONDecodeError:  *# 如果请求体不是有效的 JSON 数据* return JsonResponse({'error': 'Invalid JSON data'}, status=400)  except Exception as e:  *# 捕获其他异常并记录* logger.error(f'Error: {str(e)}')  return JsonResponse({'error': 'An error occurred'}, status=500)   else:  *# 如果不是 POST 请求* return JsonResponse({'error': 'Invalid HTTP method'}, status=405) |

### 从用户的收藏夹中移除指定的商品

|  |
| --- |
| import json from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt from myapp.models import User, Product, Favorite  @csrf\_exempt def remove\_from\_favorites(request):  *"""  从收藏夹中移除商品  """* if request.method == 'POST':  try:  data = json.loads(request.body)  openid = data.get('openid')  product\_id = data.get('product\_id')   *# 查找用户和商品* user = User.objects.get(openid=openid)  product = Product.objects.get(product\_id=product\_id)   *# 查找用户是否已经收藏了这个商品* favorite = Favorite.objects.filter(user=user, product=product).first()   if not favorite:  return JsonResponse({'success': False, 'message': '该商品不在收藏夹中'}, status=400)   *# 删除收藏* favorite.delete()   return JsonResponse({'success': True, 'message': '商品已从收藏夹移除'}, status=200)   except User.DoesNotExist:  return JsonResponse({'success': False, 'message': '用户不存在'}, status=404)  except Product.DoesNotExist:  return JsonResponse({'success': False, 'message': '商品不存在'}, status=404)  except Exception as e:  return JsonResponse({'success': False, 'message': f'发生错误: {str(e)}'}, status=500)   return JsonResponse({'success': False, 'message': '仅支持 POST 请求'}, status=400) |

### 将用户浏览商品的记录添加到历史记录

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import json from myapp.models import User, Product, History  @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def add\_to\_history(request):  if request.method == 'POST':  try:  body = json.loads(request.body) *# 解析请求体* openid = body.get('openid') *# 获取用户的 openid* product\_id = body.get('product\_id') *# 获取商品的 product\_id* if not openid or not product\_id:  return JsonResponse({'error': 'openid and product\_id are required'}, status=400)   *# 查找用户和商品* user = User.objects.get(openid=openid)  product = Product.objects.get(product\_id=product\_id)   *# 查询是否已经有该商品的历史记录* existing\_history = History.objects.filter(user=user, product=product).first()   if existing\_history:  *# 如果存在历史记录，更新（或者删除再创建）* existing\_history.delete() *# 删除旧记录  # 重新创建新的历史记录* History.objects.create(user=user, product=product)   return JsonResponse({'success': True, 'message': '历史浏览记录已更新'}, status=200)  else:  *# 如果没有历史记录，直接创建新的记录* History.objects.create(user=user, product=product)  return JsonResponse({'success': True, 'message': '历史浏览记录添加成功'}, status=200)   except User.DoesNotExist:  return JsonResponse({'error': 'User not found'}, status=404)  except Product.DoesNotExist:  return JsonResponse({'error': 'Product not found'}, status=404)  except json.JSONDecodeError:  return JsonResponse({'error': 'Invalid JSON data'}, status=400)  except Exception as e:  return JsonResponse({'error': f'An error occurred: {str(e)}'}, status=500)  else:  return JsonResponse({'error': 'Invalid HTTP method'}, status=405) |

### 用户清空自己的浏览历史

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import json import logging from myapp.models import User, History  logger = logging.getLogger(\_\_name\_\_)  @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def clear\_history(request):  if request.method == 'POST':  try:  body = json.loads(request.body) *# 解析请求体* openid = body.get('openid') *# 获取用户的 openid* if not openid:  return JsonResponse({'error': 'openid is required'}, status=400)   *# 查找用户是否存在* user = User.objects.get(openid=openid)   *# 删除该用户的所有历史浏览记录* deleted\_count, \_ = History.objects.filter(user=user).delete()   *# 判断是否有记录被删除* if deleted\_count > 0:  return JsonResponse({'success': True, 'message': f'清空了 {deleted\_count} 条历史浏览记录'}, status=200)  else:  return JsonResponse({'success': False, 'message': '没有找到历史浏览记录'}, status=200)   except User.DoesNotExist:  *# 如果找不到用户，返回 404 错误* logger.error(f'User with openid {openid} does not exist')  return JsonResponse({'error': 'User not found'}, status=404)  except json.JSONDecodeError:  *# 如果请求体不是有效的 JSON 数据* return JsonResponse({'error': 'Invalid JSON data'}, status=400)  except Exception as e:  *# 捕获其他异常并记录* logger.error(f'Error: {str(e)}')  return JsonResponse({'error': 'An error occurred'}, status=500)   else:  *# 如果不是 POST 请求* return JsonResponse({'error': 'Invalid HTTP method'}, status=405) |

### 返回用户的历史记录

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import json import logging from myapp.models import User, History, Product  *# 设置日志记录* logger = logging.getLogger(\_\_name\_\_)  @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def get\_history(request):  if request.method == 'POST':  try:  logger.debug('Received request body: %s', request.body) *# Log raw request body   # 解析请求体* body = json.loads(request.body)  openid = body.get('openid') *# 获取用户的 openid* logger.debug('Extracted openid: %s', openid) *# Log the extracted openid   # 检查是否提供了 openid* if not openid:  logger.warning('Missing openid in request body') *# Log warning if openid is missing* return JsonResponse({'error': 'openid is required'}, status=400)   try:  *# 查找用户是否存在* user = User.objects.get(openid=openid)  logger.debug('Found user: %s', user) *# Log the user object* except User.DoesNotExist:  *# 如果用户不存在，返回 404 错误* logger.error(f'User with openid {openid} does not exist') *# Log error when user is not found* return JsonResponse({'error': 'User not found'}, status=404)   *# 获取用户浏览的历史商品* history\_items = History.objects.filter(user=user).order\_by('-viewed\_at')  logger.debug('Found %d history items for user %s', history\_items.count(), openid) *# Log number of history items   # 如果没有浏览历史，返回空列表* if not history\_items:  logger.info('No history found for user %s', openid) *# Log info if no history is found* return JsonResponse({'message': 'No history found'}, status=200)   *# 构造商品数据* history\_products = []  for history in history\_items:  try:  product = history.product *# 获取浏览的商品* logger.debug('Processing product: %s', product) *# Log each product being processed   # 构造每个商品的数据，按照需求只返回名称、价格、图片和租赁周期* product\_data = {  'name': product.name,  'price': str(product.price), *# 将价格转换为字符串* 'images': product.images, *# 商品图片* 'rental\_period': product.rental\_period *# 租赁周期* }  history\_products.append(product\_data)  except Product.DoesNotExist:  *# 如果对应商品不存在，跳过该记录并继续处理下一个* logger.warning(f"Product for history item {history.id} does not exist. Skipping.") *# Log missing product* continue *# 跳过当前历史记录   # 返回浏览历史商品的 JSON 响应* logger.debug('Returning %d history products', len(history\_products)) *# Log number of products being returned* return JsonResponse({'history': history\_products}, safe=False, status=200)   except json.JSONDecodeError as e:  *# 如果请求体不是有效的 JSON 数据* logger.error('Invalid JSON data: %s', e) *# Log the JSON decode error* return JsonResponse({'error': 'Invalid JSON data'}, status=400)  except Exception as e:  *# 捕获其他异常并记录* logger.error(f'Unexpected error occurred: {str(e)}') *# Log unexpected errors* return JsonResponse({'error': 'An error occurred'}, status=500)   else:  *# 如果不是 POST 请求* logger.warning('Invalid HTTP method used: %s', request.method) *# Log invalid HTTP method* return JsonResponse({'error': 'Invalid HTTP method'}, status=405) |

### 查找商品关联的最新订单

|  |
| --- |
| from django.http import JsonResponse from django.shortcuts import get\_object\_or\_404 from django.views.decorators.csrf import csrf\_exempt import json import logging from myapp.models import Product, Order  logger = logging.getLogger(\_\_name\_\_)  @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def select\_auction(request):  if request.method == 'POST':  try:  *# 获取前端传递的 JSON 数据* body = json.loads(request.body)  product\_id = body.get('product\_id') *# 获取传递的 product\_id* if not product\_id:  return JsonResponse({'error': 'product\_id is required'}, status=400)   *# 获取商品信息* product = get\_object\_or\_404(Product, product\_id=product\_id)   *# 获取当前订单信息（可能存在多个订单，取最新的一个）* order = Order.objects.filter(product=product).order\_by('-created\_at').first()   if not order:  return JsonResponse({'error': 'No order found for this product'}, status=404)   *# 将商品数据打包成字典并返回* product\_data = {  'product\_id': product.product\_id,  'transaction\_price': order.transaction\_price, *# 当前订单价格* 'auction\_end\_time': product.auction\_end\_time, *# 拍卖结束时间* }   *# 返回 JSON 响应* return JsonResponse(product\_data, safe=False)   except Product.DoesNotExist:  *# 如果找不到商品，返回 404 错误* logger.error(f'Product with id {product\_id} does not exist')  return JsonResponse({'error': 'Product not found'}, status=404)  except json.JSONDecodeError:  return JsonResponse({'error': 'Invalid JSON data'}, status=400)   else:  return JsonResponse({'error': 'Invalid HTTP method'}, status=405) |

