# 附件2 电子商务网站系统的概要设计说明书

**目录**

[附件2 电子商务网站系统的概要设计说明书 1](#_Toc138966440)

[1.导言 3](#_Toc138966441)

[1.1 目的 3](#_Toc138966442)

[1.2 范围 3](#_Toc138966443)

[1.3 编写说明 3](#_Toc138966444)

[1.4 术语定义 3](#_Toc138966445)

[1.5 引用标准 3](#_Toc138966446)

[1.6 参考资料 3](#_Toc138966447)

[1.7 版本更新信息 3](#_Toc138966448)

[2.系统分析 4](#_Toc138966449)

[3. 界面设计 5](#_Toc138966450)

[3.1 客户端界面设计 6](#_Toc138966451)

[4. 体系结构 11](#_Toc138966452)

[4.1 体系结构 11](#_Toc138966453)

[4.1.1 struct体系结构 11](#_Toc138966454)

[4.1.2 系统体系结构 12](#_Toc138966455)

[4.2 系统进行环境 13](#_Toc138966456)

[4.2.1 网络结构图 13](#_Toc138966457)

[4.2.2 硬件环境 14](#_Toc138966458)

[4.2.3 软件环境 14](#_Toc138966459)

[5. 数据模型 15](#_Toc138966460)

[5.1 数据库的概念结构模型设计 15](#_Toc138966461)

[5.2 数据库的逻辑结构模型设计 16](#_Toc138966462)

[5.3 数据库物理结构模型设计 19](#_Toc138966463)

[6. 模块设计 21](#_Toc138966464)

[6.1 客户端模块设计 21](#_Toc138966465)

[6.1.1表示层设计 21](#_Toc138966466)

[6.1.2 控制层 22](#_Toc138966467)

[6.1.3 模型层 24](#_Toc138966468)

[6.2 用户管理模块设计 24](#_Toc138966469)

[6.2.1 表示层设计 24](#_Toc138966470)

[6.2.2 控制层设计 25](#_Toc138966471)

[6.2.3 模型层设计 25](#_Toc138966472)

[6.3 订单管理模块设计 25](#_Toc138966473)

[6.3.1 表示层设计 25](#_Toc138966474)

[6.3.2 控制层设计 25](#_Toc138966475)

[6.3.3 模型层设计 26](#_Toc138966476)

[6.4 商品管理模块设计 26](#_Toc138966477)

[6.4.1 表示层设计 26](#_Toc138966478)

[6.4.2 控制层设计 26](#_Toc138966479)

[6.4.3 模型层设计 26](#_Toc138966480)

[6.5 物流管理模块设计 26](#_Toc138966481)

[6.5.1 表示层设计 26](#_Toc138966482)

[6.5.2 控制层设计 27](#_Toc138966483)

[6.5.3 模型层设计 27](#_Toc138966484)

[6.6 供应商管理模块设计 27](#_Toc138966485)

[6.6.1 表示层设计 27](#_Toc138966486)

[6.6.2 控制层设计 27](#_Toc138966487)

[6.6.3 模型层设计 28](#_Toc138966488)

# 1.导言

## 1.1 目的

该文档的目的是描述电子商务网站系统项目的概要设计，其主要内容包括：

·系统功能简介；

·系统结构设计；

·系统接口设计；

·数据设计；

·模块设计；

·界面设计。

本文档的预期的读者是：

·开发人员；

·项目管理人员；

·测试人员。

## 1.2 范围

该文档定义了系统的结构和单元接口，但未确定单元的实现方法，这部分内容将在详细设计/实现中确定。

## 1.3 编写说明

UML：Unified Modeling Language（统一建模语言）的缩写，是一个标准的建模语言。

## 1.4 术语定义

无

## 1.5 引用标准

无

## 1.6 参考资料

[1]《实战struct》（美）Ted Husted机械工业出版社，

[2]《软件重构》清华大学出版社，

## 1.7 版本更新信息

本文档的更新记录如表B-1所示。

**表B-1 版本更新记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修改编号 | 修改日期 | 修改后版本 | 修改位置 | 修改内容概述 |
| 000 | 2023.6.29 | 0.1 | 全部 | 初始发布版本 |

