**目 录**

[1 系统相关技术概述 3](#_Toc8435)

[1.1 平台开发相关技术 3](#_Toc1387)

[1.1.1 Uni-app框架 3](#_Toc17308)

[1.1.2 SpringBoot框架 3](#_Toc31212)

[1.1.3 JWT技术 5](#_Toc10559)

[1.1.4 SpringSecurity框架 6](#_Toc30126)

[1.1.5 MyBatis框架 7](#_Toc19866)

[1.1.6 WebSocket协议 7](#_Toc26848)

[1.1.7 数据库 8](#_Toc24029)

[1.2 推荐算法相关技术 8](#_Toc23212)

[1.2.1 基于近邻的协同过滤算法 8](#_Toc95)

[2 需求分析 11](#_Toc27797)

[2.1 概述 11](#_Toc15096)

[2.2 功能需求 12](#_Toc21995)

[2.3 系统架构设计 13](#_Toc29994)

[2.4 系统功能模块设计 14](#_Toc32729)

[2.5 数据库设计 16](#_Toc15479)

[2.5.1 ER图设计 16](#_Toc28368)

[2.5.2 数据库表设计 19](#_Toc25757)

[3 系统关键模块实现 26](#_Toc19550)

[3.1 开发工具及环境 26](#_Toc10900)

[3.2 顾客端主要功能实现 26](#_Toc15541)

[3.2.1 注册与登录模块的实现 26](#_Toc30841)

[3.2.2 菜品展示以及点餐模块的实现 30](#_Toc17770)

[3.2.3 订单模块的实现 37](#_Toc6602)

[3.2.4 评价模块的实现 40](#_Toc30523)

[3.2.5 资料修改模块的实现 45](#_Toc27305)

[3.2.6 红包卡券模块的实现 51](#_Toc23249)

[3.2.7 扫桌号模块的实现 55](#_Toc8693)

[3.3 后台管理端主要功能实现 56](#_Toc24801)

[3.3.1 菜品管理模块的实现 56](#_Toc5506)

[3.3.2 订单管理模块的实现 67](#_Toc22561)

[3.3.3 红包卡券管理模块的实现 72](#_Toc4277)

[3.3.4 顾客管理模块的实现 80](#_Toc16342)

[3.3.5 联系客服模块的实现 87](#_Toc22857)

[3.3.6 其他模块的实现 91](#_Toc17436)

[4 系统测试 96](#_Toc6931)

[参考文献 111](#_Toc22796)

1. 系统相关技术概述

平台开发相关技术

* + 1. Uni-app框架

uni-app是一种跨平台开发框架，基于Vue.js和微信小程序API开发[1]。使用uni-app可以编写一套代码，通过编译可以发布到多个平台，包括iOS、Android、H5以及各种小程序（微信、支付宝、百度等）[13]。DCloud是uni-app的开发公司，早在2012年就开始研发小程序技术，并推出了技术标准。但是由于不同平台订制了自己的标准，导致开发者需要学习不同的技术标准，增加了开发者的负担。因此，DCloud推出了uni-app框架，实现了跨平台应用发布。uni-app内置了常用的组件和API，并进行了跨平台封装，可以满足大部分的业务需求。如图，在APP平台中，HTML5Plus是一个跨iOS和Android的JavaScript增强引擎。  


图 1 Uniapp框架

* + 1. SpringBoot框架

SpringBoot是一款基于Spring框架的轻量级Java开发框架，通过简化配置来简化应用的搭建和开发过程。Spring框架则是一个开源的分层架构，旨在解决企业应用开发的复杂性。它提供了在展现层、持久层和业务层中使用的企业级应用技术，并允许开发者自由选择使用哪一种组件进行开发。了解Spring框架对于理解SpringBoot框架是必要的[2]。Spring体系结构如图所示。

Spring框架采用分层架构，其中包含大约20个模块。最底层是测试模块，方便开发者对程序进行测试。核心容器层是最基础的部分，提供IoC和依赖注入功能，包含四个模块。AOP模块支持面向切面编程，可自定义方法拦截器和切入点，增强方法功能[8]。数据访问/集成层提供JDBC抽象层、对象关系映射API和事务管理等与数据库相关的功能。网络层包括一些与web开发相关的功能模块，例如文件上传、初始化IoC容器等。Spring框架的分层设计，让开发者可以根据不同要求，灵活使用不同的组件进行开发。

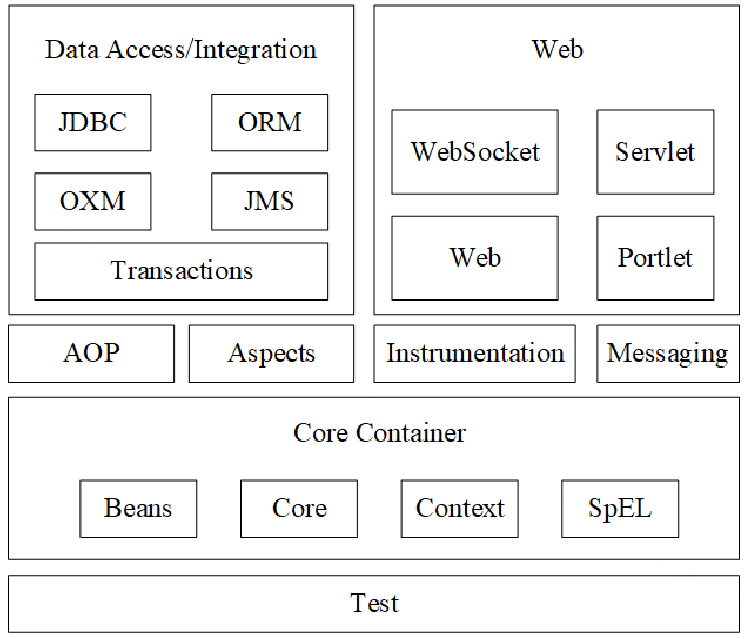


图 2 Spring框架

Spring框架相比传统的开发方式具有多项优势：方便解耦，简化开发、支持AOP编程、支持声明式事务、方便程序测试、便于集成各种优秀框架、降低了JavaEE API的使用难度。

Spring框架的配置繁琐，而SpringBoot框架简化了配置，可以轻松创建独立的Spring应用程序，内嵌Tomcat、Jetty等web容器，无需部署war文件和安装web容器软件，提供启动文件来简化配置，无需添加很多依赖文件。

* + 1. JWT技术

JWT是一种基于JSON的开放标准，全称为JSON Web Token，主要用于在不同网络应用之间传递声明。它被广泛应用于客户端与服务端之间的认证流程中。实际上，JWT是一串字符串，由头部（Header）、载荷（Payload）、签证（Signature）3个部分组成。在基于JWT的认证流程中，通常包括以下6个步骤：首先，用户使用用户名和密码进行身份验证；其次，服务器验证用户提供的凭据；然后，服务器生成JWT并将其发送回客户端；接着，客户端在每个请求中将JWT作为 Authorization header 发送到服务器；服务器在收到请求时使用密钥来验证JWT的有效性，并根据需要执行请求；最后，服务器返回数据给客户端。通过这个流程，JWT提供了一种简单、安全的方法来验证和传递用户身份信息。

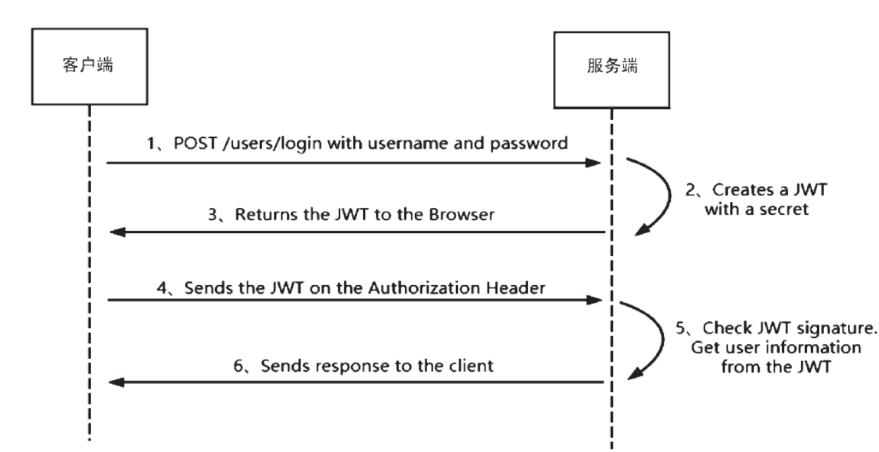


图 3 JWT认证流程图

1. 用户首次登录携带账号密码向服务器发送登录请求；
2. 服务端验证身份，根据设定的算法和JWT的结构生成一个字符串token；
3. 服务器将token返回给客户端浏览器，并在Redis中保存；
4. 然后当客户端再次请求时，只需把之前服务端返回的token发送给服务器即可；服务器发现此次请求中携带有token，则验证该用户的身份；
5. 服务器根据token中保存的信息，再将页面信息返回给客户端浏览器。

基于JWT认证方式和传统基于Session的认证方式看上去相似，但实际上有很大的区别。传统Session认证需要在服务器上存储Session，而JWT认证将token保存在客户端和Redis中，分摊了存储数据的压力，也减轻了服务器的负担。同时，Session认证依赖于Cookie传输，一旦Cookie被窃取，用户就很容易受到跨站请求伪造的攻击，而JWT认证方式则不用担心这种问题。另外，JWT中包含了用户的基本信息，可在简单业务场景下避免多次查询数据库，提高效率[9]。总的来说，JWT认证方式的优点包括以下4点。

1. 易于扩展。JWT的令牌包含了请求用户的所有基本信息，无需存储在服务器上或保存Session，有效减轻了服务器的负担。
2. 可复用性强。JWT生成的令牌是根据用户的个人信息生成的，与服务器和操作系统无关。因此，无论是在不同的服务器、操作系统平台或不同的域名下，都可以使用同一个令牌进行登录认证。此外，同一个令牌还可以在不同的应用程序之间使用。
3. 高安全性。因为JWT的认证方式不依赖于Cookie，在每次请求中不需要发送Cookie，因此可以有效地避免跨站请求伪造的攻击。同时，可以根据需求对令牌的有效时间进行设置，在令牌过期之后，用户需要重新进行登录认证。
4. 高效率。相对于传统的Session认证，JWT认证每次接收登录请求时不需要反序列化Session，只需要对令牌的签名进行校验即可。除了用于登录认证，令牌还可以提供用户的基本信息。
   * 1. SpringSecurity框架

Spring Security是Spring生态系统中的一个关键组件，它为Java企业应用程序提供了高度的安全性和可扩展性。Spring Security的主要目标是保护应用程序免受各种安全威胁，例如身份验证（authentication）和授权（authorization）等。通过使用Spring Security，开发人员可以轻松地实现安全访问控制，从而减少开发过程中需要编写的冗余代码。

Spring Security提供了一组可配置的安全过滤器链，这些过滤器用于处理用户身份验证、访问控制和其他安全相关的任务。其中，认证（authentication）是验证用户身份的过程，授权（authorization）则是决定用户是否有权限执行某些操作的过程。Spring Security提供了多种身份验证方式，包括表单验证、基本认证、OAuth2认证等，并允许开发人员根据需要自定义身份验证方式。

Spring Security还支持许多常见的安全功能，例如会话管理、加密和解密、防止跨站点请求伪造（CSRF）攻击等。此外，Spring Security提供了基于注解和基于XML的配置方式，开发人员可以根据需要选择适合自己的配置方式。

总之，Spring Security是一个成熟的安全框架，它提供了丰富的安全功能和可扩展性，使得开发人员可以轻松地保护他们的应用程序免受各种安全威胁。[10]。

* + 1. MyBatis框架

Mybatis是一个开源的持久层框架，最初由Apache的一个开源项目iBatis发展而来，后来被迁移到谷歌代码中并更名为Mybatis。它封装了JDBC的操作并扩展了参数设置和返回结果集的功能，通过解析配置文件和映射文件，扩展了原生JDBC的功能，使应用程序和数据库之间可以进行简单的交互。Mybatis的映射文件是它的重要特点之一，它将数据库中的表映射为Java对象，将SQL语句映射为Java方法，并提供了丰富的标签来支持不同的SQL语句。因为Mybatis是一个开源项目，所以它可以被广泛地使用和扩展，适用于各种规模的应用程序[14]。

* + 1. WebSocket协议

WebSocket是一种全双工通信协议，可以在Web浏览器和Web服务器之间创建实时的、双向的数据通信通道。与传统的HTTP请求-响应协议不同，WebSocket在客户端和服务器之间建立一条持久连接，双方可以随时通过该连接发送和接收数据。WebSocket使用标准的HTTP和WebSocket协议进行握手，建立连接后，通信双方可以直接以二进制或文本的方式发送消息，不需要经过任何额外的协议或格式。另外，WebSocket的消息传输是基于帧的，每个帧包含一个控制帧或一个数据帧，可以按需发送或接收。通过这种方式，WebSocket可以实现低延迟、高效率的双向通信。WebSocket在实时性要求较高的应用场景下具有广泛的应用，例如在线游戏、股票交易、即时通讯等。另外，由于WebSocket是一种基于标准的协议，因此可以跨平台、跨语言地进行通信[11]。

图示

描述已自动生成

图 4 WebSocket流程图

* + 1. 数据库

数据库是一种用于组织、存储和管理数据的仓库，其按照数据结构来组织数据。每个数据库都提供一个或多个不同的API，用于创建、访问、管理、搜索和复制所保存的数据。在本文设计的系统中，使用了MySQL和Redis两个不同类型的数据库。

MySQL是目前最流行的关系型数据库之一，它是由瑞典MySQL AB公司开发，现归属于Oracle公司[4]。MySQL建立在关系模型基础上，使用集合代数等数学概念和方法来处理数据库中的数据。与非关系型数据库将所有数据都放在一个大仓库中不同，关系型数据库将数据分散保存在不同的表中，以提高数据查询的速度。MySQL是开源的、免费使用的，具有良好的性能，可以处理拥有上千万条记录的大型数据表。其支持标准的SQL语言，可以运行在多种系统之上，并支持多种编程语言，例如Java、Python等。

Redis是一种高性能的非关系型数据库，它使用ANSI C语言编写。与MySQL不同，Redis是一个键值对型的开源数据库[3]。它不仅可以将数据存储在缓存中，还支持数据的持久化，即将缓存中的数据保存到磁盘中。Redis的安装部署更加简单，存储数据的格式更加丰富，扩展性更强，并且由于NOSQL将数据存储于缓存之中，所以查询速度更快[6]。在性能上，Redis具有读取数据的速度最高可达110000次/s，写入数据的速度最高可达81000次/s，性能极高。在使用上，Redis经常被用来缓存应用的数据，并且辅助秒杀、抢购等活动。

关系型数据库和非关系型数据库通常是互补而非对立的关系。在本文设计的系统中，使用关系型数据库MySQL存储应用的基本数据，使用非关系型数据库Redis存储一些需要经常查询且变化较少的数据。这样做的好处是可以利用MySQL的强大查询能力来获取基本数据，同时利用Redis的快速读取能力来提高系统的响应速度，从而优化系统的整体性能。

推荐算法相关技术

* + 1. 基于近邻的协同过滤算法

协同过滤算法是一种常用于推荐系统中的算法，其主要思想是通过分析用户的历史行为数据，预测用户可能会感兴趣的物品或服务，并将其推荐给用户。协同过滤算法的基本原理是利用已知的评分信息预测未知评分，然后根据预测的评分高低来进行物品推荐。其主要分为基于用户和基于物品两种类型[5]。

基于近邻的协同过滤算法是其中一种常用的算法类型，也被称为基于记忆的算法。其基于的一个事实是相似的用户会有相似的行为模式进行物品评分，同时相似的物品也往往能够获得相似的评分。基于近邻的协同过滤算法可以分为基于用户和基于物品两种类型。基于用户的协同过滤算法是一种基于用户行为模式进行推荐的算法，通过寻找与目标用户行为模式相似的其他用户，进行预测或推荐。而基于物品的协同过滤算法则是一种基于物品间相似度进行推荐的算法，通过寻找与目标物品相似的其他物品，进行预测或推荐。

在协同过滤算法中，用户评分和物品评分是其关键概念之一。用户评分指用户对于物品的打分或评价，是协同过滤算法进行推荐的基础。而物品评分则是用户对于物品的打分或评价，是基于物品的协同过滤算法进行推荐的基础。此外，相似度也是协同过滤算法中的一个重要概念，用于衡量用户或物品之间的相似程度。行为模式则指用户在评价物品时所表现出来的一系列行为和偏好，是基于用户的协同过滤算法的基础。

基于用户的协同过滤算法主要的原理是首先计算某一个用户与其他用户之间的相似度，然后选取与该用户最相似的几个其他用户，将这些用户购买过或感兴趣的物品推荐给该用户。举个例子，假设用户A购买了物品a、b、c，用户B购买了物品d，用户C购买了物品b、c、e。通过比较用户A、B、C的购买记录，可以发现用户A与用户C都购买了物品b和c，而用户B只购买了物品d。因此，可以判定用户A与C两个用户具有较高的相似性。在对用户C进行推荐时，可以将用户A购买过的物品a推荐给它。

表1 用户购买记录

|  |  |
| --- | --- |
| 用户 | 购买记录 |
| A | 物品a、b、c |
| B | 物品d |
| C | 物品b、c、e |

基于物品的协同过滤算法与基于用户的协同过滤算法类似，不同的是基于物品的协同过滤算法利用的是物品之间的相似度而不是用户之间的相似度。其推荐思想是：某个用户购买过物品a或对其感兴趣，则会为其推荐与物品a相似的其他物品。这个思想与基于内容的推荐算法类似，但是基于物品的协同过滤算法对物品间相似度的计算并不是依据物品的内容属性，而是通过分析用户的购买或其他行为记录去计算物品之间的相似度，因为该算法认为物品a和物品b相似是因为喜欢物品a的用户大都也喜欢物品b。举个例子，分析表1的用户购买记录，统计出物品同时被购买的次数，如表2所示，“-”表示没有被同时购买。从表中可以看出物品c与物品b同时被购买的次数最多，所以物品c与物品b最相似。

表2 物品购买共现矩阵

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 物品A | 物品B | 物品C | 物品D | 物品E |
| 物品A | - | 1 | 1 | - | - |
| 物品B | - |  | 2 | - | - |
| 物品C | 1 | 2 | - | - | 1 |
| 物品D | - | - | - | - | - |
| 物品E | - | 1 | 1 | - | - |

