长 沙 学 院

**软件工程基础实训Ⅲ** 实训

|  |  |
| --- | --- |
| **题目** | **网上蛋糕商城**  **——前后台管理** |
| **学院** | **计算机科学与工程** |
| **专业(班级)** | **软件工程（20软件02班）** |
| **姓名** | **戴博** |
| **学号** | **B20200404210** |
| **指导教师** | **张晓霞** |
| **起止日期** | **2022.12.05～2022.12.18** |

实训任务书

**课程名称：软件工程基础实训Ⅲ**

**实训题目：网上蛋糕商城**

### 已知技术参数和设计要求：

**1. 问题描述（功能要求）：**

本次案例将完成一个基于B/S架构的网上蛋糕商城。

该项目应满足以下需求。

（1）.统一友好的操作界面，具有良好的用户体验。

（2）.商品分类详尽，可按不同类别分别查看商品信息。

（3）.可通过条幅展示推荐商品。

（4）.热销推荐和新品推荐的展示。

（5）.用户注册、验证和登录功能。

（6）.通过蛋糕名称模糊搜索相关商品。

（7）.通过购物车一次购买多件商品。

（8）.提供简单的安全模型，用户必须登录后才可以购买商品。

（9）.用户选择商品后可以在线提交订单。

（10）.用户可以查看自己的订单信息。

（11）.设计网站后台，用于管理网站的各项基本数据，包括订单管理、商品管理、用户管理以及商品类目。

（12）.系统运行安全稳定且响应及时。

网上蛋糕商城项目实训主要要求。

（1）.进行网上蛋糕商城的功能设计。

（2）.设计网上蛋糕商城的数据库。

（3）.利用JSP、JS、AJAX、jQuery、Bootstrap框架等技术设计开发友好的人机接口。

（4）.利用JSP、Servlet、JDBC数据库连接池技术等进行项目的后台业务处理。

网上蛋糕商城主要有4大功能模块：

（1）.用户管理。

（2）.商品类目。

（3）.商品管理。

（4）.订单管理。

网上蛋糕商城从用户角色又分为前端系统和后端系统。

网上蛋糕商城可增加商品的评价，模拟支付（事务处理），按用户名和商品名等多条件查询订单，用户登录显示历史购物车等功能。项目可由1-2人完成。

**2. 运行环境要求：**

（1）客户端：

windows操作系统，JDK1.6，Eclipse或IDEA，浏览器。

（2）服务器：

MySQL5.6或 SQL Server 2008 数据库服务器、Tomcat。

**3. 技术要求：**

1） 掌握软件工程的需求分析和系统设计方法。

2) 掌握数据库的分析与设计，完成网上蛋糕商城数据库的设计。

3) 掌握JAVAWEB的前端设计方法，完成订单管理、商品管理、用户管理以及商品类目界面等人机接口。

4） 掌握JAVAWEB程序设计方法，设计系统所需的各种类。

5) 掌握JDBC方法，对数据库进行操作，完成项目的后台业务处理。

### 实训工作量：

40课时

### 工作计划：

1.班级

20软件1-4

2.课时及教室安排

见实训课程工作计划表。

实训报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学生姓名 | 戴博 | 学号 | B20200404210 | 班级 | 20软件02班 |
| 专业 | 软件工程 | | 指导教师姓名 | 张肖霞 | |
| 项目名称 | 网上蛋糕商城 | | | | |
| **实训过程**：（说明实训各阶段的工作过程，包括需求分析、系统设计、编码实现、系统测试、交付实施等阶段的主要工作。）  需求分析：随着网络的普及与发展，网上购物逐渐成为一种主流消费的方式，它以方便，足不出户，款式多全面，低廉的价格、送货上门的便捷等等优点称为了人们购物的主要方式，在顺应互联网大势的情况下经过多方面的调查需求，现推出网上蛋糕订购系统来满足个体蛋糕店来打造自己品牌的需求。  总体设计：系统分为两个主模块用户和管理员。对于用户模块在进入网站主页时用户可以先浏览商店信息可以在商品分类里面进行商品分类查询或者在热销和新品里面进行浏览，然后把想要购买的商品加到购物车上然后在行订单查询里面进行支付。对于管理员模块除了可以进行和用户部分一样的功能以外还可以进入后台管理模块，进行订单管理和客户管理还有商品管理，类别管理以及退出。  编码实现：见文档末尾附件。  **系**统测试：见说明测试部分。 | | | | | |
| **实训成果**：  参见网上蛋糕商城设计说明书。 | | | | | |
| **实训总结**：（包括心得体会、存在的问题和改进方向。）  这是我第一次使用前后端分离的方式，使用Vue+SpringBoot框架来编写javaweb项目，整个项目都是边学边做的，所以考虑不到的东西有很多。一是由于刚学遇到的问题太多了，导致没有对整个项目的时间有一个大概的估计，花费的时间远远超出了预期，之后在制定项目计划时一定还有考虑好自身的水平条件；二可能是最大的一个问题，数据库的设计问题很大，复杂度、效率极低、sql语句的设计，还有表的主键外键之间的关系这些根本没有考虑到位，虽然这个在程序中有一些实现，但是在数据库的层面上来说是错误的，所以说在设计数据库时不仅仅要考虑到自己表的作用关系还需要考虑表与表之间的联系；三很多检验都是在前端检验的，后端只是做了业务的处理，这样极易导致项目收到攻击；四是学到框架都是一些皮毛，仅仅只是能用一些简单的东西，但是真正的重要的东西还没学，后面就是要好好学一下ssm框架，还有Vue的基础知识点。  虽然说我还是存在很多问题，但是相比起上次的实验项目而言来说，确实是强了很多，无论是项目的设计还是代码的编写熟练了很多，现在相比之下更多的问题可能是框架的不熟练，以及项目的效率问题 | | | | | |

**网上蛋糕商城**

设计说明书

作者：戴博(B20200404210)

计算机科学与工程学院

2022年 12 月 16日

**摘要**

摘要应概括反映出课程设计的内容、方法、成果和结论。结果和结论性字句是摘要的重点，在文字论述上要多些，以加深读者的印象；用精炼、概括的语言来表达，每项内容不宜展开论证或说明，要客观陈述，不宜加主观评价；要独立成文，选词用语要避免与全文尤其是前言和结论部分雷同；摘要中不宜使用公式、图表，不标注引用文献编号。避免将摘要写成目录式的内容介绍。

关键词是供检索用的主题词条，应采用能覆盖全文主要内容的通用技术词条(参照教材附录A的名词索引或其它技术术语标准)。关键词一般列3~5个，按词条的外延层次排列（外延大的在前面）。

**关键词：Vue**，Element，SpringBoot， Mybatis，MVC，前后端分离，网上蛋糕商城

目录

[1 引言 1](#_Toc11882504)

[1.1 编写目的 1](#_Toc11882505)

[1.2 参考资料 1](#_Toc11882506)

[2 需求规约 2](#_Toc11882507)

[2.1 功能需求 2](#_Toc11882508)

[2.2 界面需求 3](#_Toc11882509)

[2.3 数据需求 3](#_Toc11882510)

[3 系统设计 4](#_Toc11882511)

[3.1 运行环境 4](#_Toc11882512)

[3.2 系统静态结构设计 4](#_Toc11882513)

[3.3 人机接口设计 5](#_Toc11882514)

[3.4 XXX模块设计说明 7](#_Toc11882515)

[3.5 XXX模块设计说明 7](#_Toc11882516)

[4 数据库设计 8](#_Toc11882517)

[4.1 数据库环境说明 8](#_Toc11882518)

[4.2 数据库的命名规则 8](#_Toc11882519)

[4.3 逻辑结构设计 8](#_Toc11882520)

[4.4 物理结构设计 9](#_Toc11882521)

[4.5 安全性设计 10](#_Toc11882522)

[5 测试用例设计 12](#_Toc11882523)

[附录 14](#_Toc11882524)

[附录1 程序运行结果 14](#_Toc11882525)

[附录2 程序源主要代码 15](#_Toc11882526)

# 1 引言

## 1.1 编写目的

本文档是“网上蛋糕商城”的软件设计说明书。编写目的是：明确软件需求、概要设计、详细设计和功能测试用例，为软件后续开发和维护提供指导。本文档的读者对象是需求分析人员、系统设计人员、系统开发人员、系统测试人员。

## 1.2 参考资料

[1] 文东. 数据库系统开发[M]. 北京: 北京科海电子出版社，2011．

[2] 黑马程序员. Java Web程序设计任务教程第2版[M]. 人民邮电出版社，2021

# 2 需求规约

## 2.1 功能需求

2.1.1 模糊查询用户用例说明

用例名称：动态模糊查询用户

用例描述：后台管理员动态模糊查询用户信息

前置条件：管理员输入想要查询的条件,点击查询按钮

后置条件：管理员或得相应用户信息

活动步骤：

1. 管理员请求获取模糊查询信息。
2. 发起axios请求并携带相应参数至后台。
3. 后台获取请求并查询返回json信息
4. 前台渲染分页显示用户信息

2.1.2 添加订单用例说明

用例名称：添加订单

用例描述：用户购买后添加订单

前置条件：用户将商品加入购物车并准备结算

后置条件：管理员已加入订单

活动步骤：

1. 用户点击添加购买按钮
2. 用户确定订单内容
3. 发起请求携带订单项参数将一个订单项加入订单表
4. 将该订单信息存放在订单表中
5. 订单列表发起请求重新刷新订单

2.1.3 订单管理用例说明

用例名称: 订单管理

用例描述：管理员登录后台查询订单，发货等

前置条件: 管理员登录后台，点击订单管理

活动步骤:

1. 管理员查询未发货订单
2. 对未处理的订单进行发货处理
3. 对发货成功的订单进行完成确定

2.1.4 商品管理用例说明

用例名称: 商品管理

用例描述：管理员登录后台后，对商品进行添加，修改，删除等

前置条件: 管理员登录后台，点击商品管理

后置条件: 管理员完成了商品管理的相关操作

活动步骤:

1. 用户按二级分类查询商品
2. 添加新上架商品
3. 对已上架商品进行修改，删除

2.1.5后台数据统计用例说明

用例名称: 后台数据统计

用例描述：用户登录后台后，查看相关数据统计图，便于管理

前置条件: 用户登录后台，点击后台数据统计

后置条件: 用户完成了后台数据统计的查看操作

活动步骤:

1. 用户点击后台数据统计
2. 查看相关数据信息

## 2.2 界面需求

**注:以下信息指代(用户信息,订单信息,商品信息等)**

1. 统一友好的操作界面，具有良好的用户体验

2. 可按不同类别分别查看信息

3. 提供添加修改删除信息以及模糊查询分页功能。

4. 通过名称等条件模糊搜索相关信息

5. 通过批量删除按钮一次删除多项信息

6. 提供信息统计图

## 2.3 数据需求

表2-1 用户数据表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 长度 | 描述 | 是否允许为空 |
| telephone | Varchar | 11 | 电话号码 | 否 |
| username | varchar | 20 | 用户名 | 否 |
| password | varchar | 30 | 用户密码 | 否 |
| Sex | varchar | 2 | 性别 | 是 |
| address | varchar | 255 | 收货地址 | 是 |

表2-2 系统管理员数据表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 长度 | 字段信息描述 | 是否为null |
| id | int | 11 | ID(主键) | 否 |
| username | varchar | 255 | 管理员名称 | 否 |
| password | varchar | 255 | 密码 | 否 |

表2-3　商品信息数据表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 长度 | 字段描述 | 是否可以为null |
| goodsId | int | 11 | 商品ID(主键) | 否 |
| goodsName | varchar | 30 | 商品名称 | 是 |
| goodsCategory | varchar | 50 | 商品分类 | 是 |
| goodsLogo | varchar | 100 | 商品图片 | 是 |
| goodsPrice | double | / | 商品价格 | 是 |
| goodsExist | boolean | / | 商品是否上线 | 是 |
| goodsDescription | varchar | 100 | 商品描述 | 是 |

表2-4　订单信息数据表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 长度 | 字段描述 | 是否可以为null |
| orderID | int | 11 | ID(主键) | 否 |
| orderTelephone | varchar | 11 | 电话号码 | 否 |
| orderPrice | double | / | 订单价格 | 否 |
| orderSatatus | varchar | 20 | 订单状态 | 否 |
| orderContents | varchar | 50 | 订单内容 | 否 |
| orderTime | varchar | 255 | 下单时间 | 是 |
| orderAddress | varchar | 255 | 下单地址 | 是 |

