基于 python 的美妆产品服务系统设计与实现

姓　　名：叶宇翎

学　　号：1230696014

指导教师：

2025 年 7 月

摘要

本文围绕基于 Python 的美妆产品服务系统展开，详细阐述其设计与实现过程，该系统旨在为用户打造便捷的美妆产品购物体验，为管理员提供高效的后台管理功能。

系统后端选用 Flask 框架，结合 SQLite 数据库与 SQLAlchemy ORM 技术。实现了涵盖用户注册登录、商品浏览、购物车管理、订单处理、商品评论等丰富的用户功能，以及数据统计、商品管理、订单管理、用户管理、分类管理等全面的管理员功能。

系统前端运用 HTML5、CSS3 和 JavaScript 技术，搭配 Jinja2 模板引擎，达成响应式设计，能适配不同设备屏幕。同时，通过 CSRF 保护、安全的会话管理、表单数据验证以及利用 ORM 防止 SQL 注入等措施，保障系统安全。

系统结构清晰合理，由主应用文件、数据模型定义、Flask 扩展配置、静态资源、HTML 模板及数据库文件等部分构成。本文详细介绍了系统的功能设计、技术实现及部署过程，并通过严格测试验证了系统的可用性与稳定性。结果显示，该系统能够满足用户与管理员的基本需求，为美妆电商行业提供了切实可行的解决方案。

关键词：Django　Flask　SQLite　响应式设计　身份验证

Abstract

This paper focuses on a beauty product service system based on Python and elaborates on its design and implementation process. The system aims to provide users with a convenient shopping experience for beauty products and offer efficient backend management functions for administrators.

The backend of the system employs the Flask framework, in conjunction with the SQLite database and SQLAlchemy ORM technology. It implements a wide range of user features, including user registration and login, product browsing, shopping cart management, order processing, and product reviews. Additionally, it provides comprehensive administrative functions such as data statistics, product management, order management, user management, and category management.

The frontend of the system utilizes HTML5, CSS3, and JavaScript, along with the Jinja2 template engine, to achieve a responsive design that can adapt to various device screens. The system also incorporates several security measures, including CSRF protection, secure session management, form data validation, and the use of ORM to prevent SQL injection, thereby ensuring the security of the system.

The system is well-structured, comprising the main application file, data model definitions, Flask extension configurations, static resources, HTML templates, and database files. This paper provides a detailed account of the system's functional design, technical implementation, and deployment process. Through rigorous testing, the system's usability and stability have been verified. The results demonstrate that the system can meet the basic needs of both users and administrators, offering a viable solution for the beauty e-commerce industry.

目　录

1 绪论

1.1 研究背景

随着互联网技术的迅猛发展以及消费者购物习惯的逐步转变，电子商务已然成为现代零售行业的关键组成部分。尤其是在美妆领域，消费者对于便捷、高效且个性化的购物体验需求日益增长。传统的美妆购物模式往往受限于实体店的地理位置、营业时间以及有限的产品展示，难以满足消费者随时随地选购美妆产品的需求。此外，消费者在购买美妆产品时，往往需要参考大量的产品信息、用户评价以及专业的使用建议，而线下购物模式在这方面存在明显的不足。

与此同时，美妆行业的市场规模不断扩大，消费者对于美妆产品的种类、品质和个性化需求也在持续提升。根据相关市场研究报告，全球美妆市场的年增长率保持在较高水平，尤其是在亚洲和欧美市场，线上美妆购物的渗透率正在快速上升。这表明，开发一个基于互联网的美妆产品服务系统，不仅能够满足消费者日益增长的购物需求，还能为美妆品牌和商家提供更广阔的市场空间和更高效的销售渠道。

此外，随着移动互联网技术的普及和智能设备的广泛使用，消费者对于购物平台的便捷性和用户体验提出了更高的要求。一个能够实现多设备适配、提供个性化推荐、支持安全支付和便捷订单管理的美妆电商系统，将成为吸引和留住用户的关键因素。同时，随着消费者对美妆产品安全性和质量的关注度不断提高，系统的安全性、可靠性和数据保护能力也成为至关重要的考量因素。

因此，设计并实现一个基于 Python 的美妆产品服务系统具有重要的现实意义和广阔的应用前景。通过整合先进 Web 开发技术、数据库管理和安全防护措施，该系统能够为用户提供高效、便捷、安全的购物体验，同时为管理员提供强大的后台管理功能，满足美妆电商行业快速发展的需求。

1.2 研究内容

本研究旨在设计并实现一个基于 Python 的美妆产品服务系统，其研究内容主要包括以下几个方面：

1. 系统需求分析：分析用户需求，研究用户在美妆产品购物过程中的行为

习惯、需求痛点以及期望的购物体验，包括商品信息展示、购物便捷性、支付安全性等方面；分析管理员需求，梳理管理员在后台管理中的各项功能需求，如商品管理、订单处理、用户管理、数据统计等，以提高管理效率和决策支持；分析系统安全性需求，评估系统在数据保护、用户隐私、支付安全等方面的需求，确保系统能够抵御常见的网络攻击和数据泄露风险。

2. 系统设计：选择合适的后端框架（如 Flask）、数据库（如 SQLite）和前端技术栈（如 HTML5、CSS3、JavaScript），设计系统的整体架构，确保系统的可扩展性、可维护性和安全性；设计合理的数据库模型，包括用户表、商品表、购物车表、订单表等，以支持系统的各项功能需求；设计用户模块（注册、登录、浏览、评论等）、购物模块（购物车、订单处理等）和管理员模块（商品管理、订单管理、用户管理等），明确各模块的功能和交互流程。

3. 系统实现：使用 Python 和 Flask 框架实现系统的后端逻辑，包括用户认证、商品管理、订单处理等核心功能；利用 HTML5、CSS3 和 JavaScript 技术，结合 Jinja2 模板引擎，实现系统的前端页面，确保页面的响应式设计和良好的用户体验；通过 CSRF 保护、会话管理、表单验证等技术手段，增强系统的安全性，防止常见的网络攻击。

4. 系统测试与优化：对系统的各项功能进行测试，确保功能的完整性和正确性；评估系统的性能，包括响应时间、并发处理能力等，优化系统性能；测试系统的安全性，确保系统能够有效抵御各种安全威胁。

5. 系统部署与维护：选择合适的部署方案（如 Gunicorn 或 uWSGI），确保系统能够在生产环境中稳定运行；制定系统的维护策略，包括数据备份、系统更新、漏洞修复等，确保系统的长期稳定运行。

1.3 研究目标

本研究的目标是设计并实现一个高效、安全、用户友好的美妆产品服务系统，具体目标如下：

1. 功能完整性：实现用户注册、登录、商品浏览、购物车管理、订单处理、商品评论等用户功能；实现数据统计、商品管理、订单管理、用户管理、分类管理等管理员功能；提供一个完整的在线购物体验，满足用户和管理员的基本需求。

2. 系统安全性：通过 CSRF 保护、安全的会话管理、表单验证和 ORM 技术，

防止 SQL 注入等安全威胁；确保用户数据和交易的安全性，保护用户隐私。

3. 用户体验：实现响应式设计，确保系统在不同设备上都能提供良好的用户体验；提供清晰的商品信息展示、便捷的购物操作流程和高效的订单处理机制。

4. 系统稳定性：通过性能测试和优化，确保系统在高并发情况下能够稳定运行；设计合理的部署方案，确保系统在生产环境中的可靠性和可扩展性。

5. 可维护性：采用清晰的代码结构和模块化设计，便于系统的维护和扩展；制定完善的维护策略，确保系统的长期稳定运行。

通过上述研究内容和目标的实现，本研究旨在为美妆电商行业提供一个高效、安全、用户友好的解决方案，推动美妆电商行业的发展。

2 相关技术介绍

2.1 Flask Web 框架

Flask 是一个用 Python 编写的轻量级 Web 应用框架。它由 Armin Ronacher 领导的 Pocoo 团队开发，首次发布于 2010 年。Flask 的核心目标是提供一个简单、灵活且易于扩展的 Web 开发基础，使开发者能够快速构建 Web 应用。它基于 Werkzeug（一个 WSGI 工具库）和 Jinja2（一个模板引擎），并提供了构建 Web 应用所需的基本功能，如路由、请求处理、模板渲染等。

Flask 的主要特点有：

1. 轻量级与灵活性：Flask 的核心非常简洁，没有强制性的依赖关系，这使得常轻量级。开发者可以根据项目需求自由选择扩展和组件，从而实现高度的定制化。例如，开发者可以选择使用不同的数据库（如 SQLite、MySQL、PostgreSQL 等）和 ORM 工具（如 SQLAlchemy），而不必受限于框架的默认选择。

2. 易于上手：Flask 的文档非常详细，提供了丰富的示例和教程，使得初学者能够快速上手。它的 API 设计简洁直观，开发者可以轻松地构建简单的 Web 应用，同时也能逐步扩展到更复杂的应用。

3. 强大的扩展支持：Flask 拥有一个丰富的扩展生态系统，这些扩展提供了各种功能，如数据库集成（Flask-SQLAlchemy）、用户认证（Flask-Login）、表单处理（Flask-WTF）、文件上传处理（Flask-Uploads）等。这些扩展使得开发者能够快速实现复杂的功能，而无需从头开始编写大量代码。

4. 内置开发服务器和调试器：Flask 内置了一个开发服务器，方便开发者在

本地进行应用开发和测试。此外，它还提供了一个交互式的调试器，当应用出现错误时，开发者可以在浏览器中直接查看错误信息并进行调试，大大提高了开发效率。

5. 安全性：Flask 提供了多种安全机制，如 CSRF 保护（通过 Flask-WTF 扩展实现）、安全的会话管理（通过 Flask-Session 扩展实现）等。这些机制能够有效防止常见的 Web 安全威胁，如跨站请求伪造（CSRF）和会话劫持。

Flask 作为一个轻量级的 Web 框架，以其简洁、灵活和强大的扩展支持，成为 Python Web 开发中的一个热门选择。在本项目中，Flask 为美妆产品服务系统提供了稳定、高效且易于扩展的后端支持，帮助我们快速实现了系统的各项功能。尽管存在一些局限性，但通过合理的设计和扩展，Flask 能够满足大多数 Web 开发需求，是一个值得推荐的框架选择。

2.2 前端开发技术

在现代Web开发中，前端开发技术是构建用户界面和交互体验的核心。本系统采用了HTML5、CSS3、JavaScript和Jinja2模板引擎作为前端开发的主要技术栈，以实现一个响应式、动态且用户友好的美妆产品服务系统。

2.2.1 HTML5

HTML5（HyperText Markup Language 5）是HTML的最新版本，用于构建和呈现互联网内容。HTML5在传统HTML的基础上引入了许多新特性，增强了Web页面的语义化、多媒体支持和交互性。

HTML5引入了如<header>、<nav>、<section>等语义化标签，这些标签不仅有助于搜索引擎优化（SEO），还能使页面结构更加清晰，便于开发者和维护者理解；提供了<audio>和<video>标签，使得在网页中嵌入音频和视频内容变得非常简单，无需依赖第三方插件；为表单控件提供了更多新特性，如placeholder属性、required属性、pattern属性等，增强了表单的用户体验和数据验证能力；结合CSS3的媒体查询（Media Queries），可以实现响应式设计，使网页能够自动适应不同设备的屏幕尺寸。

在本项目中，HTML5用于构建页面的基本结构，确保页面内容的语义化和可访问性。例如，商品详情页使用<section>标签划分不同的内容区域，使用<article>标签包裹商品描述，使用<form>标签构建用户评论表单。

2.2.2 CSS3

CSS3是CSS的最新版本，用于控制HTML文档的外观和格式。CSS3引入了许多新特性，极大地增强了页面的视觉效果和布局能力。

CSS3提供了更多的样式属性，如border-radius（圆角边框）、box-shadow（阴影效果）、text-shadow（文本阴影）等，使得页面设计更加丰富和美观；@keyframes规则和transition属性使得开发者可以轻松实现复杂的动画和过渡效果，提升用户体验；媒体查询允许开发者根据设备的屏幕尺寸、分辨率等条件应用不同的样式规则，实现响应式设计；引入了Flexbox和Grid布局，这些布局工具使得页面布局更加灵活和强大，能够轻松实现复杂的多列布局和响应式布局。