基于物品的协同过滤算法是一种推荐算法，其推荐思想是如果某个用户购买过物品a或对其感兴趣，则会为该用户推荐与物品a相似的其他物品。相比于基于用户的协同过滤算法，基于物品的协同过滤算法利用的是物品之间的相似度而不是用户之间的相似度[15]。其计算方法是通过分析用户的购买或其他行为记录来计算物品之间的相似度，因为该算法认为物品a和物品b相似是因为喜欢物品a的用户大都也喜欢物品b。这种思想与基于内容的推荐算法类似，但是基于物品的协同过滤算法不依赖于物品的内容属性进行相似度计算。举个例子，通过分析用户购买记录，可以统计出物品同时被购买的次数，如表2所示，“-”表示没有被同时购买。从表中可以看出物品c与物品b同时被购买的次数最多，因此认为物品c与物品b最相似。

1. 需求分析

概述

该系统的功能可以从宏观上分为两个部分：顾客消费前端和后台管理员后端。顾客消费前端包括了菜品评价、注册登录、修改资料、查看订单、点餐买单、查看菜品、查看红包优惠券、联系前台等。而后台管理员端则包括了登录、管理菜品、管理顾客、管理评价、管理订单、管理角色、管理优惠券以及查看统计信息等。系统用例图如下所示。

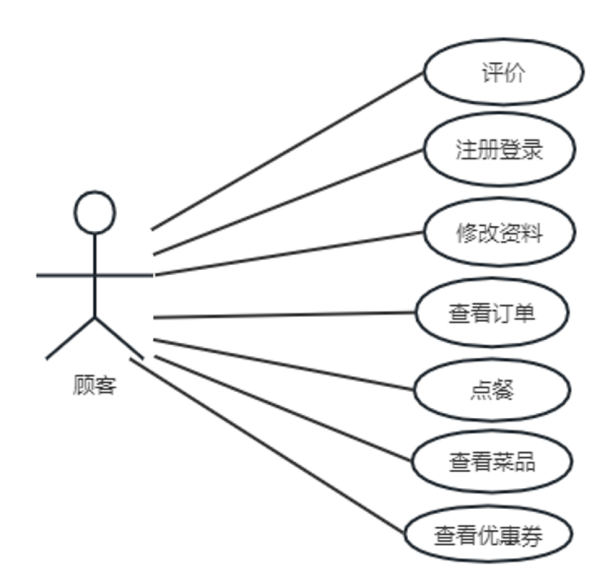


图 5 顾客用例图

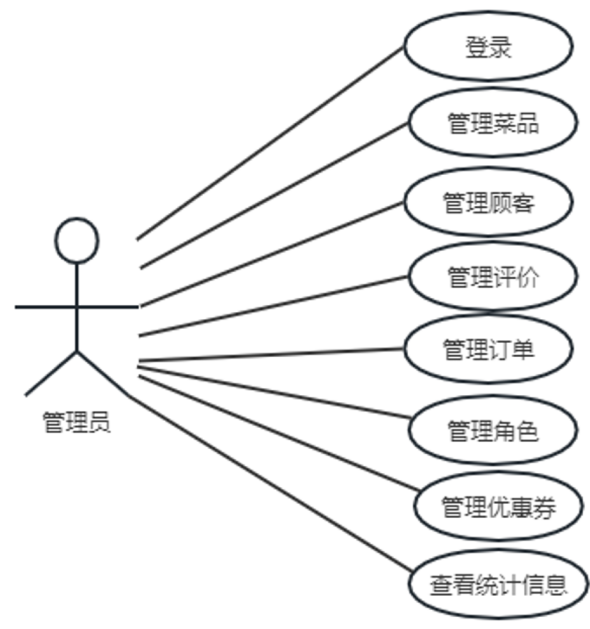


图 6 管理员用例图

功能需求

顾客消费端的详细功能需求包括：

① 扫码选桌

顾客可以通过扫描桌上的二维码选择座位。

② 评价

顾客可以在订单页面，点击未评价的订单进行评价，评价可以进行五星评分、上传图片和文字，还可以查看他人的评价，预览他人的图片等。

③ 查看订单

顾客下单后，可以在订单页面查看此次订单的详细内容，例如下单时间、下单数量、堂食或外卖、下单金额、优惠券等。

④ 修改个人资料

顾客可以修改个人资料以便管理员更好地处理订单。

⑤ 查看和使用优惠券

顾客可以在个人中心中查看自己的优惠券，优惠券界面将展示过期时间、满减条件、优惠详细信息等，还可以使用管理员发放的条码兑换优惠券。在点餐时，可以使用优惠券进行金额抵消。

⑥ 点餐

顾客可以选择堂食或外卖，选择菜品后，可以在购物袋中进行结算，结算时可以对订单进行额外的备注，然后选择支付方式，支付后形成本次订单。

⑦ 查看菜品

顾客可以进入点餐系统，查看分类显示的已上架菜品。点击菜品后，还可以查看详细的规格说明和菜品的用料说明等，并且可以添加进购物袋。

⑧ 注册登录

顾客可以通过输入自身的资料进行注册，从而可以使用注册的资料登录系统并使用系统的功能。

后台管理员端的详细功能需求包括：

① 登录

管理员可以登录后台信息管理系统。

② 管理菜品

管理员可以对菜品进行增删改查等操作。

③ 管理顾客

管理员可以对顾客进行增删改查等操作。

④ 管理评价

管理员可以对评价进行增删改查等操作。

⑤ 管理角色

管理员可以对角色进行增删改查等操作，并在登录时对角色进行辨别，赋予不同的使用本系统权限。

⑥ 管理优惠券

管理员可以对优惠券进行增删改查等操作，还可以批量生成优惠券兑换码等。

⑦ 查看统计信息

管理员可以查看订单等相关数据报表。

⑧ 管理订单

管理员可以对订单进行增删改查等操作，并可以对订单的支付状态或制作状态进行修改。

⑨ 管理首页

管理员可以对首页图片进行增删改查等操作。

系统架构设计

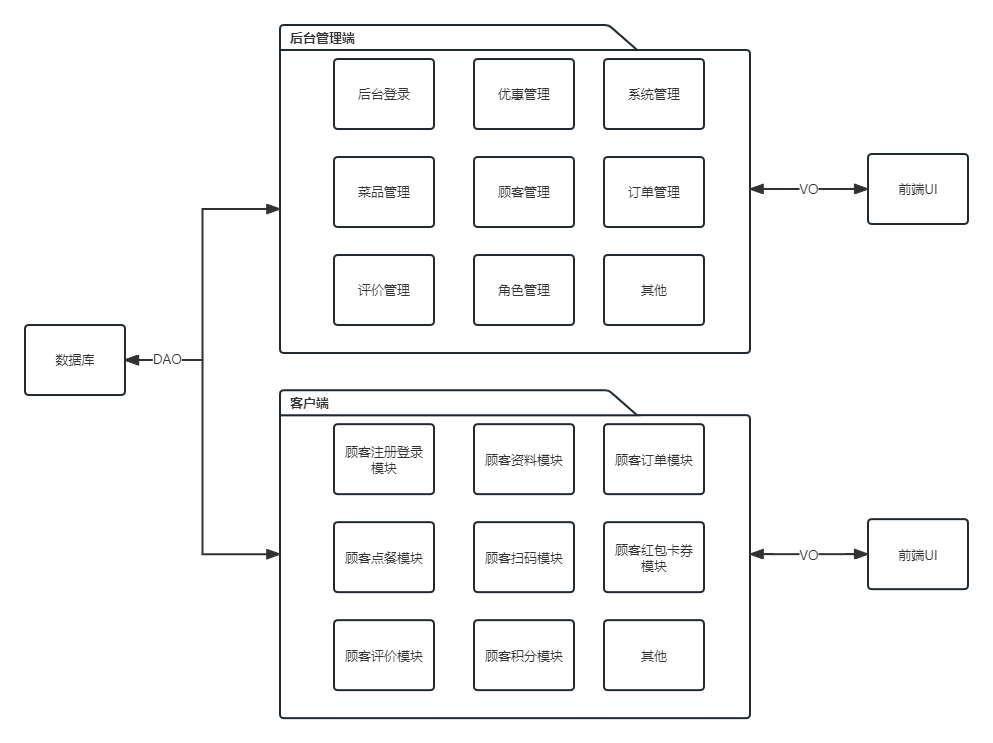


图 7 点餐系统开发设计图

如上图所示，整个系统以点餐功能为基础进行设计，采用了B/S的模式，并且将系统分为了三层，包含交互层，业务层以及数据层。交互层主要是提供用户可视化操作接口，让用户更加便捷的使用服务。业务层主要负责处理交互层中用户传来的操作命令信息并接收其所传来的各种数据，根据不同的操作命令执行不同的业务流程，并将所获得的业务结果返回给交互层。数据层主要是给业务层提供数据持久化的功能，业务层可以对数据库的数据进行增删改查的操作[7]。业务层和前端UI还采用了MVC架构，M（model）指一个存取数据的对象，C（controller）指的是处理数据的一个控制器，V（view）则指的是前端UI（User Interface），即数据的可视化。通常，前后端进行数据交互时，将数据序列化，也就是将其采用JSON的交互格式进行互传。在本系统中，将业务层分成了后台管理端和客户端两方面，减少两者的耦合度[12]。

系统功能模块设计

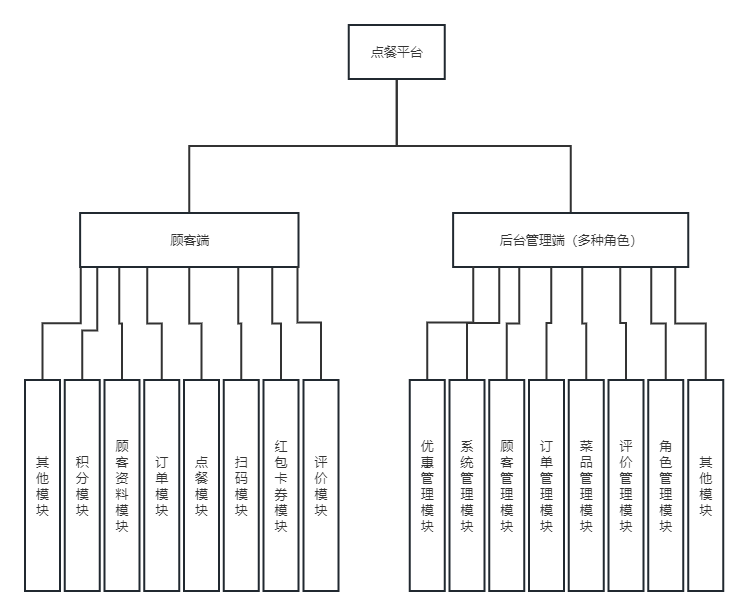


图 8 点餐系统功能模块设计图

根据上图所示，点餐系统包含顾客端和后台管理端两部分。顾客端主要为点餐顾客提供点餐、订餐等服务的平台，而后台管理端则是管理点餐系统内部信息的平台。使用者在登录时，根据其所属角色来提供不同的系统功能。

后台管理端包含：

1. 优惠管理模块：所属角色可以对红包卡券相关信息进行相应权限的操作（增删改查）。
2. 订单管理模块：所属角色可以对订单进行相应权限的操作。
3. 菜品管理模块：所属角色可以对菜品进行相应权限的操作。
4. 顾客管理模块：所属角色可以对顾客资料进行相应权限的操作。
5. 评价管理模块：所属角色可以对菜品评价进行相应权限的操作。
6. 系统管理模块：所属角色可以对系统其他信息进行相应权限的操作。
7. 角色模块：所属角色可以对用户赋予角色，使得其拥有对应的权限操作。
8. 联系顾客模块：所属角色可以跟用户进行聊天，从而解决前台顾客端在其联系客服模块里所发送的问题。。
9. 其他模块：例如可以获得今天的销量，又或是一些数据的可视化展示。

顾客端包含：

1. 顾客资料模块：顾客可以对其自身资料进行修改，例如：姓名、性别、密码等。
2. 点餐模块：顾客可以在此查看菜品，并可以选择合适的规格套餐等，进行下单点餐。
3. 订单模块：顾客在点餐后可以在此查看自己的订单详细，例如：金额、数量、菜品名等。
4. 扫码模块：顾客可以扫码下座。
5. 红包卡券模块：顾客可以查看自己的红包卡券信息，并且可以在下单时使用红包卡券进行金额抵扣。
6. 评价模块：顾客可以查看他人的评价，并且还可以对自己以买过的食品进行评分、评价。
7. 积分模块：顾客可以看到自己的积分，下单时可以获取积分。
8. 联系客服模块：顾客可以联系到后台的客服、服务员，从而可以对其进行必要的催单、问答，而服务员或者客服也可以在其自身后台管理系统中的聊天模块进行聊天。
9. 其他模块

数据库设计

* + 1. ER图设计

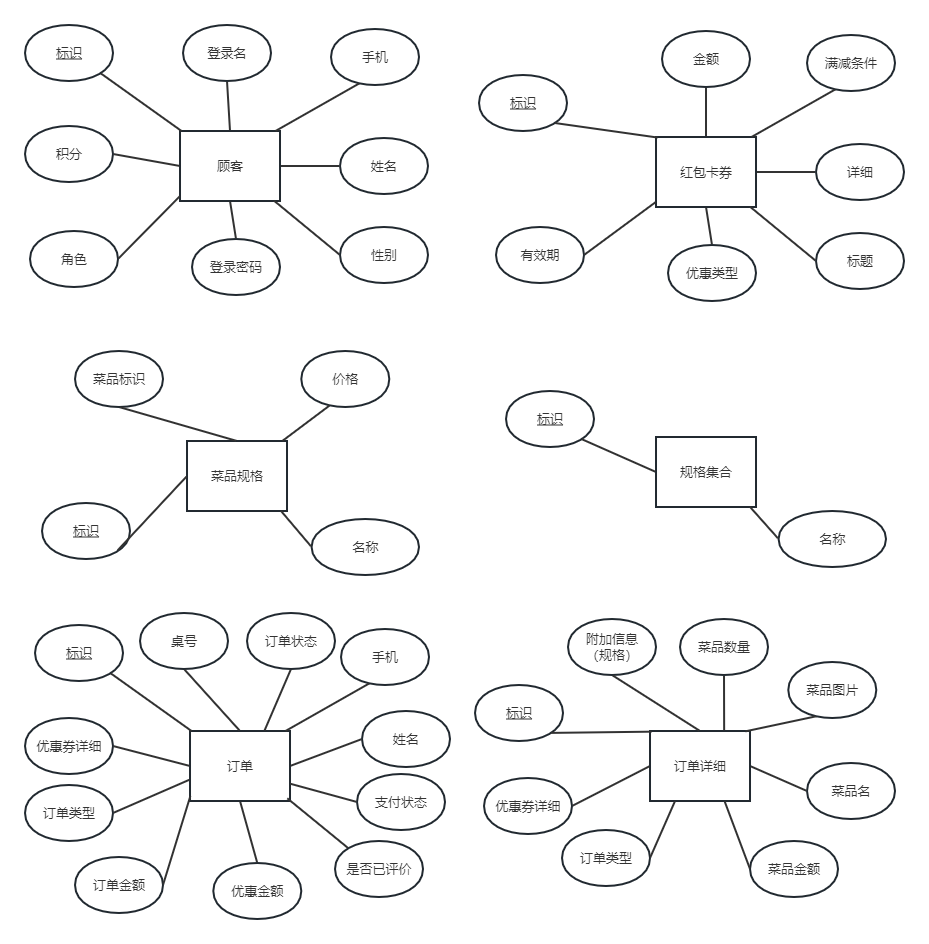


图 9 点餐系统主要实体图

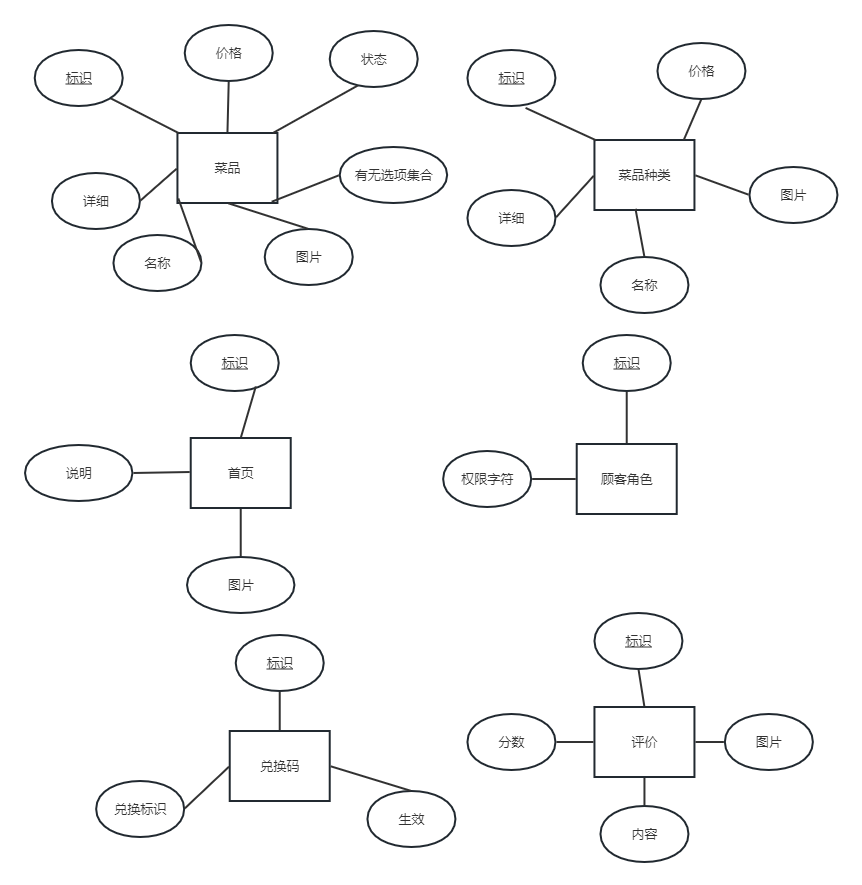


图 10 点餐系统主要实体图

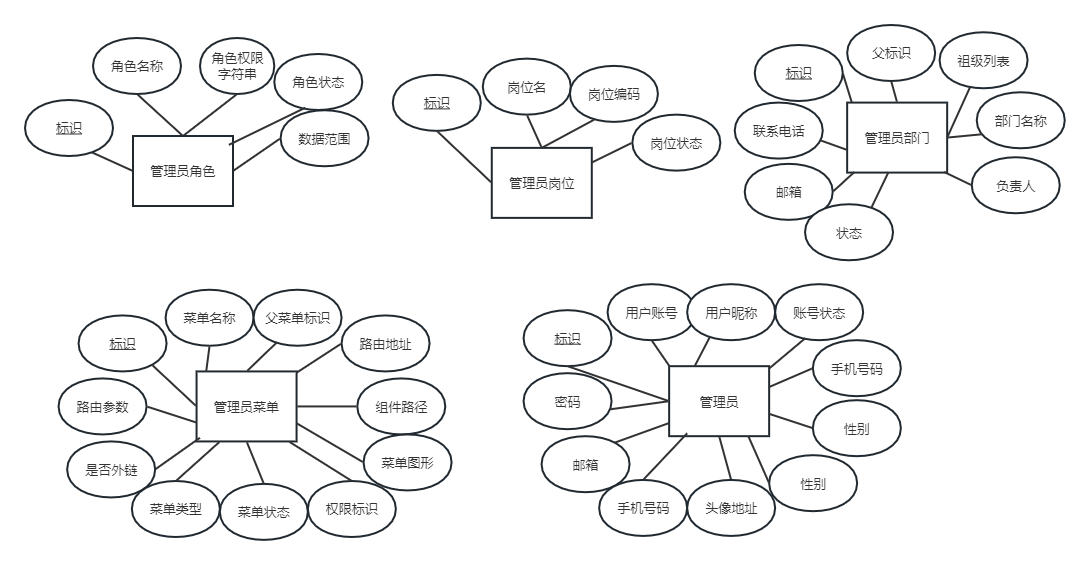


图 11 点餐系统主要实体图

如图所示，该系统的实体应包含顾客、顾客角色、兑换码、红包卡券、评价、订单、订单详细、管理员、管理员角色、管理员岗位、管理员部门、菜品、菜品种类、菜品规格、规格集合等。