### 获取商品的详细信息

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import json import logging from myapp.models import Product, User  logger = logging.getLogger(\_\_name\_\_) @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def select\_goods(request):  if request.method == 'POST':  *# 获取前端传递的 product\_id 参数* try:  body = json.loads(request.body) *# 解析请求体* product\_id = body.get('product\_id') *# 获取 product\_id* if not product\_id:  return JsonResponse({'error': 'product\_id is required'}, status=400)   *# 查询商品信息* product = Product.objects.get(product\_id=product\_id)  seller\_name = product.user.username if product.user else 'Unknown Seller'  seller\_openid = product.user.openid if product.user else 'Unknown Seller'   image\_urls = []  for image\_path in product.images:  *# 去掉 /media/ 前缀* image\_url = image\_path.replace('/media/', '')  image\_urls.append(image\_url)   *# 将商品数据打包成字典并返回* product\_data = {  'product\_id': product.product\_id,  'name': product.name,  'price': str(product.price), *# 将价格转换为字符串* 'description': product.description,  'phone': product.phone,  'address': product.address,  'images': image\_urls, *# 商品图片* 'provide\_service': product.provide\_service,  'rental\_period': product.rental\_period,  'seller\_name': seller\_name, *# 卖家的用户名* 'seller\_openid': seller\_openid,  }   *# 返回 JSON 响应* return JsonResponse(product\_data, safe=False)   except Product.DoesNotExist:  *# 如果找不到商品，返回 404 错误* logger.error(f'Product with id {product\_id} does not exist')  return JsonResponse({'error': 'Product not found'}, status=404)  except json.JSONDecodeError:  return JsonResponse({'error': 'Invalid JSON data'}, status=400)   else:  return JsonResponse({'error': 'Invalid HTTP method'}, status=405) |

### 创建并保存订单

|  |
| --- |
| from django.shortcuts import get\_object\_or\_404 from myapp.models import Product, Order, ProductStatus, Category1, OrderStatus from django.http import JsonResponse import json from django.views.decorators.csrf import csrf\_exempt  @csrf\_exempt def create\_order(request):  if request.method == 'POST':  try:  *# 解析请求体中的JSON数据* data = json.loads(request.body)   *# 获取传递的 product\_id 和 buyer\_openid* product\_id = data.get('product\_id')  buyer\_openid = data.get('buyer\_openid')   *# 检查参数是否传递完整* if not product\_id or not buyer\_openid:  return JsonResponse({'message': '缺少参数: product\_id 或 buyer\_openid'}, status=400)   *# 获取对应的商品对象，如果商品不存在则返回 404 错误* product = get\_object\_or\_404(Product, product\_id=product\_id)  if product.category1 == '拍卖':  *# 创建订单* order = Order.objects.create(  buyer\_openid=buyer\_openid,  product=product, *# 关联商品* transaction\_price=product.price, *# 订单成交价格* )  else: order = Order.objects.create(  product=product, *# 关联商品* transaction\_price=product.price, *# 订单成交价格* )     *# 设置商品状态和订单状态* if product.category1 == Category1.AUCTION:  *# 对于拍卖类商品，订单状态为 交易中 (ONGOING)* product.product\_status = ProductStatus.LIST *# 拍卖商品上架* order.order\_status = OrderStatus.ONGOING *# 订单状态为进行中* else:  *# 对于二手商品，订单状态为 已发布 (PUBLISH)* product.product\_status = ProductStatus.LIST *# 商品上架* order.order\_status = OrderStatus.PUBLISH *# 订单状态为已发布   # 保存商品和订单的状态* product.save()  order.save()   *# 返回订单创建成功的响应* return JsonResponse({  'order\_id': order.order\_id,  'product\_name': order.product.name,  'buyer\_openid': order.buyer\_openid,  'seller\_openid': order.product.user.openid,  'rental\_period': order.rental\_period, *# 返回租期信息* 'message': '订单创建成功'  })   except json.JSONDecodeError:  *# 如果请求体无法解析为 JSON，返回格式错误的响应* return JsonResponse({'message': '请求体格式错误'}, status=400)   except Exception as e:  *# 捕获其他异常并返回错误信息* return JsonResponse({'message': str(e)}, status=500) |

### 获取用户的订单信息

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import json import logging from myapp.models import Order, User, Product  logger = logging.getLogger(\_\_name\_\_)  @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def get\_user\_orders(request):  if request.method == 'POST':  try:  *# 获取前端传递的 JSON 数据* body = json.loads(request.body)  openid = body.get('openid') *# 获取传递的 openid* category1 = body.get('category1') *# 获取传递的 category1* if not openid:  return JsonResponse({'error': 'openid is required'}, status=400)   *# 查询该用户作为买家的所有订单* buyer\_orders = Order.objects.filter(buyer\_openid=openid).select\_related('product\_\_user')   *# 如果传递了 category1 参数，进行筛选* if category1:  buyer\_orders = buyer\_orders.filter(product\_\_category1=category1)   *# 如果该用户没有订单，返回空的订单列表* if not buyer\_orders.exists():  return JsonResponse({'message': '没有找到该用户的订单'}, status=200)   order\_list = []   *# 处理买家订单* for order in buyer\_orders:  product = order.product  seller = product.user *# 获取商品的卖家信息* buyer = User.objects.filter(openid=order.buyer\_openid).first() *# 获取买家信息* if not seller:  logger.error(f"Seller for product {product.product\_id} not found.")  continue *# 如果找不到卖家信息，跳过该订单* order\_data = {  'type': 'buyer', *# 标记为买家订单* 'buyer': {  'username': buyer.username, *# 买家用户名* 'phone': buyer.phone, *# 买家联系电话* 'address': buyer.address, *# 买家地址* 'profile\_picture': buyer.profile\_picture.url if buyer.profile\_picture else None *# 买家头像* },  'order\_id': order.order\_id,  'product\_id': product.product\_id,  'category1': product.category1,  'product\_name': product.name,  'product\_images': product.images,  'transaction\_price': str(order.transaction\_price), *# 交易价格* 'order\_status': order.get\_order\_status\_display(), *# 获取订单状态的可读值* 'created\_at': order.created\_at.strftime('%Y-%m-%d %H:%M:%S'), *# 格式化时间* 'rental\_period': str(order.rental\_period) if order.rental\_period else None, *# 租期（如果有）* 'seller': {  'username': seller.username, *# 卖家用户名* 'phone': seller.phone, *# 卖家联系电话* 'address': seller.address, *# 卖家地址* 'profile\_picture': seller.profile\_picture.url if seller.profile\_picture else None *# 卖家头像* }  }  order\_list.append(order\_data)   *# 返回所有买家订单* return JsonResponse({'orders': order\_list}, status=200)   except json.JSONDecodeError:  return JsonResponse({'error': 'Invalid JSON data'}, status=400)   except Exception as e:  logger.error(f"Error: {str(e)}")  return JsonResponse({'error': str(e)}, status=500)   else:  return JsonResponse({'error': 'Invalid HTTP method'}, status=405) |

### 修改订单和商品状态

|  |
| --- |
| from django.views.decorators.csrf import csrf\_exempt from django.http import JsonResponse import json from django.shortcuts import get\_object\_or\_404 import logging from myapp.models import Product, Order, ProductStatus, OrderStatus, User  *# 设置日志记录* logger = logging.getLogger(\_\_name\_\_)  @csrf\_exempt def change\_order(request):  if request.method == 'POST':  *# 解析 JSON 数据* try:  data = json.loads(request.body)  except json.JSONDecodeError:  return JsonResponse({'error': '无效的 JSON 数据'}, status=400)   openid = data.get('openid')  product\_id = data.get('product\_id')   logger.info(f"openid: {openid}, product\_id: {product\_id}")   *# 参数验证* if not openid or not product\_id:  return JsonResponse({'error': 'openid 和 product\_id 是必填的'}, status=400)   try:  *# 获取用户信息* user = get\_object\_or\_404(User, openid=openid)   *# 获取商品信息* product = get\_object\_or\_404(Product, product\_id=product\_id)  logger.info(f"Found product: {product}")   *# 查找与该商品相关的订单* order = get\_object\_or\_404(Order, product=product)  logger.info(f"Found order: {order}")   *# 修改商品状态为锁定* product.product\_status = ProductStatus.LOCK  product.save()   *# 修改订单状态为交易中* order.buyer\_openid = openid  order.order\_status = OrderStatus.ONGOING  order.save()   *# 返回成功响应* return JsonResponse({'message': '订单状态和商品状态已成功更新'}, status=200)   except Exception as e:  logger.error(f"Error occurred: {e}")  return JsonResponse({'error': str(e)}, status=500)   return JsonResponse({'error': '请求方法不支持'}, status=405) |

### 取消订单

|  |
| --- |
| from django.shortcuts import get\_object\_or\_404 from myapp.models import Product, Order, ProductStatus, OrderStatus from django.http import JsonResponse import json from django.views.decorators.csrf import csrf\_exempt  @csrf\_exempt def cancel\_order(request):  if request.method == 'POST':  try:  *# 解析请求体中的JSON数据* data = json.loads(request.body)   *# 获取传递的 order\_id和buyer\_openid* order\_id = data.get('order\_id')  buyer\_openid = data.get('buyer\_openid')   *# 检查参数是否传递完整* if not order\_id or not buyer\_openid:  return JsonResponse({'message': '缺少参数: order\_id 或 buyer\_openid'}, status=400)   *# 获取对应的商品对象，如果商品不存在则返回 404 错误* order = get\_object\_or\_404(Order, order\_id=order\_id)   *# 检查交易是否处于交易中* if order.order\_status != OrderStatus.ONGOING:  return JsonResponse({'message': ' 订单处于不可取消状态'}, status=400)   *#订单取消* order.order\_status = OrderStatus.FAIL  order.save()   *# 商品重新上架* product = order.product  product.product\_status = ProductStatus.LIST  product.save()   *# 返回订单创建成功的响应* return JsonResponse({  'order\_id': order.order\_id,  'message': '订单已取消'  })   except json.JSONDecodeError:  *# 如果请求体无法解析为 JSON，返回格式错误的响应* return JsonResponse({'message': '请求体格式错误'}, status=400)   except Exception as e:  *# 捕获其他异常并返回错误信息* return JsonResponse({'message': str(e)}, status=500) |