# 

# 3 系统设计

后台模块设计，主要围绕增删改查对数据进行显示，修改，添加，按数据库表进行分块管理：

1.用户管理:注册用户、登录、退出。

2.订单管理:显示所有订单以及实现不同状态订单查询，可删除订单。

3.商品管理:可按一级分类查看蛋糕、可添加蛋糕，修改蛋糕详情及分类，还可以删除该蛋糕。

4.类别管理：可添加新类别，删除类别，修改类别信息

5.数据统计：通过引入echars,通过柱状图和饼图来显示相关数据。

## 3.1 运行环境

操作系统: Windows XP以上版本。

服务器软件:Tomcat6.0以上版本。

浏览器: Chrome、Firfox等主流浏览器。

## 3.2 系统静态结构设计

系统静态结构说明

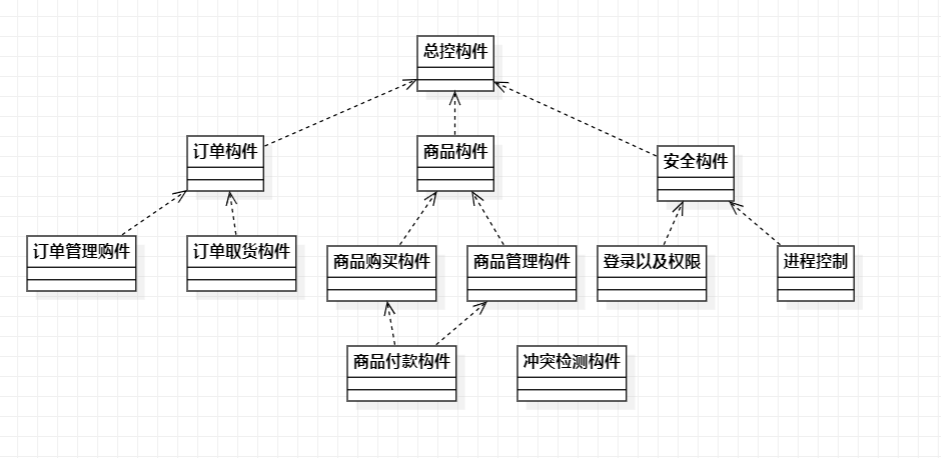


图3.2 网上蛋糕商城构件关系图

## 3.3 人机接口设计

3.3.1 设计应用类

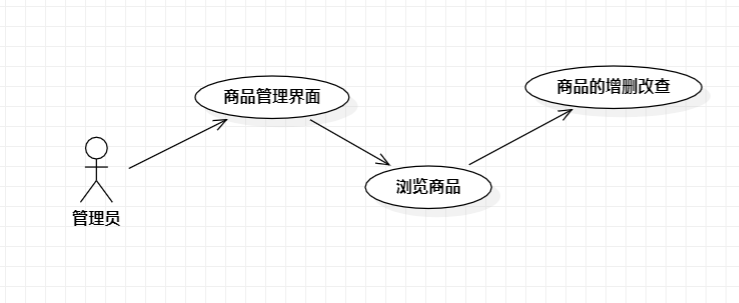


图3.2 商品管理的设计类图

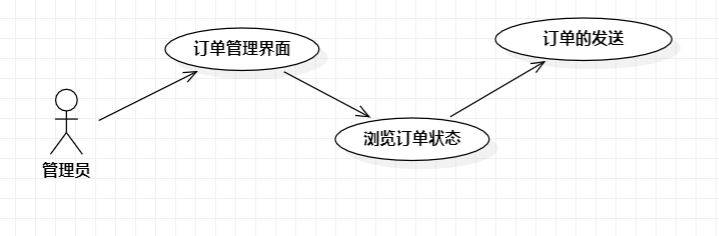


图3.3 订单管理的设计类图

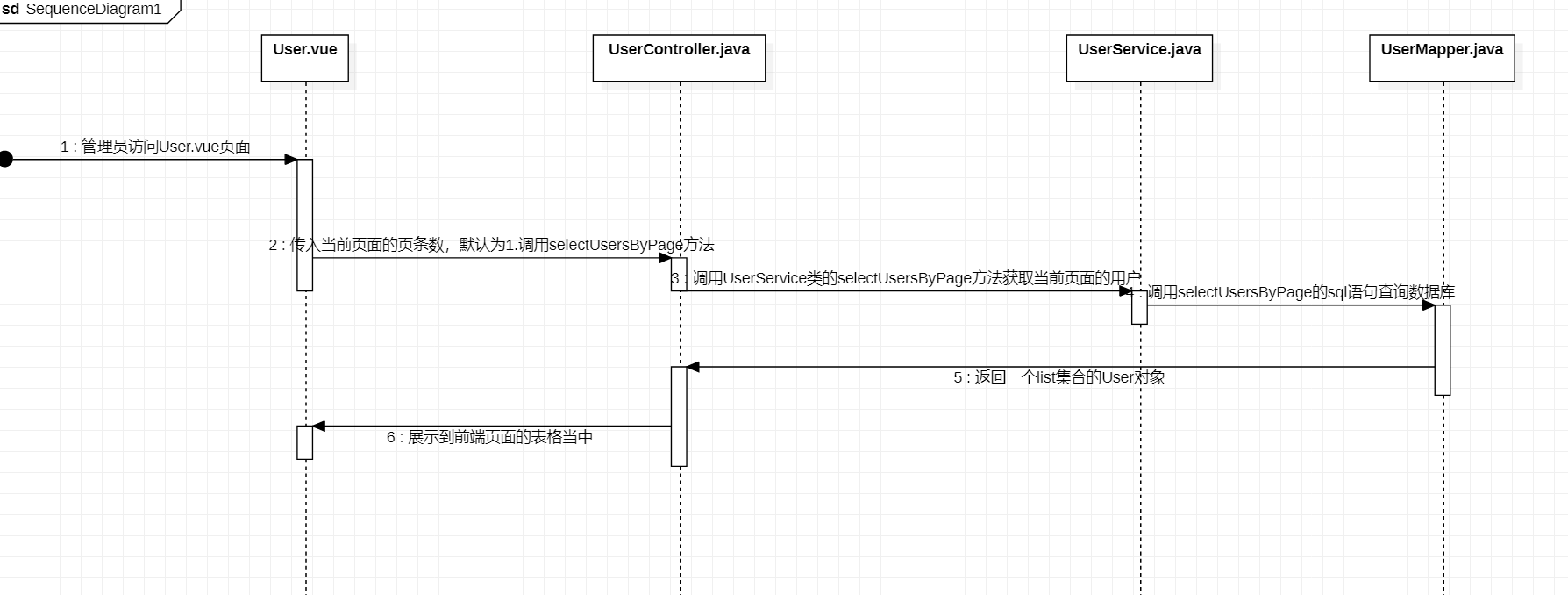
3.3.1 用户接口1（具体标识符）设计



## 3.4 普通用户模块设计说明

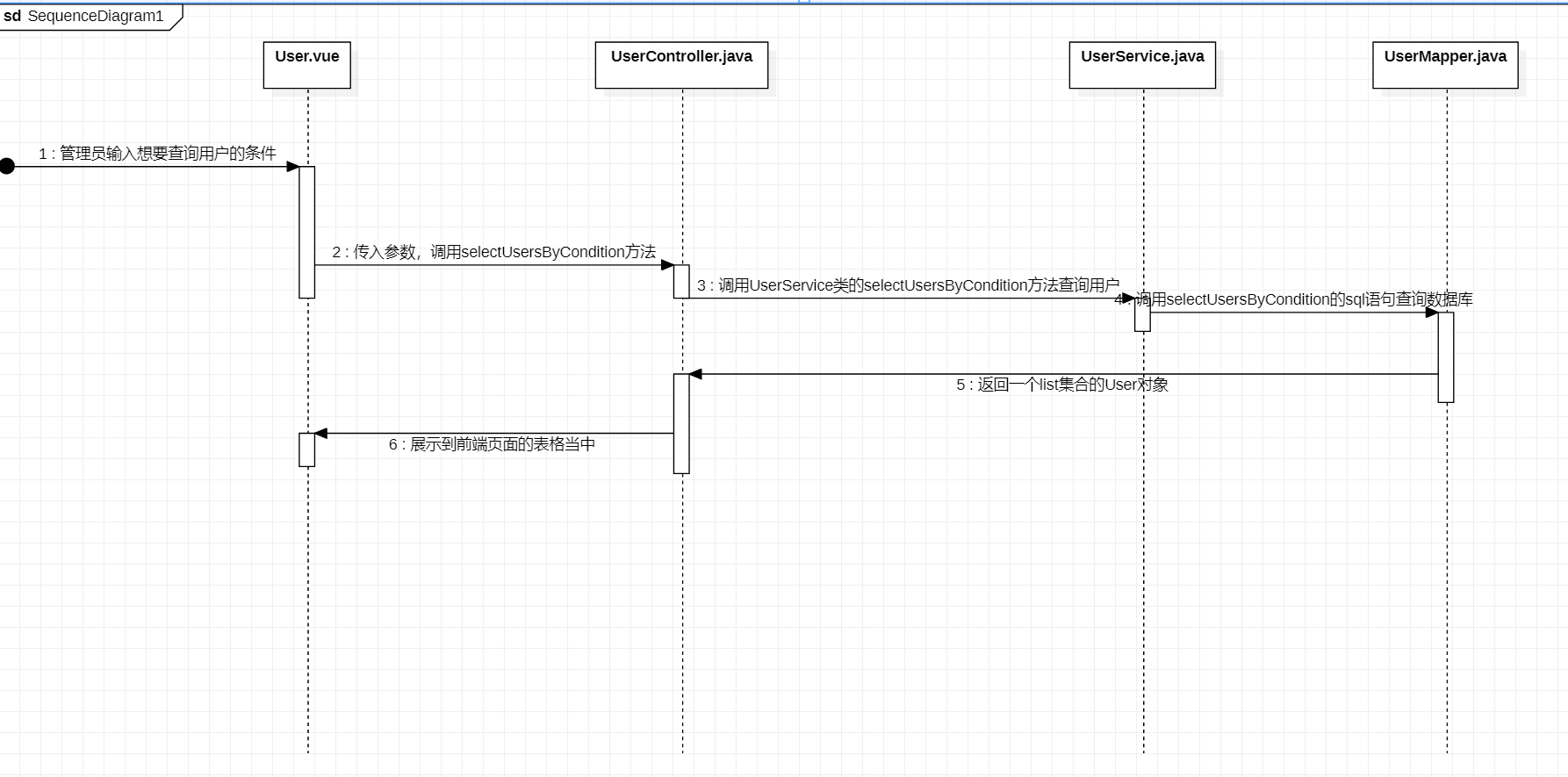
3.4.1、分页展示所有的用户

管理员访问user.vue的时候，在此之前就会向数据库发出请求，获取到对应的所有用户，其中它传递了一个数值为当前的页面，默认为第一页，之后传递给UserController的selectUserByPage()方法，然后再调用UserService方法中的selectUsersByPage方法，在这里，它会将传来的页面进行变化，变成(pageNum-1)\*10之后，传递给UserMapper，这是因为在UserMapper当中的sql语句为limit 起始条数，读取条数，当读取到第二页时就是从第十一条开始的了（这里我规定每一页有十条数据）。查询之后返回一个User对象的list集合



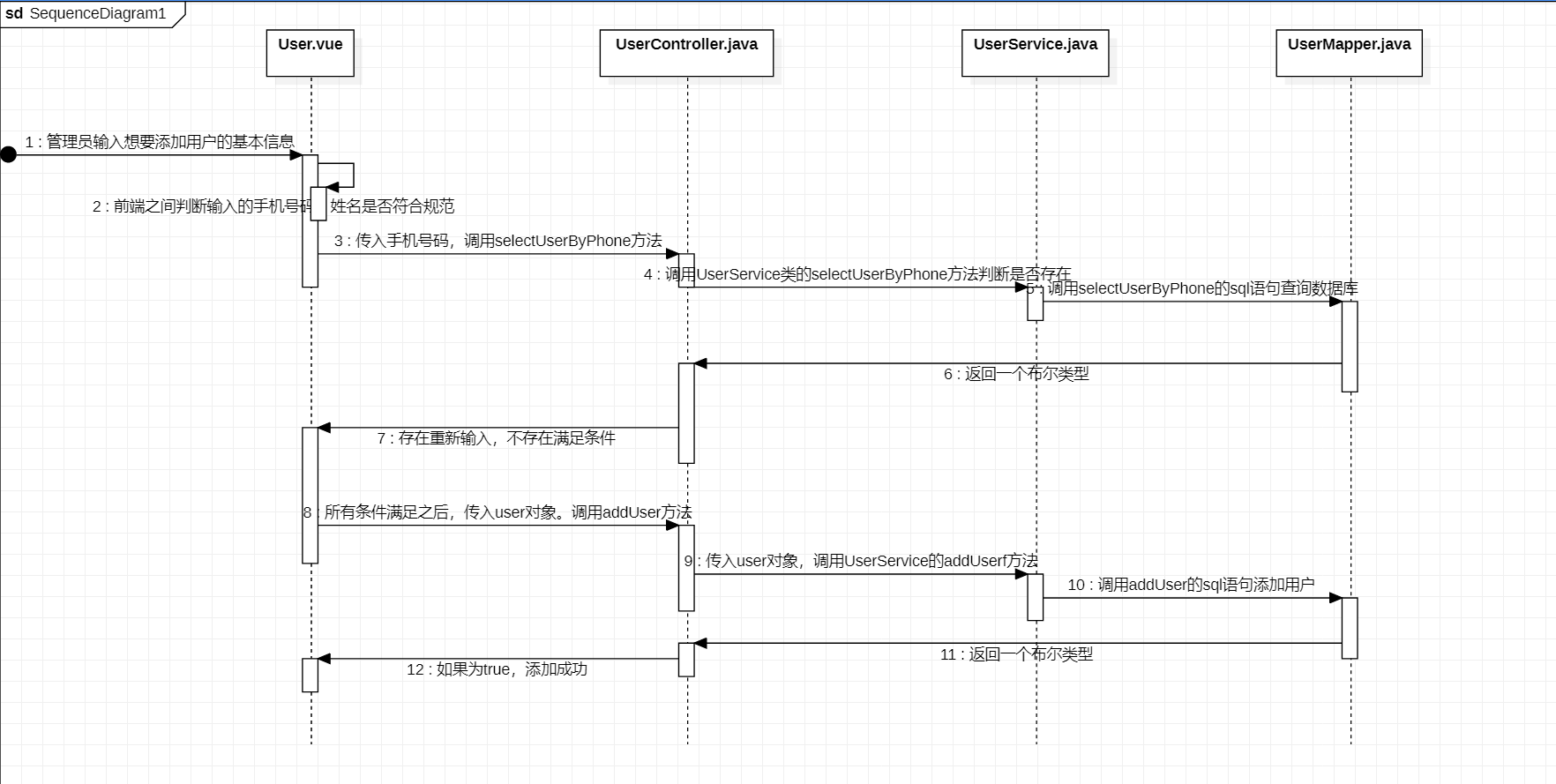
3.4.2、动态模糊查询用户

管理员访问user.vue页面，此时页面上面有三个输入框（包括电话号码、用户名和地址）和一个查询按钮。管理员可以任意往其中一个输入输入一个模糊查询的内容。点击查询之后，就会向UserController类的selectUsersByCondition方法发送这三个参数，之后交给UserService处理，进而交由UserMapper，在其中是UserMapper中的sql语句，在进行查询时会先对查询的参数进行判断，判断该数值是否为null或者为’’，如果是它就不会将这选项作为一个条件，如果一个条件没有则查询所有的用户。返回一个list集合的user对象给前端，前端然后转换成表格的形式展示给管理员



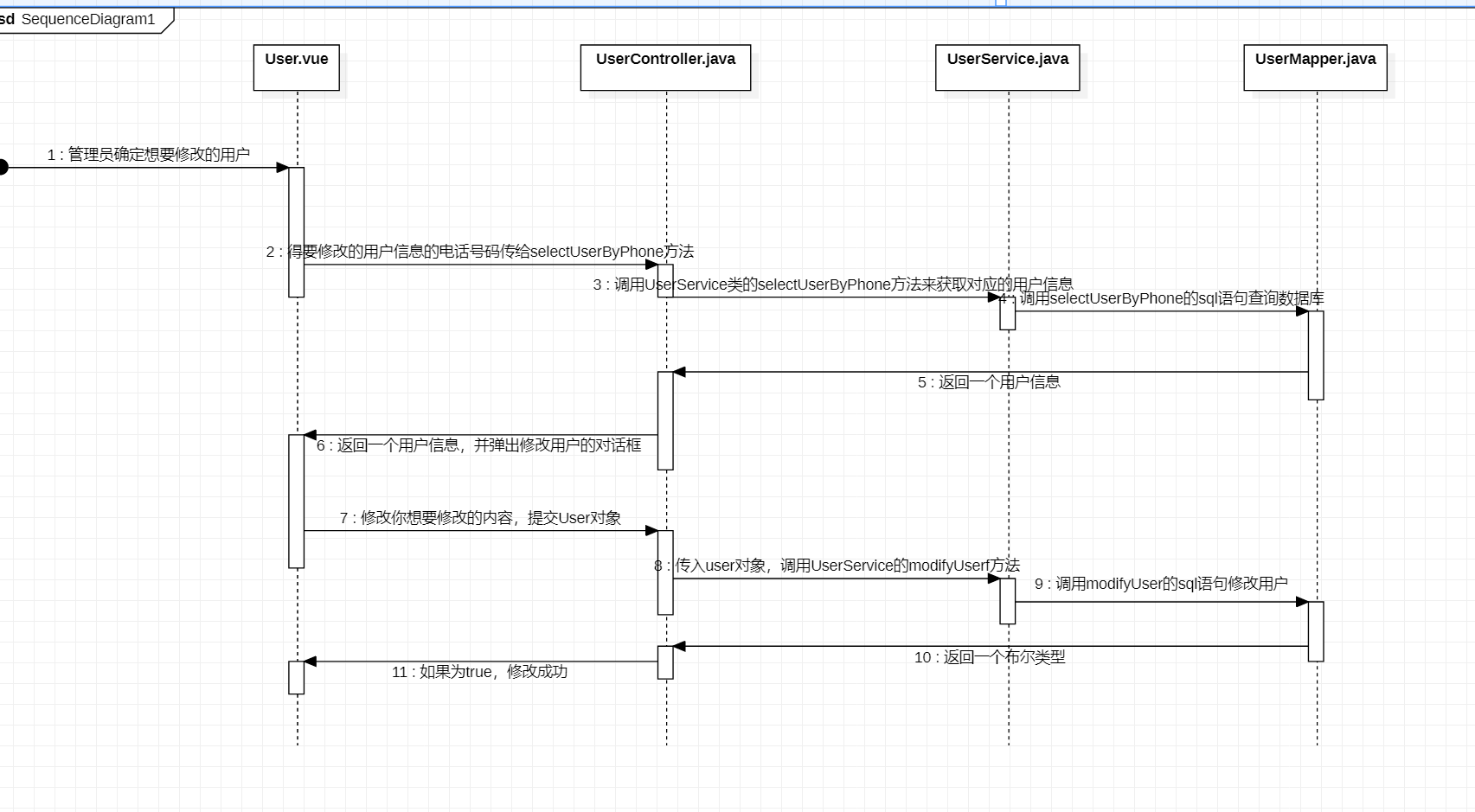
3.4.3、添加用户

管理员首先访问user.vue页面的添加按钮页面，此时就会弹出添加用户的对话框。其中存在输入添加用户的电话号码的输入框，首先前端会验证你输入的手机号码的格式是否符合规范，不符合规范就会弹出提示，当符合规范之后就会向UserController类中的selectUserByPhone()方法发出请求，判断该用户是否已经被注册。这个selectUserByPhone()方法是调用了UserService的selectUserByPhone()方法，而这个service中又调用了UserMapper的数据库查询。如果查询到返回true，此时会弹出已存在该用户的提示。之后还有一些用户名的长度限制在4~10位。当这些所有都符合之后。程序才会向后端UserController类的addUser()方法发出添加用户的请求。进而调用UserService中的UserMapper，实现数据的插入



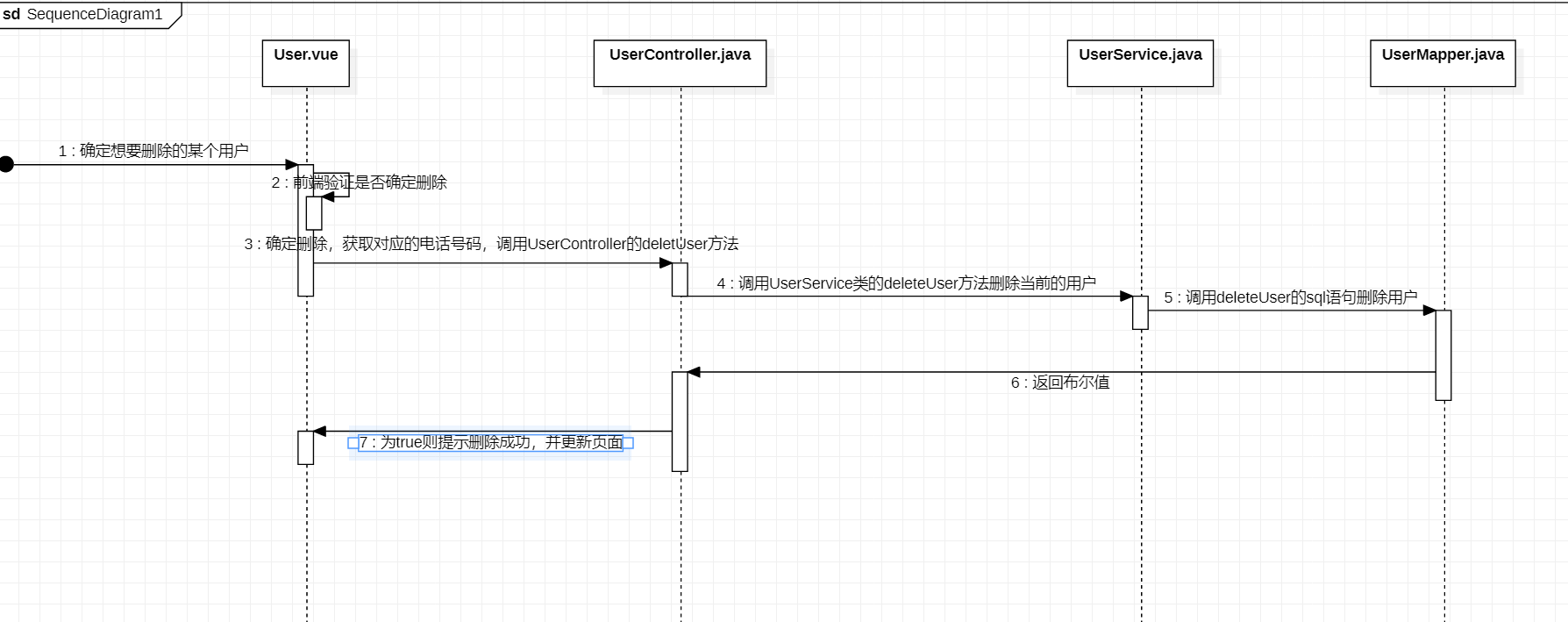
3.4.4、修改用户

管理员首先访问user.vue页面，选择你想要修改的那一列的的用户，然后获取到这一列的用户电话号码，之后传入参数调用UserController的selectUserByPhone方法，然后调用UserService的selectUserByPhone当中的UserMapper的selectUserByPhone的sql语句，查询到对应的用户的基本信息，然后再弹出一个修改用户的对话框展示在页面上面，用户可以修改你想要修改的内容，由于手机号码为唯一的主键，所以手机号码是禁止修改的。修改好你对应的基本信息之后，提交修改按钮，之后就会向UserController的modifyUser方法，进而执行UserService的modifyUser方法当中UserMapper的modifyUser的sql语句，实现修改用户，之后返回一个布尔类型的数据，如果为true，弹出提示修改成功，并且会关闭对话框并且刷新该表格。



3.4.5、删除用户

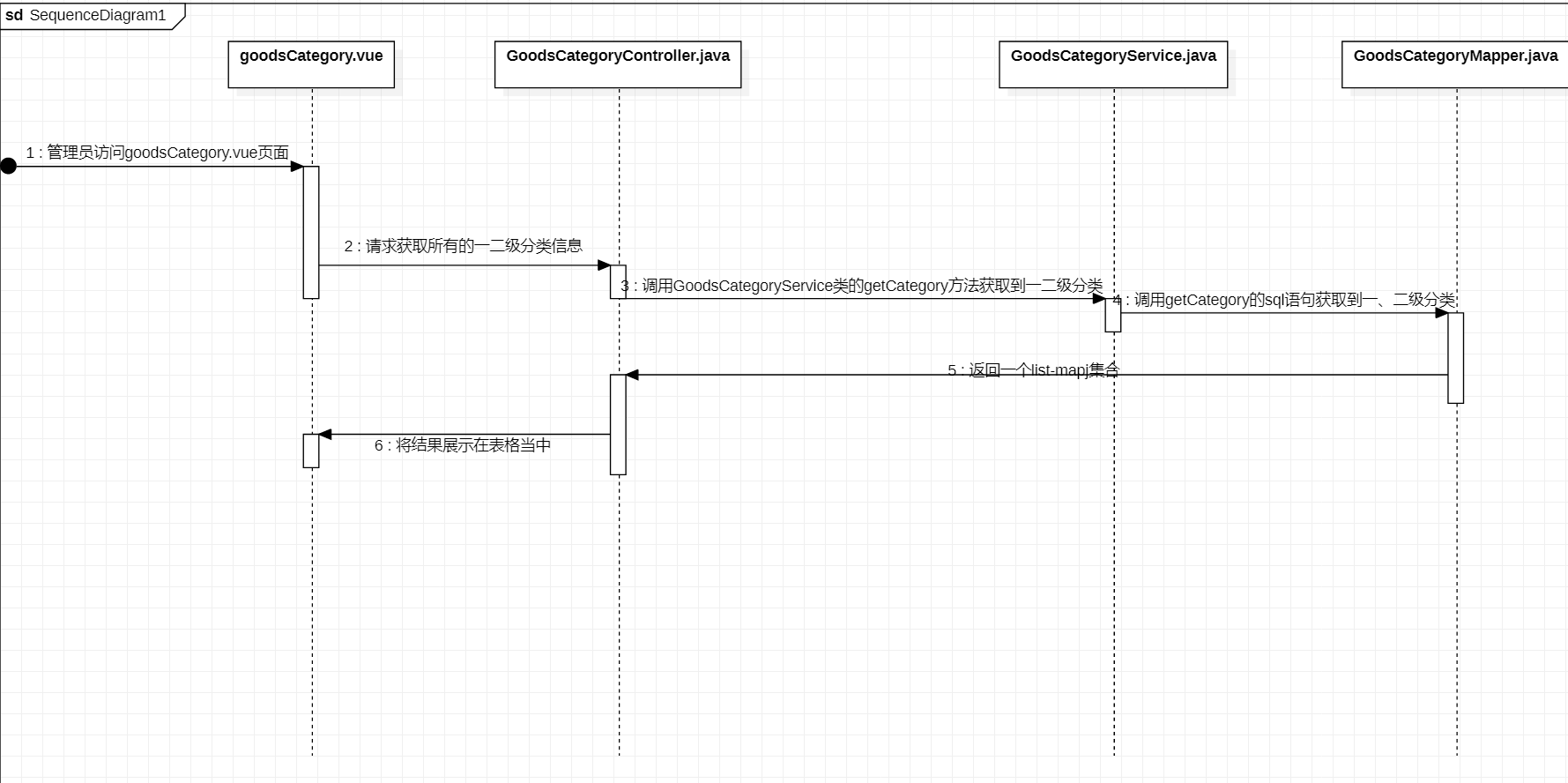
管理员首先访问user.vue页面，选择你想要删除的那一列的的用户，在删除之前会进行确认，确认是否删除，取消返回页面，点击确定之后获取到这一列的用户电话号码，之后传入参数调用UserController的deleteUser方法，然后调用UserService的deleteUser方法进而调用UserMapper实现对用户的删除



## 3.5 后台商品分类管理模块设计说明

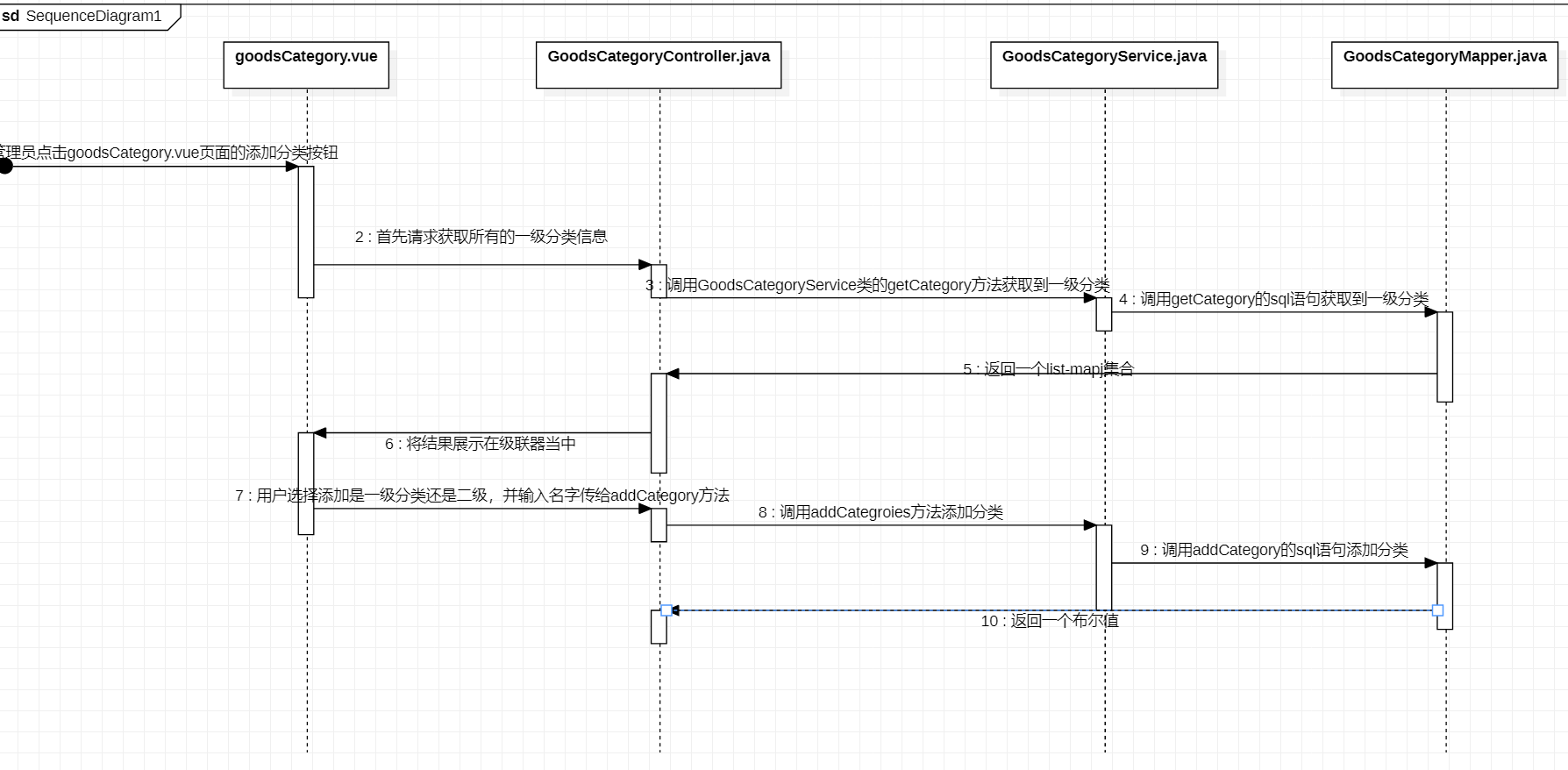
3.5.1：分类展示

管理员点击goodsCategory.vue页面后，页面会调用GoodsCategoryController类的selectCategroies方法的，然后去调用GoodsCategoryService的getCategory方法去获取到对应的一二级分类，由于这里的一级分类为List<Map<Object,Object>>，所以二级分类就为List<Map<Object,List<Map<Object,Object>>>>，这样当分类级数过大是显然需要使用到递归的方式来实现，所以在getCategory方法中调用getChilren方法，以此来获取到它的子类，如此递归方可实现，否则以一般的思维难以实现。查询到之后，返回list的map集合，最终展示对应的一、二级分类的基本信息



3.5.2：添加商品分类

管理员点击goodsCategory.vue页面的添加分类按钮，程序首先会查询到对应的一级分类的信息，这是由于我设置的多级分类只是二级分类，所有只需要展示一级分类，这里和3.5.1的类似，获取到一级分类展示到添加分类的对话框， 接着管理员可以选择添加的是一级分类还是二级分类，一级分类默认为空 ，二级分类选择属于哪个二级分类，之后添加好分类名称，就会向GoodsCategoryController类的addCategroies方法发出添加分类的请求，而它则会向GoodsCategoryService业务层的addCategory方法实现添加用户，在这里程序首先设置好是否存在父类，获取到父类的id，等信息整理好后，执行goodsCategoryMapper的addCategory的sql语句，以此来添加分类，之后返回一个布尔值，为真表示添加分类成功



3.5.3：修改商品分类、删除商品分类

这里与用户管理的修改用户与删除用户类似

## 3.6 后台统计表模块设计说明

3.6.1：今日销售额、本月销售额、今日订单数、本月订单数

管理员访问首页的welcome页面之后就会调用StatistiucController类的getStatistic方法来获取今日销售额、本月销售额、今日订单数、本月订单数。而getStatistic则是调用了StatisticService业务层的getStatistic方法当中的StatisticMapper的getStatisticsql语句来实现的，在这里由于要查询四个信息，所以需要四个sql语句，执行完四个sql语句之后，对于不同时间的查询，这里采用的是concat(DATE\_FORMAT(NOW(), '%Y-%m'),'%')的方式。 后端以json的格式返回对应的数值，然后展示在页面上

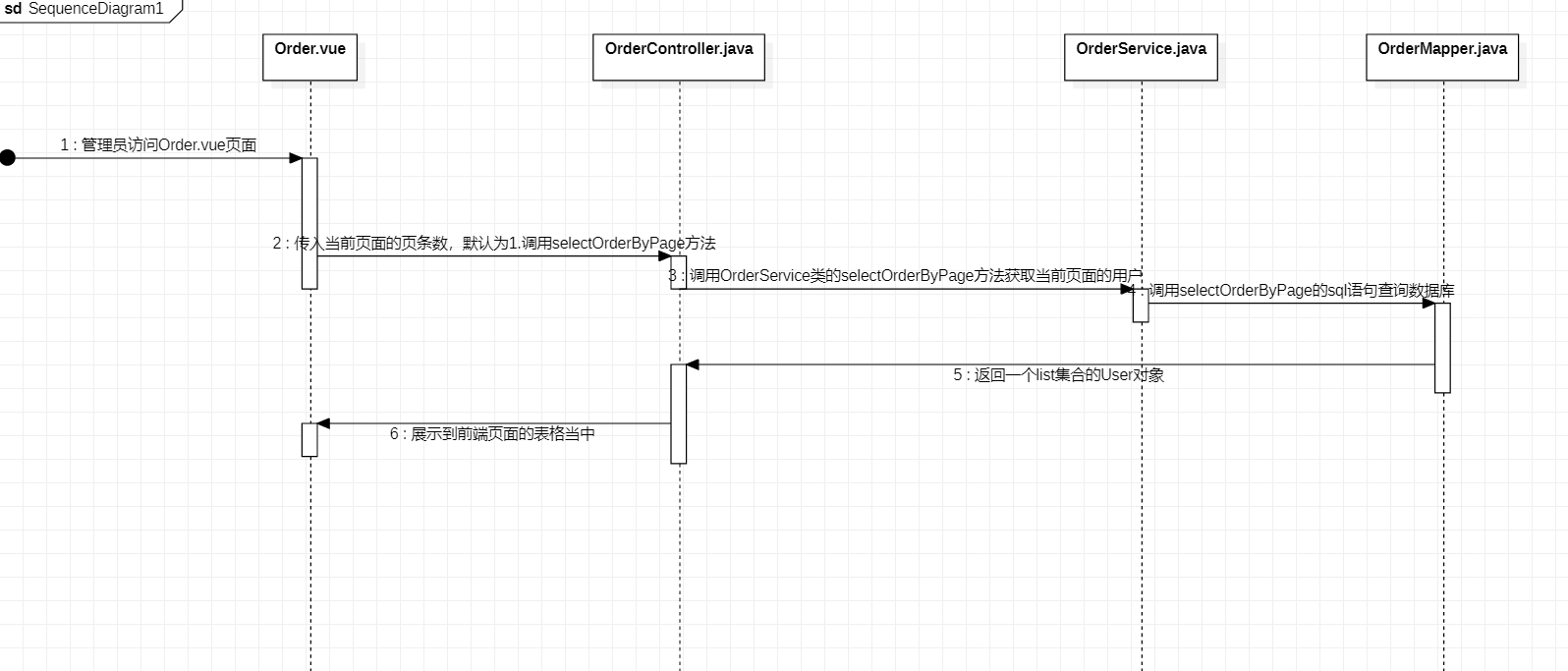
3.6.2：商品销售情况（今日购买、本周购买、总购买）、3.6.3：今年的每个月分的销售统计

与3.6.1类似

## 3.7 后台订单模块设计说明

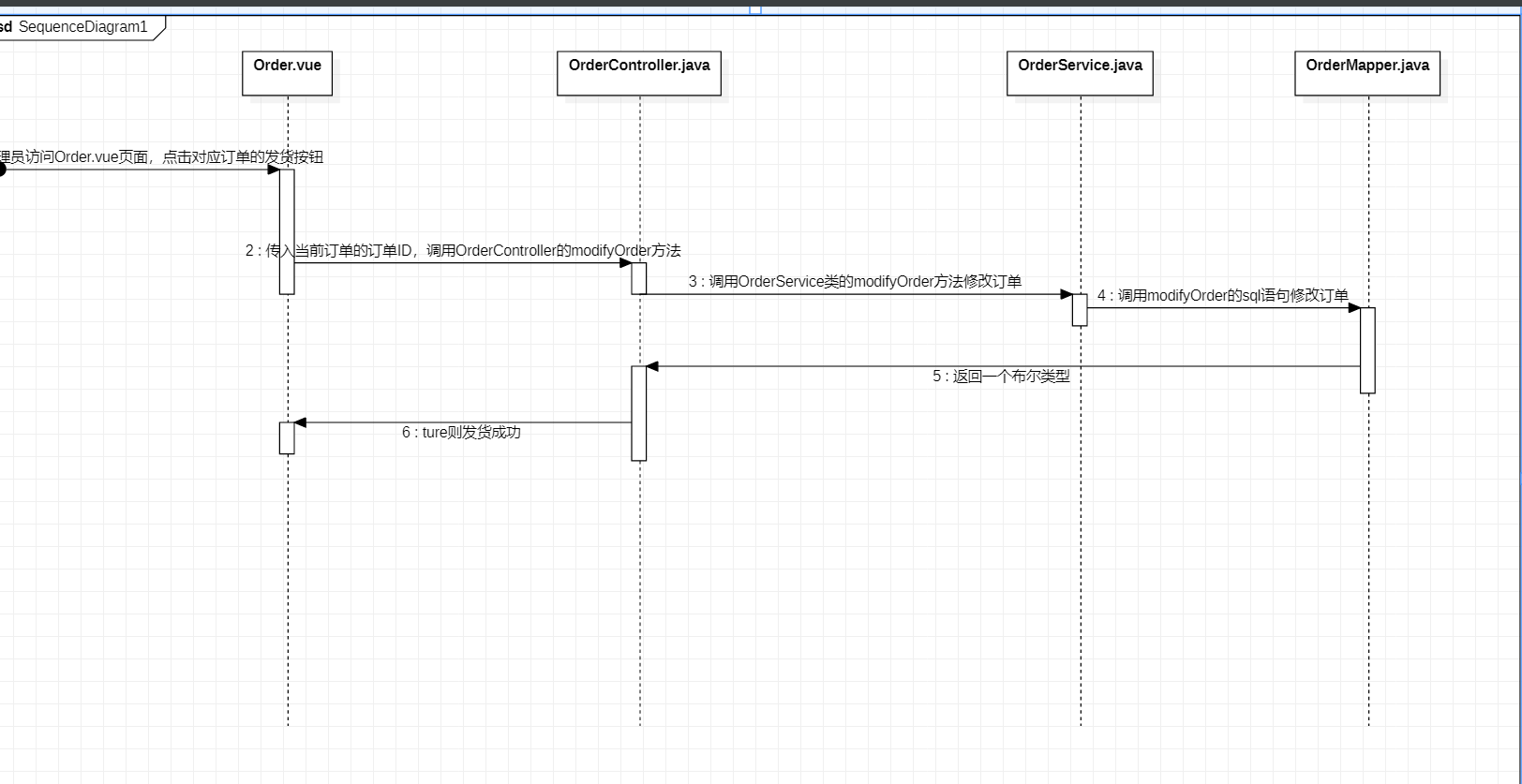
3.7.1、分页展示所有的订单

管理员访问order.vue的时候，在此之前就会向数据库发出请求，获取到对应的所有用户，其中它传递了一个数值为当前的页面，默认为第一页，之后传递给OrderController的selectOrderByPage()方法，然后再调用OrderService方法中的selectOrderByPage方法，在这里，它会将传来的页面进行变化，变成(pageNum-1)\*10之后，传递给OrderMapper，这是因为在OrderMapper当中的sql语句为limit 起始条数，读取条数，当读取到第二页时就是从第十一条开始的了（这里我规定每一页有十条数据）。查询之后返回一个Order对象的list集合。在这之后，会对你传过来的订单状态进行判断，不同的订单状态会展示不同的按钮