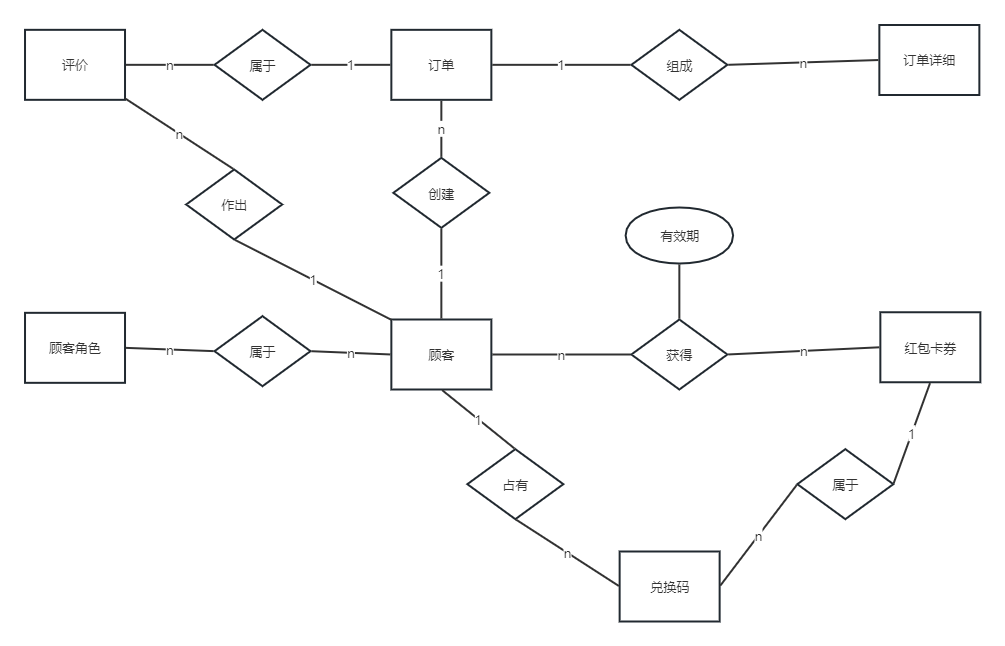


图 12 点餐系统主要ER图

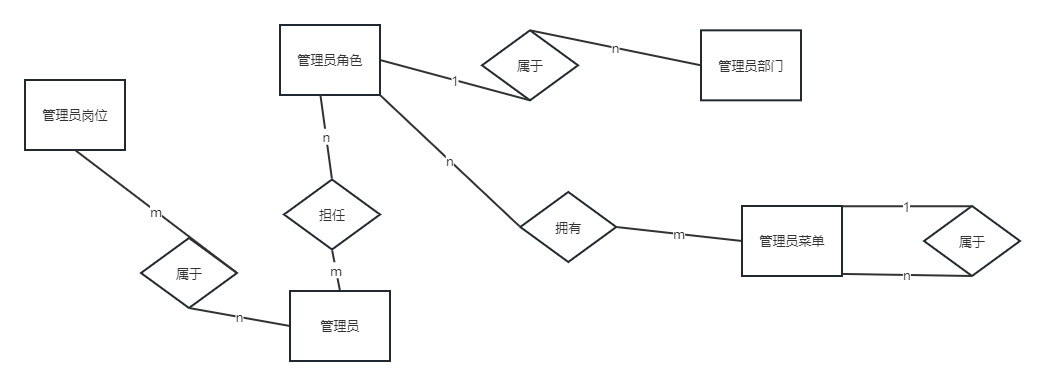


图 13 点餐系统主要ER图

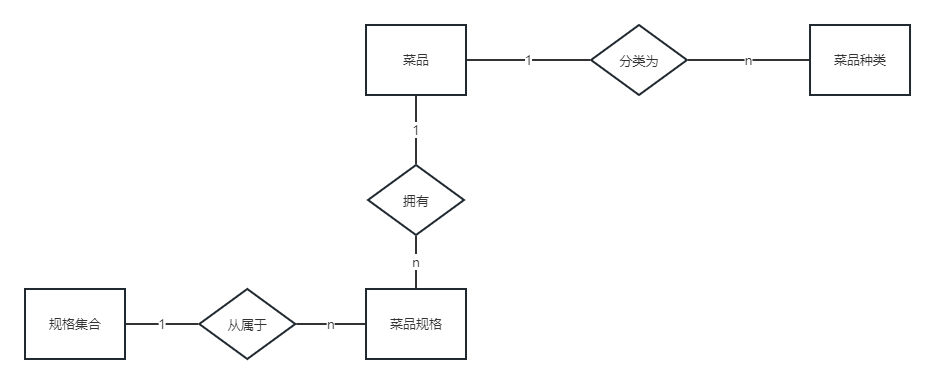


图 14 点餐系统主要ER图

如图所示，ER图是从现实世界的系统需求分析中抽象成计算机语言的一种关系图形。ER图表示方法展示了各种实体之间不同的关系，其中实体之间多对多的联系包括顾客角色和顾客、顾客和红包卡券、管理员角色和管理员岗位、管理员角色和管理员；实体之间一对多的联系包括顾客和兑换码、兑换码和红包卡券、顾客和评价、订单和顾客、订单和订单详细，管理员角色和管理员部门、管理员部门和管理员岗位、菜品和菜品种类、菜品和菜品规格、规格集合和菜品规格。通过实体之间多对多、一对一、一对多关系生成后续物理模型中必要的中间表以及实体对应的表。

* + 1. 数据库表设计

本系统的数据库的实际的物理存储设计采用MYSQL8.2版本的数据库管理系统，并且通过本文上述的ER图、本饭店点餐系统的需求、实际的业务逻辑需求以及在运行调试中所需要修改的要求生成了如下的数据库表。

表3 顾客红包卡券表 (customer\_discount)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| cd\_id | int | - | NO | 唯一标识 |
| customer\_id | int | - | YES | 顾客标识 |
| discount\_id | int | - | YES | 红包标识 |
| validate | datetime | - | YES | 有效期 |

表4 顾客表 (customer\_info)

| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| --- | --- | --- | --- | --- |
| customer\_id | int | - | NO | 顾客标识 |
| customer\_addr | varchar | 255 | YES | 顾客地址 |
| customer\_img | varchar | 255 | YES | 顾客头像 |
| customer\_name | varchar | 255 | NO | 顾客姓名 |
| customer\_passwd | varchar | 255 | YES | 顾客密码 |
| customer\_phone | varchar | 255 | YES | 顾客手机 |
| customer\_score | int | - | YES | 顾客积分 |
| customer\_sex | char | 1 | YES | 性别 |
| customer\_type | int | - | YES | 顾客角色 |
| del\_flag | char | 1 | YES | 逻辑删除 |
| id\_no | varchar | 255 | YES | 顾客身份号 |
| nickname | varchar | 255 | YES | 顾客昵称 |
| update\_by | varchar | 255 | YES | 更新者 |
| update\_time | datetime | - | YES | 更新时间 |
| create\_by | varchar | 255 | YES | 创建者 |
| create\_time | datetime | - | YES | 创建时间 |

表5 顾客所属权限表 (customer\_info\_perms)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| info\_id | int | - | NO | 顾客标识 |
| perms\_id | int | - | NO | 权限标识 |

表6 顾客权限表 (customer\_perms)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| id | bigint | - | NO | 唯一标识 |
| perms | varchar | 100 | YES | 权限标识 |
| remark | varchar | 500 | YES | 备注 |
| create\_by | bigint | - | YES | 创建者 |
| create\_time | datetime | - | YES | 创建时间 |
| update\_by | bigint | - | YES | 更新者 |
| update\_time | datetime | - | YES | 更新时间 |

表7 红包卡券表 (discount\_info)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| discount\_id | int | - | NO | 红包卡券标识 |
| create\_by | varchar | 255 | YES | 更新者 |
| create\_time | datetime | - | YES | 创建时间 |
| discount\_account | decimal | - | YES | 红包卡券数额 |
| discount\_condition | decimal | - | YES | 红包满减条件 |
| discount\_detail | varchar | 255 | YES | 红包卡券详细 |
| discount\_title | varchar | 255 | YES | 红包卡券标题 |
| discount\_type | char | 2 | YES | 红包卡券类型 |
| isValidate | char | 2 | YES | 红包是否生效 |
| update\_by | varchar | 255 | YES | 更新时间 |
| update\_time | datetime | - | YES | 更新时间 |

表8 菜品表 (food)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| create\_by | varchar | 255 | YES | 创建者 |
| create\_time | datetime | - | YES | 创建时间 |
| food\_detail | text | 65535 | YES | 菜品详细 |
| food\_id | int | - | NO | 菜品标识 |
| food\_img | text | 65535 | YES | 菜品图片 |
| food\_name | varchar | 255 | NO | 菜品名 |
| food\_price | decimal | - | YES | 菜品价格 |
| food\_status | char | 1 | YES | 菜品状态 |
| food\_type | int | - | YES | 菜品所属类型 |
| is\_single | char | 2 | YES | 选项集合 |
| update\_by | varchar | 255 | YES | 更新者 |
| update\_time | datetime | - | YES | 更新时间 |

表9 菜品规格集合表 (food\_collect)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| collect\_id | int | - | NO | 集合标识 |
| collect\_name | varchar | 255 | YES | 集合名称 |
| create\_by | varchar | 255 | YES | 创建者 |
| create\_time | datetime | - | YES | 创建时间 |
| update\_by | varchar | 255 | YES | 更新者 |
| update\_time | datetime | - | YES | 更新时间 |

表10 评分表 (food\_score)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| score\_id | int | - | NO | 评分标识 |
| create\_by | varchar | 255 | YES | 创建者 |
| create\_time | datetime | - | YES | 创建时间 |
| customer\_id | int | - | YES | 顾客标识 |
| order\_id | varchar | 255 | YES | 订单标识 |
| score | char | 2 | YES | 菜品评分 |
| score\_detail | varchar | 255 | YES | 评价 |
| score\_img | text | 65535 | YES | 评分上传图片 |
| update\_by | varchar | 255 | YES | 更新者 |
| update\_time | datetime | - | YES | 更新时间 |

表11 菜品规格所属表 (food\_sepcification)

| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| --- | --- | --- | --- | --- |
| collect\_id | int | - | YES | 规格所属集合 |
| create\_by | varchar | 255 | YES | 创建者 |
| create\_time | datetime | - | YES | 创建时间 |
| food\_id | int | - | YES | 菜品名称 |
| spcf\_id | int | - | NO | 规格标识 |
| spcf\_name | varchar | 255 | YES | 规格名 |
| spcf\_price | decimal | - | YES | 规格价格(增量) |
| update\_by | varchar | 255 | YES | 更新者 |
| update\_time | datetime | - | YES | 更新时间 |

表12 菜品类型表 (food\_sepcification)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| foodtype\_id | int | - | NO | 菜品类型标识 |
| create\_by | varchar | 255 | YES | 创建者 |
| create\_time | datetime | - | YES | 创建时间 |
| foddtype\_img | text | 65535 | YES | 菜品类型图片 |
| foodtype\_detail | text | 65535 | YES | 菜品类型详细 |
| foodtype\_name | varchar | 255 | YES | 菜品类型名 |
| update\_by | varchar | 255 | YES | 更新者 |
| update\_time | datetime | - | YES | 更新时间 |

表13 图片展示表 (index\_pic)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| index\_id | int | - | NO | 首页图片标识 |
| create\_by | varchar | 255 | YES | 创建者 |
| create\_time | datetime | - | YES | 创建时间 |
| index\_detail | text | 65535 | YES | 首页图片备注 |
| index\_img | text | 65535 | YES | 首页图片 |
| index\_status | char | 1 | YES | 首页图片状态 |
| update\_by | varchar | 255 | YES | 更新者 |
| update\_time | datetime | - | YES | 更新时间 |

表14 订单详细表 (order\_detail)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| detail\_id | int | - | NO | 订单详情标识 |
| create\_by | varchar | 255 | YES | 创建者 |
| create\_time | timestamp | - | NO | 创建时间 |
| food\_additional | text | 65535 | YES | 规格信息 |
| food\_count | int | - | YES | 菜品数目 |
| food\_id | int | - | YES | 菜品标识 |
| food\_img | text | 65535 | YES | 菜品图片 |
| food\_name | varchar | 255 | YES | 菜品名 |
| food\_price | decimal | - | YES | 菜品金额 |
| order\_id | int | - | YES | 订单主表标识 |
| update\_by | varchar | 255 | YES | 更新者 |
| update\_time | timestamp | - | NO | 更新时间 |

表15 订单表 (order\_info)

| **字段名** | **字段类型** | **字段大小** | **是否可空** | **字段备注** |
| --- | --- | --- | --- | --- |
| order\_id | int | - | NO | 订单主表标识 |
| chair\_no | int | - | YES | 桌号 |
| create\_by | varchar | 255 | YES | 创建者 |
| create\_time | timestamp | - | NO | 创建时间 |
| customer\_addr | varchar | 255 | YES | 顾客地址 |
| customer\_id | int | - | YES | 顾客标识 |
| customer\_name | varchar | 255 | YES | 顾客名称 |
| customer\_phone | varchar | 255 | YES | 手机 |
| discount\_amount | varchar | 255 | YES | 优惠券金额 |
| discount\_detail | varchar | 255 | YES | 优惠券消费 |
| discount\_id | int | - | YES | 优惠券编号 |
| isEval | char | 2 | YES | 是否已评价 |
| order\_amount | decimal | - | YES | 订单金额 |
| order\_status | char | 1 | YES | 订单状态 |
| order\_type | char | 1 | YES | 订单类型 |
| pay\_status | char | 1 | YES | 支付状态 |
| update\_by | varchar | 255 | YES | 更新者 |
| update\_time | timestamp | - | NO | 更新时间 |

1. 系统关键模块实现

开发工具及环境

开发工具：Hbuilder、Intellij idea、Navicat

开发框架：Springboot、Vue、Uniapp

服务器：Tomcat

数据库：MySQL

缓存数据库（NOSQL）：Redis

环境：windows 10、安卓系统

顾客端主要功能实现

顾客端的前端主要采用基于Vue.js和uni-app前端框架进行开发，顾客端的后端则主要采用基于SpringBoot后端框架进行开发，角色权限管理系统采用了SpringSecurity框架,并对其进行了必要的修改，使得这一框架能更好的适应管理端系统。其主要功能包括登录注册、菜单分类展示、菜品展示、扫码下座、下单点餐、规格选取、优惠减免、支付订单、评分评价、联系客服等。

* + 1. 注册与登录模块的实现

1. 功能描述

如下图所示，进入“注册”界面后，用户可以输入个人信息，包括手机号码、登录用户名、登录密码、姓名和性别。前端会对数据进行校验，然后与后端进行信息的验证。若验证通过，则会跳转到“登录”界面。在“登录”界面输入正确的用户名和登录密码，前端会再次进行比对，后端也会对比用户输入的信息，如果验证通过，用户就可以进入“我的”界面。

图形用户界面, 文本, 应用程序

描述已自动生成图形用户界面, 文本, 应用程序

描述已自动生成图片包含 文本

描述已自动生成

图 15 点餐平台注册、登录以及登录后的界面

1. 模块设计

如图所示，使用者通过Post方式将登录信息传递到后端控制层的LoginController。控制层调用业务层中的LoginService的login方法，并通过SpringSecurity提供的AuthenticationManager.authenticate方法进行用户名和密码的验证。验证通过后，认证流程继续，到达UserDetailService，再通过Dao层的CustomerMapper的selectOne方法获取用户信息和PermsMapper的selectPermsByCustomerId方法获取用户权限。最终，将结果传递回LoginController，最终以Json格式数据将数据传递给前端。

图表, 箱线图

描述已自动生成

图 16 登录时序图

如图所示，使用者在前端界面填写注册信息后，通过Post请求将信息发送到后端控制层的LoginController中。LoginController通过调用业务层中的IcustomerInfoService的registerCustomer方法，将用户信息存储到数据库中。在存储用户信息时，采用Spring Security提供的PasswordEncoder对用户密码进行加密，确保用户信息的安全性。最后，将处理结果通过Json格式数据返回给前端。