### 创建或修改一个租赁订单

|  |
| --- |
| from django.shortcuts import get\_object\_or\_404 from myapp.models import Product, Order, ProductStatus, OrderStatus from django.http import JsonResponse import json from django.views.decorators.csrf import csrf\_exempt  @csrf\_exempt def create\_rental\_order(request):  if request.method == 'POST':  try:  *# 解析请求体中的JSON数据* data = json.loads(request.body)   *# 获取传递的 product\_id、buyer\_openid 和 rental\_period* product\_id = data.get('product\_id')  buyer\_openid = data.get('buyer\_openid')  rental\_period = data.get('rental\_period') *# 获取租期参数   # 检查参数是否传递完整* if not product\_id or not buyer\_openid or rental\_period is None:  return JsonResponse({'message': '缺少参数: product\_id, buyer\_openid 或 rental\_period'}, status=400)   *# 将租期转换为整数类型* try:  rental\_period = int(rental\_period)  except ValueError:  return JsonResponse({'message': '租期必须是有效的数字'}, status=400)   *# 检查租期是否是有效数字* if rental\_period <= 0:  return JsonResponse({'message': '租期必须是正整数'}, status=400)   *# 获取对应的商品对象，如果商品不存在则返回 404 错误* product = get\_object\_or\_404(Product, product\_id=product\_id)   *# 查找与该商品关联的正在进行中的订单* existing\_order = Order.objects.filter(product=product, order\_status=OrderStatus.PUBLISH).first()   if not existing\_order:  *# 如果商品没有进行中的订单，返回错误* return JsonResponse({'message': '没有找到进行中的订单'}, status=404)   *# 计算成交价（商品价格 \* 租期）* transaction\_price = product.price \* rental\_period   *# 修改现有订单* existing\_order.buyer\_openid = buyer\_openid *# 更新买家openid* existing\_order.rental\_period = rental\_period *# 更新租期* existing\_order.order\_status = OrderStatus.ONGOING *# 设置订单状态为交易中* existing\_order.transaction\_price = transaction\_price *# 更新成交价* existing\_order.save()   *# 修改商品状态为锁定* product.product\_status = ProductStatus.LOCK  product.save()   *# 返回订单修改成功的响应* return JsonResponse({  'order\_id': existing\_order.order\_id,  'product\_name': existing\_order.product.name,  'buyer\_openid': existing\_order.buyer\_openid,  'seller\_openid': existing\_order.product.user.openid,  'rental\_period': existing\_order.rental\_period, *# 返回租期信息* 'transaction\_price': existing\_order.transaction\_price, *# 返回计算后的成交价* 'message': '订单修改成功'  })   except json.JSONDecodeError:  *# 如果请求体无法解析为 JSON，返回格式错误的响应* return JsonResponse({'message': '请求体格式错误'}, status=400)   except Exception as e:  *# 捕获其他异常并返回错误信息* return JsonResponse({'message': str(e)}, status=500) |

### 将订单标记为完成

|  |
| --- |
| from django.shortcuts import get\_object\_or\_404 from myapp.models import Product, Order, ProductStatus, OrderStatus from django.http import JsonResponse import json from django.views.decorators.csrf import csrf\_exempt  @csrf\_exempt def finish\_order(request):  if request.method == 'POST':  try:  *# 解析请求体中的JSON数据* data = json.loads(request.body)   *# 获取传递的 order\_id和buyer\_openid* order\_id = data.get('order\_id')   *# 检查参数是否传递完整* if not order\_id:  return JsonResponse({'message': '缺少参数: order\_id 或 seller\_openid'}, status=400)   *# 获取对应的商品对象，如果商品不存在则返回 404 错误* order = get\_object\_or\_404(Order, order\_id=order\_id)   *# 检查交易是否处于交易中* if order.order\_status != OrderStatus.RECEIVE:  return JsonResponse({'message': ' 订单处于不可完成状态'}, status=400)   *#订单完成* order.order\_status = OrderStatus.FINISH  order.save()   *# 返回订单创建成功的响应* return JsonResponse({  'order\_id': order.order\_id,  'message': '订单已完成'  })   except json.JSONDecodeError:  *# 如果请求体无法解析为 JSON，返回格式错误的响应* return JsonResponse({'message': '请求体格式错误'}, status=400)   except Exception as e:  *# 捕获其他异常并返回错误信息* return JsonResponse({'message': str(e)}, status=500) |

### 处理卖家订单的查询

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import json import logging from myapp.models import Order, User  logger = logging.getLogger(\_\_name\_\_)  @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def get\_seller\_orders(request):  if request.method == 'POST':  try:  *# 获取前端传递的 JSON 数据* body = json.loads(request.body)  openid = body.get('openid') *# 获取传递的 openid* category1 = body.get('category1') *# 获取传递的 category1* if not openid:  return JsonResponse({'error': 'openid is required'}, status=400)   *# 查询该用户作为卖家的所有订单* seller\_orders = Order.objects.filter(product\_\_user\_\_openid=openid).select\_related('product', 'product\_\_user')   *# 如果传递了 category1 参数，进行筛选* if category1:  seller\_orders = seller\_orders.filter(product\_\_category1=category1)   *# 如果该用户没有作为卖家的订单，返回空的订单列表* if not seller\_orders.exists():  return JsonResponse({'message': '没有找到该用户的卖家订单'}, status=200)   order\_list = []   *# 处理卖家订单* for order in seller\_orders:  product = order.product  seller = product.user *# 获取商品的卖家信息* buyer = User.objects.filter(openid=order.buyer\_openid).first() *# 获取买家信息* if not buyer:  buyer = None *# 如果没有买家（未售出），将 buyer 设置为 None* order\_data = {  'type': 'seller', *# 标记为卖家订单* 'seller': {  'username': seller.username, *# 卖家用户名* 'phone': seller.phone, *# 卖家联系电话* 'address': seller.address, *# 卖家地址* 'profile\_picture': seller.profile\_picture.url if seller.profile\_picture else None *# 卖家头像* },  'order\_id': order.order\_id,  'product\_id': product.product\_id,  'category1': product.category1,  'product\_name': product.name,  'product\_images': product.images,  'transaction\_price': str(order.transaction\_price), *# 交易价格* 'order\_status': order.get\_order\_status\_display(), *# 获取订单状态的可读值* 'created\_at': order.created\_at.strftime('%Y-%m-%d %H:%M:%S'), *# 格式化时间* 'rental\_period': str(order.rental\_period) if order.rental\_period else None, *# 租期（如果有）* 'buyer': {  'username': buyer.username if buyer else None, *# 买家用户名，如果没有买家则为 None* 'phone': buyer.phone if buyer else None, *# 买家联系电话* 'address': buyer.address if buyer else None, *# 买家地址* 'profile\_picture': buyer.profile\_picture.url if buyer and buyer.profile\_picture else None *# 买家头像* } if buyer else {} *# 如果没有买家，返回空的字典* }  order\_list.append(order\_data)   *# 返回所有卖家订单* return JsonResponse({'orders': order\_list}, status=200)   except json.JSONDecodeError:  return JsonResponse({'error': 'Invalid JSON data'}, status=400)   except Exception as e:  logger.error(f"Error: {str(e)}")  return JsonResponse({'error': str(e)}, status=500)   else:  return JsonResponse({'error': 'Invalid HTTP method'}, status=405) |

### 处理订单收货操作

|  |
| --- |
| from django.shortcuts import get\_object\_or\_404 from myapp.models import Product, Order, ProductStatus, OrderStatus from django.http import JsonResponse, Http404 import json from django.views.decorators.csrf import csrf\_exempt import logging logger = logging.getLogger(\_\_name\_\_)   @csrf\_exempt def get\_goods(request):  if request.method == 'POST':  try:  *# 记录原始请求体数据用于调试* logger.info(f"Request body: {request.body.decode('utf-8')}")   *# 解析请求体中的JSON数据* data = json.loads(request.body)  logger.info(f"Received data: {data}")   *# 获取 order\_id 和 buyer\_openid* order\_id = data.get('order\_id')  buyer\_openid = data.get('buyer\_openid')   *# 检查参数是否缺失* if not order\_id or not buyer\_openid:  return JsonResponse({'message': '缺少参数: order\_id 或 buyer\_openid'}, status=400)   *# 尝试获取订单，捕获 404 错误* try:  order = get\_object\_or\_404(Order, order\_id=order\_id)  except Http404:  logger.error(f"Order with id {order\_id} not found.")  return JsonResponse({'message': '订单未找到'}, status=404)   logger.info(f"Found order: {order}")   *# 检查订单状态是否为 ONGOING* if order.order\_status != OrderStatus.ONGOING:  return JsonResponse({'message': '该订单处于不可收货状态'}, status=400)   *# 更新订单状态* order.order\_status = OrderStatus.RECEIVE  order.save()  logger.info(f"Order status updated: {order}")   return JsonResponse({  'order\_id': order.order\_id,  'message': '已收货'  })   except json.JSONDecodeError:  logger.error("JSON decoding error")  return JsonResponse({'message': '请求体格式错误'}, status=400)   except Exception as e:  logger.error(f"Error: {e}")  return JsonResponse({'message': str(e)}, status=500)  else:  return JsonResponse({'message': '仅支持 POST 请求'}, status=405) |