3.7.2、订单发货

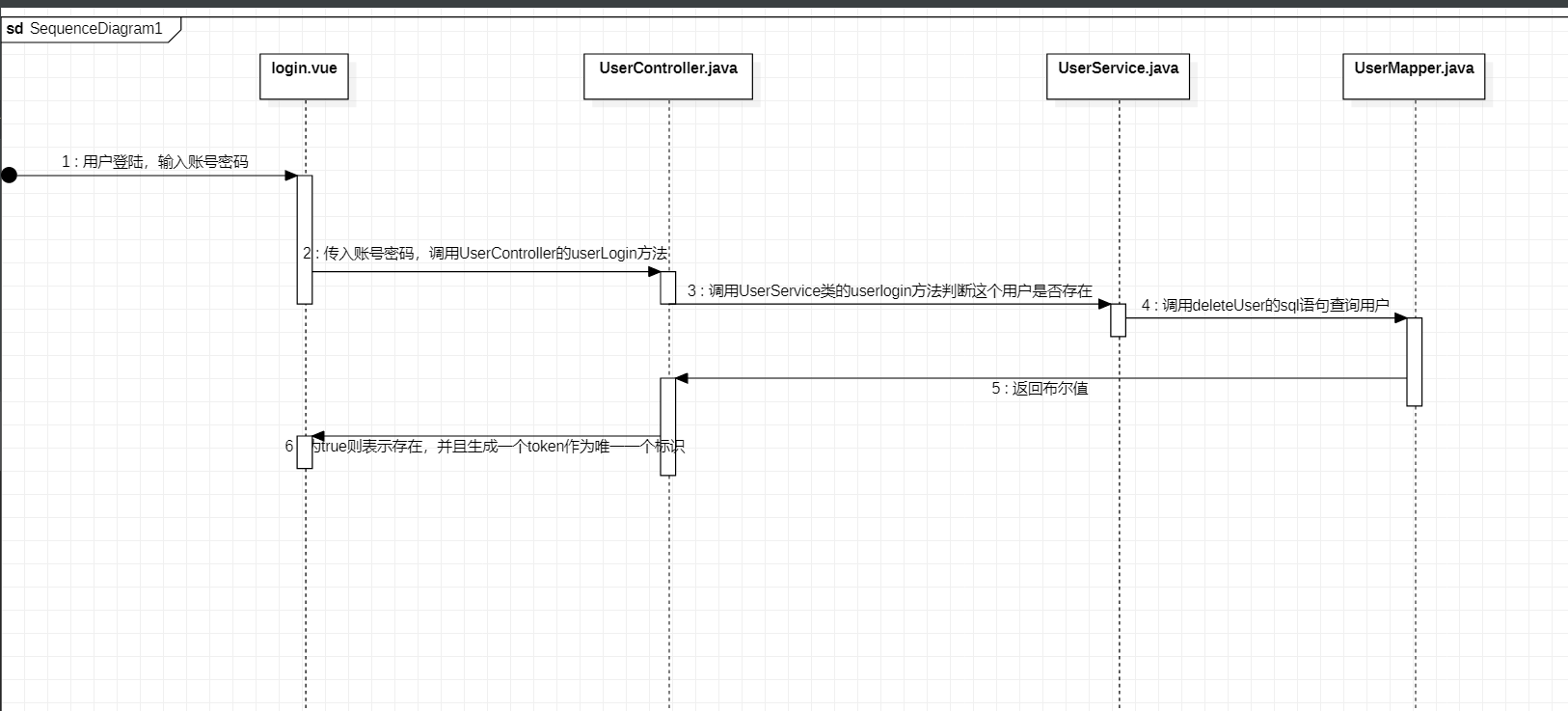
当订单的状态为未发货时，管理员可以点击发货给用户发货。这里是调用了OrderController的modifyOrder方法，在modifyOrder方法中调用了OrderService业务层的modifyOrder方法，进而调用OrderMapper的modifyOrder的sql语句。这里修改订单，仅仅是将订单的状态修改为已发货状态



## 3.8 前台登陆注册模块设计说明

3.8.1、用户登陆

用户输入自己的账号密码，然后向UserController的userLogin的方法发出请求，进而调用UserService的userLogin方法，在该方法中它执行了UserMapper的loginsql语句，实现查询用户的功能，如果查询到的为true，则将对应的手机号、用户名作为一个属性，将其转化成jwt的形式已token1的形式传给前端，这个token是这个用户的唯一标识，当你修改了你的token之后，用户就需要重新登陆。而且你的这个token里面的电话号码作为一个用户的唯一标识，会在后端进行请求的时候将其取出来做为一个条件



3.8.2、用户注册

与后台的添加用户的方式一样

## 3.9 前台热销产品设计说明

3.9.1、获取商品热销

与后台的获取商品总销售量的方式类似

## 3.10 前台商品分类模块设计说明

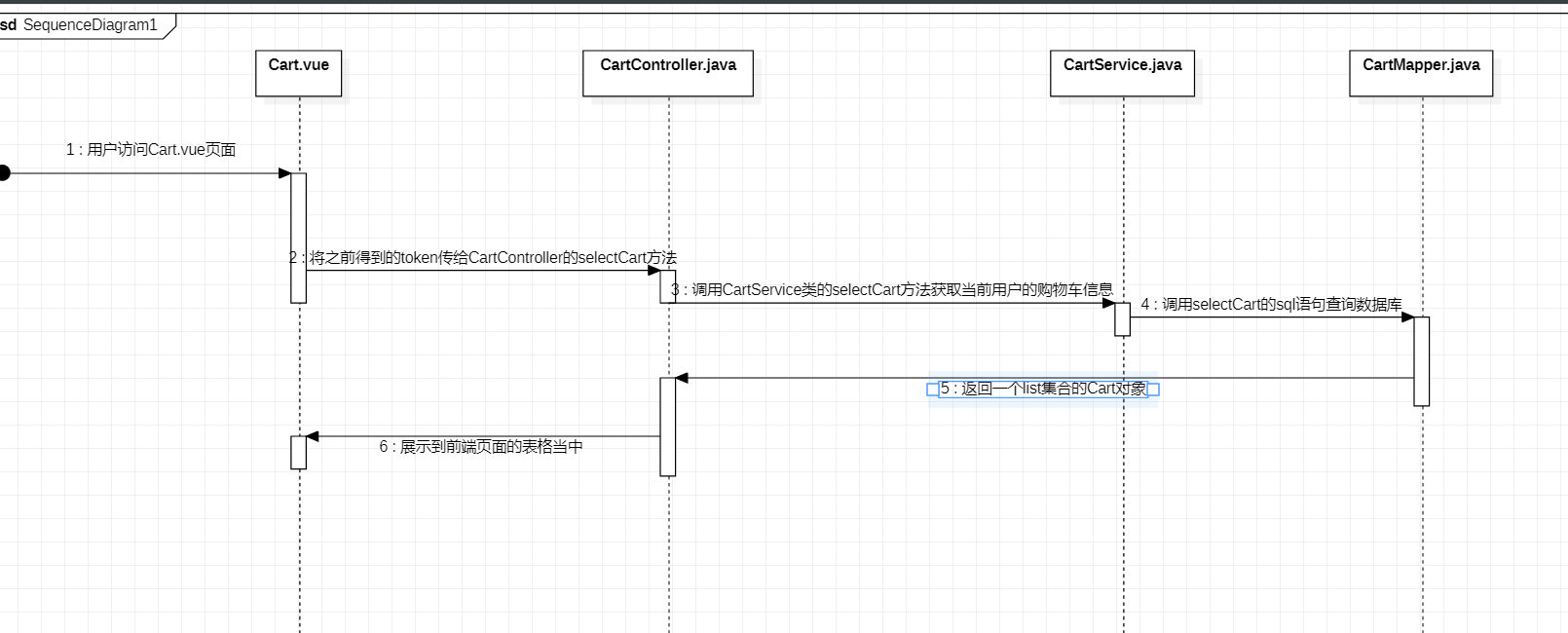
3.10.1、获取商品分类

与后台的获取商品分类以及内容的方式类似

## 3.11 前台购物车模块设计说明

3.11.1、获取购物车内容

用户登录成功之后，会将得到的token传给CartController类的selectCart方法，selectCart方法首先会将得到的token进行解密，如果被修改导致解密不出则会直接返回false，然后前端就会要求重新登陆。如果解密出来就会将其中的电话号码作为唯一id传给CartService业务层的selectCart方法，在业务层，它主要是执行CartMapper的selectCartByphone的sql语句查询当前用户的所有购物车信息，查询之后已list的cart返回给前端展示



3.11.2、前台添加购物车

当用户选择合适的商品之后就会加入购物车，此时将数值传递给CartController的addCart方法中，然后调用CartService的addCart方法中，在执行sql是数据库之前会进行判断，判断当前的商品是否存在，如果存在则直接在原有的数量上进行增加，否则则会插入新的数据内容。同时在购物车里面也可以修改商品的数量，当你添加一个时就会调用addCart方法，以此来修改对应的数量以及对应的总金额

## 

## 3.12 前台订单模块设计说明

3.12.1、前台订单管理

当购买成功之后就会生成订单信息，通过查看myOrder.vue可以查看订单详情。在订单中存在订单状态，购买成功之后是未发货状态，这就需要后台管理员进行发货操作，具体的业务实现流程与后台的订单管理流程类似

# 4 数据库设计

## 4.1 数据库环境说明

数据库系统：5.7.26

设计工具：Navicat for MySQL 2013

编程工具：idea

## 4.2 数据库的命名规则

根据各个模块的各个模块，明确设计此网上蛋糕商城项目都需要哪些实体对象，根据实体对象之间的关系设计数据库。在设计是时刻谨记数据库名称能够直观表示实体类型和属性之间的关联关系。

## 4.3 逻辑结构设计

E-R图

关系模型

## 4.4 物理结构设计

4.4.1 数据库表

表4.1 表汇总

|  |  |
| --- | --- |
| 表 名 | 功 能 说 明 |
| cart | 购物车表 存储购物车数据 |
| user | 用户表 存储用户信息数据 |
| goods | 蛋糕表 存储蛋糕商品数据 |
| goodsCategory | 分类表 存储分类数据 |
| order | 订单表 存储订单数据 |
| statistic | 订单明细表 存储订单明细数据 |

表4.2 购物车表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | cart | | | | | |
| 数据库用户 | |  | | | | | |
| 主 键 | | telephone | | | | | |
| 其他排序字段 | |  | | | | | |
| 索引字段 | |  | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | telephone | int | N | Y |  |  | 自增. 主键 |
| 2 | goodsPriceSum | int | N | N |  |  | 商品总金额 |
| 3 | goodsId | int | N | N |  |  | 商品id 外键 |
| 4 | goodsNum | int | N | N |  |  | 数量 |
| sql脚本 | | create table cart  (  telephone varchar(11) comment '用户电话号码',  goodsID int comment '商品ID' ,  goodsNum int comment '商品数量' ,  goodsPriceSum double comment '商品价格'  # foreign key (telephone) references user (telephone),  # foreign key (goodsID) references goods (goodsID)  ); | | | | | |
| 备注 | |  | | | | | |

表4.3 分类表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | goodsCategory | | | | | |
| 数据库用户 | |  | | | | | |
| 主 键 | | id | | | | | |
| 其他排序字段 | |  | | | | | |
| 索引字段 | | fatherId | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | id | int | N | Y |  |  | 自增. 主键 |
| 2 | name | varchar(100) | Y | Y |  | null | 分类名 |
| 3 | fatherId | int | Y | N |  | null | 父id, 外键 |
| 4 | type | int | Y | N |  | null | 几级分类 |
| sql脚本 | | create table cake.category  (  id int auto\_increment comment '主键'  primary key,  name varchar(100) null comment '分类名称',  fatherId int null comment '父级id',  type int null comment '几级分类（1代表一级分类， 2代表二级分类）'  )  comment '分类表' collate = utf8mb4\_unicode\_ci;  create index category\_category\_id\_fk  on cake.category (fatherId); | | | | | |
| 备注 | |  | | | | | |

表4.4 订单表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | order | | | | | |
| 数据库用户 | |  | | | | | |
| 主 键 | | orderID | | | | | |
| 其他排序字段 | |  | | | | | |
| 索引字段 | | telephone | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | OrderiId | int | N | Y |  |  | 自增. 主键 |
| 2 | orderTelephone | varchar(255) | Y | Y |  | null | 订单号码 |
| 3 | orderPrice | int | Y | N |  | null | 购买价格 |
| 4 | orderStatus | int | Y | N |  | null | 订单状态 |
| 5 | orderAddress | varchar(255) | Y | N |  | null | 收货地址 |
| 6 | orderContents | varchar(255) | Y | N |  | null | 收货人 |
| 7 | orderTime | varchar(20) | Y | N |  | null | 下单时间 |
| sql脚本 | | create table `order`  (  orderID int comment '订单编号' auto\_increment primary key,  orderTelephone varchar(11) comment '电话号码',  orderPrice double comment '订单价格' not null,  orderStatus varchar(20) comment '订单状态' not null,  orderContents varchar(100) comment '内容' not null,  orderTime timestamp comment '下单时间',  orderAddress varchar(50) comment '发货地址',  foreign key (orderID) references statistics (orderID)  );  alter table `order`  auto\_increment = 1000; | | | | | |
| 备注 | |  | | | | | |

表4.5 订单明细表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | Statistic | | | | | |
| 数据库用户 | |  | | | | | |
| 主 键 | | id | | | | | |
| 其他排序字段 | |  | | | | | |
| 索引字段 | | orderID,goodsID | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | id | int | N | Y |  |  | 自增. 主键 |
| 2 | goodsID | varchar(100) | Y | N |  | null | 商品id 外键 |
| 3 | orderID | int | Y | N |  | null | 订单id 外键 |
| 4 | goodsNum | int | Y | N |  | null | 数量 |
| 5 | buyTime | Varchar(100) | Y | N |  | null | 购买时间 |
| sql脚本 | | create table statistics  (  id int comment '统计编号' auto\_increment primary key,  orderID int comment '订单编号',  goodsID int comment '商品ID',  goodsNum double comment '商品数量' not null,  buyTime timestamp comment '购买时间'  ); | | | | | |
| 备注 | |  | | | | | |