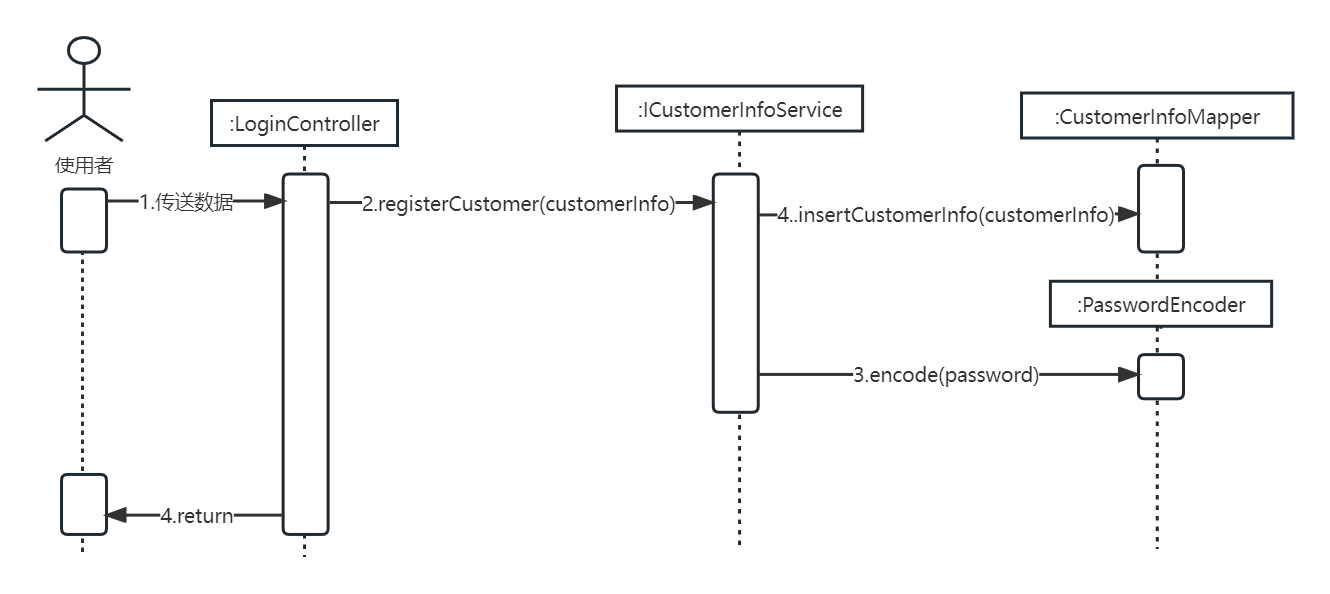


图 17 注册时序图

1. 相关代码

“注册”界面进行数据校验以及前端将信息传送给后端的主要代码如下所示：

save() {

if (!this.validate('phone')) return;

if (!this.validate('username')) return;

if (!this.validate('nickname')) return;

// if(!this.validate('code')) return;

if (!this.validate("password")) return;

uni.showLoading({

title: '注册中',

mask: true

});

this.$request({

url: '/anms/reg',

data: {

customerPhone: this.phone,

nickname: this.nickname,

customerName: this.username,

customerPasswd: this.password,

customerSex: this.sex

},

method: 'post'

}).then(res => {

setTimeout(function() {

uni.hideLoading();

}, 100);

console.log(res);

if (res.code == 200) {

uni.showToast({

title: '注册成功'

})

setTimeout(function() {

uni.navigateTo({

url: '/pages/login/login'

})

}, 1000);

} else {

uni.showToast({

title: res.data,

icon: 'none'

})

}

}).catch(res => {

uni.showToast({

title: "服务器出错",

icon: 'none'

})

})

}

注册界面后端控制器的主要代码如下：

@RestController

@RequestMapping({"user","anms"})

public class LoginController {

@Autowired

ILoginService ILoginService;

@Autowired

private ICustomerInfoService customerService;

@PostMapping("login")

public R login( Customer customer){

return ILoginService.login(customer);

}

@PostMapping("reg")

public R regCustomer(CustomerInfo customerInfo){

return customerService.registerCustomer(customerInfo);

}

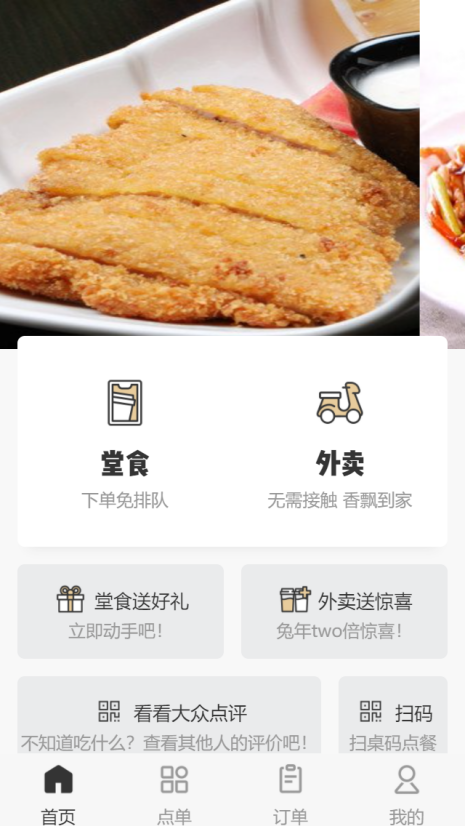
}

* + 1. 菜品展示以及点餐模块的实现

1. 功能描述

如上图所示，该流程大致为：首先点击“堂食”或“外卖”按钮，进入“点单”界面。如果选择“堂食”，则会弹出选择桌号的界面，需要选择当前下单的桌号。接着进入“菜品”选择界面，选择所喜欢的菜品，点击“选规格”，可以选择自己所喜欢的规格，例如：“黑椒汁”、“很热”等规格。之后，点击“加入购物袋”，就可以在下面的购物袋里面看到所添加的菜品。接着点击“结算”，将跳转到“订单结算”界面，接着点击“支付”，最后跳转到“订单界面”。

在“首页”界面中，顾客可以根据自身需求选择“堂食”或者“外卖”。在“点单”界面中，如果之前选择了“堂食”，则可以选择当前所处的桌号，告知对方具体上菜位置；如果选择“外卖”，则会弹出“收货地址”的界面，可以选择、修改或删除自身的地址，这里设置默认地址的时候，不会重复设置多个默认地址，假如当前即将设置的地址是顾客的默认地址，那么插入后，后端会自动将其他的一些地址全部设置为非默认地址。除此之外，可选规格的菜品可以进行“选规格”，不可选的则可点击“+”直接添加到购物袋中。在“选规格”界面中，可以看到“产品描述”以及其对应的规格，可以根据顾客自身需求选择所需的规格，然后点击“加入购物袋”即可加入到购物袋中。在“购物袋”界面，可以看到刚刚添加的菜品，可以通过点击“+”或“-”来调整数量，也可以通过“清空购物车”来删除所有已添加的菜品。点击“结算”可以跳转到“订单结算”界面。在“订单结算”界面中，可以看到“商品列表”、“备注”、“优惠券”、“合计”等一系列信息。点击“支付”后，会跳转到“订单”列表界面。

图形用户界面, 应用程序, PowerPoint

描述已自动生成

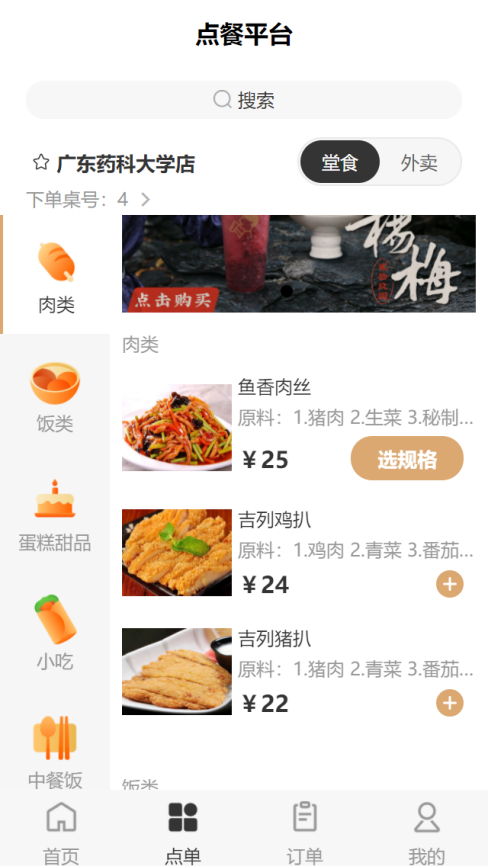
图 18 点餐模块图

图片包含 图形用户界面

描述已自动生成图形用户界面, 文本, 应用程序

描述已自动生成

图 19 点餐模块图

图形用户界面

描述已自动生成

图 20 点餐模块图

 图形用户界面, 文本, 应用程序

描述已自动生成图形用户界面, 应用程序

描述已自动生成

图 21 点餐模块图

1. 模块设计

当用户在前端通过get请求进入控制层FoodController后，FoodController会调用业务层FoodTypeService中的getFoodTypeAndFood方法，该方法会进入Dao层的FoodTypeMapper、FoodMapper、FoodSpecificationMapper和FoodCollectMapper，分别调用SelectFoodTypeList、selectFoodList、selectFoodSpecificationList和selectFoodCollectByCollectId方法，以获取菜品类型、菜品和规格集合信息。获取完这些信息后，Dao层将结果返回给业务层FoodTypeService，再由业务层将结果返回给控制层FoodController。最后，FoodController将结果通过Json格式返回给前端。

图形用户界面, 文本

描述已自动生成

图 22 菜品展示时序图

使用者在前端通过POST请求将购物车数据传递给控制层CartController，随后调用业务层CartService中的postCart方法。该方法进一步调用Dao层的OrderInfoMapper、FoodMapper、FoodSpecificationMapper、OrderDetailMapper、CustomerDiscountMapper和OrderInfoMapper，分别使用insertOrderInfo、selectFoodByFoodId、selectFoodSpecificationBySpcId、insertOrderDetail、selectCustomerDiscountByCdId、selectDiscountInfoByCondition和updateOrderInfo方法来插入订单、查询菜品信息和规格信息、计算菜品数量和金额、插入订单详细信息、查询优惠券信息、计算订单总金额并更新订单信息。最终结果由CartController返回给前端，并以JSON格式返回，以便前端进行必要的渲染。

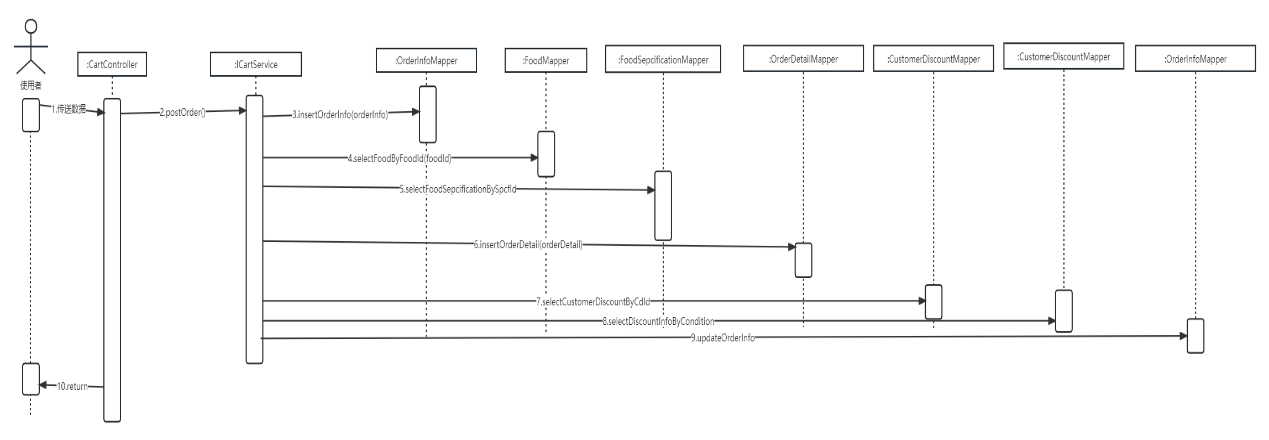


图 23 购物车时序图

1. 相关代码

前端对后端传送过来的菜品数据进行展示的主要代码如下：

getFoods() {

this.$request({

url: "/food/allfood"

}).then(res => {

if (res.code == 200) {

let arr = res.data;

this.categories = arr;

this.$nextTick(() => this.calcSize());

this.newCommend();

}

})}

前端对后端传送订单以及支付数据的主要代码如下：

handlePay() {

let obj = {

cdId: this.cdId,

orderType: this.orderType,

chairNo: this.chairNo,

foods: []

}

if (this.orderType == 1 && this.address != null) {

this.$set(obj, "customerPhone", this.address.phone);

this.$set(obj, "customerName", this.address.name);

this.$set(obj, "customerAddr", this.address.complete\_address + this.address.address);

}

this.cart.forEach((item, index) => {

let temp = {

foodId: this.cart[index].id,

foodNumber: this.cart[index].number,

speciIds: this.cart[index].spciIds,

}

obj.foods.push(temp);

})

console.log(obj)

this.$request({

url: "/cart/postcart",

data: obj,

method: "post",

contentType: "application/json",

}).then(res => {

console.log(res);

if (res.code == 200) {

uni.removeStorageSync('orderType');

uni.removeStorageSync('cart');

uni.showToast({

icon: 'success',

title: '操作成功'

})

uni.switchTab({

url: "/pages/order/order"

})

}

})

}

后端对订单数据处理的控制器的主要代码如下：

@RestController

@RequestMapping({"cart"})

public class CartController {

@Autowired

private ICartService cartService;

@PostMapping("postcart")

@PreAuthorize("hasAuthority('customer:info:all')")

public R postCart(@RequestBody Cart cart){

return cartService.postOrder(cart);

}

}

后端将菜品数据展示给前端的控制器的主要代码如下：

@RestController

@RequestMapping({"anms","food"})

public class FoodController {

@Autowired

private IFoodService foodService;

@Autowired

private IFoodTypeService foodTypeService;

@GetMapping("allfood")

public R getFoodAndFoodType(){

return foodTypeService.getFoodTypeAndFood();

}

}

* + 1. 订单模块的实现

1. 功能描述

在“订单”页面中，用户可以查看每次下单的相关信息，包括堂食或外卖方式、订单编号、桌号、下单时间、下单价格等。此外，还提供了“查看评论”、“去评价”、“再来一单”的功能。订单页面还分为“当前订单”和“历史订单”两部分，其中“当前订单”表示正在制作中的订单，“历史订单”则表示已完成或已取消的订单。用户可以点击任意一笔订单，进入“订单详细”页面查看详细信息，包括菜品名称、数量、价格、下单时间、订单编号和合计金额等。

图形用户界面, 应用程序

描述已自动生成 图形用户界面, 应用程序

描述已自动生成 图形用户界面, 文本, 应用程序

描述已自动生成

图 24 订单模块图

1. 模块设计

前端用户通过GET请求传递订单信息给OrderController控制器，然后调用OrderInfoService业务层的getOrderInfoList方法。接下来通过authentication的getAuthentication方法获取已登录顾客标识，然后进入Dao层OrderInfoMapper，调用selectOrderInfoList方法查找对应的订单列表，并将结果返回给OrderController。最终，将查询结果通过JSON格式返回给前端，以便让前端进行必要的渲染。

图表, 图示, 箱线图

描述已自动生成

图 25 订单时序图

使用者在前端通过GET请求传递给OrderDetailController，控制器调用OrderDetailService的getOrderDetailList方法，该方法进一步调用Dao层的OrderDetailMapper的selectOrderDetailList方法，以查找与订单主表标识相关的订单详细信息。查询结果被返回给OrderDetailController，最后以JSON格式返回给前端，以便让前端进行必要的渲染。。

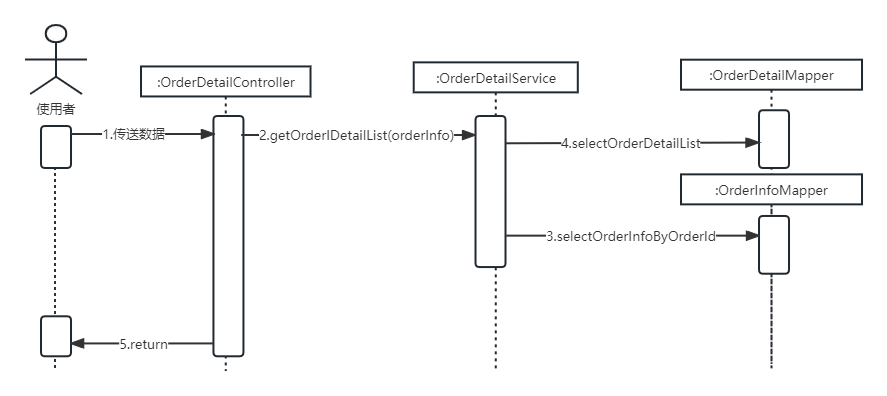


图 26 订单详细时序图

1. 相关代码

前端展示后端传来的订单列表信息的主要代码如下：

getOrders() {

this.$request({

url: "/order/info/list?pageNum=" + this.per

}).then(res => {

if (res.code == 200) {

let arr = res.data;

if (arr.length == 0) {

this.isStart = false;

return;

}

console.log("orders");

console.log(arr);

this.orders.push(...arr);

}

})

}

后端传给前端的订单列表信息的控制器的主要代码如下：

@RequestMapping({"anms","order"})

@RestController

public class OrderController {

@Autowired

private IOrderDetailService orderDetailService;

@Autowired

private IOrderInfoService orderInfoService;

@GetMapping("info/list")

@PreAuthorize("hasAuthority('customer:info:all')")

public R getOrderInfo(OrderInfo orderInfo){

PageHelper.startPage(PageUtil.getPageDomain());

return orderInfoService.getOrderInfoList(orderInfo);

}

@GetMapping("detail/list")

@PreAuthorize("hasAuthority('customer:info:all')")

public R getOrderDetail(OrderDetail orderDetail){

PageHelper.startPage(PageUtil.getPageDomain());

return orderDetailService.getOrderDetail(orderDetail);

}

}

* + 1. 评价模块的实现

1. 功能描述

评价模块在点餐系统中有两个实现方式：第一个是在“首页”界面，点击“看看大众点评”，即可浏览其他顾客的评价信息。第二个是在完成订单后，对菜品进行评价。具体来说，用户可以在“订单历史”页面找到已完成的订单，点击“查看评论”或“去评价”，即可进入评价页面进行评价。在评价页面，用户可以选择评分和填写评论内容，并可以上传照片。完成评价后，评价信息将被提交并保存到系统中。其中，图片存储到服务器的磁盘上，而评论内容以及评分持久化存储到对应的数据库表之中。如下图所示。

图形用户界面, 网站

描述已自动生成图形用户界面, 文本, 应用程序

描述已自动生成

图 27 评价模块图

在首页界面，点击“看看大众点评”按钮，可以进入“大众点评”界面，展示所有顾客的评分、评价以及图片。点击图片可以查看在当前界面查看完整放大的图片，并且还配有“X”的按钮，即客户是可以通过点击该按钮关闭当前的完整放大图片的预览界面。