### 发布商品

|  |
| --- |
| from django.conf import settings from django.utils import timezone from datetime import datetime import json from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import logging from myapp.models import Product, User, ProductStatus, Category1, Order  logger = logging.getLogger(\_\_name\_\_)  @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def publish\_product(request):  if request.method == 'POST':  try:  data = json.loads(request.body) *# 解析请求体中的JSON数据* openid = data.get('openid') *# 获取 openid   # 获取前端传递的商品信息* category1 = data.get('category1')  category2 = data.get('category2')  name = data.get('name')  price = data.get('price')  description = data.get('description')  phone = data.get('phone')  images = data.get('images')  address = data.get('address')  provide\_service = data.get('provideService', False) *# 默认值 False* rental\_period = data.get('rentalPeriod', '0')  auction\_end\_date = data.get('auctionEndDate') *# 获取拍卖结束日期* auction\_end\_time = data.get('auctionEndTime') *# 获取拍卖结束时间* if auction\_end\_date and auction\_end\_time:  auction\_end\_datetime\_str = f"{auction\_end\_date} {auction\_end\_time}" *# 格式： "YYYY-MM-DD HH:MM"* try:  auction\_end\_datetime\_naive = datetime.strptime(auction\_end\_datetime\_str, "%Y-%m-%d %H:%M")   *# 转换为带时区的时间，使用当前时区（例如，Django的默认时区）* auction\_end\_datetime = timezone.make\_aware(auction\_end\_datetime\_naive,  timezone.get\_current\_timezone())   *# 将时区感知的时间转换为本地时间* auction\_end\_datetime\_local = timezone.localtime(auction\_end\_datetime)   *# 如果 USE\_TZ = False，将带时区的时间转换为朴素的时间* if not settings.USE\_TZ:  auction\_end\_datetime\_local = auction\_end\_datetime\_local.replace(tzinfo=None)   except ValueError:  return JsonResponse({"success": False, "message": "无效的日期或时间格式"}, status=400)  else:  auction\_end\_datetime\_local = None *# 如果没有提供拍卖结束日期和时间   # 日志：查看接收到的数据* logger.info(f"Received data: {data}")   *# 检查必填项是否完整* if not name or not price or not description or not phone or not address or not images:  return JsonResponse({'success': False, 'message': '请提供完整的商品信息'})   *# 类型转换和验证* try:  price = float(price) *# 确保价格是浮动类型* except ValueError:  return JsonResponse({'success': False, 'message': '价格必须是数字'})   *# 检查图片数组* if not isinstance(images, list) or not all(isinstance(image, str) for image in images):  return JsonResponse({'success': False, 'message': '图片必须是一个包含图片路径的字符串列表'})   *# 验证租赁周期（rentalPeriod）字段* if category1 == '租赁':  if rental\_period not in ['/天', '/月']:  return JsonResponse({'success': False, 'message': '无效的租赁周期，必须为 "天" 或 "月"'})  else:  rental\_period = '0' *# 非租赁商品，设置租赁周期为 0   # 获取对应的用户* user = User.objects.filter(openid=openid).first()   if not user:  return JsonResponse({'success': False, 'message': '用户未登录或无效的 openid'})   *# 日志：打印即将传入 create 的数据* logger.info(  f"Creating product with values: category1={category1}, category2={category2}, name={name}, price={price}, description={description}, phone={phone}, address={address}, images={images}, provide\_service={provide\_service}")   *# 创建商品记录* product = Product.objects.create(  user=user, *# 关联用户* category1=category1,  category2=category2,  name=name,  price=price,  description=description,  phone=phone,  address=address,  images=images,  provide\_service=provide\_service, *# 使用 provideService 字段* rental\_period=rental\_period,  auction\_end\_time=auction\_end\_datetime\_local, *# 存入本地时区的时间* product\_status=ProductStatus.LIST, *# 假设商品状态是 LIST* ) |

### 拍卖系统

|  |
| --- |
| from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt from myapp.models import Order, Product import json  @csrf\_exempt def bid(request):  if request.method == 'POST':  try:  *# 解析请求数据* data = json.loads(request.body)  print("请求数据:", data) *# 打印请求体，便于调试* buyer\_openid = data.get('buyer\_openid')  product\_id = data.get('product\_id')  bid\_price = data.get('bid\_price')   *# 检查参数是否有效* if not buyer\_openid or not product\_id or not bid\_price:  return JsonResponse({'success': False, 'message': '缺少必要的参数'}, status=400)   *# 查找商品* product = Product.objects.filter(product\_id=product\_id).first()  print(f"查询到的商品: {product}") *# 打印商品信息，便于调试* if not product:  return JsonResponse({'success': False, 'message': '商品不存在'}, status=404)   *# 查询与该商品关联的所有订单，并找出当前正在进行的订单* order = Order.objects.filter(product\_id=product\_id, order\_status='交易中').first()  print(f"商品 {product.name} 关联的订单: {order}") *# 打印查询到的订单信息，便于调试* if not order:  return JsonResponse({'success': False, 'message': '商品尚未拍卖'}, status=400)   *# 检查是否为同一买家出价* if order.buyer\_openid == buyer\_openid:  return JsonResponse({'success': False, 'message': '您已经是当前的买家，不能再次出价'}, status=400)   *# 检查用户出价是否有效（出价必须高于当前最高价）* if float(bid\_price) <= float(order.transaction\_price):  return JsonResponse({'success': False, 'message': '出价必须高于当前最高价'}, status=400)   *# 更新订单的价格和买家* order.transaction\_price = float(bid\_price)  order.buyer\_openid = buyer\_openid  order.save()   return JsonResponse({'success': True, 'message': '出价成功', 'new\_transaction\_price': bid\_price})   except json.JSONDecodeError:  return JsonResponse({'success': False, 'message': '请求体格式错误'}, status=400)   except Exception as e:  return JsonResponse({'success': False, 'message': f'出价失败: {str(e)}'}, status=500)   return JsonResponse({'success': False, 'message': '仅支持POST请求'}, status=405) |

### 拍卖结束后锁定商品

|  |
| --- |
| from django.conf import settings from django.utils import timezone from datetime import datetime import json from django.http import JsonResponse from django.views.decorators.csrf import csrf\_exempt import logging from myapp.models import Product, ProductStatus  logger = logging.getLogger(\_\_name\_\_)   @csrf\_exempt *# 禁用CSRF验证（开发时使用，生产环境应启用并处理）* def check\_auction\_status(request):  if request.method == 'POST':  try:  *# 解析请求体中的JSON数据* data = json.loads(request.body)   *# 获取前端传递的商品 ID* product\_id = data.get('product\_id') *# 获取前端传递的商品 ID* if not product\_id:  return JsonResponse({"success": False, "message": "缺少商品ID参数"})   *# 获取传递的商品 ID* try:  product = Product.objects.get(product\_id=product\_id, category1='拍卖',  product\_status=ProductStatus.LIST)  except Product.DoesNotExist:  return JsonResponse({"success": False, "message": "商品未找到或不符合条件"})   *# 打印商品的拍卖结束时间，便于调试* logger.info(f"Checking product: {product.name}, Auction end time: {product.auction\_end\_time}")   *# 如果拍卖时间已过，将商品状态更新为锁定* product.product\_status = ProductStatus.LOCK  product.save()   logger.info(f"Product {product.name} (ID: {product.product\_id}) has been locked due to auction end time.")  return JsonResponse({"success": True, "message": f"商品 {product.name} 已被锁定"})   except Exception as e:  logger.error(f"Error in check\_auction\_status: {e}")  return JsonResponse({"success": False, "message": f"操作失败: {str(e)}"})   return JsonResponse({'success': False, 'message': '仅支持POST请求'}) |

## 配置文件示例

前端界面设计主要分为以下几个部分：

auction（拍卖商品界面），homecenter（个人主页）、publish（发布商品界面）、rental（租借商品界面）、secondhand（二手商品界面）、总界面（登录）

### 总界面（登陆页面）

|  |
| --- |
| // app.js  App({    onLaunch() {      // 展示本地存储能力      const logs = wx.getStorageSync('logs') || []      logs.unshift(Date.now())      wx.setStorageSync('logs', logs)      // 登录      wx.login({        success: res => {          // 发送 res.code 到后台换取 openId, sessionKey, unionId        }      })    },    globalData: {      userInfo: null,  // 用户信息      openid: '',      sessionKey: '',    },  }) |

**对上述代码进行分析：**

这段代码是一个微信小程序的登录页面逻辑。使用微信提供的API进行登录操作和本地存储操作。将 wx.login 方法中获取到的 code 发送到服务器，然后服务器会使用这个 code 去微信的服务器换取用户的 openId 和 sessionKey。

### 拍卖商品界面（以书籍子分类为例子）

由于拍卖商品分为“书籍”，“服装”，“电子”，“运动”，“其他”几个子分类，在此选取书籍子分类进行介绍。

### data对象

|  |
| --- |
| data: {      category1:'拍卖',      category2:'书籍',      secondhandGoods: [],  // 用来存储商品列表      productIds: [],       // 存储所有商品的 product\_id      imagesUrl:[],    } |

**对上述代码进行分析：**

Data对象定义了页面的初始数据，包括分类信息 category1 和 category2，商品列表 secondhandGoods，商品的 product\_id 列表 productIds，以及图片 URL 列表 imagesUrl。

### 页面加载方法

|  |
| --- |
| onLoad() {      // 获取分类信息      const category1 = this.data.category1;      const category2 = this.data.category2;      wx.request({        url: 'https://101972498yahu.vicp.fun/api/selectcategory/', // 后端接口URL        method: 'POST',  // 修改为POST方法        data: JSON.stringify({          category1: category1,          category2: category2        }),  // 传递 category1 和 category2        header: {          'Content-Type': 'application/json'        },        success: (res) => {          if (res.statusCode === 200) {            // 获取商品列表            const products = res.data.products;            // 遍历商品列表，为每个商品的 images 数组中的图片文件名拼接完整的 URL            products.forEach(product => {              // 获取图片文件名              const imageFileName = product.images[0];  // 获取第一个图片的文件名                // 拼接成完整的 URL              product.imageUrl = `https://101972498yahu.vicp.fun/media/uploads/${imageFileName}`;              // 如果你需要所有图片路径，遍历 product.images 数组              product.images = product.images.map(imageFileName => {                return `https://101972498yahu.vicp.fun/media/uploads/${imageFileName}`;              });            });            // 更新商品列表            this.setData({              secondhandGoods: products,            });            // 打印商品详情            console.log("商品列表:", products);          } else {            wx.showToast({              title: '加载商品失败',              icon: 'none'            });          }        },        fail: (err) => {          wx.showToast({            title: '请求失败，请稍后重试',            icon: 'none'          });        }      });    } |

**对上述代码进行分析：**

获取分类信息：从 data 对象中获取 category1 和 category2。

发起网络请求：使用 wx.request 方法向后端接口发送 POST 请求，请求分类信息。

请求参数：将 category1 和 category2 作为 JSON 字符串传递给后端。

请求头：设置 Content-Type 为 application/json。

成功回调：如果请求成功（statusCode 为 200），则处理返回的商品列表，包括拼接图片的完整 URL，并更新页面的 secondhandGoods 数据。

失败回调：如果请求失败，显示提示信息。

### goodListClickHandle 方法

|  |
| --- |
| goodListClickHandle(e) {    const productId = e.currentTarget.dataset.productId; // 获取点击商品的 product\_id    wx.navigateTo({       url: `/pages/auction/goods/goods?query=${encodeURIComponent(productId)}`,    });  } |

**对上述代码进行分析：**

获取 product\_id：从点击事件的当前目标中获取 data-productId 属性值。

跳转到商品详情页：使用 wx.navigateTo 方法跳转到商品详情页，并传递 product\_id 作为查询参数。

### 商品详情界面

|  |
| --- |
| data: {      goods: {},      buyer: {},      productPrice: '',      productId: '',      productName: '',      productPrice: '',      productImage: '',      goods\_img: [],      goods\_info: {},      remainingTime: '', // 存储剩余时间      highestPrice: '',      goods\_description: '',      provide\_service: true,      rental\_period: '',      auctionEndTime: '', // 拍卖结束时间      transactionPrice: '',      buyer\_openid: '',      timer: null, // 定时器    } |

**对上述代码进行分析：**

定义了页面的初始数据，包括商品信息、买家信息、产品价格、产品ID、产品名称、产品图片、商品描述、拍卖结束时间、当前最高价等。

### 加载商品详情界面方法

|  |
| --- |
| onLoad: function (options) {  const productId = options.query;  const openid = wx.getStorageSync('openid');  this.setData({  productId: productId,  }); |

**对上述代码进行分析：**

从一个名为options的对象中获取查询参数query，将其值赋给变量productId。然后，从本地存储中获取用户的唯一标识符openid。最后，它使用setData方法更新小程序页面的数据，将productId设置到页面的数据对象中。

|  |
| --- |
| // 获取商品信息      wx.request({        url: 'https://101972498yahu.vicp.fun/api/selectgoods/',        method: 'POST',        data: {          product\_id: productId,        },        success: (res) => {          if (res.statusCode === 200) {            const productData = res.data;            const fullImageUrls = productData.images.map(image => `https://101972498yahu.vicp.fun/media/uploads/${image}`);              this.setData({              goods\_info: productData,              productName: productData.name,              productPrice: productData.price,              goods\_img: fullImageUrls,              productImage: fullImageUrls[0] || '',              goods\_description: productData.description,              provide\_service: productData.provide\_service,              seller\_name: productData.seller\_name,            });              if (productData.rental\_period !== 0) {              this.setData({                rental\_period: productData.rental\_period,              });            }          } else {            wx.showToast({              title: '商品信息加载失败',              icon: 'none',            });          }        },        fail: (err) => {          wx.showToast({            title: '请求失败，请稍后再试',            icon: 'none',          });        }      }); |

**对上述代码进行分析：**

发送网络请求：使用微信小程序的wx.request方法，向服务器发送一个POST请求，目的是获取特定商品的详细信息。

携带商品ID：在请求的数据中包含一个product\_id，这个ID用于在服务器端查询对应的商品信息。

处理响应：

成功响应：如果服务器返回的状态码是200（表示请求成功），则处理返回的数据。

数据映射：将商品的图片信息从相对路径转换为完整的URL地址。

更新页面数据：使用this.setData方法更新小程序页面的数据，包括商品的名称、价格、图片、描述等信息。

特殊字段处理：如果商品有租赁期限，还会更新页面数据中的租赁期限字段。

错误处理：

请求失败：如果请求没有成功（状态码非200），则显示一个提示框告知用户商品信息加载失败。

网络错误：如果请求完全失败（如网络问题导致请求无法发出），则显示一个提示框告知用户请求失败，并建议稍后再试。

|  |
| --- |
| // 获取拍卖结束时间和当前最高价      const self = this;      wx.request({        url: 'https://101972498yahu.vicp.fun/api/selectauction/',        method: 'POST',        data: {          product\_id: self.data.productId        },        success(res) {          if (res.statusCode === 200) {            const data = res.data;            if (data && data.transaction\_price !== undefined && data.auction\_end\_time) {              const auctionEndTime = new Date(data.auction\_end\_time); // 拍卖结束时间              const transactionPrice = data.transaction\_price;  // 当前交易价格                self.setData({                transactionPrice: transactionPrice,                auctionEndTime: auctionEndTime,              });                // 判断拍卖是否结束              const currentTime = new Date();              if (currentTime >= auctionEndTime) {                // 拍卖已结束，向后端发送请求处理商品                self.sendAuctionEndRequest(productId);              } else {                // 启动倒计时                self.startCountdown(auctionEndTime);              }            } else {              wx.showToast({                title: '拍卖信息加载失败',                icon: 'none',              });            }          } else {            wx.showToast({              title: res.data.error || '获取拍卖信息失败',              icon: 'none',            });          }        },        fail(error) {          wx.showToast({            title: '请求失败，请稍后再试',            icon: 'none',          });        }      });    } |

**对上述代码进行分析：**

发送网络请求：使用微信小程序的wx.request方法，向服务器发送一个POST请求，目的是获取特定商品的拍卖信息，包括拍卖结束时间和当前最高价。

携带商品ID：在请求的数据中包含一个product\_id，这个ID用于在服务器端查询对应的拍卖信息。

处理响应：

成功响应：如果服务器返回的状态码是200（表示请求成功），则处理返回的数据。

数据解析：从响应中提取拍卖结束时间和当前交易价格。

更新页面数据：使用this.setData方法更新小程序页面的数据，包括拍卖结束时间和当前交易价格。

拍卖状态判断：

拍卖已结束：如果当前时间大于或等于拍卖结束时间，则调用sendAuctionEndRequest方法向后端发送请求处理商品。

拍卖未结束：如果拍卖未结束，则调用startCountdown方法启动倒计时。

错误处理：

数据错误：如果响应数据中缺少必要的字段（如transaction\_price或auction\_end\_time），则显示一个提示框告知用户拍卖信息加载失败。

状态码错误：如果服务器返回的状态码不是200，显示一个提示框告知用户获取拍卖信息失败，并显示服务器返回的错误信息。

网络错误：如果请求完全失败（如网络问题导致请求无法发出），则显示一个提示框告知用户请求失败，并建议稍后再试。

|  |
| --- |
| sendAuctionEndRequest(productId) {      wx.request({        url: 'https://101972498yahu.vicp.fun/api/checkauctionstatus/',        method: 'POST',        data: {          product\_id: productId,        },        success: (res) => {          if (res.statusCode === 200 && res.data.success) {            wx.showToast({              title: '拍卖已结束，商品已处理',              icon: 'none',            });            setTimeout(() => {              wx.reLaunch({                url: '/pages/auction/auction',              });            }, 1000);          } else {            wx.showToast({              title: res.data.message || '操作失败',              icon: 'none',            });          }        },        fail: (err) => {          wx.showToast({            title: '请求失败，请重试',            icon: 'none',          });        }      });    } |

**对上述代码进行分析：**

发送网络请求：使用wx.request方法，向服务器发送一个POST请求，目的是检查拍卖状态。

请求参数：请求的数据中包含product\_id，这是用于在服务器端查询拍卖状态的商品ID。

成功响应：如果服务器返回的状态码是200，并且响应数据中的success字段为true，则显示一个提示框告知用户拍卖已结束，商品已处理。

页面跳转：在显示提示后，使用setTimeout延迟1秒，然后使用wx.reLaunch方法重新启动小程序到指定的拍卖页面（/pages/auction/auction）。

|  |
| --- |
| startCountdown(auctionEndTime) {      const self = this;      // 定时器每秒更新剩余时间      this.data.timer = setInterval(() => {        const currentTime = new Date(); // 当前时间        const timeDifference = auctionEndTime - currentTime; // 剩余时间差        // 如果时间差小于或等于0，停止倒计时并设置状态        if (timeDifference <= 0) {          clearInterval(self.data.timer);          self.setData({            remainingTime: '拍卖已结束',          });        } else {          // 格式化倒计时          const hours = Math.floor(timeDifference / (1000 \* 60 \* 60));          const minutes = Math.floor((timeDifference % (1000 \* 60 \* 60)) / (1000 \* 60));          const seconds = Math.floor((timeDifference % (1000 \* 60)) / 1000);          self.setData({            remainingTime: `${hours}：${minutes}：${seconds}`, // 更新剩余时间          });        }      }, 1000);  // 每秒更新一次    } |

**对上述代码进行分析：**

设置定时器：使用setInterval方法设置一个定时器，每秒执行一次。

计算剩余时间：

获取当前时间currentTime。

计算拍卖结束时间auctionEndTime与当前时间currentTime之间的差值timeDifference。

倒计时结束：

如果timeDifference小于或等于0，说明拍卖已经结束。

使用clearInterval方法清除定时器。

使用this.setData方法更新页面数据，将剩余时间设置为“拍卖已结束”。

格式化倒计时：

如果timeDifference大于0，说明拍卖还在进行中。

计算剩余时间中的小时数、分钟数和秒数。

使用this.setData方法更新页面数据，将剩余时间格式化为“小时：分钟：秒”的形式。

每秒更新：定时器每秒执行一次，更新剩余时间。

|  |
| --- |
| bid: function () {      const openid = wx.getStorageSync('openid');      const currentHighestPrice = this.data.transactionPrice;      if (!openid) {        wx.showToast({          title: '未获取到用户信息，请登录后再试',          icon: 'none',        });        return;      }      wx.showModal({        title: '输入出价',        editable: true,        placeholderText: `当前最高价：${currentHighestPrice}，请输入更高的价格`,        success: (res) => {          if (res.confirm) {            const userBid = parseFloat(res.content);            if (isNaN(userBid) || userBid <= currentHighestPrice) {              wx.showToast({                title: '出价过低，请重新出价',                icon: 'none',              });            } else {              wx.request({                url: 'https://101972498yahu.vicp.fun/api/bid/',                method: 'POST',                data: {                  buyer\_openid: openid,                  bid\_price: userBid,                  product\_id: this.data.productId,                },                success: (response) => {                  if (response.data.success) {                    wx.showToast({                      title: '出价成功',                      icon: 'success',                    });                    this.setData({                      transactionPrice: userBid,                    });                  } else {                    wx.showToast({                      title: response.data.message || '出价失败，请重试',                      icon: 'none',                    });                  }                },                fail: () => {                  wx.showToast({                    title: '网络错误，请稍后再试',                    icon: 'none',                  });                },              });            }          }        },      });    }, |

**对上述代码进行分析：**

获取用户信息：

使用wx.getStorageSync('openid')获取存储在本地的openid，这是一个标识用户的唯一标识符。

检查用户信息：

如果没有获取到openid，则显示一个提示框告知用户未获取到用户信息，并要求用户登录后再试。

显示输入框：

使用wx.showModal显示一个模态对话框，让用户输入出价。

editable设置为true，允许用户编辑输入框的内容。

placeholderText显示当前最高价，并提示用户输入更高的价格。

处理用户输入：

如果用户点击确认按钮（res.confirm为true），则获取用户输入的内容，并尝试将其转换为浮点数。

如果转换结果不是数字或者用户出价不高于当前最高价，则显示提示框告知用户出价过低。

发送出价请求：

如果用户出价有效，则使用wx.request向服务器发送POST请求，包含买家的openid、出价金额和商品ID。

请求成功后，根据服务器的响应更新页面数据，并显示出价成功或失败的提示。

|  |
| --- |
| collect() {      const openid = wx.getStorageSync('openid');      const product\_id = this.data.productId;      wx.request({        url: 'https://101972498yahu.vicp.fun/api/addtofavorites/',        method: 'POST',        data: {          openid: openid,          product\_id: product\_id        },        success: (res) => {          if (res.data.success) {            this.setData({              isFavorited: true            });            wx.showToast({              title: '收藏成功',              icon: 'success'            });          } else {            wx.showToast({              title: res.data.message || '操作失败',              icon: 'none'            });          }        },        fail: (err) => {          wx.showToast({            title: '请求失败，请重试',            icon: 'none'          });        }      });    }, |

**对上述代码进行分析：**

获取用户信息：

使用wx.getStorageSync('openid')从本地存储中获取用户的openid，这是一个标识用户的唯一标识符。

获取商品ID：

从页面的data对象中获取productId，这是要收藏的商品的ID。

发送收藏请求：

使用wx.request向服务器发送一个POST请求，请求的URL是https://101972498yahu.vicp.fun/api/addtofavorites/，请求的数据中包含用户的openid和商品的product\_id。

处理响应：

如果服务器响应成功（res.data.success为true），则更新页面数据，将isFavorited设置为true，表示商品已收藏，并显示一个提示框告知用户收藏成功。

如果服务器响应失败，则显示一个提示框告知用户操作失败，并显示服务器返回的错误消息。

### 搜索商品方法

|  |
| --- |
| Search() {      const { searchQuery, history } = this.data;      if (searchQuery && !history.includes(searchQuery)) {        // 添加新搜索词到历史记录        const newHistory = [searchQuery, ...history].slice(0, 10); // 保留最近10条历史        this.setData({          history: newHistory,        });        wx.setStorageSync('searchHistory', newHistory);  // 将历史记录存储到本地      }      // 跳转到搜索结果页面并传递搜索关键字      wx.navigateTo({        url: `/pages/auction/search/searchresults/searchresults?query=${encodeURIComponent(searchQuery)}`,      });    } |

**对上述代码进行分析：**

这段代码的主要作用是：管理用户的搜索历史记录，确保历史记录中只包含最新的10条记录。将用户的搜索请求传递给搜索结果页面，以便显示相关的搜索结果。

|  |
| --- |
| onSelectHistory(e) {      const selectedQuery = e.currentTarget.dataset.query;      this.setData({        searchQuery: selectedQuery,      });       // 跳转到搜索结果页面并传递历史记录的搜索关键字       wx.navigateTo({        url: `/pages/auction/search/searchresults/searchresults?query=${encodeURIComponent(selectedQuery)}`,      });    } |

**对上述代码进行分析：**

当用户从搜索历史中选择一个历史记录项时，自动填充搜索框，并跳转到搜索结果页面，显示与该历史记录项相关的搜索结果。

|  |
| --- |
| wx.request({        url: 'https://101972498yahu.vicp.fun/api/auctionsearch/',  // 后端接口URL        method: 'POST',        data: { keyword: query },  // 传递查询的关键字        success: (res) => {          if (res.statusCode === 200) {            // 获取返回的商品列表            const products = res.data.products || [];            // 遍历商品列表，为每个商品的 images 数组中的图片文件名拼接完整的 URL            products.forEach(product => {              // 获取图片文件名              const imageFileName = product.images[0];  // 获取第一个图片的文件名                // 拼接成完整的 URL              product.imageUrl = `https://101972498yahu.vicp.fun/media/uploads/${imageFileName}`;              // 如果你需要所有图片路径，遍历 product.images 数组              product.images = product.images.map(imageFileName => {                return `https://101972498yahu.vicp.fun/media/uploads/${imageFileName}`;              });            });            // 更新商品列表            this.setData({              results: products,  // 存储搜索结果            });            // 打印搜索结果            console.log("搜索结果:", products);          } else {            wx.showToast({              title: '没有找到相关商品',              icon: 'none',            });            this.setData({              results: [],            });          }        },        fail: (err) => {          console.error('搜索失败:', err);          wx.showToast({            title: '请求失败，请稍后重试',            icon: 'none',          });          this.setData({            results: [],          });        },      }) |

**对上述代码进行分析：**

发送网络请求：使用wx.request方法，向服务器发送一个POST请求，请求URl，用于执行拍卖搜索。

传递查询关键字：在请求的数据中传递keyword，这是用户输入的搜索关键字。

### 租借界面

由于租借界面和前文拍卖界面大体相似，仅介绍特殊部分。

### 租借商品

|  |
| --- |
| // 处理租期输入     onRentalPeriodInput: function (e) {      const rentalPeriod = e.detail.value;  // 获取用户输入的租期      this.setData({        rentalnumber: rentalPeriod  // 更新 data 中的租期      });    } |

**对上述代码进行分析：**

租赁商品主要特殊处在于获取租期，计算租金。

### 二手交易商品界面

同理，二手交易界面与租赁，拍卖商品类似，仅介绍特殊部分。

|  |
| --- |
| data: {      category1:'二手交易',      category2:'书籍',      secondhandGoods: [],  // 用来存储商品列表      productIds: [],       // 存储所有商品的 product\_id      imagesUrl:[],} |

**对上述代码进行分析：**

二手交易商品为基础模块，拍卖和租赁都是在其基础上增添功能。

### 商品发布界面

|  |
| --- |
| data: {      categories1: ['二手交易', '拍卖', '租赁'],  // 商品类别      selectedCategory1: '二手交易',  // 默认选择类别      categories2: ['电子产品', '服装', '书籍', '运动', '其他'],      selectedCategory2: '电子产品',      name: '',  // 商品名称      price: '',  // 商品价格      description: '',  // 商品介绍      phone: '',  // 联系电话      address: '',  // 卖家地址      images: [],    // 存储已选择的图片      uploadImageSrc: '/images/uploadimages.png',      provideService: true,      rentalPeriod: ['/天', '/月'],      selectedrentalperiod: '/天', // 默认值      nickName: '',  // 用户昵称      avatarUrl: '',  // 用户头像      openid: '',  // 存储 openid      auctionEndTime: '',  // 拍卖结束时间      product\_id:'',    } |

**对上述代码进行分析：**

定义了一个微信小程序页面的 data 对象，它包含了页面所需的初始数据和状态。用于商品发布或编辑页面，允许用户输入商品信息、选择类别、上传图片等。

|  |
| --- |
| // 选择商品类别    onCategoryChange1(e) {      const selectedIndex1 = e.detail.value;      this.setData({        selectedCategory1: this.data.categories1[selectedIndex1]      });      // 如果选择的是拍卖类别，显示时间选择框      if (this.data.categories1[selectedIndex1] === '拍卖') {        this.setData({          auctionStartTime: '',  // 清空拍卖开始时间          auctionEndTime: '',    // 清空拍卖结束时间        });      }    }  // 选择拍卖结束日期  onDateChange: function (e) {    const date = e.detail.value;    this.setData({      auctionEndDate: date,  // 更新拍卖结束日期    });  },  // 选择拍卖结束时间  onTimeChange: function (e) {    const time = e.detail.value;    this.setData({      auctionEndTime: time,  // 更新拍卖结束时间    });  } |

**对上述代码进行分析：**

当用户在类别选择器中选择一个商品类别时，更新页面数据以反映用户的选择。特别地，如果用户选择了“拍卖”类别，清空拍卖时间的选择，以便用户重新输入，同时提供了填写拍卖结束时间的函数。

|  |
| --- |
| // 商品名称输入处理  onNameChange(e) {    let name = e.detail.value;    if (name.length > 10) {      name = name.substring(0, 10);  // 截取前 10 个字符      wx.showToast({        title: '商品名称不能超过10个字',        icon: 'none',      });    }    this.setData({      name: name  // 更新商品名称，自动删除超出的部分    });  }  // 商品价格输入处理    onPriceChange(e) {      this.setData({        price: e.detail.value      });    },    // 商品介绍输入处理    onDescriptionChange(e) {      this.setData({        description: e.detail.value      });    },    // 联系电话    onPhoneChange(e) {      const phone = e.detail.value;      // 仅保留数字      const phoneNumber = phone.replace(/\D/g, '');      this.setData({        phone: phoneNumber      });      // 检查是否为 11 位      if (phoneNumber.length > 11) {        wx.showToast({          title: '联系电话不能超过 11 位',          icon: 'none',        });        this.setData({          phone: phoneNumber.slice(0, 11)  // 限制为 11 位        });      }    },    // 选择是否提供上门服务    onServiceChange(e) {      const provideService = e.detail.value.includes('provideService');      this.setData({        provideService: provideService      });    }, |

**对上述代码进行分析：**该函数支持用户输入商品名称，价格，介绍以及是否上门

|  |
| --- |
| // 上传图片    chooseImage() {      const currentImages = this.data.images;      if (currentImages.length >= 3) {        wx.showToast({          title: '最多只能上传 3 张图片',          icon: 'none',        });        return; // 如果已上传 3 张图片，停止选择      }      wx.chooseMedia({        count: 3 - currentImages.length,  // 根据已上传的图片数量限制选择数量        mediaType: ['image'],  // 只选择图片        sourceType: ['album', 'camera'],  // 可从相册或相机选择        success: (res) => {          const newImages = res.tempFiles.map(file => file.tempFilePath);          this.setData({            images: [...this.data.images, ...newImages],  // 将新选中的图片添加到 images 数组中            uploadImageSrc: newImages[0],  // 更新上传按钮为选中的图片          });        },        fail: (err) => {          console.error('选择图片失败', err);        }      });    },    // 删除图片    deleteImage(e) {      const index = e.currentTarget.dataset.index; // 获取图片索引      const images = this.data.images;      images.splice(index, 1);  // 从数组中删除该图片      this.setData({        images: images,  // 更新图片数组        uploadImageSrc: images.length === 0 ? '/images/upload-image.png' : this.data.uploadImageSrc,  // 如果没有图片，则恢复默认按钮图片      });    }, |

**对上述代码进行分析：**支持图片的相关操作

### 用户中心界面

### 用户中心登录界面

|  |
| --- |
| // 根据头像和昵称是否存在，显示不同的按钮      if (nickName && avatarUrl) {        // 如果头像和昵称都存在，则显示头像和退出登录按钮        this.setData({          showLogoutButton: true,          showLoginButton: false        });      } else {        // 如果头像和昵称不存在，显示授权登录按钮        this.setData({          showLogoutButton: false,          showLoginButton: false        });      } |

**对上述代码进行分析：**用户登录逻辑

|  |
| --- |
| // 确认登录    confirmLogin() {      wx.login({        success: (res) => {          if (res.code) {            // 发起网络请求，将登录code发送给后端            wx.request({              url: 'https://101972498yahu.vicp.fun/api/login/',  // 后端接口地址              method: 'POST',              data: {                code: res.code,  // 发送登录code                nickName: this.data.nickName,  // 发送用户昵称                avatarUrl: this.data.avatarUrl,  // 发送用户头像              },              success: (response) => {                if (response.data.success) {                  const { openid, session\_key } = response.data;                  wx.setStorageSync('openid', openid);                  wx.setStorageSync('sessionKey', session\_key);                  wx.setStorageSync('nickName', this.data.nickName);                  wx.setStorageSync('avatarUrl', this.data.avatarUrl);                  wx.setStorageSync('userInfo', {                    nickName: this.data.nickName,                    avatarUrl: this.data.avatarUrl,                    openid: openid,                    session\_key: session\_key,                  });                  wx.showToast({                    title: '登录成功',                    icon: 'success',                  });                  this.setData({                    showLoginButton: false,  // 登录成功后隐藏按钮                    showLogoutButton: true,  // 显示退出登录按钮                  });                } else {                  wx.showToast({                    title: response.data.message || '登录失败，请重试',                    icon: 'none',                  });                }              },              fail: (err) => {                wx.showToast({                  title: '网络请求失败，请稍后再试',                  icon: 'none',                });                console.error(err);              }            });          } else {            console.log('登录失败！' + res.errMsg);          }        }      });    } |

**对上述代码进行分析：**

微信登录：使用wx.login方法调用微信的登录功能，成功登录后会得到一个登录凭证code。

检查登录凭证：如果成功获取到code，则使用这个code以及其他用户信息（如昵称和头像）发起网络请求，将这些信息发送给后端服务器进行验证。

|  |
| --- |
| // 退出登录    logout() {      wx.removeStorageSync('userInfo'); // 删除本地存储的用户信息      wx.removeStorageSync('sessionKey');      wx.removeStorageSync('nickName');      wx.removeStorageSync('openid');      wx.removeStorageSync('avatarUrl');      this.setData({        nickName: '',        avatarUrl: '',        showLoginButton: false,  // 登录按钮可见        showLogoutButton: false,  // 退出登录按钮不可见      });      wx.showToast({        title: '已退出登录',        icon: 'success',      });    } |

**对上述代码进行分析：**退出登录功能操作

### 浏览历史记录界面

|  |
| --- |
| wx.request({        url: `${BASE\_URL}/api/gethistory/`,        method: 'POST',        data: {          openid: openid,          page: this.data.page,        },        success: (res) => {          console.log(res.data);          if (res.data && res.data.history) {            const newHistory = res.data.history.map((record) => {              // 拼接图片 URL 如果需要              record.images = record.images.map((imageFileName) => {                if (imageFileName.startsWith('/media/')) {                  imageFileName = imageFileName.replace('/media/', '');                }                return `${BASE\_URL}/media/uploads/${imageFileName}`;              });              return record;            });              this.setData({              history: [...this.data.history, ...newHistory], // 追加历史记录              page: this.data.page + 1,              noMoreData: newHistory.length === 0, // 判断是否还有更多数据            });          } else {            wx.showToast({              title: '获取历史记录失败',              icon: 'none',            });          }        },        fail: (err) => {          console.error(err);          wx.showToast({            title: '网络错误，请稍后再试',            icon: 'none',          });        },        complete: () => {          this.setData({ loading: false });        },      }); |

**对上述代码进行分析：**分页获取用户的历史记录，并处理每条记录中的图片URL。更新页面数据以显示新获取的历史记录。给用户反馈，告知他们历史记录获取的状态。

|  |
| --- |
| // 清空历史浏览记录    clearHistory() {      const openid = this.getOpenId();      if (!openid) {        wx.showToast({          title: '请先登录',          icon: 'none',        });        return;      }      wx.request({        url: 'https://101972498yahu.vicp.fun/api/clearhistory/',        method: 'POST',        data: {          openid: openid,        },        success: (res) => {          if (res.data.success) {            wx.showToast({              title: '历史记录已清空',              icon: 'success',            });            this.setData({ history: [], page: 1, noMoreData: false });          } else {            wx.showToast({              title: '清空失败',              icon: 'none',            });          }        },        fail: (err) => {          console.error(err);          wx.showToast({            title: '网络错误，请稍后再试',            icon: 'none',          });        },      });    } |

**对上述代码进行分析：**用于清空用户的浏览历史记录

### 商品收藏界面

|  |
| --- |
| // 获取收藏夹内容    fetchFavorites() {      if (this.data.noMoreData || this.data.loading) return;      wx.getStorage({        key: 'openid',        success: (res) => {          const openid = res.data;          const BASE\_URL = "https://101972498yahu.vicp.fun"; // 基础 URL          this.setData({ loading: true });          wx.request({            url: `${BASE\_URL}/api/getfavorites/`,            method: 'POST',            data: { openid },            success: (res) => {              if (res.data) {                const newFavorites = res.data.favorites.map((item) => {                  const fixedImages = item.images.map((imageFileName) => {                    if (imageFileName.startsWith('/media/')) {                      imageFileName = imageFileName.replace('/media/', '');                    }                    return `${BASE\_URL}/media/uploads/${imageFileName}`;                  });                  return {                    ...item,                    images: fixedImages,                    imageUrl: fixedImages[0],                  };                });                const noMoreData = newFavorites.length === 0;                this.setData({                  favorites: [...this.data.favorites, ...newFavorites],                  page: this.data.page + 1,                  noMoreData,                });              } else {                wx.showToast({ title: '获取收藏夹失败', icon: 'none' });              }            },            fail: () => wx.showToast({ title: '网络错误', icon: 'none' }),            complete: () => this.setData({ loading: false }),          });        },        fail: () => wx.showToast({ title: '获取 openid 失败', icon: 'none' }),      });      } |

**对上述代码进行分析：**用于获取用户的收藏夹内容

|  |
| --- |
| // 移除单个订单    removeFavorite(e) {      const { id } = e.currentTarget.dataset;      wx.getStorage({        key: 'openid',        success: (res) => {          const openid = res.data;          wx.request({            url: 'https://101972498yahu.vicp.fun/api/removefromfavorites/',            method: 'POST',            data: { openid, product\_id: id },            success: (res) => {              if (res.data) {                this.setData({                  favorites: this.data.favorites.filter((fav) => fav.product\_id !== id),                });                wx.showToast({ title: '移除成功', icon: 'success' });              } else {                wx.showToast({ title: '移除失败', icon: 'none' });              }            },          });        },        fail: () => wx.showToast({ title: '获取 openid 失败', icon: 'none' }),      });    } |

**对上述代码进行分析：**移除收藏商品：从用户的收藏夹中移除一个特定的商品。获取用户标识：使用本地存储中的openid来标识用户。发送网络请求：向服务器发送请求，请求从收藏夹中移除指定的商品。更新界面：根据服务器的响应，更新小程序界面上的收藏列表，并给出相应的提示信息。

|  |
| --- |
| // 清空收藏夹    clearAllFavorites() {      wx.getStorage({        key: 'openid',        success: (res) => {          const openid = res.data;          wx.showModal({            title: '提示',            content: '确定清空收藏夹吗？',            success: (modalRes) => {              if (modalRes.confirm) {                wx.request({                  url: 'https://101972498yahu.vicp.fun/api/clearfavorites/',                  method: 'POST',                  data: { openid },                  success: (res) => {                    if (res.data) {                      this.setData({ favorites: [] });                      wx.showToast({ title: '清空成功', icon: 'success' });                    } else {                      wx.showToast({ title: '清空失败', icon: 'none' });                    }                  },                });              }            },          });        },        fail: () => wx.showToast({ title: '获取 openid 失败', icon: 'none' }),      });    } |

**对上述代码进行分析：**获取用户OpenID：从本地存储中获取用户的openid，用于识别用户。确认操作：通过wx.showModal显示一个模态对话框，询问用户是否确定要清空收藏夹。

### 订单界面

|  |
| --- |
| // 请求买家和卖家订单（合并到一个订单数组）    fetchOrders() {      if (this.data.isLoading || !this.data.hasMore) return;      this.setData({ isLoading: true }); // 设置加载中      // 请求卖家订单      wx.request({        url: `${BASE\_URL}/api/getsellerorder/`,        method: "POST",        data: {          openid: this.data.buyerOpenid,          category1: this.data.category1, // 当前选择的商品类别          page: this.data.page, // 页码        },        success: (res) => {          if (res.statusCode === 200 && res.data.orders) {            const sellerOrders = res.data.orders.map((order) => {              // 处理商品图片链接              order.product\_images = order.product\_images.map((imageFileName) => {                if (imageFileName.startsWith("/media/")) {                  imageFileName = imageFileName.replace("/media/", "");                }                return `${BASE\_URL}/media/uploads/${imageFileName}`;              });              return order;            });            // 合并卖家订单到 orders 数组            this.setData({              orders: [...this.data.orders, ...sellerOrders], // 追加卖家订单            });          }        },        fail: () => {          wx.showToast({ title: '网络请求失败', icon: 'none' });        },      });      // 请求买家订单      wx.request({        url: `${BASE\_URL}/api/getorder/`,        method: "POST",        data: {          openid: this.data.buyerOpenid,          category1: this.data.category1, // 当前选择的商品类别          page: this.data.page, // 页码        },        success: (res) => {          if (res.statusCode === 200 && res.data.orders) {            const buyerOrders = res.data.orders.map((order) => {              // 处理商品图片链接              order.product\_images = order.product\_images.map((imageFileName) => {                if (imageFileName.startsWith("/media/")) {                  imageFileName = imageFileName.replace("/media/", "");                }                return `${BASE\_URL}/media/uploads/${imageFileName}`;              });              return order;            });            // 合并买家订单到 orders 数组            this.setData({              orders: [...this.data.orders, ...buyerOrders], // 追加买家订单              page: this.data.page + 1, // 翻到下一页              hasMore: buyerOrders.length > 0, // 判断是否还有更多订单            });          } else {            wx.showToast({ title: '没有更多订单', icon: 'none' });          }        },        fail: () => {          wx.showToast({ title: '网络请求失败', icon: 'none' });        },        complete: () => {          this.setData({ isLoading: false }); // 请求完成，停止加载状态        }      });    } |

**对上述代码进行分析：**同时请求买家和卖家的订单，并将它们合并到一个订单数组中。

|  |
| --- |
| // 取消订单    cancelOrder(e) {      const orderId = e.currentTarget.dataset.id;      wx.request({        url: `${BASE\_URL}/api/cancelorder/`,        method: "POST",        data: {          order\_id: orderId,          buyer\_openid: this.data.buyerOpenid        },        success: (res) => {          if (res.data) {            wx.showToast({ title: "订单已取消", icon: "success" });            setTimeout(() => {              wx.reLaunch({                url: 'pages/homecenter/homecenter',              });            }, 1000);          }        }      });    } |

**对上述代码进行分析：**取消订单

|  |
| --- |
| // 确认收货    confirmReceipt(e) {      const orderId = e.currentTarget.dataset.id;      wx.request({        url: `${BASE\_URL}/api/getgoods/`,        method: "POST",        data: {          order\_id: orderId,          buyer\_openid: this.data.buyerOpenid        },        success: (res) => {          if (res.statusCode === 200) {            wx.showToast({              title: res.data.message,  // 显示后端返回的信息              icon: "success"            });            this.fetchOrders(); // 重新加载订单          } else {            wx.showToast({              title: res.data.message,              icon: 'none'            });          }        },        fail: () => {          wx.showToast({            title: '网络请求失败',            icon: 'none'          });        }      });    } |

**对上述代码进行分析：**确认收货

|  |
| --- |
| // 删除商品（卖家）    deleteProduct(e) {      const productId = e.currentTarget.dataset.id;      wx.request({        url: `${BASE\_URL}/api/deleteproduct/`,        method: "POST",        data: {          product\_id: productId        },        success: (res) => {          if (res.data) {            wx.showToast({ title: "商品已下架", icon: "success" });            this.fetchOrders(); // 重新加载订单            setTimeout(() => {              wx.reLaunch({                url: '/pages/homecenter/homecenter',              });            }, 1000);          }        }      });    } |

**对上述代码进行分析：**删除商品

|  |
| --- |
| // 完成交易（卖家）    finishOrder(e) {      const orderId = e.currentTarget.dataset.id;      wx.request({        url: `${BASE\_URL}/api/finishorder/`,        method: "POST",        data: {          order\_id: orderId        },        success: (res) => {          if (res.data) {            wx.showToast({ title: "交易已完成", icon: "success" });            wx.reLaunch({              url: 'pages/homecenter/ordersbrowsing/ordersbrowsing',            });          }        }      });    } |

**对上述代码进行分析：**完成交易

### 个人信息界面

|  |
| --- |
| // 输入框绑定的值改变    handleInputChange(e) {      const { field } = e.currentTarget.dataset;      this.setData({        [field]: e.detail.value,      });    }  // 保存修改    saveChanges() {      const { wechatid, username, phone, email, address } = this.data;        // 分别存储变量      wx.setStorage({        key: 'wechatid',        data: wechatid,        success: () => console.log('Wechat ID 保存成功'),        fail: (err) => console.error('Wechat ID 保存失败:', err),      });        wx.setStorage({        key: 'username',        data: username,        success: () => console.log('Username 保存成功'),        fail: (err) => console.error('Username 保存失败:', err),      });        wx.setStorage({        key: 'phone',        data: phone,        success: () => console.log('Phone 保存成功'),        fail: (err) => console.error('Phone 保存失败:', err),      });        wx.setStorage({        key: 'email',        data: email,        success: () => console.log('Email 保存成功'),        fail: (err) => console.error('Email 保存失败:', err),      });        wx.setStorage({        key: 'address',        data: address,        success: () => {          // 更新 hasAddress 状态          this.setData({            hasAddress: !!address, // 如果 address 存在，hasAddress 为 true，否则为 false          });            wx.showToast({            title: '保存成功',            icon: 'success',          });        },        fail: (err) => {          wx.showToast({            title: '保存失败，请重试',            icon: 'none',          });          console.error('Address 保存失败:', err);        },      });        // 发送 POST 请求保存到服务器      wx.getStorage({        key: 'openid',        success: (res) => {          const openid = res.data;          wx.request({            url: 'https://101972498yahu.vicp.fun/api/changeuserdata/',            method: 'POST',            data: {              openid: openid,              username: username,              email: email,              phone: phone,              address: address,              wechat\_id: wechatid,            },            success: (res) => {              if (res.data.success) {                wx.showToast({                  title: '服务器保存成功',                  icon: 'success',                });              } else {                wx.showToast({                  title: '服务器保存失败，请重试',                  icon: 'none',                });              }            },            fail: (err) => {              wx.showToast({                title: '网络请求失败',                icon: 'none',              });              console.error('服务器请求错误:', err);            },          });        },        fail: () => {          wx.showToast({            title: '获取 openid 失败',            icon: 'none',          });        },      });    } |

**对上述代码进行分析：**个人信息修改及保存

|  |
| --- |
| onLoad(options) {      // 分别从本地存储加载用户信息      wx.getStorage({        key: 'wechatid',        success: (res) => {          this.setData({ wechatid: res.data });        },        fail: () => console.log('未找到存储的 Wechat ID'),      });      wx.getStorage({        key: 'username',        success: (res) => {          this.setData({ username: res.data });        },        fail: () => console.log('未找到存储的 Username'),      });      wx.getStorage({        key: 'phone',        success: (res) => {          this.setData({ phone: res.data });        },        fail: () => console.log('未找到存储的 Phone'),      });      wx.getStorage({        key: 'email',        success: (res) => {          this.setData({ email: res.data });        },        fail: () => console.log('未找到存储的 Email'),      });      wx.getStorage({        key: 'address',        success: (res) => {          this.setData({            address: res.data,            hasAddress: !!res.data, // 判断是否有地址          });        },        fail: () => console.log('未找到存储的 Address'),      });    } |

**对上述代码进行分析：**上传个人信息