表4.6 产品表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | goods | | | | | |
| 数据库用户 | |  | | | | | |
| 主 键 | | goodsId | | | | | |
| 其他排序字段 | |  | | | | | |
| 索引字段 | |  | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | GoodsId | int | N | Y |  |  | 自增. 主键 |
| 2 | GoodsName | varchar(50) | Y | Y |  | null | 商品名 |
| 3 | goodsPrice | float | Y | N |  | null | 商品价格 |
| 4 | goodsCategory | varchar(50) | Y | N |  | null | 商品分类 |
| 5 | GoodsExist | enum(‘0’,‘1’) | Y | N |  | null | 商品状态 |
| 6 | goodsLogo | varchar(255) | Y | N |  | null | 商品图片url |
| 7 | goodsDescription | varchar(255) | Y | N |  | null | 商品描述 |
| 8 | goodsSale | int | Y | N |  | null | 销售情况 |
| sql脚本 | | create table goods  (  goodsID int comment '商品ID' auto\_increment primary key,  goodsName varchar(30) comment '商品名称' not null,  goodsPrice double comment '单价' not null,  goodsCategory varchar(50) comment '所属类别' not null,  goodsLogo varchar(100) comment '商品图片' not null,  goodsExist boolean comment '商品是否上线',  goodsDescription varchar(100) comment '商品描述',  # goodssize double comment '商品大小',  goodssale int comment '商品销售情况'  ); | | | | | |
| 备注 | |  | | | | | |

表4.7 用户表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 表 名 | | user | | | | | |
| 数据库用户 | |  | | | | | |
| 主 键 | | id | | | | | |
| 其他排序字段 | |  | | | | | |
| 索引字段 | |  | | | | | |
| 序号 | 字段名称 | 数据类型（精度范围） | 允许为空Y/N | 唯一Y/N | 区别度 | 默认值 | 约束条件/说明 |
| 1 | id | int | N | Y |  |  | 自增. 主键 |
| 2 | username | varchar(20) | N | Y |  |  | 账号 |
| 3 | password | varchar(20) | N | N |  |  | 密码 |
| 4 | name | varchar(20) | Y | N |  | null | 姓名 |
| 5 | gender | enum ('0', '1', '-1') | Y | N |  | null | 性别 |
| 6 | address | varchar(50) | Y | N |  | null | 地址 |
| sql脚本 | | create table cake.user  (  id int auto\_increment comment '主键id'  primary key,  username varchar(20) not null comment '账号',  password varchar(20) not null comment '密码',  name varchar(20) null comment '姓名',  gender enum ('0', '1', '-1') null comment '性别0女1男-1秘密',  address varchar(50) null comment '地址', | | | | | |
| 备注 | |  | | | | | |

## 4.5 安全性设计

4.5.1 角色与权限

=表4.3 角色与权限

|  |  |  |
| --- | --- | --- |
| 角色 | 可以访问的表与列 | 操作权限 |
| 游客 |  |  |
|  |  |
|  |  |
| 普通用户 |  |  |
|  |  |
|  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 5 测试用例设计

表5.1 登录-注册测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 登录 | yl-001 | 输入‘用户名’和‘密码’ | admin，123456 | 登录成功 | 通过 |
| yl-002 | admin，1234567 | 登录失败 | 通过 |
| yl-003 | admin1，123456 | 登录失败 | 通过 |
| yl-004 | admin,空 | 登录失败 | 通过 |
| yl-005 | 空,123456 | 登录失败 | 通过 |
| yl-006 | admin,一个空格 | 登录失败 | 通过 |
| yl-007 | 一个空格，123456 | 登录失败 | 通过 |
| yl-008 | admin，密码长度超过32 | 登录失败 | 通过 |
| yl-009 | 用户名超过36，123456 | 登录失败 | 通过 |
| 注册 | yl-010 | 用户名，密码，确认密码，真是姓名，性别，出生日期，地址，邮政编码，电话，email，密保问题，密保答案，验证码 | Tony,123456,123456,test1,男，1991-10-10，长沙，410014，85577424，505@qq.com，我是谁，test1. | 创建成功 | 通过 |
|  |  |  |  |  |  |

表5.2 商品管理测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 登录 | yl-001 | 输入‘商品名’、‘数量’和‘价格’ | admin，123456，123 | 登录成功 | 通过 |
| yl-002 | admin，1234567，1234 | 登录失败 | 通过 |
| yl-003 | admin1，123456，123 | 登录失败 | 通过 |
| yl-004 | admin,空，123 | 登录失败 | 通过 |
| yl-005 | 空,123456，123 | 登录失败 | 通过 |
| yl-006 | admin,一个空格，123 | 登录失败 | 通过 |
| yl-007 | 一个空格，123456，123 | 登录失败 | 通过 |
| yl-008 | admin，密码长度超过8，123 | 登录失败 | 通过 |
| yl-009 | 商品名超过8，123456，123 | 登录失败 | 通过 |