图形用户界面, 应用程序

描述已自动生成图形用户界面

描述已自动生成图形用户界面, 应用程序

描述已自动生成

图 28 评价模块图

在“订单”界面中，可以点击未完成评价的订单的“去评价”按钮，对该订单进行评分、评价和上传图片，并在完成后点击“发布”按钮，即可成功发布评论。同时，在“订单”界面中，也可以查看已完成评价的订单的“查看评价”按钮，进入“详细评论”界面，查看自己曾经的评分、评价和上传的图片。此外，还可以点击页面底部的按钮，跳转到其他人的评价界面。

1. 模块设计

使用者通过前端的GET请求将对应的请求传递给控制层RateController，RateController接着会调用业务层FoodScoreService中的selectFoodScoreId方法。然后，FoodScoreMapper会进入Dao层并调用selectFoodScoreList方法，在数据库中查找相应的评价列表。查找结果将被返回给RateController，并以JSON格式传回前端，以便让前端进行必要的渲染。。

图示

描述已自动生成

图 29 获取评分列表时序图

使用者在前端通过 POST 请求将评价信息传递给控制层 RateController，接着调用业务层 FoodScoreService 中的 insertFoodScore 方法，之后进入 Dao 层的 FoodScoreMapper，调用 insertFoodScore 方法插入评价信息，将结果返回给 FoodScoreService。接下来将结果设置到 OrderInfoMapper 中的 updateOrderInfo 方法，以更新订单主表，若更新成功，则标记为已评价，否则标记为未评价。最终将结果返回给 RateController，以 JSON 格式返回给前端，让前端进行渲染。

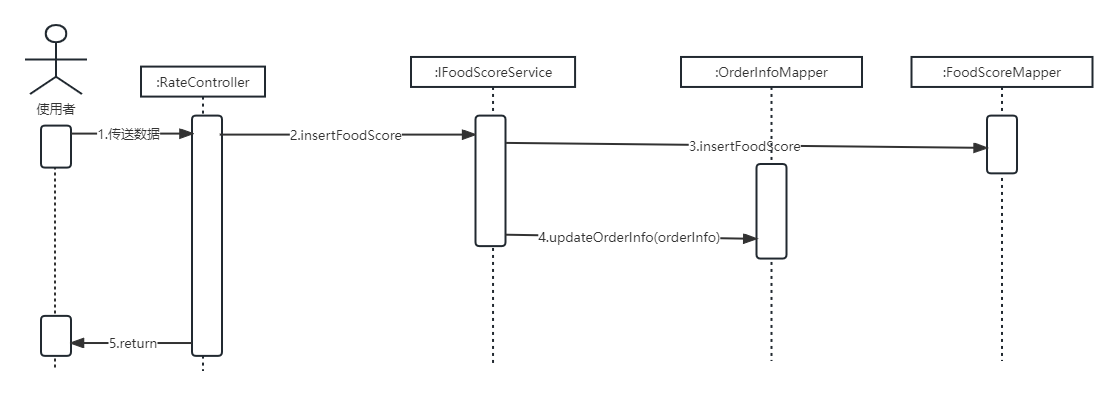


图 30 添加评价时序图

使用者在前端通过get请求将数据传递数据给控制层RateController后，RateController会调用业务层FoodScoreService中的selectAutoCustomerId方法，并且使用authentication的getAuthentication方法获取已登录顾客的标识。接着，控制层将控制流程传递给Dao层的FoodScoreMapper，调用selectFoodScoreList方法来查找对应的评价列表，并将结果返回给RateController。最后，RateController通过Json格式将结果返回给前端，以便让前端进行必要的渲染。

图示

描述已自动生成

图 31 获取顾客评价时序图

1. 相关代码

前端获取后端传来的所有评价信息的主要代码：

getRateList() {

this.$request({

url: "/anms/rateinfo?pageNum="+this.per

}).then(res => {

if (res.code == 200) {

let arr = res.data;

if(arr==null||arr.length==0){

this.isStart=false;

return;

}

console.log(res);

this.ratelist.push(...arr);

}

})

}

前端上传评分、评价、图片内容到后端的主要代码：

handlePost(){

this.$request({

url:"/anms/uprate",

data:{

orderId:this.orderId,

customerId:this.customerId,

score:this.score,

scoreDetail:this.scoreDetail,

scoreImg:this.srclist.join(",")

},

method:"post"

}).then(res=>{

console.log(res);

if(res.code==200){

uni.switchTab({

url:"/pages/order/order"

});

uni.showToast({

icon:'success',

title:'操作成功'

});

}

})

}

后端传送评价相关的数据到前端、以及接受前端传来的评价数据的控制器的主要代码：

@RestController

@RequestMapping({"anms","rate"})

public class RateController {

@Autowired

private IFoodScoreService foodScoreService;

@PostMapping("uprate")

@PreAuthorize("hasAuthority('customer:info:all')")

public R uprate(FoodScore foodScore){

int i = foodScoreService.insertFoodScore(foodScore);

return RU.OK(i);

}

@GetMapping("rateinfo")

public R getInfo(){

PageHelper.startPage(PageUtil.getPageDomain());

List<FoodScore> foodScores = foodScoreService.selectFoodScoreList(null);

return RU.OK(foodScores);

}

@GetMapping("cusrate")

@PreAuthorize("hasAuthority('customer:info:all')")

public R getCusRate(FoodScore foodScore){

List<FoodScore> foodScores = foodScoreService.selectAutoCustomerId(foodScore);

return RU.OK(foodScores);

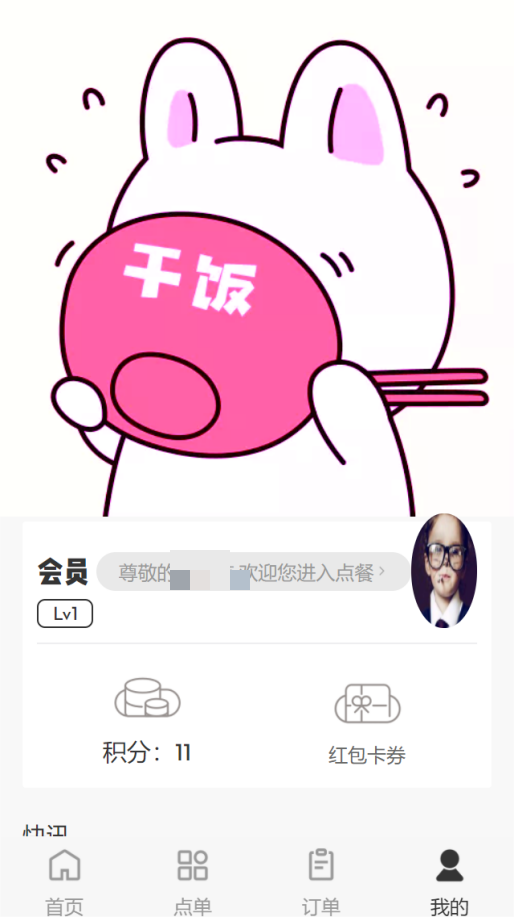
}

}

* + 1. 资料修改模块的实现

1. 模块设计

如图所示，在“我的”界面中，点击头像可以进入“个人资料”的界面，里面可以修改个人资料（姓名、手机号码、性别），点击“保存”即可保存正确的个人信息，若信息不符合给定的规则将会弹出错误的提示框，除此之外，点击“修改密码”，可以进入“修改密码”的界面。在“修改密码”的界面中，可以输入新的密码，可以“查看密码”或“隐藏密码”，点击“保存”即可保存正确的密码，并且弹到“登录”的界面，若两次密码不符合规范，将会弹出错误的提示框。

 图形用户界面, 应用程序

描述已自动生成

图 32 资料修改模块图

图形用户界面, 应用程序

描述已自动生成

图 33 资料修改模块图

1. 功能描述

使用者在前端通过get请求将数据传给控制层CustomerController，接着调用业务层CustomerInfoService中的getCustomer，然后调用authentication的getAuthentication获取已登录的顾客标识，之后进入Dao层的CustomerInfoMapper调用selectCustomerInfoList查找对应的顾客资料，把结果返回给CustomerController，最后通过Json格式返回给前端，以便让前端进行必要的渲染。

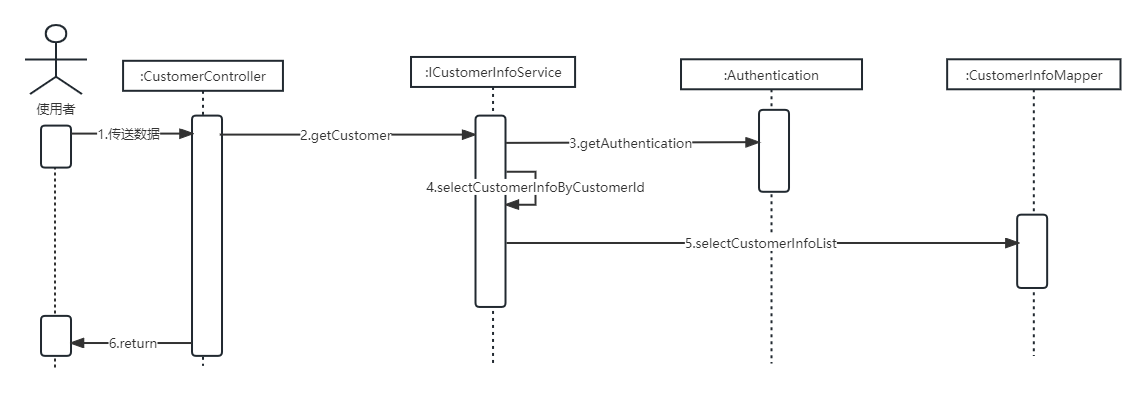


图 34 获取顾客资料时序图

使用者在前端通过post请求将数据传给控制层CustomerController，接着调用业务层CustomerInfoService中的updateCustomer，然后调用authentication的getAuthentication获取已登录的顾客标识，之后进入Dao层的CustomerInfoMapper调用updateCustomer查找对应的顾客资料，把结果返回给CustomerController，最后通过Json格式返回给前端，以便让前端进行必要的渲染。

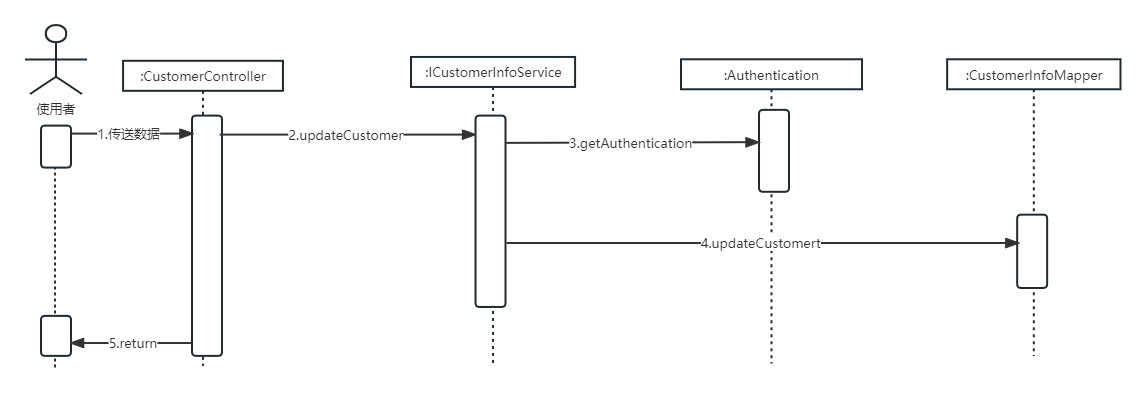


图 35 更新顾客资料时序图

1. 相关代码

前端传送给后端个人信息的主要代码：

updCustomer() {

if(!this.validate('customerPhone')) return;

if(!this.validate('nickname')) return;

this.uploadForm.customerId = this.form.customerId;

this.uploadForm.nickname = this.form.nickname;

this.uploadForm.customerPhone = this.form.customerPhone;

this.uploadForm.customerSex = this.form.customerSex;

uni.showLoading({

title: "保存中..."

})

uni.request({

url: this.$url + "/customer/updinfo",

header: {

token: uni.getStorageSync("token"),

'content-type': 'application/x-www-form-urlencoded'

},

data: this.uploadForm,

method: "POST",

success: res => {

console.log(this.form)

console.log(res)

uni.hideLoading();

if (res.data.code == 200) {

uni.showToast({

title: "保存成功..."

})

} else

uni.showToast({

title: "保存失败..."

})

// console.log(this.swipers);

},

fail: res => {

uni.hideLoading();

uni.showToast({

title: "保存失败..."

})

}

})

｝

前端传送给后端密码的主要代码：

updCustomer() {

if(!this.validate('password')) return;

if(!this.validPass()) return;

this.uploadForm.customerPasswd = this.form.password;

uni.showLoading({

title: "保存中..."

})

uni.request({

url: this.$url + "/customer/updPwd",

header: {

token: uni.getStorageSync("token"),

'content-type': 'application/x-www-form-urlencoded'

},

data: this.uploadForm,

method: "POST",

success: res => {

console.log(this.form)

console.log(res)

uni.hideLoading();

if (res.data.code == 200) {

uni.showToast({

title: "修改成功...",

});

uni.removeStorageSync('token');

uni.navigateTo({

url: '/pages/login/login'

});

} else

uni.showToast({

title: "修改失败..."

});

// console.log(this.swipers);

},

fail: res => {

uni.hideLoading();

uni.showToast({

title: "操作失败..."

});

}

})

}

后端处理前端传送的个人信息以及处理传送的密码的控制器的代码：

@RestController

@RequestMapping({"customer"})

public class CustomerController {

@Autowired

private ICustomerInfoService customerService;

@GetMapping("info")

@PreAuthorize("hasAuthority('customer:info:all')")

public R getCustomerInfo(){

return customerService.getCustomer();

}

@PostMapping("updinfo")

@PreAuthorize("hasAuthority('customer:info:all')")

public R updateCustomer(CustomerInfo customerInfo){

System.out.println(customerInfo);

return RU.OK(customerService.updateCustomerInfo(customerInfo));

}

@PostMapping("updPwd")

@PreAuthorize("hasAuthority('customer:info:all')")

public R updateCustomerPwd(CustomerInfo customerInfo){

return RU.OK(customerService.updateCustomerPwd(customerInfo));

}

}

* + 1. 红包卡券模块的实现

1. 功能描述

红包卡券模块在点餐系统中的两处地方实现：第一处是展示顾客的红包卡券信息，第二处是在下单时使用红包卡券，如下图所示。

图片包含 图形用户界面

描述已自动生成图形用户界面, 应用程序

描述已自动生成图形用户界面, 文本, 应用程序

描述已自动生成

图 36 红包卡券模块图

在“我的”界面中，点击“红包卡券”，即可进入“红包卡券”界面。在或“红包卡券”的界面中，有“可使用”和“已过期”两个分界面。“可使用”界面中包含未过期和未使用的红包卡券（红包卡券类型、红包标题、红包详细、红包金额、红包的截止时间），“已过期”界面中包含已使用和已过期的红包卡券。

图形用户界面, 文本, 应用程序

描述已自动生成图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成图形用户界面, 文本, 应用程序

描述已自动生成

图 37 红包卡券模块图

在“订单结算”界面中，点击“优惠券”，即可进入“优惠减免”界面。在“优惠减免”的界面中，有“不使用红包卡券”和其他所有的符合满减条件的红包卡券，点击其中一项红包卡券即可使用，然后进入“订单结算”界面，即可看到在“优惠券”处看到减免金额。

图形用户界面, 应用程序, 网站

描述已自动生成图形用户界面, 应用程序

描述已自动生成

图 38 红包卡券码兑换图

在“我的”界面中，点击“兑换红包卡券码”，即可跳转到“兑换卡券”页面，点击“兑换”即可完成红包卡券码的兑换。

1. 模块设计

使用者在前端通过get请求将数据传给控制层DiscountController，接着调用业务层CustomerDiscountService中的updateCustomer，然后调用authentication的getAuthentication获取已登录的顾客标识，之后进入Dao层的CustomerInfoMapper调用selectCustomerDiscountList查找对应的顾客卡券，把结果返回给CustomerDiscountController，最后通过Json格式返回给前端，以便让前端渲染。

图示

描述已自动生成

图 39 获取顾客红包卡券时序图

1. 相关代码

前端对于后端传来的红包卡券信息进行处理的主要代码：

getDiscount() {

this.$request({

url: "/discount/validCondition?discountCondition="+this.discountCondition+"&pageNum=" + this.per

}).then(res => {

if (res.code == 200) {

let arr = res.data;

if (arr == null || arr.length == 0) {

this.isStart = false;

return;

}

console.log(res);

this.discount.push(...arr);

}

})

}

后端处理红包卡券信息的主要代码：

@RestController

@RequestMapping({"discount"})

public class DiscountController {

@Autowired

private ICustomerDiscountService customerDiscountService;

@GetMapping("/discountinfo")

@PreAuthorize("hasAuthority('customer:info:all')")

public R getInfo() throws Exception {

PageHelper.startPage(PageUtil.getPageDomain());

List list = customerDiscountService.selectCustomerDiscountList();

return RU.OK(list);

}

@GetMapping("/novalidatediscountinfo")

@PreAuthorize("hasAuthority('customer:info:all')")

public R getNoValidateInfo(){

PageHelper.startPage(PageUtil.getPageDomain());

List list = customerDiscountService.selectNoValidateDiscountList();

return RU.OK(list);

}

@GetMapping("/validCondition")

@PreAuthorize("hasAuthority('customer:info:all')")

public R getValidByCondition(DiscountInfo discountInfo){

PageHelper.startPage(PageUtil.getPageDomain());

List list = customerDiscountService.selectCustomerDiscountList(discountInfo);

return RU.OK(list);

}

}

* + 1. 扫桌号模块的实现

****

图 40 扫桌号模块图

在“首页”界面中，点击“扫码”即可扫桌号就餐，扫完桌号后，弹至“点单”界面，这时可以看到右上角“下单编号”已经显示桌号，无需手动选择。

前端主要实现代码：

handleScan(){

uni.scanCode({

onlyFromCamera: true,

success: (res)=>{

this.scanmain=res.result.deskId;

let temp=JSON.parse(res.result);

if(temp.deskId!=null){

uni.setStorageSync("deskId",temp.deskId);

uni.switchTab({

url:"/pages/index/index"

})

}else{

uni.showToast({

icon:'error',

title:"未发现有关信息"

})

}

}

});