在本项目中，CSS3用于实现页面的视觉设计和布局。例如，使用圆角和阴影效果，使用媒体查询实现页面的响应式布局，确保在不同设备上都能提供良好的用户体验。

2.2.3 JavaScript

JavaScript是一种运行在浏览器中的脚本语言，用于实现页面的交互功能。JavaScript的灵活性和强大的功能使其成为Web开发中不可或缺的一部分。

JavaScript可以响应用户的操作，如点击、输入、滚动等，实现动态的页面效果。例如，用户点击购物车图标时，JavaScript可以动态加载购物车内容；可以在用户提交表单之前进行数据验证，确保用户输入的数据符合要求，减少服务器端的验证压力；通过AJAX技术可以实现与服务器的异步通信，无需重新加载页面即可更新页面内容；拥有丰富的框架和库，如jQuery、Vue.js、React等，这些工具可以简化开发过程，提高开发效率。

在本项目中，JavaScript用于实现页面的交互功能，如购物车的动态更新、用户评论的异步提交、商品详情页的图片轮播等。通过JavaScript，我们能够为用户提供更加流畅和动态的购物体验。

2.2.4 Jinja2 模板引擎

Jinja2是一个现代的、设计者友好的、基于Python的模板引擎，用于生成HTML、XML或其他标记语言的文本。Jinja2模板引擎通过模板文件和Python代码的结合，实现了动态页面的生成。

Jinja2支持模板继承，允许开发者定义一个基础模板，然后在其他模板中继承并扩展这个基础模板。这使得页面的公共部分（如头部、尾部）可以复用，减少了代码重复；通过模板文件和Python代码的结合，实现了动态页面的生成。开发者可以在模板中使用变量、循环、条件语句等，将后端数据动态地插入到HTML页面中；提供了自动转义功能，可以防止XSS（跨站脚本攻击）等安全问题。开发者也可以手动控制变量的转义，确保页面的安全性。

在本项目中，Jinja2模板引擎用于实现页面的动态生成。例如，在商品列表页，Jinja2模板会根据数据库中存储的商品信息动态生成商品卡片；在用户订单页，Jinja2模板会根据用户的订单数据动态生成订单列表。通过Jinja2模板引擎，我们能够实现页面的动态化和个性化，提升用户体验。

HTML5、CSS3、JavaScript和Jinja2模板引擎共同构成了本系统的前端开发技术栈。通过这些技术的结合，我们构建了一个响应式、动态且用户友好的美妆产品服务系统，为用户提供了良好的购物体验。

2.3 数据库技术

在本项目中，我们采用了SQLite数据库和SQLAlchemy ORM技术来实现数据的存储和管理。这两种技术的结合为系统提供了高效、灵活且易于开发的数据处理解决方案。

2.3.1 SQLite

SQLite 是一种轻量级的嵌入式关系型数据库，它以单一磁盘文件存储所有数据，无需单独的服务器进程即可运行。这使得 SQLite 非常适合小型到中型的应用程序，尤其是在开发和测试阶段。

SQLite 的设计目标是简单、轻量且易于使用。它不需要复杂的安装和配置，也不需要单独的数据库服务器，这使得它非常适合快速开发和小型项目；SQLite将所有数据存储在一个单一的磁盘文件中，这使得数据的备份和迁移变得非常简单。开发者可以通过简单的文件复制操作来备份数据库，或者将数据库文件迁移到其他系统；尽管 SQLite 是轻量级的，但它仍然支持 ACID（原子性、一致性、隔离性、持久性）特性，确保数据的完整性和可靠性；SQLite 支持大多数 SQL 标准，包括事务、外键、视图等高级特性，这使得开发者可以使用标准的 SQL 语句来操作数据库。

在本项目中，SQLite 作为后端数据库，用于存储用户信息、商品信息、订单数据等。它为系统提供了快速开发和测试的便利性，同时也满足了生产环境中的基本需求。

2.3.2 SQLAlchemy ORM

SQLAlchemy是Python中最流行的SQL工具包和对象关系映射（ORM）工具。它提供了数据库的高级抽象，使得开发者可以使用 Python 对象来操作数据库，而无需直接编写SQL语句。

SQLAlchemy的核心功能是将Python类映射到数据库表中。开发者可以通过定义 Python 类来描述数据库表的结构，SQLAlchemy会自动处理SQL语句的生成和执行；提供了丰富的数据库操作接口，包括查询、插入、更新和删除等。通过ORM，开发者可以使用Python的语法来操作数据库，而无需直接编写SQL语句；支持事务操作，确保数据的一致性和完整性。开发者可以通过事务来管理数据库的修改，确保在出现错误时能够回滚到安全状态；提供了数据库迁移工具（如Alembic），使得开发者可以方便地管理数据库的版本和迁移。这在开发过程中非常有用，尤其是在团队开发和持续集成环境中。

在本项目中，SQLAlchemy ORM 用于实现数据库的抽象和操作。通过定义 Python 类（如 User、Product、Order 等），我们将数据库表映射为 Python 对象，使得开发者可以使用 Python 的语法来操作数据库。

通过 SQLite 和 SQLAlchemy ORM 的结合，本项目实现了高效、安全且灵活的数据存储和管理，为系统的稳定运行提供了坚实的基础。

2.4 Web 安全技术

在现代 Web 开发中，安全性是至关重要的。Web 应用面临着各种安全威胁，如跨站请求伪造（CSRF）、会话劫持、SQL 注入等。为了确保系统的安全性和可靠性，本项目采用了多种 Web 安全技术，包括 CSRF 保护和会话管理等。这些技术共同构成了系统的安全防线，保护用户数据和交易安全。

2.4.1 CSRF 保护

CSRF 是一种常见的 Web 安全漏洞，攻击者通过伪装成用户的身份，向 Web 应用发送恶意请求，从而执行未经授权的操作。例如，攻击者可能会在用户不知

情的情况下修改用户的账户信息或提交订单。

CSRF 的工作原理是：攻击者通过电子邮件、即时通讯或恶意广告等方式，诱导用户访问一个包含恶意代码的网站；当用户访问恶意网站时，恶意代码会自动向目标 Web 应用发送请求，这些请求会携带用户当前的会话信息（如 Cookie），从而使 Web 应用误以为请求是由用户自己发起的；目标 Web 应用在验证了会话信息后，会执行请求中的操作，如修改用户信息、提交订单等。

在本项目中，我们采用了以下措施来防止 CSRF 攻击：

1. 使用 CSRF 令牌：通过 Flask-WTF 扩展，我们在每个表单中嵌入一个 CSRF 令牌。当用户提交表单时，服务器会验证令牌的有效性。如果令牌无效，请求将被拒绝。

2. 验证 HTTP Referer 头：在某些情况下，我们还会检查 HTTP 请求的 Referer 头，确保请求来自合法的页面。

3. 限制会话 Cookie 的使用范围：通过设置 SESSION_COOKIE_HTTPONLY 和 SESSION_COOKIE_SAMESITE 属性，我们限制了会话 Cookie 的使用范围，防止恶意网站通过 JavaScript 访问 Cookie。

2.4.2 会话管理

会话管理是 Web 应用中用于跟踪用户状态的重要机制。它通过在用户浏览器中存储会话标识来实现。如果会话管理不当，可能会导致会话劫持等安全问题。

例如：会话劫持，攻击者通过窃取用户的会话标识（如 Cookie），伪装成用户的身份，访问用户的账户信息或执行未经授权的操作；会话固定攻击：攻击者通过诱导用户使用一个已知的会话标识，从而在用户登录后获取该会话的控制权。

在本项目中，我们采用了以下措施来确保会话管理的安全性：

1. 使用安全的会话标识：通过 Flask 的`session`机制，我们为每个用户生成一个唯一的会话标识，并将其存储在浏览器的 Cookie 中。

2. 设置 Cookie 属性：

HttpOnly：设置`SESSION_COOKIE_HTTPONLY=True`，防止 JavaScript 访问会话 Cookie，从而减少会话标识被窃取的风险。

Secure：设置`SESSION_COOKIE_SECURE=True`，确保会话 Cookie 仅通过

HTTPS 传输，防止中间人攻击。

SameSite：设置`SESSION_COOKIE_SAMESITE='Lax'`，限制会话 Cookie 的跨站点使用，防止 CSRF 攻击。

3. 会话超时：设置会话的超时时间，当用户长时间未活动时，自动注销会话，减少会话被劫持的风险。

4. 会话更新：在用户登录或执行敏感操作时，更新会话标识，防止会话固定攻击。

除了 CSRF 保护和会话管理，本项目还采用了以下安全措施：

1. SQL 注入防护：通过使用 SQLAlchemy ORM，我们避免了直接拼接 SQL 语句，从而有效防止了 SQL 注入攻击。

2. 表单验证：在用户提交表单时，我们通过 Flask-WTF 扩展对表单数据进行验证，确保用户输入的数据符合要求。

3. 密码加密：用户密码在存储到数据库之前，通过哈希算法进行加密，确保密码的安全性。

4. HTTPS：在生产环境中，我们建议使用 HTTPS 协议，确保数据传输的安全性。

通过采用 CSRF 保护、会话管理、SQL 注入防护、表单验证和密码加密等多种安全技术，本项目构建了一个安全可靠的 Web 应用。这些技术共同构成了系统的安全防线，保护用户数据和交易安全，为用户提供了一个安全、可靠的购物环境。

2.5 其他相关技术

在本项目中，除了核心的后端和前端技术外，还采用了多种其他相关技术，包括 Bootstrap/Tailwind CSS、AJAX 和 RESTful API 等。这些技术进一步提升了系统的用户体验、开发效率和可维护性。

2.5.1 Bootstrap、Tailwind CSS

Bootstrap 和 Tailwind CSS 是两种流行的前端 CSS 框架，用于快速构建响应式和现代的 Web 界面。

Bootstrap 是一个开源的前端框架，提供了丰富的 HTML、CSS 和 JavaScript 组件，用于快速构建响应式 Web 应用。它包含预定义的样式和布局组件，如导航

栏、按钮、卡片、模态框等，使得开发者能够快速实现复杂的页面设计。

在本项目中，Bootstrap 用于快速构建页面的基本布局和组件，例如商品列表页的卡片布局、用户登录注册表单等。通过使用 Bootstrap 的预定义样式，我们能够快速实现响应式设计，减少 CSS 代码的编写量。

Tailwind CSS 是一个实用工具优先的 CSS 框架，它提供了一系列低级的 CSS 工具类，开发者可以通过这些工具类直接在 HTML 中构建自定义的设计。与 Bootstrap 不同，Tailwind CSS 不提供预定义的组件样式，而是通过组合工具类来实现设计，这使得 Tailwind 更加灵活和可定制。

在本项目中，Tailwind CSS 用于实现高度定制化的页面设计，特别是在需要精细控制样式的情况下。例如，商品详情页的布局和样式可以通过 Tailwind CSS 的工具类进行灵活调整。

2.5.2 AJAX

AJAX 是一种用于在不重新加载整个页面的情况下与服务器异步通信的技术。它通过 JavaScript 的 XMLHttpRequest 对象或现代的 fetch API 实现，使得页面可以动态更新内容，提升用户体验。

AJAX 允许页面在后台与服务器通信，而无需重新加载整个页面；页面内容可以根据用户操作动态更新，例如加载更多商品、提交评论等；少了不必要的页面加载，提高了应用的响应速度。

在本项目中，AJAX 用于实现购物车的动态更新。当用户点击“加入购物车”按钮时，通过 AJAX 将商品信息发送到服务器，并动态更新购物车的数量和内容，而无需重新加载整个页面；另外是用户提交评论时，通过 AJAX 将评论数据发送到服务器，并动态更新页面上的评论列表，提升用户体验。

2.5.3 RESTful API

RESTful API 是一种基于 HTTP 协议的 Web 服务接口设计风格，它通过标准的 HTTP 方法（如 GET、POST、PUT、DELETE）来操作资源。RESTful API 的设计使得 Web 应用之间的通信更加简单和标准化。

在本项目中，后端通过 Flask 框架实现了 RESTful API，前端通过 AJAX 调用这些 API 来实现数据的交互。例如，用户浏览商品时，前端通过 GET 请求获取商品列表；用户提交订单时，前端通过 POST 请求将订单数据发送到服务器。

同时，RESTful API 也便于未来扩展，例如支持移动应用或第三方服务调用。

通过采用 Bootstrap/Tailwind CSS、AJAX 和 RESTful API 等技术，本项目不仅提升了用户体验，还提高了开发效率和系统的可维护性。Bootstrap/Tailwind CSS 提供了快速构建响应式界面的能力，AJAX 实现了页面的动态更新，而 RESTful API 则为前后端分离和未来扩展提供了坚实的基础。这些技术的结合使得系统更加高效、灵活且易于扩展。

3 需求分析

3.1 系统可行性分析

在开发基于 Python 的美妆产品服务系统之前，进行全面的可行性分析是确保项目成功实施的关键步骤。本系统从技术可行性、经济可行性和操作可行性三个方面进行了详细的评估。

1. 技术可行性

技术可行性主要评估现有技术是否能够支持系统的设计和实现。本系统采用的技术栈包括 Python、Flask 框架、SQLite 数据库、HTML5、CSS3、JavaScript 以及 Jinja2 模板引擎。这些技术均为成熟且广泛应用于 Web 开发领域，具有良好的社区支持和丰富的文档资源。

Python 作为一种高级编程语言，Python 具有简洁的语法和强大的库支持，适合快速开发。Flask 框架提供了灵活的 Web 开发解决方案，能够满足系统的需求。

SQLite 作为轻量级的嵌入式数据库，SQLite 无需单独的服务器进程，适合小型到中型项目。SQLAlchemy ORM 进一步简化了数据库操作，提高了开发效率。

前端技术 HTML5、CSS3 和 JavaScript 是现代 Web 开发的基础技术，能够实现响应式设计和动态交互效果。Jinja2 模板引擎则提供了高效的页面渲染能力。

此外，系统采用了多种安全技术，如 CSRF 保护、会话管理、SQL 注入防护等，确保系统的安全性。这些技术的结合使得系统在技术上完全可行。

2. 经济可行性

经济可行性主要评估项目的成本和预期收益。从开发成本来看，本系统采用的技术栈均为开源技术，无需额外的软件许可费用。开发团队可以利用现有的

Python 和 Web 开发知识，快速上手并完成开发任务。此外，系统基于轻量级的 Flask 框架和 SQLite 数据库，减少了硬件资源的需求，降低了服务器成本。

从预期收益来看，美妆电商市场具有巨大的增长潜力。通过提供便捷的购物体验和高效的后台管理功能，系统能够吸引更多的用户和商家，从而增加销售额和市场份额。因此，从经济角度来看，本项目具有较高的可行性。

3. 操作可行性

操作可行性主要评估系统的易用性和维护性。本系统采用了清晰的模块化设计，代码结构简洁明了，便于开发和维护。系统提供了用户友好的前端界面，支持响应式设计，能够适配不同设备的屏幕尺寸，提升了用户体验。同时，系统后台管理功能强大，支持数据统计、商品管理、订单管理等，能够满足管理员的日常管理需求。

此外，系统采用了多种自动化工具，如代码版本控制（Git）、自动化测试工具等，进一步提高了开发效率和系统的稳定性。因此，从操作角度来看，本系统具有较高的可行性和可维护性。

综合考虑技术可行性、经济可行性和操作可行性，本基于 Python 的美妆产品服务系统具有较高的实施可行性。通过采用成熟的技术栈和安全措施，系统能够满足用户和管理员的需求，同时在开发成本和预期收益方面具有明显的优势。因此，本项目具有较高的实施价值和市场潜力。

3.2 数据需求分析

本系统旨在为用户提供一个便捷、高效的美妆产品购物平台，同时为管理员提供强大的后台管理功能。以下是系统功能需求的详细分析，包括用户端和管理员端的功能需求。

3.2.1 用户端功能需求

用户端功能需求主要围绕用户在美妆产品购物过程中的行为习惯和需求展开，旨在提供一个流畅、便捷的购物体验。

1. 用户注册与登录：用户可以通过电子邮箱或手机号码进行注册，设置用户名和密码；提供第三方登录（如微信、QQ、微博）选项，方便用户快速登录；登录后，用户可以查看个人资料、修改密码、绑定手机号码等。

2.商品浏览：用户可以在首页浏览所有商品，支持按分类（如彩妆、护肤、香水等）筛选商品；支持关键词搜索功能，用户可以通过商品名称、品牌、成分等关键词快速查找商品；商品详情页展示商品的详细信息，包括图片、价格、规格、成分、使用方法、用户评价等。

3.购物车管理：用户可以将商品添加到购物车，支持修改购物车中商品的数量或删除商品；购物车页面显示商品的总价和优惠信息，用户可以随时查看购物车内容；提供“结算”按钮，用户点击后进入订单确认页面。

4.订单处理：用户在购物车中选择商品后，点击“结算”进入订单确认页面，填写收货信息（收货人、电话、地址）；支持多种支付方式（如微信支付、支付宝、银行卡支付等），用户可以选择支付方式并完成支付；用户可以在“我的订单”页面查看所有订单的状态，包括待处理、已发货、已完成、已取消等。

5.商品评论与评分：用户购买商品后，可以在商品详情页发表评论和评分，分享使用体验；评论支持图片上传，用户可以上传使用后的效果图片；系统自动计算商品的平均评分，并在商品详情页展示。

6.个人中心：用户可以查看个人资料、订单历史、收藏夹、浏览历史等；提供地址管理功能，用户可以添加、编辑或删除收货地址；提供订单跟踪功能，用户可以实时查看订单的物流信息。

3.2.2 管理员端功能需求

管理员端功能需求主要围绕后台管理展开，旨在为管理员提供高效、便捷的管理工具，以支持系统的日常运营和数据分析。

1.用户管理：管理员可以查看所有用户的信息，包括用户名、注册时间、订单数量等；支持对用户进行分类管理，如普通用户、VIP 用户等；管理员可以删除违规用户或禁用用户账户。

2.商品管理：管理员可以添加新商品，设置商品的名称、价格、分类、库存、品牌、成分、使用方法等信息；支持批量导入商品信息，管理员可以通过 CSV 文件批量添加商品；管理员可以编辑现有商品信息，包括修改价格、更新库存、更新图片等；支持商品分类管理，管理员可以添加、编辑或删除商品分类。

3.订单管理：管理员可以查看所有订单的详细信息，包括订单编号、用户信

息、订单金额、订单状态；支持对订单进行状态更新，如将订单状态从“待处理”更新为“已发货”；管理员可以查看订单的物流信息，并将物流信息同步到用户端。

4.数据统计与分析：管理员可以查看商品总数、用户总数、订单总数、总收入等统计数据；提供订单统计功能，管理员可以按时间段（如日、周、月）查看订单数量和销售额；提供商品销售排行榜，管理员可以查看哪些商品最受欢迎；提供用户行为分析，如用户浏览历史、购买偏好等。

5.评论管理：管理员可以查看所有用户评论，支持对评论进行审核和删除；提供评论筛选功能，管理员可以根据商品、用户、评分等条件筛选评论。

6.系统设置：管理员可以配置系统的基本信息，如网站标题、网站描述、联系方式等；支持设置支付方式、物流方式等系统参数；提供数据备份和恢复功能，管理员可以定期备份数据库，确保数据安全。

通过上述功能需求分析，本系统能够满足用户在美妆产品购物过程中的各种需求，同时为管理员提供强大的后台管理功能。用户端功能注重用户体验和便捷性，管理员端功能注重高效管理和数据分析。这些功能的实现将为用户提供一个安全、便捷、高效的美妆产品购物平台，同时也为管理员提供了强大的运营支持。

3.3 安全性需求分析

本美妆产品服务系统高度重视安全性，以保护用户数据和交易安全。系统采用多种安全措施，确保数据传输、存储和用户身份认证的安全性。首先，系统通过哈希算法加密用户密码，并在传输和存储敏感信息时使用加密技术，同时定期备份数据库以防止数据丢失。其次，系统采用多因素认证增强用户账户安全性，并通过设置会话超时防止会话劫持。此外，系统通过 CSRF 令牌、SQLAlchemy ORM 和 HTTPS 协议，分别防止跨站请求伪造攻击、SQL 注入和确保数据传输安全。同时，系统对用户输入进行严格验证，并实施基于角色的权限控制，以限制非法访问。这些综合安全措施共同构建了一个安全可靠的购物环境，有效抵御常见网络威胁，保障用户隐私和交易安全。并发量大于 100 次/秒时，系统响应时间≤1 秒。

4 系统设计

4.1 系统整体架构设计

本系统采用分层架构设计，主要包括前端展示层、后端业务逻辑层和数据持久化层。各层之间通过 RESTful API 接口进行通信，确保系统的模块化和可扩展性。

1. 前端展示层：使用 HTML5、CSS3、JavaScript 和 Jinja2 模板引擎构建，实现响应式设计和动态内容渲染。

2. 后端业务逻辑层：基于 Python 和 Flask 框架开发，处理业务逻辑和用户请求，提供 RESTful API 接口。

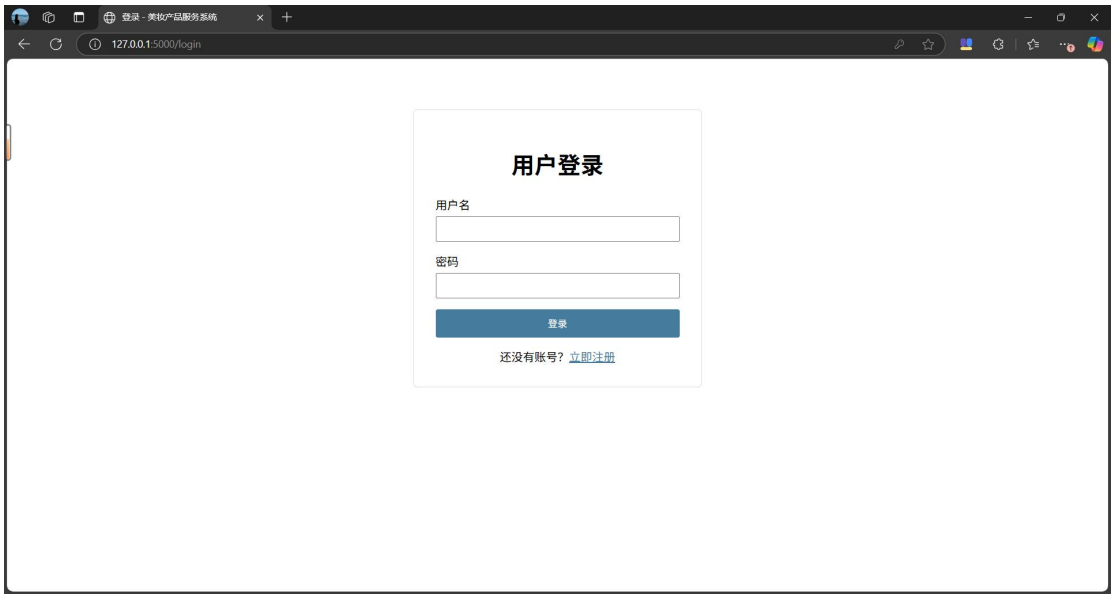
3. 数据持久化层：采用 SQLite 数据库和 SQLAlchemy ORM 实现数据存储和管理，确保数据的安全性和一致性。

4.2 功能模块划分

4.2.1 用户端模块设计

用户端模块设计旨在为用户提供便捷的购物体验，主要包括以下功能模块：

1.用户注册与登录：支持邮箱/手机号注册和第三方登录，提供用户认证和会话管理。



2. 商品浏览：支持按分类、关键词搜索商品，查看商品详情。

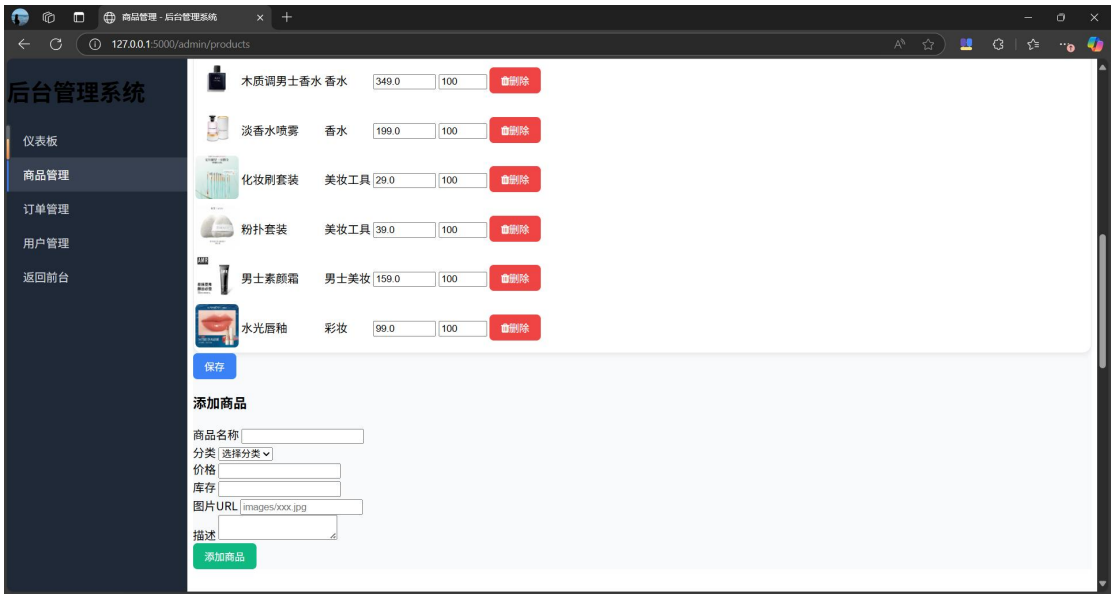
3. 购物车管理：添加、删除商品，修改数量，查看购物车总价。

4. 订单处理：填写收货信息，选择支付方式，提交订单。
5. 商品评论：用户购买后可发表评论和评分。
6. 个人中心：查看个人资料、订单历史、收藏夹、浏览历史等。

4.2.2 管理员端模块设计

管理员端模块设计旨在为管理员提供强大的后台管理功能，主要包括以下功能模块：

1. 用户管理：查看用户列表，编辑用户信息，删除用户。
2. 商品管理：添加、编辑、删除商品，管理商品分类。



3. 订单管理：查看订单列表，更新订单状态，跟踪物流信息。
4. 数据统计：统计商品总数、用户总数、订单总数、总收入等。
5. 评论管理：审核和删除用户评论。

4.3 数据库设计

4.3.1 数据表结构设计

以下是主要数据表的结构设计：

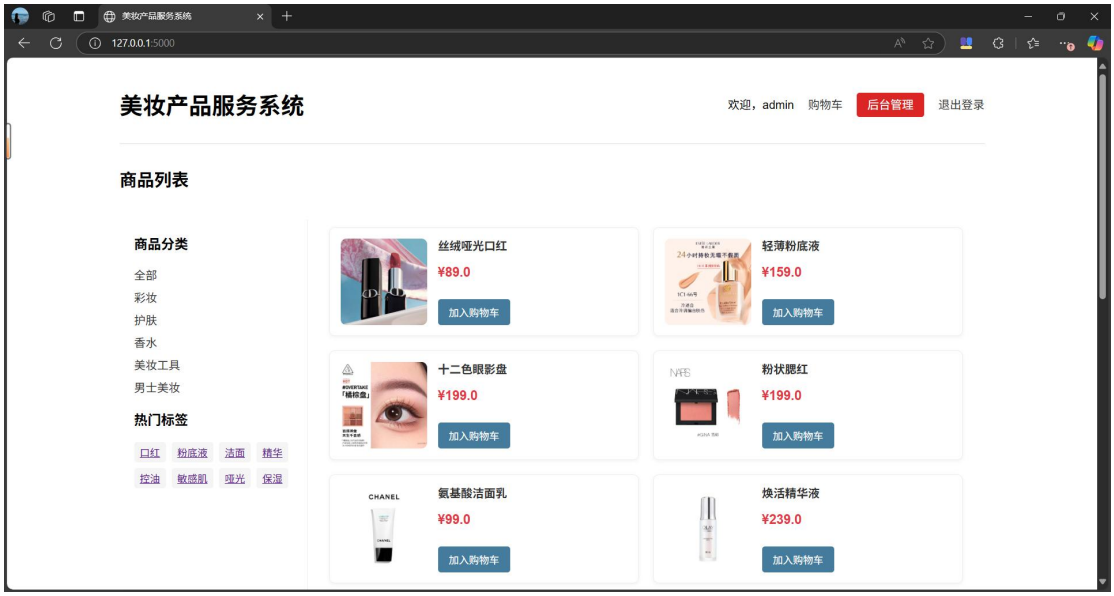
表名	字段列表						
用户表（User）	id	username	password	email	phone	created_at	
商品表（Product）	id	name	price	stock	category	brand	description
购物车表	id	user_	product_	quan			

（CartItem）		id	id	tity				
订单表（Order）	id	user_id	total_amount	status	created_at	recipient	phone	address
订单项表（OrderItem）	id	order_id	product_id	quantity	price			
评论表（Review）	id	product_id	user_id	content	rating	created_at		

4.4 前端页面设计

前端页面设计采用响应式设计，确保适配不同设备屏幕。主要页面包括：

1. 首页：展示商品分类、热门商品、推荐商品等。
2. 商品详情页：展示商品详细信息、用户评价、购买选项等。
3. 购物车页面：展示购物车中的商品、总价、结算选项等。
4. 订单确认页：填写收货信息、选择支付方式、提交订单。
5. 个人中心：查看个人资料、订单历史、收藏夹、浏览历史等。
6. 管理员后台：提供用户管理、商品管理、订单管理、数据统计等功能。



4.5 系统安全设计

系统采用多种安全措施，确保用户数据和交易安全：

1. 密码加密：用户密码通过哈希算法加密存储。
2. 会话管理：设置会话超时，防止会话劫持。
3. CSRF 保护：使用 CSRF 令牌验证表单提交。
4. SQL 注入防护：通过 SQLAlchemy ORM 避免直接拼接 SQL 语句。

5. HTTPS：在生产环境中使用 HTTPS 协议，确保数据传输安全。
6. 表单验证：对用户输入进行严格验证，防止恶意数据注入。
7. 权限控制：基于角色的权限控制，限制非法访问。

4.6 系统流程设计

4.6.1 注册流程

1. 用户访问注册页面。
2. 填写注册信息（用户名、密码、邮箱、手机号）。
3. 提交注册表单，系统验证信息合法性。
4. 系统发送验证邮件或短信，用户验证身份。
5. 注册成功，跳转到登录页面。

4.6.2 登录流程

1. 用户访问登录页面。
2. 输入用户名和密码。
3. 点击登录按钮，系统验证用户信息。
4. 登录成功，创建会话，跳转到首页。
5. 登录失败，提示错误信息。

4.6.3 下单流程

1. 用户将商品添加到购物车。
2. 点击“结算”按钮，跳转到订单确认页。
3. 用户填写收货信息，选择支付方式。
4. 点击“提交订单”，系统生成订单。
5. 系统跳转到支付页面，用户完成支付。
6. 支付成功，系统更新订单状态为“已支付”。

4.6.4 支付流程

1. 用户在订单确认页选择支付方式。
2. 点击“支付”按钮，跳转到支付页面。
3. 用户选择支付方式（如微信支付、支付宝）。
4. 用户在支付页面完成支付操作。

5. 支付成功，系统返回支付结果，更新订单状态。

6. 支付失败，提示用户支付失败信息。

5 系统实现

5.1 开发环境与工具

本系统的开发环境选择了 Windows 10 操作系统，开发语言为 Python 3.7 及以上版本。后端采用了轻量级 Web 框架 Flask 2.x，结合 SQLAlchemy 实现 ORM 数据库操作，数据库选用 SQLite 3，便于开发和测试。前端部分主要使用 HTML5、CSS3、JavaScript 以及 Jinja2 模板引擎，实现页面的动态渲染和交互。依赖管理通过 pip 工具完成，保证了项目环境的一致性。开发过程中主要使用 Visual Studio Code（或 PyCharm）作为集成开发环境，配合 Chrome 浏览器进行页面调试，接口测试则借助 Postman 工具。整体开发环境配置简洁高效，能够满足美妆电商系统的开发与测试需求。

5.2 后端实现

5.2.1 Flask 应用结构

本系统采用 Flask 作为后端开发框架，项目结构遵循“分层、模块化”的设计思想。主程序文件 app.py 负责应用的初始化，包括 Flask 实例的创建、数据库和 CSRF 扩展的配置。所有业务逻辑通过路由函数进行组织，模板文件和静态资源分别存放于 templates 和 static 目录下。数据库模型定义在 models.py 中，便于统一管理和维护。通过这种结构设计，系统具备良好的可读性和可扩展性，便于后续功能的添加和维护。

5.2.2 主要路由与业务逻辑实现

系统后端实现了丰富的业务路由，包括用户注册、登录、登出、商品浏览与详情、购物车管理、订单处理、评论提交等功能。每个路由对应一个具体的业务处理函数，负责接收前端请求、执行业务逻辑、与数据库交互，并将结果渲染到前端模板或以 JSON 格式返回。购物车和订单相关路由支持商品的添加、数量修改、结算与支付等操作，确保用户购物流程的完整性。管理员相关路由则实现了商品、用户、订单的增删改查和状态管理，保障了系统的后台运营能力。

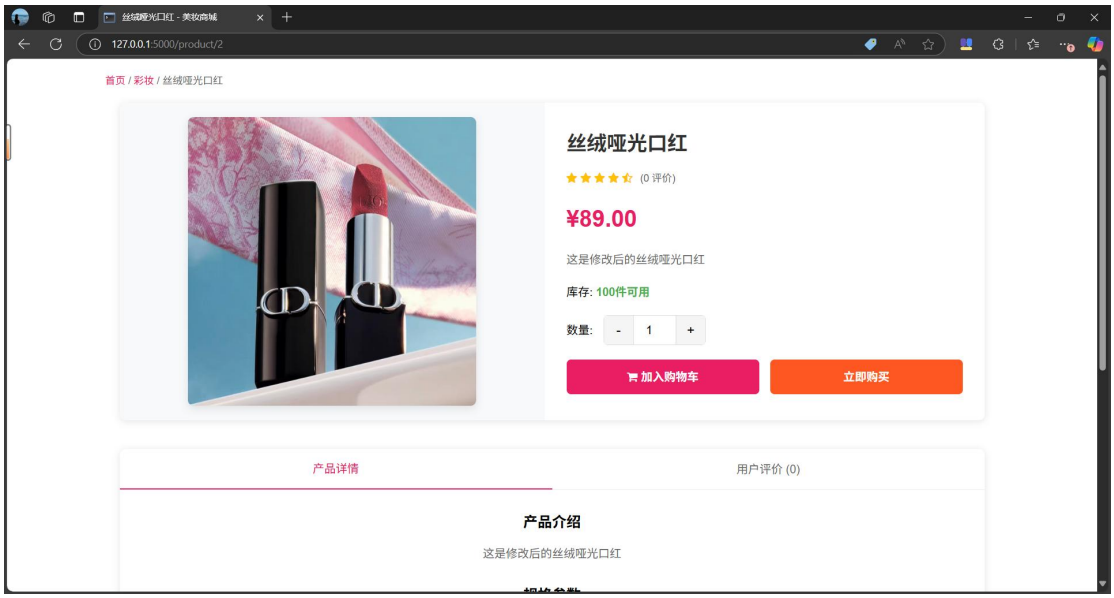
5.2.3 数据库模型实现

系统数据库采用 SQLAlchemy 进行 ORM 建模，主要包括用户（User）、商品（Product）、购物车项（CartItem）、订单（Order、OrderItem）、评论（Review）等模型。各模型通过外键建立关联，如订单与用户、订单项与商品、评论与商品等，保证了数据的完整性和一致性。每个模型均定义了必要的字段和数据类型，支持常用的增删改查操作。通过 ORM 的方式，开发者可以以面向对象的方式操作数据库，简化了数据访问层的实现，提高了开发效率和代码可维护性。

5.3 前端实现

5.3.1 页面布局与交互实现

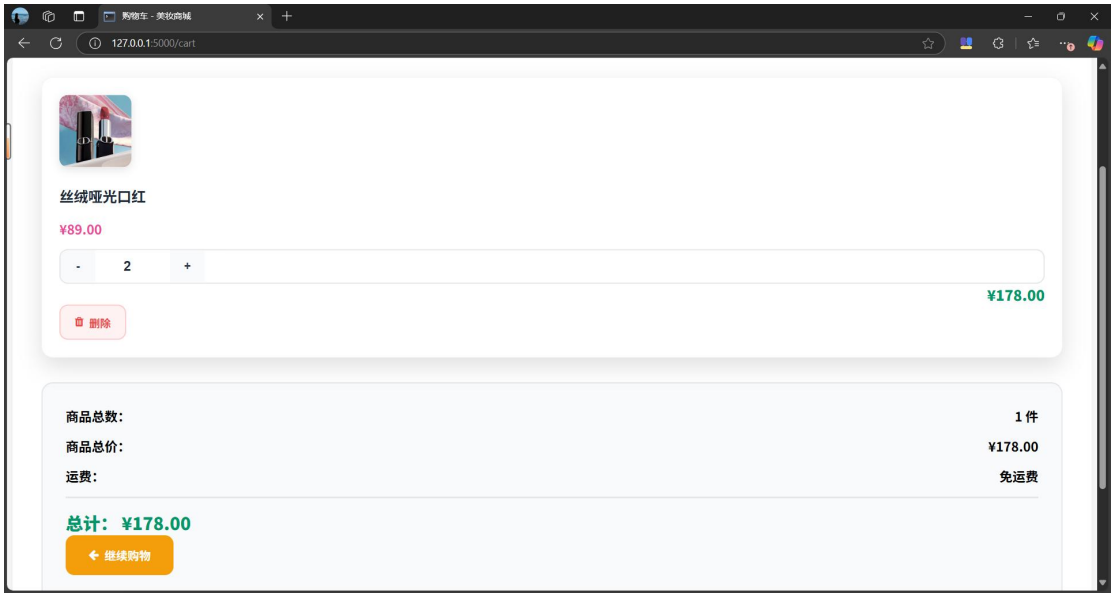
本系统前端采用 HTML5、CSS3 和 JavaScript 进行开发，结合 Jinja2 模板引擎实现页面的动态渲染。主要页面包括首页、商品详情页、用户登录/注册页、购物车页、结算页以及管理员后台页面。各页面布局简洁美观，注重用户体验和操作便捷性。系统采用响应式设计，能够适配不同尺寸的终端设备。前端与后端的数据交互主要通过表单提交和 AJAX 请求实现，用户的操作如登录、注册、添加购物车、下单等均能获得及时的反馈。部分页面还引入了交互动画和提示信息，提升了整体的用户体验。



5.3.2 购物车与订单流程实现

购物车和订单流程是系统前端实现的核心功能之一。用户在浏览商品时可以

一键将商品加入购物车，购物车页面支持商品数量的修改和删除操作。结算流程中，用户需填写收货信息并选择支付方式，前端会对输入内容进行基本校验，确保数据的有效性。订单提交后，系统会及时反馈订单状态，并在用户中心或后台展示订单详情。整个购物流程流畅，前端通过与后端 API 的高效交互，实现了数据的实时更新和页面的无刷新体验，极大提升了用户的购物满意度。



5.4 管理员后台实现

5.4.1 商品管理实现

管理员后台为系统运营和维护提供了强有力的支持。商品管理模块允许管理员对商品信息进行增删改查操作。管理员可以通过后台界面添加新商品，设置商品的名称、分类、价格、库存、图片等详细信息；也可以对已有商品进行编辑和删除。系统还支持商品的批量更新和库存管理，方便管理员及时调整商品信息，保证前台展示的商品数据准确无误。所有操作均通过与后端 API 的交互实现，确保数据的实时同步和一致性。

5.4.2 用户管理实现

用户管理模块主要用于管理员对平台用户的管理。管理员可以在后台查看所有注册用户的详细信息，包括用户名、注册时间等。对于存在违规行为的用户，管理员有权进行删除操作，但系统会保护管理员账号不被误删。通过权限控制机制，只有具有管理员权限的用户才能访问和操作该模块，保障了系统的安全性和

管理的规范性。

5.4.3 订单管理实现

订单管理模块为管理员提供了对所有订单的集中管理能力。管理员可以在后台查看所有用户的订单列表，查询订单的详细信息，包括订单状态、商品明细、收货信息等。系统支持订单状态的更新，如发货、完成、取消等操作，方便管理员对订单进行流程管理和售后处理。通过订单管理模块，管理员能够及时掌握平台的运营状况，提高服务效率和用户满意度。

5.5 系统安全实现

5.5.1 CSRF 防护

为防止跨站请求伪造（CSRF）攻击，本系统在表单和敏感操作中集成了 CSRF 防护机制。后端采用 Flask-WTF 扩展自动为每个表单生成唯一的 CSRF 令牌，前端在提交表单或发送 AJAX 请求时会自动携带该令牌。后端在接收到请求时会校验 CSRF 令牌的有效性，只有校验通过的请求才会被处理。通过这种方式，有效防止了恶意网站伪造用户请求，保障了用户数据和操作的安全性。

5.5.2 Session 与权限控制

系统采用 Session 机制实现用户身份认证和会话管理。用户登录成功后，系统会在 Session 中保存用户的唯一标识，后续请求通过 Session 判断用户的登录状态。对于需要权限控制的操作（如购物车、下单、后台管理等），后端会校验 Session 信息，未登录用户或权限不足的用户将被重定向或拒绝访问。管理员权限的判定通过 Session 中的用户名或角色字段实现，确保只有管理员可以访问和操作后台管理功能。通过 Session 与权限控制机制，系统有效防止了未授权访问和越权操作，提升了整体安全性。

5.6 关键代码分析

本系统在实现过程中，部分核心代码体现了良好的设计思路和编程规范。以用户登录功能为例，后端通过 Flask 路由接收用户提交的用户名和密码，与数据库中的用户信息进行比对验证。登录成功后，系统会在 Session 中保存用户信息，并重定向到首页。购物车功能的实现则通过 AJAX 技术实现无刷新操作，用户点击"加入购物车"按钮后，前端会发送 POST 请求到后端，后端验证用户登录状态

和商品库存后，将商品信息保存到购物车表中。订单处理功能涉及多个数据表的操作，系统采用事务机制确保数据的一致性，如订单创建、库存扣减、购物车清空等操作要么全部成功，要么全部回滚。这些关键代码的实现思路清晰，逻辑严谨，为系统的稳定运行提供了有力保障。同时，代码中充分考虑了异常处理和边界情况，提高了系统的健壮性和用户体验。

6 系统测试

6.1 测试环境搭建

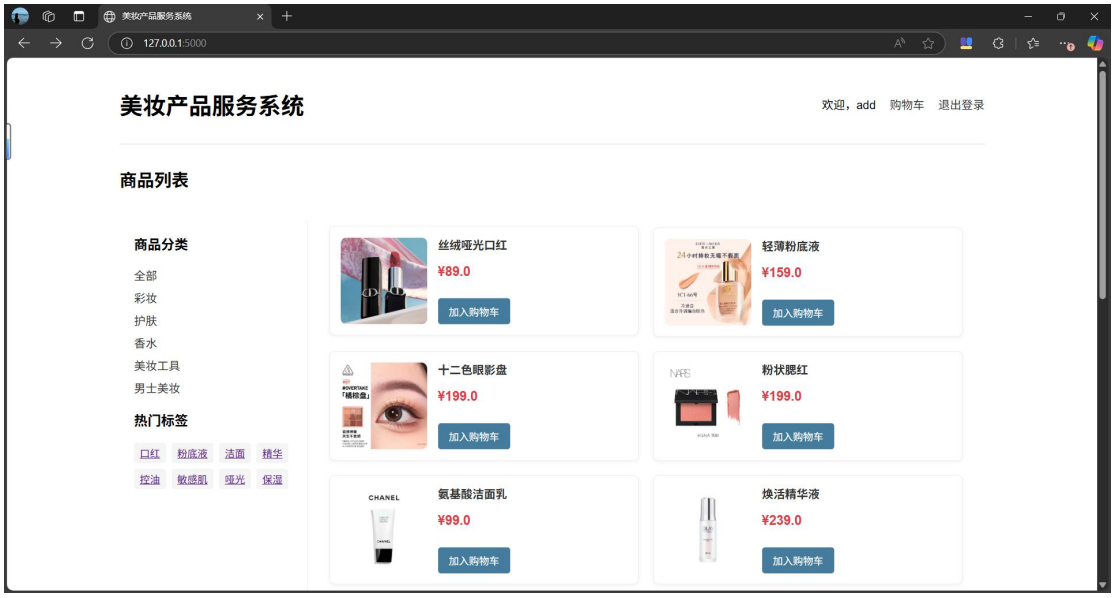
本系统的测试环境搭建在 Windows 10 操作系统下，使用 Python 3.7 及以上版本。后端服务通过 Flask 框架运行，数据库采用 SQLite。前端页面在 Chrome 浏览器中进行测试，接口测试工具选用 Postman。所有依赖通过 pip 进行安装，确保测试环境与开发环境一致。测试过程中，系统以开发模式运行，便于实时查看日志和调试信息。

6.2 功能测试

为保证系统各项功能的正确性和稳定性，分别对用户端和管理员端的主要功能进行了全面测试。

6.2.1 用户端功能测试

用户端测试内容包括用户注册、登录、商品浏览、商品详情查看、购物车管理、订单结算、评论提交等。测试过程中，验证了用户能够顺利完成注册和登录，商品信息能够正确展示，购物车的添加、修改、删除功能正常，订单流程顺畅，评论功能可用。所有功能均通过实际操作和数据校验，确保用户体验的完整性。



6.2.2 管理员端功能测试

管理员端测试主要包括商品管理、用户管理、订单管理等功能。测试内容涵盖商品的添加、编辑、删除，用户列表的查看与删除，订单的查询与状态更新等。测试结果表明，管理员能够顺利进行各项管理操作，后台数据与前台展示保持一致，权限控制有效，未授权用户无法访问后台管理页面。

6.3 性能测试

系统性能测试主要关注页面加载速度、接口响应时间和并发处理能力。通过多次刷新页面和模拟多用户同时操作，观察系统在高并发情况下的表现。测试结果显示，系统在常规使用场景下响应迅速，页面加载流畅，能够满足小型电商平台的性能需求。

6.4 安全性测试

安全性测试包括 CSRF 防护、Session 管理、权限控制等方面。通过构造非法请求和模拟未登录、非管理员用户访问敏感页面，验证系统能够有效阻止未授权操作。表单和接口均通过 CSRF 令牌校验，Session 信息安全可靠，权限判定严格，未发现明显的安全漏洞。

6.5 测试结果与分析

经过全面测试，系统各项功能均能正常运行，用户端和管理员端体验良好。性能和安全性均达到预期目标。测试过程中发现的个别小问题已及时修复，如表

单校验提示不够友好、部分页面兼容性细节等。整体来看，系统具备较高的稳定性和可靠性。

6.6 问题与优化建议

在测试过程中，发现系统在大数据量和高并发场景下可能存在性能瓶颈，建议后续可考虑优化数据库索引、增加缓存机制等措施。此外，前端页面的响应式设计和交互体验还有提升空间，可以引入更多现代前端框架。安全性方面建议进一步完善密码加密存储、日志审计等功能。通过持续优化，系统有望更好地满足实际应用需求。

7 部署与运维

7.1 部署环境说明

本系统支持在 Windows 和 Linux 操作系统下部署。后端依赖 Python 3.7 及以上版本，Flask 框架和相关依赖通过 pip 进行安装。数据库采用 SQLite，适合小型项目和开发测试环境。前端页面可通过任意现代浏览器访问。生产环境建议使用更高性能的 Web 服务器（如 Gunicorn、uWSGI）和反向代理（如 Nginx）进行部署，以提升系统的稳定性和并发处理能力。

7.2 部署流程（开发环境、生产环境）

在开发环境下，部署流程如下：

1. 安装 Python 及 pip 工具。
2. 克隆项目代码，进入项目目录。
3. 通过`pip install -r requirements.txt`安装所有依赖库。
4. 运行`python app.py`启动开发服务器。
5. 在浏览器访问`http://localhost:5000`即可使用系统。

在生产环境下，建议流程如下：

1. 配置好 Python 运行环境和虚拟环境。
2. 安装依赖并初始化数据库。
3. 使用 Gunicorn 或 uWSGI 等 WSGI 服务器运行 Flask 应用。
4. 配置 Nginx 等反向代理，将外部请求转发到 WSGI 服务器。
5. 配置防火墙和安全策略，确保系统安全运行。

7.3 运维与数据备份策略

为保障系统的稳定运行和数据安全，需定期对数据库文件进行备份。可以通过定时任务自动复制`instance/beauty.db`文件到安全位置。对于生产环境，建议采用自动化脚本或第三方备份工具实现增量备份和异地备份。系统日志应定期检查，及时发现和处理异常。服务器应定期更新安全补丁，防止安全漏洞。

7.4 系统上线后的维护建议

系统上线后，应持续关注用户反馈和系统运行状态，及时修复发现的问题。建议定期进行功能测试和安全扫描，确保系统稳定可靠。随着用户量和数据量的增长，可考虑优化数据库结构、引入缓存机制、升级服务器配置等措施以提升性能。对于前端页面和用户体验，也应根据实际需求不断优化和完善，确保系统能够长期稳定地服务于用户。

8 总结与展望

8.1 主要工作总结

本文围绕美妆电商系统的设计与实现，系统性地介绍了项目的开发全过程。首先，分析了当前电商平台的发展现状和用户需求，明确了系统的设计目标和功能定位。随后，详细阐述了系统采用的关键技术，包括 Flask 后端开发框架、前端页面设计、数据库建模与安全机制等。在系统实现部分，分别介绍了用户端、管理员端的主要功能模块及其实现方法。通过系统测试，验证了各项功能的正确性、系统的性能和安全性。最后，完成了系统的部署与运维方案设计。整体来看，本系统实现了美妆商品的在线展示、购物、下单、支付、评论以及后台管理等核心功能，具备良好的用户体验和可维护性，为后续的优化和扩展奠定了坚实基础。

8.2 项目创新点与不足

本系统在设计与实现过程中，结合了当前主流电商平台的功能特点，并针对美妆行业的实际需求进行了定制化开发。创新点主要体现在以下几个方面：

（1）系统采用模块化设计，前后端分工明确，便于后续功能扩展和维护；

（2）实现了商品的多维度展示与分类，提升了用户的购物体验；

（3）集成了购物车、订单、评论等完整的电商业务流程，功能较为完善；

（4）管理员后台支持商品、用户、订单的高效管理，提升了平台运营效率。

但由于开发时间和个人能力有限，系统仍存在一些不足：

（1）前端页面美观性和交互体验有待进一步提升；

（2）安全性措施如密码加密、日志审计等尚不完善；
（3）系统在大数据量和高并发场景下的性能优化不足，适合小型应用场景；
（4）部分功能如商品推荐、数据统计分析等尚未实现。

8.3 后续优化与发展方向

针对上述不足，后续可从以下几个方面进行优化和扩展：
（1）前端方面可引入现代化前端框架（如 Vue、React），提升页面的美观性和交互性；
（2）完善安全机制，实现用户密码加密存储、操作日志记录、异常监控等功能，提升系统安全性；
（3）优化数据库结构和查询效率，引入缓存机制，提高系统在大数据量和高并发下的性能表现；
（4）增加商品推荐、个性化营销、数据统计分析等智能化功能，提升平台的商业价值；
（5）支持多种支付方式和第三方登录，拓展系统的应用场景和用户群体。
通过持续优化和功能扩展，系统有望发展成为功能更完善、体验更优良的美妆电商平台。

参考文献

[1] 郭春平.响应式 Web 设计中 CSS Grid 与 Flexbox 布局的性能对比与优化实践[J].电子元器件与信息技术,2025,9(02):40-42+45.

[2] 于平.“HTML5+CSS3 WEB 前端设计”课程思政的融入现状[J].黑龙江科学,2024,15(13):147-149.

[3] 管玲玲,李浩然.基于 Flask 前后端分离 Web 开发的实现[J].电脑编程技巧与维护,2025,(06):168-170.

[4] 杨硕,史亚平.基于 Python+Flask 的在线考试系统设计与实现[J].电脑知识与技术,2025,21(02):47-49+56.

[5] 杨朝升.基于 Python 的 SQL 时间盲注攻击技术分析[J].石家庄职业技术学院学报,2024,36(06):33-38.

