

本科生毕业设计(论文)

中文题目 置闲物品交易系统的设计与实现

外文题目 Design and Implementation of

Idle Goods Trading System

|  |  |  |
| --- | --- | --- |
| 学 | 号 | 201626705010 |
| 姓 学  专 | 名 院  业 | 陈德斌  软件学院  软件工程 |

指导教师 徐虹

完成时间

江西师范大学教务处制

声 明

本人郑重声明： 所呈交的毕业设计（论文）是本人在指导教师指导下进行的研究

工作及取得的研究成果。其中除加以标注和致谢的地方，以及法律规 定允许的之外，不包含其他人已经发表或撰写完成并以某种方式公开 过的研究成果，也不包含为获得其他教育机构的学位或证书而作的材 料。其他同志对本研究所做的任何贡献均已在文中作了明确的说明并 表示谢意。

本毕业设计（论文）成果是本人在江西师范大学读书期间在指导 教师指导下取得的，成果归江西师范大学所有。

特此声明。

声明人（毕业设计（论文）作者）学号：201626705010

声明人（毕业设计（论文）作者）签名：陈德斌

签名日期： 年 月 日

**摘 要**

随着大学生经济消费能力的日益增强，一些东西被买回去后，由于相对用处不大而闲置在寝室，但对其他同学来说，这些物品也许有相当大的使用价值，甚至，一些还有使用价值的物品被同学们随意丢弃，不仅造成了资源浪费，还加剧了环境污染。我校大批毕业生离校之际有许多拿不走、带不回去的物品如用过的考研、计算机二级等的资料、水壶衣架、书桌、台灯等等生活用品，就这些物品如何处置，开发一个校园置闲物品交易系统，可以说是众望所归了。

本系统基于.net core Api、Entityframeworkcore、领域驱动设计框架、HTML、Css、JQuery、axios、VueJS、JsonWebToken等技术开发，网站整体功能分为用户管理模块、选购商品模块、商品管理模块和订单管理模块。用户管理模块提供注册、登录和个人信息修改功能；选购商品模块提供商品列表的展示、商品详情和将商品加入购物车功能；商品管理模块提供商品的发布、编辑修改和下架功能；订单管理模块提供生成订单，查看订单，查看订单详情和删除订单功能。

系统以C#语言为依托，使用Mssql为系统数据库，Kestrel作为服务器，利用JQuery和VueJs为脚本语言，采用CSS样式美化页面。系统采用前后端分离技术，采用axios技术来完成Ajax请求，客户端请求数据，服务端返回数据。

身份验证采用JsonWebToken技术。

论文首先介绍了物品置换管理系统的现状和意义，然后对网站进行了需求分析和可行性分析，在对网站进行详细分析后，依次阐述了各个模块的实现，通过测试分析，该系统运行稳定可靠，最后对系统进行了总结和展望。

#### 关键词：置闲物品、.net core、领域驱动设计、VueJs、前后端分离

**Abstract**

With the increasing economic consumption ability of college students, some things are bought back and left idle in the dorm because of their relative uselessness, but for other students, these items may have considerable use value, and even some The valuable items were discarded by students at will, which not only caused waste of resources, but also aggravated environmental pollution. When a large number of graduates from our school leave the school, there are many items that cannot be taken or brought back, such as used postgraduate examinations, computer secondary information, kettle hangers, desks, table lamps and other daily necessities. How to dispose of these items, It can be said that the development of a campus idle goods trading system is well received.

This system is based on .net core Api, Entityframeworkcore, domain-driven design framework, HTML, Css, JQuery, axios, VueJS, JsonWebToken and other technologies. The overall functions of the website are divided into user management module, merchandise purchase module, merchandise management module and order management Module. The user management module provides functions of registration, login and modification of personal information; the optional merchandise module provides the display of merchandise lists, merchandise details and the function of adding merchandise to the shopping cart; the merchandise management module provides the functions of publishing, editing, modifying and delisting merchandise; The module provides functions to generate orders, view orders, view order details and delete orders.

The system is based on the C # language, using Mssql as the system database, Kestrel as the server, using JQuery and VueJs as the scripting language, and using CSS styles to beautify the page. The system uses front-end and back-end separation technologies, and uses axios technology to complete Ajax requests, the client requests data, and the server returns data.Identity verification uses JsonWebToken technology.

The thesis first introduces the status and significance of the item replacement management system, and then conducts a demand analysis and feasibility analysis on the website. After a detailed analysis of the website, the implementation of each module is explained in turn. Through test analysis, the system runs stable and reliable Finally, the system is summarized and prospected.

#### Key Words: idle items、 .net core、domain-driven design、 VueJs、front-end separation

目录

**目 录**

目录

[第 1 章 绪论 1](#_Toc37529868)

[1.1 系统开发的背景 1](#_Toc37529869)

[1.2 系统开发的目标、意义 1](#_Toc37529870)

[1.3 本文的主要工作 1](#_Toc37529871)

[1.4 论文结构 2](#_Toc37529872)

[第 2 章 需求分析 3](#_Toc37529873)

[2.1 系统功能需求 3](#_Toc37529874)

[2.2 数据需求 15](#_Toc37529883)

[2.3系统非功能性需求 17](#_Toc37529884)

[2.4可行性分析 18](#_Toc37529889)

[第 3 章 系统设计 20](#_Toc37529894)

[3.1 功能模块设计 20](#_Toc37529895)

[3.2 数据库设计 35](#_Toc37529900)

[第 4 章 模块实现 38](#_Toc37529901)

[4.1 系统软件架构 38](#_Toc37529902)

[4.2 用户管理实现 38](#_Toc37529906)

[4.3 商品选购实现 39](#_Toc37529911)

[4.4 商品管理实现 39](#_Toc37529916)

[4.5 订单管理实现 39](#_Toc37529921)

[第 5 章 总结和展望 41](#_Toc37529926)

[5.1 总结 41](#_Toc37529927)

[5.2 展望 41](#_Toc37529928)

[参考文献 42](#_Toc37529929)

# 第 1 章 绪论

## 1.1 系统开发的背景

随着大学生经济消费能力的日益增强，一些东西被买回去后，由于相对用处不大而闲置在寝室，但对其他同学来说，这些物品也许有相当大的使用价值，甚至，一些还有使用价值的物品被同学们随意丢弃，不仅造成了资源浪费，还加剧了环境污染。目前，保护生态环境是我国面临的一项十分艰巨的任务，大学生是社会的中坚力量，树立正确的生态环境道德和环境保护意识是社会发展的必然要求，也是公民道德建设的重要内容。在环境保护的践行者之中，大学生是最有积极性和执行力的群体，有效地利用好这个群体，将有力推动环境保护的发展，倡导节约新风尚。我校大批毕业生离校之际有许多拿不走、带不回去的物品如用过的考研、计算机二级等的资料、水壶衣架、书桌、台灯等等生活用品，就这些物品如何处置，开发一个校园置闲物品交易平台，可以说是众望所归了。

## 1.2 系统开发的目标、意义

Internet技术迅速发展，互联网已日益成为收集提供信息的最佳渠道并且进入了传统的流通领域。高新技术的迅猛发展，信息技术的广泛应用，全球经济趋于一体化，使电子商务开始流行起来，一种全新的购物理念开始形成并逐步发展。人们的生活方式受其影响发生着巨大的变化。电子商务的出现满足了人们对交易迅速便捷的需要。一般人如果对自己的一些资源和用品有出售的愿望，但是以传统的方式进行出售显得非常的麻烦，因为没有合适的出售平台，而且面对的购买者范围相当狭窄。建立一个网上二手交易平台可以解决人们需要出售商品的愿望，在网络上以虚拟商店的形式发布自己的出售商品，可以利用网络这个巨大的资源找到广泛的顾客群。同时也给需要购买廉价商品的用户提供一个便捷有效的途径。与传统方式相比，开发网上二手商品交易系统具有许多优势：