}

后台管理端主要功能实现

* + 1. 菜品管理模块的实现

1. 功能描述

如图，使用者可以对菜品模块进行信息的增删改查等操作。

图形用户界面, 应用程序

描述已自动生成

图 41 菜品管理图

电脑软件的截图

描述已自动生成

图 42 菜品详细模块图



图 43 菜品详细模块图

电脑截图

描述已自动生成

图 44 菜品类型模块图

图形用户界面, 应用程序

描述已自动生成

图 45 菜品类型模块图



图 46 菜品类规格集合图

图形用户界面, 应用程序

描述已自动生成

图 47 菜品规格模块图

1. 模块设计

使用者在前端通过post请求将数据传送给控制层FoodController，接着调用业务层FoodService中的updateFood，之后进入Dao层的FoodMapper调用updateFood更新菜品，把结果返回给FoodController，最后通过Json格式返回修改结果成功或失败给前端，以便让前端进行必要的渲染，而这一些图片则采用了类似异步的方法，可以让前端将图片数据先上传给后端，后端存储到服务器的文件夹后，后端再给前端返回存储后的链接地址，这就保证了图片成功的存储，进而若在客户不删除、不修改的前提下，完成此次添加或更新时，图片的URL就会在此次调用中传回给后端进行链接URL的存储，这样就可以保证用户在使用时更加顺畅。

图示

描述已自动生成

图 48 更新菜品时序图

使用者在前端通过post请求将菜品的一些必要且相关的数据传给后端的控制层FoodTypeController，接着会在这个方法里面更加进一步地调用业务层FoodTypeService中的updateFoodType，在此期间的数据都是采用了Model层的实体类进行了必要的封装，进而将其视为整体进行数据的传送，之后进入Dao层的FoodTypeMapper调用updateFoodType更新菜品，把结果返回给FoodTypeController，最后通过Json格式返回修改结果成功或失败给前端，以便让前端进行必要的渲染。

图示

描述已自动生成

图 49 更新菜品类型时序图

使用者在前端通过post请求将菜品规格的一些必要且相关的数据传给后端的控制层FoodCollectController，接着会在这个方法里面更加进一步地调用业务层FoodCollectService中的updateFoodCollect，在此期间的数据都是采用了Model层的实体类进行了必要的封装，进而将其视为整体进行数据的传送，之后进入Dao层的FoodCollectMapper调用updateFoodCollect更新菜品规格的信息，把最后的结果返回给之前调用的那一层的FoodCollectController，最后通过Json格式返回成功或失败的给前端，以便让前端进行必要的渲染。

图示

描述已自动生成

图 50 更新规格集合时序图

1. 相关代码

前端主要代码：

getList() {

this.loading = true;

listFood(this.queryParams).then(response => {

this.foodList = response.rows;

this.total = response.total;

this.loading = false;

});

}

后端主要代码：

@RestController

@RequestMapping("/food/food")

public class FoodController extends BaseController

{

@Autowired

private IFoodService foodService;

@PreAuthorize("@ss.hasPermi('food:food:list')")

@GetMapping("/list")

public TableDataInfo list(Food food)

{

startPage();

List<Food> list = foodService.selectFoodList(food);

return getDataTable(list);

}

@PreAuthorize("@ss.hasPermi('food:food:export')")

@Log(title = "菜品测试管理", businessType = BusinessType.EXPORT)

@PostMapping("/export")

public void export(HttpServletResponse response, Food food)

{

List<Food> list = foodService.selectFoodList(food);

ExcelUtil<Food> util = new ExcelUtil<Food>(Food.class);

util.exportExcel(response, list, "菜品测试管理数据");

}

@PreAuthorize("@ss.hasPermi('food:food:query')")

@GetMapping(value = "/{foodId}")

public AjaxResult getInfo(@PathVariable("foodId") Long foodId)

{

return success(foodService.selectFoodByFoodId(foodId));

}

@PreAuthorize("@ss.hasPermi('food:food:add')")

@Log(title = "菜品测试管理", businessType = BusinessType.INSERT)

@PostMapping

public AjaxResult add(@RequestBody Food food)

{

return toAjax(foodService.insertFood(food));

}

@PreAuthorize("@ss.hasPermi('food:food:edit')")

@Log(title = "菜品测试管理", businessType = BusinessType.UPDATE)

@PutMapping

public AjaxResult edit(@RequestBody Food food)

{

return toAjax(foodService.updateFood(food));

}

@PreAuthorize("@ss.hasPermi('food:food:remove')")

@Log(title = "菜品测试管理", businessType = BusinessType.DELETE)

@DeleteMapping("/{foodIds}")

public AjaxResult remove(@PathVariable Long[] foodIds)

{

return toAjax(foodService.deleteFoodByFoodIds(foodIds));

}

}

获取菜品类型前端主要代码：

getList() {

this.loading = true;

listType(this.queryParams).then(response => {

this.typeList = response.rows;

this.total = response.total;

this.loading = false;

});

}

后端主要代码：

@RestController

@RequestMapping("/food/type")

public class FoodTypeController extends BaseController

{

@Autowired

private IFoodTypeService foodTypeService;

@PreAuthorize("@ss.hasPermi('food:type:list')")

@GetMapping("/list")

public TableDataInfo list(FoodType foodType)

{

startPage();

List<FoodType> list = foodTypeService.selectFoodTypeList(foodType);

return getDataTable(list);

}

@PreAuthorize("@ss.hasPermi('food:type:export')")

@Log(title = "菜品类型", businessType = BusinessType.EXPORT)

@PostMapping("/export")

public void export(HttpServletResponse response, FoodType foodType)

{

List<FoodType> list = foodTypeService.selectFoodTypeList(foodType);

ExcelUtil<FoodType> util = new ExcelUtil<FoodType>(FoodType.class);

util.exportExcel(response, list, "菜品类型数据");

}

@PreAuthorize("@ss.hasPermi('food:type:query')")

@GetMapping(value = "/{foodtypeId}")

public AjaxResult getInfo(@PathVariable("foodtypeId") Long foodtypeId)

{

return success(foodTypeService.selectFoodTypeByFoodtypeId(foodtypeId));

}

@PreAuthorize("@ss.hasPermi('food:type:add')")

@Log(title = "菜品类型", businessType = BusinessType.INSERT)

@PostMapping

public AjaxResult add(@RequestBody FoodType foodType)

{

return toAjax(foodTypeService.insertFoodType(foodType));

}

@PreAuthorize("@ss.hasPermi('food:type:edit')")

@Log(title = "菜品类型", businessType = BusinessType.UPDATE)

@PutMapping

public AjaxResult edit(@RequestBody FoodType foodType)

{

return toAjax(foodTypeService.updateFoodType(foodType));

}

@PreAuthorize("@ss.hasPermi('food:type:remove')")

@Log(title = "菜品类型", businessType = BusinessType.DELETE)

@DeleteMapping("/{foodtypeIds}")

public AjaxResult remove(@PathVariable Long[] foodtypeIds)

{

Return toAjax(foodTypeService.deleteFoodTypeByFoodtypeIds(foodtypeIds));

}

}

获取规格前端主要代码：

getList() {

this.loading = true;

listSepcification(this.queryParams).then(response => {

this.sepcificationList = response.rows;

this.total = response.total;

this.loading = false;

});

}

后端主要代码：

@RestController

@RequestMapping("/food/sepcification")

public class FoodSepcificationController extends BaseController

{

@Autowired

private IFoodSepcificationService foodSepcificationService;

@PreAuthorize("@ss.hasPermi('food:sepcification:list')")

@GetMapping("/list")

public TableDataInfo list(FoodSepcification foodSepcification)

{

startPage();

List<FoodSepcification> list = foodSepcificationService.selectFoodSepcificationList(foodSepcification);

return getDataTable(list);

}

@PreAuthorize("@ss.hasPermi('food:sepcification:export')")

@Log(title = "菜品规格", businessType = BusinessType.EXPORT)

@PostMapping("/export")

public void export(HttpServletResponse response, FoodSepcification foodSepcification)

{

List<FoodSepcification> list = foodSepcificationService.selectFoodSepcificationList(foodSepcification);

ExcelUtil<FoodSepcification> util = new ExcelUtil<FoodSepcification>(FoodSepcification.class);

util.exportExcel(response, list, "菜品规格数据");

}

@PreAuthorize("@ss.hasPermi('food:sepcification:query')")

@GetMapping(value = "/{spcfId}")

public AjaxResult getInfo(@PathVariable("spcfId") Long spcfId)

{

return success(foodSepcificationService.selectFoodSepcificationBySpcfId(spcfId));

}

@PreAuthorize("@ss.hasPermi('food:sepcification:add')")

@Log(title = "菜品规格", businessType = BusinessType.INSERT)

@PostMapping

public AjaxResult add(@RequestBody FoodSepcification foodSepcification)

{

return toAjax(foodSepcificationService.insertFoodSepcification(foodSepcification));

}

@PreAuthorize("@ss.hasPermi('food:sepcification:edit')")

@Log(title = "菜品规格", businessType = BusinessType.UPDATE)

@PutMapping

public AjaxResult edit(@RequestBody FoodSepcification foodSepcification)

{

return toAjax(foodSepcificationService.updateFoodSepcification(foodSepcification));

}

@PreAuthorize("@ss.hasPermi('food:sepcification:remove')")

@Log(title = "菜品规格", businessType = BusinessType.DELETE)

@DeleteMapping("/{spcfIds}")

public AjaxResult remove(@PathVariable Long[] spcfIds)

{

return toAjax(foodSepcificationService.deleteFoodSepcificationBySpcfIds(spcfIds));

}

}

* + 1. 订单管理模块的实现

1. 功能描述

如图，使用者可以对订单管理模块进行增删改查等操作

图形用户界面, 应用程序

描述已自动生成

图 51 订单管理图

电脑萤幕截图

描述已自动生成

图 52 订单管理图



图 53 订单管理图

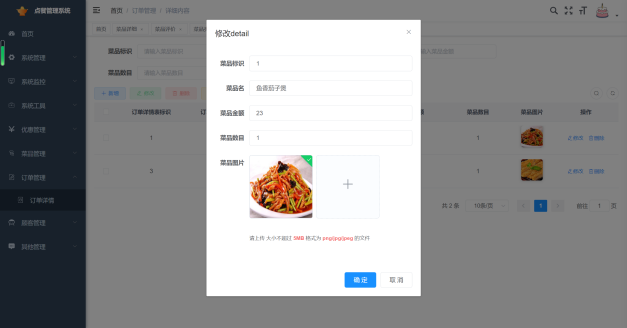


图 54 订单管理图

1. 模块设计

使用者在前端通过post请求将订单数据传送给控制层OrderInfoController，接着调用业务层OrderInfoService中的updateOrderInfo，在此期间的数据都是采用了Model层的实体类进行了必要的封装，进而将其视为整体进行数据的传送，之后进入Dao层的OrderInfoMapper调用updateOrderInfo修改订单内容，把结果返回给OrderInfoController，最后通过Json格式返回成功或错误的修改结果给前端，并且让其进行界面的渲染。而这一些图片则采用类似异步的方法，可以让前端先上传给后端，后端再给前端返回链接地址，进而在此次调用中传回给后端进行链接URL的存储，这样就可以保证用户在使用时更加顺畅。

图示

描述已自动生成

图 55 更新订单时序图

使用者在前端通过post请求传给控制层OrderDetailController，接着调用业务层OrderDetailService中的updateOrderDetail，之后进入Dao层的OrderDetailMapper调用updateOrderDetail修改订单详细内容，把结果返回给OrderDetailController，最后通过Json格式返回修改结果成功或错误给前端，并且让其进行界面的渲染。

图示

描述已自动生成

图 56 更新订单详细时序图

1. 相关代码

前端主要代码：

getList() {

this.loading = true;

listInfo(this.queryParams).then(response => {

this.infoList = response.rows;

this.total = response.total;

this.loading = false;

});

}

后端主要代码：

@RestController

@RequestMapping("/order/detail")

public class OrderDetailController extends BaseController

{

@Autowired

private IOrderDetailService orderDetailService;

@PreAuthorize("@ss.hasPermi('order:detail:list')")

@GetMapping("/list")

public TableDataInfo list(OrderDetail orderDetail)

{

startPage();

List<OrderDetail> list = orderDetailService.selectOrderDetailList(orderDetail);

return getDataTable(list);

}

@PreAuthorize("@ss.hasPermi('order:detail:export')")

@Log(title = "detail", businessType = BusinessType.EXPORT)

@PostMapping("/export")

public void export(HttpServletResponse response, OrderDetail orderDetail)

{

List<OrderDetail> list = orderDetailService.selectOrderDetailList(orderDetail);

ExcelUtil<OrderDetail> util = new ExcelUtil<OrderDetail>(OrderDetail.class);

util.exportExcel(response, list, "detail数据");

}

@PreAuthorize("@ss.hasPermi('order:detail:query')")

@GetMapping(value = "/{detailId}")

public AjaxResult getInfo(@PathVariable("detailId") Long detailId)

{

return success(orderDetailService.selectOrderDetailByDetailId(detailId));

}

@PreAuthorize("@ss.hasPermi('order:detail:add')")

@Log(title = "detail", businessType = BusinessType.INSERT)

@PostMapping

public AjaxResult add(@RequestBody OrderDetail orderDetail)

{

return toAjax(orderDetailService.insertOrderDetail(orderDetail));

}

@PreAuthorize("@ss.hasPermi('order:detail:edit')")

@Log(title = "detail", businessType = BusinessType.UPDATE)

@PutMapping

public AjaxResult edit(@RequestBody OrderDetail orderDetail)

{

return toAjax(orderDetailService.updateOrderDetail(orderDetail));

}

@PreAuthorize("@ss.hasPermi('order:detail:remove')")

@Log(title = "detail", businessType = BusinessType.DELETE)

@DeleteMapping("/{detailIds}")

public AjaxResult remove(@PathVariable Long[] detailIds)

{

return toAjax(orderDetailService.deleteOrderDetailByDetailIds(detailIds));

}

}

* + 1. 红包卡券管理模块的实现

1. 功能描述

使用者可以对优惠管理模块进行增删改查等操作。

图形用户界面, 应用程序

描述已自动生成

图 57 红包卡券详细图



图 58 红包卡券详细图

电脑软件截图

描述已自动生成

图 59 兑换码图

电脑软件截图

描述已自动生成

图 60 兑换码生成图

图形用户界面, 文本, 应用程序

描述已自动生成

图 61 兑换码生成图

1. 模块设计

使用者在前端通过post请求传给控制层DiscountInfoController，接着调用业务层DiscountInfoService中的updateDiscountInfoDetail，之后进入Dao层的DiscountInfoMapper调用updateDiscountInfo修改红包卡券的内容，把结果返回给DiscountInfoController，最后通过Json格式返回修改结果成功或错误给前端，并且让前端进行渲染。

图示

描述已自动生成

图 62 更新红包卡券时序图

使用者在前端通过post请求传给控制层DiscountKeyController，接着调用工具层Jwtutil的createToken方法，将优惠券信息传过去形成token，接着插入到业务层的DiscountKeyService中的updateDiscountKey，之后进入Dao层的DiscountKeyMapper调用updateDiscountKey插入生成的兑换码token，把结果返回给DiscountKeyController，最后通过Json格式返回插入结果成功或错误以及生成的优惠券token给前端，进而让前端对这些数据进行一些必要的渲染。

图表, 图示, 箱线图

描述已自动生成

图 63 生成兑换码时序图

1. 相关代码

红包卡券前端主要代码：

getList() {

this.loading = true;

listInfo(this.queryParams).then(response => {

this.infoList = response.rows;

this.total = response.total;

this.loading = false;

});

}

红包卡券后端主要代码：

@RestController

@RequestMapping("/discount/info")

public class DiscountInfoController extends BaseController

{

@Autowired

private IDiscountInfoService discountInfoService;

@PreAuthorize("@ss.hasPermi('discount:info:list')")

@GetMapping("/list/createKey")

public AjaxResult createKey(OtherDiscountInfo otherDiscountInfo)

{

String token="";

List<String> list=new ArrayList<>();

for(int i=0;i<otherDiscountInfo.getNumber();i++){

token = JwtUtil.createJWT(otherDiscountInfo.getDiscountId() + "");

list.add(token);

}

return success(list);

}

@PreAuthorize("@ss.hasPermi('discount:info:list')")

@GetMapping("/list")

public TableDataInfo list(DiscountInfo discountInfo)

{

startPage();

List<DiscountInfo> list = discountInfoService.selectDiscountInfoList(discountInfo);

return getDataTable(list);

}

@PreAuthorize("@ss.hasPermi('discount:info:export')")

@Log(title = "红包卡券", businessType = BusinessType.EXPORT)

@PostMapping("/export")

public void export(HttpServletResponse response, DiscountInfo discountInfo)

{

List<DiscountInfo> list = discountInfoService.selectDiscountInfoList(discountInfo);

ExcelUtil<DiscountInfo> util = new ExcelUtil<DiscountInfo>(DiscountInfo.class);

util.exportExcel(response, list, "红包卡券数据");

}

@PreAuthorize("@ss.hasPermi('discount:info:query')")

@GetMapping(value = "/{discountId}")

public AjaxResult getInfo(@PathVariable("discountId") Long discountId)

{

return success(discountInfoService.selectDiscountInfoByDiscountId(discountId));

}

@PreAuthorize("@ss.hasPermi('discount:info:add')")

@Log(title = "红包卡券", businessType = BusinessType.INSERT)

@PostMapping

public AjaxResult add(@RequestBody DiscountInfo discountInfo)

{

return toAjax(discountInfoService.insertDiscountInfo(discountInfo));

}

@PreAuthorize("@ss.hasPermi('discount:info:edit')")

@Log(title = "红包卡券", businessType = BusinessType.UPDATE)

@PutMapping

public AjaxResult edit(@RequestBody DiscountInfo discountInfo)

{

return toAjax(discountInfoService.updateDiscountInfo(discountInfo));

}

@PreAuthorize("@ss.hasPermi('discount:info:remove')")

@Log(title = "红包卡券", businessType = BusinessType.DELETE)

@DeleteMapping("/{discountIds}")

public AjaxResult remove(@PathVariable Long[] discountIds)

{

return toAjax(discountInfoService.deleteDiscountInfoByDiscountIds(discountIds));

}

}

兑换码页面展示的前端主要代码:

getList() {

this.loading = true;

listKey(this.queryParams).then(response => {

this.keyList = response.rows;

this.total = response.total;

this.loading = false;

});

}

后端主要代码：

@RestController

@RequestMapping("/discount/key")

public class DiscountKeyController extends BaseController

{

@Autowired

private IDiscountKeyService discountKeyService;

@PreAuthorize("@ss.hasPermi('discount:key:addKey')")

@GetMapping("/list/createKey")

public AjaxResult createKey(DiscountKey discountKey)

{

System.out.println(discountKey);

List<String> tmp = discountKeyService.createDiscountKey(discountKey);

return success(tmp);