附录

主要代码	app.py

```
from flask import Flask, render_template, request, redirect, url_for, session, jsonify
from models import db, Product, Review, User, CartItem, Order, OrderItem  # 补充导入所有模型
from flask_wtf.csrf import CSRFProtect
import os

app = Flask(__name__)
app.secret_key = 'a_random_secret_key_for_session_123'  # 用于 session 加密
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///beauty.db'  # 数据库文件
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
# 新增 session 相关配置
app.config['SESSION_COOKIE_HTTPONLY'] = True
app.config['SESSION_COOKIE_SAMESITE'] = 'Lax'
app.config['SESSION_COOKIE_SECURE'] = False  # 本地开发用

CATEGORY_DATA = [
    {"id": 1, "name": "全部"},
    {"id": 2, "name": "彩妆"},
    {"id": 3, "name": "护肤"},
    {"id": 4, "name": "香水"},
    {"id": 5, "name": "美妆工具"},
    {"id": 6, "name": "男士美妆"},
]
HOT_TAGS = [
    "口红", "粉底液", "洁面", "精华",
    "控油", "敏感肌", "哑光", "保湿"
]

# 配置数据库
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///beauty.db'  # 使用 SQLite 数据库
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# 初始化数据库
db.init_app(app)
csrf = CSRFProtect(app)

# 配置 CSRF 豁免
@csrf.exempt
def csrf_exempt():
```

```
    pass

# 创建数据库表（首次运行时需要）
with app.app_context():
    db.create_all()
    # 检查是否有商品数据，如果没有则添加一些测试数据
    if not Product.query.first():
        products = [
            # 彩妆类
            Product(name="水光唇釉", price=129.0, image="/static/images/chunyou.jpg", description="水生透橘色，原生气血感", category="彩妆",
                    brand="Beauty Plus", origin="中国", shelf_life="3年", net_weight="3.5g",
                    ingredients="水、甘油、聚二甲基硅氧烷、辛酸/癸酸甘油三酯",
                    usage="取适量涂抹于双唇，打造水润光泽效果"),

            Product(name="丝绒哑光口红", price=89.0, image="/static/images/yaguang.jpg", description="持久不脱色，多种色号可选", category="彩妆",
                    brand="Beauty Plus", origin="中国", shelf_life="3年", net_weight="3.8g",
                    ingredients="蓖麻油、蜂蜡、羊毛脂、氧化铁",
                    usage="直接涂抹于双唇，打造哑光质感妆容"),

            Product(name="轻薄粉底液", price=159.0, image="/static/images/fendi.jpg", description="轻薄自然，遮瑕效果好", category="彩妆",
                    brand="Beauty Plus", origin="中国", shelf_life="2年", net_weight="30ml",
                    ingredients="水、甘油、二氧化钛、氧化锌",
                    usage="取适量点涂于面部，用海绵或手指均匀推开"),

            Product(name="十二色眼影盘", price=199.0, image="/static/images/yanying.jpg", description="12色眼影，易上色不飞粉", category="彩妆",
                    brand="Beauty Plus", origin="中国", shelf_life="3年", net_weight="12g",
                    ingredients="滑石粉、云母、氧化铁、二氧化钛",
                    usage="使用眼影刷蘸取适量眼影，涂抹于眼睑部位"),

            Product(name="粉状腮红", price=199.0, image="/static/images/saihong.jpg", description="杏粉色腮红，少女感满满",
```

```
category="彩妆",
                    brand="Beauty Plus", origin="中国", shelf_life="3年",
net_weight="4g",
                    ingredients="滑石粉、云母、氧化铁、二氧化钛",
                    usage="使用腮红刷蘸取适量，轻扫于颧骨位置"),

            # 护肤类
            Product(name="氨基酸洁面乳", price=99.0,
image="/static/images/ximiannai.jpg", description="温和清洁，适合敏感肌",
category="护肤",
                    brand="Beauty Plus", origin="中国", shelf_life="3年",
net_weight="100ml",
                    ingredients="水、椰油酰甘氨酸钠、甘油、透明质酸钠",
                    usage="取适量于掌心，加水揉搓起泡，轻柔按摩面部后冲
洗"),

            Product(name="焕活精华液", price=239.0,
image="/static/images/jinghua.jpg", description="修护肌肤，提亮肤色", category="
护肤",
                    brand="Beauty Plus", origin="中国", shelf_life="2年",
net_weight="30ml",
                    ingredients="水、烟酰胺、透明质酸钠、维生素 E",
                    usage="洁面后取 2-3 滴，轻拍至完全吸收"),

            Product(name="保湿补水面膜", price=39.0,
image="/static/images/mianmo.jpg", description="单片装，深层补水", category="护
肤",
                    brand="Beauty Plus", origin="中国", shelf_life="3年",
net_weight="25ml",
                    ingredients="水、甘油、透明质酸钠、芦荟提取物",
                    usage="洁面后敷于面部 15-20 分钟，取下后轻拍至吸收"),

            Product(name="控油爽肤水", price=89.0,
image="/static/images/shui.jpg", description="平衡油脂，收缩毛孔", category="护肤
",
                    brand="Beauty Plus", origin="中国", shelf_life="2年",
net_weight="150ml",
                    ingredients="水、金缕梅提取物、薄荷提取物、甘油",
                    usage="洁面后取适量于化妆棉，轻拍于面部"),

            Product(name="修护面霜", price=189.0,
image="/static/images/minxiuhu.jpg", description="夜间修护，适合干性肌肤",
category="护肤",
                    brand="Beauty Plus", origin="中国", shelf_life="2年",
```

```
net_weight="50g",
                    ingredients="水、甘油、角鲨烷、神经酰胺",
                    usage="晚间护肤最后一步，取适量均匀涂抹于面部"),

            # 香水类
            Product(name="清新花果香水", price=299.0, image="/static/images/dior.jpg", description="前调柑橘，中调玫瑰", category="香水",
                    brand="Beauty Plus", origin="法国", shelf_life="5年", net_weight="50ml",
                    ingredients="酒精、香精、水",
                    usage="喷洒于手腕、颈部等脉搏处，让香气自然散发"),

            Product(name="木质调男士香水", price=349.0, image="/static/images/weilan.jpg", description="檀香基调，持久留香", category="香水",
                    brand="Beauty Plus", origin="法国", shelf_life="5年", net_weight="100ml",
                    ingredients="酒精、香精、檀香精油",
                    usage="喷洒于手腕、颈部，适合商务场合使用"),

            Product(name="淡香水喷雾", price=199.0, image="/static/images/lv.jpg", description="清新淡雅，适合日常使用", category="香水",
                    brand="Beauty Plus", origin="法国", shelf_life="3年", net_weight="30ml",
                    ingredients="酒精、香精、水",
                    usage="轻喷于衣物或身体，适合日常使用"),

            # 美妆工具类
            Product(name="化妆刷套装", price=29.0, image="/static/images/shua.jpg", description="10支装，含腮红刷、眼影刷", category="美妆工具",
                    brand="Beauty Plus", origin="中国", shelf_life="长期", net_weight="套装",
                    ingredients="人造纤维毛、木柄",
                    usage="根据妆容需要选择合适的刷具，轻柔使用"),

            Product(name="粉扑套装", price=39.0, image="/static/images/fenpu.jpg", description="3个装，干湿两用", category="美妆工具",
                    brand="Beauty Plus", origin="中国", shelf_life="长期", net_weight="套装",
                    ingredients="海绵、聚氨酯",
```

```
                    usage="干用可定妆，湿用可上粉底，定期清洗保持卫生"),

            # 男士美妆类
            Product(name="男士素颜霜", price=159.0, image="/static/images/suyan.jpg", description="自然提亮，均匀肤色", category="男士美妆",
                    brand="Beauty Plus", origin="中国", shelf_life="2年", net_weight="30ml",
                    ingredients="水、二氧化钛、甘油、透明质酸钠",
                    usage="洁面后取适量均匀涂抹于面部，打造自然裸妆效果")
        ]
        db.session.add_all(products)
        db.session.commit()

# 路由定义
@app.route('/')
def index():
    # 获取筛选参数
    category_id = request.args.get('category')  # 分类 ID
    tag = request.args.get('tag')  # 标签

    # 基础查询
    query = Product.query

    # 分类筛选（CATEGORY_DATA 中"全部"的 id 是 1，不筛选）
    if category_id and category_id != '1':
        # 找到对应分类名称
        category_name = next((c['name'] for c in CATEGORY_DATA if str(c['id']) == category_id), None)
        if category_name:
            query = query.filter_by(category=category_name)

    # 标签筛选（模糊匹配商品名称或描述）
    if tag:
        query = query.filter(
            (Product.name.like(f'%{tag}%')) |
            (Product.description.like(f'%{tag}%'))
        )

    # 执行查询
    products = query.all()

    cart_count = 0
```

```
    if 'user_id' in session:
        cart_items = CartItem.query.filter_by(user_id=session['user_id']).all()
        cart_count = sum(item.quantity for item in cart_items)

    return render_template(
        'index.html',
        products=products,
        categories=CATEGORY_DATA,
        hot_tags=HOT_TAGS,
        current_category=category_id,
        current_tag=tag,
        cart_count=cart_count   # 添加购物车数量
    )

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        print(f"Debug: 登录尝试 - 用户名: {username}, 密码: {password}")

        user = User.query.filter_by(username=username).first()
        if user:
            print(f"Debug: 数据库中用户: {user.username}, 密码: {user.password}")
        else:
            print("Debug: 数据库中无此用户")

        if user and user.password == password:   # 明文比较
            session['user_id'] = user.id
            session['username'] = user.username
            print(f"Debug: 登录成功 - session: {dict(session)}")
            return redirect(url_for('index'))
        else:
            print(f"Debug: 登录失败 - 用户不存在或密码错误")
            return "用户名或密码错误，请重试"

    return render_template('login.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
```

```
            password = request.form['password']

            # 检查用户名是否已存在
            if User.query.filter_by(username=username).first():
                return "用户名已存在，请更换"

            # 创建新用户
            new_user = User(username=username, password=password)
            db.session.add(new_user)
            db.session.commit()

            return redirect(url_for('login'))

        return render_template('register.html')

@app.route('/logout')
def logout():
        session.pop('user_id', None)
        session.pop('username', None)
        return redirect(url_for('index'))

# 产品详情页路由
@app.route('/product/<int:product_id>')
def product_detail(product_id):
        # 获取产品详情
        product = Product.query.get_or_404(product_id)

        # 获取产品库存
        stock = product.stock   # 从数据库获取实际库存

        # 从数据库查询评论
        reviews = Review.query.filter_by(product_id=product_id).all()

        # 计算购物车商品总数
        cart_count = 0
        if 'user_id' in session:
            cart_items = CartItem.query.filter_by(user_id=session['user_id']).all()
            cart_count = sum(item.quantity for item in cart_items)

        # 找到产品对应的分类 ID
        category_id = next((c['id'] for c in CATEGORY_DATA if c['name'] ==
product.category), 1)

        return render_template(
```

```
            'product_detail.html',
            product=product,
            stock=stock,
            reviews=reviews,
            categories=CATEGORY_DATA,
            cart_count=cart_count,   # 传递购物车数量到模板
            category_id=category_id   # 传递分类 ID
        )

# 购买页面路由
@app.route('/checkout/<int:product_id>/<int:quantity>')
def checkout(product_id, quantity):
    if 'user_id' not in session:
        return redirect(url_for('login', next=url_for('checkout', product_id=product_id, quantity=quantity)))

    product = Product.query.get_or_404(product_id)
    total_price = product.price * quantity

    return render_template(
        'checkout.html',
        product=product,
        quantity=quantity,
        total_price=total_price
    )

# 处理支付路由
@app.route('/process_payment/<int:product_id>/<int:quantity>', methods=['POST'])
def process_payment(product_id, quantity):
    if 'user_id' not in session:
        return jsonify({'status': 'error', 'message': '请先登录'})

    try:
        data = request.get_json()
        if not data:
            return jsonify({'status': 'error', 'message': '请求数据无效'})

        # 验证收货信息
        required_fields = ['recipient', 'phone', 'address', 'payment_method']
        if not all(k in data for k in required_fields):
            return jsonify({'status': 'error', 'message': '请填写完整收货信息'})

        product = Product.query.get(product_id)
        if not product:
```

```
                return jsonify({'status': 'error', 'message': '商品不存在'})

            if product.stock < quantity:
                return jsonify({'status': 'error', 'message': '库存不足'})

            # 创建订单
            total_amount = product.price * quantity
            order = Order(
                user_id=session['user_id'],
                total_amount=total_amount,
                status='completed',
                recipient=data['recipient'],
                phone=data['phone'],
                address=data['address'],
                payment_method=data['payment_method']
            )
            db.session.add(order)
            db.session.flush()  # 获取订单 ID

            # 创建订单项
            order_item = OrderItem(
                order_id=order.id,
                product_id=product_id,
                quantity=quantity,
                price=product.price
            )
            db.session.add(order_item)

            # 更新库存
            product.stock -= quantity

            db.session.commit()

            return jsonify({
                'status': 'success',
                'message': '支付成功，订单已提交',
                'redirect_url': url_for('index')
            })

        except Exception as e:
            db.session.rollback()
            return jsonify({'status': 'error', 'message': f'支付处理失败: {str(e)}'})

@app.route('/cart')
```

```
def cart():
    if 'user_id' not in session:
        return redirect(url_for('login'))

    # 获取当前用户的购物车商品
    cart_items = CartItem.query.filter_by(user_id=session['user_id']).all()
    total_price = 0
    # 计算总价并保留商品信息
    for item in cart_items:
        product = Product.query.get(item.product_id)
        if product:
            item.product = product
            total_price += product.price * item.quantity

    return render_template('cart.html', cart_items=cart_items, total_price=total_price)

@app.route('/add_to_cart', methods=['POST'])
def add_to_cart():
    try:
        if 'user_id' not in session:
            return jsonify({'status': 'error', 'message': '请先登录'})

        # 获取数量参数，默认为 1
        data = request.get_json()
        if not data:
            return jsonify({'status': 'error', 'message': '请求数据无效'})

        quantity = data.get('quantity', 1)
        product_id = data.get('product_id')
        user_id = session['user_id']

        print(f"Debug: user_id={user_id}, product_id={product_id}, quantity={quantity}")

        if not product_id:
            return jsonify({'status': 'error', 'message': '商品 ID 无效'})

        # 检查商品是否存在
        product = Product.query.get(product_id)
        if not product:
            return jsonify({'status': 'error', 'message': '商品不存在'})

        # 检查库存
```

```
            if product.stock < quantity:
                return jsonify({'status': 'error', 'message': '库存不足'})

            # 检查购物车中是否已有该商品
            cart_item = CartItem.query.filter_by(user_id=user_id,
product_id=product_id).first()
            if cart_item:
                cart_item.quantity += quantity  # 已有则增加数量
            else:
                cart_item = CartItem(user_id=user_id, product_id=product_id,
quantity=quantity)
                db.session.add(cart_item)

            db.session.commit()

            # 计算购物车总数量
            cart_count = sum(item.quantity for item in
CartItem.query.filter_by(user_id=user_id).all())

            return jsonify({
                'status': 'success',
                'message': '已加入购物车',
                'cart_count': cart_count
            })
        except Exception as e:
            print(f"Error in add_to_cart: {str(e)}")
            return jsonify({'status': 'error', 'message': f'服务器错误: {str(e)}'})

# 更新购物车商品数量
@app.route('/update_cart/<int:item_id>', methods=['POST'])  # 改为 POST 方法
def update_cart(item_id):
    if 'user_id' not in session:
        return jsonify({'status': 'error', 'message': '请先登录'})

    data = request.get_json()
    change = data.get('change', 0)
    if change == 0:
        return jsonify({
            'status': 'error',
            'message': '无效的数量变更'
        })

    cart_item = CartItem.query.filter_by(
        id=item_id,
```

```
            user_id=session['user_id']
        ).first()

        if not cart_item:
            return jsonify({
                'status': 'error',
                'message': '购物车商品不存在'
            })

        # 检查库存
        new_quantity = cart_item.quantity + change
        if new_quantity < 1:
            return jsonify({
                'status': 'error',
                'message': '数量不能小于 1'
            })

        if new_quantity > cart_item.product.stock:
            return jsonify({
                'status': 'error',
                'message': '库存不足'
            })

        cart_item.quantity = new_quantity
        db.session.commit()

        return jsonify({
            'status': 'success',
            'message': '数量已更新'
        })

@app.route('/cart_checkout')
def cart_checkout():
        if 'user_id' not in session:
            return redirect(url_for('login'))

        cart_items = CartItem.query.filter_by(user_id=session['user_id']).all()
        if not cart_items:
            return redirect(url_for('cart'))

        total_price = 0
        # 获取商品详情并计算总价
        for item in cart_items:
            product = Product.query.get(item.product_id)
```

```
            if product:
                item.product = product
                total_price += product.price * item.quantity

    return render_template('cart_checkout.html', cart_items=cart_items,
total_price=total_price)

@app.route('/process_cart_payment', methods=['POST'])
def process_cart_payment():
    try:
        if 'user_id' not in session:
            return jsonify({'status': 'error', 'message': '请先登录'})

        data = request.get_json()
        print(f"Debug: Received payment data: {data}")

        # 验证收货信息
        if not data:
            return jsonify({'status': 'error', 'message': '请求数据无效'})

        required_fields = ['recipient', 'phone', 'address', 'payment_method']
        if not all(k in data for k in required_fields):
            missing_fields = [field for field in required_fields if field not in data]
            return jsonify({'status': 'error', 'message': f'请填写完整收货信息，缺
少: {", ".join(missing_fields)}'})

        # 获取用户购物车商品
        cart_items = CartItem.query.filter_by(user_id=session['user_id']).all()
        if not cart_items:
            return jsonify({'status': 'error', 'message': '购物车为空'})

        print(f"Debug: Found {len(cart_items)} items in cart")

        # 计算总金额
        total_amount = 0
        for item in cart_items:
            product = Product.query.get(item.product_id)
            if product:
                total_amount += product.price * item.quantity

        # 创建订单
        order = Order(
            user_id=session['user_id'],
            total_amount=total_amount,
```

```
                status='completed',
                recipient=data['recipient'],
                phone=data['phone'],
                address=data['address'],
                payment_method=data['payment_method']
            )
            db.session.add(order)
            db.session.flush()  # 获取订单 ID

            # 检查库存并创建订单项
            for item in cart_items:
                product = Product.query.get(item.product_id)
                if not product:
                    return jsonify({'status': 'error', 'message': f'商品不存在 (ID: {item.product_id})'})

                if product.stock < item.quantity:
                    return jsonify({'status': 'error', 'message': f'商品 {product.name} 库存不足 (当前: {product.stock}，需要: {item.quantity})'})

                # 创建订单项
                order_item = OrderItem(
                    order_id=order.id,
                    product_id=item.product_id,
                    quantity=item.quantity,
                    price=product.price
                )
                db.session.add(order_item)

                # 更新库存
                product.stock -= item.quantity

            # 清空购物车
            for item in cart_items:
                db.session.delete(item)

            db.session.commit()
            print("Debug: Payment processed successfully")

            return jsonify({
                'status': 'success',
                'message': '支付成功，订单已提交',
                'redirect_url': url_for('index')
            })
```

```
        except Exception as e:
            print(f"Error in process_cart_payment: {str(e)}")
            db.session.rollback()
            return jsonify({'status': 'error', 'message': f'支付处理失败: {str(e)}'})

# 检查用户登录状态
@app.route('/check_login')
def check_login():
        return jsonify({
            'logged_in': 'user_id' in session
        })

# 后台管理路由
@app.route('/admin/dashboard')
def admin_dashboard():
        if 'user_id' not in session or session.get('username') != 'admin':
            return redirect(url_for('login'))

        # 获取统计数据
        total_products = Product.query.count()
        total_users = User.query.count()
        total_orders = Order.query.count()
        total_revenue = db.session.query(db.func.sum(Order.total_amount)).scalar() or 0

        # 获取最近订单
        recent_orders = Order.query.order_by(Order.created_at.desc()).limit(5).all()

        # 获取热门商品（按销量排序）
        popular_products = db.session.query(
            Product,
            db.func.sum(OrderItem.quantity).label('sales_count')
        ).join(OrderItem).group_by(Product.id).order_by(
            db.func.sum(OrderItem.quantity).desc()
        ).limit(5).all()

        stats = {
            'total_products': total_products,
            'total_users': total_users,
            'total_orders': total_orders,
            'total_revenue': total_revenue
        }

        return render_template('admin_dashboard.html',
```

```
                                stats=stats,
                                recent_orders=recent_orders,
                                popular_products=popular_products)

@app.route('/admin/products')
def admin_products():
    if 'user_id' not in session or session.get('username') != 'admin':
        return redirect(url_for('login'))

    products = Product.query.all()
    return render_template('admin_products.html', products=products)

@app.route('/admin/add_product', methods=['POST'])
@csrf.exempt
def admin_add_product():
    print('当前 session:', dict(session))
    if 'user_id' not in session or session.get('username') != 'admin':
        return jsonify({'status': 'error', 'message': '权限不足'})

    try:
        data = request.get_json()
        print(f"Debug: 接收到的数据: {data}")

        if not data:
            return jsonify({'status': 'error', 'message': '请求数据无效'})

        # 验证必需字段
        required_fields = ['name', 'price', 'image', 'category', 'stock']
        for field in required_fields:
            if field not in data or not data[field]:
                return jsonify({'status': 'error', 'message': f'字段 {field} 不能为
空'})

        new_product = Product(
            name=data['name'],
            price=float(data['price']),
            image=data['image'],
            description=data.get('description', ''),
            category=data['category'],
            stock=int(data['stock']),
            brand=data.get('brand', ''),
            origin=data.get('origin', ''),
            shelf_life=data.get('shelf_life', ''),
            net_weight=data.get('net_weight', ''),
```

```
                ingredients=data.get('ingredients', ''),
                usage=data.get('usage', '')
            )
            db.session.add(new_product)
            db.session.commit()
            print(f"Debug: 商品添加成功，ID: {new_product.id}")
            return jsonify({'status': 'success', 'message': '商品添加成功'})
        except Exception as e:
            print(f"Debug: 添加商品时出错: {str(e)}")
            db.session.rollback()
            return jsonify({'status': 'error', 'message': f'添加失败: {str(e)}'})

@app.route('/admin/delete_product/<int:product_id>', methods=['POST'])
@csrf.exempt
def admin_delete_product(product_id):
    print('当前 session:', dict(session))
    if 'user_id' not in session or session.get('username') != 'admin':
        return jsonify({'status': 'error', 'message': '权限不足'})

    try:
        product = Product.query.get(product_id)
        if product:
            db.session.delete(product)
            db.session.commit()
            return jsonify({'status': 'success', 'message': '商品删除成功'})
        else:
            return jsonify({'status': 'error', 'message': '商品不存在'})
    except Exception as e:
        return jsonify({'status': 'error', 'message': f'删除失败: {str(e)}'})

@app.route('/admin/edit_product/<int:product_id>', methods=['POST'])
@csrf.exempt
def admin_edit_product(product_id):
    print('当前 session:', dict(session))
    if 'user_id' not in session or session.get('username') != 'admin':
        return jsonify({'status': 'error', 'message': '权限不足'})

    try:
        data = request.get_json()
        print(f"Debug: 编辑商品 {product_id}，接收到的数据: {data}")

        if not data:
            return jsonify({'status': 'error', 'message': '请求数据无效'})
```

```
            product = Product.query.get(product_id)
            if product:
                # 验证必需字段
                required_fields = ['name', 'price', 'image', 'category', 'stock']
                for field in required_fields:
                    if field not in data or not data[field]:
                        return jsonify({'status': 'error', 'message': f'字段 {field} 不
能为空'})

                product.name = data['name']
                product.price = float(data['price'])
                product.image = data['image']
                product.description = data.get('description', '')
                product.category = data['category']
                product.stock = int(data['stock'])
                # 更新其他字段
                product.brand = data.get('brand', '')
                product.origin = data.get('origin', '')
                product.shelf_life = data.get('shelf_life', '')
                product.net_weight = data.get('net_weight', '')
                product.ingredients = data.get('ingredients', '')
                product.usage = data.get('usage', '')

                db.session.commit()
                print(f"Debug: 商品 {product_id} 更新成功")
                return jsonify({'status': 'success', 'message': '商品更新成功'})
            else:
                return jsonify({'status': 'error', 'message': '商品不存在'})
        except Exception as e:
            print(f"Debug: 编辑商品时出错: {str(e)}")
            db.session.rollback()
            return jsonify({'status': 'error', 'message': f'更新失败: {str(e)}'})

@app.route('/admin/get_product/<int:product_id>')
def admin_get_product(product_id):
    if 'user_id' not in session or session.get('username') != 'admin':
        return jsonify({'status': 'error', 'message': '权限不足'})

    try:
        product = Product.query.get(product_id)
        if product:
            return jsonify({
                'status': 'success',
                'product': {
```

```
                            'id': product.id,
                            'name': product.name,
                            'price': product.price,
                            'image': product.image,
                            'description': product.description,
                            'category': product.category,
                            'stock': product.stock,
                            'brand': product.brand,
                            'origin': product.origin,
                            'shelf_life': product.shelf_life,
                            'net_weight': product.net_weight,
                            'ingredients': product.ingredients,
                            'usage': product.usage
                        }
                    })
            else:
                return jsonify({'status': 'error', 'message': '商品不存在'})
        except Exception as e:
            return jsonify({'status': 'error', 'message': f'获取失败: {str(e)}'})

@app.route('/admin/orders')
def admin_orders():
    if 'user_id' not in session or session.get('username') != 'admin':
        return redirect(url_for('login'))

    # 获取所有订单，按创建时间倒序排列
    orders = Order.query.order_by(Order.created_at.desc()).all()
    return render_template('admin_orders.html', orders=orders)

@app.route('/admin/users')
def admin_users():
    if 'user_id' not in session or session.get('username') != 'admin':
        return redirect(url_for('login'))

    users = User.query.all()
    return render_template('admin_users.html', users=users)

@app.route('/admin/categories')
def admin_categories():
    if 'user_id' not in session or session.get('username') != 'admin':
        return redirect(url_for('login'))

    # 暂时返回空页面
    return render_template('admin_categories.html', categories=[])
```

```
@app.route('/admin/delete_user/<int:user_id>', methods=['POST'])
def admin_delete_user(user_id):
    if 'user_id' not in session or session.get('username') != 'admin':
        return jsonify({'status': 'error', 'message': '权限不足'})

    try:
        user = User.query.get(user_id)
        if user and user.username != 'admin':
            db.session.delete(user)
            db.session.commit()
            return jsonify({'status': 'success', 'message': '用户删除成功'})
        else:
            return jsonify({'status': 'error', 'message': '用户不存在或不能删除管
理员'})
    except Exception as e:
        return jsonify({'status': 'error', 'message': f'删除失败: {str(e)}'})

@app.route('/admin/update_order_status/<int:order_id>', methods=['POST'])
def admin_update_order_status(order_id):
    if 'user_id' not in session or session.get('username') != 'admin':
        return jsonify({'status': 'error', 'message': '权限不足'})

    try:
        data = request.get_json()
        if not data or 'status' not in data:
            return jsonify({'status': 'error', 'message': '请求数据无效'})

        order = Order.query.get(order_id)
        if not order:
            return jsonify({'status': 'error', 'message': '订单不存在'})

        order.status = data['status']
        db.session.commit()

        return jsonify({'status': 'success', 'message': '订单状态更新成功'})
    except Exception as e:
        return jsonify({'status': 'error', 'message': f'更新失败: {str(e)}'})

# 确保在 app.py 中有以下路由
@app.route('/submit_review/<int:product_id>', methods=['POST'])
def submit_review(product_id):
    """处理商品评论提交"""
    try:
```

```
if 'user_id' not in session:
    return jsonify({
        'status': 'error',
        'message': '请先登录'
    }), 401  # 未授权状态码

# 获取 JSON 数据
data = request.get_json()
if not data or 'content' not in data:
    return jsonify({
        'status': 'error',
        'message': '评论内容不能为空'
    }), 400  # 错误请求状态码

content = data['content'].strip()
if not content:
    return jsonify({
        'status': 'error',
        'message': '评论内容不能为空'
    }), 400

# 检查商品是否存在
product = Product.query.get(product_id)
if not product:
    return jsonify({
        'status': 'error',
        'message': '商品不存在'
    }), 404  # 未找到状态码

# 创建新评论
new_review = Review(
    product_id=product_id,
    user_id=session['user_id'],
    content=content,
    rating=5  # 可以根据实际需求修改为用户评分
)

db.session.add(new_review)
db.session.commit()

return jsonify({
    'status': 'success',
    'message': '评论发表成功'
}), 200  # 成功状态码
```

```
    except Exception as e:
        # 捕获所有异常并记录
        db.session.rollback()
        print(f"评论提交错误: {str(e)}")
        return jsonify({
            'status': 'error',
            'message': '服务器错误，评论提交失败'
        }), 500  # 服务器错误状态码

# 从购物车移除商品
@app.route('/remove_from_cart/<int:item_id>', methods=['POST'])
def remove_from_cart(item_id):
    if 'user_id' not in session:
        return jsonify({
            'status': 'error',
            'message': '请先登录'
        })

    cart_item = CartItem.query.filter_by(
        id=item_id,
        user_id=session['user_id']
    ).first()

    if not cart_item:
        return jsonify({
            'status': 'error',
            'message': '购物车商品不存在'
        })

    db.session.delete(cart_item)
    db.session.commit()

    return jsonify({
        'status': 'success',
        'message': '商品已从购物车移除'
    })

@app.route('/test_browser')
def test_browser():
    return render_template('test_browser.html')

@csrf.exempt
@app.route('/admin/batch_update_products', methods=['POST'])
```

```
def batch_update_products():
    if 'user_id' not in session or session.get('username') != 'admin':
        return jsonify({'status': 'error', 'message': '权限不足'})
    try:
        data = request.get_json()
        updates = data.get('updates', [])
        for item in updates:
            product = Product.query.get(item['id'])
            if product:
                product.price = float(item['price'])
                product.stock = int(item['stock'])
        db.session.commit()
        return jsonify({'status': 'success', 'message': '批量保存成功'})
    except Exception as e:
        db.session.rollback()
        return jsonify({'status': 'error', 'message': f'保存失败: {str(e)}'})

if __name__ == '__main__':
    app.run(debug=True)
```