# 附录

## 附录1 程序运行结果



图1 统计界面



图2 用户界面

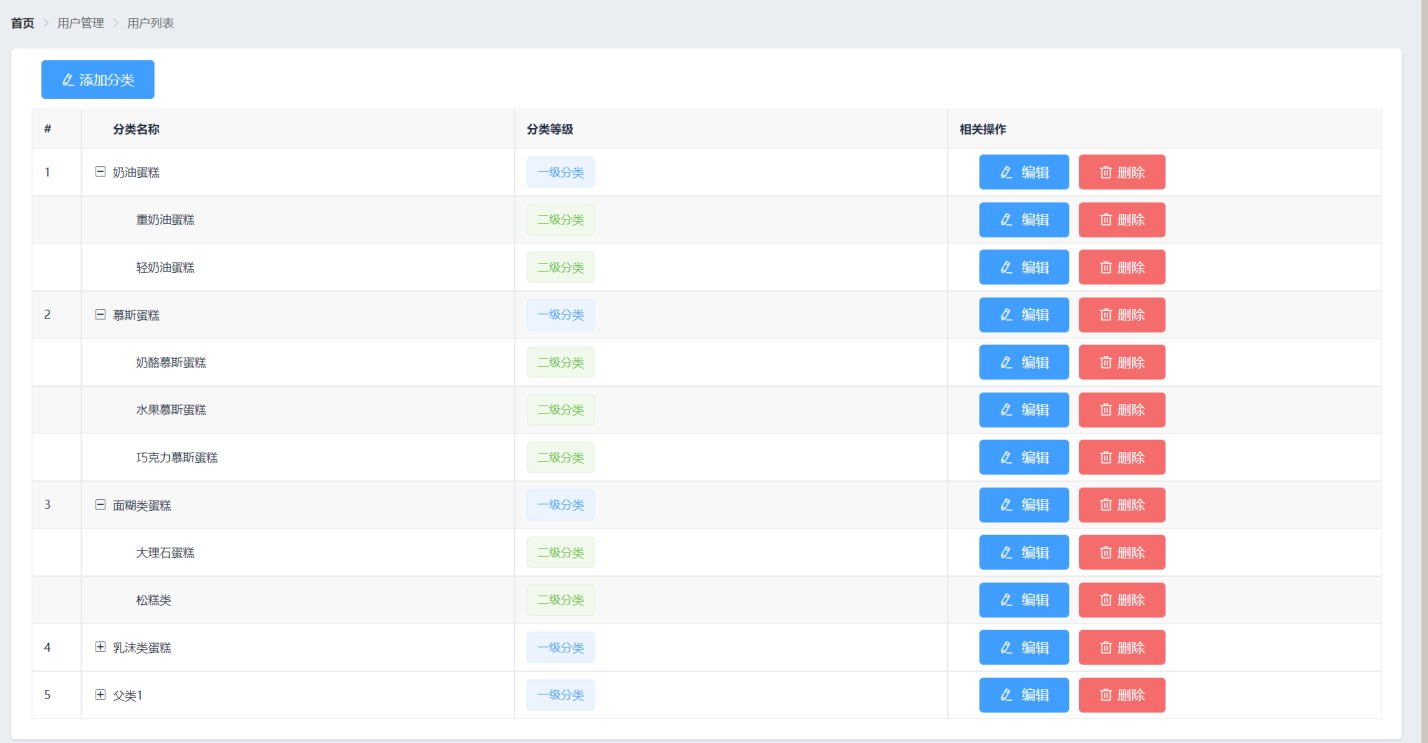


图3 商品分类界面

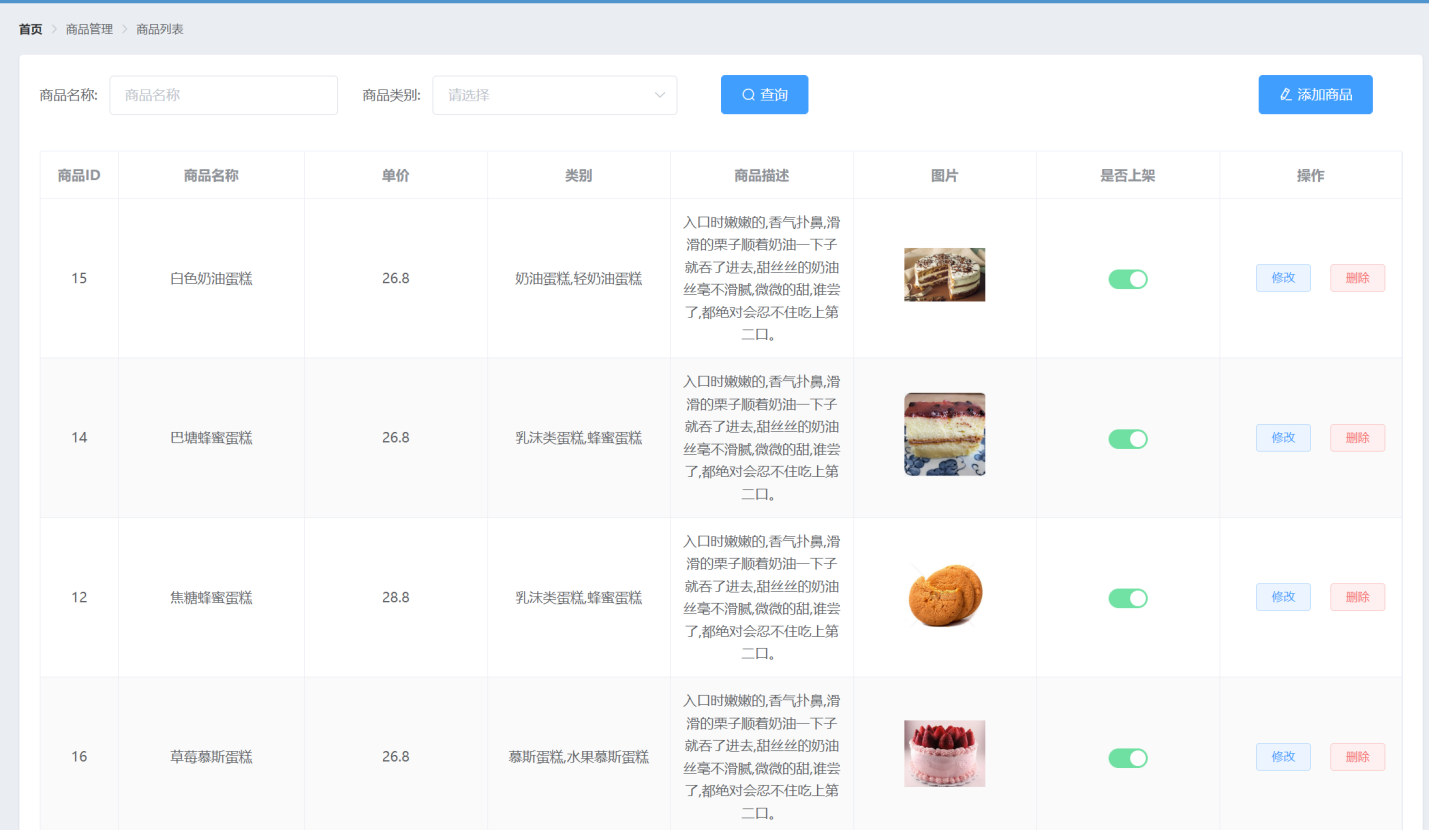


图4 商品界面

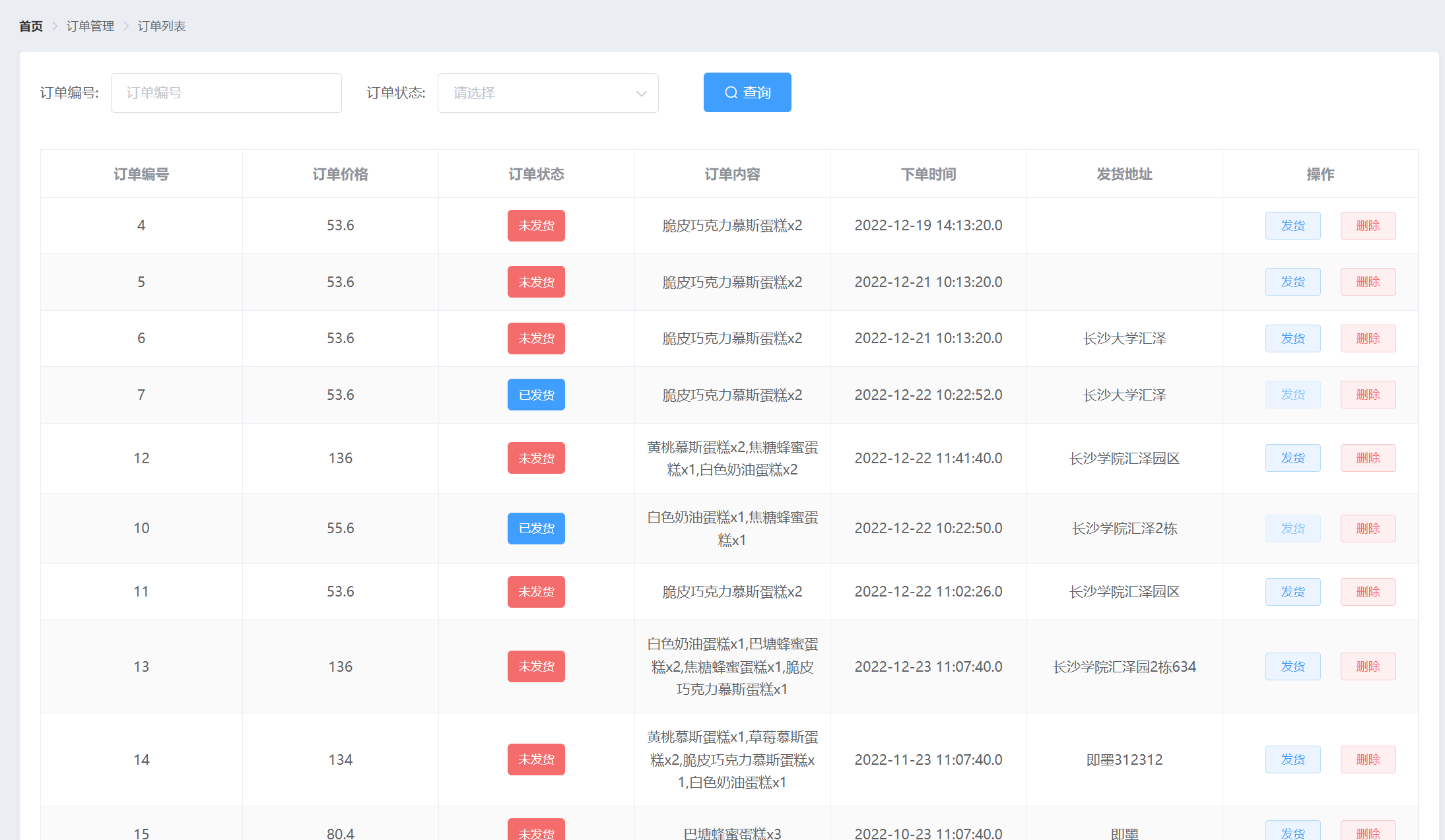


图5 订单界面

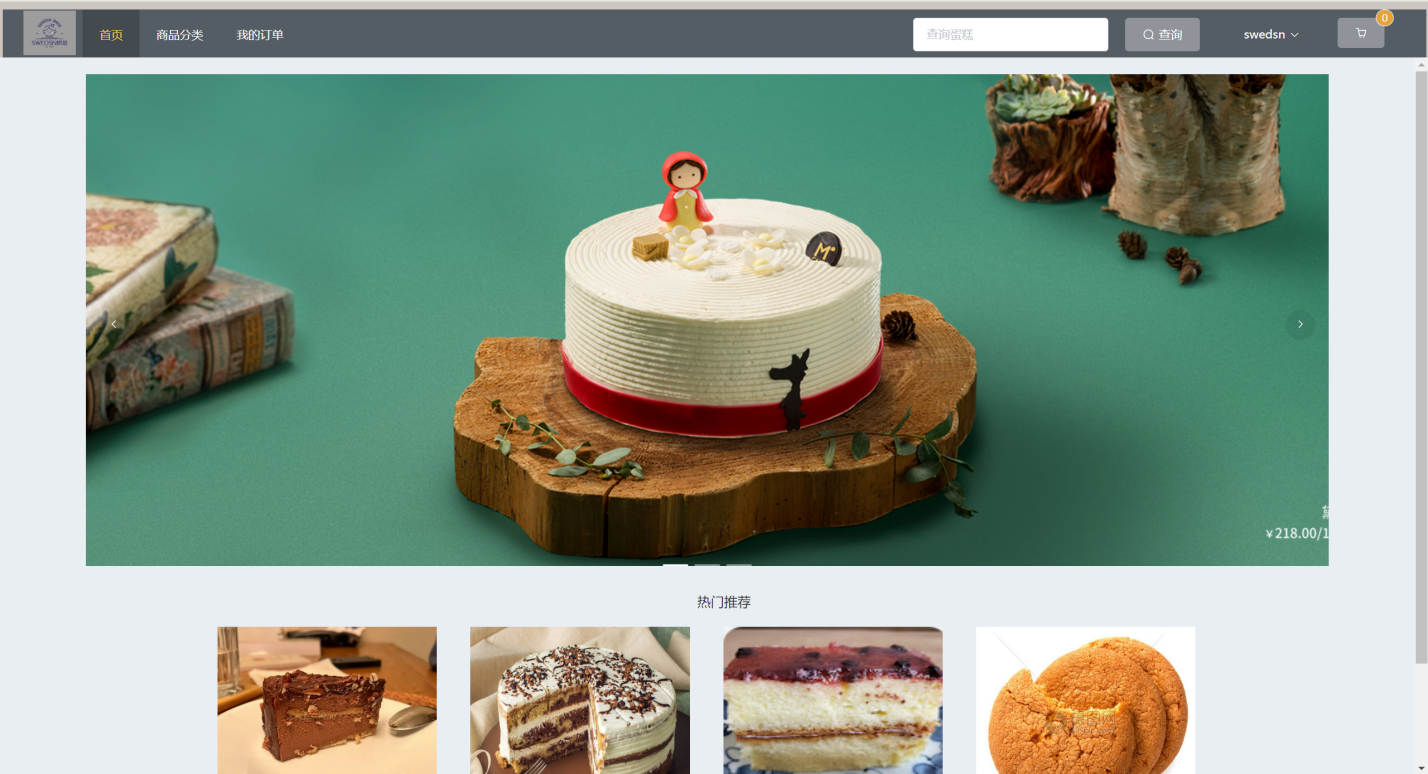


图6 前台首页界面

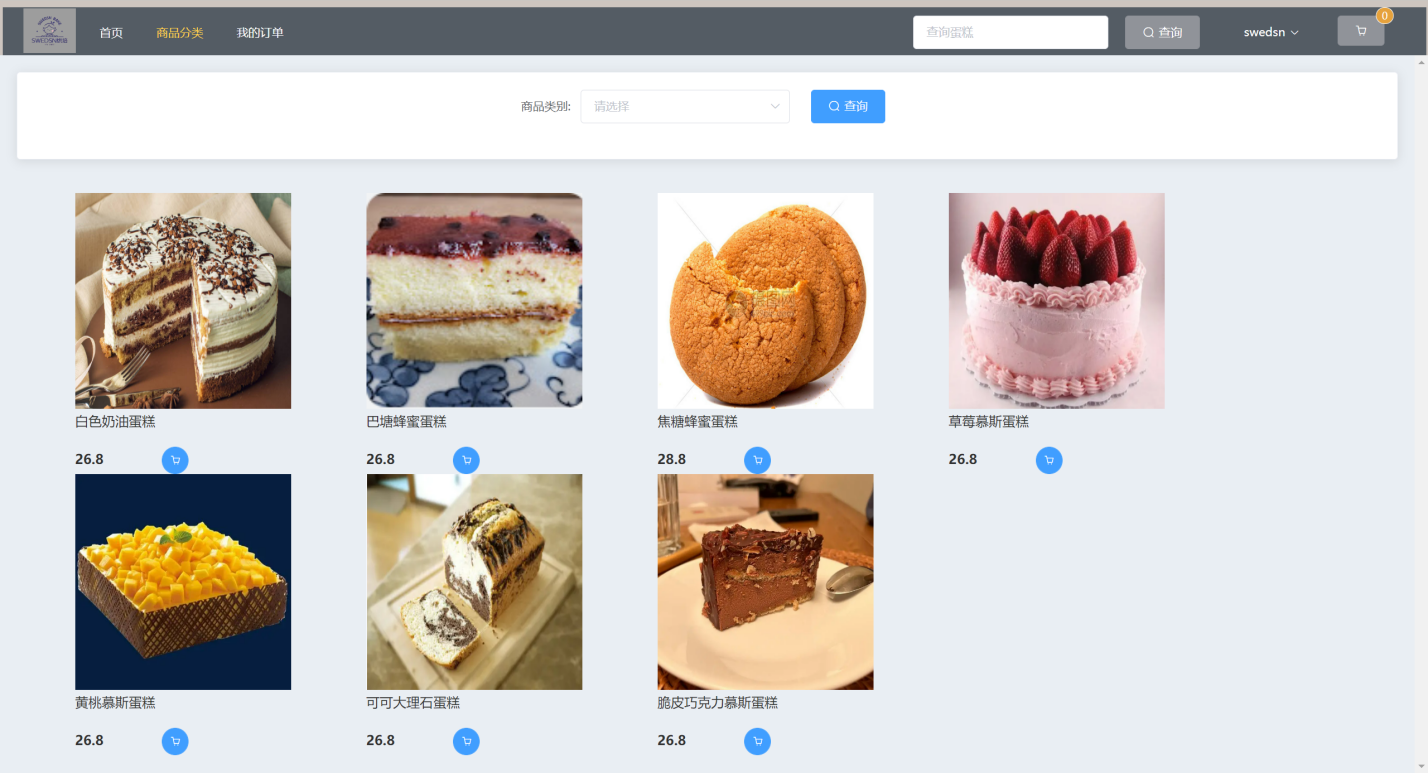


图7 前台分类界面



图8 前台用户订单界面

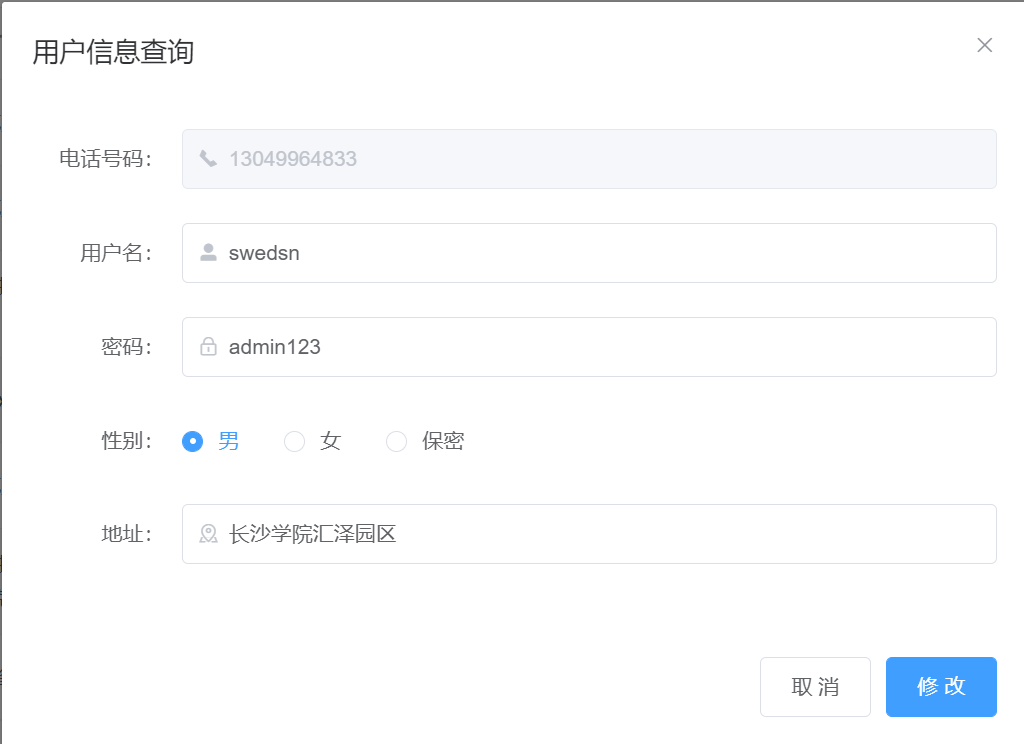


图9 前台用户基本信息界面



图10 前台购物车界面

## 附录2 程序主要源代码

CartController.java

package com.db.controller;  
  
import com.alibaba.fastjson.JSON;  
import com.alibaba.fastjson.JSONObject;  
import com.auth0.jwt.JWT;  
import com.auth0.jwt.interfaces.DecodedJWT;  
import com.db.entiy.Cart;  
import com.db.service.CartService;  
import com.db.service.GoodsService;  
import com.db.service.StatisticService;  
import com.db.service.impl.CartImpl;  
import com.db.service.impl.GoodsImpl;  
import com.db.service.impl.StatisticImpl;  
import com.db.util.JwtUtils;  
import org.springframework.web.bind.annotation.CrossOrigin;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.List;  
import java.util.Map;  
  
/\*\*  
 \* @author swedsn  
 \* @version 1.0  
 \* @date 2022-12-19 11:56  
 \*/  
@RestController  
@CrossOrigin  
public class CartController {  
  
 CartService cartService = new CartImpl();  
 StatisticService statisticService = new StatisticImpl();  
 @GetMapping("/addCart")  
 public String addCart(Cart cart) {  
 cart.setGoodsNum(1);  
 *//cart.setGoodsPriceSum(cart.getGoodsPriceSum());* System.out.println(cart.getGoodsPriceSum());  
 Boolean flag = cartService.addCart(cart);  
 return JSON.*toJSONString*(flag);  
 }  
  