}

@PreAuthorize("@ss.hasPermi('discount:key:list')")

@GetMapping("/list")

public TableDataInfo list(DiscountKey discountKey)

{

startPage();

List<DiscountKey> list = discountKeyService.selectDiscountKeyList(discountKey);

return getDataTable(list);

}

@PreAuthorize("@ss.hasPermi('discount:key:export')")

@Log(title = "红包卡券兑换码", businessType = BusinessType.EXPORT)

@PostMapping("/export")

public void export(HttpServletResponse response, DiscountKey discountKey)

{

List<DiscountKey> list = discountKeyService.selectDiscountKeyList(discountKey);

ExcelUtil<DiscountKey> util = new ExcelUtil<DiscountKey>(DiscountKey.class);

util.exportExcel(response, list, "红包卡券兑换码数据");

}

@PreAuthorize("@ss.hasPermi('discount:key:query')")

@GetMapping(value = "/{keyId}")

public AjaxResult getInfo(@PathVariable("keyId") Long keyId)

{

return success(discountKeyService.selectDiscountKeyByKeyId(keyId));

}

@PreAuthorize("@ss.hasPermi('discount:key:add')")

@Log(title = "红包卡券兑换码", businessType = BusinessType.INSERT)

@PostMapping

public AjaxResult add(@RequestBody DiscountKey discountKey)

{

System.out.println(discountKey);

List<String> tmp = discountKeyService.createDiscountKey(discountKey);

return success(tmp);

}

@PreAuthorize("@ss.hasPermi('discount:key:edit')")

@Log(title = "红包卡券兑换码", businessType = BusinessType.UPDATE)

@PutMapping

public AjaxResult edit(@RequestBody DiscountKey discountKey)

{

return toAjax(discountKeyService.updateDiscountKey(discountKey));

}

@PreAuthorize("@ss.hasPermi('discount:key:remove')")

@Log(title = "红包卡券兑换码", businessType = BusinessType.DELETE)

@DeleteMapping("/{keyIds}")

public AjaxResult remove(@PathVariable Long[] keyIds)

{

return toAjax(discountKeyService.deleteDiscountKeyByKeyIds(keyIds));

}

}

* + 1. 顾客管理模块的实现

1. 功能描述

电脑软件截图

描述已自动生成

图 64 顾客管理图

图形用户界面, 应用程序

描述已自动生成

图 65 顾客管理图

电脑软件截图

描述已自动生成

图 66 顾客权限图

图形用户界面, 应用程序

描述已自动生成

图 67 权限管理图

1. 模块设计

使用者在前端调用通过post请求传给后端的控制层CustomerInfoController，接着调用业务层的CustomerInfoService中的updateCustomerInfo方法，之后进入Dao层的CustomerInfoMapper调用updateCustomerInfo方法更新顾客信息，把权限管理的结果返回给之前调用这一方法的控制层CustomerInfoController，在此前的所有的一系列传送数据以及方法调用的操作中，这些权限相关的数据在传送都是运用了Model层里面的实体类进行数据的封装，进而进行数据的传送，最后通过Json格式返回插入结果成功或错误给前端，进而让前端进行必要的渲染。

图示

描述已自动生成

图 68 更新顾客信息时序图

使用者在前端调用方法，通过post请求传给控制层CustomerPermsController，接着调用业务层的CustomerPermsService中的updateCustomerPerms方法，之后进入Dao层的CustomerPermsMapper调用updateCustomerPerms方法更新顾客权限，把顾客信息更新反馈的结果返回给CustomerPermsController，在此前的数据传送以及方法调用之中，均采用Model层的实体类进行必要的封装，最后通过Json格式返回修改结果成功或错误给前端，让前端进行必要的渲染。

图示, 箱线图

描述已自动生成

图 69 更新顾客权限时序图

1. 相关代码

顾客模块前端主要代码：

getList() {

this.loading = true;

listInfo(this.queryParams).then(response => {

this.infoList = response.rows;

this.total = response.total;

this.loading = false;

});

}

后端主要代码：

@RestController

@RequestMapping("/customer/info")

public class CustomerInfoController extends BaseController

{

@Autowired

private ICustomerInfoService customerInfoService;

@PreAuthorize("@ss.hasPermi('customer:info:list')")

@GetMapping("/list")

public TableDataInfo list(CustomerInfo customerInfo){

startPage();

List<CustomerInfo> list = customerInfoService.selectCustomerInfoList(customerInfo);

return getDataTable(list);

}

@PreAuthorize("@ss.hasPermi('customer:info:export')")

@Log(title = "info", businessType = BusinessType.EXPORT)

@PostMapping("/export")

public void export(HttpServletResponse response, CustomerInfo customerInfo)

{

List<CustomerInfo> list = customerInfoService.selectCustomerInfoList(customerInfo);

ExcelUtil<CustomerInfo> util = new ExcelUtil<CustomerInfo>(CustomerInfo.class);

util.exportExcel(response, list, "info数据");

}

@PreAuthorize("@ss.hasPermi('customer:info:query')")

@GetMapping(value = "/{customerId}")

public AjaxResult getInfo(@PathVariable("customerId") Long customerId)

{

return success(customerInfoService.selectCustomerInfoByCustomerId(customerId));

}

@PreAuthorize("@ss.hasPermi('customer:info:add')")

@Log(title = "info", businessType = BusinessType.INSERT)

@PostMapping

public AjaxResult add(@RequestBody CustomerInfo customerInfo){

return toAjax(customerInfoService.insertCustomerInfo(customerInfo));}

@PreAuthorize("@ss.hasPermi('customer:info:edit')")

@Log(title = "info", businessType = BusinessType.UPDATE)

@PutMapping

public AjaxResult edit(@RequestBody CustomerInfo customerInfo)

{

return toAjax(customerInfoService.updateCustomerInfo(customerInfo));

}

@PreAuthorize("@ss.hasPermi('customer:info:remove')")

@Log(title = "info", businessType = BusinessType.DELETE)

@DeleteMapping("/{customerIds}")

public AjaxResult remove(@PathVariable Long[] customerIds)

{

return toAjax(customerInfoService.deleteCustomerInfoByCustomerIds(customerIds));

}

}

顾客权限前端主要代码：

getList() {

this.loading = true;

listPerms(this.queryParams).then(response => {

this.permsList = response.rows;

this.total = response.total;

this.loading = false;

});

}

后端主要代码：

@RestController

@RequestMapping("/customer/perms")

public class CustomerPermsController extends BaseController

{

@Autowired

private ICustomerPermsService customerPermsService;

@PreAuthorize("@ss.hasPermi('customer:perms:list')")

@GetMapping("/list")

public TableDataInfo list(CustomerPerms customerPerms)

{

startPage();

List<CustomerPerms> list = customerPermsService.selectCustomerPermsList(customerPerms);

return getDataTable(list);

}

@PreAuthorize("@ss.hasPermi('customer:perms:export')")

@Log(title = "权限", businessType = BusinessType.EXPORT)

@PostMapping("/export")

public void export(HttpServletResponse response, CustomerPerms customerPerms)

{

List<CustomerPerms> list = customerPermsService.selectCustomerPermsList(customerPerms);

ExcelUtil<CustomerPerms> util = new ExcelUtil<CustomerPerms>(CustomerPerms.class);

util.exportExcel(response, list, "权限数据");

}

@PreAuthorize("@ss.hasPermi('customer:perms:query')")

@GetMapping(value = "/{id}")

public AjaxResult getInfo(@PathVariable("id") Long id){

return success(customerPermsService.selectCustomerPermsById(id));}

@PreAuthorize("@ss.hasPermi('customer:perms:add')")

@Log(title = "权限", businessType = BusinessType.INSERT)

@PostMapping

public AjaxResult add(@RequestBody CustomerPerms customerPerms)

{

return toAjax(customerPermsService.insertCustomerPerms(customerPerms));

}

@PreAuthorize("@ss.hasPermi('customer:perms:edit')")

@Log(title = "权限", businessType = BusinessType.UPDATE)

@PutMapping

public AjaxResult edit(@RequestBody CustomerPerms customerPerms)

{

return toAjax(customerPermsService.updateCustomerPerms(customerPerms));

}

@PreAuthorize("@ss.hasPermi('customer:perms:remove')")

@Log(title = "权限", businessType = BusinessType.DELETE)

@DeleteMapping("/{ids}")

public AjaxResult remove(@PathVariable Long[] ids)

{

return toAjax(customerPermsService.deleteCustomerPermsByIds(ids));}}

* + 1. 联系客服模块的实现

1. 功能描述

前台饭店点餐系统的客户可以通过联系前台模块跟后台管理员平台的聊天模块的服务员进行在线聊天、在线催单等。

图形用户界面, 应用程序

描述已自动生成

图 70 实时聊天图

1. 模块设计

使用者在前端通过与服务器建立WebSocket长链接，之后就可以通过这一链接进行实时对话，而实时对话的聊天格式还是采用JSON进行交互。管理员端与每一个餐桌的聊天记录都会存在本地的SessionStorage存储中，每次选择某一餐桌的客户时，都会自动显示与其之前的对话内容到聊天框之中。客户的聊天记录也会每次都存储在本地的SessionStorage存储之中，下次再次打开这一界面时，会自动弹出与后台服务员的所有对话内容。后端的数据传送以及方法调用仍然采用Model层的实体类进行必要的封装。

1. 相关代码

前端代码：

send() {

if (!this.message.msg) {

uni.showToast({

title: '请输入信息',

icon: 'error',

duration: 2000

})

} else {

var that=this;

this.message.from=this.deskId;

this.message.time=new Date().toLocaleTimeString();

socket.send(JSON.stringify(this.message));

let obj={time:that.message.time,

to: '客服',

from:that.deskId,

msg:that.message.msg

}

this.msgList=[...this.msgList,obj]

this.message.msg = '';

}

},

init() {

let that = this;

let socketUrl = "ws://localhost:8080/websocket/"+this.deskId;

socket = new WebSocket(socketUrl);

socket.onopen = function () {

console.log("websocket已打开");

};

socket.onmessage = function (msg) {

let data = JSON.parse(msg.data)

if(data.to=="all"){

console.log(data);

}else

that.msgList=[...that.msgList,data]};

socket.onclose = function () {

console.log("websocket已关闭");};

socket.onerror = function () {

console.log("websocket发生了错误");}}}

后端代码：

@Component

@ServerEndpoint("/websocket/{deskId}")

public class WebSocketChat {

public static final Map<String, Session> sessionMap = new ConcurrentHashMap<>();

@OnOpen

public void onOpen(Session session, @PathParam("deskId") String deskId) {

System.out.println("当前=="+deskId);

sessionMap.put(deskId, session);

Message message = new Message();

message.setFrom("客服");

message.setTo(deskId);

message.setMsg("请问有什么问题呢?");

sendMessage(message,session);

sendAllMessage(setUserList());

}

@OnClose

public void onClose(@PathParam("deskId") String username) {

sessionMap.remove(username);

}

@OnMessage

public void onMessage(String message,Session session) {

try{

System.out.println( message);

ObjectMapper mapper = new ObjectMapper();

Message mess = mapper.readValue(message,Message.class);

String toName = mess.getTo();

String data = mess.getMsg();

String from = mess.getFrom();

Session toName1 = sessionMap.get(toName);

sendMessage(mess,toName1);

System.out.println(message);

}catch (Exception e){

Message msg = new Message();

msg.setFrom("服务器");

msg.setTo("客服");

msg.setMsg("对方已下线或服务器出错，暂时无法联系!");

this.sendMessage(msg,session);

System.out.println("对方已下线或服务器出错，暂时无法联系!");

e.printStackTrace();

}

}

@OnError

public void onError(Session session, Throwable error) {

System.out.println("发生错误");

error.printStackTrace();

}

private String setUserList(){

ArrayList<String> list = new ArrayList<>();

for(String key:sessionMap.keySet()){

list.add(key);

}

return String.join(",",list);

}

private void sendMessage(Message message, Session toSession) {

try {

toSession.getBasicRemote().sendText(JSON.toJSONString(message));

} catch (IOException e) {

e.printStackTrace();

}

}

private void sendAllMessage(String message) {

Message message1 = new Message();

message1.setMsg(message);

message1.setTo("all");

System.out.println(message);

try {

for (Session session : sessionMap.values()) {

session.getBasicRemote().sendText(JSON.toJSONString(message1));

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

* + 1. 其他模块的实现

1. 功能描述

所有的后台使用者均可看到订单的营业额以及菜品销售量等报表。

图形用户界面, 应用程序

描述已自动生成

图 71 首页报表模块图

1. 模块设计

使用者在前端调用方法，通过get请求将请求的相关数据通过URL的拼接传给控制层OrderInfoController，接着调用后端的业务层的OrderInfoService中的updateOrderInfo方法，之后进入Dao层的OrderInfoMapper调用updateOrderInfo方法查找报表，把结果返回给之前调用这一方法的控制层OrderInfoController，接着在这一层里面再对其进行分类分期整合等操作。在此之前的所有操作的数据传送都采用Model层里面的实体类进行了必要的数据封装，最后通过Json格式返回结果给前端，让前端的方法对这一JSON格式的数据进行解析，从而进行必要的渲染。

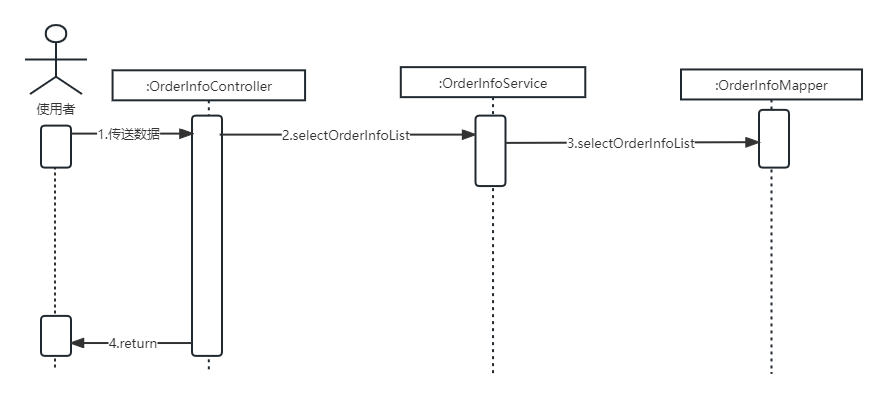


图 72 首页报表模块图

1. 相关代码

前端主要代码：

setMoney(e){

getMoneyChart(e!=null?this.addDateRange({pay\_status:1}, this.mondateRange):{pay\_status:1}).then(response => {

console.log(response.data);

this.chartMon=[...response.data.val];

this.chartTime=[...response.data.name];

this.showMonChart();

console.log(this.chartMon)

console.log(this.chartTime)

});

}

后端主要代码：

@PreAuthorize("@ss.hasPermi('order:info:sale')")

@GetMapping("/sale")

public TableDataInfo sale()

{

OrderInfo orderInfo=new OrderInfo();

Map ret=new HashMap();

Map map = new HashMap<>();

Date date=new Date();

SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd 00:00:00");

Calendar calendar=new GregorianCalendar();//今天销售额

calendar.setTime(date);

calendar.add(Calendar.DATE,1);

map.put("beginTime",sdf.format(date));

map.put("endTime",sdf.format(calendar.getTime()));

orderInfo.setParams(map);

orderInfo.setPayStatus("1");

List<OrderInfo> list = orderInfoService.selectOrderInfoList(orderInfo);

BigDecimal bigDecimal=new BigDecimal(0);

for(OrderInfo tmp:list){

bigDecimal=bigDecimal.add(tmp.getOrderAmount());

}

ret.put("todaysale",bigDecimal);//昨天销售额

calendar.setTime(date);

calendar.add(Calendar.DATE,-1);

map.put("beginTime", sdf.format(calendar.getTime()));

map.put("endTime",sdf.format(date));

orderInfo.setParams(map);

List<OrderInfo> list1 = orderInfoService.selectOrderInfoList(orderInfo);

BigDecimal bigDecimal1=new BigDecimal(0);

for(OrderInfo tmp:list1){

bigDecimal1=bigDecimal1.add(tmp.getOrderAmount());

}

ret.put("yestodaysale",bigDecimal1);//今年销售额

SimpleDateFormat sdf1=new SimpleDateFormat("yyyy-01-01 00:00:00");

calendar.setTime(date);

calendar.add(Calendar.YEAR,1);

map.put("beginTime", sdf1.format(date));

map.put("endTime",sdf1.format(calendar.getTime()));

orderInfo.setParams(map);

List<OrderInfo> list2 = orderInfoService.selectOrderInfoList(orderInfo);

BigDecimal bigDecimal2=new BigDecimal(0);

for(OrderInfo tmp:list2){

bigDecimal2=bigDecimal2.add(tmp.getOrderAmount());

}

ret.put("thisyearsale",bigDecimal2);

ArrayList arrayList=new ArrayList();

arrayList.add(ret);

//去年销售额

SimpleDateFormat sdf2=new SimpleDateFormat("yyyy-01-01 00:00:00");

calendar.setTime(date);

calendar.add(Calendar.YEAR,-1);

map.put("beginTime", sdf2.format(calendar.getTime()));

map.put("endTime",sdf2.format(date));

orderInfo.setParams(map);

List<OrderInfo> list3 = orderInfoService.selectOrderInfoList(orderInfo);

BigDecimal bigDecimal3=new BigDecimal(0);

for(OrderInfo tmp:list3){

bigDecimal3=bigDecimal3.add(tmp.getOrderAmount());

}

ret.put("lastyearsale",bigDecimal3);

ArrayList arrayList1=new ArrayList();

arrayList1.add(ret);

return getDataTable(arrayList1);}

1. 系统测试

表16 用户登录测试

|  |  |
| --- | --- |
| 测试用例名称 | 用户登录测试 |
| 测试目的 | 确保用户可以成功登录到系统 |
| 前置条件 | |
| 系统已启动，并且登录注册模块正常运行；  存在已注册的测试账号和密码。 | |
| 测试步骤 | |
| 打开系统登录页面；  输入正确的测试账号和密码；  点击“登录”按钮。 | |
| 预期结果 | |
| 页面跳转到个人界面；  用户成功登录到系统；  用户名和头像正确显示在页面右上角；  用户登录状态被记录，可以在系统中正常使用所有功能。 | |

表17 用户注册测试

|  |  |
| --- | --- |
| 测试用例名称 | 用户注册测试 |
| 测试目的 | 确保用户可以成功注册到系统 |
| 前置条件 | |
| 系统已启动，并且登录注册模块正常运行。  测试环境中不存在与待注册的账号重复的账号。 | |
| 测试步骤 | |
| 打开系统注册页面；  输入正确的用户名、密码和确认密码；  点击“注册”按钮。 | |
| 预期结果 | |
| 页面跳转到系统登录页面；  注册成功的提示信息正确显示；  可以使用已注册的账号和密码进行登录。 | |