1、其营业成本完全有理由比传统意义上的商店低得多，不需支付昂贵的店铺租金，花费大量的装修费用。

2、网上交易更容易开拓市场，互联网发展势头强烈凸现其资源的优越性。可实现24小时营业，不需要营业员，可以被看作为一个自动售货机。

3、网上交易系统可以实现实时的商品发布和按需购买，具有极强的方便性。

4、开发环境和资源相对完善，此类开发的技术已经逐渐趋于成熟，开发时较为便利。

## 1.3 本文的主要工作

该项目为本人个人项目，本人完成了用户管理模块、选购商品模块、商品管理模块和订单管理模块的设计与实现。

## 1.4 论文结构

以下论文主要分为四部分包括需求分析、系统设计、模块实现和对系统的总结和展望。第二章需求分析包括系统功能需求、数据需求系统、非功能性需求和可行性分析；第 三 章 系统设计包括功能模块设计和数据库设计；第 四章 模块实现包括系统软件架构和具体模块的实现；第 五 章 为总结和展望。

# 第 2 章 需求分析

## 2.1 系统功能需求

<<首先是概述，然后是具体介绍。概述部分可列出主要功能和主要用户类型。 具体介绍的功能则比较详细，但只包括后面测试部分涉及的功能。>>

### 2.1.1 角色分析

<<从系统的角度分析系统的参与者，并给出每一个参与者的描述。>>

从置闲物品交易系统的实际需求分析，系统涉及到以下角色如表 2-1 所示：

表 2.1 系统角色分析

|  |  |
| --- | --- |
| **角色** | **职责** |
| 游客 | 负责注册、登录和浏览商品 |
| 会员 | 负责选购商品、商品管理和订单管理 |
| 管理员 | 负责商品审核、用户管理和日志管理 |

### 2.1.2 总体用例分析

网上购物系统的参与者主要包括游客、会员和管理员三个，经过分析，系统主要包括注册、登录、商品浏览、商品选购、商品管理、订单管理、商品审核、用户管理和日志管理等9个用例，如下图 2.1 所示：



图 2.1 系统总体用例图

在本置闲物品交易系统的开发中，本人负责注册、登录、商品浏览、商品选购、商品管理、订单管理、商品审核、用户管理和日志管理等 9个功能（以下称为**置闲物品交易系统**）的分析、设计与实现。 下面对相关功能进行分析。

### 2.1.3 商品选购分析

商品选购用例中包括检索商品、查看商品详情、将商品放入购物车和将商品

移出购物车等 4 个子用例，其用例图如图 2.2 所示。



图 2.2 商品选购用例图

下面以检索商品子用例、查看商品详情子用例进行分析，它们的用例描述分 别如表 2.2、2.3 所示。

表2.2 检索商品用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 检索商品 |
| **用例描述** | 描述了会员使用本系统进行检索商品的过程 |
| **参与者表** | 会员 |
| **前置条件** | 无 |
| **后置条件** | 系统显示所检索的商品列表 |
| **基本操作流** | 1.会员提供关键字；  2.查看关键字相关商品列表。 |
| **可选操作流** | 无 |

表2.3 查看商品详情用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 查看商品详情 |
| **用例描述** | 描述了会员使用本系统查看商品详情的过程 |
| **参与者表** | 会员 |
| **前置条件** | 系统已提供商品列表 |
| **后置条件** | 系统显示所选中的商品详情信息 |
| **基本操作流** | 1.会员选中待查看的商品记录；  2.查看相关商品的详情信息。 |
| **可选操作流** | 无 |

### 2.1.4 商品管理分析

商品管理用例中包括发布商品、修改商品和下架商品等3个用例，其用例图如图2.4所示。



图 2.4 商品选购用例图

下面对其用例进行分析，它们的用例描述分别如表 2.5、2.6 和2.7所示

表2.5 发布商品用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 发布商品 |
| **用例描述** | 描述了会员使用本系统进行发布商品的过程 |
| **参与者表** | 会员 |
| **前置条件** | 会员已登录系统 |
| **后置条件** | 发布商品等待审核 |
| **基本操作流** | 会员提供商品相关信息 |
| **可选操作流** | 无 |

表2.6 修改商品用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 修改商品 |
| **用例描述** | 描述了会员使用本系统进行修改商品的过程 |
| **参与者表** | 会员 |
| **前置条件** | 会员已登录系统 |
| **后置条件** | 商品等待审核 |
| **基本操作流** | 会员提供商品相关修改信息 |
| **可选操作流** | 无 |

表2.7 删除商品用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 删除商品 |
| **用例描述** | 描述了会员使用本系统进行删除商品的过程 |
| **参与者表** | 会员 |
| **前置条件** | 会员已登录系统 |
| **后置条件** |  |
| **基本操作流** | 会员删除商品 |
| **可选操作流** | 无 |

### 2.1.5 订单管理分析

订单管理用例中包括提交订单、查看订单、查看订单明细和删除订单等4个用例，其用例图如图2.8所示



图 2.8 订单管理用例图

下面对其用例进行分析，它们的用例描述分别如表 2.9、2.10、2.11和2.12 所示

表2.9 提交订单用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 提交订单 |
| **用例描述** | 描述了会员使用本系统进行提交订单的过程 |
| **参与者表** | 会员 |
| **前置条件** | 会员已登录系统 |
| **后置条件** |  |
| **基本操作流** | 1.会员在商品界面点击购买；  2.会员在购物车点击购买 |
| **可选操作流** | 无 |

表2.10 查看订单用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 查看订单 |
| **用例描述** | 描述了会员使用本系统进行查看订单的过程 |
| **参与者表** | 会员 |
| **前置条件** | 1. 会员已登录系统 2. 会员已有提交的订单 |
| **后置条件** |  |
| **基本操作流** | 查看相关订单。 |
| **可选操作流** | 查看相关订单明细 |

表2.11 查看订单明细用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 查看订单明细 |
| **用例描述** | 描述了会员使用本系统进行查看订单明细的过程 |
| **参与者表** | 会员 |
| **前置条件** | 1.会员已登录系统  2.会员已有提交的订单 |
| **后置条件** |  |
| **基本操作流** | 查看订单相关明细 |
| **可选操作流** | 无 |

表2.12 删除订单用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 删除订单 |
| **用例描述** | 描述了会员使用本系统进行删除订单的过程 |
| **参与者表** | 会员 |
| **前置条件** | 1. 会员已登录系统 2. 会员已有提交订单 |
| **后置条件** |  |
| **基本操作流** | 删除选中订单 |
| **可选操作流** | 无 |

### 2.1.6 商品审核分析

商品审核用例中包括查看待审核商品、审核商品和上架商品等3个用例，其用例图如图2.13所示



图 2.13 商品审核用例图

下面对其用例进行分析，它们的用例描述分别如表 2.14、2.15和2.16 所示

表2.14 查看待审核商品用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 查看待审核商品 |
| **用例描述** | 描述了管理员使用本系统进行查看待审核商品的过程 |
| **参与者表** | 管理员 |
| **前置条件** | 管理员已登录系统 |
| **后置条件** | 审核商品 |
| **基本操作流** | 查看待审核商品 |
| **可选操作流** | 无 |

表2.15 审核商品用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 审核商品 |
| **用例描述** | 描述了管理员使用本系统进行审核商品的过程 |
| **参与者表** | 管理员 |
| **前置条件** | 管理员已登录系统 |
| **后置条件** | 上架商品 |
| **基本操作流** | 管理员审核商品 |
| **可选操作流** | 无 |

表2.16 上架商品用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 上架商品 |
| **用例描述** | 描述了管理员使用本系统进行上架商品的过程 |
| **参与者表** | 管理员 |
| **前置条件** | 管理员已登录系统 |
| **后置条件** |  |
| **基本操作流** | 管理员对审核通过的商品进行上架。 |
| **可选操作流** | 无 |

### 2.1.7 用户管理分析

用户管理用例中包括查看用户、增加用户和删除用户等3个用例，其用例图如图2.17所示



图 2.17 用户管理用例图

下面对其用例进行分析，它们的用例描述分别如表 2.18、2.19和2.20所示

表2.18 查看用户用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 查看用户 |
| **用例描述** | 描述了管理员使用本系统进行查看用户的过程 |
| **参与者表** | 管理员 |
| **前置条件** | 管理员已登录系统 |
| **后置条件** |  |
| **基本操作流** | 查看相关用户列表。 |
| **可选操作流** |  |

表2.19 增加用户用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 增加用户 |
| **用例描述** | 描述了管理员使用本系统进行增加用户的过程 |
| **参与者表** | 管理员 |
| **前置条件** | 管理员已登录系统 |
| **后置条件** |  |
| **基本操作流** | 1.管理提供关键字；  2.增加关键字相关用户。 |
| **可选操作流** | 无 |

表2.20 删除用户用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 删除用户 |
| **用例描述** | 描述了管理员使用本系统进行删除用户的过程 |
| **参与者表** | 管理员 |
| **前置条件** | 管理员已登录系统 |
| **后置条件** |  |
| **基本操作流** | 1.管理员提供关键字；  2.删除关键字相关用户。 |
| **可选操作流** | 无 |

### 2.1.8 日志管理

用户管理用例中包括查看事件日志、查看系统日志和查看安全日志等3个用例，其用例图如图2.21所示



图 2.21 日志管理用例图

表2.22 查看事件日志用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 查看事件日志 |
| **用例描述** | 描述了管理员使用本系统进行查看事件日志的过程 |
| **参与者表** | 管理员 |
| **前置条件** | 管理员已登录系统 |
| **后置条件** |  |
| **基本操作流** | 查看事件日志信息 |
| **可选操作流** | 无 |

表2.23 查看系统日志用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 查看系统日志 |
| **用例描述** | 描述了管理员使用本系统进行查看系统日志的过程 |
| **参与者表** | 管理员 |
| **前置条件** | 管理员已登录系统 |
| **后置条件** |  |
| **基本操作流** | 查看系统日志 |
| **可选操作流** | 无 |

表2.24 查看安全日志用例描述

|  |  |
| --- | --- |
| **描述项** | **说明** |
| **用例名称** | 查看安全日志 |
| **用例描述** | 描述了管理员使用本系统进行查看安全日志的过程 |
| **参与者表** | 管理员 |
| **前置条件** | 管理员已登录系统 |
| **后置条件** |  |
| **基本操作流** | 查看安全日志 |
| **可选操作流** | 无 |

## 2.2 数据需求

经过分析后得知系统中所需的类和各类之间的关系，利用概念类图进行描 述。图中后台管理界面类提供对用户、商品和日志的维护工作。购物界面类商品、购物车和订单各类操作，如图2.3 所示。

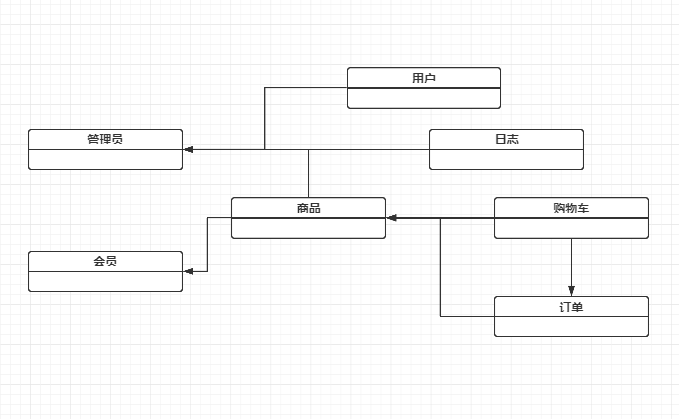


图 2.3 系统概念类图

根据以上概念类图的内容，可知网上购物系统包括商品类型、商品信息、购 物车、买家（即：用户）、订单、订单订单明细等实体。其中， 商品类型和商品信息的对应关系为一对多，购物车记录和商品信息为一对一的关 系。买家和购物车为一对多关系；一次购物车记录对应一个订单，二者为一对一 关系；一个订单具有多个订单明细，二者关系为一对多关系。置闲物品交易系统的 ER 图如图2.4 所示。



图2.4 置闲物品交易系统 E-R 图

## 2.3系统非功能性需求

### 2.3.1 可靠性和可用性需求

**安全措施需求：**

注册用户可能会忘记密码或者账号信息，但是注册用户可以通过密码找回功能将密码以及账号信息找回，并使用。

**安全性需求：**

为了防止没有修改权限的用户无意间修改了数据，用户登录分为：注册用户登录，管理员登录；不同的用户，有不同的权限，注册用户有注册用户的登录界面，管理员有管理员的登录界面。当用户登录时，系统会根据用户ID和用户密码，系统会自动进行核实，鉴定此用户是否合法，如果合法，用户才能进一步操作。

**软件质量属性：**

本系统易用性良好，使用方便，具有较好的可靠性，有效性可维护性可移植性。

### 2.3.2 出错及处理需求

1.**出错信息**

出现错误时，捕捉到网站错误，存入错误记录文件中，供相关人员查看。用户不会看到具体的错误而是跳转到错误页面。

2**.补救措施**

网站出现错误后可采取的措施：

1. 根据错误记录文件，做出相应错误处理。
2. 检测是否启动服务。
3. 备份数据，重新部署网站。
4. 重新启动服务。

3.**网站维护设计**

为方便网站的维护，在代码设计中，以对象设计为主、结构设计为辅，对于难以理解的步骤给出相应的注释，方便维护人员查看代码和阅读代码。还有会写出相应的文档来记录编码时的问题，为以后网站升级做好基础。

### 2.3.3接口需求

1、用户接口需求：只需要用户有智能手机，并有网络连接我们的网站服务器，即可进入我们的网站并进行一些需要的操作。

2、app接口需求：需要计算机与网站服务器连接。

3、通信接口需求：app运行时要链接相应的服务器；用户进行注册，然后进行登入即可进入app进行需要的操作。

### 2.3.4 网站未来可能需求

1、未来用户需求：

如果“置闲物品交易网站”有幸得到大家喜爱，随着用户的增多可能会增加更多好玩的活动、更多的用户体验，从而使得本网站更加丰富，更能满足用户的需求。

2、未来系统需求：

分析未来的用户量的剧增，那就对网站的内部要求更苛刻了，我们会分地区建立多个服务器，然后由一个总的服务器进行管理下面的服务器，把服务器的硬件加大，对数据库进行数量增加。

## 2.4可行性分析

### 2.4.1网站可行性研究

可行性研究阶段的主要任务是在网站初步调查的基础上，对新网站是否能够实现和值得实现等问题做出判断，避免在花费了大量的人力和物力之后才发现网站不能实现或新网站投入使用后没有任何实际意义而引起的浪费，对新网站可行性的分析，要求用最小的代价在尽量短的时间内确定网站是否可行。

### 2.4.2网站技术可行性分析

在线考试系统的开发是一项复杂的系统工程。为了保证系统开发成功，必须采用工程化的系统开发方法，并研究出一些符合工程化标准的开发方法。这些方法旨在指导开发者进行工程化的系统开发，从而加快系统开发的速度，保证质量以及降低开发成本。工程化的系统开发方法确实在开发实践中取得了一定的效果。此次开发使用C#作为开发语言，采用ASP.NET技术, Access数据库IIS服务器。

### 2.4.3运行可行性分析

随着计算机知识的普及和推广，越来越多的人掌握了计算机的基本使用方法和技能。随着Internet的发展，用户对于网络、WINDOWS等环境下的软件使用比较熟悉，对于新鲜事物，用户表现出极大的兴趣和热情。

### 2.4.4市场可行性分析

网上物品买卖（交流交易）给人们带来了方便同时给商家带来了成本的节省，成为一种全新的物品销售模式，商家可以免去店面的成本和雇佣人员的成本，更大限度的做到还利益于消费者，所以构建一个网上商店在经济上是完全可行的。就本网站而言，随着电脑的普及，为消费者在网络上进行购物提供有利的条件。而一个网上交易商店可以为消费者提供软件条件，这样消费者便可以足不出户的买商品了。商家可以从中获得利润，两全其美。通过以上的分析，开发网上交流网站是完全可行的。

# 第 3 章 系统设计

## 3.1 功能模块设计

根据需求阶段的功能分析可知，置闲物品交易系统可以划分为登录注册、商品选购、商品管理和订单管理4 个模块，其功能包图如图 3.1 所示。



图3.1 置闲物品交易系统功能包图

本节中，将对置闲物品交易系统功能包图中的各个功能模块进行具体设计，

具体如下。

### 3.1.1 用户管理设计

**（1）功能结构设计**

“用户管理”功能分为注册、登录和修改个人信息三个部分，其包图如图3.2所示。



图 3.2 用户管理包图

**（2）类图设计**



图 3.3 用户管理类

如上图 3.3 所示是用户管理类图中，一共有 2个类，其中：用户类主要负责获取用户、查找用户、添加用户、删除用户和修改用户功能；用户信息类负责修改用户信息等功能。用户与用户信息类为一对一的对应关系。

**（3）顺序图设计**

注册功能的顺序图如图 3.4 所示。

图 3.4 注册功能顺序图

登录功能的顺序图如图 3.5 所示

图 3.5登录功能顺序图

修改用户信息功能的顺序图如图 3.6 所示



图 3.6修改用户信息功能顺序图

**（4）核心处理流程设计**

针对上述顺序图，图 3.5 登录功能和图 3.6 修改个人信息功能这两个功

能还可以进一步细化，如图 3.7 和图 3.8 所示。 

图 3.7 登录活动图



图 3.8 修改个人信息活动

### 3.1.2 商品选购设计

**（1）功能结构设计**

“商品选购”功能主要目的是为了用户提供购买卖家提供的商品的功能。此外，该功能还负责将用户选中的商品放入购物车，对购物车进行管理。因此,商品选购模块主要涉及商品信息的查询获取以及购物车管理两个子模块。其中商品信息获取模块主要分为检索商品和查看商品详细信息两个组成部分。购物车管理又分为将商品放入购物车和将商品从购物车移除两个部分，其包图如图 3.9所示。



图 3.9 商品选购包图

**（2）类图设计**

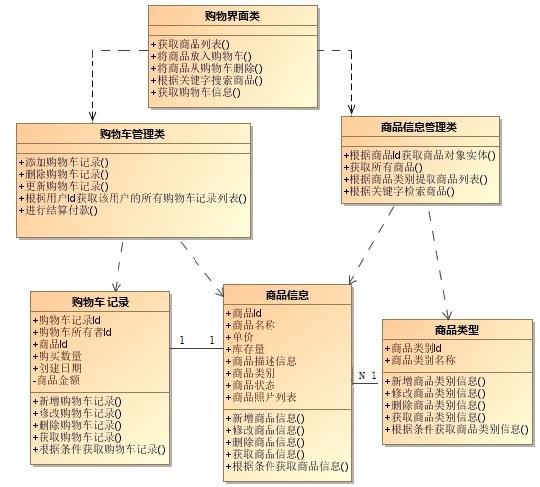


图 3.10 商品选购类图

如上图3.10 所示是商品选购类图中,一共有 6 个类，其中：购物界面类主要负责响应页面发出的商品信息获取请求，负责购物车的维护，它的执行依赖 于购物车管理类和商品信息管理类；购物车管理类是购物车管理主功能类；商 品信息管理类主要负责根据各类格式或条件获取商品的信息以及对商品类型的 管理，依赖于商品信息类和购物车类两个实体类；商品信息类是商品信息实体 类，具备单元信息维护功能；商品类别类是商品类别实体类，具备单元信息维 护功能；购物车类是购物车实体类，具备单元信息维护功能。商品信息类与购 物车类为多对多的关联关系；商品类型和商品信息为一对多的对应关系。

**（3）顺序图设计**

商品信息获取功能的顺序图如图 3.11 所示。

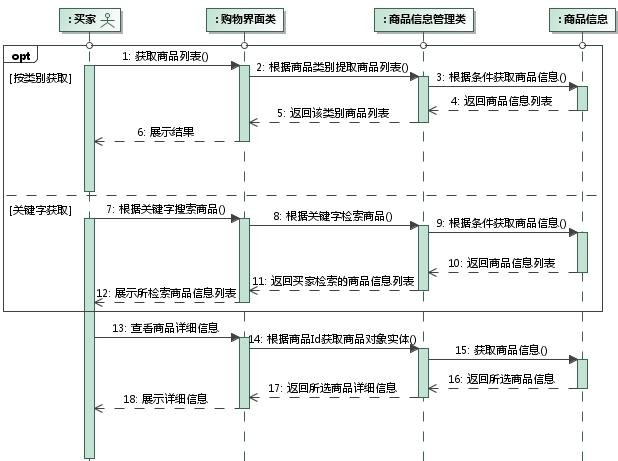


图 3.11 商品信息获取顺序图

购物车管理功能的顺序图如图 3.12 所示。

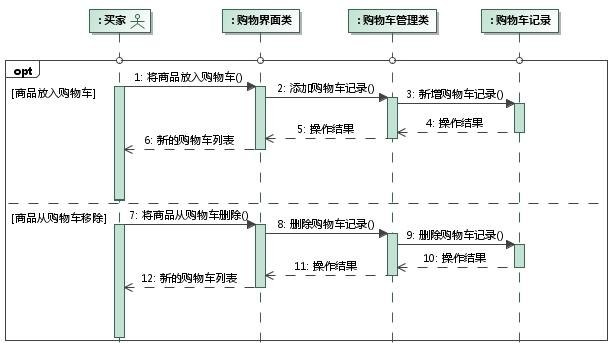


图 3.12 购物车管理顺序图

**（4）核心处理流程设计**

针对上述顺序图，图 3.11 中“根据关键字搜索商品”和图 3.12 中“添加购物车

记录”这两个方法还可以进一步细化，如图 3.13 和图 3.14 所示。

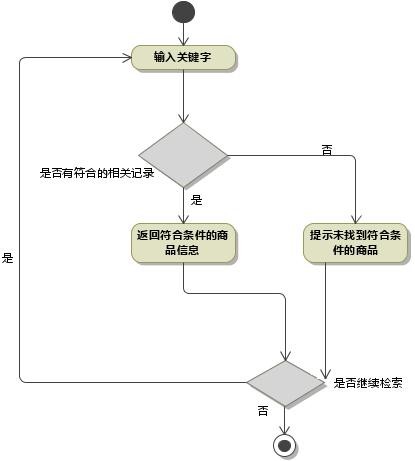


图 3.13 根据关键字搜索商品活动图

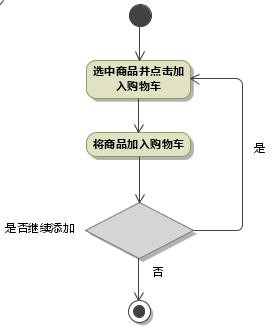


图 3.14 添加购物车记录活动图

### 3.1.3 商品管理设计

**（1）功能结构设计**

“商品管理”功能的主要目的是为了用户能上传自己的置闲物品，以提供其他用户购买的功能。此外还负责对自己上传的商品信息进行修改和下架商品。因此商品管理模块主要是商品信息的获取后对商品的增删改模块。其中商品获取模块主要分为展示用户商品列表和对商品的操作模块。其包图如图3.15所示。



图 3.15 商品管理包图

**（2）类图设计**



图 3.16 商品管理类图

如上图3.16所示的商品管理类图中，一共有3个类，其中：商品类负责获取商品信息、添加商品、修改商品和删除商品功能；商品类型类负责获取商品类型、添加类型、修改类型和删除类型功能；用户商品类主要是建立用户与商品的映射关系，获取用户的商品功能。商品类与商品类型类为多对一的对应关系；商品类与用户商品类为多对一的对应关系。

**（3）顺序图设计**

添加商品功能的顺序图如图3.17所示。

图 3.17 添加商品顺序图

查询用户商品列表功能的顺序图如图3.18所示



图 3.18 获取商品列表顺序图

**（4）核心处理流程设计**

针对上述顺序图，对图进行进一步细化。如图3.19和3.20所示。



图 3.19 添加商品活动图



图 3.20 商品获取活动图

### 3.1.4 订单管理设计

**（1）功能结构设计**

“订单管理”功能主要分为提交订单、取消订单和查询订单三部分。其中查询订单分为查看所有订单和查看单个订单信息两个部分。其包图如图3.21所示。

图 3.21 订单管理功能包图

**（2）类图设计**



图 3.22 订单管理类图

如上图3.22所示是订单管理类中，一共有两个类，其中订单类负责获取用户订单等功能；订单明细表负责提交订单和获取订单明细信息等功能。其中订单类与订单明细类为一对多的对应关系。

**（3）顺序图设计**

订单列表获取功能的顺序图如图3.23所示。



图 3.23 获取订单列表顺序图



订单明细获取功能的顺序图如图3.24所示。

图 3.24 获取订单明细顺序图

**（4）核心处理流程设计**

针对上述顺序图，对其进一步细化，如图3.25所示



图 3.25 订单管理活动图

## 3.2 数据库设计

置闲物品交易系统的主要表的结构如下所示。

（1）商品类别

保存商品类别的基本信息。如表 3.26 所示。

表3.26商品类别表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **数据类型** | **长度** | **主键/外键** | **描述** |
| TypeID | Guid |  | **主键** | 商品类别 ID |
| TypeName | nvarchar(20) | 20 | 否 | 商品类别名称 |

（2）商品信息

保存商品的基本信息。如表3.27 所示。

表3.27 商品信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **数据类型** | **长度** | **主键/外键** | **描述** |
| ProductID | Guid |  | **主键** | 商品 ID |
| ProductName | nvarchar(150) | 150 | 否 | 商品名称 |
| Price | int |  | 否 | 单价 |
| Stock | int |  | 否 | 库存量 |
| Description | nvarchar(255) | 255 | 否 | 描述信息 |
| PhotoList | Binary |  | 否 | 商品照片 |
| Type | nvarchar(50) | 50 | 否 | 商品类别 |
| Status | nvarchar(10) | 10 | 否 | 商品状态 |

（3）购物车记录

保存购物车记录的基本信息。如表 3.28 所示。

表3.28 购物车记录表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **数据类型** | **长度** | **主键/外键** | **描述** |
| Id | Guid |  | **主键** | 购物车记录 ID |
| OwnerId | Guid |  | **外键** | 购物车所有者 ID |
| GoodId | Guid |  | **外键** | 商品 ID |
| Quantity | int |  | 否 | 购买数量 |
| CreateDate | Datetime |  | 否 | 创建日期 |
| Money | int |  | 否 | 商品总额 |

（4）订单项

保存订单项的基本信息。如表 3.29 所示。

表3.29 订单表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **数据类型** | **长度** | **主键/外键** | **描述** |
| ID | Guid |  | **主键** | ID |
| OrderID | Guid |  | **外键** | 订单 ID |
| **UserID** | Guid |  | **外键** | 客户ID |
| Status | int |  | 否 | 订单状态 |
| TotalPrice | int |  | 否 | 商品总额 |

（5）订单

保存订单的基本信息。如表 3.30 所示。

表3.30 订单明细表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **数据类型** | **长度** | **主键/外键** | **描述** |
| Id | Guid |  | **主键** | 订单 ID |
| OrderID | Guid |  | **外键** | 客户 ID |
| UserID | Guid |  | **外键** | 卖家ID |
| ProductID | Guid |  | **外键** | 商品ID |
| Quantity | Int |  |  | 数量 |
| UnitPrice | Decimal |  |  | 单价 |
| CreateTime | DataTime |  |  | 创建时间 |
| Status | String |  |  | 订单状态 |
| Address | String |  |  | 地址 |

（6）用户表

保存用户的基本信息。如表 3.31 所示

表 3.31 用户表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **数据类型** | **长度** | **主键/外键** | **描述** |
| **UserID** | Guid |  | **主键** | 用户 ID |
| **UserName** | nvarchar(10) | 10 | 否 | 用户姓名 |
| **Password** | nvarchar(20) | 20 | 否 | 密码 |

（7）用户信息表

保存用户的基本信息。如表 3.32 所示。

表 3.32 用户信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **数据类型** | **长度** | **主键/外键** | **描述** |
| UserID | Guid |  | **主键** | 用户 ID |
| Email | nvarchar(50) | 50 | 否 | 电子邮件 |
| MobilePhone | nvarchar(50) | 50 | 否 | 手机号码 |
| Tel | nvarchar(50) | 50 | 否 | 座机号码 |
| Address | nvarchar(255) | 255 | 否 | 发货地址 |
| Sex | Byte |  | 否 | 性别 |
| Avatar | String |  | 否 | 头像 |

给出数据表之间的关系图，如图3.33所示。



表 3.33数据表关系图表

# 第 4 章 模块实现

## 4.1 系统软件架构

### 4.1.1 系统开发架构

本系统遵循DDD（领域驱动设计）的原则，基于asp.net core的开发框架，将工程分为四个层，如图4.1所示。



图4.1 系统架构图

如图4.1所示的系统架构中，具体的层次职责如下：

（1）展现层（Presentation）：提供用户界面，实现用户交互操作。

（2）Web层：提供应用程序编程接口（Web Api）。

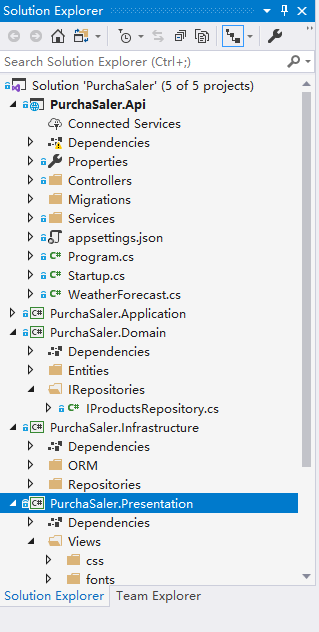
（3）应用层（Application）：进行展现层与领域层之间的协调，协调业务对象来执行应用程序的任务。

（4）领域层（Domain）：领域层就是业务层，是一个项目的核心，所有的业务规则都应该在领域层实现。实体（Entity）代表业务领域的数据和操作，在实践中，通过用来映射成数据库表；仓储接口（IRepository）：仓储用来操作数据库进行数据存取。仓储接口在领域层定义。

（5）基础设施层（Infrastructure）：提供通用技术来支持更高的层。ORM工具使用EntityFrameWorkCore，仓储（Repository）：对领域层仓储接口的实现。

DDD（领域驱动设计）架构中，Presentation（展示层）、Web层(Api)、Application（应用层）、Domain（领域层）和Infrastructure（基础设施层）各司其职，意在职责分离。Infrastructure层的Repository实现Domain层的IRepository接口，相当于IDAL和DAL，通过Infrastructure层ORM来实现与数据库的交互，Web层Api来接收这些数据，Presentation层的HTML通过请求API资源来渲染页面。

置闲物品交易系统实现的工程文件结构图如图4.2所示。



工程文件结构图4.2

### 4.1.2 .net core框架全局配置appsetting.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"ConnectionStrings": {

"constr": "Data Source=localhost;Database=PurchaSalerV1;User id=sa;Password=Pass@word"

},

"JwtSetting": {

"issuer": "PurchaSaler.com",

"audience": "PurchaSaler.com",

"secretkey": "rtryuiqnjkxnjjkslakjbjnzxhhfulzxl"

},

"AllowedHosts": "\*"

}

### 4.1.3 StartUp配置

StartUp配置如下：

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using PurchaSaler.Domain.IRepositories;

using PurchaSaler.Infrastructure.ORM;

using PurchaSaler.Infrastructure.Repositories;

using System;

using System.Text;

namespace PurchaSaler.Api

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddScoped<IProductsRepository, ProductsRepository>();

services.AddControllers();

//jwt验证

var key = Encoding.UTF8.GetBytes(Configuration["JwtSetting:secretkey"]);

services.AddAuthentication(option =>

{

option.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

option.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

option.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;

}).AddJwtBearer(option =>

{

option.RequireHttpsMetadata = false;

option.SaveToken = false;

option.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(key),

ValidateIssuer = false,

ValidateAudience = false,

ClockSkew = TimeSpan.Zero

};

});