# 2.系统分析

管理端功能：

·注册/登陆/忘记密码：管理端用户可以注册新账户、登陆已有账户或重置密码。

·货物查询：管理端用户可以查询货物信息，包括库存数量、价格等。

·订单处理：管理端用户可以接受/拒绝订单，并通过手动操作完成订单处理。

·通知客户：管理端用户可以使用E-mail等方式向客户发送订单通知。

·发送订单给供应商：管理端用户可以将订单发送给供应商，以便供应商派送货物给客户。

·销售统计：管理端用户可以查看销售数据统计，包括订单数量、销售额等。

客户端功能：

·创建账户/登陆账户：客户可以通过浏览器创建新账户或登陆已有账户。

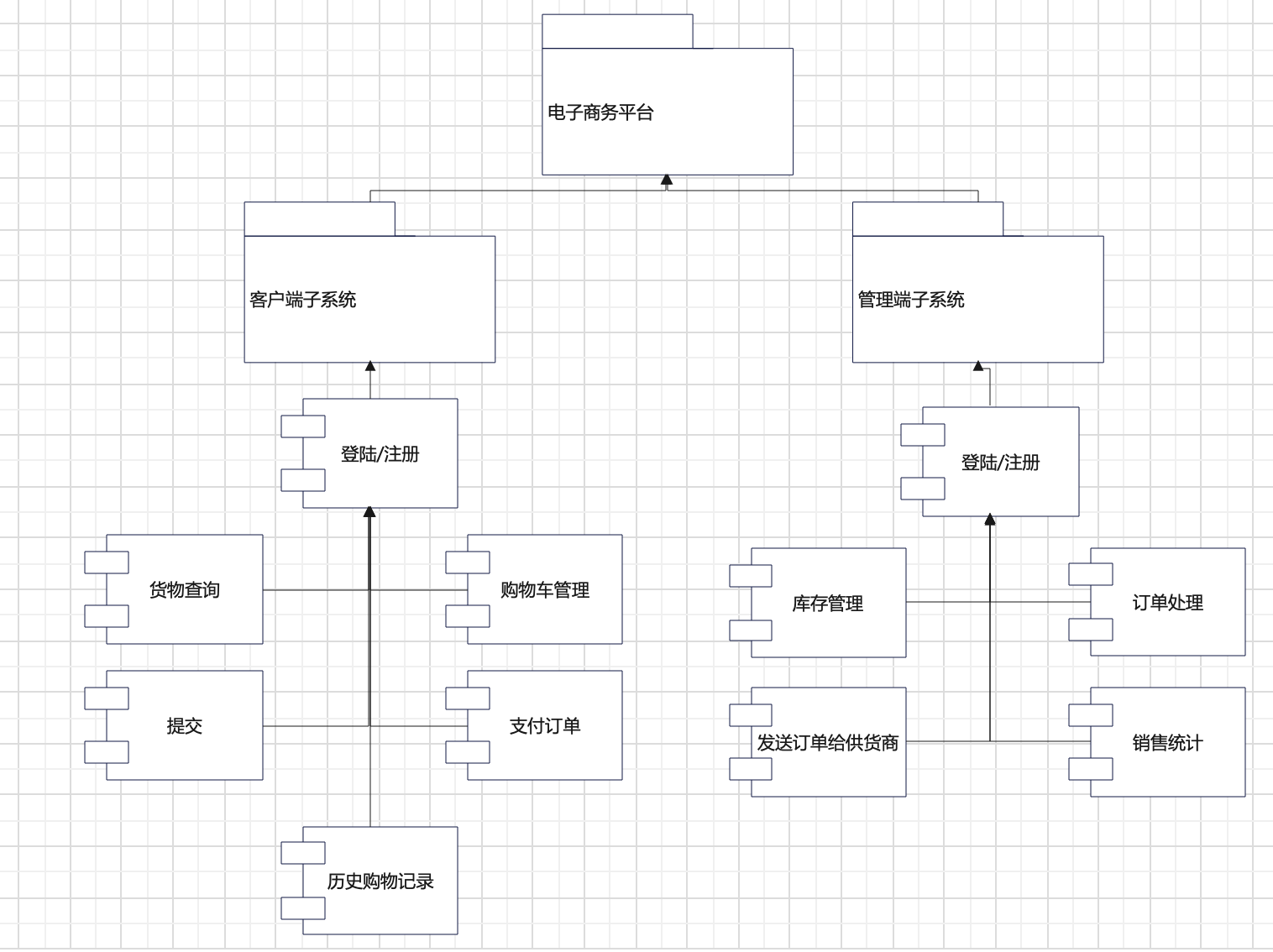
·货物查询：客户可以查询货物信息，包括库存数量、价格等。

·购物车管理：客户可以将货物放入购物车，并对购物车进行管理，如增加/删除货物。

·提交订单：客户可以创建并提交订单，包括选择货物、数量等信息。

·二维码支付：客户可以通过扫描二维码进行支付，完成订单交易。

·查询历史购物记录：客户可以查询以往的购物记录，包括订单详情、支付信息等。



**图B-1 客户端与管理端子系统图示**

# 3. 界面设计

本系统的用户界面按功能分为客户端界面和管理端界面。

用户购物流程和用户登录流程图如下所示。

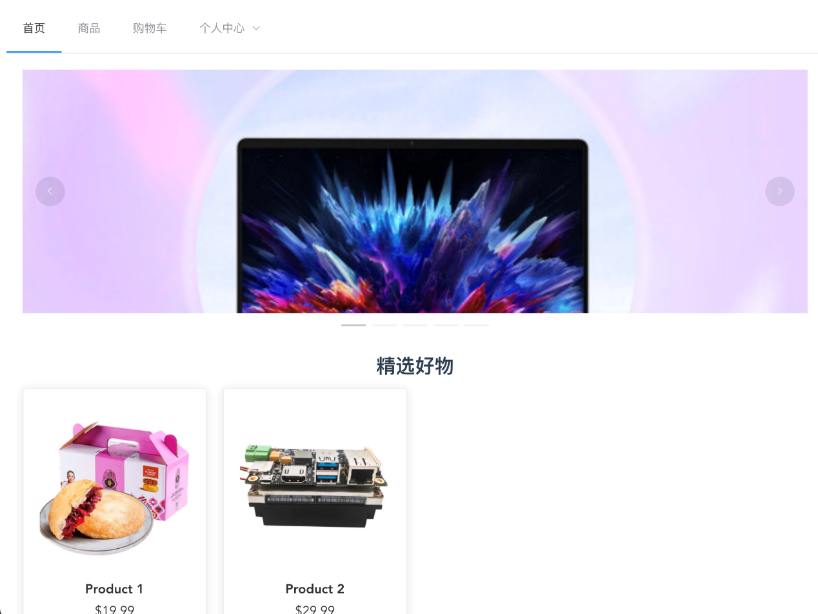
**图B-2 用户购物流程**

**图示

描述已自动生成**

**图B-3 用户登录流程**

## 3.1 客户端界面设计



**图B-4 电子商务平台客户端首页**



**图B-5 电子商务平台客户端商品页**



**图B-6 电子商务平台客户端购物车页**



**图B-7 电子商务平台客户端购物车结算页**

****

**图B-8 电子商务平台客户端订单支付页**



**图B-9 电子商务平台客户端支付失败页**



**图B-10 电子商务平台客户端支付成功页**



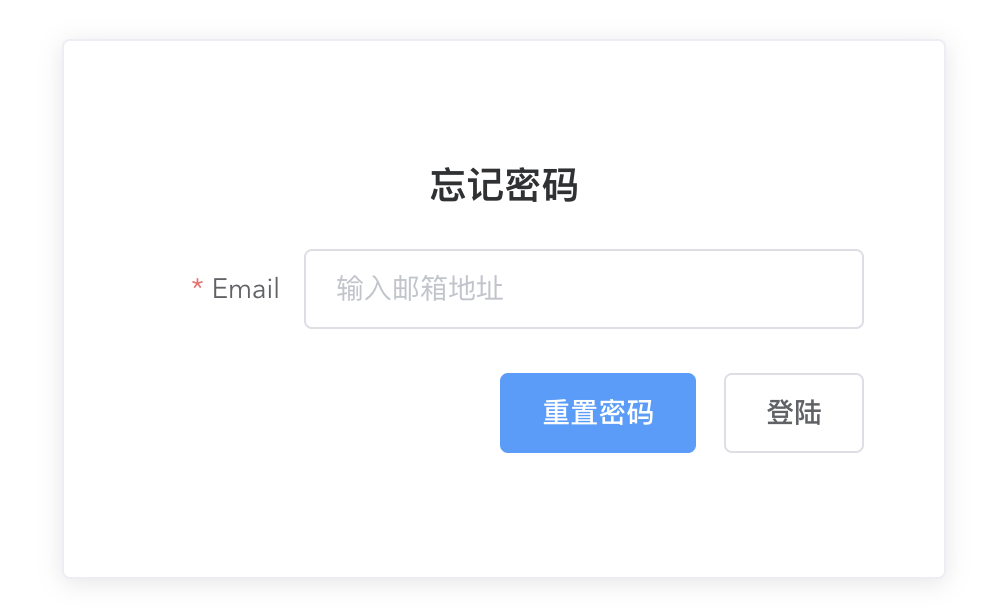
**图B-11 电子商务平台客户端个人中心页**



**图B-12 电子商务平台客户端订单管理页**



**图B-13 电子商务平台客户端注册页**



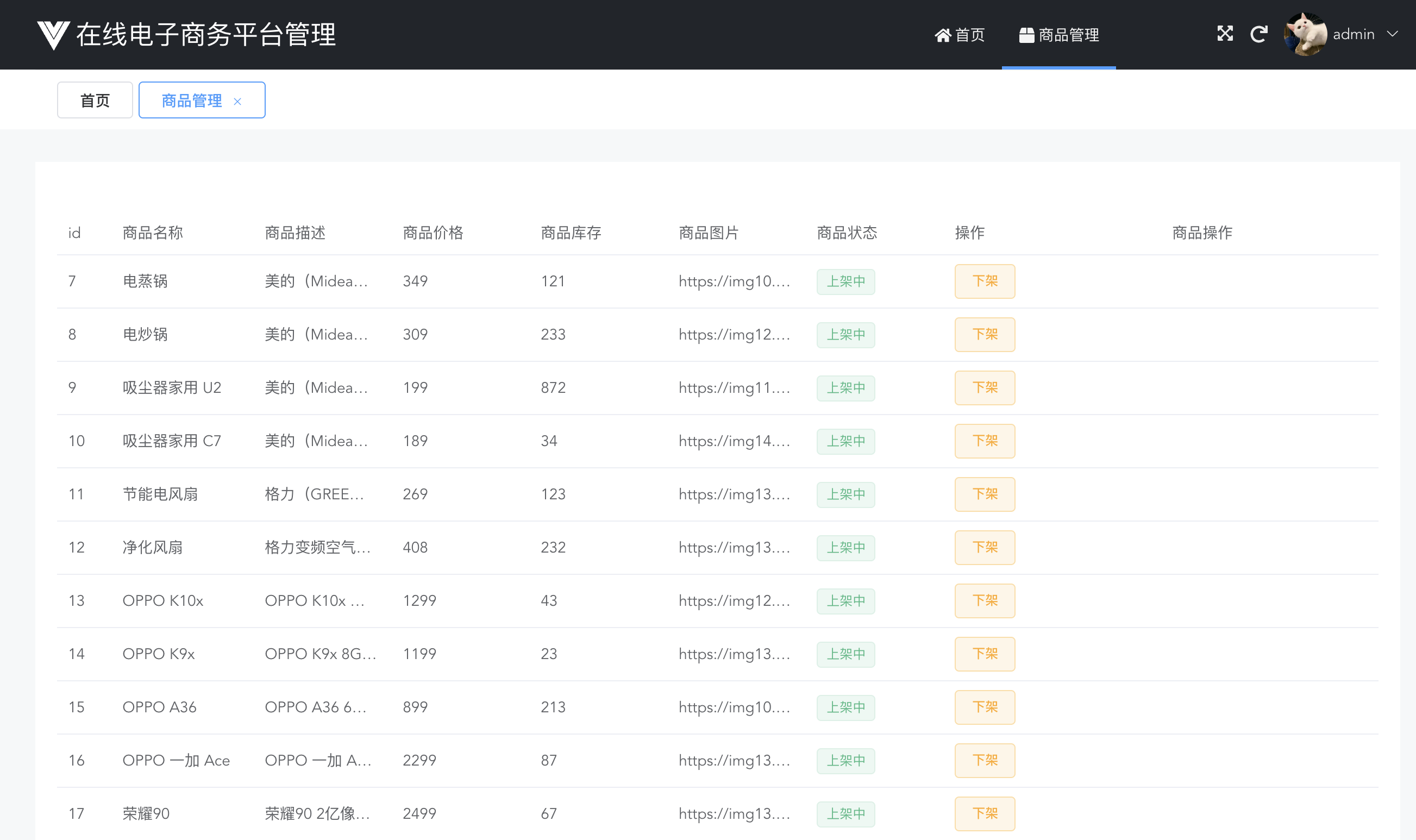
**图B-14 电子商务平台客户端忘记密码页**



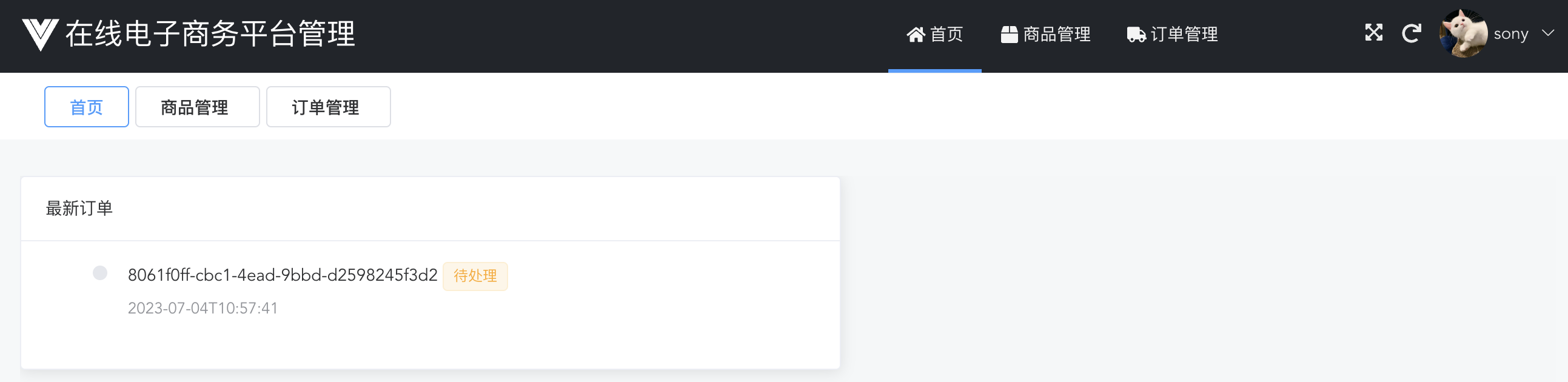
**图B-15 电子商务平台客户端登录页**



**图B-16 电子商务平台管理端登录页**



**图B-17 电子商务平台管理端管理员商品管理页**



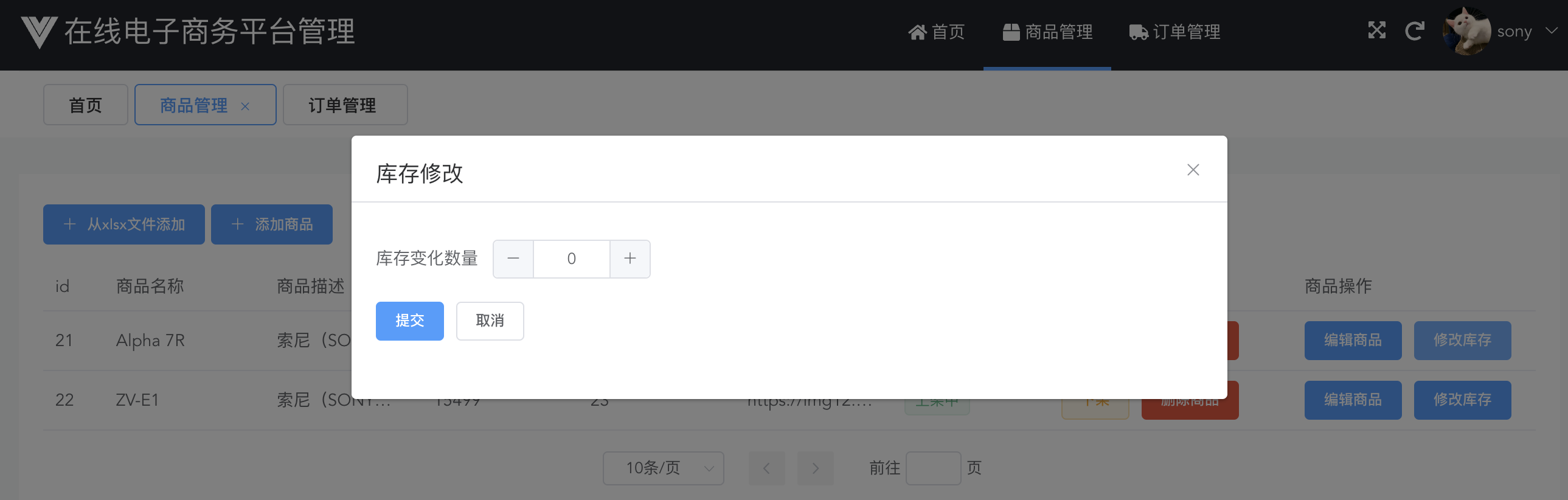
**图B-18 电子商务平台管理端供应商首页**



**图B-19 电子商务平台管理端供应商商品管理页**



**图B-20 电子商务平台管理端商品编辑页**



**图B-21 电子商务平台管理端库存管理页**



**图B-22 电子商务平台管理端订单管理页**

# 4. 体系结构

系统的总体结构设计遵循如下原则。

1）系统应具有良好的适应性：能适应用户对系统的软件环境、管理内容、模式和界面的要求；

2）系统应具有可靠性：采用成熟的技术方法和软件开发平台，以保证系统在以后的实际应用中安全、可靠；

3）系统应具有较好的安全性：应提高安全机制和用户权限限制机制的完善程度，确保数据的受限访问；

4）系统应具有良好的可维护性：系统应易于维护、安装；

5）系统应具有良好的可扩展性：系统应适应未来信息化建设的要求，能方便地进行功能扩展，以建立完善的信息集成管理体系。

## 4.1 体系结构

本系统使用了前后端分离的架构风格和MVC的软件设计模式。

### 4.1.1 前后端分离体系结构

前后端分离是一种软件架构模式，将应用程序的前端（客户端）和后端（服务器端）进行解耦，使它们可以独立开发、部署和扩展。这种架构模式的目标是提高系统的可维护性、可扩展性和灵活性。

在前后端分离结构中，前端主要负责处理用户界面和用户交互逻辑，通常使用 HTML、CSS 和 JavaScript 等技术来实现。前端可以通过 HTTP 请求与后端进行通信，获取数据或提交用户操作。

后端主要负责处理业务逻辑、数据存储和处理等任务。后端可以使用各种编程语言和框架来实现，例如 Java、Python、Node.js 等。后端通过 API（Application Programming Interface）提供数据和功能给前端，可以使用 RESTful API、GraphQL 等方式进行通信。

RESTful API： REST（Representational State Transfer）是一种软件架构风格，它通过使用HTTP协议定义了资源的标识、状态和操作。在前后端分离架构中，RESTful API被用作前后端之间的通信机制。

RESTful API设计遵循以下几个关键原则：

·资源标识：每个资源都有一个唯一的标识符（URI）。

·状态无关性：请求应该包含所有必要的信息，服务器不应依赖于会话状态来处理请求。

·统一的接口：使用统一的HTTP方法（如GET、POST、PUT、DELETE）执行特定操作。

·自描述消息：响应应该包含足够的信息，使客户端能够理解结果并进行下一步操作。

通过RESTful API，前端可以发送HTTP请求来获取或修改后端的数据。例如，前端可以使用GET请求获取数据，使用POST请求创建新数据，使用PUT请求更新现有数据，使用DELETE请求删除数据。后端根据收到的请求执行相应的操作，并返回JSON或其他格式的数据作为响应。

MVC模式： MVC（Model-View-Controller）是一种常见的软件设计模式，用于组织应用程序的代码结构和逻辑。尽管在传统的MVC模式中，View、Controller和Model之间存在紧密的耦合关系，但在前后端分离架构中，借鉴了MVC思想来更好地组织前端和后端代码。

·Model（模型）：Model层负责处理数据的存储、访问和处理。它可能涉及数据库操作、文件系统访问等。在前后端分离架构中，后端通常负责实现Model层，处理数据的增删改查等操作。

·View（视图）：View层代表用户界面，负责展示数据和与用户的交互。在前后端分离架构中，前端负责实现View层，使用HTML、CSS和JavaScript等技术构建用户界面。

·Controller（控制器）：Controller层负责处理请求和业务逻辑。它接收来自前端的请求并根据需要调用Model层进行数据操作，最后将结果返回给前端。在前后端分离架构中，后端负责实现Controller层。

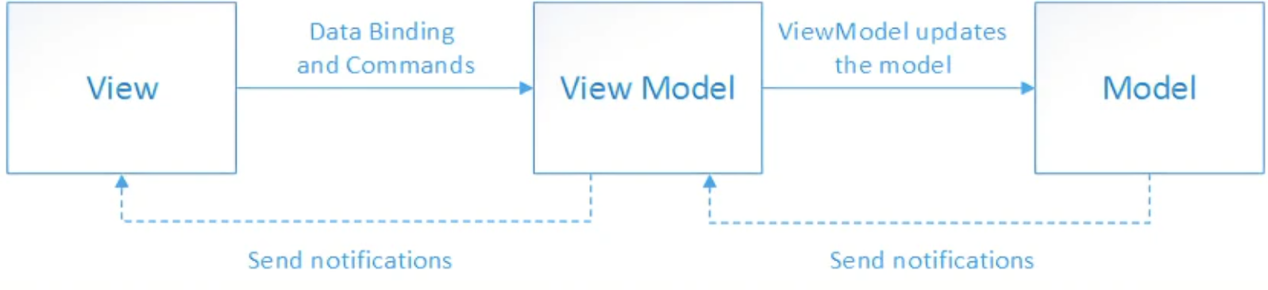
通过采用MVC模式，前后端可以更好地解耦，各自专注于自己的职责。前端负责展示界面和与用户的交互，后端负责处理数据和业务逻辑。这种分离使得开发更加灵活，不同团队成员可以并行开发并独立调整各自的部分，提高了可维护性和可扩展性。图形用户界面

中度可信度描述已自动生成

**图B-23 前后端分离软件框架**

### 4.1.2 系统体系结构

本系统的前端体系结构如图B-24所示。



**图B-24 前端的体系结构**

其中：

**·模型Model，**Model代表应用程序的数据和业务逻辑。它负责管理数据的获取、处理和存储，以及提供与数据相关的操作接口。在MVVM中，Model是独立于View和ViewModel的，它可以由多个ViewModel共享或多个View使用。；

**·视图View，**View是用户界面的可视化部分，负责展示数据给用户并接收用户的输入。它通常是由标记语言（如HTML、XAML等）和样式表（如CSS）来定义。View应该尽量减少自身的逻辑，而仅关注于呈现ViewModel提供的数据并响应用户交互；

**·视图模型，**ViewModel是连接View和Model之间的中间层。它从Model中获取数据，并对数据进行处理和准备，然后将处理后的数据提供给View进行显示。ViewModel还负责处理用户的输入和交互，并将用户的操作转换为对Model的请求。它不直接依赖于具体的View实现，而是暴露可供View绑定的属性和命令。

MVVM本质上就是MVC 的改进版，将其中的View 的状态和行为抽象化，使得视图 UI 和业务逻辑分开。

本系统的后端可以看作是MVC模型的M和C部分，

后端体系结构的具体拓扑图示如图B-25所示。

**图B-25体系结构拓扑图**

其中：

**·模型（Model）**

定义数据模型：使用Python的数据类（data class）或ORM（对象关系映射）框架（如SQLAlchemy）来定义数据模型。在模型中定义属性和方法，表示数据的结构和操作。

**· 控制器（Controller）**

定义路由函数（Route functions）：使用FastAPI的@app.route()装饰器定义路由函数，它们对应于不同的HTTP请求方法（如GET、POST等），处理相应的URL路径。处理请求和响应：在路由函数中，

根据需要获取参数、查询数据库、调用模型的方法等来处理请求，并准备好将数据作为响应返回给客户端。

### 4.1.3 Web服务器

本系统选用ASGI作为Web服务器，ASGI（Asynchronous Server Gateway Interface）是一种Python Web服务器和应用程序之间的异步通信协议。相比于传统的同步服务器架构，ASGI具有以下优点：

·异步处理：ASGI允许Web服务器以异步方式处理请求并与应用程序进行通信。这种异步处理能力可以提高服务器的并发性能，使得服务器能够同时处理多个请求而无需等待每个请求完成。

·高性能：由于ASGI支持异步处理，它可以更有效地利用计算资源，从而提供更高的性能和吞吐量。ASGI服务器可以处理大量并发请求，并且在高负载情况下能够保持响应速度。

·可扩展性：ASGI提供了一种可扩展的架构，可以通过将应用程序与多个服务器实例进行水平扩展来处理更多的请求负载。这使得应用程序能够适应不断增长的流量需求，而无需对整个应用程序进行修改。

·多协议支持：ASGI不仅支持HTTP协议，还支持其他协议如WebSocket。这意味着使用ASGI可以构建支持实时双向通信的应用程序，如聊天应用、实时数据传输等。

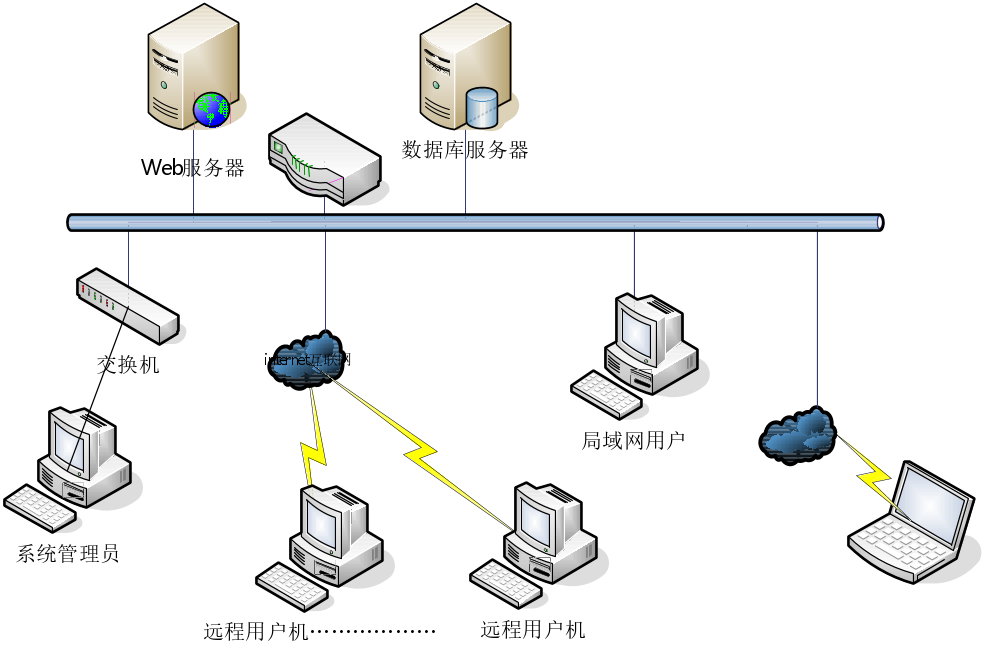
·生态系统支持：ASGI已成为Python生态系统的标准，许多现代化的Python Web框架和工具都提供了对ASGI的支持。这包括FastAPI、Django、Starlette等，使得开发者可以轻松地在这些框架中使用ASGI进行异步处理。

## 4.2 系统进行环境

下面讲述系统运行的网络结构，硬件、软件环境。

### 4.2.1 网络结构图

本系统的网络拓扑图如图B-8所示。



**图B-26 网络拓扑图**

其中的局域网用户机主要是公司内部的人员可以使用的机器，运程用户机主要是指通过互联网登录系统的人员使用的机器，可以是供应商，也可以是顾客。

### 4.2.2 硬件环境

本系统的硬件环境如下：

客户机：普通PC

· CPU：Intel系列/AMD系列

· 内存：8G以上

· 能够运行IE10.0/Chrome等的机器

· 分辨率：推荐使用1920×1080像素

Web服务器

· CPU： Intel XEON系列

· 内存：128G以上

· 硬盘：1TB以上

· 网卡：10Gb/s速度

数据库服务器

· CPU：Intel XEON等

· 内存：32GB以上

· 硬盘：80T以上

### 4.2.3 软件环境

本系统的软件环境如下：

· 操作系统：UNIX/Linux/Windows 10或以上版本

· 数据库：MySQL

· 开发工具包：Python 系列

·Web服务器：ASGI等

·浏览器：Chrome等

**1） 数据库及操作系统：**对于核心数据库来说，选择一个合适的数据库系统对我们的系统运行是很重要的，选择数据库的关键因素是要考虑预计会有多少人同时访问数据库；正常工作时间的级别；用来访问数据库的应用程序的类型；运行数据库的服务器的硬件和操作系统类型以及管理人员的专业技术水平。目前市场上适用于中小型企业的数据库产品有IBM DB2、Microsoft SQL Server系列，Oracle系列，MySQL。所有这些产品都基于SQL语言。同时，它们还拥有精度复杂的安全控制以适应不同的商业需要。

考虑到价格因素、易用性，我们使用MySQL 8.0作为系统后台数据库系统，服务器操作系统采用Ubuntu Server 22.04。

**2）Web服务软件：**目前的Web服务器软件有很多种，成熟而且稳定的有Apache、Tomcat和Microsoft的IIS，它们占据着Web服务器市场最大的份额，同时也有AGSI新兴Web服务器软件。本项目选择AGSI作为Web服务器主要是是基于以下考虑：

首先，AGSI具有高性能和可扩展性。作为一个新兴的Web服务器软件，AGSI专注于处理大量并发请求和高负载情况，确保系统能够快速响应用户请求并保持稳定性。

其次，AGSI支持多种编程语言和技术。作为一个灵活的Web服务器，AGSI提供了广泛的编程语言和技术支持。这使我们可以根据项目需求选择合适的技术栈，能够更好地满足业务需求。

此外，AGSI提供了丰富的功能和扩展性。它不仅支持常见的Web服务功能，还提供了一些额外的功能和扩展选项，并为系统添加定制化的功能。AGSI的功能丰富性使用户能够构建出更加灵活和强大的Web应用程序。

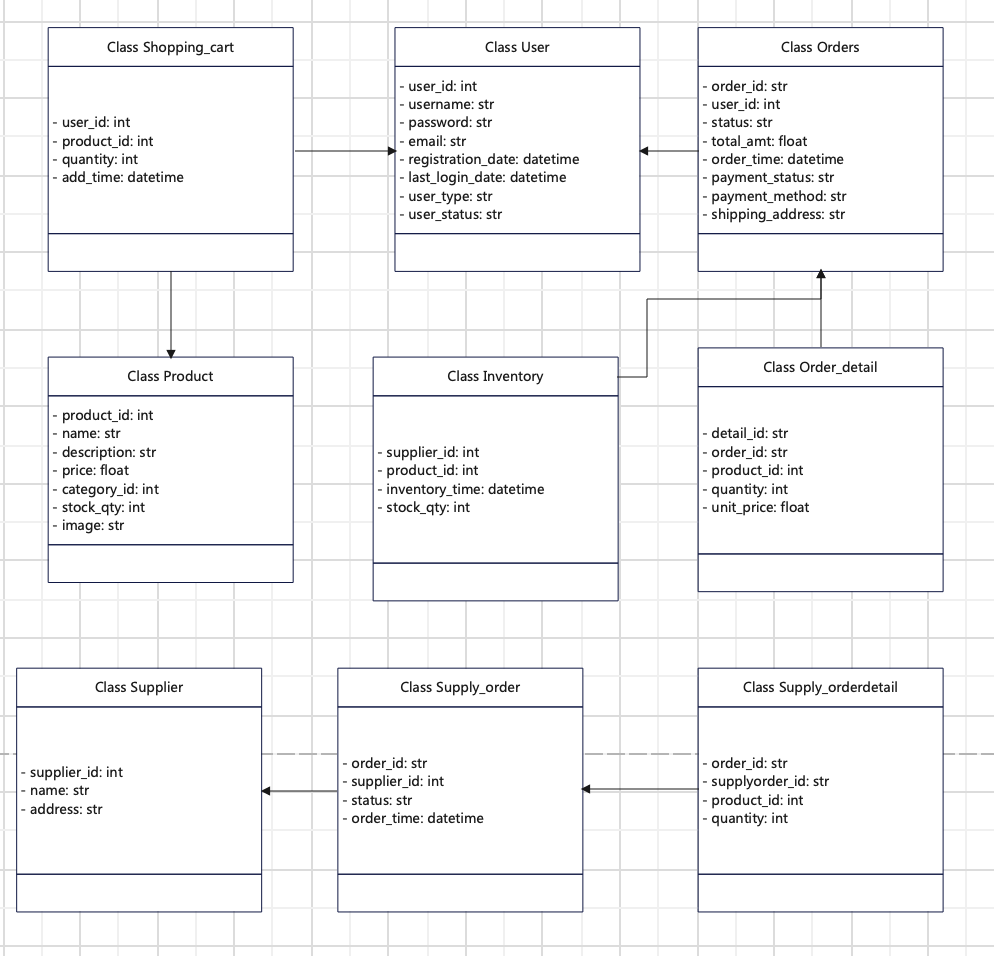
综上所述，基于AGSI的高性能和可扩展性、多语言支持、丰富的功能和扩展性，以及活跃的社区支持和详细的文档资源，本项目选择AGSI作为我们的Web服务器。

# 5. 数据模型

本系统的数据模型设计内容主要是进行数据库的设计。

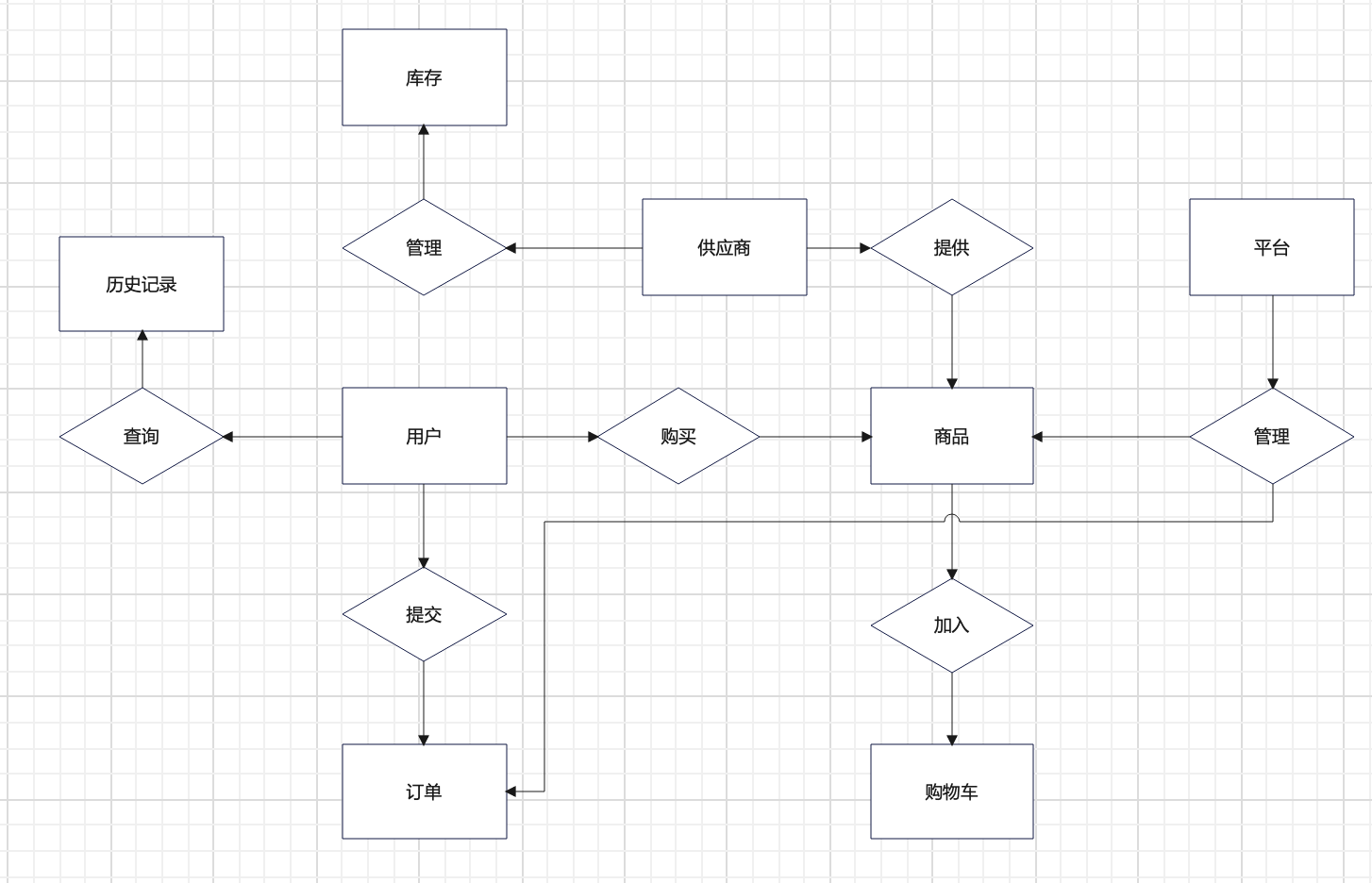
## 5.1 数据库的概念结构模型设计

概念设计用来反映现实世界中的实体、属性和它们之间的关系等的原始数据形式，建立数据库的每一幅用户视图。图B-28是系统E-R图。



**图B-27 数据库设计概念模型类图**

## 5.2 数据库的逻辑结构模型设计



**图B-28 数据库设计的逻辑模型E-R图**

数据库的逻辑设计是将各局部的E-R图进行分解、合并后重新组织起来形成数据库全局逻辑结构，包括所确定的关键字和属性、重新确定的记录结构、所建立的各个数据之间的相互关系。根据本系统需求分析，功能和对应库表如下。

用户管理：

User：存储用户信息，包括用户ID、用户名、密码、邮箱等。

商品管理：

Product：存储商品信息，包括商品ID、名称、描述、价格、库存等。

购物车管理：

Shopping\_cart：存储用户购物车信息，包括用户ID、商品ID、数量、添加时间等。

订单管理：

Orders：存储订单信息，包括订单ID、用户ID、订单状态、总金额、下单时间、支付状态、支付方式、收货地址等。

Order\_detail：存储订单详情信息，包括订单详情ID、订单ID、商品ID、数量、单价等。

支付管理：

Payment：存储支付信息，包括支付ID、订单ID、支付状态、支付金额、支付时间等。

账户管理：

Account：存储用户账户信息，包括账户ID、用户ID、余额、创建时间等。

供应商管理：

Supplier：存储供应商信息，包括供应商ID、名称、地址等。

Supplier\_product：存储供应商和产品之间的关系，包括供应商ID、产品ID。

库存管理：

Inventory：存储库存信息，包括供应商ID、产品ID、库存时间、库存数量等。

**表B-2 用户表(User)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段代码 | 字段类型 | 主(外)键 | 是否为空 | 备注(代码字典表) |
| 用户ID | user\_id | integer | Y | N |  |
| 用户名 | username | varchar | Y | N |  |
| 密码 | password | varchar | N | N | 6-16位 |
| 邮箱 | email | varchar | N | N |  |
| 注册时间 | registration\_date | datetime | N | N | 日期时间 |
| 最后登陆时间 | last\_login\_date | datetime | N | N | 日期时间 |
| 用户类型 | user\_type | varchar | N | N | admin/supplier/customer |
| 用户状态 | user\_status | varchar | N | N | active |

**表B-3 商品表(Product)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段代码 | 字段类型 | 主(外)键 | 是否为空 | 备注(代码字典表) |
| 商品ID | product\_id | integer | Y | N |  |
| 商品名称 | product\_name | varchar | Y | N | 最大长度36字符 |
| 商品描述 | product\_description | varchar | Y | N | 最大长度255字符 |
| 商品价格 | product\_price | float | N | N |  |
| 商品分类 | category\_id | integer | N | N |  |
| 商品库存数量 | stock\_quantity | integer | N | N |  |
| 商品图片 | product\_image | varchar | N | N | 图片URL |

**表B-4 购物车表(Shopping\_cart)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段代码 | 字段类型 | 主(外)键 | 是否为空 | 备注(代码字典表) |
| 用户ID | user\_id | integer | Y | N |  |
| 商品ID | product\_id | integer | Y | N |  |
| 商品数量 | quantity | integer | N | N |  |
| 添加时间 | add\_time | datetime | N | N | 日期时间 |

**表B-5 订单表(Order)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段代码 | 字段类型 | 主(外)键 | 是否为空 | 备注(代码字典表) |
| 订单ID | order\_id | uuid | Y | N | 128位uuid |
| 用户ID | user\_id | integer | Y | N |  |
| 订单状态 | order\_status | varchar | N | N | pending/shipped/accepted |
| 订单总金额 | total\_amount | float | N | N |  |
| 下单时间 | order\_time | datetime | N | N | 日期时间 |
| 支付状态 | payment\_status | varchar | N | N | Paid/unpaid |
| 支付方式 | payment\_method | varchar | N | N |  |
| 配送地址 | shipping\_address | varchar | N | N | 最大长度255个字符 |

**表B-6 订单详情表(Order\_detail)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段代码 | 字段类型 | 主(外)键 | 是否为空 | 备注(代码字典表) |
| 订单ID | order\_id | uuid | Y | N |  |
| 商品ID | product\_id | integer | Y | N |  |
| 商品数量 | quantity | integer | N | N |  |
| 商品单价 | unit\_price | float | N | N |  |

**表B-7 供应商表(Supplier)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段代码 | 字段类型 | 主(外)键 | 是否为空 | 备注(代码字典表) |
| 供应商ID | supplier\_id | integer | N | N |  |
| 供应商名称 | supplier\_name | varchar | N | N | 最大长度36个字符 |
| 供应商地址 | supplier\_address | varchar | N | N | 最大长度255个字符 |

**表B-8 供应商对应商品表(Supplier\_product)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段代码 | 字段类型 | 主(外)键 | 是否为空 | 备注(代码字典表) |
| 供应商ID | supplier\_id | integer | N | N |  |
| 商品ID | product\_id | integer | N | N |  |

**表B-9 供应商订单表(Supply\_order)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段代码 | 字段类型 | 主(外)键 | 是否为空 | 备注(代码字典表) |
| 供应商订单ID | supplyorder\_id | uuid | Y | N | 128位uuid |
| 供应商ID | supplier\_id | integer | N | N |  |
| 订单状态 | order\_status | varchar | N | N | Pending/Has paid/shipped |
| 订单生成时间 | order\_time | datetime | N | N | 日期时间 |

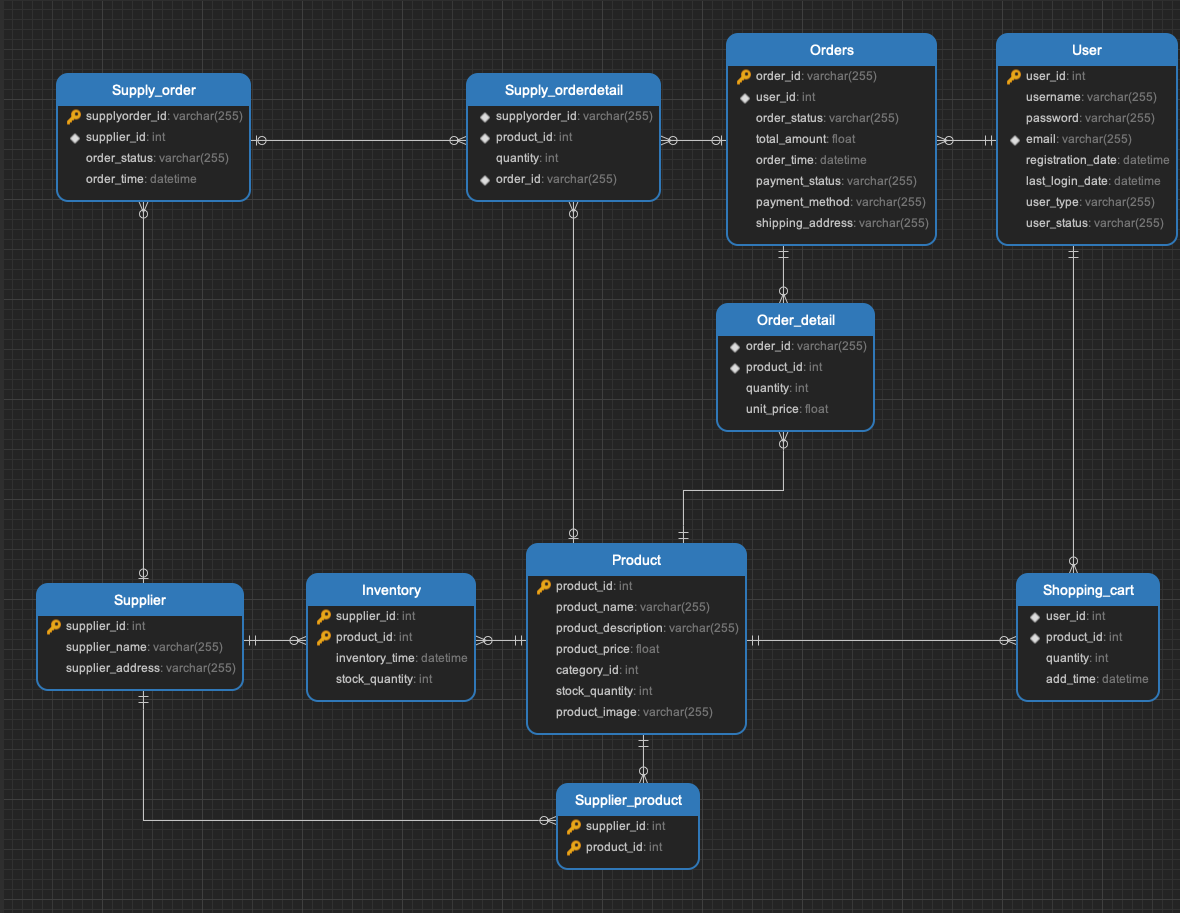
**表B-10 供应商订单详情表(Supply\_orderdetail)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段代码 | 字段类型 | 主(外)键 | 是否为空 | 备注(代码字典表) |
| 供应商订单ID | supplyorder\_id | uuid | N | N | 128位uuid |
| 商品ID | product\_id | integer | N | N |  |
| 商品数量 | quantity | integer | N | N |  |
| 用户订单ID | order\_id | uuid | N | N | 128位uuid |

**表B-11 库存表(Inventory)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段代码 | 字段类型 | 主(外)键 | 是否为空 | 备注(代码字典表) |
| 供应商ID | supplier\_id | integer | Y | N |  |
| 商品ID | product\_id | integer | Y | N |  |
| 入库时间 | inventory\_time | datetime | N | N | 日期时间 |
| 库存数量 | stock\_quantity | integer | N | N |  |

在确定了各个表主键字段的基础上，依据表与表相关字段之间的联系建立了各表之间的关系，如图B-29所示。



**图B-29 数据库设计的物理模型**

## 5.3 数据库物理结构模型设计

信息存储结构的设计在系统的设计中至关重要，要考虑到数据冗余、系统执行效率、信息控制以及维护等方面的要求。信息的管理离不开数据库的支持，我们采用MySQL数据库管理系统。

数据库的物理设计主要是对数据在内存中的安排，包括对索引区、缓冲区的设计，对使用的外存设备及外存空间的组织，包括索引区、数据块的组织与划分，设置访问数据的方式方法。需在非系统卷（操作系统所在卷以外的其他卷）上安装MySQL程序及数据库文件。内存是影响MySQL系统性能的一个重要因素，应在MySQL数据库安装后进行内存选项设置，根据系统总的物理内存来确定最适宜的内存需求，理想的情况是给MySQL分配尽可能多的内存，而不产生页面调度。同时，还需设置服务器的虚拟内存大小。

对MySQL的属性配置可以根据具体需求进行调整，包括缓冲区大小、索引设置、查询优化等。

通过pymysql与mysql建立连接

1）首先创建Connection对象。

DB\_CONFIG = {

'host': 'localhost',

'port': 3306,

'user': 'root',

'password': '123456',

'database': 'E-commerce\_platform',

'autocommit': True

}

并设置为自动提交

#建立连接

def get\_connection():

return pymysql.connect(\*\*DB\_CONFIG)

#关闭连接

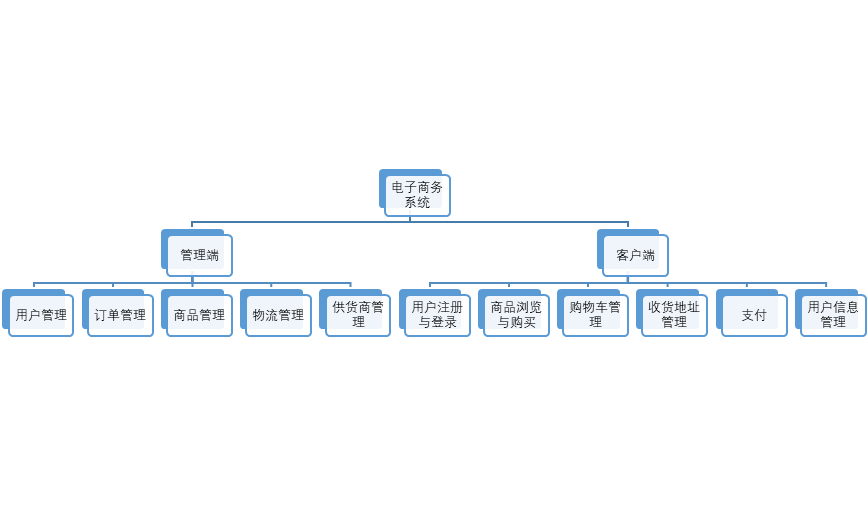
def close\_connection(connection):

connection.close()

定义上述函数来获取和释放数据库资源

# 6. 模块设计

按照功能分解，本系统分为客户端管理系统和管理端系统。根据页面流的设计，管理端系统又分用户管理、订单管理、商品管理、物流管理和供应商管理5个模块，如图B-30所示。

**图B-30 模块设计**

以下将分小节介绍各个部分的设计。

## 6.1 客户端模块设计

客户端模块是电子商务网站建设中的核心模块之一，它为客户提供了与网站进行交互的界面和功能。客户端模块的设计旨在实现客户的购物、下订单、订单管理和个人信息管理等功能。

首先，客户端模块包括用户注册与登录功能。用户可以通过注册账户来创建个人账户，然后使用登录功能进行身份验证，以便访问网站的各项功能。此外，还提供了忘记密码的功能，以帮助用户恢复账户访问权限。

其次，客户端模块实现了商品浏览与购买功能。客户可以通过浏览器浏览网站上的各类货物，查看详细信息、价格和库存情况。客户可以将感兴趣的货物放入购物车，并在需要时随时进行管理。购物车管理功能允许客户添加、删除或修改购物车中的货物。

第三，客户端模块提供了收货地址管理和支付功能。客户可以管理多个收货地址，选择适合的地址进行订单配送。二维码支付功能允许客户使用二维码或其他支付方式进行订单支付，确保安全、便捷的交易过程。

最后，客户端模块还包括用户信息管理功能，客户可以管理个人信息，如修改个人资料、密码和通讯方式等。

### 6.1.1表示层设计

表示层负责用户界面的展示和用户交互，根据上述功能介绍，总结出客户端的页面设计如表B-12所列。

**表B-12 客户端的页面设计**

|  |  |  |
| --- | --- | --- |
| 界面 | Vue | 功能描述 |
| 主页面 | index.vue | 客户端的主页面，展示电子商务网站的概览信息 |
| 商品列表 | productList.vue | 显示网站上所有的商品列表 |
| 商品详情 | productDetail.vue | 显示某个商品的详细信息 |
| 注册页面 | register.vue | 提供用户注册账户的页面 |
| 登录页面 | login.vue | 提供用户登录账户的页面 |
| 个人信息 | profile.vue | 显示和管理用户的个人信息 |
| 购物车 | shoppingCart.vue | 显示用户选择的商品和进行购物车管理 |
| 收货地址 | address.vue | 显示和管理用户的收货地址信息 |
| 订单提交 | order.vue | 提供用户提交订单的页面 |
| 支付页面 | payment.vue | 提供用户选择支付方式和进行支付操作的页面 |
| 购物记录 | purchaseHistory.vue | 显示用户的历史购物记录 |
| 管理端入口 | admin.vue | 提供管理员登录入口，进入管理端系统 |

在Vue中，组件和数据绑定用于处理HTML表单数据的传递。每个HTML表单可以使用Vue组件表示，并通过数据绑定将表单数据传递给控制组件。在Vue组件中，可以定义数据属性并使用Vue的表单验证功能对用户输入进行验证。通常，每个表单字段都与组件中的一个数据属性相对应。

**表B-13  客户端模块的ActionForm**

|  |  |  |
| --- | --- | --- |
| 界面 | Vue | 功能描述 |
| 主页面 | index.vue |  |
| 商品列表 | productList.vue | ProductListForm |
| 商品详情 | productDetail.vue | ProductDetailForm |
| 填写个人信息 | userInfo.vue | UserInfoForm |
| 购物车管理 | Cart.vue | CartForm |
| 收货地址管理 | address.vue | AddressForm |
| 支付页面 | payment.vue | PaymentForm |
| 历史购物记录 | orderHistory.vue | OrderHistoryForm |

### 6.1.2 控制层

在Vue中，控制层主要由Vue组件和相关业务逻辑函数组成，负责处理客户端系统的功能，包括用户注册与登录、商品浏览与购买、购物车管理、收货地址管理、支付和用户信息管理等。

在用户注册与登录功能中，控制层负责验证用户的输入信息，对密码进行加密处理，并将注册信息保存到数据库中。对于登录功能，控制层验证用户输入的凭证信息，并进行身份验证，允许合法用户访问系统。

在商品浏览与购买功能中，控制层接收用户对商品的查询请求，从数据库中获取相应的商品信息，并将结果传递给前端界面进行展示。当用户选择购买商品时，控制层将商品添加到购物车中，并更新购物车的内容。

购物车管理功能中，控制层负责展示购物车的内容，包括已选商品的列表和总价。用户可以增加、删除或修改购物车中的商品数量，控制层将相应的操作反馈给后端，并更新购物车信息。

收货地址管理功能中，控制层允许用户添加、修改或删除收货地址信息。控制层将用户的操作传递给后端进行相应的处理，并更新数据库中的地址信息。

支付功能中，控制层负责接收用户的支付请求，将相关支付信息传递给支付接口或支付服务提供商，并处理支付结果的回调通知。

用户信息管理功能中，控制层允许用户查看和修改个人信息，包括姓名、联系方式等。控制层验证用户的权限，并将用户的修改请求传递给后端进行处理。

在Vue中，控制层通过调用相关业务逻辑函数来处理用户的请求，将用户的操作转化为相应的业务操作，并与后端系统进行交互。控制层的设计需要考虑安全性、数据验证、异常处理等方面，以确保系统的稳定性和用户体验。

**表B-14 客户端应用的Action映射**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 事件 | API/函数 | 入口 | 数据模型 | 出口 |
| 进入商品列表 | EnterProductLit | index.vue | ProductListForm | productlist.jsp |
| 浏览商品详情 | ProductDetail | productlist.vue | ProductDetailForm | productdetail.jsp |
| 进入购物界面 | EnterShopping | productdetail.vue | UserInfoForm | userinfo.jsp |
| 添加购物车 | AddToCart | userinfo.vue | CartForm | cart.jsp |
| 管理购物车 | ManageCart | cart.vue | CartForm | cart.jsp |
| 提交订单 | SubmitOrder | cart.vue | OrderForm | order.jsp |
| 进行支付 | MakePayment | order.vue | PaymentForm | payment.jsp |
| 浏览历史订单 | ViewOrderHistory | index.vue | OrderHistoryForm | orderhistory.jsp |
| 进入个人中心 | EnterUserProfile | index.vue | UserProfileForm | userprofile.jsp |
| 编辑个人信息 | EditUserInfo | userprofile.vue | UserInfoForm | userinfo.jsp |
| 修改密码 | ChangePassword | userprofile.vue | ChangePasswordForm | userinfo.jsp |
| 查看订单详情 | ViewOrderDetail | orderhistory.vue | OrderDetailForm | orderdetail.jsp |
| 申请退款 | ApplyRefund | orderdetail.vue | RefundForm | refund.jsp |
| 处理用户消息 | ProcessUserMessage | index.vue | UserMessageForm | usermessage.jsp |
| 发送邮件通知 | SendEmailNotification | userinfo.vue | EmailNotificationForm | emailnotification.jsp |

### 6.1.3 模型层

在Vue中，电子商务系统的模型层仍然扮演着重要的角色，负责处理业务逻辑。可以使用Vue组件、API接口函数等形式来实现模型组件，用于处理与后端服务器的数据交互。

**表B-15 客户端的模型层**

|  |  |
| --- | --- |
| 模型组件 | 描述 |
| User | 用户模型，用于表示系统中的用户信息。 |
| Product | 商品模型，用于表示系统中的商品信息。 |
| ShoppingCart | 购物车模型，用于管理用户的购物车数据。 |
| Order | 订单模型，用于表示用户的订单信息。 |
| Payment | 二维码支付模型，用于处理用户通过二维码进行支付的相关逻辑。 |
| OrderHistory | 历史订单模型，用于管理用户的历史订单数据。 |
| OrderProcessing | 订单处理模型，用于处理订单的接受、拒绝和通知等相关逻辑。 |
| Supplier | 供应商模型，用于表示系统中的供应商信息。 |
| Inventory | 库存模型，用于管理系统中商品的库存信息。 |
| SalesStatistics | 销售统计模型，用于统计系统中的销售数据 |

## 6.2 用户管理模块设计

用户管理模块负责管理系统中的用户信息，包括注册新用户、登录已有账户、忘记密码和管理个人信息等功能。用户管理模块需要提供用户界面，允许管理员进行用户信息的查看、创建、更新和删除操作。

### 6.2.1 表示层设计

在表示层，可以使用Vue组件和前端技术实现用户管理界面，包括用户列表、用户详情和用户编辑等页面。根据上述的功能介绍，总结出用户管理的页面如表B-16所示。

**表B-16 用户管理模块的表示层设计**

|  |  |  |
| --- | --- | --- |
| 界面 | Vue组件 | 功能描述 |
| 登录界面 | login.vue | 用户登录的主页面 |
| 主页面 | main.vue | 用户管理主页面 |
| 忘记密码页面 | forgotPassword.vue | 用户忘记密码页面 |
| 注册页面 | register.vue | 用户注册的页面 |

### 6.2.2 控制层设计

控制层负责处理用户管理模块的请求和逻辑处理，接收前端请求，调用相应的API和业务逻辑函数进行用户数据的操作和验证，并返回结果给前端。

**表B-17 用户管理模块的控制层设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 事件 | API/函数 | 入口 | 数据模型 | 出口 |
| 用户登录 | Login | login.vue | LoginForm | main.vue |
| 用户登出 | Logout | main.vue | - | login.vue |
| 创建用户 | CreateUser | main.vue | CreateUserForm | main.vue |
| 编辑用户 | EditUser | main.vue | EditUserForm | main.vue |
| 删除用户 | DeleteUser | main.vue | DeleteUserForm | main.vue |

### 6.2.3 模型层设计

模型层包括用户数据的实体类、数据访问对象和业务逻辑处理。用户实体类用于表示用户信息，数据访问对象用于与数据库进行交互，业务逻辑处理包括用户验证、密码加密等操作。

**表B-18 用户管理模块的模型层设计**

|  |  |
| --- | --- |
| 模型组件 | 描述 |
| DBUtil | 数据库工具类，封装了基本的数据库操作方法 |
| Permission | 权限类，用于判断用户权限和访问控制 |
| UserClass | 用户实体类，用于表示用户的信息，并提供与数据库的交互方法 |

## 6.3 订单管理模块设计

订单管理模块负责接收、处理和管理用户的订单信息。管理员可以手动接受或拒绝订单，并使用电子邮件通知客户订单状态的更新。此外，订单管理模块还需将有效订单发送给供应商，并跟踪订单的处理进度。

### 6.3.1 表示层设计

订单管理界面需要提供订单列表、订单详情和订单处理等功能，以方便管理员查看订单信息、更新订单状态和与客户进行沟通。

**表B-19 订单管理模块表示层设计**

|  |  |  |
| --- | --- | --- |
| 界面 | Vue组件 | 功能描述 |
| 订单列表 | orderList.vue | 显示订单列表和相关操作 |
| 订单详情 | orderDetail.vue | 显示订单详细信息 |
| 创建订单 | createOrder.vue | 创建新订单 |

### 6.3.2 控制层设计

控制层负责处理订单管理模块的请求和业务逻辑，包括接收订单、更新订单状态、发送邮件通知等操作。

**表B-20 订单管理模块的控制层设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 事件 | API/函数 | 入口 | 数据模型 | 出口 |
| 获取订单列表 | getOrderList | orderList.vue | - | orderList.vue |
| 查看订单详情 | getOrderDetail | orderList.vue | - | orderDetail.vue |
| 创建订单 | CreateOrder | createOrder.vue | CreateOrderForm | orderDetail.vue |
| 编辑订单 | EditOrder | orderDetail.vue | EditOrderForm | orderDetail.vue |
| 删除订单 | DeleteOrder | orderDetail.vue | DeleteOrderForm | orderList.vue |

### 6.3.3 模型层设计

模型层包括订单实体类、数据访问对象和订单处理的业务逻辑。订单实体类用于表示订单信息，数据访问对象用于与数据库交互，业务逻辑包括订单验证、状态更新等处理。

**表B-21 订单管理模块的模型层设计**

|  |  |
| --- | --- |
| 模型组件 | 描述 |
| OrderUtil | 提供订单相关的数据库操作方法 |
| OrderClass | 订单模型类，用于表示订单信息，并提供与数据库的交互方法 |

## 6.4 商品管理模块设计

商品管理模块负责管理网站上的商品信息，包括添加、编辑、删除和展示商品等功能。管理员可以对商品进行分类、设定价格和库存，以便客户进行浏览和购买。

### 6.4.1 表示层设计

商品管理界面需要提供商品列表、商品详情和商品编辑等功能，以方便管理员进行商品信息的管理和操作。

**表B-22 商品管理模块的表示层设计**

|  |  |  |
| --- | --- | --- |
| 界面 | Vue组件 | 功能描述 |
| 商品列表 | productList.vue | 显示商品列表和相关操作 |
| 商品详情 | productDetail.vue | 显示商品详细信息 |
| 创建商品 | createProduct.vue | 创建新商品 |

### 6.4.2 控制层设计

控制层负责处理商品管理模块的请求和逻辑处理，包括添加商品、更新商品信息、删除商品等操作。

**表B-23 商品管理模块的控制层设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 事件 | API/函数 | 入口 | 数据模型 | 出口 |
| 获取商品列表 | getProductList | productList.vue | - | productList.vue |
| 查看商品详情 | getProductDetail | productList.vue | - | productDetail.vue |
| 创建商品 | CreateProduct | createProduct.vue | CreateProductForm | productDetail.vue |
| 编辑商品 | EditProduct | productDetail.vue | EditProductForm | productDetail.vue |
| 删除商品 | DeleteProduct | productDetail.vue | DeleteProductForm | productList.vue |

### 6.4.3 模型层设计

模型层包括商品实体类、数据访问对象和商品管理的业务逻辑。商品实体类用于表示商品信息，数据访问对象用于与数据库进行交互，业务逻辑包括商品验证、价格设定、库存管理等处理。

**表B-24 商品管理模块的模型层设计**

|  |  |
| --- | --- |
| 模型组件 | 描述 |
| ProductUtil | 提供商品相关的数据库操作方法 |
| ProductClass | 商品实体类，用于表示商品信息，并提供与数据库的交互方法 |

## 6.5 物流管理模块设计

物流管理模块负责处理订单的物流和配送流程。管理员可以跟踪订单的派送状态，与供应商协调货物的发货和送达，并提供基于Web的库存管理，以维护商品的库存数据库。

### 6.5.1 表示层设计

物流管理界面需要提供订单跟踪、物流信息管理和库存管理等功能，以方便管理员进行物流和库存的操作和管理。

**表B-25 物流管理模块的表示层设计**

|  |  |  |
| --- | --- | --- |
| 界面 | Vue组件 | 功能描述 |
| 物流列表 | logisticsList.vue | 显示物流列表和相关操作 |
| 物流详情 | logisticsDetail.vue | 显示物流详细信息 |
| 创建物流 | createLogistics.vue | 创建新物流 |

### 6.5.2 控制层设计

控制层负责处理物流管理模块的请求和逻辑处理，包括订单跟踪、物流信息更新和库存管理等操作。

**表B-26 物流管理模块的控制层设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 事件 | API/函数 | 入口 | 数据模型 | 出口 |
| 获取物流列表 | getLogisticsList | logisticsList.vue | - | logisticsList.vue |
| 查看物流详情 | getLogisticsDetail | logisticsList.vue | - | logisticsDetail.vue |
| 创建物流 | CreateLogistics | createLogistics.vue | CreateLogisticsForm | logisticsDetail.vue |
| 编辑物流 | EditLogistics | logisticsDetail.vue | EditLogisticsForm | logisticsDetail.vue |
| 删除物流 | DeleteLogistics | logisticsDetail.vue | DeleteLogisticsForm | logisticsList.vue |

### 6.5.3 模型层设计

模型层包括物流实体类、数据访问对象和物流管理的业务逻辑。物流实体类用于表示物流信息，数据访问对象用于与数据库进行交互，业务逻辑包括物流状态更新、库存管理等处理。

**表B-27 物流管理模块的模型层设计**

|  |  |
| --- | --- |
| 模型组件 | 描述 |
| LogisticsUtil | 提供物流相关的数据库操作方法 |
| LogisticsClass | 物流模型类，用于表示物流信息，并提供与数据库的交互方法 |

## 6.6 供应商管理模块设计

供应商管理模块负责管理系统中的供应商信息，包括接受订单、派送货物给用户和维护库存数据库等功能。管理员可以与供应商进行合作，确保供应链的顺畅。

### 6.6.1 表示层设计

供应商管理界面需要提供供应商列表、供应商详情和供应商合作管理等功能，以方便管理员进行供应商信息的管理和操作。

**表B-28 供应商管理模块的表示层设计**

|  |  |  |
| --- | --- | --- |
| 界面 | Vue组件 | 功能描述 |
| 供应商列表 | supplierList.vue | 显示供应商列表和相关操作 |
| 供应商详情 | supplierDetail.vue | 显示供应商详细信息 |
| 创建供应商 | createSupplier.vue | 创建新供应商 |

### 6.6.2 控制层设计

控制层负责处理供应商管理模块的请求和逻辑处理，包括供应商信息的增删改查、订单派送等操作。

**表B-29 供应商管理模块的控制层设计**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 事件 | API/函数 | 入口 | 数据模型 | 出口 |
| 获取供应商列表 | getSupplierList | supplierList.vue | - | supplierList.vue |
| 查看供应商详情 | getSupplierDetail | supplierList.vue | - | supplierDetail.vue |
| 创建供应商 | CreateSupplier | createSupplier.vue | CreateSupplierForm | supplierDetail.vue |
| 编辑供应商 | EditSupplier | supplierDetail.vue | EditSupplierForm | supplierDetail.vue |
| 删除供应商 | DeleteSupplier | supplierDetail.vue | DeleteSupplierForm | supplierList.vue |

### 6.6.3 模型层设计

模型层包括供应商实体类、数据访问对象和供应商管理的业务逻辑。供应商实体类用于表示供应商信息，数据访问对象用于与数据库进行交互，业务逻辑包括供应商信息的验证、订单派送等处理。

**表B-30 供应商管理模块的模型层设计**

|  |  |
| --- | --- |
| 模型组件 | 描述 |
| SupplierUtil | 提供供应商相关的数据库操作方法 |
| SupplierClass | 供应商模型，用于表示供应商信息，并提供与数据库的交互方法 |