表18 用户创建订单测试

|  |  |
| --- | --- |
| 测试用例名称 | 创建订单 |
| 测试目的 | 验证用户创建订单的功能是否正常 |
| 前置条件 | |
| 用户已经登录成功，选择了购买的商品。 | |
| 测试步骤 | |
| 用户点击“购买”按钮  系统弹出订单确认页面  用户核对订单信息，确认无误后点击“确认支付”按钮  系统生成订单并跳转到支付页面  用户完成支付，系统提示订单创建成功 | |
| 预期结果 | |
| 系统能够正常跳转到订单确认页面  订单确认页面能够正确显示用户选择的商品信息  确认订单无误后，系统能够正常生成订单并跳转到支付页面  用户能够正常完成支付，系统提示订单创建成功 | |

表19 用户查看订单详情测试

|  |  |
| --- | --- |
| 测试用例名称 | 查看订单详情 |
| 测试目的 | 验证用户查看订单详情功能是否正常 |
| 前置条件 | |
| 用户已经登录成功并创建了订单。 | |
| 测试步骤 | |
| 用户点击“订单列表”按钮  系统跳转到订单列表页面  用户选择需要查看的订单并点击“查看详情”按钮  系统跳转到订单详情页面用户核对订单信息，确认无误后点击“确认支付”按钮  系统生成订单并跳转到支付页面  用户完成支付，系统提示订单创建成功 | |
| 预期结果 | |
| 系统能够正常跳转到订单列表页面  订单列表页面能够正确显示用户创建的订单信息  确认订单无误后，系统能够正常跳转到订单详情页面  订单详情页面能够正确显示订单的详细信息，包括商品信息、订单状态、付款信息、下单桌号等 | |

表20 用户评价提交测试

|  |  |
| --- | --- |
| 测试用例名称 | 评价提交 |
| 测试目的 | 确保用户可以成功提交评价并且评价信息能够正确地保存到数据库中。 |
| 前置条件 | |
| 用户已经登录成功。 | |
| 测试步骤 | |
| 登录系统并打开评价页面。  填写评价信息，包括评分、评价内容等。  选择星星评分  上传本地图片，并且进行图片格式判断  点击提交按钮。  检查数据库中是否有新增的评价记录，并且评价信息是否正确。 | |
| 预期结果 | |
| 用户填写的评价信息能够成功提交到数据库中。  数据库中新增一条评价记录，并且评价信息与用户填写的信息一致。 | |

表21 用户评价列表展示测试

|  |  |
| --- | --- |
| 测试用例名称 | 评价列表展示 |
| 测试目的 | 确保系统能够正确地展示评价列表，并且每条评价的信息都能够正确显示。 |
| 前置条件 | |
| 用户已经登录成功。 | |
| 测试步骤 | |
| 登录系统并打开评价列表页面。  检查页面是否能够正确地展示所有的评价记录，包括订单编号、评分、评价内容等信息。 | |
| 预期结果 | |
| 系统能够正确地展示所有的评价记录，并且每条评价的信息都能够正确显示。 | |

表22 展示菜品列表测试

|  |  |
| --- | --- |
| 测试用例名称 | 展示菜品列表 |
| 测试目的 | 验证系统是否能够正确地展示所有的菜品信息。 |
| 前置条件 | |
| 用户已经登录成功。 | |
| 测试步骤 | |
| 进入菜品展示页面。  检查页面中是否能够正确地展示所有的菜品信息。  检查每个菜品信息是否包括名称、价格、图片等基本信息。  确认所有的菜品信息是否能够正确地显示，并且能够正常加载。 | |
| 预期结果 | |
| 所有的菜品信息都应该正确地显示在页面上。  每个菜品信息应该包括名称、价格、图片等基本信息。  所有的菜品信息应该能够正常加载，并且不应该出现加载失败等问题。 | |

表23 点餐下单测试

|  |  |
| --- | --- |
| 测试用例名称 | 点餐下单 |
| 测试目的 | 验证系统能够正确地处理用户的点餐订单。 |
| 前置条件 | |
| 用户已经登录成功。 | |
| 测试步骤 | |
| 进入菜品展示页面。  选择想要点餐的菜品。  输入数量以及其他相关信息，如规格集合等。  选择优惠券进行下单。  点击下单按钮，提交订单。  检查页面是否能够正确地显示已提交的订单信息。 | |
| 预期结果 | |
| 用户选择的菜品信息应该能够正确地显示在页面上。  用户输入的数量以及其他相关信息应该能够正确地被系统处理。  系统应该能够正确地提交用户的订单，并且不应该出现提交失败等问题。  提交订单后，页面应该能够正确地显示已提交的订单信息。 | |

表24 查看红包卡券详情测试

|  |  |
| --- | --- |
| 测试用例名称 | 查看红包卡券详情 |
| 测试目的 | 验证系统能够正确地显示红包卡券内容。 |
| 前置条件 | |
| 系统中已有用户账号并登录成功。  系统中已有可用红包卡券，并已经关联到该用户账号下。 | |
| 测试步骤 | |
| 在用户账号登录成功的状态下，进入红包卡券列表页面  点击指定红包卡券的详情按钮  确认红包卡券的相关信息（例如名称、有效期、折扣等）均正确显示 | |
| 预期结果 | |
| 红包卡券详情页面中该红包卡券的相关信息均正确显示 | |

表25 查看红包卡券列表测试

|  |  |
| --- | --- |
| 测试用例名称 | 查看红包卡券列表 |
| 测试目的 | 验证系统能够正确地显示红包卡券内容。 |
| 前置条件 | |
| 系统中已有用户账号并登录成功。  系统中已有可用红包卡券，并已经关联到该用户账号下。 | |
| 测试步骤 | |
| 在用户账号登录成功的状态下，进入红包卡券列表页面  确认页面中显示了该用户账号下的所有可用红包卡券，包括红包和折扣券  确认每个红包卡券的相关信息（例如名称、有效期、折扣等）均正确显示 | |
| 预期结果 | |
| 红包卡券列表页面显示正确。  每个红包卡券的相关信息均正确显示。 | |

表26 使用红包卡券下单测试

|  |  |
| --- | --- |
| 测试用例名称 | 使用红包卡券下单 |
| 测试目的 | 验证系统能够正确地显示红包卡券内容。 |
| 前置条件 | |
| 在用户账号登录成功的状态下，进入订单页面。  选择需要购买的菜品，并在结算页面选择使用指定的红包卡券进行抵扣。 | |
| 测试步骤 | |
| 在用户账号登录成功的状态下，进入订单页面。  选择需要购买的菜品。  在结算页面选择使用指定的红包卡券进行抵扣。  提交订单。  进入订单列表页面。  确认该订单中已经成功使用了指定的红包卡券进行抵扣。 | |
| 预期结果 | |
| 订单列表页面中该订单已经成功使用了指定的红包卡券进行抵扣。  在红包卡券页面刚刚使用的优惠券状态变为已过期。 | |

表27 修改个人资料测试

|  |  |
| --- | --- |
| 测试用例名称 | 修改个人资料 |
| 测试目的 | 验证系统能够正确地修改个人资料。 |
| 前置条件 | |
| 用户已登录，并且当前显示的个人资料信息与需要修改的信息不同。 | |
| 测试步骤 | |
| 点击个人资料修改按钮。  修改需要更改的信息（例如昵称、手机号等）。  点击保存按钮。  检查页面是否成功跳转到个人资料页面，并且页面显示的信息已经更新为修改后的信息。 | |
| 预期结果 | |
| 个人资料修改成功。  页面显示的信息已经更新为修改后的信息。 | |

表28 新增菜品测试

|  |  |
| --- | --- |
| 测试用例名称 | 新增菜品 |
| 测试目的 | 验证系统能够正确地新增菜品。 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 进入菜品管理页面。  点击新增按钮。  填写菜品名称、价格、描述等必要信息。  选择菜品图片并上传。  点击保存按钮。 | |
| 预期结果 | |
| 页面跳转到菜品管理列表页面  新增菜品信息显示在列表中 | |

表29 修改菜品测试

|  |  |
| --- | --- |
| 测试用例名称 | 修改菜品 |
| 测试目的 | 验证系统能够正确地修改菜品。 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 进入菜品管理页面。  找到需要修改的菜品信息，点击对应的编辑按钮。  修改菜品的名称、价格、描述等信息，选择新的菜品图片并上传。  点击保存按钮。 | |
| 预期结果 | |
| 页面跳转到菜品管理列表页面  修改后的菜品信息显示在列表中 | |

表30 删除菜品测试

|  |  |
| --- | --- |
| 测试用例名称 | 删除菜品 |
| 测试目的 | 验证系统能够正确地删除菜品。 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 进入菜品管理页面。  找到需要删除的菜品信息，点击对应的删除按钮。  在弹出的确认提示框中点击确认按钮。 | |
| 预期结果 | |
| 页面跳转到菜品管理列表页面。  删除的菜品信息不再显示在列表中。 | |

表31 搜索菜品测试

|  |  |
| --- | --- |
| 测试用例名称 | 搜索菜品 |
| 测试目的 | 验证系统能够正确地删除菜品。 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 进入菜品管理页面。  在搜索框中输入关键词，如菜品名称、菜品分类等。  点击搜索的按钮。 | |
| 预期结果 | |
| 页面自动在当前页面跳转到搜索结果列表页面。  列表中显示符合搜索条件的菜品信息。  列表中可以进行修改、删除等操作。  列表还可以查看菜品的图片。 | |

表32 管理员查询订单列表测试

|  |  |
| --- | --- |
| 测试用例名称 | 管理员查询订单列表 |
| 测试目的 | 确保管理员能够正确地查询订单列表 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 进入订单管理页面。  点击“查询订单”按钮。  验证订单列表是否正确显示，并且包含了所有的订单信息，如订单编号、下单时间、用户信息、订单状态等。 | |
| 预期结果 | |
| 管理员能够成功地查询到订单列表。  订单列表中显示的订单信息与数据库中存储的订单信息一致。 | |

表33 管理员修改订单状态测试

|  |  |
| --- | --- |
| 测试用例名称 | 管理员修改订单状态 |
| 测试目的 | 确保管理员能正确修改订单状态 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 进入订单管理页面。  找到待修改状态的订单，并点击该订单的“修改状态”按钮。  选择新的订单状态，并点击“确认”按钮。  验证订单状态是否已经成功地修改。  并且数据库中的订单状态也已经被更新。 | |
| 预期结果 | |
| 管理员能够成功地修改订单状态  订单状态已经被成功地更新到数据库中 | |

表34 管理员查看订单详情测试

|  |  |
| --- | --- |
| 测试用例名称 | 管理员查看订单详情 |
| 测试目的 | 确保管理员能够正确地查看订单详情 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 进入订单管理页面。  找到待查看详情的订单，并点击该订单的“查看详情”按钮。  验证订单详情是否正确显示，并且包含了所有的订单信息，如订单编号、下单时间、用户信息、订单状态、订单中的菜品等。 | |
| 预期结果 | |
| 管理员能够成功地查看订单详情。  订单详情中显示的订单信息与数据库中存储的订单信息一致。 | |

表35 管理员添加新红包卡券测试

|  |  |
| --- | --- |
| 测试用例名称 | 管理员添加新红包卡券 |
| 测试目的 | 确保管理员能够成功添加新的红包卡券 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 点击 "添加红包卡券" 按钮。  填写红包卡券信息，包括名称、面值、数量、有效期等等。  点击 "确认" 按钮提交红包卡券信息。 | |
| 预期结果 | |
| 页面提示 "添加成功"。  新添加的红包卡券出现在红包卡券列表中。 | |

表36 管理员编辑红包卡券信息测试

|  |  |
| --- | --- |
| 测试用例名称 | 管理员编辑红包卡券信息 |
| 测试目的 | 确保管理员能够成功编辑红包卡券的信息。 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 找到要编辑的红包卡券，并点击 "修改" 按钮。  修改红包卡券的信息，例如修改面值、有效期等等。  点击 "确认" 按钮提交修改。 | |
| 预期结果 | |
| 页面提示 "修改成功"。  编辑后的红包卡券信息在红包卡券列表中更新。 | |

表37 管理员删除红包卡券测试

|  |  |
| --- | --- |
| 测试用例名称 | 管理员删除红包卡券 |
| 测试目的 | 确保管理员能够成功删除红包卡券。 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 找到要删除的红包卡券，并点击 "删除" 按钮。  在弹出的确认对话框中点击 "确认" 按钮。 | |
| 预期结果 | |
| 页面提示 "删除成功"。  之后页面自动刷新。  被删除的红包卡券不再出现在红包卡券列表中。 | |

表38 管理员编辑顾客资料测试

|  |  |
| --- | --- |
| 测试用例名称 | 管理员编辑顾客资料 |
| 测试目的 | 确保管理员能够编辑顾客资料。 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 点击“修改”按钮。  修改需要编辑的资料，如姓名、手机号等。  点击“保存”按钮。 | |
| 预期结果 | |
| 点击“修改”按钮后，所有资料应该处于可编辑状态。  修改完需要编辑的资料后，保存按钮应该处于可用状态。  点击“保存”按钮后，应该有提示信息提示保存成功。  刷新页面后，应该可以看到修改后的顾客资料。 | |

表39 管理员添加顾客资料测试

|  |  |
| --- | --- |
| 测试用例名称 | 管理员添加顾客资料 |
| 测试目的 | 确保管理员能够添加顾客资料。 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 点击“添加”按钮。  输入需要添加的资料，如姓名、手机号等。  点击“保存”按钮。 | |
| 预期结果 | |
| 点击“添加”按钮后，所有资料应该处于可编辑状态。  输入完需要添加的资料后，保存按钮应该处于可用状态。  点击“保存”按钮后，应该有提示信息提示保存成功。  刷新页面后，应该可以看到新添加的顾客资料。 | |

表40 管理员删除顾客资料测试

|  |  |
| --- | --- |
| 测试用例名称 | 管理员删除顾客资料 |
| 测试目的 | 确保管理员能够添加顾客资料。 |
| 前置条件 | |
| 用户已登录。 | |
| 测试步骤 | |
| 选中要删除的顾客资料。  点击“删除”按钮。 | |
| 预期结果 | |
| 点击“删除”按钮后，应该有提示信息提示确认是否删除。  点击“确认”按钮后，应该有提示信息提示删除成功。  刷新页面后，应该可以看到被删除的顾客资料已经不存在。 | |

表41 管理员生成红包卡券兑换码测试

|  |  |
| --- | --- |
| 测试用例名称 | 管理员生成红包卡券兑换码 |
| 测试目的 | 确保管理员能够正确地生成相应的红包卡券兑换码。 |
| 前置条件 | |
| 用户已登录，红包卡券已存在于数据库中。 | |
| 测试步骤 | |
| 登录后台管理系统，进入红包卡券管理模块。  在红包卡券列表中选择一张红包卡券，获取其编码。  进入根据红包卡券编码生成用户兑换码模块。  在输入框中输入红包卡券编码。  点击生成兑换码按钮。  页面之后自动刷新。  检查页面中是否出现生成的兑换码。  查询数据库，验证生成的兑换码是否被正确保存。 | |
| 预期结果 | |
| 系统能够登录到后台管理系统。  进入红包卡券管理模块。  能够选择一张红包卡券并获取其编码。  能够进入根据红包卡券编码生成用户兑换码模块。  能够输入红包卡券编码。  点击生成兑换码按钮后能够在页面中出现生成的兑换码。  数据库中能够查询到生成的兑换码。 | |

表42 联系客服测试

|  |  |
| --- | --- |
| 测试用例名称 | 联系客服 |
| 测试目的 | 确保管理员能够与客户沟通。 |
| 前置条件 | |
| 管理员用户已登录，点餐客户已下桌 | |
| 测试步骤 | |
| 管理员用户登录后台管理系统，进入聊天模块。  在聊天列表中，选择一位客户。  发送相关点餐的消息。  点餐客户登录前台点餐系统，进入联系客服模块。  在聊天界面中，输入相关点餐的信息。  点击发送按钮进行发送信息。 | |
| 预期结果 | |
| 系统能够登录到后台管理系统以及能够进入到前台点餐系统。  管理员用户和点餐客户成功进入聊天模块。  双方发送的信息都显示在了聊天框。  下次打开时还可以显示在界面。  发送的相关点餐的消息在两方系统都能接收到。 | |

参考文献

1. 朱二华. 基于Vue.js的Web前端应用研究[J]. 科技与创新, 2017(20):119-121.
2. 杨家炜. 基于Spring Boot的web设计与实现[J]. 轻工科技, 2016(7):86-89.
3. 李子骅. Redis入门指南[M]. 人民邮电出版社, 2013.
4. 唐汉明, 翟振兴，关宝军. 深入浅出MySQL[M]. 人民邮电出版社, 2014.
5. 朱文奇．推荐系统用户相似度计算方法研究[D]．重庆：重庆大学，2014：18-22．
6. 吴中飞．面向家电产品售后服务的数据挖掘和分析技术研究[D]．浙江：浙江大学，2017：78-79．
7. 李翩,何明祥. 基于MVC的实验室管理系统设计[J].软件导刊,2016,15(10):94-96.
8. 葛汉强．基于Java EE的轻量级Web应用框架研究与实现[D]．湖北：武汉大学，2008：11-22．
9. 肖双林,何迎生,田杰,侯冬晴. 基于JWT+Spring Security的动态权限管理系统[J].信息与电脑(理论版), 2021,33(14):131-134.
10. 周常志,甘恒. 基于SpringBoot的智慧就业服务平台的设计与实现[J].电脑知识与技术, 2021,17(28):182-184+187.
11. 惠苗,赖道健. 基于WebSocket协议的即时通讯系统的开发[J].榆林学院学报, 2019,29(06):76-79.
12. 胡圣凯. 基于MVC框架的企业外勤管理系统的设计[J].软件, 2022,43(07):19-23.
13. 滕毅,马焯文. 基于Uniapp的校园拼车多端小程序设计与实现[J].电子制作, 2022,30(16):43-46.
14. Yao Zhang Li, Sheng Gao, Jing Pan, Bi Feng Guo, Pei Feng Xie.Research and Application of Template Engine for Web Back-end Based on MyBatis-Plus[Z].Procedia Computer Science.
15. Mohammed Alshammari, Aadil Alshammari.Friend Recommendation Engine for Facebook Users via Collaborative Filtering[Z].INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL.