上海大学

计算机学院

期末大作业报告

课程名称： Web **开发技术**

学期： 2026 **年春季**

课题名称：_____

学号：_____

姓名：_____

成绩：_____

任课教师签名：_____

目 录

第一章　系统概要设计

1　系统总体架构设计

系统已经完成服务器部署，并以正式域名提供访问，具备实际运行环境。当前站点通过 shopping.vertexf.top 对外提供访问，运行于 Ubuntu 22.04 服务器，由 Nginx 反向代理 Gunicorn 托管的 Flask 应用。

本系统在总体结构上采用“前端展示层—应用服务层—业务数据层—静态资源层—后台任务层”的分层组织方式。围绕当前电商网站的实际业务链路形成：普通用户需要完成商品浏览、购物车、下单与支付；商家需要持续维护商品、处理订单并经营店铺；管理员需要承担审核、站点配置和平台治理职责。系统首先确立了以职责边界为基础的分层结构，并在此基础上组织各功能模块之间的协同关系。

前端展示层承担用户可见页面的组织与交互反馈职责，主要通过 Flask 配合 Jinja2 模板完成页面渲染。系统功能模块之间存在大量与登录态、角色状态、订单状态直接相关的动态页面逻辑，模板渲染能够更直接地完成服务端数据到页面的绑定。前端层最终形成了首页、商品列表、商品详情、购物车、订单页、钱包页、地址管理页、商家后台页和管理员后台页等页面集合。

应用服务层是系统的核心层，由 Flask 应用主程序承担主要业务处理职责。该层负责路由分发、登录态恢复、角色判断、表单处理、订单状态流转、上传校验、错误响应等功能，是前端展示层与业务数据层之间的中介。将大部分业务规则收敛在应用服务层，有助于保证不同页面在访问同一业务对象时遵循一致的规则。例如，商品库存校验、订单支付前状态判断、商家发货条件控制、管理员审核通过后的商家与店铺创建等逻辑，均由该层统一组织，从而避免将业务规则散落到模板或数据库脚本中。

业务数据层采用 SQL Server 作为核心业务数据库，负责保存用户、角色、商家、店铺、商品、购物车、订单、地址、钱包与交易流水等结构化数据。该层的设计目标不是单纯“存数据”，而是通过清晰的实体划分和关联关系，为上层业务提供稳定的数据支撑。系统将交易过程中的主要对象拆分为多个主题表，例如订单主表、订单明细表、订单地址表、订单状态日志表和发货表分别承担不同职责，使得订单创建、支付、发货、确认收货和状态追踪在数据库层面具备清晰的可追溯性。

静态资源层用于存放头像、商品图片、店铺 Logo、站点图标等上传内容。该层被独立设计而非直接混放在模板目录中，主要考虑到两点：其一，用户上传资源与模板文件属于不同类型的数据，应在文件组织上与源码分离；其二，线上运行环境中上传资源通过独立路径对外提供访问，更便于反向代理、缓存策略和资源清理。实际运行时，系统通过统一的 `/uploads/` 访问路径对外提供图片资源。

后台任务层承担订单状态轮询与自动流转职责。电商系统中的订单并非全部依赖用户点击推动，例如超时未支付订单的取消、已发货订单的自动推进等，都具有后台检查需求。因此，系统在主应用之外保留后台任务机制，以降低将所有状态推进完全绑定在用户请求中的风险。该层的设计目的在于补充交互式请求难以覆盖的时间维度控制，使订单状态管理更加完整。

系统在设计结果上形成了一种边界相对清晰的运行结构：页面负责展示与输入，应用服务负责业务判断与流程控制，数据库负责实体存储与关系支撑，上传目录负责资源管理，后台任务负责状态轮询。该架构为后续数据库设计、详细功能实现以及部署运行提供了统一基础。

2 角色与权限结构设计

本系统采用基于角色的访问控制方式组织用户功能，其设计基础在于：电商平台中的不同参与者并不共享相同操作集合，若仅以“是否登录”作为权限判断标准，势必会导致后台功能暴露范围过大，页面之间职责边界不清，甚至引发业务越权。基于此，系统在概要设计阶段即明确将角色体系划分为普通用户、商家和管理员三类，并围绕三类角色分别组织功能入口与操作边界。

普通用户角色对应系统最主要的前台使用群体。该角色的设计重点在于保证其能够完整参与交易流程，但不接触平台治理与商家经营能力。因此，普通用户拥有商品浏览、搜索、购物车管理、地址维护、订单支付、确认收货、钱包查看和个人资料维护等功能权限。该角色的功能集合围绕“购买”展开，其页面入口主要集中在首页、商品页、订单页、钱包页和用户主页等前台页面中。

商家角色是在普通用户基础上扩展出的经营角色。系统没有在注册阶段直接开放商家权限，而是采用“申请—审核—授权”的转换路径。这样设计的原因在于，商家功能涉及商品发布、订单发货和店铺主页维护等经营能力，若缺乏审核机制，平台将无法控制经营主体的进入条件。基于此，系统通过商家申请表记录用户的申请信息，再由管理员审核后正式分配商家角色，并建立对应商家实体与店铺实体。商家角色的功能边界包括商品管理、订单处理、店铺信息编辑与商家后台访问，不参与平台治理层功能。

管理员角色用于承担平台治理职责。该角色的设计重点不在交易本身，而在于系统级配置和审核控制。因此，管理员能够进入管理员后台主页，对商家申请进行审核，对站点名称、浏览器标题和图标等品牌信息进行修改，并对用户钱包执行调账操作。

为了使角色划分真正落地，系统在实现层面采用路由级权限控制而非模板层隐藏。也就是说，商家后台、管理员后台等页面并不只是通过前端按钮可见性来区分，而是在请求进入后由后端判断当前用户是否具备对应角色，再决定是否允许访问。

最终系统形成了“普通用户负责交易、商家负责经营、管理员负责治理”的角色结构。该结构使主要业务路径和后台维护路径得到分离，为后续功能扩展和

安全控制提供了基础。

3 功能模块划分设计

在总体架构和角色体系确定之后，系统进一步按照业务对象和操作链路划分功能模块。模块划分的目标不是简单罗列页面，而是根据系统中需要长期维护的功能单元建立相对独立的业务边界，使每个模块在职责上具有明确指向，并能够通过有限接口与其他模块协同。

3.1 用户认证与账号模块

用户认证与账号模块承担注册、登录、邮箱验证、登录态恢复、退出登录、个人资料维护等基础能力。该模块被优先独立，是因为所有交易行为、经营行为和治理行为都建立在“可识别的当前用户”之上。系统在认证链路中采用 Firebase 邮箱认证作为主要登录方式，并保留本地兼容登录逻辑，使认证模块既能够满足邮箱验证需求，又能够兼顾系统内部的业务用户同步机制。该模块的设计结果是：前台所有与身份相关的页面和接口都围绕统一登录态展开，角色判断也建立在此模块输出的业务用户信息之上。

3.2 前台商品展示模块

商品展示模块面向所有访问者，负责首页内容、全部商品页、商品详情页、搜索结果页以及推荐店铺和全部店铺页的组织。它更强调信息呈现、入口引导和商品曝光。系统在此模块中不仅展示商品本身，还通过店铺入口将前台浏览行为与商家经营主体建立联系。设计上将商品展示从购物车、订单等流程中抽离，有助于保持页面结构简洁，并使后续推荐逻辑、分类逻辑和店铺展示逻辑具备独立扩展空间。

3.3 购物车与结算模块

购物车与结算模块承担“商品选择到订单生成”之间的过渡功能。若将其与订单模块完全混合，会导致下单前暂存状态与正式订单状态界限不清。由购物车表和购物车明细表负责保存用户当前待购商品；结算页则在此基础上完成地址选择、库存校验和订单拆分。

3.4 订单与支付模块

订单与支付模块是系统的核心业务模块之一，负责订单创建后的状态组织，包括待支付、已支付待发货、已发货、已完成、已取消等主要状态，以及单订单支付、批量支付、订单详情查看、确认收货和订单状态日志记录等功能。该模块被重点独立，是因为订单对象在电商系统中具有最强的业务中心性：它连接用户、商品、地址、钱包、发货和结算等多个实体。系统在概要设计阶段即确定订

单采用主表、明细表、地址表、状态日志表和发货表分离的结构，并以统一状态流转逻辑组织支付与履约行为。

3.5 钱包与地址管理模块

钱包与地址模块负责交易链路中的两个辅助对象。地址信息是下单必需条件，钱包余额是支付功能的重要基础。地址模块负责地址新增、默认地址设置、删除与结算选择；钱包模块负责余额查看、流水查看与支付后的余额变化记录。

3.6 商家经营模块

商家经营模块包括商家申请、商家后台主页、商品管理、订单管理、发货处理、店铺主页编辑和富文本图片上传等功能。系统将商家经营行为统一收敛到商家后台中，从而保证商家相关页面、数据库操作和上传行为具有统一入口。其结果是，商品发布、店铺展示和订单履约之间形成了连续的经营链条。

3.7 管理员治理模块

管理员治理模块负责平台配置与审核控制，包括后台概况、商家申请审核、站点品牌设置、钱包调账等功能。系统在设计时将治理能力独立组织，不仅提高了后台结构清晰度，也便于在后续平台维护中区分治理功能边界。

3.8 安全防护与异常处理模块

安全防护与异常处理模块不是某个具体页面集合，而是跨模块运行的支撑层。该模块包括登录保护、角色判断、CSRF 防护、安全响应头、请求限流、上传图片校验、富文本内容清洗以及统一错误页和 JSON 异常响应等能力。将安全能力视为独立模块，是因为这些机制并不服务于单一页面，而服务于整个平台的运行边界控制。其设计结果在于：系统的前台交易、商家经营和管理员治理功能，均能够在相对一致的安全约束下运行。

通过上述模块划分，系统最终形成了以认证为入口、以商品与订单为核心、以商家经营和管理员治理为后台延伸、以安全与异常处理为支撑边界的模块结构。

4 业务流程组织设计

系统概要设计除了确定分层和模块，还需要明确主要业务流程的组织方式。业务流程设计的重点并不在于列出所有页面跳转，而在于说明系统如何将不同模块串联成稳定的业务路径，从而保证用户操作、数据库变化和后台状态推进之间保持一致。

4.1 普通用户交易流程

系统将“浏览商品—加入购物车—提交结算—生成订单—支付订单—等待发货—确认收货”组织为一条连续主链路。在此过程中，商品展示模块、购物车模块、地址模块、订单模块和钱包模块依次参与。设计上将这些步骤按时间顺序展开，可以保证每一步都以明确的数据对象作为输入：浏览阶段面向商品，购物车阶段面向购物车明细，结算阶段面向地址与库存校验，支付阶段面向订单与钱包余额，收货阶段面向发货记录与订单状态。

4.2 商家经营流程

该流程以“入驻申请—审核通过—商品维护—订单查看—执行发货—维护店铺主页”为主线。商家经营模块并不参与用户的支付操作，但会在订单支付完成后接管后续履约环节。系统将商家流程与用户交易流程在“已支付待发货”这一节点上连接起来，从而使用户侧与商家侧职责自然衔接。订单支付前主要由用户推动，订单支付后主要由商家和后台状态机制共同推动。

4.3 管理员治理流程

该流程主要围绕“审核、配置、调账”展开，与商品浏览和订单下单流程保持相对独立。管理员审核商家申请后，系统会同步建立商家和店铺实体；管理员修改站点名称、浏览器标题和图标后，前台与后台页面的品牌信息随之变化；管理员执行钱包调账后，用户钱包余额和流水记录同步更新。

4.4 后台状态推进流程

订单超时未支付、已发货后的自动推进等操作，并不完全依赖用户点击触发，而由后台任务进行轮询检查。这样的设计能够补足请求驱动模型在时间维度上的不足，使系统在无人工干预的情况下仍能保持订单状态合理流转。业务流程不再完全依赖单次 HTTP 请求，而形成“用户操作 + 后台任务”共同推动的双重组织方式。

通过上述流程设计，系统在总体上形成了三条主线：面向普通用户的交易主线、面向商家的经营主线、面向管理员的治理主线，并由后台任务补充状态推进逻辑。

5 运行结构设计

系统实际运行于 Ubuntu 22.04 服务器环境。整体运行结构采用 Nginx、Gunicorn、Flask 与 SQL Server 组合方式构建。

在访问链路上，Nginx 作为站点外部访问入口，负责接收客户端的 HTTP/HTTPS 请求，并将动态请求转发至本机应用服务。Flask 应用由 Gunicorn 托管运行，监听服务器本机回环地址，由其完成页面渲染、业务处理和数据库访问。SQL Server

作为独立的数据服务层，负责保存用户、角色、商品、订单、地址、钱包及交易流水等业务数据。

在资源访问方面，头像、商品图片、店铺 Logo 和站点图标等上传内容通过独立路径对外提供访问，与动态页面处理过程分离。除前台访问链路外，系统还包含后台状态任务，用于处理订单超时检查和状态推进，从而补充请求驱动之外的业务流转。

系统最终形成了由反向代理层、应用服务层、数据服务层、资源访问层和后台任务层组成的运行结构。

第二章　数据库设计

1　用户与权限模块设计

用户与权限模块是系统数据库设计中的基础部分，其主要任务是保存用户身份信息、定义角色类型，并建立用户与角色之间的对应关系，支持角色扩展和权限分配。系统在该模块中设置了 users、roles 和 user_roles 三张表，分别承担用户实体存储、角色定义和角色关联管理三项职责。

通过“用户主表 + 角色定义表 + 用户角色关联表”构建出权限基础结构。users 表负责保存用户主体信息，roles 表负责保存角色类别，user_roles 表负责完成二者之间的映射关系。使系统在数据库层面具备了用户身份管理与角色扩展能力，也为前台访问控制、商家入驻授权和管理员后台治理提供了统一的数据基础。

1.1　用户表设计

users 表用于保存系统中的基础用户信息。以 user_id 作为主键，设置 username 和 email 两个唯一约束，用于分别保证用户名和邮箱地址在系统中的唯一性。

users 表除用户名、密码散列值和邮箱等基础认证字段外，还包含手机号、状态、创建时间、更新时间、昵称和头像地址等扩展字段。其中，password_hash 字段用于保存密码散列结果；status 字段用于表示用户当前可用状态，并通过默认值使新建用户自动进入启用状态；created_at 和 updated_at 字段用于记录用户数据的创建与更新时间；nickname 和 avatar_url 服务于前台个人资料展示与用户主页显示。

1.2　角色表设计

roles 表用于定义系统中的角色类型。以 role_id 为主键，以 role_name 保存角色名称，并对 role_name 设置唯一约束。该设计的目的在于将角色定义从用户表中独立出来，使系统在权限组织上采用“角色字典 + 角色分配”模式，而不是在用户表中直接设置单一角色字段。角色类型具有独立管理能力，后续若增加新的角色类别，也不需要修改用户主表结构。

从系统当前实现来看，角色表承担的是权限抽象层的作用。在数据库中形成角色定义，再由后端根据角色关联结果决定当前用户可访问的功能范围。

1.3 用户角色关联表设计

user_roles 表用于建立用户与角色之间的关联关系。该表由 user_id 和 role_id 两个字段组成，并以二者的组合作为联合主键，同时分别通过外键连接到 users 表和 roles 表。系统在权限模型上采用的是多对多关联方式：一个用户可以关联多个角色，一个角色也可以被分配给多个用户。

采用独立关联表而非在用户表中直接设置角色字段：系统权限模型不被限制为单角色结构，能够兼容用户在不同阶段获得不同身份的情况；角色分配逻辑被独立管理后，用户实体信息与权限信息在数据库层面实现了解耦，有利于后续维护和审核操作。该结构能够较好支撑当前系统中普通用户申请成为商家、管理员审核后为其授予商家角色的业务过程，也使管理员角色与普通用户角色的并存成为可能。

2 商家与店铺模块设计

商家与店铺模块用于支持普通用户申请成为商家，并在审核通过后进入实际经营状态。系统在该模块中设置了 merchant_applications、merchants 和 shops 三张表，分别承担申请信息保存、商家主体信息保存和店铺信息保存三项职责。

通过“申请表 + 商家表 + 店铺表”构建出商家侧基础结构。merchant_applications 表负责保存入驻申请过程数据，merchants 表负责保存审核通过后的商家主体信息，shops 表负责保存店铺主页、店铺标识和展示信息。该结构使商家申请流程与正式经营数据在数据库层面保持分离，也使审核前后两类数据具有清晰边界。

2.1 商家申请表设计

merchant_applications 表用于保存用户提交的商家入驻申请信息。以 application_id 作为主键，并通过 user_id 外键连接到 users 表，用于标识申请所属用户。表中同时保存 shop_name、contact_name、contact_phone、business_scope 和 application_reason 等字段，用于记录申请时提交的店铺名称、联系人信息、经营范围和申请说明。

该表还设置了 status、review_remark、reviewed_by、created_at 和 reviewed_at 等字段，用于保存申请状态、审核意见、审核人和审核时间。这样设计后，申请数据不仅能够支持“提交申请”这一动作，也能够完整记录后续审核过程，使商家入驻流程具有独立的数据基础。

2.2 商家表设计

merchants 表用于保存审核通过后的商家主体信息。以 merchant_id 作为主键，通过 user_id 外键连接到 users 表，并对 user_id 设置唯一约束，用于

保证一个用户在系统中最多对应一个商家主体。

该表主要包含 shop_name、description、status 和 created_at 等字段。其中，shop_name 用于保存商家初始店铺名称，description 用于保存商家描述信息，status 用于表示商家当前状态。该设计使商家主体能够从普通用户信息中独立出来，形成系统中的正式经营实体。

2.3 店铺表设计

shops 表用于保存店铺的展示与经营页面信息。以 shop_id 作为主键，通过 merchant_id 外键连接到 merchants 表，并对 shop_slug 设置唯一约束，用于保证店铺访问标识在系统中的唯一性。表中同时对 merchant_id 建立索引，以支持店铺与商家主体之间的快速关联。

shops 表主要包含 shop_name、shop_slug、logo_url、banner_url、intro、announcement、contact_phone、status、created_at 和 updated_at 等字段。其中,logo_url 和 banner_url 用于保存店铺页面资源地址,intro 和 announcement 用于保存店铺简介与公告信息，shop_slug 则作为店铺主页访问标识。通过这些字段，店铺表能够同时承担店铺展示信息和店铺页面标识的管理功能。

3 商品模块设计

商品模块用于完成平台中的商品分类、商品主体信息保存和商品图片资源管理。系统在该模块中设置了 categories、products 和 product_media 三张表，分别承担分类信息保存、商品主体信息保存和商品媒体资源管理三项职责。

通过“分类表 + 商品表 + 商品媒体表”构建出商品基础结构。categories 表负责保存商品分类信息，products 表负责保存商品主体数据，product_media 表负责保存商品相关图片资源。该结构使商品分类、商品信息和商品展示资源在数据库层面保持分离，也使商品模块具备较清晰的组织层次。

3.1 分类表设计

categories 表用于保存商品分类信息。以 category_id 作为主键，对 category_name 设置唯一约束，用于保证分类名称在系统中的唯一性。表中还设置了 parent_id 字段，并通过外键连接到本表主键，用于支持分类层级关系的建立。

该设计使分类表不仅能够保存基础分类名称，也能够支持父子分类结构，从而为商品分类扩展提供基础。在当前系统中，分类结构已经具备层级设计能力，即使现阶段以前台一级分类使用为主，数据库层面仍然保留了后续扩展空间。

3.2 商品表设计

products 表用于保存商品主体信息。以 product_id 作为主键，并通过 merchant_id、category_id 和 shop_id 分别连接到 merchants、categories 和 shops 表，用于表示商品所属商家、所属分类和所属店铺。

该表主要包含 product_name、description、price、stock、status、created_at 和 updated_at 等字段。其中，product_name 用于保存商品名称，description 用于保存商品说明内容，price 用于保存商品价格，stock 用于保存库存数量，status 用于表示商品当前状态，创建时间与更新时间字段用于记录商品数据的生命周期变化。

products 表同时对 shop_id 建立索引，用于支持商品与店铺之间的快速关联。通过这些字段和约束，商品表能够同时承担商品信息保存、库存记录、状态控制和店铺归属管理等功能，是商品模块中的核心业务表。

3.3 商品媒体表设计

product_media 表用于保存商品的媒体资源信息。以 media_id 作为主键，通过 product_id 外键连接到 products 表，用于表示媒体资源所属商品。表中设置了 media_type、file_url、sort_order 和 created_at 等字段，用于保存资源类型、文件地址、排序顺序和创建时间。

商品图片资源不直接存放在商品表中，而是作为独立对象进行管理。便于维护多张图片资源以及后续按排序规则组织商品展示内容。

4 购物车与地址模块设计

购物车与地址模块用于支持用户在正式下单前完成商品暂存与收货信息维护。系统在该模块中设置了 carts、cart_items 和 user_addresses 三张表，分别承担购物车主体信息保存、购物车商品项保存和用户收货地址保存三项职责。

通过“购物车表 + 购物车明细表 + 地址表”构建出下单前基础结构。carts 表负责保存用户购物车主体，cart_items 表负责保存购物车中的具体商品项，user_addresses 表负责保存用户可选收货地址。该结构使商品暂存数据与订单正式数据在数据库层面保持分离，也使地址信息能够在多次结算过程中重复使用。

4.1 购物车表设计

carts 表用于保存用户购物车主体信息。以 cart_id 作为主键，通过 user_id 外键连接到 users 表，并对 user_id 设置唯一约束，用于保证一个用户在系统中只对应一个购物车。表中同时设置 created_at 字段，用于记录购物车创建时间。

该设计使购物车主体与用户之间形成一对一关系，便于系统在加入购物车、

修改购物车和删除购物车项时始终围绕同一购物车对象进行处理。

4.2 购物车明细表设计

cart_items 表用于保存购物车中的具体商品项。以 cart_item_id 作为主键，通过 cart_id 外键连接到 carts 表，通过 product_id 外键连接到 products 表。表中设置 quantity 字段用于保存商品数量，设置 created_at 字段用于记录购物车项创建时间。

该设计使购物车中的每一项商品都作为独立记录保存，便于系统对商品数量进行修改、删除和结算前校验。

4.3 用户地址表设计

user_addresses 表用于保存用户收货地址信息。以 address_id 作为主键，通过 user_id 外键连接到 users 表，并对 user_id 建立索引，用于支持用户地址查询。表中主要包含 receiver_name、receiver_phone、province、city、district、detail_address、postal_code、is_default、created_at 和 updated_at 等字段。

其中，is_default 字段用于表示当前地址是否为默认地址，收件人姓名、电话和详细地址字段用于保存结算时所需的收货信息。该设计使用户地址能够独立维护，并在后续订单结算过程中直接选用。

5 订单模块设计

订单模块用于保存用户提交订单后的正式交易数据，并支持订单明细、收货地址、状态日志和发货信息的独立管理。系统在该模块中设置了 orders、order_items、order_addresses、order_status_logs 和 shipments 五张表，分别承担订单主体信息保存、订单商品项保存、订单地址快照保存、订单状态变化记录和发货信息保存五项职责。

通过“订单主表 + 订单明细表 + 订单地址表 + 状态日志表 + 发货表”构建出订单基础结构。orders 表负责保存订单主体信息，order_items 表负责保存订单中的具体商品项，order_addresses 表负责保存订单提交时的收货地址快照，order_status_logs 表负责记录订单状态变化过程，shipments 表负责保存订单发货与签收信息。该结构使订单数据在数据库层面形成较完整的链路，也使订单状态变化与发货过程能够独立记录。

5.1 订单主表设计

orders 表用于保存订单主体信息。以 order_id 作为主键，通过 user_id 外键连接到 users 表，用于表示订单所属用户。表中设置了 total_amount、order_status、created_at 和 paid_at 等字段，用于保存订单总金额、订单

状态、创建时间和支付时间。

其中，order_status 字段用于表示订单当前所处状态，paid_at 字段用于记录订单实际支付时间。该设计使订单主表能够作为订单流程中的核心对象，统一关联后续的订单明细、地址、日志和发货数据。

5.2 订单明细表设计

order_items 表用于保存订单中的具体商品项。以 order_item_id 作为主键，通过 order_id、product_id、merchant_id 和 shop_id 分别连接到 orders、products、merchants 和 shops 表，用于表示该商品项所属订单、所属商品、所属商家和所属店铺。

表中设置了 price、quantity 和 subtotal 字段，用于保存下单时的商品单价、购买数量和该商品项小计金额。shop_id 字段同时建立索引，用于支持按店铺维度查询订单商品项。该设计使订单中的每一件商品都能够作为独立记录保存，便于后续订单展示、发货处理和结算计算。

5.3 订单地址表设计

order_ addresses 表用于保存订单提交时的收货地址快照。以 order_ address_id 作为主键，通过 order_id 外键连接到 orders 表，并对 order_id 设置唯一约束，用于保证一个订单只对应一条地址快照记录。

表中主要包含 receiver_name、receiver_phone、province、city、district、detail_address、postal_code 和 created_at 等字段，用于保存订单提交当时的完整收货信息。该设计使订单地址信息在下单后独立固定，不受用户后续修改地址表数据的影响。

5.4 订单状态日志表设计

order_status_logs 表用于记录订单状态变化过程。以 log_id 作为主键，通过 order_id 外键连接到 orders 表，通过 operator_user_id 外键连接到 users 表，用于表示状态变化所属订单及操作人。

表中设置了 old_status、new_status、remark 和 created_at 等字段，用于保存状态变更前后值、变更说明和变更时间。该设计使订单状态推进过程能够被独立记录，便于后续查看订单生命周期中的关键节点。

5.5 发货表设计

shipments 表用于保存订单发货与签收信息。以 shipment_id 作为主键，通过 order_id 和 shop_id 分别连接到 orders 表和 shops 表，并对 order_id 建立索引，用于支持按订单维度查询发货记录。

表中主要包含 carrier_name、tracking_no、shipped_at、received_at、status 和 created_at 等字段，用于保存物流公司名称、运单号、发货时间、签

收时间和发货状态。该设计使发货信息能够从订单主表中独立出来，便于记录订单履约过程。

6 钱包与安全支撑模块设计

钱包与安全支撑模块用于保存用户钱包信息、资金流水记录以及请求限流状态。系统在该模块中设置了 wallets、wallet_transactions 和 request_rate_limits 三张表，分别承担钱包主体信息保存、钱包交易流水保存和限流状态记录三项职责。

通过“钱包表 + 钱包流水表 + 限流表”构建出资金记录与安全支撑结构。wallets 表负责保存用户钱包主体信息，wallet_transactions 表负责保存钱包变动流水，request_rate_limits 表负责保存请求限流状态。该结构使资金数据与限流数据在数据库层面分别管理，也使支付、调账和高频请求控制具有独立的数据基础。

6.1 钱包表设计

wallets 表用于保存用户钱包主体信息。以 wallet_id 作为主键，通过 user_id 外键连接到 users 表，并对 user_id 设置唯一约束，用于保证一个用户只对应一个钱包。表中设置了 balance 和 updated_at 字段，用于保存当前余额和最近更新时间。

其中，balance 字段采用两位小数金额类型保存钱包余额，默认值为 0。该设计使钱包主体能够独立于用户表存在，并作为支付、结算和调账操作的直接数据对象。

6.2 钱包流水表设计

wallet_transactions 表用于保存钱包交易流水。以 transaction_id 作为主键，通过 wallet_id 外键连接到 wallets 表，通过 related_order_id 外键连接到 orders 表，用于表示该笔流水所属钱包及关联订单。

表中设置了 tx_type、amount、balance_after、remark 和 created_at 等字段，用于保存流水类型、变动金额、变动后余额、备注信息和创建时间。该设计使钱包余额变化过程能够独立记录，便于后续查看支付、结算和调账过程中的资金变化。

6.3 请求限流表设计

request_rate_limits 表用于保存请求限流状态。以 scope_key 作为主键，用于标识当前限流对象。表中设置了 hit_count、limit_count、window_seconds、window_started_at 和 updated_at 等字段，用于保存当前命中次数、限制次数、时间窗口长度、窗口起始时间和最近更新时间。

该设计使限流状态能够在数据库层面独立保存，便于系统对登录、支付、发货和上传等高频操作进行统一控制。

第三章　系统详细设计

1　用户注册与登录模块设计

用户注册与登录模块用于完成账号注册、邮箱验证、统一登录、密码重置和退出登录，是系统全部业务功能的入口模块。系统在该模块中设置了 Firebase 邮箱注册、Firebase 邮箱登录、本地兼容登录和本地账号重置密码四类入口，并通过统一页面入口组织认证流程。

该模块在实现上将注册、登录、密码重置、业务会话建立和退出登录分别组织。注册与登录页面负责用户输入和前端认证过程，后端接口负责业务用户同步、角色识别和会话写入。这样处理后，邮箱认证链路与本地兼容链路能够在同一模块内协同完成。

1.1　功能实现

本模块主要实现以下功能：用户通过邮箱注册新账号；注册成功后发送邮箱验证邮件；用户通过统一登录页完成邮箱登录或本地兼容登录；用户可通过邮箱方式发起密码重置；管理员可对旧测试账号或本地账号执行本地密码重置；用户退出系统时清空当前业务会话。

系统当前将 `/firebase-register` 和 `/firebase-login` 作为主要页面入口，将 `/localLogin` 和 `/sessionLogin` 作为主要登录接口入口。旧的 `/login` 和 `/register` 路径继续保留，用于跳转到统一认证页面。

1.2　控制器设计

本模块的控制器主要包括 `firebase_register()`、`firebase_login()`、`local_login()`、`session_login()`、`reset_password()`、`local_reset_password()` 和 `logout()`。

其中，`firebase_register()` 和 `firebase_login()` 负责返回注册页和登录页模板；`local_login()` 负责处理本地兼容登录请求；`session_login()` 负责处理 Firebase 认证成功后的业务会话建立；`reset_password()` 用于返回邮箱密码重置页面；`local_reset_password()` 用于处理本地账号密码修改；`logout()` 用于清空当前业务会话并退出系统。

控制器划分后，页面访问、异步登录接口和密码处理逻辑分别对应不同入口，模块边界较为清晰。

1.3 Model 设计

本模块主要依赖 users、roles 和 user_roles 三张表。users 表用于保存系统中的基础用户信息，包括用户名、密码散列值、邮箱、状态、昵称和头像地址等字段；roles 表用于定义角色类别；user_roles 表用于建立用户与角色之间的对应关系。

在本模块中，users 表直接参与本地兼容登录和本地密码重置过程。登录接口根据用户名或邮箱查询用户记录，并结合密码散列值完成校验；本地密码重置接口在确认目标账号存在后更新用户密码散列值。角色数据通过 roles 和 user_roles 读取，并在登录成功后写入业务会话，用于标识当前用户是否具有商家或管理员身份。

对于邮箱认证链路，系统在用户完成邮箱验证并登录成功后，再同步内部业务用户数据。这样处理后，外部认证结果与系统内部业务用户之间形成映射关系。

1.4 视图与前端交互实现

本模块对应的主要视图包括 firebase_register.html、firebase_login.html、reset_password.html、login.html 和 local_reset_password.html。

注册页面提供邮箱和密码输入项，前端通过页面脚本调用 Firebase 接口创建账号并发送验证邮件。当检测到邮箱已被注册但尚未完成验证时，页面脚本会尝试重新登录该账号并再次发送验证邮件，从而保证注册流程可以继续进行。

登录页面提供“用户名或邮箱”和“密码”两个输入项。页面脚本先根据输入内容判断登录方式：当输入内容为邮箱时，先通过 Firebase 完成认证，再调用 /sessionLogin 建立系统业务会话；当输入内容为用户名时，则直接调用 /localLogin 完成本地兼容登录。由此，统一登录页能够兼容两类认证入口。

邮箱密码重置页面通过前端脚本发送重置邮件；本地账号密码重置页面采用表单提交方式，由后端直接处理新密码写入。旧登录页 login.html 不再承担实际登录处理，而作为过渡页面将访问请求跳转到统一登录页。

1.5 模块实现特点

本模块的实现特点主要体现在统一入口、双链路兼容和业务会话分离三个方面。系统通过统一登录页和统一注册页组织主要认证入口；同时保留 Firebase 邮箱认证与本地兼容登录两条链路；前端认证结果与后端业务会话采用分步建立方式处理，使系统内部用户数据与外部认证服务保持相对独立。

2 首页展示模块设计

首页展示模块用于完成站点首页的综合展示，是用户进入系统后的主要入口页面。该模块集中展示站点导航、搜索入口、商品分类、热门商品、推荐店铺以

及商品总数和店铺总数等信息。

2.1 功能实现

本模块主要实现首页信息聚合展示。用户进入首页后，可以查看商品分类、热门商品和推荐店铺，并通过页面中的导航、搜索框和快捷入口进入后续功能页面。首页不承担具体交易处理，而是负责组织前台主要浏览入口。

2.2 控制器设计

本模块主要由 home() 控制器实现。控制器在页面渲染前完成首页所需数据的集中查询，包括热门商品、分类列表、推荐店铺、商品总数和店铺总数。

在具体处理过程中，控制器首先查询已上架商品，并结合已完成订单中的销量统计结果，按销量降序选取热门商品；随后读取前若干个商品分类，用于构成首页分类入口；之后再读取正常营业店铺，并结合订单明细中的成交额统计结果生成推荐店铺列表。全部查询完成后，控制器将结果统一传递给首页模板进行渲染。

2.3 Model 设计

本模块主要使用 products、product_media、categories、shops、merchants、order_items 和 orders 等表。商品表、商品图片表和分类表用于组织首页商品展示内容，店铺表和商家表用于组织推荐店铺主体信息，订单表和订单明细表用于提供销量和成交额统计结果。

其中，热门商品区域直接依赖商品销量统计结果，推荐店铺区域直接依赖店铺成交额统计结果，因此首页展示内容与系统中的实际交易数据保持一致。

2.4 视图实现

本模块对应视图为 home.html。该模板主要包含顶部导航区、搜索区、分类区、快捷入口区、热门商品区和推荐店铺区。模板中通过服务端传入的数据直接渲染商品卡片、店铺卡片和分类入口，并在页面顶部统一显示用户导航信息。

首页中的搜索入口采用表单方式提交，商品和店铺入口采用超链接方式跳转，因此页面本身不依赖复杂前端脚本，而是以模板渲染为主完成首页展示。

2.5 模块实现特点

本模块的特点是将前台主要浏览入口集中在同一页面中，并通过商品销量和店铺成交额对首页展示内容进行排序，使首页同时承担站点入口和内容聚合展示作用。

3 搜索模块设计

搜索模块用于完成站内统一检索，支持商品、店铺和用户三类对象的搜索，是前台浏览模块中的补充入口。该模块将搜索输入、结果分类和结果展示集中在同一页面中，用户可以在不同结果类型之间切换查看。

3.1 功能实现

本模块主要实现统一关键词搜索。用户输入关键词后，系统根据当前搜索类型返回商品、店铺或用户结果，并在结果页中显示三类结果数量。搜索页既可以作为独立入口使用，也可以由首页和商品页中的搜索框跳转进入。

3.2 控制器设计

本模块主要由 `search()` 控制器实现，同时配合 `_coerce_search_type()` 和 `_trim_search_text()` 两个辅助函数完成参数整理与文本截断。

控制器首先读取关键词参数 `q` 和搜索类型参数 `type`，并对搜索类型进行标准化处理，保证结果只在商品、店铺和用户三类中切换。随后，控制器根据关键词构造模糊匹配条件，分别执行商品搜索、店铺搜索和用户搜索，并将三类查询结果同时保存。最后，控制器统计三类结果数量，连同当前搜索类型一并传递给搜索模板进行渲染。

在具体处理过程中，商品搜索优先匹配商品名称，并同时检索商品说明、分类名称和店铺名称；店铺搜索匹配店铺名称、店铺简介和公告；用户搜索匹配昵称、用户名和商家店铺名称。这样处理后，搜索模块能够覆盖前台主要浏览对象。

3.3 Model 设计

本模块主要使用 `products`、`product_media`、`categories`、`shops`、`merchants`、`users`、`roles`、`user_roles`、`orders` 和 `order_items` 等表。商品搜索依赖商品表、商品图片表、分类表和店铺表；店铺搜索依赖店铺表、商家表、订单表和订单明细表；用户搜索依赖用户表、角色表、用户角色关联表以及商家、店铺关联数据。

其中，商品结果中包含商品封面图、价格、库存、分类和所属店铺；店铺结果中包含店铺 Logo、简介、销量和成交额；用户结果中会根据角色关联结果区分普通用户和商家用户，并在商家用户存在公开店铺时提供店铺入口。用户搜索中还排除了管理员账号，使前台搜索结果保持面向普通浏览场景。

3.4 视图实现

本模块对应视图为 `search.html`。该模板主要包含搜索输入区、结果类型切换区和结果展示区。页面顶部提供统一搜索表单，搜索结果区域通过标签形式切换商品、店铺和用户三类结果，并在标签上显示当前结果数量。

在结果展示上，商品结果以商品卡片形式展示，店铺结果以店铺卡片形式展示，用户结果则根据是否为商家用户显示不同标识。搜索页本身未使用复杂前端交互脚本，结果切换通过链接参数完成，页面主要采用服务端渲染方式组织。

3.5 模块实现特点

本模块的特点是将商品、店铺和用户三类检索对象统一组织在同一搜索页面中，并通过结果数量和标签切换保持页面结构一致。这样处理后，系统搜索入口集中，前台检索路径也更加清晰。

4 商品列表与商品详情模块设计

商品列表与商品详情模块用于完成商品集中展示与单个商品的详细查看，是前台浏览流程中的核心模块。该模块将商品筛选、商品摘要展示、商品详情展示和加入购物车入口分别组织，使用户能够从商品列表页进入商品详情页，再进入后续购物流程。

4.1 功能实现

本模块主要实现商品列表展示、按分类筛选商品、查看商品详情和从商品页加入购物车。用户进入商品列表页后，可以查看全部在售商品，也可以按分类筛选；进入商品详情页后，可以查看商品价格、库存、所属店铺、所属分类、商品简介和商品说明，并直接执行加入购物车操作。

4.2 控制器设计

本模块主要由 `products()` 和 `product_detail()` 两个控制器实现，同时使用 `make_product_intro()` 对商品说明生成摘要文本。

其中，`products()` 控制器负责读取分类参数并查询当前在售商品。控制器先读取全部分类列表，再根据当前是否传入分类编号决定是否追加筛选条件，最后将商品结果和分类结果统一传递给商品列表模板。商品结果中同时包含商品名称、价格、库存、分类名称、所属店铺、封面图和简介摘要。

`product_detail()` 控制器负责读取单个商品的完整信息。控制器根据商品编号查询商品主体、所属分类、所属店铺和封面图，并组织商品详情页所需数据。该页除基础字段外，还会展示商品说明内容和简介文本，用于区分列表页摘要展示与详情页完整展示。

4.3 Model 设计

本模块主要使用 `products`、`product_media`、`categories` 和 `shops` 四张表。商品表提供商品名称、价格、库存、状态和说明内容；商品图片表提供商品封面图；分类表提供分类名称；店铺表提供所属店铺名称和店铺主页标识。

在列表页查询中，系统只读取已上架商品，并通过商品图片表中的首张图片作为封面图，同时生成商品简介摘要用于卡片展示。在详情页查询中，系统按商品编号读取单个商品记录，并同时带出分类和店铺信息，使详情页能够完整展示商品所属关系。

4.4 视图实现

本模块对应视图为 products.html 和 product_detail.html。商品列表页模板主要包含分类筛选条、搜索入口和商品卡片列表。每个商品卡片展示商品封面图、名称、分类、所属店铺、库存、简介摘要和价格，并提供“查看详情”和“加入购物车”两个入口。

商品详情页模板主要展示商品封面图、商品名称、所属店铺、所属分类、简介摘要、价格、库存和完整商品说明。页面下方设置商品说明区域，用于展示商品详细内容，并保留加入购物车和返回商品列表两个操作入口。

在前端交互上，商品列表页和商品详情页都为加入购物车操作设置了提示逻辑。表单提交后，页面脚本会通过本地存储或查询参数读取提示信息，并在页面右上角显示“已加入购物车”的提示框。

4.5 模块实现特点

本模块的特点是将商品列表浏览和商品详情查看分为两个层次组织。列表页强调批量展示和筛选，详情页强调单个商品的完整展示，并在两个页面中都保留加入购物车入口，使商品浏览与后续购买流程能够自然衔接。

5 购物车模块设计

购物车模块用于保存用户当前待购买商品，并提供商品加入、数量修改、删除和结算入口。该模块位于商品浏览与正式下单之间，是前台交易流程中的过渡模块。

5.1 功能实现

本模块主要实现查看购物车、加入购物车、修改购物车商品数量和删除购物车商品项四项功能。用户在商品列表页或商品详情页点击“加入购物车”后，商品会进入当前用户的购物车；进入购物车页后，可以继续调整数量、删除商品，或直接进入结算流程。

5.2 控制器设计

本模块主要由 cart()、cart_add()、cart_update() 和 cart_remove() 四个控制器实现，并通过 get_or_create_cart() 保证当前用户始终对应一个购物车主体。

其中，cart() 控制器负责读取当前用户购物车中的全部商品项，并计算商品总数与总金额，最后返回购物车页面。cart_add() 在接收到商品编号后，先检查商品是否存在、是否上架以及库存是否充足，再决定是新增购物车项还是在原有数量基础上加一。cart_update() 根据用户提交的新数量更新购物车项，并在更新前按商品库存限制最终数量。cart_remove() 则根据购物车项编号直接删除对应记录。

5.3 Model 设计

本模块主要使用 carts、cart_items、products 和 product_media 四张表。carts 表用于定位当前用户的购物车主体，cart_items 表用于保存购物车中的具体商品项，products 表用于读取商品价格、库存和状态，product_media 表用于读取商品封面图。

在数据处理上，购物车页读取每个购物车项的商品名称、单价、库存、封面图和小计金额，并在后端完成总金额和总数量统计。加入购物车和修改数量时，系统都会重新读取商品库存信息，从而保证购物车数据不会脱离当前商品状态。

5.4 视图实现

本模块对应视图为 cart.html。该模板用于展示购物车商品列表、商品小计、总数量和总金额，并提供修改数量、删除商品和进入结算的操作入口。模板顶部保留前台统一导航，主体区域围绕购物车商品项进行组织。

购物车页中的数量修改和删除操作采用表单提交方式实现，页面逻辑以服务端渲染为主，没有引入复杂前端交互脚本。商品加入购物车后的提示信息则由商品列表页和商品详情页中的表单提交结果带入，购物车页本身主要负责展示当前购物车状态。

5.5 模块实现特点

本模块的特点是将购物车作为独立对象组织，使商品暂存状态与正式订单状态保持分离。加入购物车、修改数量和删除商品都围绕购物车明细表进行处理，而不直接生成订单数据。

6 地址管理模块设计

地址管理模块用于维护用户收货地址信息，为结算流程提供可选地址数据。该模块支持地址新增、地址列表查看、默认地址设置和地址删除四项功能，是下单流程中的基础模块。

6.1 功能实现

本模块主要实现用户地址新增、地址查看、设为默认地址和删除地址。用户进入地址管理页后，可以填写收件人姓名、电话、省份、城市、区县、详细地址和邮编，并可在新增时直接设置默认地址。页面下方同时展示当前用户全部地址，并提供“设为默认”和“删除”两个操作入口。

6.2 控制器设计

本模块主要由 addresses()、address_set_default() 和 address_delete() 三个控制器实现。

其中，addresses() 同时处理地址新增和地址列表展示。请求方式为 GET 时，控制器按默认地址优先、地址编号倒序的方式读取当前用户全部地址；请求方式为 POST 时，控制器接收表单提交的地址信息，在数据完整的情况下写入新地址记录。若用户在新增时勾选“设为默认地址”，控制器会先将当前用户已有地址全部取消默认状态，再写入新的默认地址。

address_set_default() 用于处理默认地址切换。控制器先校验目标地址是否属于当前用户，再将该用户全部地址重置为非默认状态，随后把目标地址更新为默认地址。

address_delete() 用于处理地址删除。控制器在删除前先读取目标地址并确认其归属关系；若被删除的是默认地址，则会在删除后继续查询该用户剩余地址，并将其中一条地址重新设置为默认地址。这样可以保证用户地址集合中始终保留明确的默认地址状态。

6.3 Model 设计

本模块主要使用 user_addresses 表。该表用于保存用户的收货地址信息，包括收件人姓名、收件人电话、省份、城市、区县、详细地址、邮编、默认地址标记、创建时间和更新时间等字段。地址数据通过 user_id 与当前业务用户关联。

在数据处理上，地址列表按 is_default 和地址编号排序，使默认地址始终显示在最前方。新增地址、默认地址切换和地址删除都直接围绕 user_addresses 表执行，不与订单表混合处理，从而保持地址维护逻辑的独立性。

6.4 视图实现

本模块对应视图为 addresses.html。模板页面由新增地址表单和地址列表两部分组成。表单区域用于录入新地址信息，列表区域用于展示当前用户全部地址，并在每条地址下提供默认地址设置和删除按钮。默认地址在页面中使用单独标记显示。

页面中的新增、设为默认和删除操作均采用表单提交方式实现，前端逻辑以服务端渲染为主。地址状态变化后，页面通过重新加载展示最新地址结果。

6.5 模块实现特点

本模块的特点是将地址维护从结算流程中独立出来，并通过默认地址机制为后续下单提供直接可用的收货信息。新增地址、默认地址切换和默认地址删除后的自动补位都围绕同一地址集合处理。

7 结算与拆单模块设计

结算与拆单模块用于完成购物车商品校验、收货地址选择、金额汇总和订单生成，是购物车进入正式订单的关键环节。该模块将用户购物车中的商品转换为订单数据，并在存在多店铺商品时按店铺进行拆单处理。

7.1 功能实现

本模块主要实现结算页展示、收货地址选择、商品库存与状态校验、运费计算和订单生成。用户从购物车进入结算页后，可以查看当前待下单商品、选择收货地址，并在提交后生成正式订单。若购物车中包含多个店铺的商品，系统会按店铺拆分为多个订单。

7.2 控制器设计

本模块主要由 `checkout()` 控制器实现，并配合 `validate_cart_items_for_checkout()`、`_split_checkout_items_by_shop()` 和 `_create_split_orders()` 三个函数完成结算处理。

其中，`checkout()` 在进入结算页时先取得当前用户购物车，再校验购物车商品是否为空、商品是否上架以及库存是否充足。随后读取当前用户全部收货地址，并按默认地址优先组织地址列表。在金额处理上，控制器先统计商品总金额与商品总数量，再按“满 99 元包邮，否则收取 8 元运费”的规则计算运费和应付金额。

当用户提交结算时，控制器先检查是否已选择地址，再读取对应地址快照，并再次执行商品状态和库存校验。校验通过后，系统先按商家和店铺对购物车商品分组，再调用订单创建函数生成正式订单、订单明细、订单地址快照和初始状态日志。订单创建完成后，购物车明细会被清空；若只生成一个订单，则跳转至订单详情页，若生成多个订单，则跳转至批量支付页。

7.3 Model 设计

本模块主要使用 `carts`、`cart_items`、`products`、`user_addresses`、`orders`、`order_items`、`order_addresses` 和 `order_status_logs` 等表。购物车表和购物车明细表用于读取当前待结算商品，商品表用于校验库存和状态，地址表用于读取收货地址，订单相关各表用于保存正式下单后的数据。

在订单生成过程中，系统不会直接复用用户地址表记录，而是将当前选中的

地址信息写入订单地址表，形成独立地址快照。订单拆分后，每个订单都拥有各自的订单主表记录、订单明细记录和订单地址记录，同时写入一条“用户提交订单，等待支付”的状态日志。

7.4 视图实现

本模块对应视图为 checkout.html。该模板用于展示待结算商品列表、地址选择区域、金额汇总信息和结算提交入口。页面中同时展示商品总数量、商品总金额、运费和应付金额，并在地址缺失、库存不足或订单创建失败时显示对应错误信息。

结算页采用服务端渲染方式组织内容，地址选择和提交订单均通过表单完成，不依赖复杂前端脚本。页面的主要作用是展示结算信息并承接正式下单操作。

7.5 模块实现特点

本模块的特点是将购物车数据转换为正式订单数据，并在结算阶段完成按店铺拆单处理。这样可以使多店铺商品在一次结算中生成多笔独立订单，同时保留统一的地址快照和状态初始化过程。

8 订单支付模块设计

订单支付模块用于完成待支付订单的单笔支付与批量支付，是结算完成后进入正式交易状态的核心模块。该模块将钱包余额校验、商品状态校验、库存扣减、订单状态更新、发货记录初始化和支付流水写入组织在同一流程中。

8.1 功能实现

本模块主要实现单订单支付和批量支付两项功能。用户在订单详情页中可对单个待支付订单发起付款，也可以在批量付款页中一次支付多笔待支付订单。支付成功后，订单状态由待支付变为已支付待发货，并同步写入钱包流水、订单状态日志和发货初始记录。

8.2 控制器设计

本模块主要由 batch_pay_orders() 和 order_pay() 两个控制器实现。

其中，order_pay() 用于处理单个订单支付。控制器先读取当前用户对应的目标订单，并检查订单是否仍处于待支付状态；随后读取用户钱包余额，判断余额是否充足；在支付前再次查询订单商品，校验商品是否仍然上架、库存是否足够。校验通过后，控制器依次完成钱包扣款、钱包流水写入、商品库存扣减、订单状态更新、发货记录写入以及订单状态日志写入。

batch_pay_orders() 用于处理多笔订单的统一支付。控制器先读取当前用

户全部待支付订单及其商品明细，再汇总总支付金额并检查钱包余额；随后逐笔处理待支付订单，对每个订单分别执行商品状态校验、库存校验、钱包流水写入、库存扣减、订单状态更新、发货记录初始化和状态日志写入。批量支付完成后，控制器统一更新钱包余额，并跳转至订单列表页。

8.3 Model 设计

本模块主要使用 orders、order_items、wallets、wallet_transactions、products、shipments 和 order_status_logs 等表。订单表用于读取当前待支付订单并更新支付后状态，订单明细表用于读取订单中的商品项，钱包表和钱包流水表用于完成金额扣减与流水记录，商品表用于完成库存扣减，发货表用于创建初始发货记录，状态日志表用于记录支付后的状态变化。

在数据处理上，支付并不是单纯修改订单状态，而是同时推动多张表的数据变化。钱包余额、商品库存、订单状态、发货状态和状态日志在同一支付流程中同步更新，使订单支付结果能够在数据库层面完整体现。

8.4 视图实现

本模块对应的主要视图为 batch_pay.html。该模板用于展示当前待支付订单列表、订单明细和总支付金额，并提供统一支付入口。单订单支付则不单独设置独立页面，而是通过订单详情页中的支付入口进入后端支付流程。

批量支付页以服务端渲染方式组织内容，页面中主要展示待支付订单与金额汇总信息。支付校验失败时，控制器会重新返回批量支付页，并显示余额不足、商品下架或库存不足等错误提示。

8.5 模块实现特点

本模块的特点是将支付处理与库存扣减、发货初始化和状态流转放在同一流程中完成。单订单支付与批量支付共用相同的核心处理逻辑，只是在处理对象数量上有所区别。

9 订单查询与订单流转模块设计

订单查询与订单流转模块用于完成用户订单列表展示、订单详情展示、确认收货和订单状态推进，是订单生成与支付完成后的后续管理模块。该模块同时处理订单状态显示、倒计时展示和自动流转逻辑。

9.1 功能实现

本模块主要实现订单列表查看、订单详情查看、确认收货、超时未支付自动取消、超时未发货自动取消退款和已发货订单自动签收。用户可以在订单列表页查看全部订单，在订单详情页查看商品明细、收货地址、物流信息和状态日志，

并在订单符合条件时执行确认收货。

9.2 控制器设计

本模块主要由 orders()、order_detail() 和 order_confirm_receipt() 三个控制器实现,并配合 apply_order_timeouts()、build_order_view()、cancel_textttorder() 和 settle_order_to_merchants() 完成状态处理。

其中，orders() 用于读取当前用户全部订单，并在查询前先执行一次订单状态检查。控制器读取订单主表、订单明细统计和最新发货状态，再通过 build_order_view() 将原始结果整理为页面展示数据，包括状态文本、倒计时信息和是否允许确认收货等字段。

order_detail() 用于读取单个订单的完整信息。控制器在查询订单主体后，继续读取订单商品项、订单地址快照、最新发货记录和状态日志，并统一返回订单详情页。这样处理后，订单详情页能够完整展示单笔订单的业务过程。

order_confirm_receipt() 用于处理确认收货。控制器在执行前先检查订单当前状态和发货状态，只有在已发货且满足收货条件时才允许继续处理；确认成功后，订单会进入完成状态，发货记录更新为已签收，同时执行商家结算和状态日志写入。

9.3 Model 设计

本模块主要使用 orders、order_items、order_addresses、order_status_logs、shipments、wallets 和 wallet_transactions 等表。订单表用于保存订单主体状态，订单明细表用于保存商品项，订单地址表用于保存下单时地址快照，状态日志表用于保存状态变更记录，发货表用于保存物流状态，钱包表和钱包流水表用于完成退款和商家结算。

在状态推进过程中，系统并不只修改订单表中的状态值，而是同时更新相关联的数据表。超时未支付时，订单直接取消；超时未发货时，订单取消、库存恢复并执行退款；订单确认收货或自动签收后，订单进入完成状态，发货记录更新为已签收，同时按商家汇总金额写入商家钱包和钱包流水。

9.4 视图实现

本模块对应视图为 orders.html 和 order_detail.html。订单列表页主要展示订单号、商品数量、订单金额、创建时间、支付时间、当前状态和操作入口；订单详情页进一步展示订单商品明细、收货地址、物流信息和状态日志。

两个页面都包含倒计时展示逻辑。模板通过 data-deadline 保存截止时间，前端脚本按秒刷新剩余时间，用于展示支付剩余时间、发货剩余时间或自动签收剩余时间。订单状态展示采用服务端和前端结合方式，状态文本和截止时间由后端生成，倒计时刷新由页面脚本完成。

9.5 模块实现特点

本模块的特点是将订单查询与订单状态流转放在同一模块中处理。订单列表和订单详情负责展示当前状态，确认收货、自动取消、自动退款和自动签收负责推动状态变化，二者共同构成完整的订单后续处理链路。

10 钱包模块设计

钱包模块用于展示用户当前余额和资金流水，是订单支付、退款和商家结算后的资金查看模块。该模块不直接处理支付动作，而是负责对钱包状态进行集中展示。

10.1 功能实现

本模块主要实现钱包余额查看和钱包流水查看。用户进入钱包页面后，可以查看当前余额、最近更新时间以及全部交易流水，包括流水类型、金额、变动后余额、关联订单和备注信息。

10.2 控制器设计

本模块主要由 `wallet()` 控制器实现。控制器首先取得当前业务用户编号，并调用 `get_or_create_wallet()` 保证当前用户存在钱包主体；随后查询钱包表中的余额与更新时间，再查询该钱包对应的全部流水记录，并按流水编号倒序返回页面。

这样处理后，钱包页面始终围绕当前用户已有或新建的钱包对象进行展示，不需要在页面访问前额外创建钱包数据。

10.3 Model 设计

本模块主要使用 `wallets` 和 `wallet_transactions` 两张表。`wallets` 表用于保存用户钱包主体信息，包括钱包编号、所属用户、当前余额和更新时间；`wallet_transactions` 表用于保存每一笔钱包变动记录，包括流水类型、变动金额、变动后余额、关联订单、备注和创建时间。

在数据组织上，钱包表负责保存当前状态，钱包流水表负责保存变化过程，两者共同组成钱包模块的数据基础。

10.4 视图实现

本模块对应视图为 `wallet.html`。模板页面主要展示钱包余额、更新时间和流水列表。流水区域按时间倒序展示交易记录，并保留金额、余额变化结果、关联订单和备注等信息。页面采用服务端渲染方式组织内容，不依赖复杂前端交互脚本。

10.5 模块实现特点

本模块的特点是将钱包当前状态与钱包变动过程分开展示。余额信息用于反映用户当前可用资金，流水信息用于反映支付、退款和结算后的资金变化记录。

11 用户主页模块设计

用户主页模块用于展示当前用户的基础信息，并提供昵称修改与头像上传功能，是前台用户资料维护模块。该模块围绕当前业务用户展开，不涉及订单处理和支付流程。

11.1 功能实现

本模块主要实现用户资料查看、昵称修改和头像上传。用户进入用户主页后，可以查看用户名、邮箱、昵称、头像、账号状态、注册时间以及当前角色标记，并可在页面中直接修改昵称或上传新头像。

11.2 控制器设计

本模块主要由 account() 控制器实现。控制器先通过当前业务会话取得用户编号，再查询用户表中的资料信息，并结合角色判断结果组织页面展示数据。

当请求方式为 POST 时，控制器根据表单中的 action 字段区分不同操作。若操作类型为修改昵称，则先对昵称格式进行校验，校验通过后更新用户表中的昵称字段，并同步更新当前会话中的显示名称；若操作类型为上传头像，则先读取旧头像地址，再调用头像保存函数处理上传文件，写入新的头像地址后同步更新会话中的头像信息。当前版本中，头像更新后还会检查旧头像是否为系统默认头像，若不是默认头像则删除旧文件。

11.3 Model 设计

本模块主要使用 users、roles 和 user_roles 三张表。users 表用于保存用户名、邮箱、昵称、头像地址、状态和注册时间等基础资料；角色表和用户角色关联表用于读取当前用户是否具有管理员或商家身份。

在数据处理上，昵称修改和头像修改都直接更新用户表中的资料字段，而角色信息不在该模块中修改，只作为页面展示和顶部导航判断的依据。

11.4 视图实现

本模块对应视图为 account.html。该模板用于展示用户基本资料，并提供昵称修改表单和头像上传表单。页面采用服务端渲染方式组织内容，表单提交后由控制器直接处理更新结果，再重新返回用户主页。

页面标题为“用户主页”，模板中同时保留前台统一导航和当前用户入口，使

用户可以在个人资料维护和前台浏览页面之间直接切换。

11.5 模块实现特点

本模块的特点是将用户资料展示与资料修改放在同一页面中处理，并通过业务会话同步昵称和头像变化结果，使资料更新后能够立即反映到站点顶部导航等显示区域。

12 商家入驻申请模块设计

商家入驻申请模块用于完成普通用户向商家身份的转换申请，是商家后台功能的入口模块。该模块负责收集申请信息、展示当前申请状态，并在审核通过后为用户提供进入商家后台的入口。

12.1 功能实现

本模块主要实现商家入驻申请提交和申请状态查看。普通用户进入申请页后，可以填写店铺名称、联系人、联系电话、经营范围和申请说明；提交后，申请信息进入待审核状态。若当前用户已有申请记录，页面会优先展示当前申请状态、审核备注和提交时间；若申请已通过，则页面直接提供进入商家后台的入口。

12.2 控制器设计

本模块主要由 `merchant_apply()` 控制器实现。控制器首先判断当前用户是否已经具备商家角色，若已是商家，则直接跳转到商家后台主页；否则继续查询当前用户最近一次申请记录，并将其作为页面展示依据。

当请求方式为 `POST` 时，控制器接收表单中的店铺名称、联系人、联系电话、经营范围和申请说明。若当前用户已有待审核申请，则不允许重复提交；若必填项为空或输入长度超出限制，则返回错误提示；校验通过后，控制器向申请表写入一条新的待审核记录，并重新读取最新申请结果返回页面。

12.3 Model 设计

本模块主要使用 `merchant_applications` 表，同时结合 `users`、`roles` 和 `user_roles` 判断当前用户身份。申请表用于保存用户提交的店铺名称、联系人、联系电话、经营范围、申请说明、审核状态、审核备注和时间信息。