//注册Cors服务

services.AddCors(option =>

{

option.AddPolicy("any", buider =>

{

buider.AllowAnyOrigin()

.AllowAnyMethod()

.AllowAnyHeader();

});

});

services.AddDbContext<PurchaSalerDbContext>(option => option.UseSqlServer(

Configuration.GetConnectionString("constr"),b=>b.MigrationsAssembly("PurchaSaler.Api")));

services.AddSwaggerGen(option =>

{

option.SwaggerDoc("v1.1", new OpenApiInfo

{

Title="PurchaSaler",

Version="v1.1"

});

});

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseCors();

app.UseAuthentication();

app.UseAuthorization();

app.UseSwagger();

app.UseSwaggerUI(option =>

{

option.SwaggerEndpoint("/swagger/v1.1/swagger.json", "PurchaSaler");

});

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

}

## 4.2 用户管理实现

### 4.2.1 设计思想

前后端分离，会员通过注册可以拥有自己的账号，通过登录账号可以访问系统相关资源。登录时服务端产生token，客户端保存token。请求头带上token，实现权限访问。个人中心个人资料修改。

### 4.2.2 操作界面

注册界面清新简洁，没有相当复杂的步骤，对于新入网站的游客朋友非常友好，简约而不简单。如图4.2注册界面所示。



图4.2 注册界面

登录界面清新简洁，没有相当复杂的步骤，对于新入网站的游客朋友非常友好，简约而不简单。如图4.2登录界面所示。



图4.3 登录界面

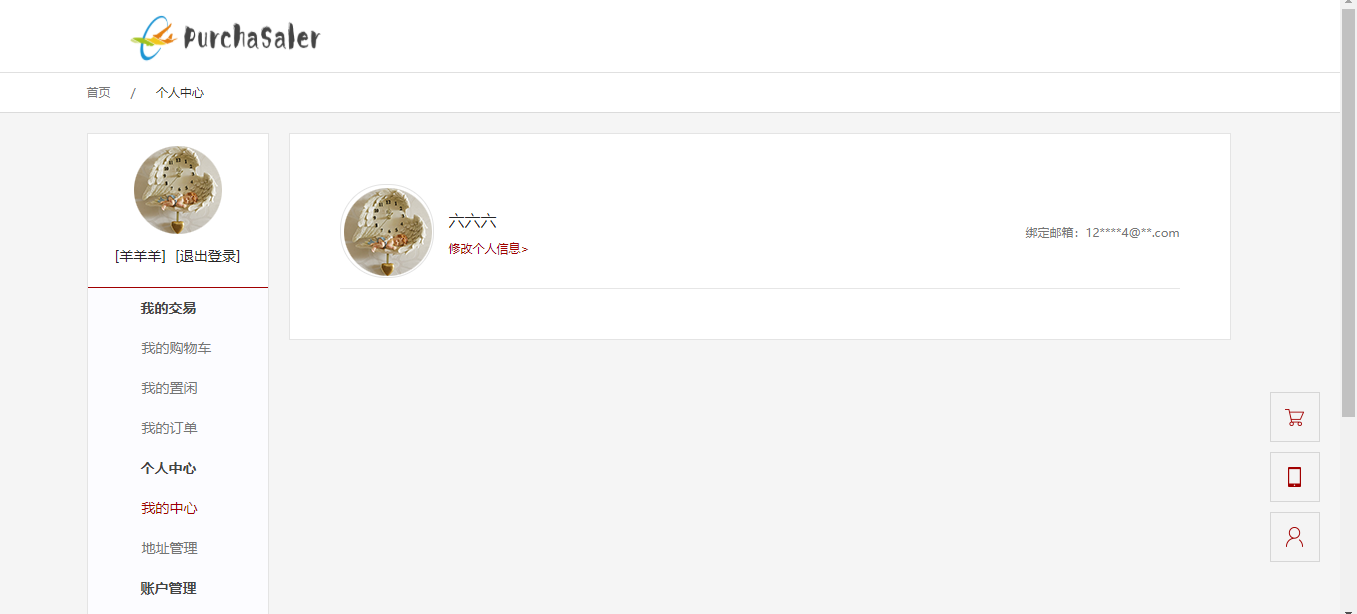


图4.4 个人中心界面

### 4.2.3 核心代码

**跨域处理**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Cors;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace PurchaSaler.Api.Controllers

{

    [EnableCors("any")]

    [Route("api/[controller]")]

    [ApiController]

    public class CorsController : ControllerBase

    {

    }

}

services.AddCors(option =>

            {

                option.AddPolicy("any", buider =>

                 {

                     buider.AllowAnyOrigin()

                     .AllowAnyMethod()

                     .AllowAnyHeader();

                 });

            });

app.UseCors();

**JsonWenToken生成**

using Microsoft.Extensions.Configuration;

using Microsoft.IdentityModel.Tokens;

using PurchaSaler.Domain.Entities;

using System;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace PurchaSaler.Api.Services

{

    public class JsonWebToken

    {

        private readonly IConfiguration \_config;

        public JsonWebToken(IConfiguration config)

        {

            \_config = config;

        }

        public string GenerateToken(Users userinfo)

        {

            var tokenDescriptor = new SecurityTokenDescriptor

            {

                Subject = new ClaimsIdentity(new Claim[]

                {

                    //使用Name存储用户id,以便取出

                    new Claim(ClaimTypes.Name, userinfo.UserID.ToString())

                }),

                Expires = DateTime.UtcNow.AddMinutes(2),

                SigningCredentials = new SigningCredentials(

                    new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config["JwtSetting:secretkey"])),

                    SecurityAlgorithms.HmacSha256Signature)

            };

            var tokenHandler = new JwtSecurityTokenHandler();

            var securityToken = tokenHandler.CreateToken(tokenDescriptor);

            var token = tokenHandler.WriteToken(securityToken);

            return token;

        }

        public SecurityToken DecodeToken(string encodetoken)

        {

            var json = new JwtSecurityTokenHandler().ReadToken(encodetoken);

            return json;

        }

    }

}

**登录前端**

<script>

    var vue = new Vue({

        el:"#app",

        data() {

            return {

                username:"",

                password:"",

                url:"http://localhost:5000/api/Users/Login",

            }

        },

        methods: {

            login(){

                axios.post(vue.url,{

                username:vue.username,

                password:vue.password

                })

                .then(function(res){

                    // console.log(res.data)

                    vue.saveToken(res.data);

                    vue.toUrl();

                })

                .catch(function(error){

                    console.log(error);

                })

            },

            saveToken(token){

                localStorage.setItem("token",token);

            },

            toUrl(){

                console.log('登录成功');

                window.location.href="index.html";

            }

        },

    });

</script>

**注册登录后端**

using Microsoft.AspNetCore.Cors;

using Microsoft.AspNetCore.Mvc;

using NETCore.Encrypt;

using PurchaSaler.Domain.Entities;

using PurchaSaler.Domain.IRepositories;

using PurchaSaler.Api.Services;

using Microsoft.Extensions.Configuration;

using PurchaSaler.Api.ViewModel;

namespace PurchaSaler.Api.Controllers

{

    [Route("api/[controller]")]

    [EnableCors("any")]

    [ApiController]

    public class UsersController : ControllerBase

    {

        private readonly IUsersRepository \_usersRepository;

        private readonly IConfiguration \_config;

        public UsersController(IUsersRepository usersRepository, IConfiguration config)

        {

            \_usersRepository = usersRepository;

            \_config = config;

        }

        [HttpPost("Register")]

        public IActionResult Register(RegisterVM user)

        {

            bool IsExisted = \_usersRepository.IsExisted(user.UserName);

            if(ModelState.IsValid && !IsExisted)

            {

                //哈希密码

                user.Password = EncryptProvider.Md5(user.Password);

                var man = new Users()

                {

                    UserName = user.UserName,

                    Email = user.Email,

                    Password = user.Password

                };

                \_usersRepository.AddUser(man);

                return Ok("注册成功");

            }

            else

            {

                return StatusCode(422,"用户已存在");

            }

        }

        [HttpPost("Login")]

        public IActionResult Login(LoginVM user)

        {

            if (ModelState.IsValid)

            {

                bool IsExisted = \_usersRepository.IsExisted(user.UserName);

                //用户是否存在

                if(IsExisted)

                {

                    //取出这个人的信息

                    var man = \_usersRepository.GetUserByName(user.UserName);

                    //验证密码

                    if (EncryptProvider.Md5(user.Password) == man.Password)

                    {

                        //生成token

                        var jwt = new JsonWebToken(\_config);

                        var token = jwt.GenerateToken(man);

                        return new JsonResult(token);

                    }

                    else

                    {

                        return StatusCode(422, "密码错误");

                    }

                }

                else

                {

                    return StatusCode(422, "用户不存在");

                }

            }

            else

            {

                return StatusCode(422, "实体验证失败");

            }

        }

    }

}

### 4.3.4 测试

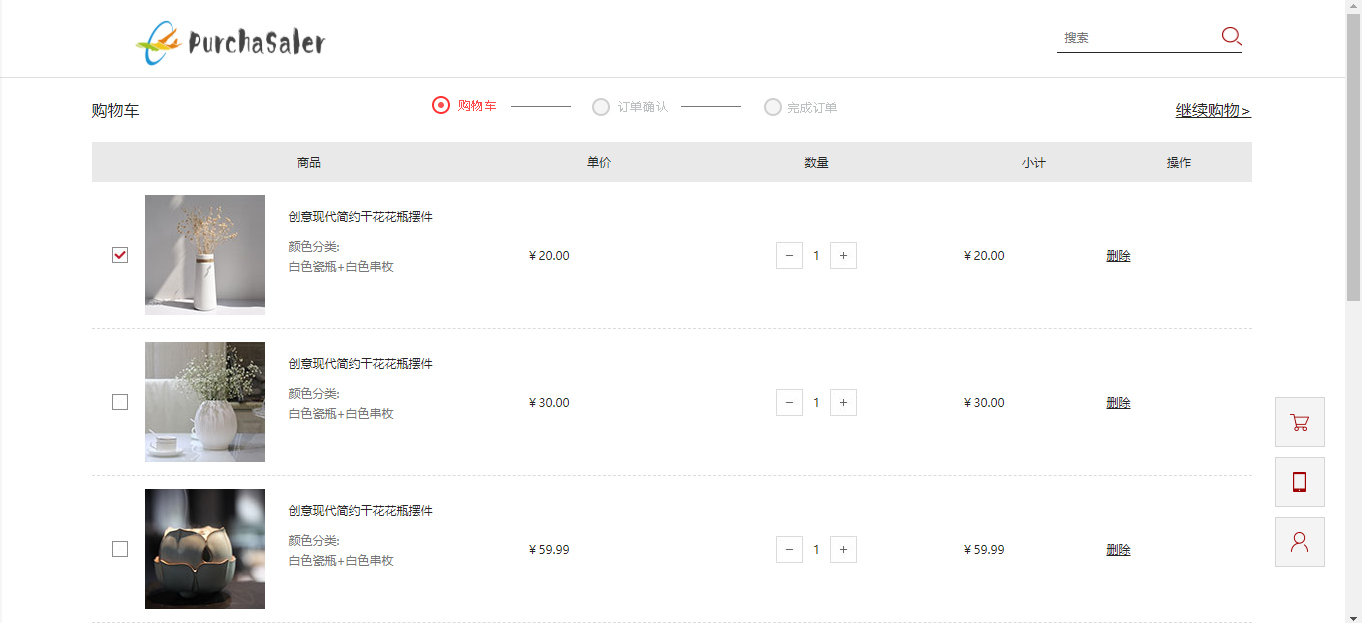
|  |  |  |
| --- | --- | --- |
| 输入/动作 | 期望的输出/响应 | 实际情况 |
| 已有用户名注册 | 用户已存在 | 用户已存在 |
| 未登录点击用户中心 | 请登录 | 请登录 |
| 输入正确用户名和密码 | 登录成功 | 登录成功 |

## 4.3 商品选购实现

### 4.3.1 设计思想

商品展示，用户通过点击可查看商品的详情，登录用户在商品详情页可将商品加入购物车和直接购买结算。

### 4.3.2 操作界面



### 4.3.3 核心代码

**商品展示**

<div class="wrapper" v-for="p in products" :key="p.id">

                <div class="pList clearfix tran" style="float: left;">

                    <a :href="'productDetail.html?pid=' + p.productID " style="margin-left: 16px;">

                        <!-- <a :href="'#/newsDetail/' + item.id "></a> -->

                        <dl>

                            <dt>

                                <span class="abl"></span>

                                <img :src="p.photos"/>

                                <span class="abr"></span>

                            </dt>

                            <dd>{{p.productName}}</dd>

                            <dd><span>￥{{p.price}}</span></dd>

                        </dl>

                    </a>

                </div>

            </div>

<script>

    var vue = new Vue({

        el:"#app",

        data() {

            return {

                products:[],

            };

        },

        mounted(){

            this.getAllProductList();

        },

        methods: {

            getAllProductList() {

                var thisVue = this;

                axios.get("http://localhost:5000/api/home")

                .then(function(res){

                    vue.products = res.data;

                    // console.log(res);

                })

                .catch(function(error){

                    console.log(error);

                })

            },

        },

    });

</script>

using System.Linq;

using Microsoft.AspNetCore.Mvc;

using PurchaSaler.Domain.IRepositories;

namespace PurchaSaler.Api.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    public class HomeController : CorsController

    {

        private readonly IProductsRepository \_productsRepository;

        public HomeController(IProductsRepository productsRepository)

        {

            \_productsRepository = productsRepository;

        }

        [HttpGet]

        public IActionResult Index()

        {

            var list = \_productsRepository.GetAllProducts();

            return new JsonResult(list);

        }

    }

}

**商品详情**

<script>

    var vue = new Vue({

        el:"#app",

        data() {

            return {

            }

        },

        mounted(){

            //获取商品id

            var pid = this.getUrlParam("pid");

        },

        methods: {

            getUrlParam(name) {

                var reg = new RegExp("(^|&)" + name + "=([^&]\*)(&|$)");

                var r = window.location.search.substr(1).match(reg);

                if (r != null) return unescape(r[2]); return null;

            },

        },

    });

</script>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using PurchaSaler.Domain.IRepositories;

using PurchaSaler.Infrastructure.ORM;

namespace PurchaSaler.Api.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    public class ProductsController : CorsController

    {

        private readonly IProductsRepository \_productsRepository;

        public ProductsController(IProductsRepository productsRepository)

        {

            \_productsRepository = productsRepository;

        }

        [HttpGet("ProductDetail")]

        public IActionResult GetProductDetail(Guid productid)

        {

            var product = \_productsRepository.GetProductByID(productid);

            return new JsonResult(product);

        }

    }

}

**购物车管理**

using System;

using System.Security.Claims;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using PurchaSaler.Domain.Entities;

using PurchaSaler.Domain.IRepositories;

namespace PurchaSaler.Api.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    public class ShoppingCartController : CorsController

    {

        private readonly IShoppingCartsRepository \_shoppingCartsRepository;

        public ShoppingCartController(IShoppingCartsRepository shoppingCartsRepository)

        {

            \_shoppingCartsRepository = shoppingCartsRepository;

        }

        [Authorize]

        [HttpGet]

        public IActionResult GetCarts(Guid userid)

        {

            var claimsIdentity = User.Identity as ClaimsIdentity;

            var userId = claimsIdentity.FindFirst(ClaimTypes.Name)?.Value;

            var shoppingcart = \_shoppingCartsRepository.GetSomeOneAllShoppingCarts(userid);

            return new JsonResult(shoppingcart);

        }

        [HttpPost]

        public IActionResult AddShoppingCarts(ShoppingCarts carts)

        {

            var claimsIdentity = User.Identity as ClaimsIdentity;

            Guid userid =new Guid (claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);

            Guid productid = carts.ProductID;

            int result = \_shoppingCartsRepository.GetShoppingCartsCount(userid,productid);

            if (result > 0)

            {

                return Content("<script>alert('该商品已存在购物车');history.go(-1)</script>");

            }

            else

            {

                carts.TotalAmount = carts.Number \* carts.UnitPrice;

                \_shoppingCartsRepository.AddShoppingCarts(carts);

                return Content("<script>alert('添加成功');history.go(-1)</script>");

            }

        }

        [HttpPost]

        public ActionResult Remove(Guid productid)

        {

            var delobj = \_shoppingCartsRepository.GetOneShoppingCart(productid);

            if (delobj != null)

            {

                \_shoppingCartsRepository.RemoveShoppingCarts(delobj);

            }

            return Ok();

        }

        [HttpPost]

        public ActionResult RemoveAll(Guid userid)

        {

            var removeall = \_shoppingCartsRepository.GetSomeOneAllShoppingCarts(userid);

            if (removeall != null)

            {

                \_shoppingCartsRepository.RemoveAllShoppingCarts(removeall);

            }

            return Ok();

        }

    }

}

### 4.3.4 测试

|  |  |  |
| --- | --- | --- |
| 输入/动作 | 期望的输出/响应 | 实际情况 |
| Token失效点击加入购物车 | 请登录 | 请登录 |
| 结算 | 结算成功，跳转订单 | 结算成功，跳转订单 |

## 4.4 商品管理实现

### 4.4.1 设计思想

用户登录后可以发布自己的置闲物品，并对自己发布的商品进行编辑修改

### 4.4.2 操作界面

### 4.4.3 核心代码

**图片上传**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace PurchaSaler.Api.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    public class ImagesController : CorsController

    {

        private readonly IWebHostEnvironment \_environment;

        public ImagesController(IWebHostEnvironment environment)

        {

            \_environment = environment;

        }

        [HttpPost("UploadImages")]

        public async Task<string> UploadImages(IFormFile file)

        {

            try

            {

                if (file.Length > 0)

                {

                    if (!Directory.Exists(\_environment.WebRootPath + "\\Upload\\"))

                    {

                        Directory.CreateDirectory(\_environment.WebRootPath + "\\Upload\\");

                    }

                    using (FileStream fileStream = System.IO.File.Create

                        (\_environment.WebRootPath + "\\Upload\\" + file.FileName))

                    {

                        await file.CopyToAsync(fileStream);

                        fileStream.Flush();

                        return "\\Upload\\" + file.FileName;

                    }

                }

                else

                {

                    return "Failed";

                }

            }

            catch (Exception ex)

            {

                return ex.Message.ToString();

            }

        }

    }

}

**商品管理**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using PurchaSaler.Domain.Entities;

using PurchaSaler.Domain.IRepositories;

using PurchaSaler.Infrastructure.ORM;

namespace PurchaSaler.Api.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    public class ProductsController : CorsController

    {

        private readonly IProductsRepository \_productsRepository;

        public ProductsController(IProductsRepository productsRepository)

        {

            \_productsRepository = productsRepository;

        }

        [HttpGet("ProductDetail")]

        public IActionResult GetProductDetail(Guid productid)

        {

            var product = \_productsRepository.GetProductByID(productid);

            return new JsonResult(product);

        }

        [HttpPost("AddProduct")]

        public IActionResult AddProduct(Products product)

        {

            var claimsIdentity = User.Identity as ClaimsIdentity;

            product.OwnerID = new Guid(claimsIdentity.FindFirst(ClaimTypes.Name)?.Value);

            \_productsRepository.AddProduct(product);

            return Ok();

        }

        [HttpPost("UpdateProduct")]

        public IActionResult UpdateProduct(Products product)

        {

            \_productsRepository.UpdateProduct(product);

            return Ok();

        }

        [HttpPost("DelProduct")]

        public IActionResult DelProduct(Guid productid)

        {

            \_productsRepository.DelProducts(productid);

            return Ok();

        }

    }

}

### 4.4.4 测试

|  |  |  |
| --- | --- | --- |
| 输入/动作 | 期望的输出/响应 | 实际情况 |
| Token失效点击编辑物品 | 请登录 | 请登录 |
| 输入相应项，发布置闲 | 发布成功 | 发布成功 |
| 下架商品 | 下架成功 | 下架成功 |

## 4.5 订单管理实现

### 4.5.1 设计思想

登录用户对某商品直接结算或对购物车商品进行结算，生成订单

### 4.5.2 操作界面



### 4.5.3 核心代码

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using PurchaSaler.Domain.Entities;

using PurchaSaler.Domain.IRepositories;

namespace PurchaSaler.Api.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    public class OrdersController : CorsController

    {

        private readonly IOrdersRepository \_ordersRepository;

        private IOrderItemsRepository \_orderItemsRepository;

        public OrdersController(IOrdersRepository ordersRepository, IOrderItemsRepository orderItemsRepository)

        {

            \_ordersRepository = ordersRepository;

            \_orderItemsRepository = orderItemsRepository;

        }

        [HttpPost]

        public IActionResult GenerateOrders(Orders orders,OrderItems orderItems)

        {

            orders.OrderID = new Guid();

            \_ordersRepository.AddOrders(orders);

            orderItems.OrderID = orders.OrderID;

            \_orderItemsRepository.AddOrderItems(orderItems);

            return Ok();

        }

    }

}

### 4.5.4 测试

|  |  |  |
| --- | --- | --- |
| 输入/动作 | 期望的输出/响应 | 实际情况 |
| Token失效点击删除订单 | 请登录 | 请登录 |
| 查看订单详情 | 展示订单详情 | 展示订单详情 |

# 第 5 章 总结和展望

## 5.1 总结

经过这几个月来对系统的分析、设计，使我对网络系统有了全面的了解与认识，同时对网站做了具体的解剖。 我认为网站成功的关键在于以下几方面：

（1）导航界面的清晰，简单设计。界面的清晰简单可以使用户一目了然，轻松的找到自己浏览本系统的目的所在。不需要在花费大量时间来寻找本系统的内容，浪费用户时间，甚至打消用户的积极性。

（2）系统的易管理性。系统的管理对于系统的维护来说很重要，社会是不断进步与发展的，用户对于网上娱乐的要求也会越来越高。因此系统要随着用户的需求而不断发生变化，不断更新系统的内容来满足用户更多的需求。本系统是利用ASP.NET作为主要实现软件（配合dreamweaver，HTML等），结合ASP.NET语言来实现的运行环境简单，一般的浏览器都可进行浏览。

本系统的大部分模块已经实现，并且也符合以上几个特性。但是，一个完整的系统仅仅靠近几个月的分析、设计来开发是远远不够的，它需要更长的时间来测试并完善。因此，本系统仍然存在着一些需要改进和不足的问题。比如系统的整体设计需要进一步的美观，系统还不够完善还需要进一步的改善和加强，系统的内容需要更加丰富和多样化。随着研究的进行，可能还会出现其他一些问题，我会继续努力，逐步改善和不断完善本系统。完成这次设计任务总共用了三个月时间，前三个星期用来收集资料、学习要用到的各项开发技术、进行系统分析，中间五个星期用来设计系统，后四个星期用来测试及修改。论文的撰写一直贯穿其中。

回顾这段时间的系统开发工作，总结起来那就是，软件的开发是相当辛苦的，由于是个人项目，所有的内容都自己包揽，但成功以后的喜悦也是非常美妙的，而且我发现，其中你投入的心血越多，成功以后你所获得的快乐与充实感也更多更强。

## 5.2 展望

.NET5即将发布，未来会考虑将Asp.net core3.1技术迁移至.NET5。

# 参考文献

[1] 作者甲,作者乙.论文的标题 [J].出版地（如北京）：XXX 出版社，年，卷（期）: 起始页-结束页.

[2] 作者甲,作者乙. Jwt.Net, a JWT (JSON Web Token) implementation for .NET [EB/OL]. <https://github.com/jwt-dotnet/jwt>.年-月-日. [3] 作者.学位论文的标题 [D].出版地：XXX 出版社，年.

[4] 作者甲,作者乙.书籍的标题 [M].出版地：XXX 出版社，年. [5] ... ...

[6] .

[7]

[8]

[9]

[10]

[11]

[12]

[13]