在数据处理上，申请记录与正式商家数据分开保存。当前模块只负责向申请表写入申请信息，并读取申请状态；是否成为正式商家，不在本模块中直接处理，而由后续审核模块完成。

12.4 视图实现

本模块对应视图为 merchant_apply.html。模板页面由申请说明区、当前申请状态区和申请表单区组成。页面先展示当前申请状态，若不存在待审核申请或当前申请已结束，则继续显示申请表单；若申请已通过，则在状态区提供进入商家后台的按钮。

页面采用服务端渲染方式组织内容，申请提交通过普通表单完成，不依赖复杂前端脚本。

12.5 模块实现特点

本模块的特点是将“申请过程数据”与“正式商家数据”分开处理，并在同一页面中同时组织申请填写与申请状态展示。这样处理后，普通用户可以通过统一入口完成商家申请，并持续查看当前审核结果。

13 管理员后台首页与站点配置模块设计

管理员后台首页与站点配置模块用于展示平台基础统计信息，并提供站点品牌信息维护功能，是管理员后台中的基础模块。该模块分别对应管理员主页和站点设置页两个入口。

13.1 功能实现

本模块主要实现平台统计信息展示和站点品牌信息维护。管理员进入后台首页后，可以查看商家申请总数、用户总数和订单总数；进入站点设置页后，可以修改网站名称、首页头顶名称、浏览器标题以及网站图标。

13.2 控制器设计

本模块主要由 admin_index() 和 admin_site_branding() 两个控制器实现，并统一通过 admin_required 进行权限控制。

其中，admin_index() 负责读取平台统计数据。控制器分别查询商家申请数量、用户数量和订单数量，并将结果传递给管理员主页模板。页面主要承担后台概览作用。

admin_site_branding() 负责读取和保存站点品牌配置。控制器在页面进入时先加载当前站点配置，在提交保存时接收网站名称、头顶名称和浏览器标题，并在长度校验通过后写入配置数据。若管理员上传了新图标，控制器会先保存新图标，再更新图标路径；当旧图标属于系统管理文件时，还会同步删除旧文件。

13.3 Model 设计

本模块主要使用 merchant_applications、users 和 orders 三张表，以及站点品牌配置数据。商家申请表、用户表和订单表用于提供管理员主页中的统计结果；站点品牌配置数据用于保存网站名称、首页头顶名称、浏览器标题和网站图标路径。

其中，后台首页主要围绕统计数据读取，站点设置页主要围绕配置数据读写，不直接涉及订单处理或用户资料修改。

13.4 视图实现

本模块对应视图为 admin_index.html 和 admin_site_branding.html。管理员主页模板主要展示后台导航和统计卡片，用于集中展示平台当前基本情况。站点设置模板主要展示配置表单，表单中包含网站名称、头顶名称、浏览器标题、当前图标预览和新图标上传入口。

站点设置页中包含少量前端脚本，用于在表单提交时检查图标文件大小，若上传文件超过限制，则阻止提交并给出提示。

13.5 模块实现特点

本模块的特点是将后台概览与站点配置集中在管理员侧统一处理。管理员主页负责展示平台整体情况，站点设置页负责维护全站通用品牌信息，二者共同构成管理员后台的基础入口。

14 商家申请审核模块设计

商家申请审核模块用于处理普通用户提交的商家入驻申请，是管理员后台中的审核模块。该模块负责展示申请列表、执行审核通过与审核拒绝操作，并在审核通过后完成正式商家与店铺数据的建立。

14.1 功能实现

本模块主要实现申请列表查看、审核通过和审核拒绝三项功能。管理员进入审核页面后，可以查看申请编号、申请用户、店铺名称、联系人、联系电话、经营范围、申请理由、申请状态和提交时间等信息，并可对待审核申请执行通过或拒绝操作。

14.2 控制器设计

本模块主要由 admin_merchant_applications()、approve_merchant_application() 和 reject_merchant_application() 三个控制器实现，并统一通过 admin_required 进行权限控制。

其中，admin_merchant_applications() 负责读取全部申请记录，并按“待审核优先、申请编号倒序”的方式组织列表结果。页面进入后，管理员可以直接查看当前所有申请的审核状态。

approve_merchant_application() 用于处理审核通过。控制器先读取目标申请并确认其仍处于待审核状态，随后检查申请用户是否已经具备商家角色；若尚未具备，则先为其补充商家角色，再创建或更新商家主体信息，并继续创建或更新对应店铺信息。最后，控制器将申请状态更新为已通过，同时写入审核备注、审核人和审核时间。

reject_merchant_application() 用于处理审核拒绝。控制器接收管理员填写的审核备注，在确认目标申请仍为待审核后，将其状态更新为已拒绝，并写入审核备注、审核人和审核时间。

14.3 Model 设计

本模块主要使用 merchant_applications、merchants、shops、roles 和 user_roles 等表。申请表用于保存用户提交的入驻申请信息；商家表和店铺表用于保存审核通过后的正式经营数据；角色表和用户角色关联表用于完成商家角色授予。

在数据处理上，审核通过并不只是修改申请状态，而是同时推动角色、商家和店铺三部分数据建立。审核拒绝则仅更新申请表中的审核结果和备注信息。

14.4 视图实现

本模块对应视图为 admin_merchant_applications.html。模板页面主要由申请说明区和申请列表区组成。每条申请记录以卡片形式展示，并在待审核状态下提供“审核通过”和“审核拒绝”两个操作入口；拒绝操作同时提供审核备注输入框。

页面采用服务端渲染方式组织内容，审核通过和审核拒绝均通过表单提交完成。

14.5 模块实现特点

本模块的特点是将申请审核结果与后续商家数据建立过程直接连接。审核通过后，用户角色、商家主体和店铺主体同步建立；审核拒绝后，申请记录保留审核结果与备注信息。

15 管理员钱包调账模块设计

管理员钱包调账模块用于处理平台侧的余额增减操作，是管理员后台中的资金管理模块。该模块支持对单个用户或全部普通用户、商家用户执行统一加款或扣款，不面向管理员自身开放。

15.1 功能实现

本模块主要实现调账对象选择、调账类型选择、金额输入、自定义说明填写和调账结果反馈。管理员进入页面后，可以先选择“单个用户”或“全部用户”，再选择“加款”或“扣款”，填写金额与说明后提交处理。页面同时展示全部可操作用户的当前余额，便于管理员在调账前进行核对。

15.2 控制器设计

本模块主要由 admin_wallet_adjustments() 控制器实现，并通过 admin_required 进行权限控制。控制器进入页面时，先读取当前管理员身份信息，再查询全部非管理员用户及其余额，用于页面展示。

当请求方式为 POST 时，控制器首先读取调账目标模式、目标用户、调账类型、金额和自定义说明，并对输入内容进行校验，包括操作目标合法性、操作类型合法性、金额范围和说明长度。校验通过后，若目标模式为单个用户，则只读取目标用户；若目标模式为全部用户，则读取全部可操作用户。

在实际执行过程中，控制器逐个取得目标用户的钱包对象，并根据当前操作类型决定增加或减少余额。若执行扣款且当前余额不足，则该用户被跳过；若执行成功，则更新钱包余额并写入一条钱包流水。全部处理完成后，控制器提交事务，并返回成功数量或跳过信息。若中途出现异常，则整体回滚并返回失败提示。

15.3 Model 设计

本模块主要使用 users、roles、user_roles、wallets 和 wallet_transactions 等表。用户表、角色表和用户角色关联表用于筛选全部可操作的非管理员用户；钱包表用于保存当前余额；钱包流水表用于记录调账后的资金变化结果。

在数据处理上，调账功能并不直接修改用户表，而是围绕钱包主体与钱包流水展开。每次调账都会同时更新余额和流水记录，使管理员加款、扣款操作能够在数据库层面完整保留。

15.4 视图实现

本模块对应视图为 admin_wallet_adjustments.html。模板页面主要包含调账表单和用户列表两部分。表单区域用于选择调账对象、输入金额和说明；列表区域用于展示当前可操作用户的昵称、头像和余额信息。

页面采用服务端渲染方式组织内容，调账操作通过表单提交完成。处理结果通过页面消息区域返回，包括成功提示、错误提示以及余额不足的跳过说明。

15.5 模块实现特点

本模块的特点是将单用户调账和批量调账放在同一页面中处理，并通过“更新余额 + 写入流水”的方式记录管理员资金操作。扣款时同时保留余额不足跳

过逻辑，使批量调账过程能够继续执行。

16 商家商品管理模块设计

商家商品管理模块用于完成商家商品列表查看和商品上下架控制，是商家后台中的基础经营模块。该模块以当前商家所属店铺为范围组织商品数据，不处理商品新增与编辑，只负责已有商品的集中管理。

16.1 功能实现

本模块主要实现商品列表查看和商品状态切换。商家进入商品管理页后，可以查看当前店铺下全部商品的名称、价格、库存、分类、主图、创建时间、更新时间和当前状态，并可对单个商品执行上架或下架操作。

16.2 控制器设计

本模块主要由 merchant_products() 和 merchant_product_toggle_status() 两个控制器实现，并统一通过 merchant_required 进行权限控制。

其中，merchant_products() 在进入页面后，先读取当前登录商家的商家编号和店铺编号，再按商家和店铺范围查询商品列表。查询结果中同时带出分类名称和商品主图，用于页面统一展示。

merchant_product_toggle_status() 用于处理商品状态切换。控制器先确认目标商品属于当前商家和当前店铺，再通过更新语句在上架状态和下架状态之间切换，并同步更新商品修改时间。处理完成后返回商品管理页。

16.3 Model 设计

本模块主要使用 products、categories 和 product_media 三张表。商品表用于保存商品名称、价格、库存、状态和时间信息，分类表用于提供分类名称，商品图片表用于读取商品主图。查询条件同时受商家编号和店铺编号限制，用于保证商家只能管理自身商品。

在数据处理上，商品管理页不修改商品主体内容，只负责读取商品当前状态并执行状态切换，因此该模块的数据操作主要集中在商品表的状态字段和更新时间字段上。

16.4 视图实现

本模块对应视图为 merchant_products.html。模板页面主要展示商品列表，并在每条商品记录下提供编辑入口和状态切换入口。页面顶部保留商家后台导航，同时提供新增商品入口，便于商家从商品管理页继续进入商品新增流程。

页面采用服务端渲染方式组织内容，商品状态切换通过表单提交完成，不依赖复杂前端交互脚本。

16.5 模块实现特点

本模块的特点是将商品集中展示与商品状态控制放在同一页面中处理，并通过“商家编号 + 店铺编号”双重范围限制商品管理对象，使商家后台的商品管理边界保持明确。

17 商品新增与编辑模块设计

商品新增与编辑模块用于完成商家商品主体信息维护，是商家后台中最核心的商品编辑模块。该模块支持商品新增、商品编辑、主图上传和商品说明维护，并通过统一表单页组织新增与编辑流程。

17.1 功能实现

本模块主要实现新增商品、编辑商品、上传商品主图和维护商品说明。商家进入新增页后，可以填写商品名称、分类、价格、库存、商品说明并上传主图；进入编辑页后，可以在原有商品数据基础上继续修改上述内容。

17.2 控制器设计

本模块主要由 `merchant_product_create()` 和 `merchant_product_edit()` 两个控制器实现，并统一通过 `merchant_required` 进行权限控制。表单数据处理主要由 `parse_product_form()` 完成，商品主图更新由 `upsert_product_cover_media()` 完成。

其中，`merchant_product_create()` 在进入页面后先读取当前商家和店铺信息，并加载可选分类列表。提交表单时，控制器先调用表单解析函数校验商品名称、分类、价格、库存和商品说明；校验通过后向商品表写入新记录，再根据是否上传主图决定写入默认主图或上传后的主图地址，最后返回商品管理页。

`merchant_product_edit()` 在进入页面时先读取目标商品，并确认该商品属于当前商家和当前店铺。提交表单时，控制器同样先执行表单解析，再更新商品表中的分类、名称、说明、价格、库存和更新时间；若本次同时上传了新的商品主图，则继续更新商品媒体表中的封面图记录。

17.3 Model 设计

本模块主要使用 `products`、`categories` 和 `product_media` 三张表。商品表用于保存商品主体信息，分类表用于提供分类选项，商品媒体表用于保存商品主图。新增商品时先写入商品主表，再补充封面图记录；编辑商品时先更新商品主体字段，再按需更新商品主图。

在数据处理上，商品说明与商品主图分开维护。商品说明直接写入商品表中的说明字段，商品主图则通过商品媒体表单独保存。这样处理后，商品主体信息和展示资源保持分离。

17.4 视图实现

本模块对应视图为 merchant_product_form.html。该模板同时服务于新增页和编辑页，通过页面标题和表单初始值区分两种模式。页面中包含商品名称、商品分类、价格、库存、商品说明和商品主图上传区域。

在前端交互上，商品说明区域使用 Quill 富文本编辑器。页面脚本在加载时读取已有说明内容作为初始值，并在表单提交前将编辑器中的 HTML 内容写入隐藏字段后再提交到后端。这样处理后，商品说明可以直接以富文本形式保存。

17.5 模块实现特点

本模块的特点是将商品新增和商品编辑统一在同一表单模板中处理，并将商品说明和商品主图分开维护。这样既保证了商品主体数据录入的一致性，也便于在编辑过程中单独更新商品展示内容。

18 商品新增与编辑模块设计

商品新增与编辑模块用于完成商家商品主体信息维护，是商家后台中最核心的商品编辑模块。该模块支持商品新增、商品编辑、主图上传和商品说明维护，并通过统一表单页组织新增与编辑流程。

18.1 功能实现

本模块主要实现新增商品、编辑商品、上传商品主图和维护商品说明。商家进入新增页后，可以填写商品名称、分类、价格、库存、商品说明并上传主图；进入编辑页后，可以在原有商品数据基础上继续修改上述内容。

18.2 控制器设计

本模块主要由 merchant_product_create() 和 merchant_product_edit() 两个控制器实现，并统一通过 merchant_required 进行权限控制。表单数据处理主要由 parse_product_form() 完成，商品主图更新由 upsert_product_cover_media() 完成。

其中，merchant_product_create() 在进入页面后先读取当前商家和店铺信息，并加载可选分类列表。提交表单时，控制器先调用表单解析函数校验商品名称、分类、价格、库存和商品说明；校验通过后向商品表写入新记录，再根据是否上传主图决定写入默认主图或上传后的主图地址，最后返回商品管理页。

merchant_product_edit() 在进入页面时先读取目标商品，并确认该商品属于当前商家和当前店铺。提交表单时，控制器同样先执行表单解析，再更新商品表中的分类、名称、说明、价格、库存和更新时间；若本次同时上传了新的商品主图，则继续更新商品媒体表中的封面图记录。

18.3 Model 设计

本模块主要使用 products、categories 和 product_media 三张表。商品表用于保存商品主体信息，分类表用于提供分类选项，商品媒体表用于保存商品主图。新增商品时先写入商品主表，再补充封面图记录；编辑商品时先更新商品主体字段，再按需更新商品主图。

在数据处理上，商品说明与商品主图分开维护。商品说明直接写入商品表中的说明字段，商品主图则通过商品媒体表单独保存。这样处理后，商品主体信息和展示资源保持分离。

18.4 视图实现

本模块对应视图为 merchant_product_form.html。该模板同时服务于新增页和编辑页，通过页面标题和表单初始值区分两种模式。页面中包含商品名称、商品分类、价格、库存、商品说明和商品主图上传区域。

在前端交互上，商品说明区域使用 Quill 富文本编辑器。页面脚本在加载时读取已有说明内容作为初始值，并在表单提交前将编辑器中的 HTML 内容写入隐藏字段后再提交到后端。这样处理后，商品说明可以直接以富文本形式保存。

18.5 模块实现特点

本模块的特点是将商品新增和商品编辑统一在同一表单模板中处理，并将商品说明和商品主图分开维护。这样既保证了商品主体数据录入的一致性，也便于在编辑过程中单独更新商品展示内容。

19 店铺主页管理模块设计

店铺主页管理模块用于完成店铺公开主页展示和商家侧主页维护，是商家后台中的店铺展示模块。该模块一端面向普通用户公开展示店铺信息，另一端面向商家提供店铺主页编辑入口。

19.1 功能实现

本模块主要实现店铺主页展示和店铺主页编辑。普通用户进入店铺主页后，可以查看店铺名称、Logo、简介、联系方式、销量、营业额、完成订单数和在售商品；商家进入编辑页后，可以修改店铺名称、简略描述、联系电话、店铺主页描述和店铺 Logo。

19.2 控制器设计

本模块主要由 shop_home() 和 merchant_shop_edit() 两个控制器实现，并配合 get_shop_home_payload() 完成店铺主页数据组织。

其中，shop_home() 用于处理公开店铺主页访问。控制器先根据店铺访问标识调用 get_shop_home_payload() 读取店铺主体信息、统计数据和在售商品，再将结果传递给店铺主页模板进行展示。

merchant_shop_edit() 用于处理商家侧店铺主页维护。控制器先读取当前商家上下文和对应店铺信息，再在提交时接收店铺名称、简略描述、联系电话、主页描述和 Logo 文件。控制器会对名称长度、简略描述长度和联系电话长度进行校验，随后更新店铺表中的主页相关字段，并同步更新商家表中的店铺名称和简介。若本次上传了新的店铺 Logo，则先保存图片，再更新店铺图标地址。

19.3 Model 设计

本模块主要使用 shops、merchants、products、categories、product_media、orders 和 order_items 等表。店铺表和商家表用于保存店铺名称、Logo、简介、主页描述和联系方式，商品表、分类表与商品图片表用于组织店铺商品展示，订单表和订单明细表用于统计销量、营业额和完成订单数量。

在数据处理上，店铺主页描述并不直接原样输出，而是在读取展示时经过清洗后再返回页面；店铺主页中的统计信息和商品列表也不是静态保存，而是在访问时根据当前订单和商品数据实时组织。

19.4 视图实现

本模块对应视图为 shop_home.html 和 merchant_shop_edit.html。店铺主页模板主要展示店铺基础信息、统计信息、店铺主页描述和店铺商品列表。编辑模板则主要提供店铺名称、简略描述、联系电话、店铺 Logo 和店铺主页描述的输入区域。

在前端交互上，店铺主页编辑页使用 Quill 富文本编辑器维护店铺主页描述。页面脚本在加载时读取已有主页内容作为初始值，在提交时再将编辑器中的 HTML 内容写入隐藏字段后提交到后端；工具栏同时支持图片上传，上传请求通过 /merchant/editor/upload-image 接口完成。

19.5 模块实现特点

本模块的特点是将店铺公开展示和店铺后台维护放在同一数据结构下处理。公开主页负责展示店铺当前经营结果，编辑页负责维护店铺展示内容，二者共同围绕店铺表中的主页数据展开。

20 全部店铺页面模块设计

全部店铺页面模块用于集中展示站内当前可公开访问的店铺，是首页推荐店铺之外的完整店铺浏览入口。该模块负责按统一规则组织店铺列表，并提供进入单个店铺主页的跳转入口。

20.1 功能实现

本模块主要实现全部店铺展示和店铺排序展示。用户进入页面后，可以查看当前正常营业店铺的店铺名称、Logo、简介、已售件数、成交额和成交订单数，并可直接进入对应店铺主页。

20.2 控制器设计

本模块主要由 `shops()` 控制器实现。控制器先查询当前状态正常的店铺，并结合订单明细和订单表统计每个店铺的成交额、销量和成交订单数；随后按成交额、销量和店铺编号进行排序，再将结果整理为页面所需的数据结构后传递给模板渲染。

在数据处理过程中，店铺简介会先经过长度截断处理，用于适配列表页展示；成交额、销量和成交订单数则在查询阶段一并完成统计。

20.3 Model 设计

本模块主要使用 `shops`、`merchants`、`order_items` 和 `orders` 等表。店铺表用于读取店铺主体信息，商家表用于校验店铺所属商家状态，订单表和订单明细表用于统计店铺成交额、销量和成交订单数量。

其中，页面只展示状态正常的店铺，同时要求关联商家处于可用状态。这样可以保证全部店铺页中的结果与当前前台可访问店铺保持一致。

20.4 视图实现

本模块对应视图为 `shops.html`。模板页面主要包含页面标题区和店铺列表区。每个店铺以卡片形式展示，卡片中包含店铺 Logo、店铺名称、简介以及三项统计指标，并通过链接跳转到对应店铺主页。

页面采用服务端渲染方式组织内容，不依赖复杂前端交互脚本。

20.5 模块实现特点

本模块的特点是将店铺主体信息与店铺经营统计结果集中展示，并作为首页推荐店铺之外的完整店铺浏览入口。

21 富文本图片上传功能设计

富文本图片上传功能用于支持商家在商品说明和店铺主页描述中插入图片，是富文本编辑能力中的配套功能。该功能通过独立上传接口处理编辑器中的图片文件，再将返回的图片地址插入到当前富文本内容中。

21.1 功能实现

本功能主要实现说明图片上传和图片地址回填。商家在富文本编辑器中点击图片按钮后，可以选择本地图片文件；图片上传成功后，系统返回可访问的图片地址，并将该地址直接插入当前编辑内容中。该功能同时服务于商品说明编辑和店铺主页描述编辑。

21.2 控制器设计

本功能主要由 merchant_editor_upload_image() 控制器实现。该控制器对应路由 /merchant/editor/upload-image，接收编辑器上传的图片文件，并以 JSON 形式返回处理结果。

控制器在处理时，先确认当前用户具有商家权限，再从请求中读取图片文件，并调用 save_product_content_image() 完成图片保存。上传成功后，控制器返回 url 和 data.url 两种字段格式，供编辑器脚本直接插入图片；上传失败时，则返回错误信息。

21.3 Model 设计

本功能不依赖独立数据表，而是将上传后的图片文件保存到商品内容图片目录中，再把生成的访问地址写入富文本内容。图片文件本身不进入数据库表单独维护，而是作为富文本内容中的资源地址存在。

在文件处理上，系统会检查图片格式、图片尺寸和像素数量，并在保存时统一生成新的文件名。这样处理后，富文本中的图片资源能够通过统一路径访问。

21.4 视图实现

本功能主要配合 merchant_product_form.html 和 merchant_shop_edit.html 中的 Quill 编辑器使用。页面脚本在工具栏中重写了图片按钮行为：当用户选择本地图片后，前端通过 fetch 将文件发送到 /merchant/editor/upload-image，成功后再调用 insertEmbed 将图片地址插入当前编辑位置。

在前端交互上，商品说明编辑页还对图片大小做了额外限制，超过限制时会直接终止上传并给出提示。上传失败时，页面则弹出错误信息。

21.5 模块实现特点

本功能的特点是将图片上传与富文本编辑过程直接结合。编辑器中的图片插入不依赖手工填写地址，而是通过上传接口返回结果后自动写入当前内容。

第四章　安全策略

1　运行时配置与敏感信息管理

系统将会话密钥、数据库连接参数、Firebase 服务账号路径以及应用运行参数从业务代码中分离，统一通过 .env.runtime 在运行时加载。主程序启动时先调用 _load_runtime_env_file() 读取配置，再通过 _env()、_env_bool() 和 _env_int() 提取具体参数，用于生成应用配置、数据库连接配置和外部认证配置。这样可以避免将敏感信息直接写入业务逻辑。

在会话配置方面，系统启用了 SESSION_COOKIE_HTTPONLY、SESSION_COOKIE_SECURE 和 SESSION_COOKIE_SAMESITE，并设置了有限的会话生命周期。该策略用于限制浏览器脚本对会话 Cookie 的直接访问，并约束 Cookie 的传输范围。

在数据库配置方面，系统通过 DB_CONFIG 统一组织数据库服务器地址、数据库名称、用户名、密码和驱动信息，再由 get_conn() 生成连接字符串。数据库连接字符串中启用了加密传输，并通过统一入口完成数据库参数管理。

在外部认证配置方面，系统将 Firebase 服务账号路径和前端 Web 配置项一并纳入运行时配置，在应用初始化阶段完成 Firebase Admin SDK 初始化，并将前端认证所需参数传递给注册、登录和密码重置页面。这样使认证配置与页面逻辑分离，并保持统一管理。

通过上述方式，系统形成了“运行时加载配置、业务代码读取配置、敏感信息不直接写入功能逻辑”的管理策略，使会话配置、数据库配置和外部认证配置都由独立配置入口统一维护。

2　认证、会话与权限控制

系统采用“Firebase 邮箱认证 + 本地兼容登录”的双链路认证方式。邮箱登录时，前端先通过 Firebase 完成身份认证，再将 idToken 提交到 /sessionLogin，由后端校验身份、确认邮箱已完成验证，并写入系统内部业务会话；本地兼容登录则通过 /localLogin 直接读取用户名或邮箱、密码和账号状态进行校验。两条链路最终都会写入统一的业务会话字段，包括用户编号、用户名、昵称、头像、邮箱以及管理员、商家角色标记。

在密码处理上，系统不直接保存明文密码，而是通过 generate_password_hash() 和 check_password_hash() 进行散列生成与校验。针对早期遗留账号，verify_and_upgrade_password() 保留了兼容逻辑：若检测到旧密码仍为明文，则在用户成功登录后自动升级为散列值。这样既兼容旧数据，又将后续密码存储统一到散列形式。

在邮箱认证链路中，系统通过 sync_business_user() 将 Firebase 用户与内

部业务用户进行映射。若当前邮箱对应用户已存在，则直接返回现有业务用户；若不存在，则自动创建用户记录，并补充默认的普通用户角色。这样使外部认证结果能够与系统内部用户体系保持一致。

在登录态恢复方面，系统通过 apply_firebase_login_to_session() 读取后端会话 Cookie，校验 Firebase Session Cookie 后恢复当前业务用户信息；login_required、admin_required 和 merchant_required 则分别用于限制普通登录访问、管理员访问和商家访问。未登录用户会被重定向到登录页，缺少管理员权限的请求直接返回 403，缺少商家权限的请求则跳转到商家申请页。这样形成了登录控制与角色控制分层执行的权限边界。

退出登录通过 /logout 实现。该接口会清空当前业务会话，并删除 Firebase Session Cookie，使登录态在前后端同时失效。

3 CSRF 防护与安全响应头

系统对所有 POST、PUT、PATCH 和 DELETE 请求启用了统一的 CSRF 校验。主程序通过 get_csrf_token() 生成 Token，并同时保存在会话和 Cookie 中；在请求进入时，由 csrf_protect() 从表单字段或请求头读取提交的 Token，再与 Cookie 中的 Token 进行常量时间比较。校验失败时，系统直接返回 400 错误并中止请求。

为减少模板遗漏隐藏字段带来的风险，系统在上下文处理器中提供了 csrf_input(),同时又在 after_request 中通过 csrf_set_cookie_and_inject_forms() 自动为 HTML 页面中的 POST 表单补入隐藏 Token 字段。这样即使模板未手动添加 CSRF 隐藏域，系统仍可以在响应阶段补全表单保护内容。当前仅有 local_login、session_login 和富文本图片上传接口被列入 CSRF 豁免名单。

在安全响应头方面，系统统一为响应添加 X-Frame-Options: DENY、X-Content-Type-Options: nosniff、Referrer-Policy: strict-origin-when-cross-origin、Permissions-Policy 和 Cross-Origin-Resource-Policy: same-origin。这些响应头分别用于限制页面被嵌入、阻止 MIME 嗅探、约束引用来源策略以及收紧浏览器可调用能力。

对于 HTML 页面,系统还统一增加了 Cache-Control: no-store, no-cache, must-revalidate, max-age=0、Pragma: no-cache 和 Expires: 0。该策略用于减少登录页、后台页和交易页等敏感页面被浏览器长期缓存的风险。

4 请求限流与高风险操作防护

系统通过 request_rate_limits 表保存限流状态，并使用 consume_rate_limit()、consume_custom_rate_limit()、rate_limit_guard()、auth_limit_json() 和 auth_limit_html() 对高风险操作进行统一限制。限流键既可以按当前登录用户生成，也可以按客户端 IP 生成；对于登录相关接口，还会额外组合账号信

息，形成更细粒度的限流标识。这样可以在多进程部署场景下保持限流结果的一致性。

在认证链路中，本地登录、本地密码重置和 Firebase 会话建立都启用了专门的限流控制。系统会对短时间内重复提交的登录或重置请求直接返回错误提示或 429 响应，从而降低暴力尝试和高频撞库带来的风险。

在业务链路中，结算、批量支付、单订单支付、商家发货、确认收货、管理员钱包调账和富文本图片上传等操作也被纳入限流范围。这样可以减少重复提交、短时间内连续点击和恶意刷接口对订单状态、库存和资金数据造成的冲击。

系统在限流命中后的处理方式区分为两类：异步接口直接返回 JSON 错误信息，页面型请求则通过消息提示并重定向回来源页面。这样使登录接口、后台接口和前台页面都能够按照自身交互方式给出限流反馈。

5 文件上传校验与资源管理

系统对头像、站点图标、商品主图、店铺 Logo 和富文本内容图片分别设置了独立的上传校验逻辑。校验内容包括扩展名限制、图片有效性验证、尺寸压缩、统一文件名生成和上传目录控制。主程序同时设置了 `MAX_CONTENT_LENGTH`，用于限制单次请求体大小，避免异常大文件直接进入处理流程。

在头像上传中，系统通过 `allowed_avatar_file()` 和 `save_avatar_file()` 限制文件类型为 png、jpg、jpeg，并使用 Pillow 对图片进行校验、缩放和重新编码。站点图标上传同样会先验证图片有效性，再统一转换为 PNG 保存；商品主图、店铺 Logo 和富文本内容图片上传也分别使用独立函数处理，从而将不同资源类型的上传规则隔离开来。

富文本图片上传通过独立接口 `/merchant/editor/upload-image` 实现。前端编辑器在用户选择图片后，通过 `fetch` 将文件发送到后端；后端保存成功后返回图片地址，再由编辑器插入到当前内容中。商品说明编辑页和店铺主页编辑页都采用这一方式完成图片插入。

在资源管理方面，系统对受管上传文件提供了旧文件清理逻辑。头像更新后，若旧头像并非默认头像，则会删除旧文件；站点图标更新时，若旧图标属于系统生成文件，也会同步清理原有资源。这样可以减少资料更新和品牌设置变更后残留的无效文件。

6 富文本内容清洗与输出控制

系统允许商家在商品说明和店铺主页中使用富文本，但不直接信任前端提交的 HTML。后端通过 `sanitize_rich_html()`、`_safe_richtext_url()` 和 `render_product_description()` 对富文本内容进行清洗和受控输出，仅保留允许的标签、属性和链接形式。这样可以降低恶意脚本、危险链接和异常 HTML 直接进入页面渲染的风险。

在内容展示方面，商品详情页和店铺主页虽然需要输出富文本内容，但实际渲染的是清洗后的结果，而不是原始提交内容。系统同时提供 rich_html_to_plain_text() 和 make_product_intro()，将富文本转换为纯文本摘要，用于商品列表页和搜索结果页展示。这样可以避免在列表场景直接输出完整富文本。

在前端编辑方面，商品编辑页和店铺主页编辑页均使用 Quill 富文本编辑器，页面脚本在提交前将编辑器中的 HTML 内容写入隐藏字段，再交由后端统一处理。这样使富文本录入和富文本清洗分别在前端编辑层与后端输出层完成。

7 交易一致性与状态安全

系统将库存校验、钱包扣减、订单状态更新、退款、发货和商家结算组织在一套受控流程中,避免只修改单张表导致交易状态失衡。结算前通过 validate_cart_items_for_checkout() 对购物车商品执行统一校验，包括商品是否上架、库存是否充足以及商家信息是否完整；支付时再次校验商品状态和库存，防止结算后、支付前出现状态变化。

在资金处理上，系统统一通过 wallets 和 wallet_transactions 维护余额与流水。用户支付、订单取消退款、管理员调账和商家结算都不是只更新余额，而是同时写入对应流水记录，使每次资金变化都可追踪。

在订单状态流转方面，系统通过 cancel_order()、settle_order_to_merchants()、apply_order_timeouts()、merchant_ship_order() 和 order_confirm_receipt() 控制不同阶段的状态推进。超时未支付时自动取消订单，超时未发货时自动取消并退款，确认收货或自动签收后再执行商家结算。各阶段都要求订单当前状态满足预期后才允许继续处理。

对于多店铺商品，系统在结算阶段先按店铺拆分，再分别写入订单主表、订单明细表、订单地址快照和状态日志。这样可以保证多商铺商品在一次结算后形成多笔独立订单，后续发货、收货、退款和结算也都在各自订单范围内处理。

8 统一错误响应机制

系统对 400、403、404、429、413 和 500 等常见错误状态设置了统一处理函数，包括 handle_bad_request()、handle_forbidden()、handle_not_found()、handle_too_many_requests()、handle_request_entity_too_large() 和 handle_internal_server_error()。这些错误处理函数最终都通过 render_error_response() 生成返回结果。

在响应组织上，系统通过 wants_json_error_response() 判断当前请求更适合返回 JSON 还是 HTML 页面，再由 build_error_page_payload() 统一组织错误状态码、标题和提示信息。这样可以使异步接口与普通页面访问在出错时分别得到适合自身场景的错误响应。

对于上传文件超限、请求过于频繁和未找到页面等情况，系统不再直接暴露

默认异常,而是统一进入受控错误响应流程。页面型请求会渲染 error_page.html，接口型请求则返回结构化 JSON 信息。这样可以保持错误处理方式一致，并减少默认异常页面直接暴露给用户。

9 后台状态任务与补充防护

除前台请求触发的状态变化外，系统还通过后台任务补充订单超时检查与自动流转。常驻任务 order_state_worker.py 周期性调用 apply_order_timeouts()，并根据待支付订单数量和活跃订单数量动态调整轮询间隔；一次性任务 order_timeout_worker.py 则用于执行单次超时检查。

在常驻任务中，系统先执行订单超时处理，再根据当前订单状态决定下一次休眠时间：若近期存在新的待支付订单，则缩短轮询间隔；若当前没有活跃订单，则延长检查间隔。这样可以在保持后台检查能力的同时，减少无效轮询。

两类后台任务都直接复用主程序中的数据库连接函数和订单状态处理函数，而不单独实现另一套状态机逻辑。这样可以保证前台请求路径与后台轮询路径在订单超时取消、超时未发货退款和自动签收等处理上保持一致，减少状态推进逻辑分叉带来的风险。

第五章　系统运行测试与安装

1　运行环境与系统部署结构

系统部署于 Ubuntu 22.04.5 LTS 服务器环境，应用层使用 Python 3.10.12 开发，Web 服务采用 Nginx 1.18.0 与 Gunicorn 25.3.0 组合部署，数据库采用 SQL Server 2022 Developer Edition for Linux。该运行环境能够同时满足 Flask 动态页面渲染、数据库访问、上传资源管理和 HTTPS 对外服务等需求。

系统对外访问域名为 shopping.vertexf.top。在部署结构上，Nginx 作为前端入口统一监听 `80` 与 `443` 端口，其中 `80` 端口请求被重定向到 HTTPS，`443` 端口负责正式对外服务；Gunicorn 仅绑定在本机回环地址 `127.0.0.1:5000`，用于承载 Flask 主程序；SQL Server 监听 `1433` 端口，负责业务数据存储。这样形成了“浏览器请求 → Nginx → Gunicorn → Flask → SQL Server”的基本运行链路。

系统首页已经可以通过正式域名正常访问，如图 5.1 所示。页面内容能够完整加载，说明域名解析、HTTPS 访问、Nginx 反向代理与 Gunicorn 应用服务之间的联通关系已经建立。

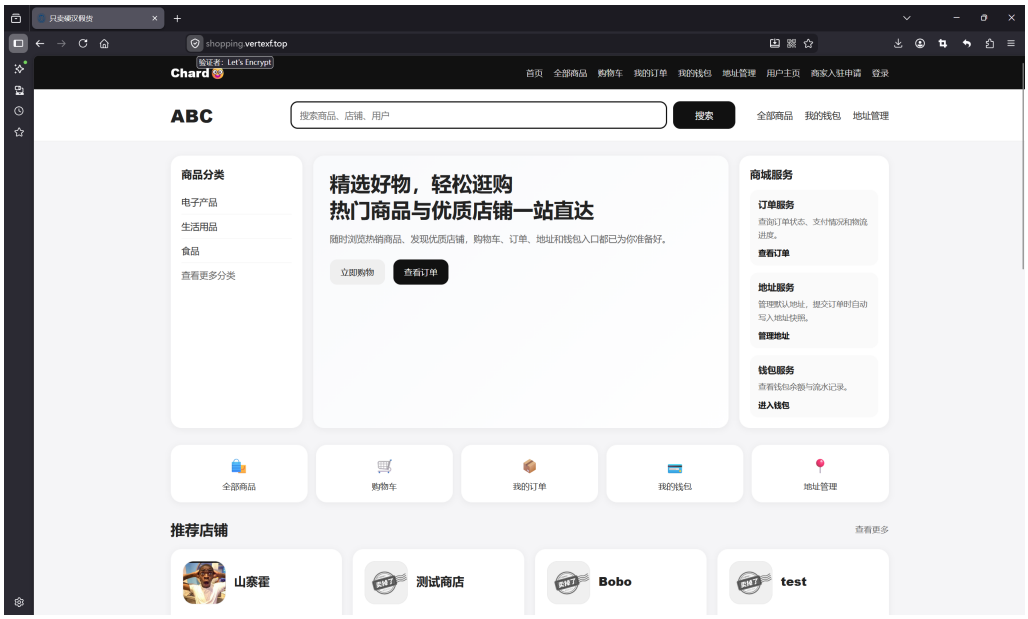

图 5.1: 系统首页通过正式域名访问结果

在静态资源访问方面，系统未将上传文件交由 Flask 主程序直接处理，而是通过 `/uploads/` 路径映射到服务器目录 `/var/www/shopping/uploads/`，由 Nginx 直接提供资源访问服务。这样可以将动态业务请求与静态资源请求分离，降低应用层处理压力。图 5.2 展示了上传资源路径可直接访问的结果，说明当前静态资源映射已经生效。

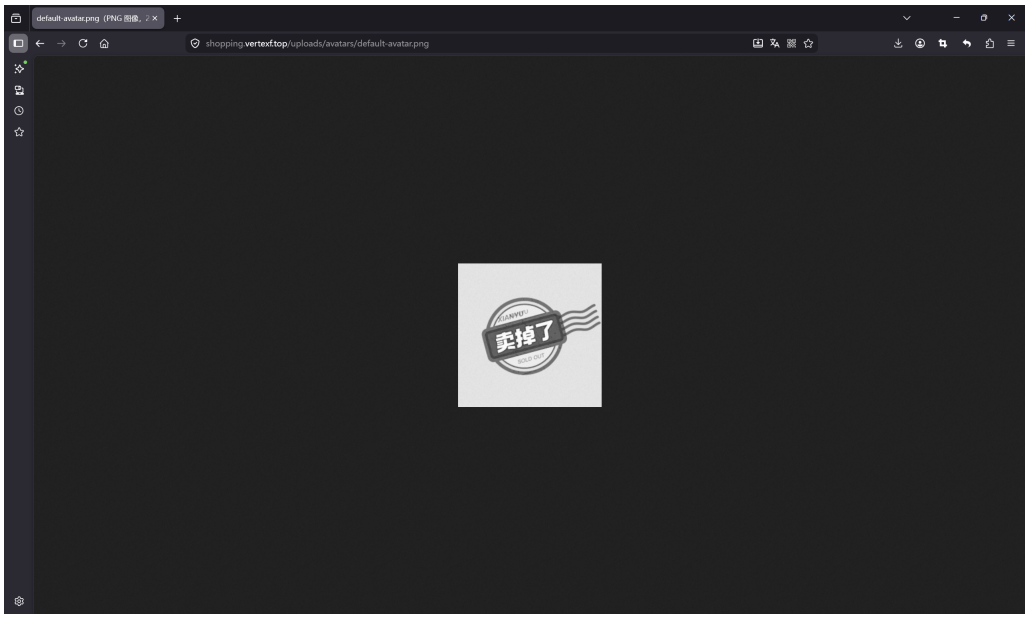

图 5.2: 上传资源路径访问结果

在 HTTPS 配置方面，Nginx 已为 shopping.vertexf.top 配置证书，并将浏览器访问统一提升到加密连接。图 5.3 展示了浏览器对站点安全连接状态的识别结果，表明当前站点已经具备正式的 HTTPS 访问能力。

图 5.3: 站点 HTTPS 安全连接状态

在服务托管方面，系统通过 `shopping_app.service` 管理 Gunicorn 进程，通过 `shopping_order_state.service` 常驻执行订单状态检查任务；`shopping_order_timeout.service` 则作为一次性任务用于手动触发订单超时扫描。这样使主站服务与后台状态处理在运行结构上保持分离，前者负责页面访问与业务请求，后者负责补充订单自动流转逻辑。

系统主程序部署目录为 `/opt/webapps/shopping_app`，其中 `app.py` 作为 Flask 主入口，`templates` 目录保存全部页面模板，`order_state_worker.py` 和 `order_timeout_worker.py` 用于补充订单状态推进。上传资源目录统一位于 `/var/www/shopping/uploads`，并按业务类型划分为 `avatars`、`products`、`shops` 和

site 等子目录，用于分别保存头像、商品图片、店铺图片和站点图标资源。

2 系统安装与启动过程

系统安装与启动过程分为服务器正式部署与 Windows 本地运行两类。两种运行方式共用同一套核心业务代码，但在启动方式和访问入口上有所区别。

服务器端部署采用“Python 虚拟环境 + Gunicorn + Nginx + SQL Server”的组合结构。应用代码部署于 /opt/webapps/shopping_app 目录，依赖安装在项目虚拟环境中，由 shopping_app.service 负责启动 Gunicorn，并将 Flask 主程序绑定到本机 127.0.0.1:5000。Nginx 作为前端入口，统一监听 80 和 443 端口，将根路径请求反向代理到 Gunicorn，同时将 /uploads/ 映射到服务器上传目录。这样使应用服务、静态资源访问和对外入口保持分离。

在正式部署中，系统已将域名绑定到 shopping.vertexf.top，并通过 Certbot 配置 HTTPS 证书。Nginx 配置中同时包含 server_name、proxy_pass、ssl_certificate 和 ssl_certificate_key 等关键项，使站点能够通过正式域名进行加密访问。

除服务器正式部署外，系统还额外提供了 Windows 本地部署包，用于在本地环境快速完成项目初始化和演示运行。本地部署包保留了项目源码、模板、上传资源和启动脚本，并通过 setup_once.bat 和 start_site.bat 将初始化过程与日常启动过程分开。

第一次启动时，用户只需执行 setup_once.bat，脚本会自动完成虚拟环境创建、依赖安装、本地环境配置和 ODBC 驱动检查。图 5.4 展示了 Windows 首次配置过程，其中包括 Python 依赖安装、本地配置修正和数据库驱动检查等步骤。

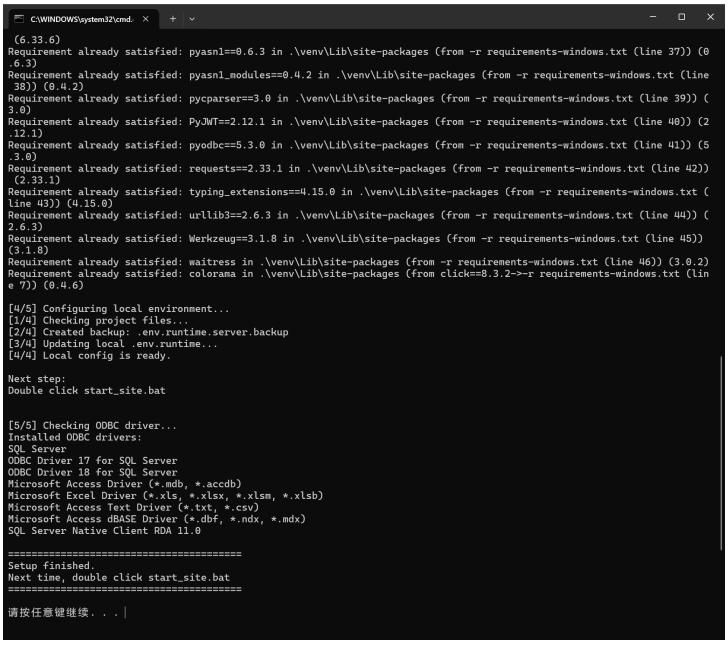

图 5.4: Windows 本地首次配置过程

首次配置完成后，用户日常启动时只需执行 `start_site.bat`。该脚本会先建立数据库 SSH 隧道，再启动本地 Web 服务，并提示浏览器访问本地地址 http://127.0.0.1:5000。图 5.5 展示了本地启动脚本运行结果，图 5.6 展示了本地浏览器访问结果，说明本地部署包已经能够独立完成网站启动与页面访问。

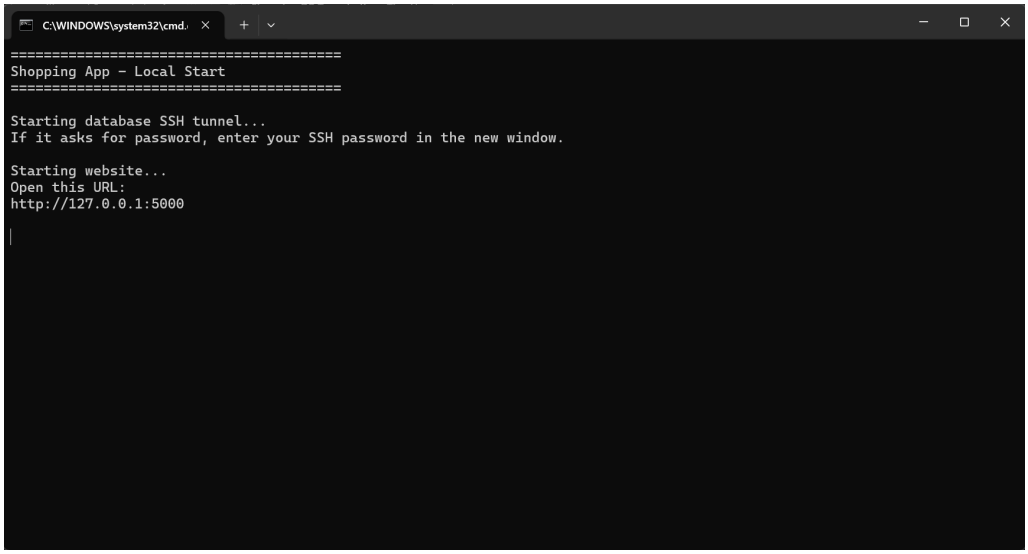

图 5.5: Windows 本地启动脚本运行结果

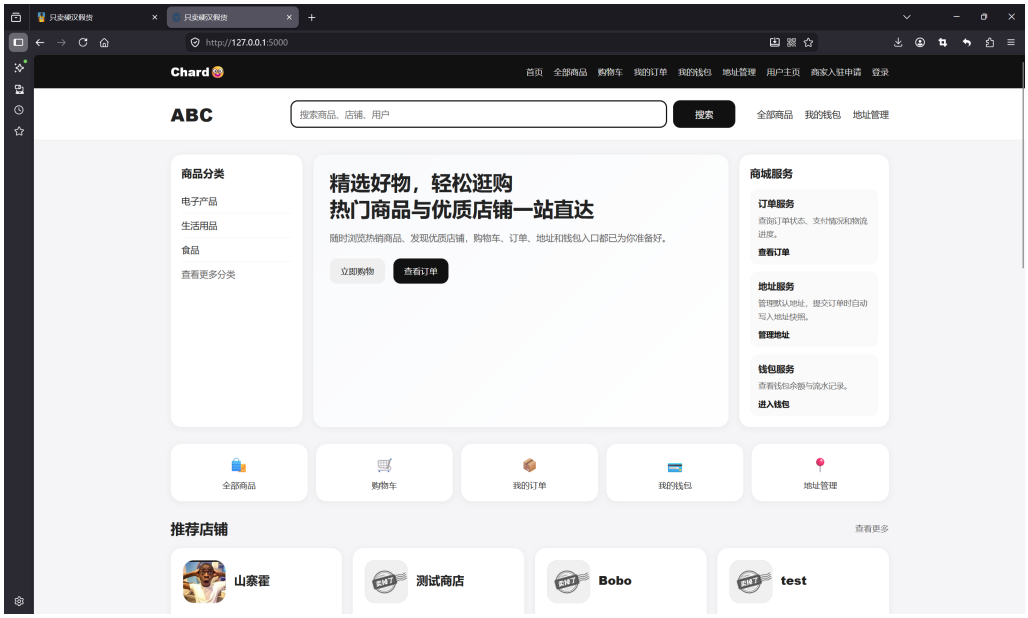

图 5.6: Windows 本地启动后页面访问结果

3 前台公共页面运行测试

前台公共页面运行测试主要面向未登录访客可直接访问的页面，重点验证首页、全部店铺页、搜索页、全部商品页、商品详情页、店铺主页以及认证相关页面是否能够正常加载，并检查公共访问场景下的基础页面状态是否符合预期。当前前台公共入口主要包括首页 shopping.vertexf.top、全部店铺页 shopping.vertexf.top/shops、搜索页 shopping.vertexf.top/search、全部商品页 shopping.vertexf.top/products、商品详情页 shopping.vertexf.top/products/8、店铺主页 shopping.vertexf.top/shop/shop-6、登录页 shopping.vertexf.top/firebase-login、注册页 shopping.vertexf.top/firebase-register 和重置密码页 shopping.vertexf.top/reset-password。

首先，对全部店铺页进行了访问测试。页面能够正常展示当前站内可公开访问的店铺列表，并显示店铺名称、简介、销量、成交额和成交订单数等信息，说明店铺列表查询与页面渲染过程正常，如图 5.7 所示。

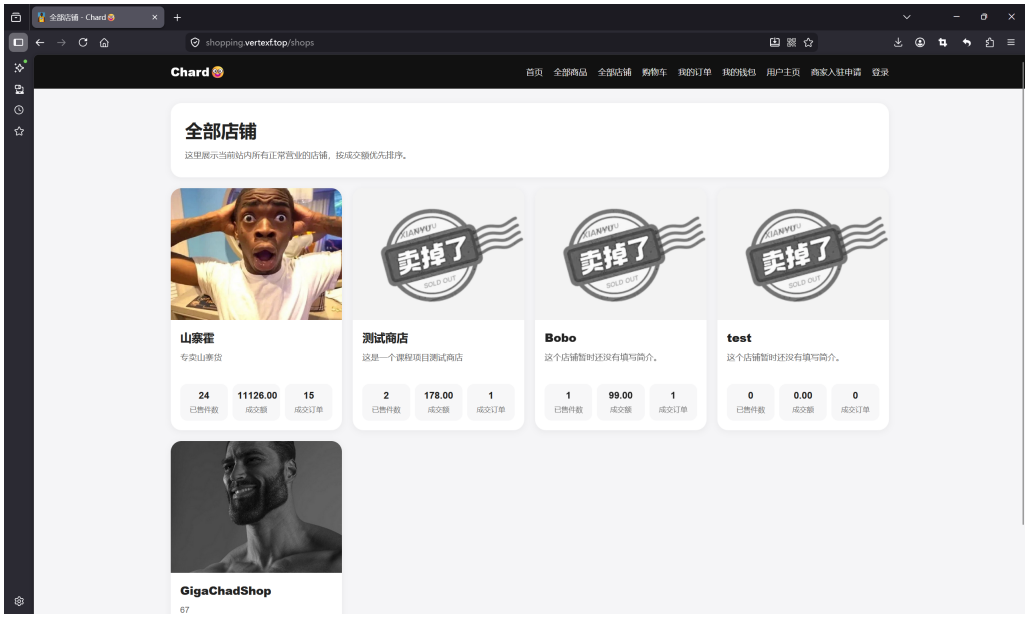

图 5.7: 全部店铺页面运行结果

搜索页在无关键词输入时能够正常打开，并显示商品、店铺和用户三类结果入口，说明统一搜索页本身能够独立访问，如图 5.8 所示。当输入关键词后，页面能够返回对应商品结果，并在页面中显示各类型结果数量与商品卡片内容，说明搜索条件提交、结果分类统计和结果展示功能均可正常运行，如图 5.9 所示。

全部商品页能够正常列出当前上架商品，并提供搜索框、分类筛选条和商品卡片展示区。商品卡片中可显示商品图片、分类、所属店铺、库存、价格以及查看详情、加入购物车等操作入口，说明商品列表模块能够正常工作，如图 5.10 所示。

在商品详情测试中，系统能够正常展示单个商品的主图、商品名称、价格、库存、所属店铺、所属分类和商品说明内容，并提供加入购物车入口，说明商品

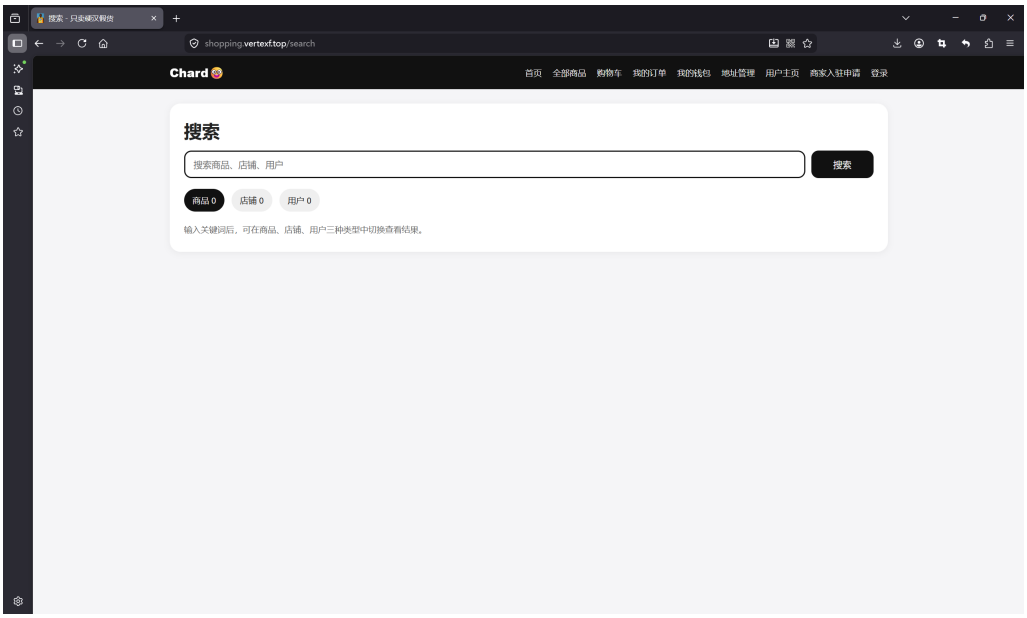

图 5.8: 搜索页面初始状态

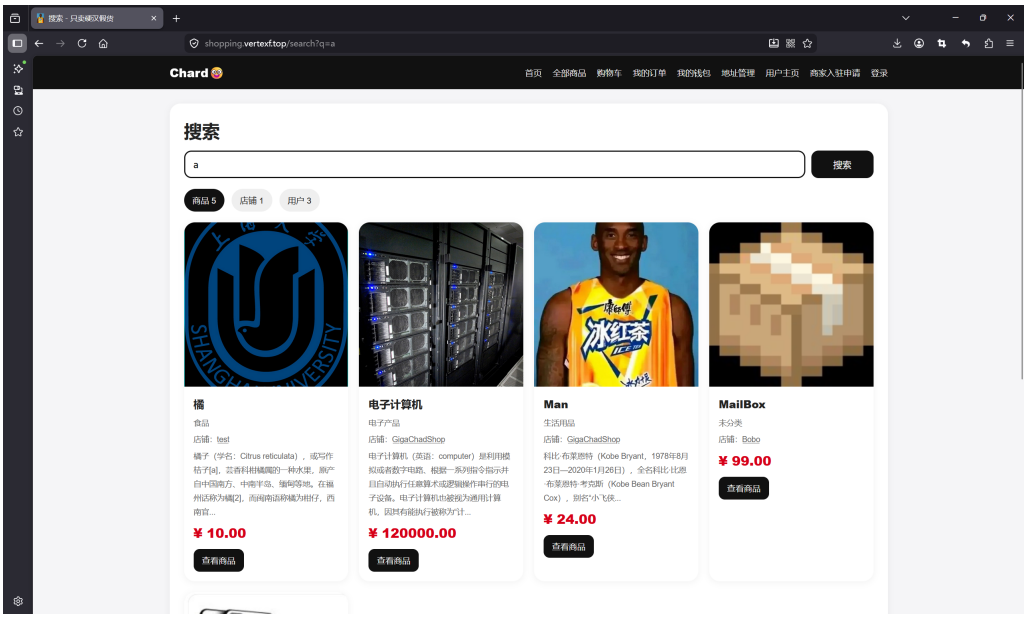

图 5.9: 搜索页面关键词查询结果

详情查询与富文本说明展示均已生效，如图 5.11 所示。

在店铺主页测试中，系统能够正常展示店铺名称、Logo、联系方式、店铺统计信息、主页描述和店铺商品数量等内容，说明店铺主页公开展示功能可正常运行，如图 5.12 所示。

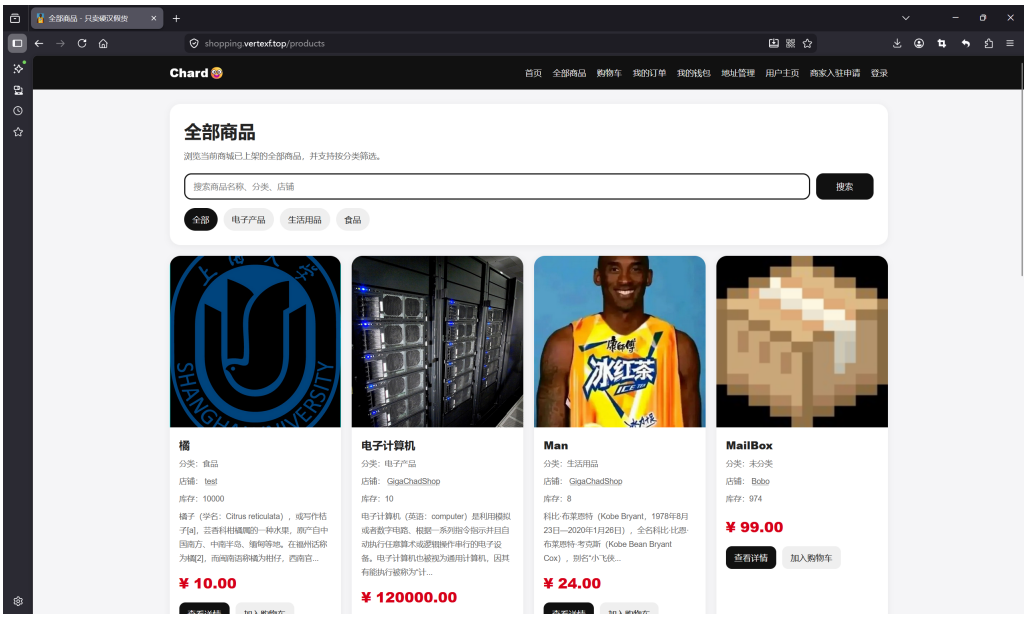

图 5.10: 全部商品页面运行结果

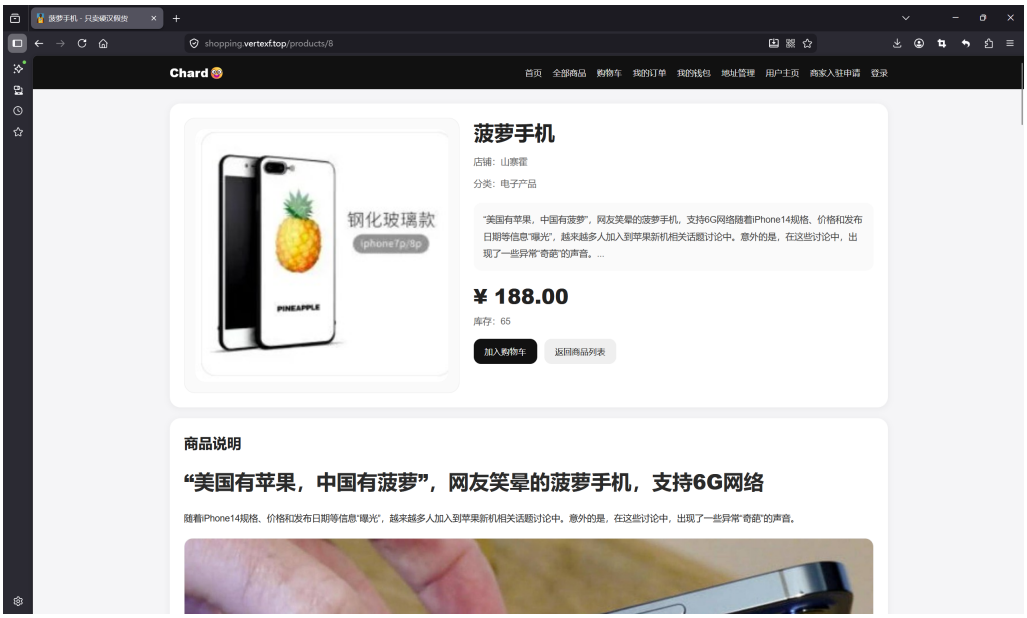

图 5.11: 商品详情页面运行结果

认证相关公共页面同样能够正常访问。登录页可正常显示用户名或邮箱输入框、密码输入框以及登录按钮；注册页可正常显示邮箱注册表单；重置密码页可正常显示邮箱输入与邮件发送入口。这说明认证前端页面已经完成基本渲染与访问组织，如图 5.13、图 5.14 和图 5.15 所示。

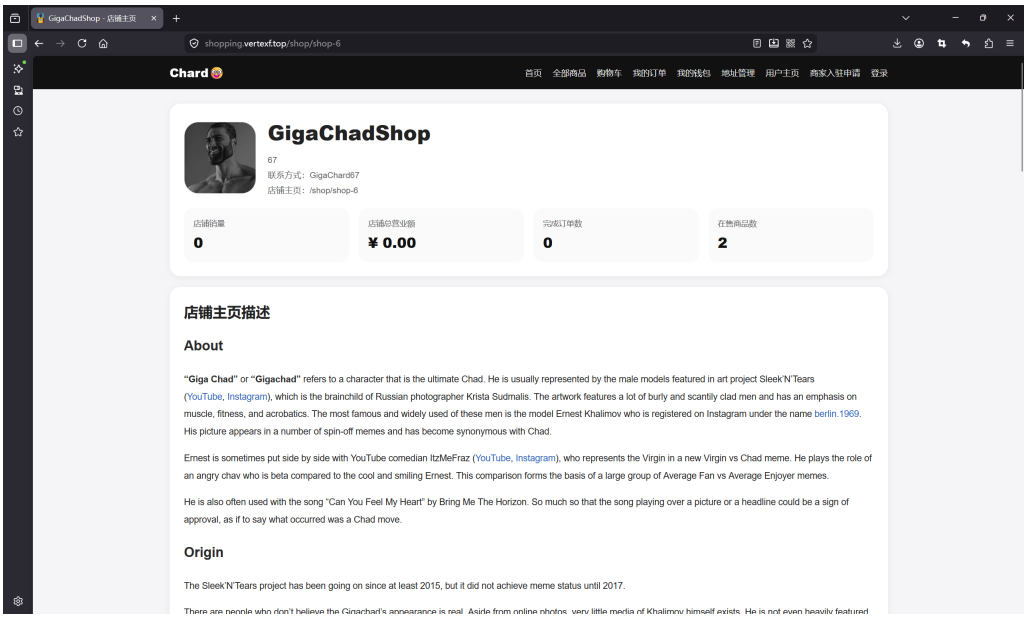

图 5.12: 店铺主页运行结果

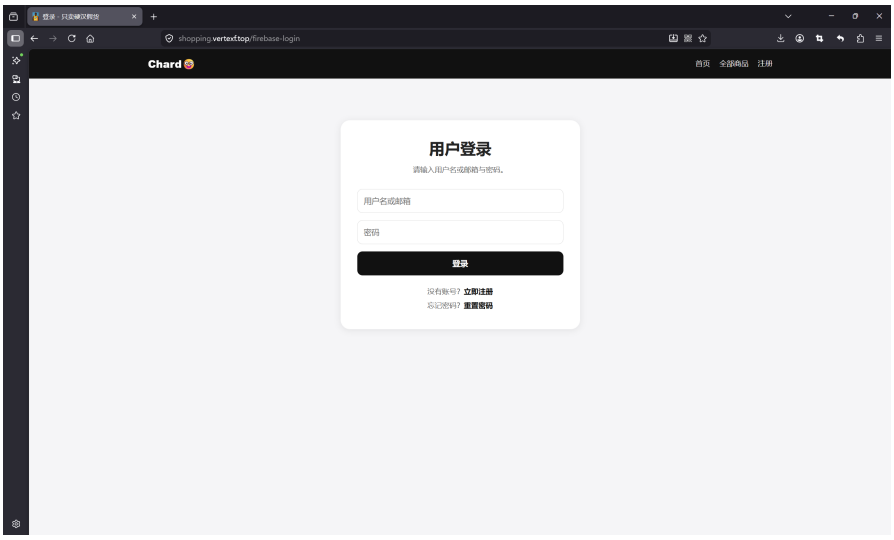

图 5.13: 登录页面运行结果

在页面状态测试中，还对错误地址访问和特殊字符搜索进行了验证。访问不存在的路径 shopping.vertexf.top/not-found-test 时，系统返回统一的 404 页面，而不是直接暴露默认服务器报错页面，如图 5.16 所示。以 `<script>` 作为搜索关键词访问搜索页时，页面能够正常返回空结果页，不会出现异常脚本执行或页面错误，说明搜索输入在公共页面场景下能够被正常处理，如图 5.17 所示。

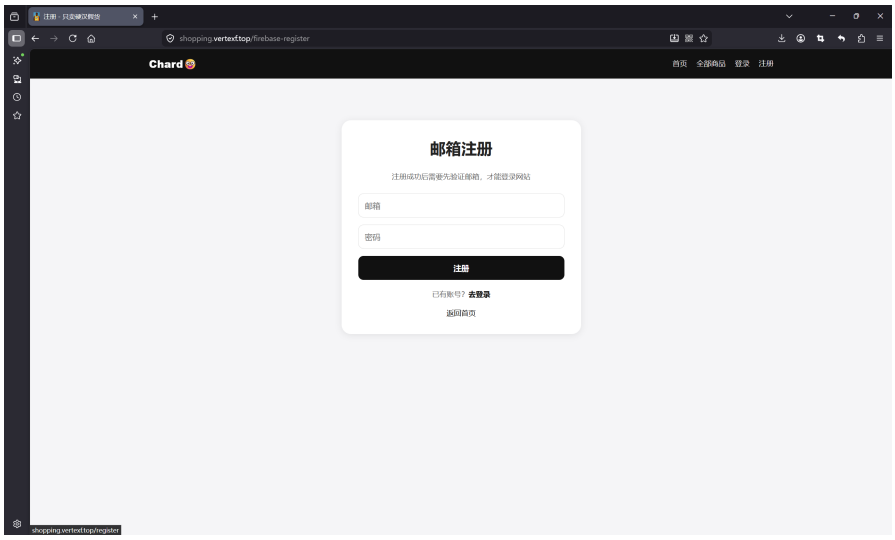

图 5.14: 注册页面运行结果

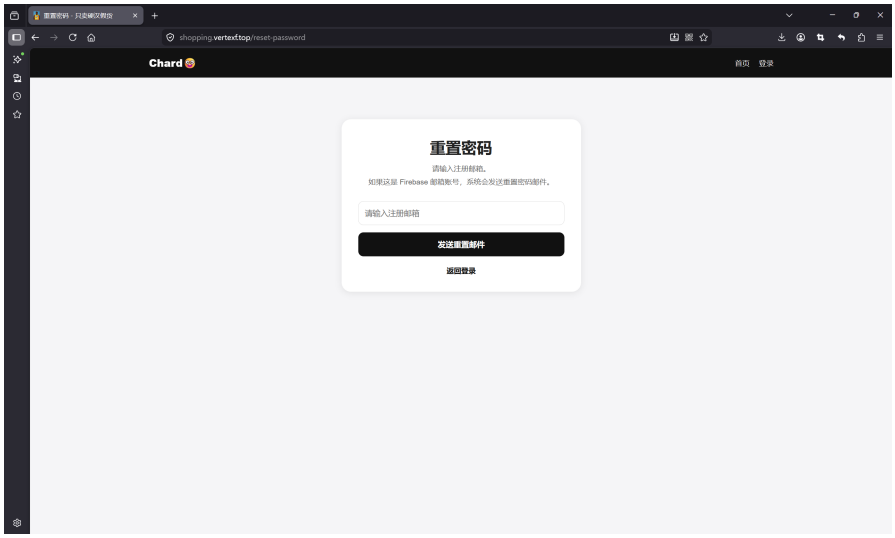

图 5.15: 重置密码页面运行结果

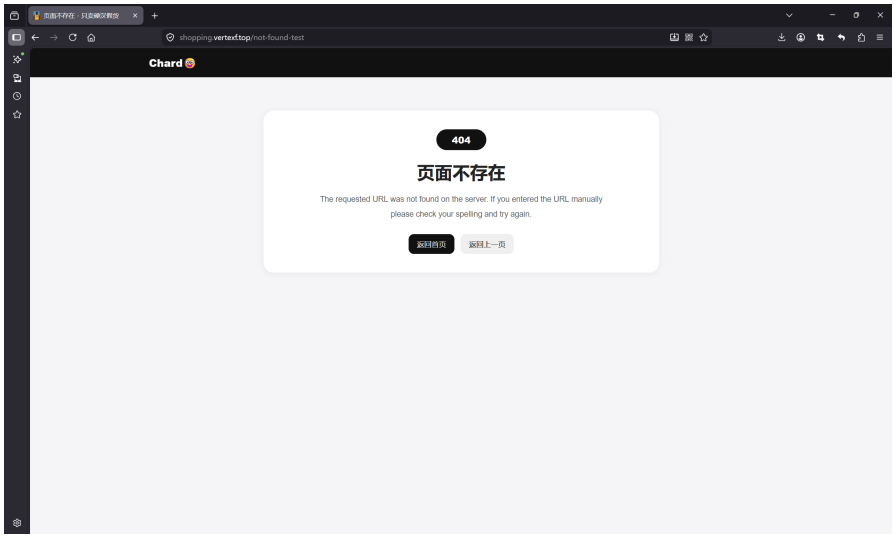

图 5.16: 统一 404 页面运行结果

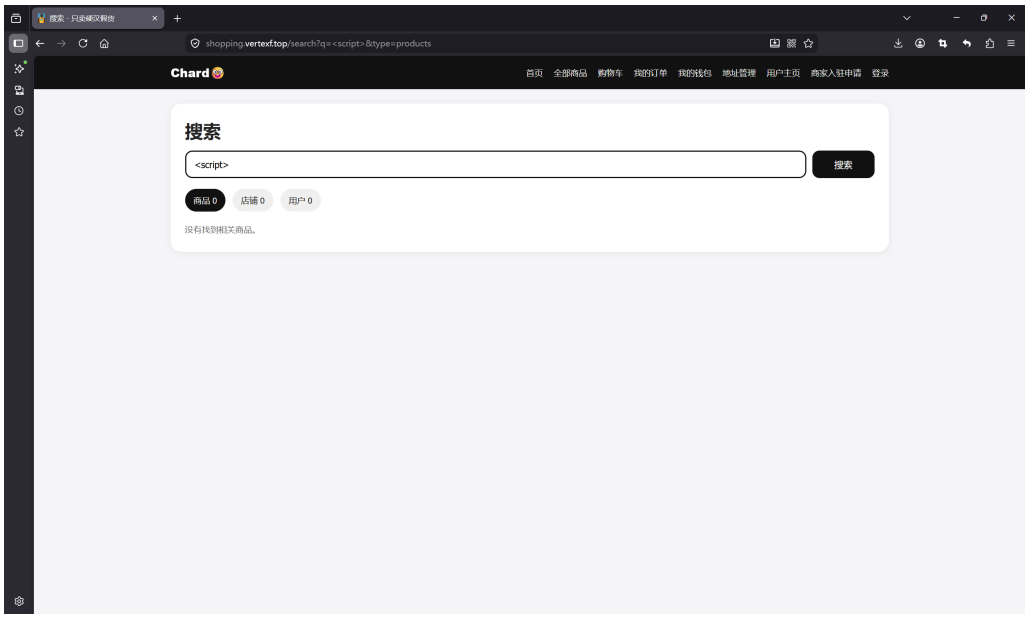

图 5.17: 特殊字符搜索输入测试结果

此外，对购物车、订单、钱包、地址管理等依赖用户身份的页面进行了未登录访问测试，访问这些路由时系统会直接跳转到登录页，而不会向未认证用户开放对应页面内容。由此可见，前台公共页面与用户私有页面在访问边界上已经被区分开来。

4 普通用户功能流程测试

普通用户功能流程测试主要围绕登录后的核心使用链路展开，重点验证用户主页、地址管理、购物车、结算、批量付款、订单查询与钱包等功能是否能够形成完整闭环。当前普通用户主要访问的功能路由包括用户主页 shopping.vertexf.top/account、地址管理 shopping.vertexf.top/addresses、购物车 shopping.vertexf.top/cart、结算页 shopping.vertexf.top/checkout、批量付款页 shopping.vertexf.top/orders/pay-batch、订单列表页 shopping.vertexf.top/orders、订单详情页和钱包页 shopping.vertexf.top/wallet。

用户登录成功后，系统首页顶部导航会切换为登录态显示方式，页面中出现购物车、我的订单、我的钱包、地址管理和用户主页等入口，同时右上角展示当前登录用户昵称与头像，说明系统已经能够正确识别并维持用户会话状态，如图 5.18 所示。

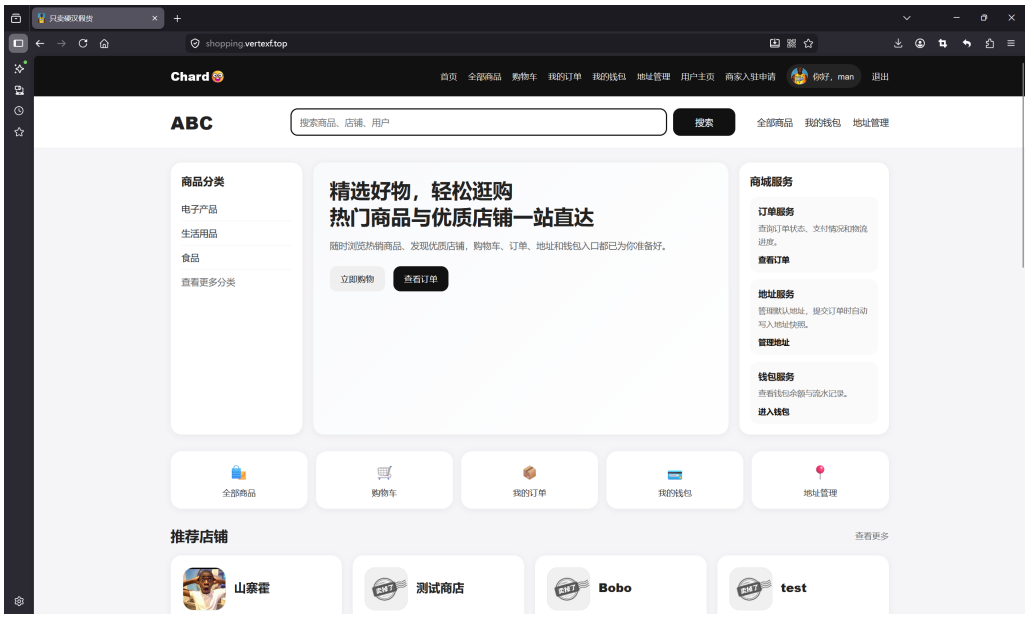

图 5.18: 用户登录后的首页状态

在用户主页测试中，系统能够正常显示当前用户头像、昵称、用户编号、用户名、邮箱和账号状态等信息，同时提供头像上传和昵称修改入口。图 5.19 展示了用户主页初始状态，图 5.20 展示了昵称与头像修改后的结果。可以看到，修改完成后页面提示操作成功，顶部导航中的显示名称与头像也同步更新，说明用户资料维护功能已经生效。

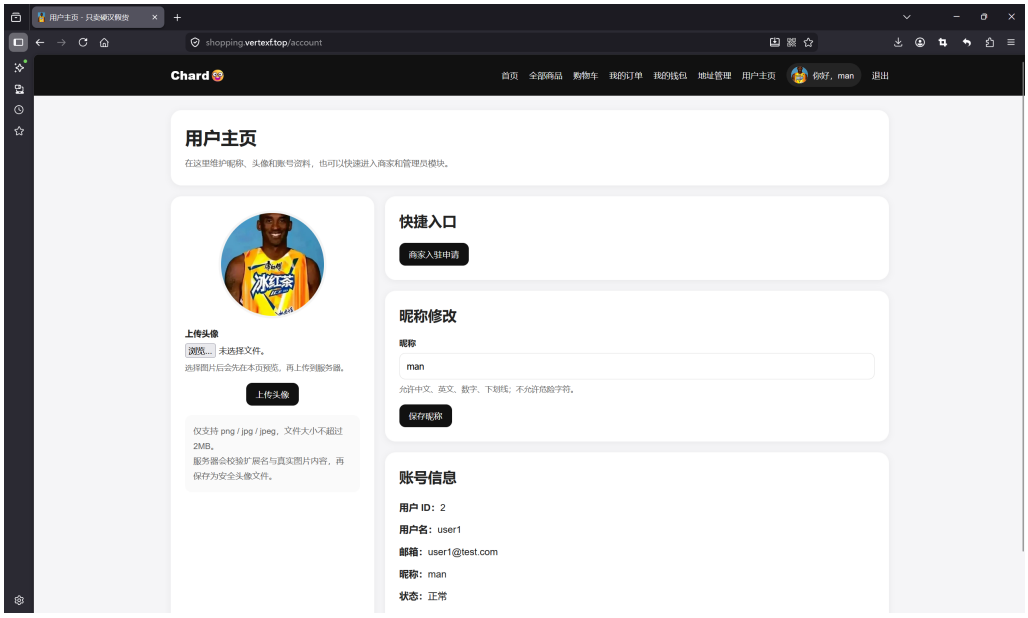

图 5.19: 用户主页初始状态

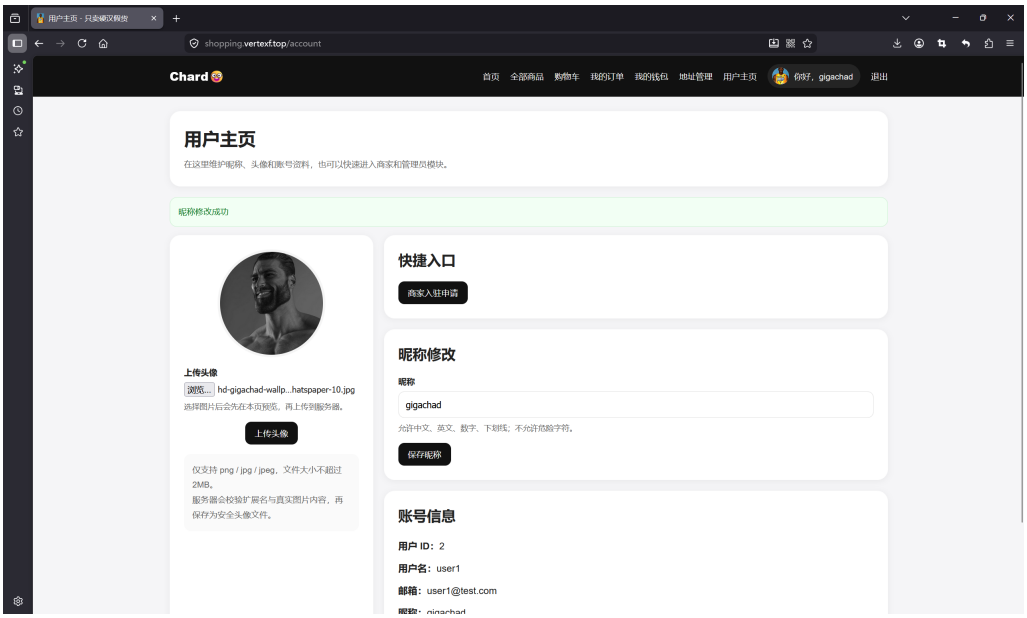

图 5.20: 用户主页修改资料后的结果

地址管理页面能够同时完成地址录入、默认地址识别和地址列表展示。图 5.21 展示了系统已有一条默认地址时的页面状态，图 5.22 展示了新增地址后的结果。页面中能够区分默认地址与普通地址，并继续提供“设为默认”和“删除”等操作入口，说明地址维护逻辑已经能够支持结算前的地址准备过程。

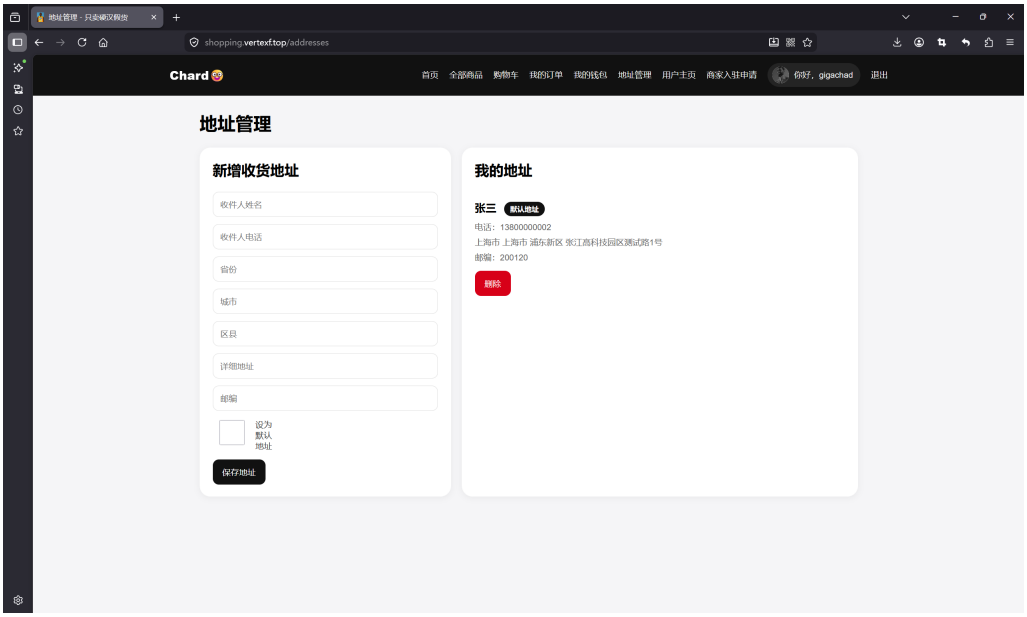

图 5.21: 地址管理页面初始状态

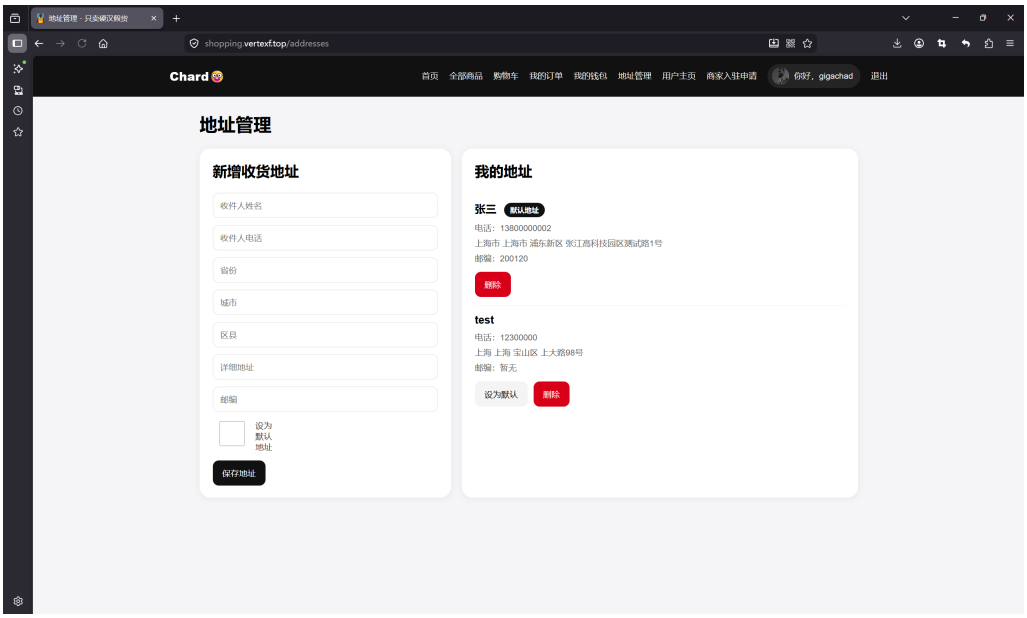

图 5.22: 新增地址后的地址管理结果

购物车模块测试中，系统能够区分空购物车状态与已有商品状态。图 5.23 展示了购物车为空时的页面提示，页面会引导用户返回商品中心继续浏览。图 5.24 展示了用户在商品列表页执行“加入购物车”后的提示结果，页面能够即时反馈加入成功。图 5.25 则展示了购物车中已有多件商品时的页面状态，系统能够正确显示商品信息、库存、数量、单价、小计和总金额，并提供更新数量与删除商品入口。

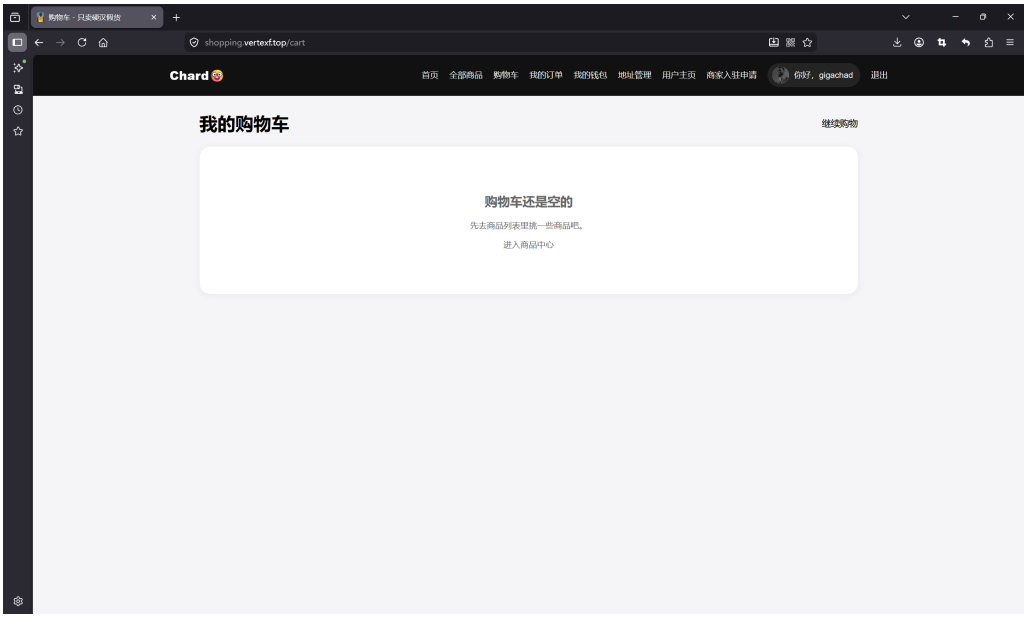

图 5.23: 空购物车页面状态

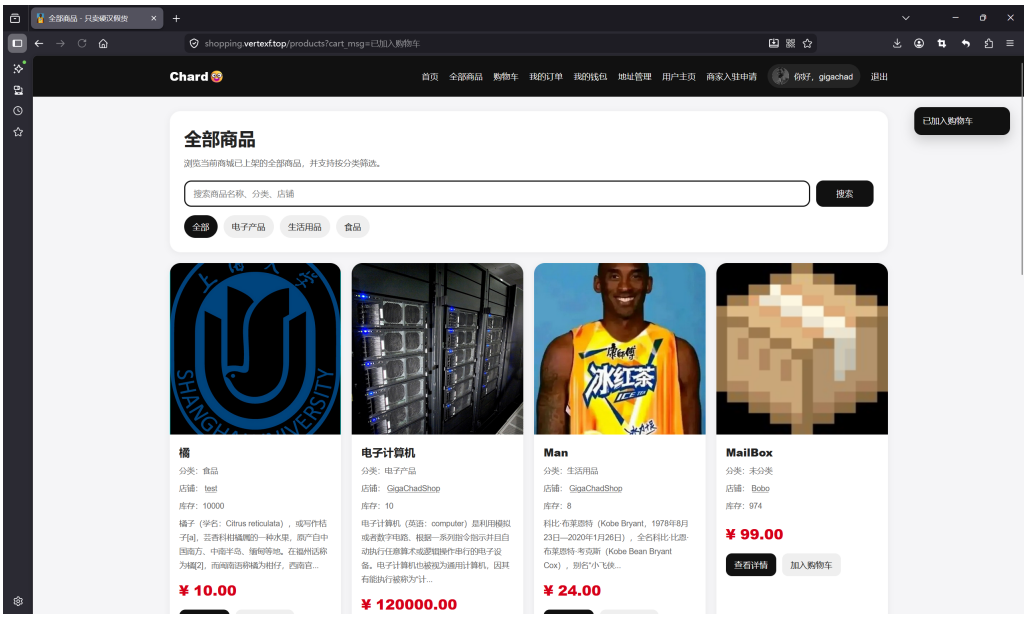

图 5.24: 商品加入购物车后的页面提示

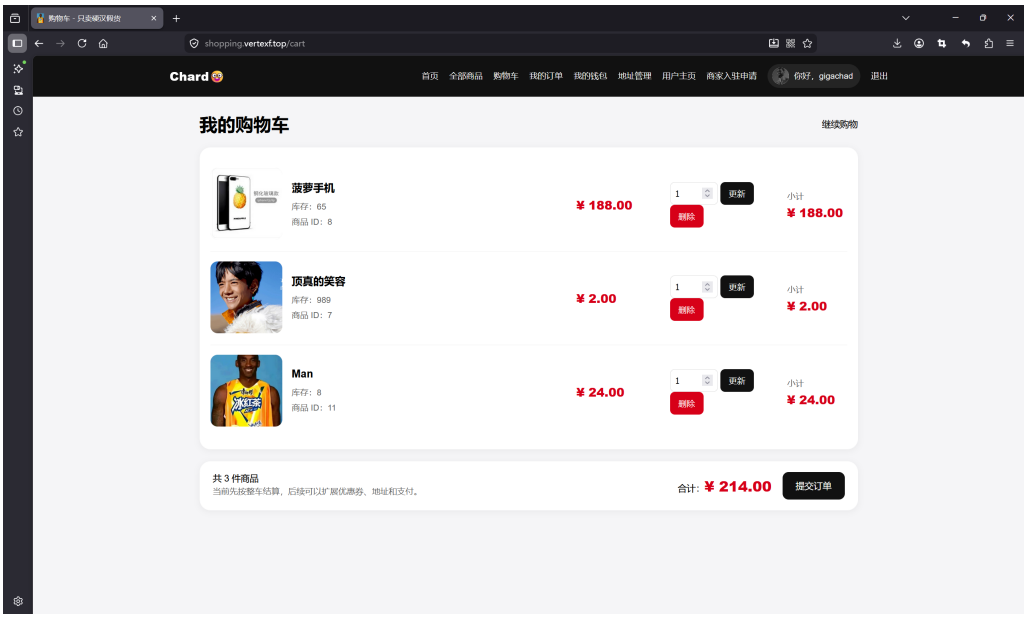

图 5.25: 购物车中已有商品时的页面状态

结算页测试表明，系统能够从购物车中读取待结算商品，并同步读取当前用户地址信息与金额汇总结果。图 5.26 中可以看到，页面已显示商品明细、地址选择区、商品数量、商品金额、运费和应付总额，说明结算前的商品校验、地址读取和金额汇总过程均已正常完成。

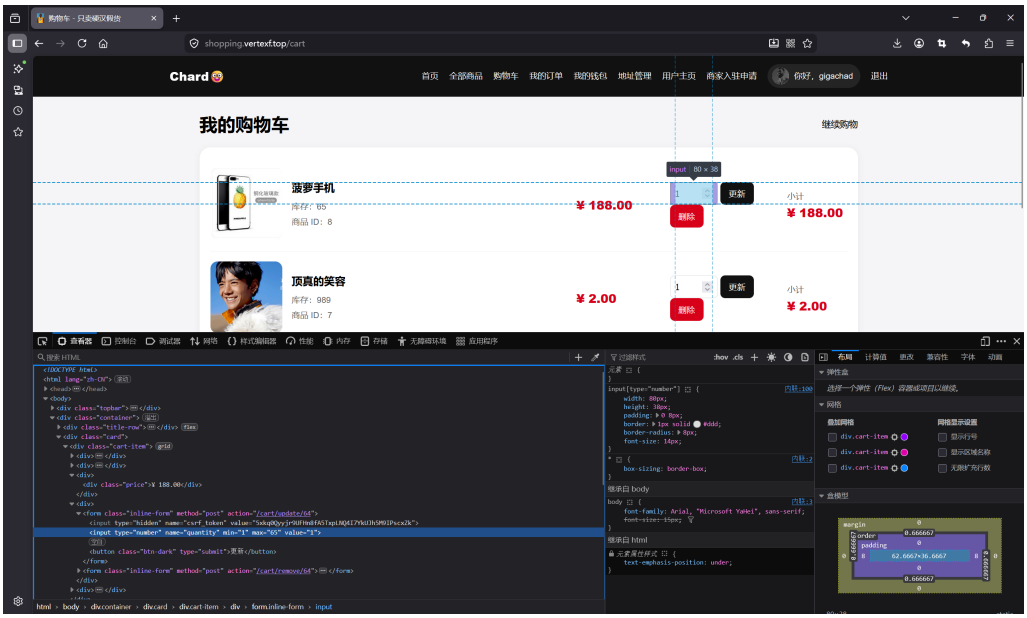

图 5.26: 结算页面运行结果

当购物车中商品来自不同店铺时，系统会自动拆分为多笔订单，并进入批量付款页面。图 5.27 展示了批量付款页的实际结果，页面中分别列出了两笔已拆分订单，并汇总总应付金额。该结果说明系统能够在结算完成后根据店铺维度自动拆单，并为后续统一付款提供页面入口。

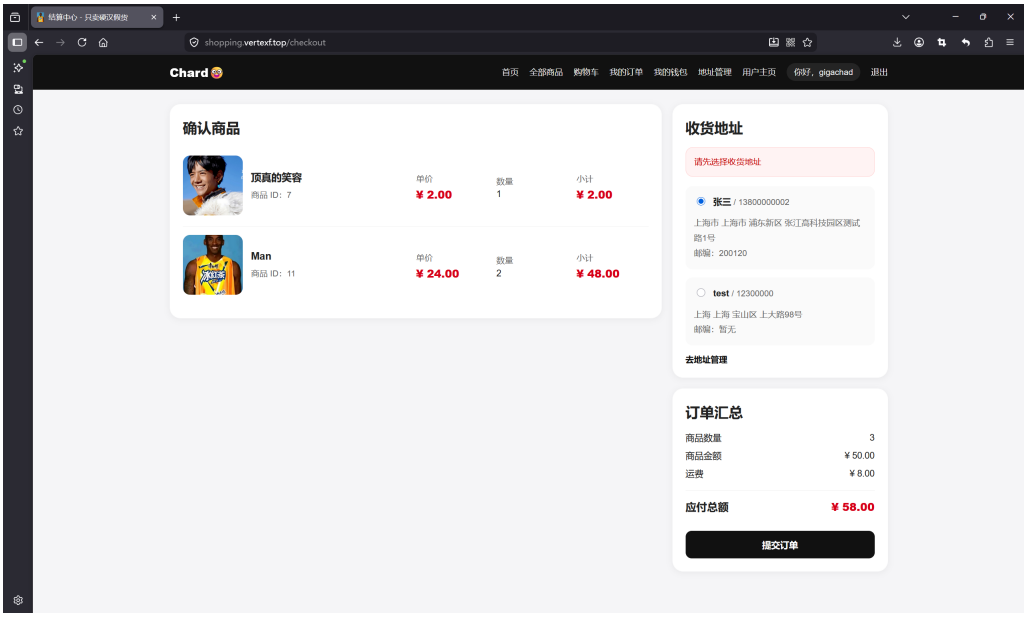

图 5.27: 多店铺订单批量付款页面

订单列表页能够集中展示当前用户全部订单，并以状态文本和倒计时方式区分不同订单的处理阶段。图 5.28 展示了订单列表中的“已支付，待发货”和“已发货”等不同状态，页面同时提供订单详情入口。进一步进入订单详情页后，系统能够显示收货地址、物流信息、商品明细和状态日志，如图 5.29 所示，说明订单查询链路已经能够完整反映订单从提交到支付后的状态变化过程。

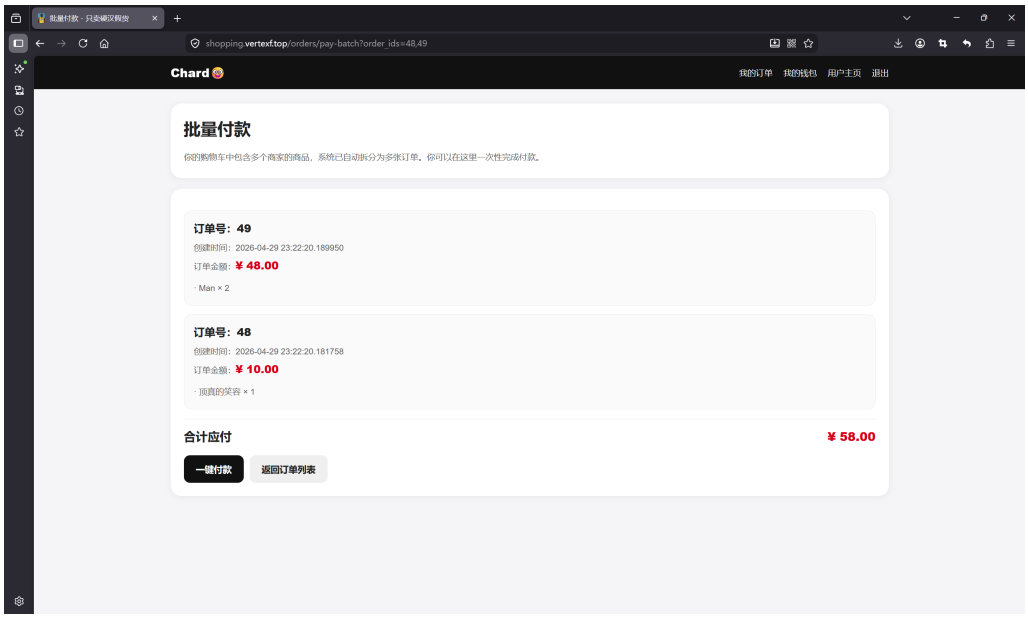

图 5.28: 订单列表页面运行结果

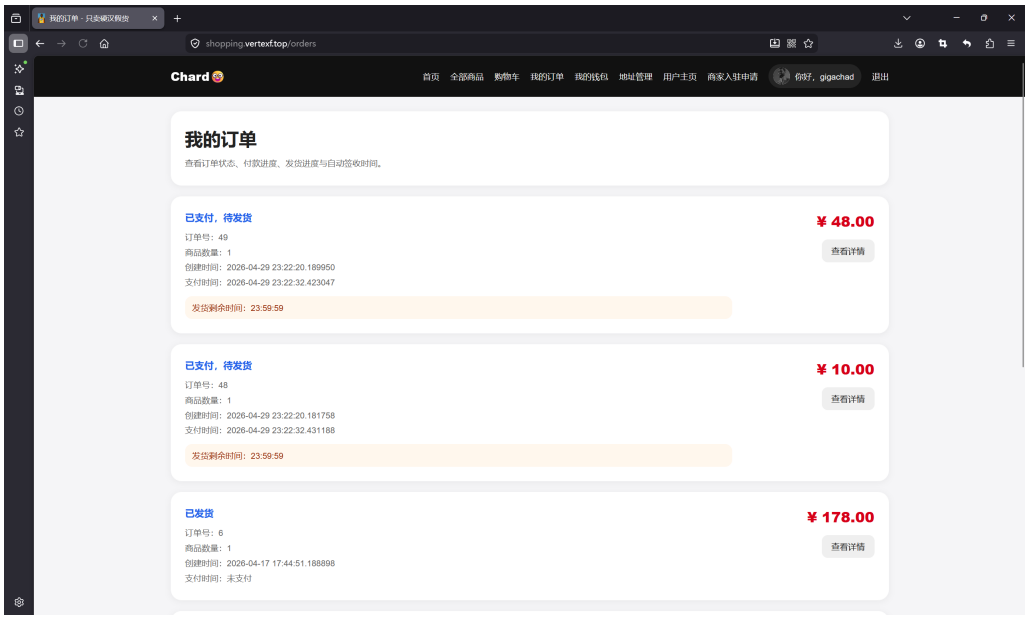

图 5.29: 订单详情页面运行结果

钱包页面能够正确显示当前余额以及历史资金流水。图 5.30 展示了钱包页面的实际结果，页面中既显示了当前余额，也展示了多笔支付相关流水及变动后余额，说明钱包余额与交易流水能够随着订单支付过程同步更新。

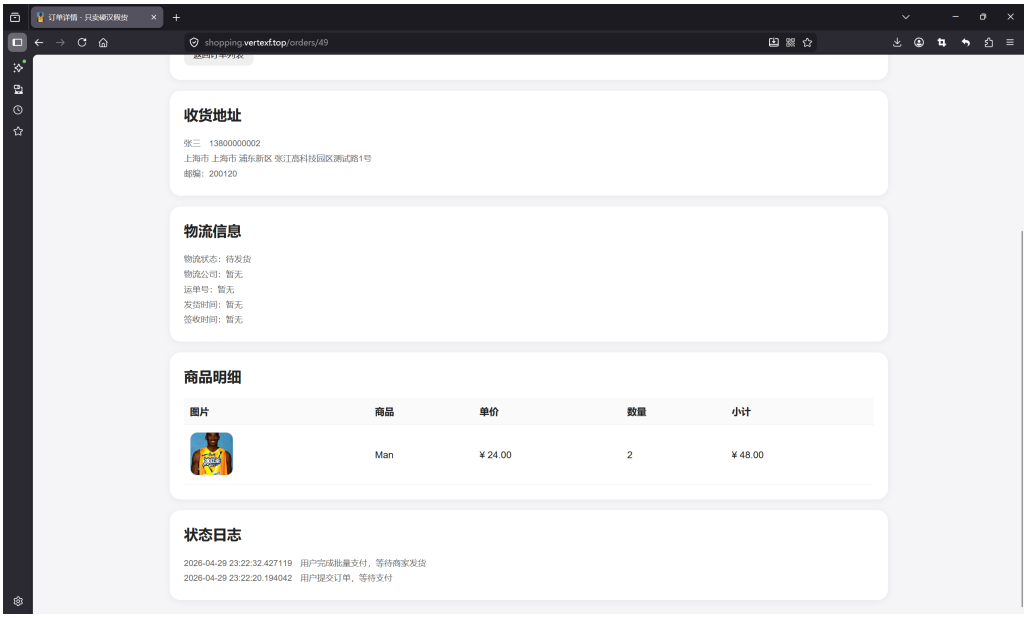

图 5.30: 钱包页面运行结果

5 商家后台功能测试

商家后台功能测试主要验证商家身份获得后，是否能够完成入驻申请、进入商家后台、管理商品、维护店铺主页以及处理订单等核心操作，并检查商家身份与管理员权限之间的访问边界是否正确区分。当前商家侧主要功能路由包括商家入驻申请页 shopping.vertexf.top/merchant/apply、商家后台首页 shopping.vertexf.top/merchant/dashboard、商品管理页 shopping.vertexf.top/merchant/products、新增商品页 shopping.vertexf.top/merchant/products/new、商品编辑页、店铺主页编辑页 shopping.vertexf.top/merchant/shop/edit 以及商家订单管理页 shopping.vertexf.top/merchant/orders。

首先，对商家入驻申请流程进行了测试。未具备商家身份的用户可以访问入驻申请页，填写店铺名称、联系人、联系电话、经营范围和申请理由后提交申请。提交后，页面会显示“商家入驻申请已提交，请等待管理员审核”的提示，并展示当前申请状态，说明商家申请链路能够正常运行，如图 5.31 和图 5.32 所示。

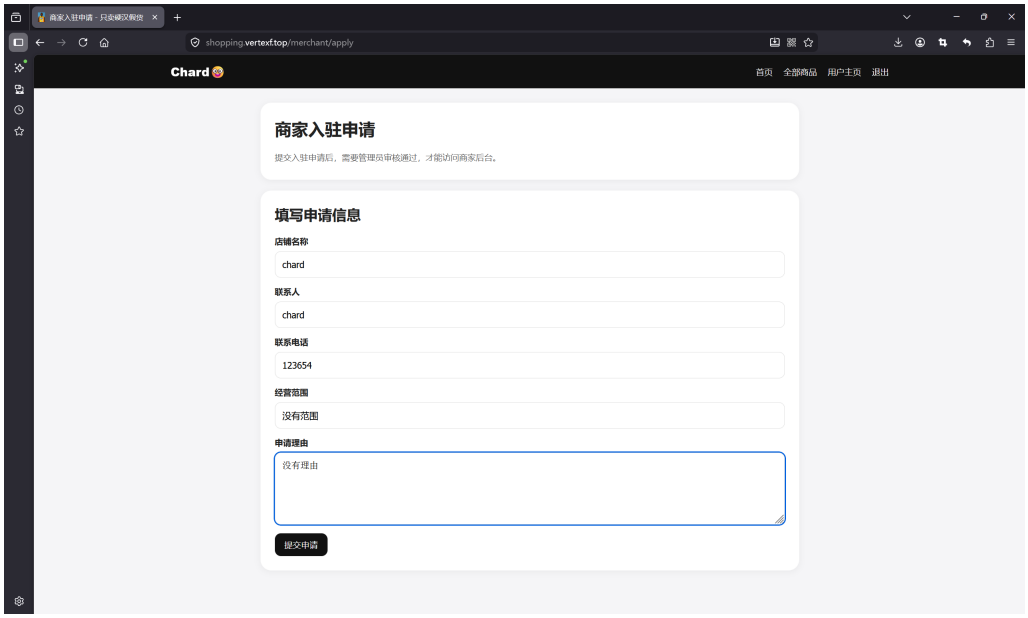

图 5.31: 商家入驻申请页面

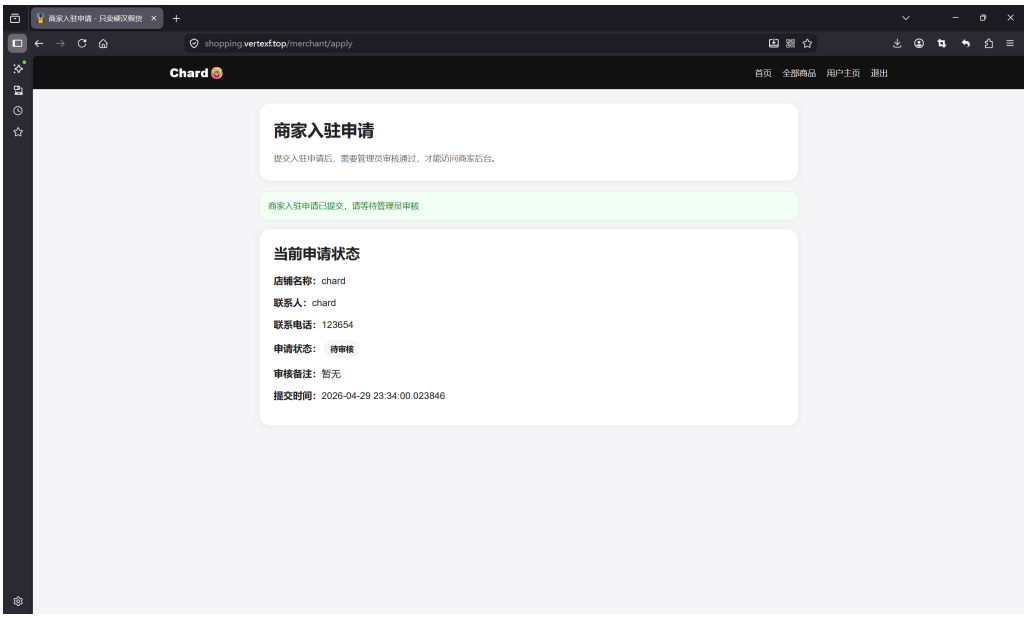

图 5.32: 商家入驻申请提交后的结果

当用户具备商家身份后，系统可正常进入商家后台首页。页面中能够显示当前店铺名称、商家编号、店铺编号、店铺标识以及商品数量、关联订单数、已支付订单数等统计信息，并提供商品管理、订单管理、新增商品、店铺主页和编辑主页等快捷入口，说明商家后台首页能够完成基础经营信息汇总，如图 5.33 所示。

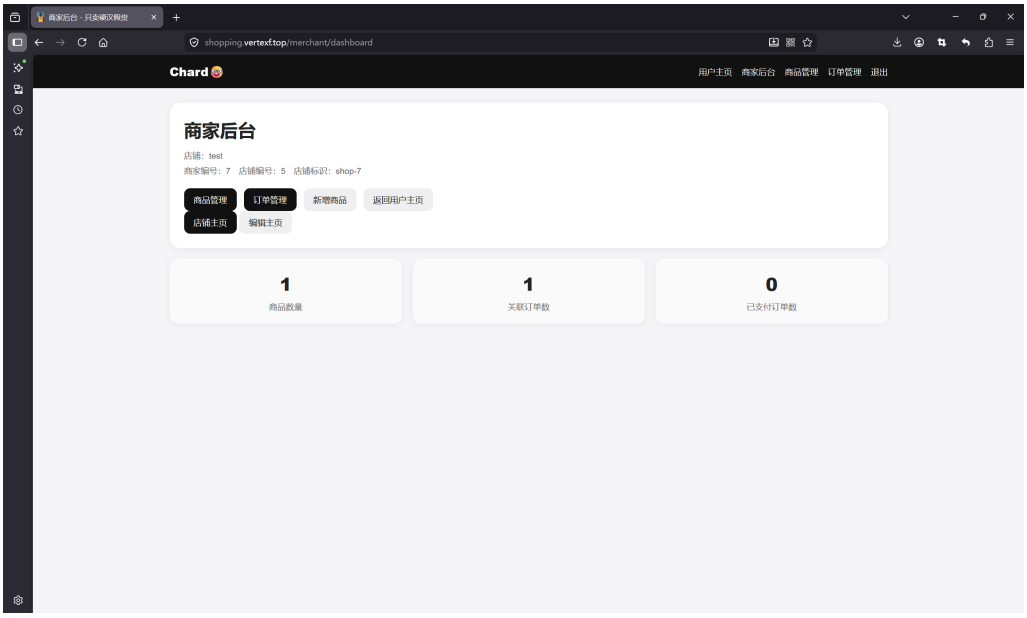

图 5.33: 商家后台首页运行结果

商品管理功能测试中，系统能够正确列出当前店铺下的商品信息，包括商品图片、商品名、分类、价格、库存、状态和更新时间，并提供编辑、查看详情以及上下架操作入口。图 5.34 展示了当前测试店铺的商品管理结果，图 5.35 展示了另一已运行商铺的商品管理结果。可以看到，页面能够区分“上架中”和“已下架”等状态，说明商品列表读取与状态展示功能正常。

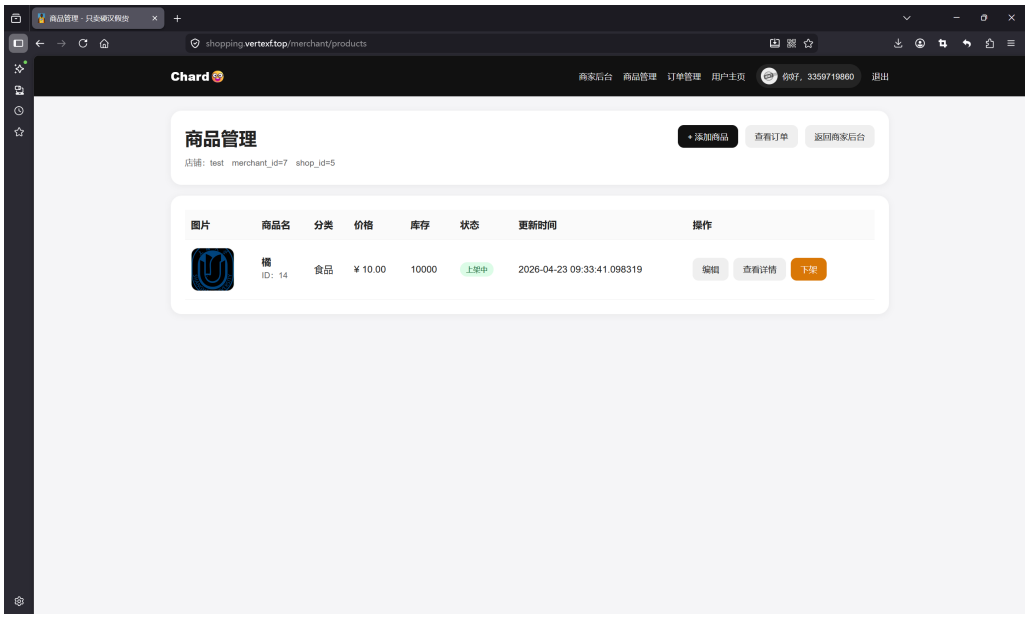

图 5.34: 当前测试店铺的商品管理页面

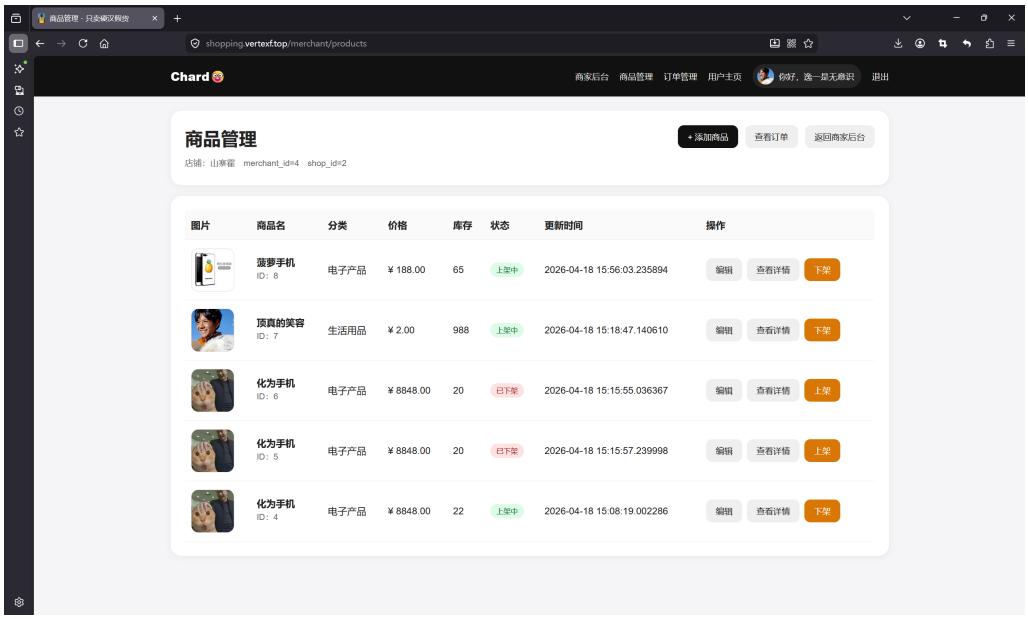

图 5.35: 已有商铺的商品管理页面状态

在商品维护测试中，新增商品页能够正常打开，并提供商品名称、分类、价格、库存、商品说明和主图上传区域，如图 5.36 所示。商品编辑页能够读取已有商品信息，并支持在富文本编辑器中继续维护商品说明内容，如图 5.37 所示。编辑保存后，前台商品详情页中的商品名称、价格、库存和说明内容能够同步显示，说明商家后台编辑结果已能够传递到前台展示页面，如图 5.38 所示。

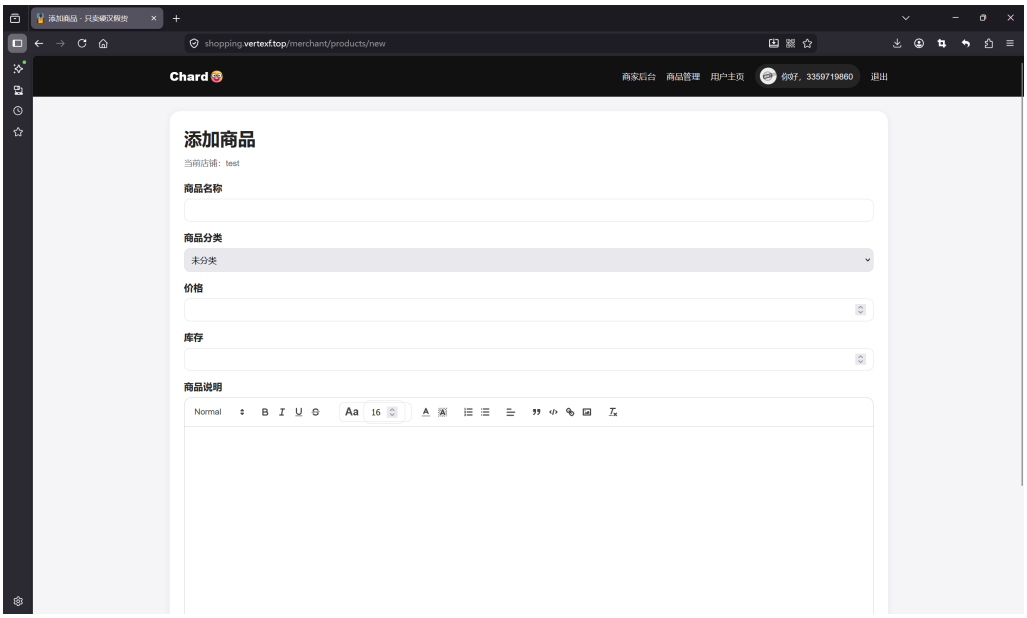

图 5.36: 新增商品页面

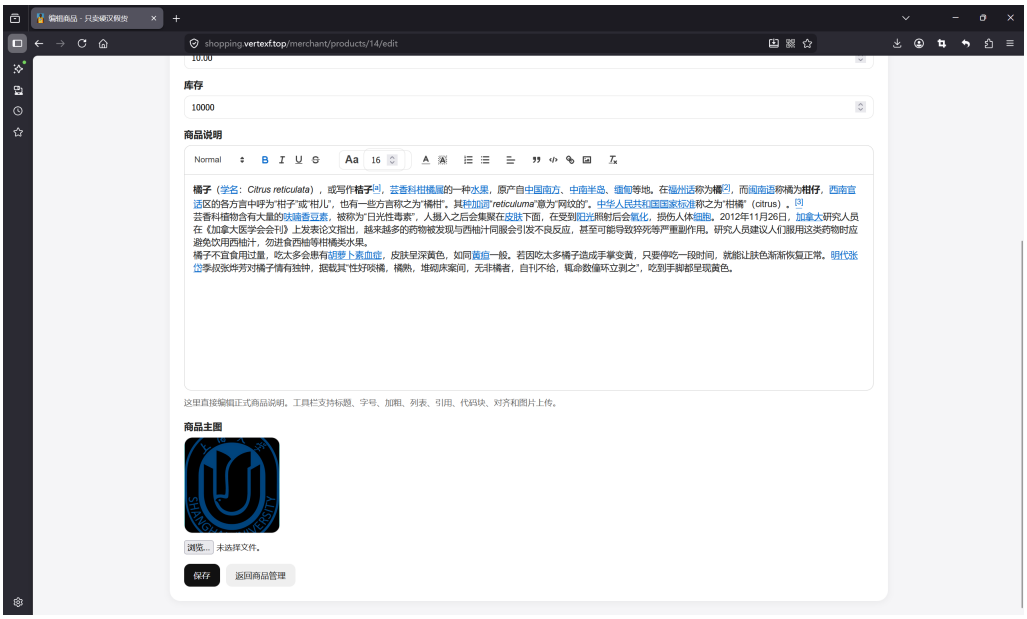

图 5.37: 商品编辑页面

图 5.38: 前台商品详情页中的编辑结果

店铺主页维护功能同样完成了测试。商家可在店铺主页编辑页中修改店铺名称、简略描述、联系方式、店铺 Logo 以及主页描述内容，富文本区域中还支持插入图片，如图 5.39 所示。保存后，公开店铺主页会同步展示更新后的店铺名称、统计信息和主页描述内容，说明商家后台对店铺主页的维护结果已能够在前台店铺页中体现，如图 5.40 所示。为补充不同店铺状态测试，又选取了一家已有运营数据的商铺主页进行展示，页面中能够显示销量、营业额、完成订单数和店铺商品展示区，如图 5.41 所示。

图 5.39: 店铺主页编辑页面

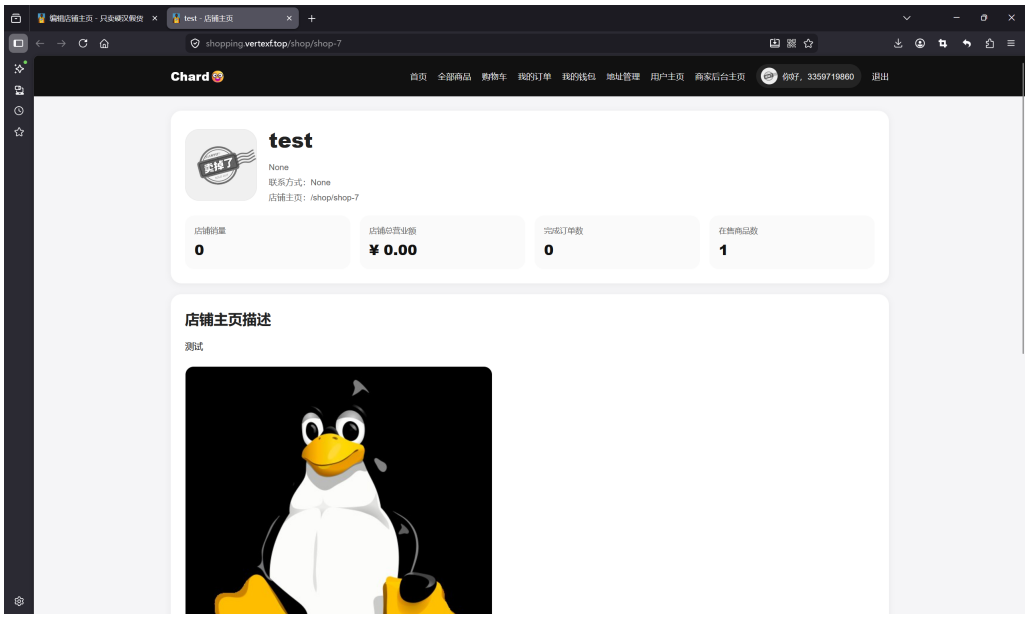

图 5.40: 当前测试店铺的公开主页结果

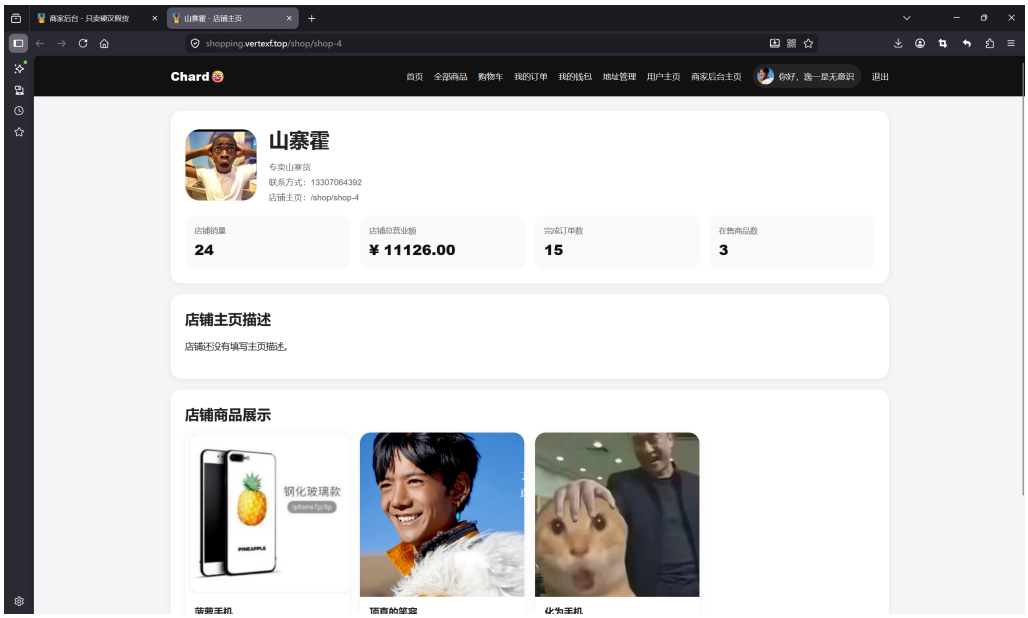

图 5.41: 已有商铺的公开主页状态

商家订单处理是商家后台的关键流程。图 5.42 展示了商家订单管理页中“已支付，待发货”的订单状态，此时页面提供“确认发货”按钮；发货完成后，系统会给出“订单已发货”提示，并将同一订单更新为“已发货，运输中”状态，如图 5.43 所示。说明商家发货操作能够正常推动订单状态变化。

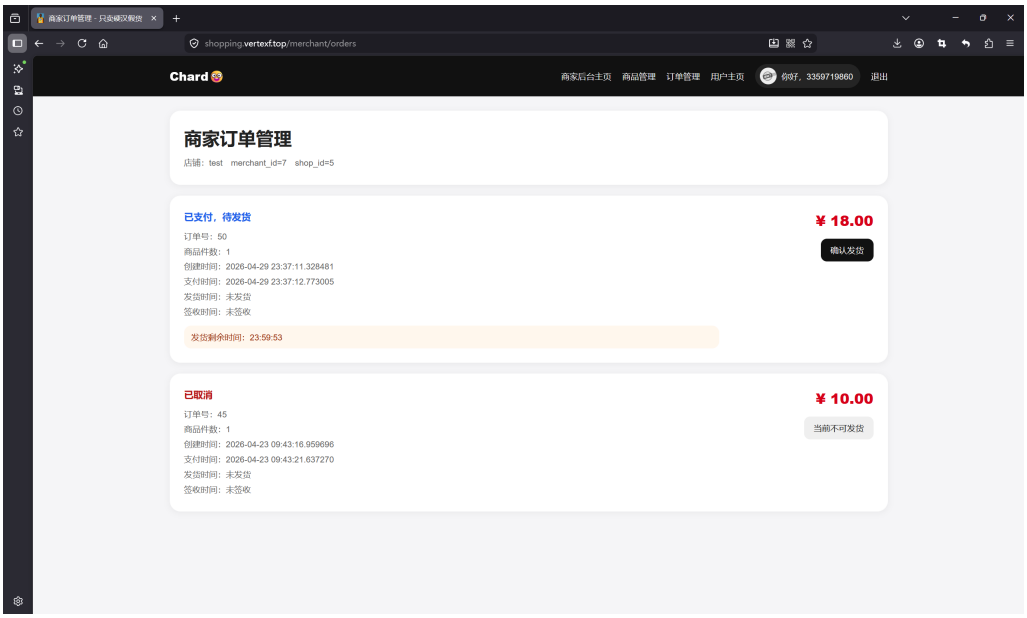

图 5.42: 商家订单管理页面中的待发货状态

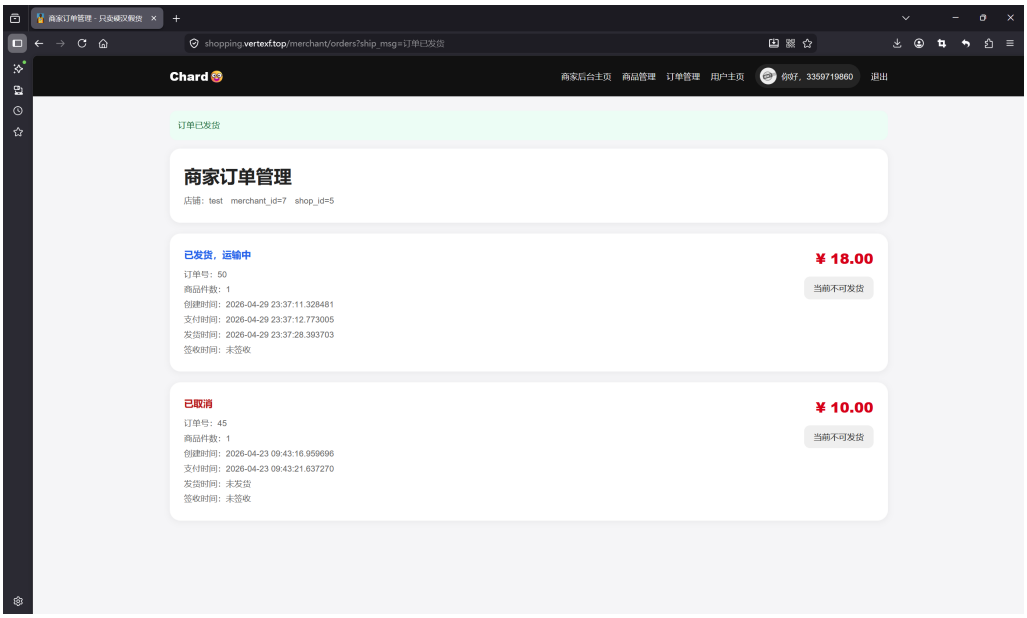

图 5.43: 商家确认发货后的订单状态

为了补充订单状态测试，又选取了已有历史订单进行观察。商家订单管理页中除待发货状态外，还能够显示“已取消”“已送达，待收货”和“已完成”等不同状态，并在不可继续发货时显示禁用提示，如图 5.44 和图 5.45 所示。由此可见，商家订单页已经能够覆盖不同阶段的订单状态展示需求。

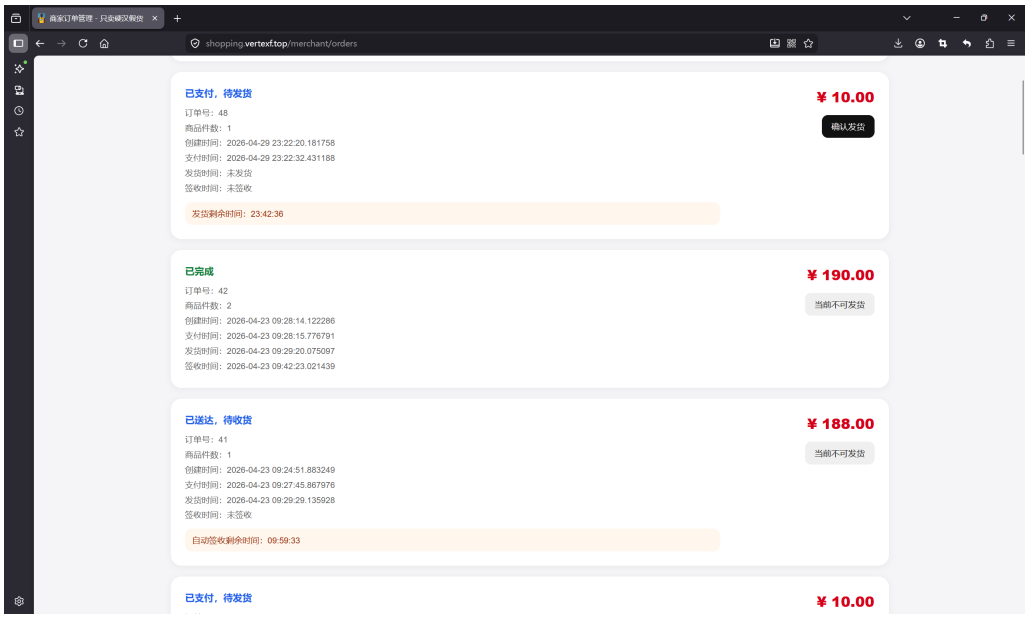

图 5.44: 商家订单管理中的多种订单状态

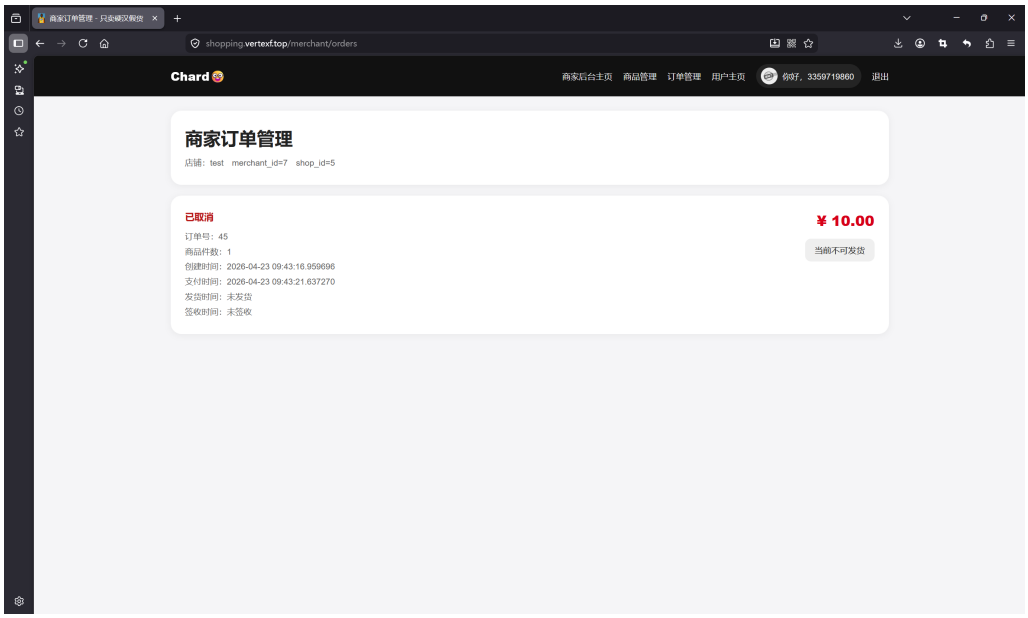

图 5.45: 商家订单管理中的已取消订单状态

在权限边界测试中，商家账号访问管理员后台地址 shopping.vertexf.top/admin/ 时，系统返回统一的 403 无权访问页面，而不会开放管理员功能页面，如图 5.46 所示。这说明商家权限与管理员权限在访问控制上已被区分开来。

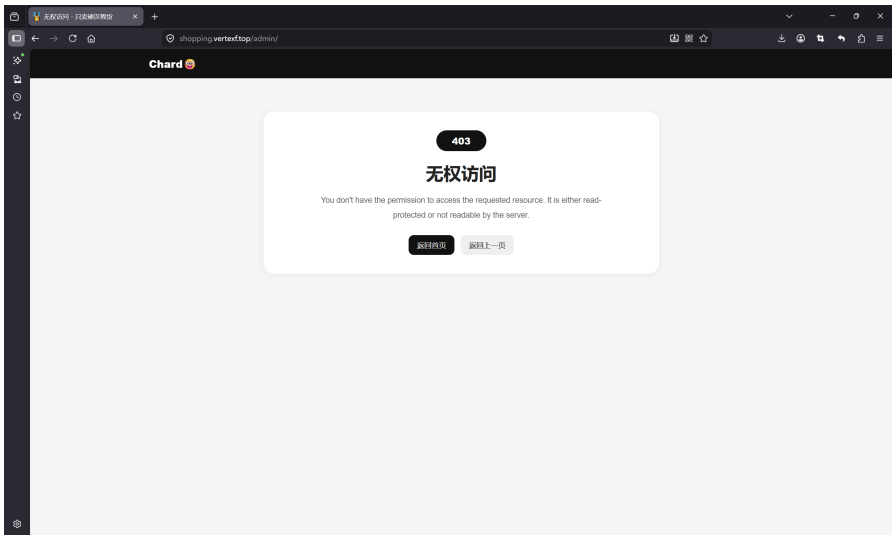

图 5.46: 商家账号访问管理员页面时的 403 结果

此外，商家身份在普通用户页面中的数据结果也能够得到体现。例如已完成订单结算后，商家账号的钱包页面能够显示结算后的余额及对应流水记录，如图 5.47 所示。说明订单结算结果已经能够进一步传递到商家钱包数据中。

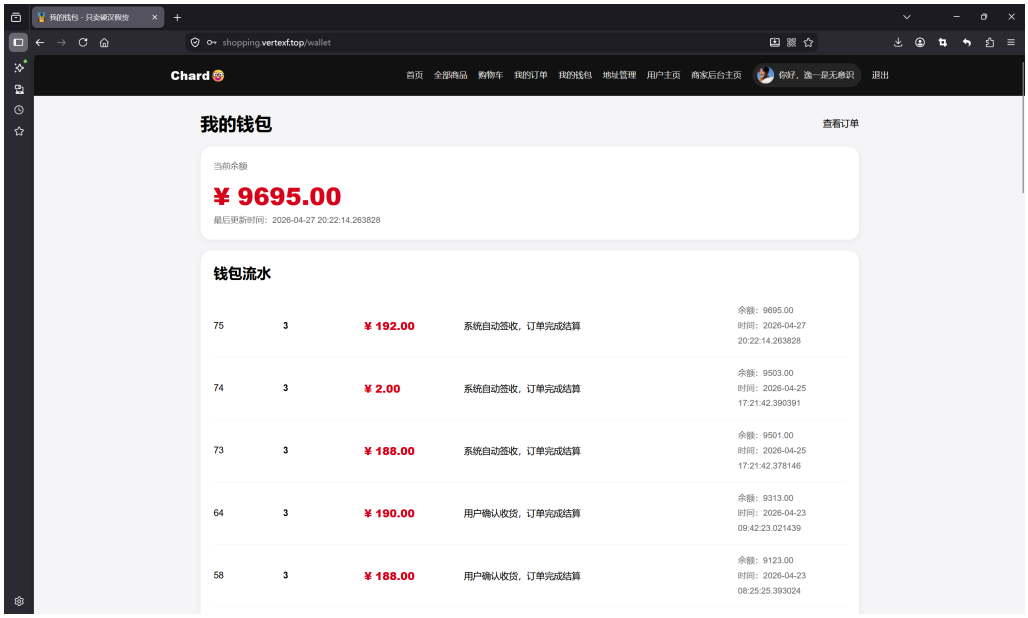

图 5.47: 商家账号钱包页面中的结算结果

6 管理员后台功能测试

管理员后台功能测试主要验证管理员身份下的平台治理功能是否能够正常运行，重点检查管理员主页、商家申请审核、钱包调账和站点设置等页面的访问与操作结果，并观察后台修改是否能够正确反映到前台页面中。当前管理员侧主要功能路由包括管理员主页 shopping.vertexf.top/admin/、商家申请审核页 shopping.vertexf.top/admin/merchant-applications、管理员钱包调账页 shopping.vertexf.top/admin/wallet-adjustments 和站点设置页 shopping.vertexf.top/admin/site-branding。

管理员登录成功后，系统首页顶部导航中会出现“管理员后台主页”和“站点设置”等后台入口，说明当前会话已正确识别管理员角色，如图 5.48 所示。进一步进入管理员主页后，页面能够集中展示商家审核、钱包调账和站点设置三类管理功能入口，并显示对应记录数量，说明管理员后台首页已经具备统一入口组织能力，如图 5.49 所示。

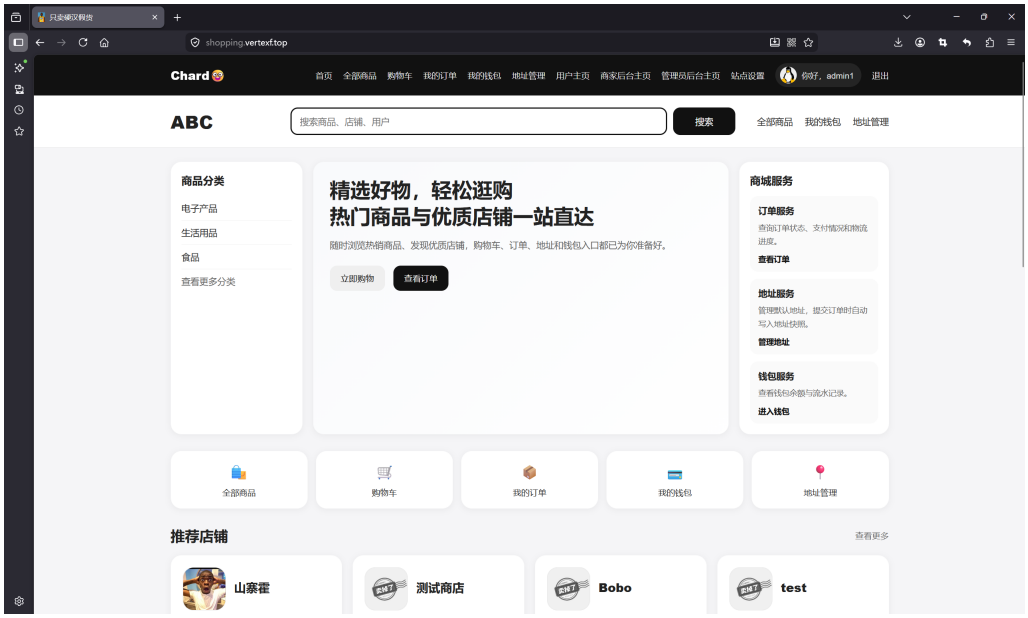

图 5.48: 管理员登录后的首页状态

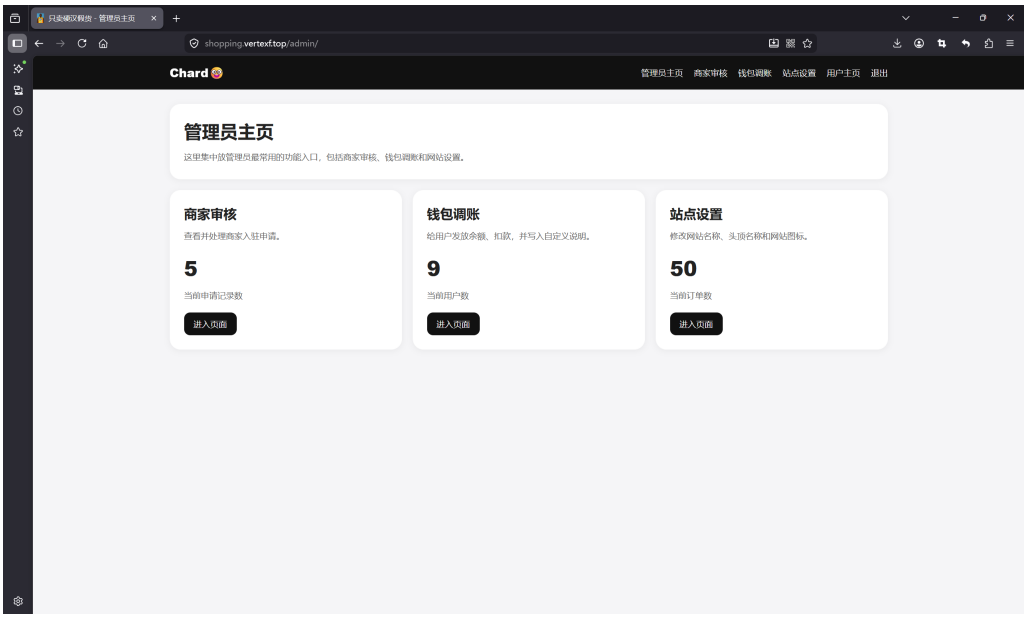

图 5.49: 管理员后台主页运行结果

在商家申请审核功能测试中，管理员可以进入商家申请审核页查看各条申请记录。页面中能够显示申请编号、申请用户、店铺名称、联系人、联系电话、经营范围、申请理由、申请状态以及审核备注等信息，并可同时看到“已通过”和“已拒绝”等不同审核结果，如图 5.50 所示。该结果说明商家申请审核记录已经能够在管理员后台集中查看，并且审核结果会被保存到申请状态中。

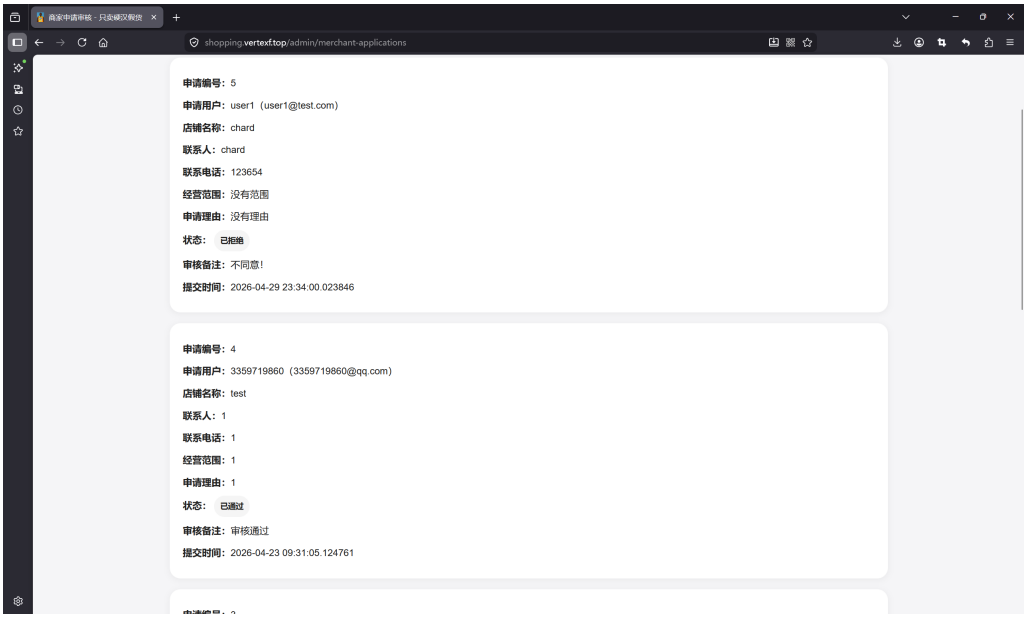

图 5.50: 商家申请审核页面运行结果

在钱包调账测试中，管理员能够进入钱包调账页，选择目标用户、操作类型和金额，并填写调账说明。页面右侧还会列出当前可操作用户及其余额信息。图 5.51 展示了调账提交后的结果，页面给出了“已成功处理 1 个用户”的提示，说明管理员调账功能能够正常执行，并完成对指定用户钱包余额的调整。

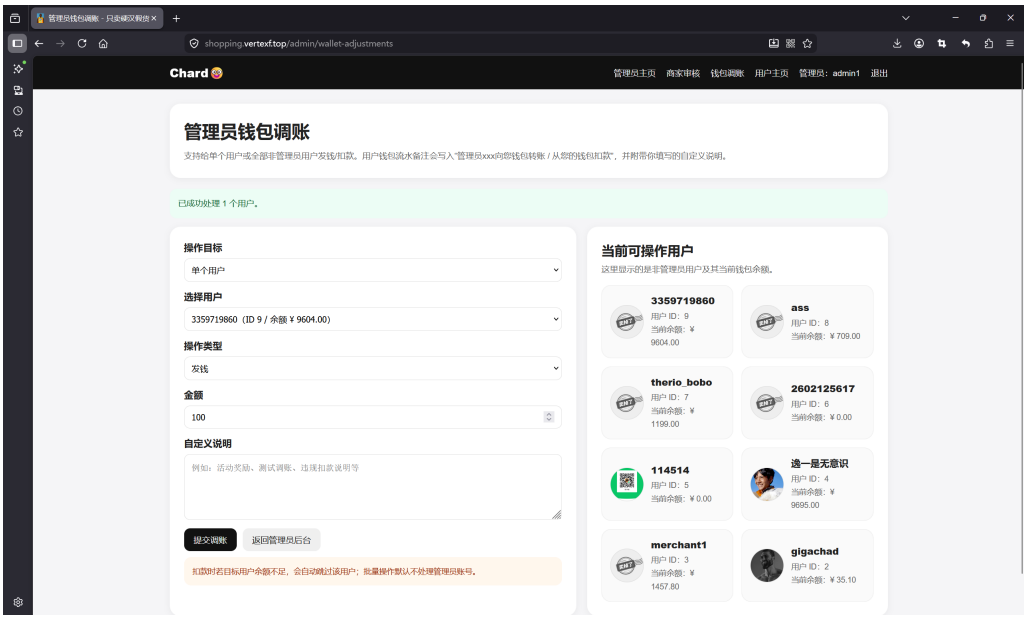

图 5.51: 管理员钱包调账页面运行结果

站点设置功能测试中，管理员可在站点设置页修改网站名称、首页头项名称、浏览器标题和网站图标。图 5.52 展示了站点设置页面的初始状态，图 5.53 展示了保存修改后的结果。可以看到，页面给出了“站点设置已保存”的提示，同时头项名称、浏览器标题和图标均已更新，说明站点品牌配置能够在后台完成维护。

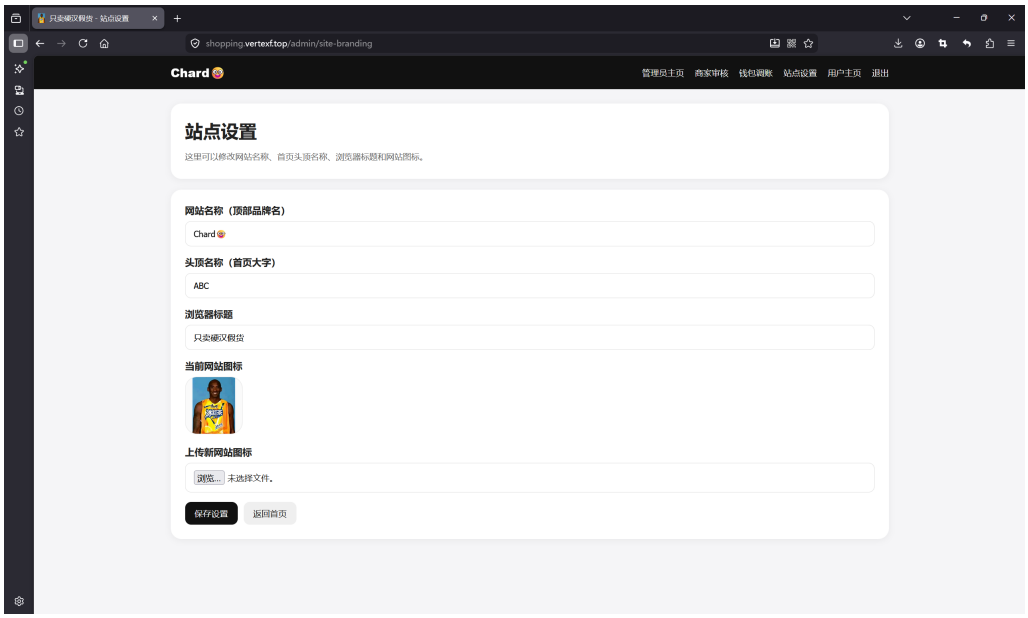

图 5.52: 站点设置页面初始状态

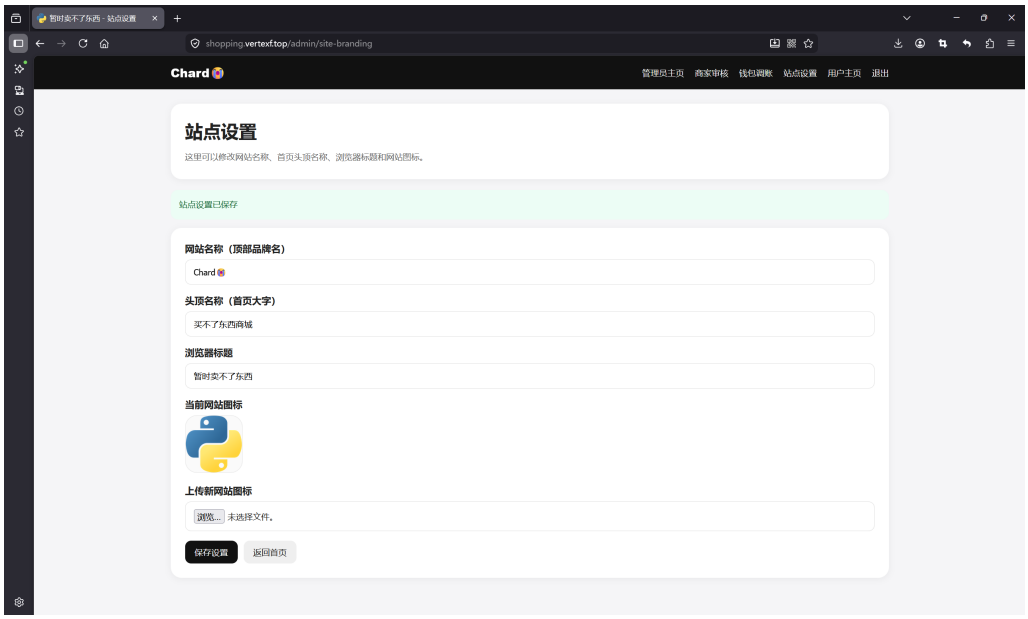

图 5.53: 站点设置保存后的结果

后台站点设置修改后，前台首页中的品牌展示内容会同步更新。图 5.54 显示，首页头项名称已经由原来的“ABC”变更为新的页面头项，浏览器标题也同步变化，说明管理员后台的站点配置修改能够直接作用于前台页面展示结果。

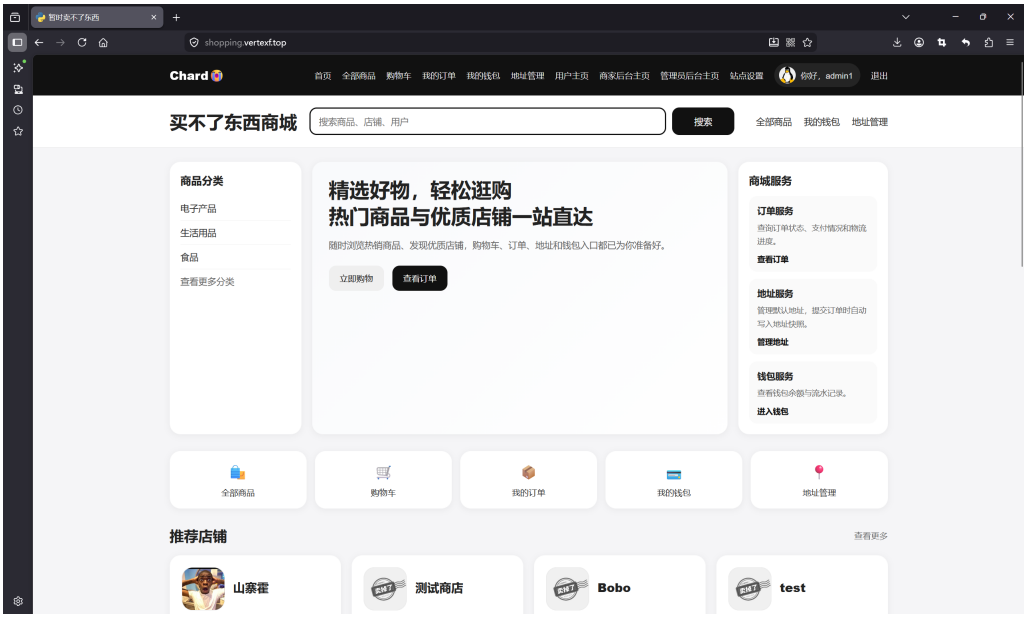

图 5.54: 站点设置修改后的前台首页结果

7 系统页面状态与运行结果分析

在完成前台公共页面、普通用户流程、商家后台和管理员后台功能测试后，还需要进一步观察系统在不同运行状态下的页面表现是否完整、是否一致。该部分不再单独验证某个页面能否访问，而是从权限状态、空数据状态、业务状态推进和前后台联动结果四个方面，对系统整体运行结果进行归纳分析。

从权限与错误状态来看，系统已经能够对不同身份的访问请求作出区分处理。未具备管理员权限的商家账号访问管理员后台时，系统返回统一的 403 无权访问页面，如图 5.46 所示；访问不存在的地址时，系统返回统一的 404 页面，如图 5.16 所示。说明系统在权限边界和错误请求场景下，并不会直接暴露默认异常页面，而是能够给出统一的页面反馈。

从空数据状态来看，系统在无有效数据时也提供了明确提示，而非出现空白页面或异常结果。例如购物车为空时，页面会提示用户当前购物车为空并引导继续浏览商品，如图 5.23 所示；搜索输入特殊字符且未匹配到结果时，页面仍能正常返回空结果状态，如图 5.17 所示。说明系统在空状态展示上已经具有基本完整性。

在业务状态推进方面，系统能够较清晰地反映申请审核、订单运输、订单完成与订单取消等过程。以商家入驻申请为例，当申请被管理员拒绝后，申请页会展示当前申请状态、审核备注和提交时间，同时保留重新填写申请信息的入口，如图 5.55 所示。这说明申请页面不仅能展示提交成功后的结果，也能容纳后续审核通过或拒绝后的状态反馈。

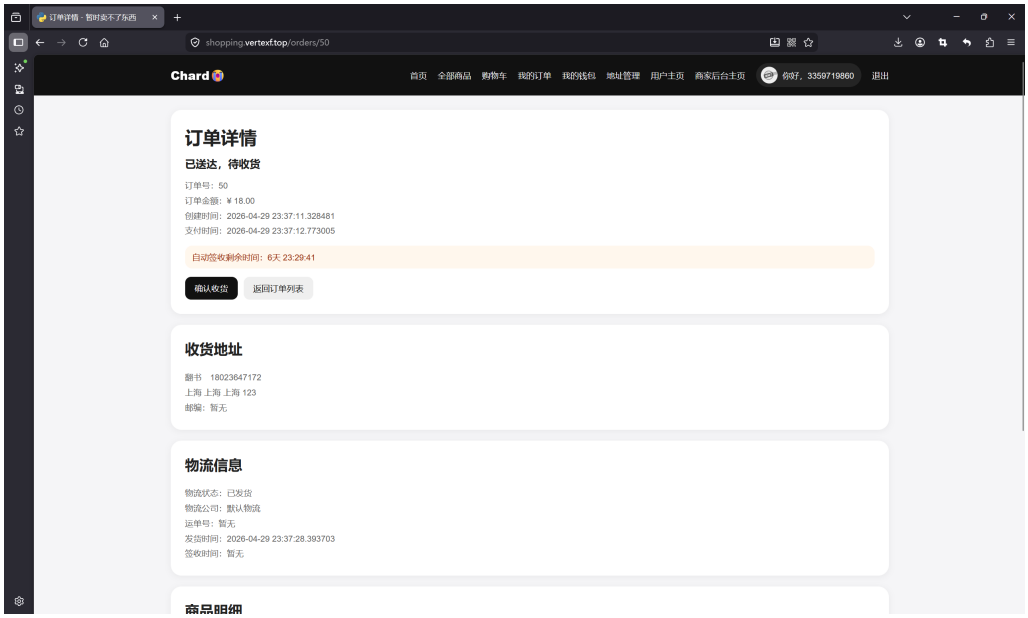

图 5.55: 商家入驻申请被拒绝后的页面状态

订单状态推进同样具有较明确的页面表现。图 5.56 展示了订单已送达、待收货时的详情页状态，此时页面显示物流状态、发货时间和确认收货按钮；图 5.57 展示了用户完成确认收货后的结果，页面中订单状态已经变为“已完成”，同时给出操作成功提示。由此可见，订单详情页能够直接反映用户操作带来的状态变化。

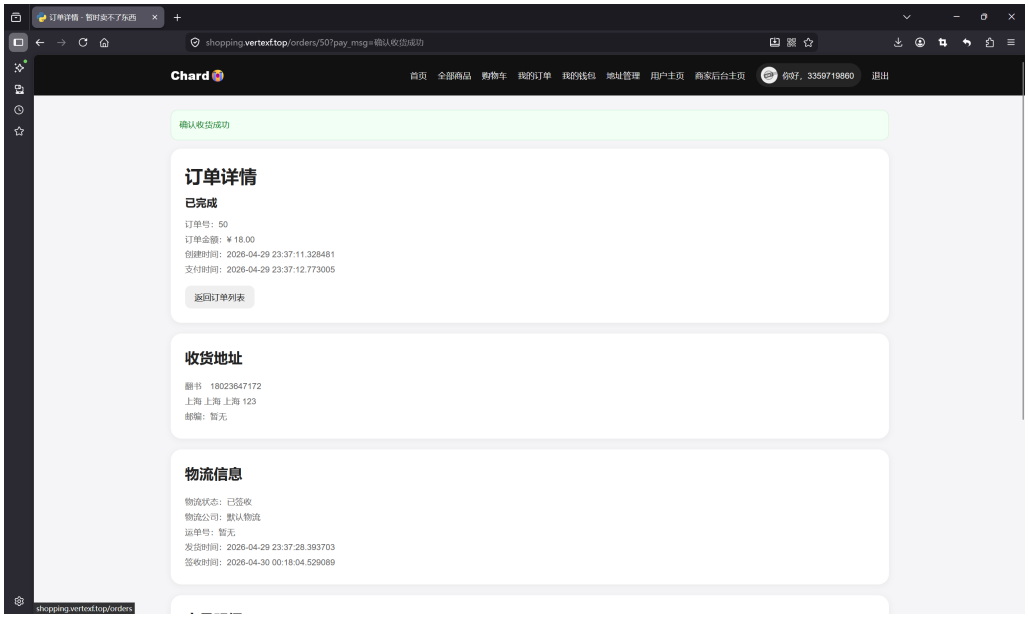

图 5.56: 订单已送达待收货时的详情状态

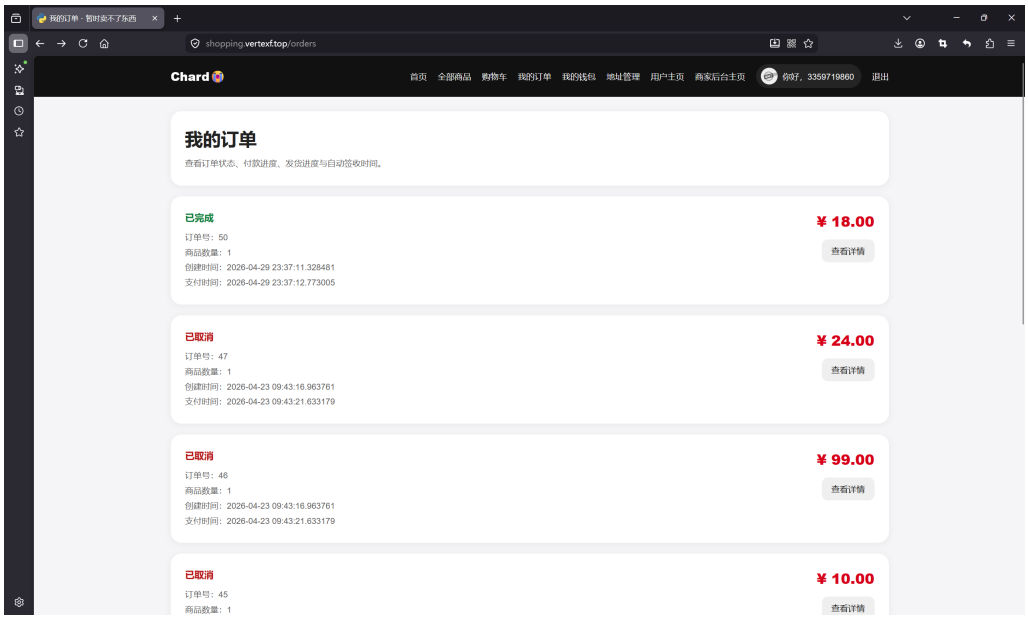

图 5.57: 确认收货后的订单完成状态

订单列表页能够进一步汇总不同订单的状态结果。图 5.58 中同时出现了“已完成”和“已取消”等不同状态，说明订单列表页已经能够集中展示订单在不同阶段下的运行结果，而不局限于单一状态。

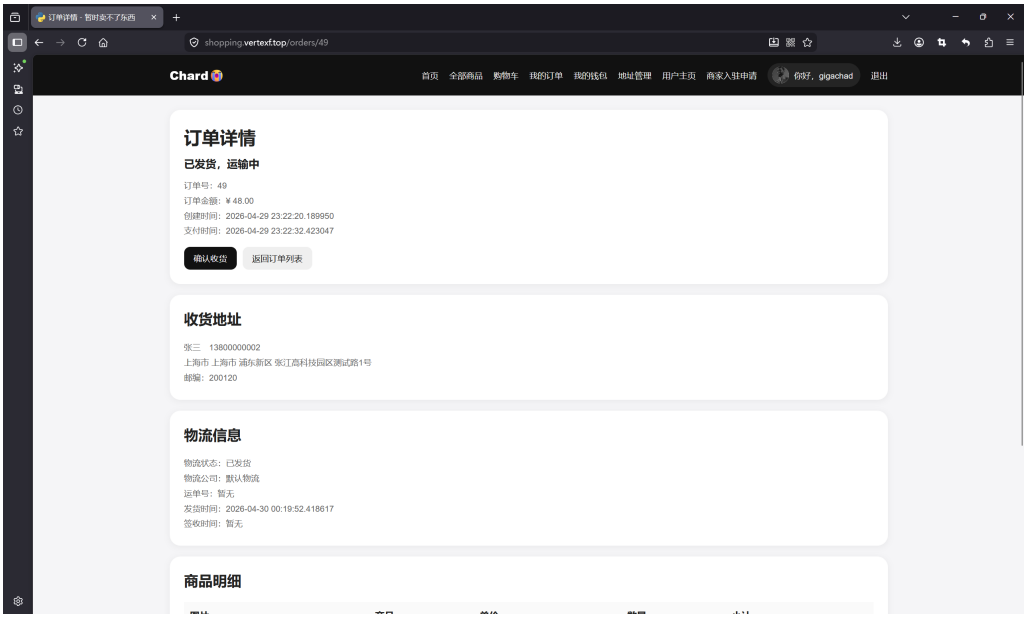

图 5.58: 订单列表中的多种订单状态结果

除正常状态推进外，系统对不满足操作条件的请求也能够给出受控反馈。图 5.59 展示了订单仍处于运输中的详情状态，此时页面保留确认收货入口；但若用户提前执行确认收货，系统不会直接完成状态变更，而是返回“商品尚未送达，暂不能确认收货”的提示，如图 5.60 所示。说明系统对订单状态推进增加了基本的条件限制，而不是允许任意越过业务阶段。

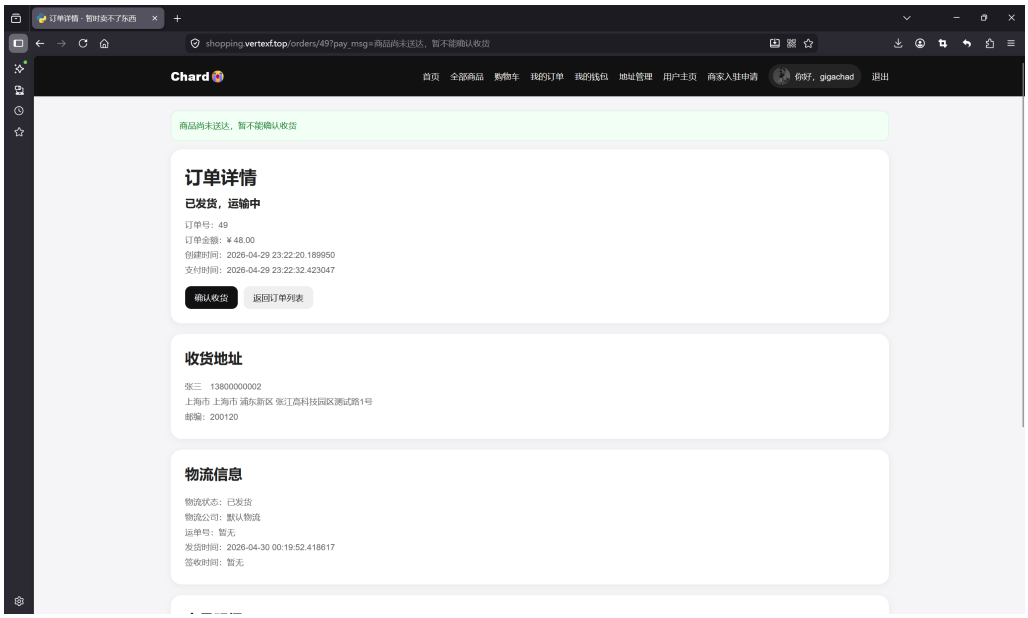

图 5.59: 运输中订单的详情页面状态

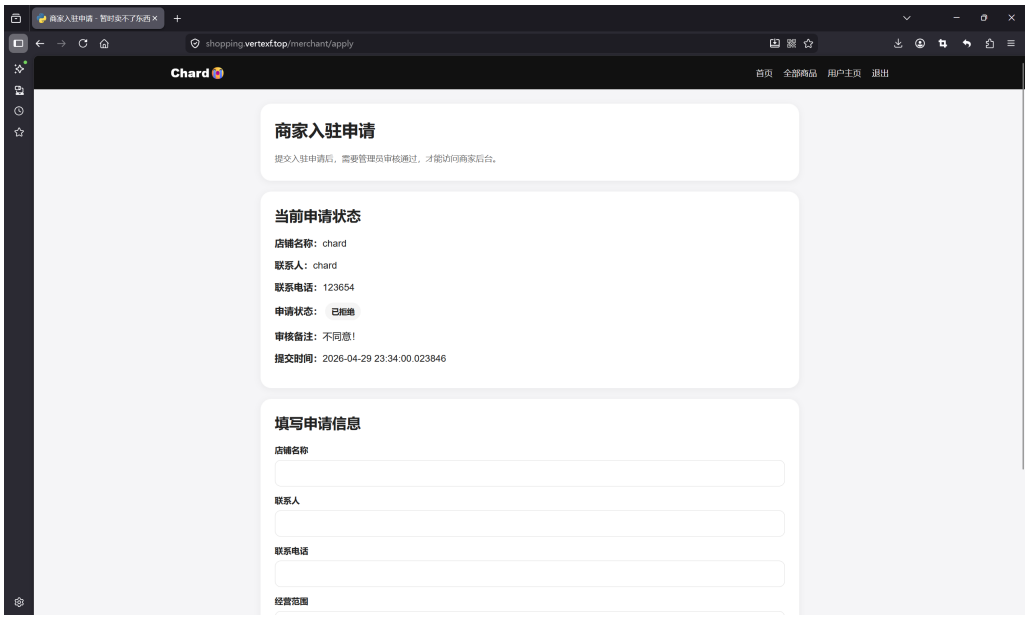

图 5.60: 未送达订单提前确认收货时的提示结果

在前后台联动结果方面，前述各节测试也已经表明，后台维护操作能够同步反映到前台页面。管理员修改站点设置后，前台首页中的头项名称、浏览器标题和图标能够同步更新，如图 5.54 所示；商家修改商品信息和店铺主页后，前台商品详情页与店铺主页也会同步更新，如图 5.38、图 5.40 和图 5.41 所示。这说明系统页面之间并非孤立运行，而是已经形成较清晰的前后台联动关系。

第六章　系统特色、不足与改进方向

1　系统特色

本系统的一个较明显特点在于功能链路较完整。系统并未停留在商品展示或用户登录这一类单点功能上，而是围绕“普通用户浏览与下单—商家维护商品与处理订单—管理员审核与配置站点”的思路，逐步形成了覆盖前台与后台的完整业务结构。用户侧已经具备商品浏览、购物车、地址管理、结算、订单查询和钱包查看等功能；商家侧能够完成商品管理、店铺主页维护和订单发货；管理员侧则能够处理商家申请、钱包调账和站点设置。这使系统在课程设计层面不只是若干页面的拼接，而是具有较明确的业务闭环。

系统在角色组织上也具有较强的层次性。数据库中采用用户、角色、用户角色关联表构成权限基础结构，程序中再结合登录态与角色判断控制页面访问范围。这样使普通用户、商家和管理员三类身份能够在同一系统中共存，并根据身份显示不同的导航、入口和后台页面。权限模型没有直接写死在单一用户字段中，而是预留了后续扩展空间，这一点对于后续增加新角色或细分后台权限具有实际意义。

系统另一项特点体现在前后台联动较为明显。商家修改商品信息后，前台商品详情页能够同步反映变化；商家维护店铺主页后，公开店铺页中的展示内容能够更新；管理员修改站点设置后，首页头项名称、浏览器标题和网站图标也会同步变化。这说明系统不是前台页面与后台页面各自独立，而是已经形成了以数据库为中介的数据驱动式页面更新机制。

在部署方式上，系统兼顾了正式运行与本地演示两种需要。正式环境中采用 Ubuntu、Nginx、Gunicorn 与 SQL Server 组合完成站点部署，并通过 HTTPS 域名对外提供服务；同时又补充了 Windows 本地一键部署包，用于课程验收、调试和演示。这种双运行方式提高了项目的可展示性，也降低了后续复现实验环境的难度。

从实现风格上看，系统重视实际运行中的页面状态反馈。无论是购物车为空、搜索无结果、权限不足、页面不存在，还是订单状态推进、申请审核结果返回，系统都尽量通过独立页面或页面提示进行反馈，而不是直接暴露原始异常信息。这使整个项目在交互完整性上比单纯完成功能更进一步。

2　系统不足

虽然系统已经完成了较为完整的主体结构，但在继续深入分析后，仍能看到不少尚未完善的部分。

当前商家模型仍然偏向“一用户——一商家——一店铺”的组织方式。现有设计

已经能够支持用户申请成为商家，并在通过审核后进入商家后台，但如果从更完整的平台化角度来看，系统尚未真正实现“单个用户管理多个商铺”的经营模式。对于需要同时运营多个店铺、区分不同店铺品牌和订单范围的场景，当前结构还不够灵活。

安全策略虽然已经完成了若干基础工作，但整体上仍处于持续加固阶段。系统已经加入了登录保护、角色访问控制、CSRF 防护、上传限制、统一错误页和部分高风险接口的提交频率控制，但这更多解决的是基础可用安全问题。若从正式生产系统要求来看，当前仍缺少更系统化的日志审计、异常行为识别、更细粒度的后台权限控制、针对上传内容的更严格校验，以及针对关键操作的二次确认机制。

部分业务功能尚未深入。当前订单流程已经覆盖下单、支付、发货、收货和结算等关键阶段，但像优惠券、退款、退货、评价、售后协商、物流轨迹接入等电商场景中的常见功能仍未真正完成。钱包模块目前主要承担支付扣款和结算流水展示作用，尚未拓展到充值、冻结金额、提现申请等更完整的资金管理过程。

系统运维能力仍较基础。当前项目已经完成服务部署、域名访问和静态资源映射，但尚未建立完整的自动备份、运行监控、性能统计、告警通知和持续部署流程。如果继续向长期运行的网站靠近，这一部分迟早需要补足。

3 改进方向

后续改进首先应放在数据结构与业务模型的扩展上。较为直接的一项改进是将现有商家结构进一步扩展为“单用户可管理多个店铺”的模式。这样用户不再被限制为只能拥有一个商家店铺，而是可以针对不同经营方向建立多个独立店铺，并分别维护商品、订单和主页内容。配合这一改进，后台也可以进一步加入店铺切换、店铺独立统计和多店铺订单筛选功能，使系统更接近真实平台场景。

邮箱注册部分可以进一步完善邮箱验证完成后的跳转、异常提示、注册失败回退、密码找回一致性等细节。引入短信验证、二次验证或登录设备识别机制，继续增强安全性，使账户体系更加稳定。

在安全方面，后续可以完善操作审计日志，对管理员审核、钱包调账、商家发货、用户确认收货等关键行为进行更清晰的日志记录；加强上传安全控制，例如进一步校验图片内容、限制更细的 MIME 类型范围、对文件名和访问路径进行更严谨的处理；对高风险操作加入更细粒度的速率限制与异常提示，避免暴力提交或频繁试探。

业务实用性方面，还有不少可以直接提升使用体验的功能。搜索页可以加入更精细的排序、筛选和分页；商品页可以补充收藏、浏览记录和相关推荐；地址管理可以加入省市区联动和默认地址自动回填；订单页可以增加按状态筛选、物流进度展示与售后入口；商家后台可以加入库存预警、销量统计、订单筛选和更灵活的店铺装修模块；管理员后台则可以增加更清晰的审核备注模板、用户查询、数据概览和操作历史记录等。

系统性能与部署方式也可以继续优化。后续可考虑将静态资源迁移到对象存储或 CDN，加快图片访问；对热门页面加入缓存；对商品列表和店铺页的查询进行进一步优化；为生产环境增加定期备份、健康检查和自动恢复脚本。将当前较集中的应用结构逐步拆分为更清晰的模块，降低后续维护难度。

第七章　参考文献

[1] 王珊，萨师煊. 数据库系统概论（第 5 版）[M/OL]. 北京：高等教育出版社，2018.

[2] 张海藩. 软件工程导论（第 6 版）[M/OL]. 北京：清华大学出版社，2013.

[3] 苗泽. Nginx 高性能 Web 服务器详解 [M/OL]. 北京：电子工业出版社，2013.

[4] 杨鹏. Linux 服务器架设 [M/OL]. 北京：清华大学出版社，2008.

[5] 刘建伟，王育民. 网络安全：技术与实践（第 3 版）[M/OL]. 北京：清华大学出版社，2017.

[6] 杨东晓，熊瑛，车碧琛. 入侵检测与入侵防御 [M/OL]. 北京：清华大学出版社，2020.

[7] 叶敏. 电子商务网站建设与维护 [M/OL]. 重庆：重庆大学电子音像出版社，2018.

[8] Pallets. Flask Documentation[EB/OL].
https://flask.palletsprojects.com/

[9] Pallets. Jinja Documentation[EB/OL].
https://jinja.palletsprojects.com/

[10] NGINX, Inc. nginx documentation[EB/OL].
https://nginx.org/en/docs/

[11] Chesneau B. Gunicorn Documentation[EB/OL].
https://docs.gunicorn.org/en/stable/

[12] Pylons Project. Waitress Documentation[EB/OL].
https://docs.pylonsproject.org/projects/waitress/en/latest/

[13] Google. Firebase Authentication Documentation[EB/OL].
https://firebase.google.com/docs/auth

[14] Google. Authenticate with Firebase using Password-Based Accounts in JavaScript[EB/OL].
https://firebase.google.com/docs/auth/web/password-auth

[15] Google. Manage Session Cookies[EB/OL].
https://firebase.google.com/docs/auth/admin/manage-cookies

[16] Canonical. Ubuntu Server documentation[EB/OL].
https://ubuntu.com/server/docs/

[17] Electronic Frontier Foundation. Certbot Instructions for Nginx on Ubuntu[EB/OL].
https://certbot.eff.org/instructions?os=ubuntufocal&ws=nginx

[18] Microsoft. SQL Server technical documentation[EB/OL].
https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver17

[19] Microsoft. Overview of SQL Server on Linux[EB/OL].
https://learn.microsoft.com/en-us/sql/linux/sql-server-linux-overview?view=sql-server-ver17

[20] mkleehammer. pyodbc Documentation[EB/OL].
https://github.com/mkleehammer/pyodbc

[21] OWASP. Cross-Site Request Forgery Prevention Cheat Sheet[EB/OL].
https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_ Prevention_Cheat_Sheet.html