 @GetMapping("/selectCart")  
 public JSONObject selectCart(String token) {  
 JSONObject str = new JSONObject();  
 DecodedJWT verify = JwtUtils.*getClaimsByToken*(token);  
 *//System.out.println("伪造的："+verify);* String telephone = verify.getClaim("telephone").asString();  
 List<Map<Object, Object>> carts = cartService.selectCart(telephone);  
 Double goodsAllPrice = 0.0;  
 for (Map<Object, Object> cart: carts){  
 goodsAllPrice = goodsAllPrice + (Double) cart.get("goodsPriceSum");  
 }  
 str.put("carts", carts);  
 str.put("goodsAllPrice", goodsAllPrice);  
 str.put("cartCount", carts.size());  
 return str;  
 }  
  
  
 @GetMapping("/modifyCart")  
 public String modifyCart(String telephone, String goodsName, int goodsNum){  
 Boolean flag = cartService.modifyCart(telephone, goodsName, goodsNum);  
 return JSON.*toJSONString*(flag);  
 }  
  
 @GetMapping("/deleteCart")  
 public String deleteCart(String token, String goodsName) {  
 JSONObject str = new JSONObject();  
 DecodedJWT verify = JwtUtils.*getClaimsByToken*(token);  
 *//System.out.println("伪造的："+verify);* if (verify == null){  
 return JSON.*toJSONString*(false);  
 }  
 String telephone = verify.getClaim("telephone").asString();  
 Boolean flag = cartService.deleteCart(telephone, goodsName);  
 return JSON.*toJSONString*(flag);  
 }  
  
}

CartMapper.java

package com.db.mapper;  
  
import com.db.entiy.Cart;  
import org.apache.ibatis.annotations.Delete;  
import org.apache.ibatis.annotations.Insert;  
import org.apache.ibatis.annotations.Select;  
import org.apache.ibatis.annotations.Update;  
  
import java.util.List;  
  
/\*\*  
 \* @author swedsn  
 \* @version 1.0  
 \* @date 2022-12-19 11:58  
 \*/  
public interface CartMapper {  
  
 @Insert("insert into cart(telephone, goodsID, goodsNum, goodsPriceSum) value(#{telephone}, #{goodsID},#{goodsNum},#{goodsPriceSum})")  
 boolean addCart(Cart cart);  
  
 @Select("select \* from cart where telephone=#{telephone}")  
 List<Cart> selectCart(String telephone);  
  
 @Select("select \* from cart where telephone=#{telephone} and goodsID = #{goodsID}")  
 Cart selectCartByUser(String telephone, int goodsID);  
  
 @Update("update cart set goodsNum = #{goodsNum},goodsPriceSum = #{goodsPriceSum} where telephone = #{telephone} and goodsID = #{goodsID}")  
 boolean modifyCart(int goodsNum, String telephone, int goodsID, Double goodsPriceSum);  
  
  
 @Delete("delete from cart where telephone=#{telephone} and goodsID = #{goodsID}")  
 Boolean deletCart(String telephone, int goodsID);  
}

GoodsCategory.java

package com.db.service.impl;  
  
import com.db.entiy.GoodsCategory;  
import com.db.mapper.GoodsCategoryMapper;  
import com.db.service.GoodsCategoryService;  
import com.db.util.SqlSessionFactoryUtils;  
import org.apache.ibatis.session.SqlSession;  
import org.apache.ibatis.session.SqlSessionFactory;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
  
/\*\*  
 \* @author swedsn  
 \* @version 1.0  
 \* @date 2022-12-08 21:56  
 \*/  
public class GoodsCategoryImpl implements GoodsCategoryService {  
 SqlSessionFactory sqlSessionFactory = SqlSessionFactoryUtils.*getSqlSessionFactory*();  
  
 *// 获取到父类菜单* public ArrayList<Object> getParents() {  
 *//2、 获取SqlSession对象，用它来执行sql* SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* GoodsCategoryMapper goodsCategoryMapper = sqlSession.getMapper(GoodsCategoryMapper.class);  
 *// 4、调用方法* List<GoodsCategory> goodsCategories = goodsCategoryMapper.selectCategories();  
  
 ArrayList<Object> categories = new ArrayList<>();  
 for (GoodsCategory parentCategory : goodsCategories) {  
 *// 得到一级菜单* if (parentCategory.isParent()) {  
 categories.add(parentCategory);  
 }  
 }  
 return categories;  
 }  
  
 *// 得到子类菜单* public List<GoodsCategory> getChildren(List<GoodsCategory> goodsCategories, int id) {  
 List<GoodsCategory> chilrenCategories = new ArrayList<>();  
 for (GoodsCategory goodsCategory : goodsCategories) {  
 if (goodsCategory.getParentID() == id) {  
 chilrenCategories.add(goodsCategory);  
 }  
 }  
 *//System.out.println("id: " + id + "chilrenCategories" + JSON.toJSONString(chilrenCategories));* return chilrenCategories;  
 }  
  
  
 @Override  
 public ArrayList<Object> getCategories() {  
 *//2、 获取SqlSession对象，用它来执行sql* SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* GoodsCategoryMapper goodsCategoryMapper = sqlSession.getMapper(GoodsCategoryMapper.class);  
 *// 4、调用方法* List<GoodsCategory> goodsCategories = goodsCategoryMapper.selectCategories();  
  
 ArrayList<Object> categories = new ArrayList<>();  
 for (GoodsCategory parentCategory : goodsCategories) {  
 *// 得到一级菜单* if (parentCategory.isParent()) {  
 *// 获取到二级菜单* List<GoodsCategory> children = null;  
 for (GoodsCategory chilrenCategory : goodsCategories) {  
 children = getChildren(goodsCategories, parentCategory.getCategoryID());  
 }  
 HashMap<Object, Object> parentMap = new HashMap<>();  
 parentMap.put("categoryID", parentCategory.getCategoryID());  
 parentMap.put("categoryName", parentCategory.getCategoryName());  
 parentMap.put("isparent", parentCategory.getParentID());  
 parentMap.put("children", children);  
 *//System.out.println("father：" + JSON.toJSONString(parentMap));* categories.add(parentMap);  
 }  
  
 }  
  
 sqlSession.close();  
 return categories;  
 }  
  
 @Override  
 public GoodsCategory selectCategoryByCategoryID(int categoryID) {  
 *//2、 获取SqlSession对象，用它来执行sql* SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* GoodsCategoryMapper goodsCategoryMapper = sqlSession.getMapper(GoodsCategoryMapper.class);  
  
 GoodsCategory goodsCategory = goodsCategoryMapper.selectCategoryByCategoryID(categoryID);  
 sqlSession.close();  
 return goodsCategory;  
 }  
  
 @Override  
 public void addCategory(String categoryName, int parentID) {  
 *//2、 获取SqlSession对象，用它来执行sql* SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* GoodsCategoryMapper goodsCategoryMapper = sqlSession.getMapper(GoodsCategoryMapper.class);  
 *// 4、调用方法* GoodsCategory goodsCategory = new GoodsCategory();  
 goodsCategory.setParent(parentID == -1);  
 goodsCategory.setParentID(parentID);  
 goodsCategory.setCategoryName(categoryName);  
 goodsCategoryMapper.addCategory(goodsCategory);  
 sqlSession.close();  
 }  
  
 @Override  
 public void deleteCategory(int categoryID) {  
 *//2、 获取SqlSession对象，用它来执行sql* SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* GoodsCategoryMapper goodsCategoryMapper = sqlSession.getMapper(GoodsCategoryMapper.class);  
  
 goodsCategoryMapper.deleteCategory(categoryID);  
 sqlSession.close();  
  
 }  
  
 @Override  
 public void modifyCategory(GoodsCategory goodsCategory) {  
 *//2、 获取SqlSession对象，用它来执行sql* SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* GoodsCategoryMapper goodsCategoryMapper = sqlSession.getMapper(GoodsCategoryMapper.class);  
  
 goodsCategoryMapper.modifyCategory(goodsCategory);  
 sqlSession.close();  
 }  
}

JWTUtil.java

package com.db.util;  
  
import com.auth0.jwt.JWT;  
import com.auth0.jwt.JWTVerifier;  
import com.auth0.jwt.algorithms.Algorithm;  
import com.auth0.jwt.interfaces.DecodedJWT;  
import com.db.entiy.User;  
  
import java.security.SignatureException;  
import java.util.\*;  
  
/\*\*  
 \* @author swedsn  
 \* @version 1.0  
 \* @date 2022-12-03 17:29  
 \*/  
public class JwtUtils {  
 private static long expire = 604800;  
  
 *// 32为密钥* private static String secret = "248abd887c7e40ce84b25ef253e6ff8d";  
  
 private static Map<String, User> tokenMap = new HashMap<>();  
 public static String generateToken(User user){  
 Calendar instance = Calendar.*getInstance*();  
 instance.add(Calendar.DATE, 30); *//默认七天过去* String token = JWT.*create*()  
 .withClaim("telephone", user.getTelephone())  
 .withClaim("username", user.getUsername())  
 .withExpiresAt(instance.getTime())  
 .sign(Algorithm.*HMAC256*(secret));  
 return token;  
  
 }  
  
 public static DecodedJWT getClaimsByToken(String token){  
 *// 创建验证对象* JWTVerifier build = JWT.*require*(Algorithm.*HMAC256*(secret)).build();  
  
  
 try{  
 DecodedJWT verify = build.verify(token);  
 *//System.out.println(verify.getClaim("telephone").asString());* return verify;  
 }catch (Exception e){  
 return null;  
 }  
  
  
 }  
}

SqlSesssionFactoryUtil.java

package com.db.util;  
  
import org.apache.ibatis.io.Resources;  
import org.apache.ibatis.session.SqlSessionFactory;  
import org.apache.ibatis.session.SqlSessionFactoryBuilder;  
  
import java.io.IOException;  
import java.io.InputStream;  
  
/\*\*  
 \* @author swedsn  
 \* @version 1.0  
 \* @date 2022-10-19 16:04  
 \*/  
public class SqlSessionFactoryUtils {  
  
 private static SqlSessionFactory sqlSessionFactory;  
 static {  
 *// 静态代码块会随着类的加载而自动执行，且只执行一次  
 // 加载mybatis的核心配置文件，获取 SqlSessionFactory：* String resource = "mybatis-config.xml";  
 InputStream inputStream = null;  
 try {  
 inputStream = Resources.*getResourceAsStream*(resource);  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  
  
 }  
 public static SqlSessionFactory getSqlSessionFactory(){  
 return sqlSessionFactory;  
 }  
}

OrderImpl.java

package com.db.service.impl;  
  
import com.db.entiy.Order;  
import com.db.entiy.Statistic;  
import com.db.mapper.CartMapper;  
import com.db.mapper.OrderMapper;  
import com.db.mapper.StatisticMapper;  
import com.db.service.CartService;  
import com.db.service.OrderService;  
import com.db.util.SqlSessionFactoryUtils;  
import org.apache.ibatis.session.SqlSession;  
import org.apache.ibatis.session.SqlSessionFactory;  
  
import java.util.List;  
  
/\*\*  
 \* @author swedsn  
 \* @version 1.0  
 \* @date 2022-12-14 14:04  
 \*/  
public class OrderImpl implements OrderService {  
  
 SqlSessionFactory sqlSessionFactory = SqlSessionFactoryUtils.*getSqlSessionFactory*();  
  
 @Override  
 public List<Order> selectOrder() {  
 *//2、 获取SqlSession对象，用它来执行sql* SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.class);  
 *// 4、调用方法* List<Order> order = orderMapper.selectOrder();  
 sqlSession.close();  
 return order;  
 }  
  
 @Override  
 public List<Order> selectOrderByCondition(int orderID, String orderStatus) {  
 *//2、 获取SqlSession对象，用它来执行sql* SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.class);  
 *// 4、调用方法* List<Order> order = orderMapper.selectOrderByCondition(orderID, orderStatus);  
 sqlSession.close();  
 return order;  
 }  
  
 @Override  
 public List<Order> selectOrderByPage(int num) {  
 SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.class);  
 *// 4、调用方法* int numStart = (num - 1) \* 10;  
 List<Order> order = orderMapper.selectOrderByPage(numStart);  
 sqlSession.close();  
 return order;  
 }  
  
 @Override  
 public int getOrderNum() {  
 SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.class);  
 *// 4、调用方法* int total = orderMapper.getOrderNum();  
 sqlSession.close();  
 return total;  
 }  
  
 @Override  
 public Boolean modifyOrderStatus(int orderID) {  
 SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.class);  
 *// 4、调用方法* Boolean flag = orderMapper.modifyOrderStatus(orderID);  
 sqlSession.close();  
 return flag;  
 }  
  
 @Override  
 public Boolean deleteOrder(int orderID) {  
 SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.class);  
 *// 4、调用方法* Boolean flag = orderMapper.deleteOrder(orderID);  
 sqlSession.close();  
 return flag;  
 }  
  
 @Override  
 public Boolean addOrder(Order order) {  
 SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.class);  
 StatisticMapper statisticMapper = sqlSession.getMapper(StatisticMapper.class);  
 CartMapper cartMapper = sqlSession.getMapper(CartMapper.class);  
 *// 4、调用方法* orderMapper.addOrder(order);  
 String orderContents = order.getOrderContents();  
 String[] strs = orderContents.split(",");  
 for (String str : strs) {  
  
 int num = Integer.*parseInt*(str.substring(str.length()-1, str.length()));  
 String goodsName = str.substring(0, str.length()-2);  
 Statistic statistic = new Statistic();  
  
 cartMapper.deletCart(order.getOrderTelephone(),statisticMapper.getGoodsIDByName(goodsName));  
  
 statistic.setOrderID(order.getOrderID());  
 statistic.setGoodsID(statisticMapper.getGoodsIDByName(goodsName));  
 statistic.setGoodsNum(num);  
 statistic.setBuyTime(order.getOrderTime());  
 statisticMapper.addStatistic(statistic);  
 }  
 sqlSession.close();  
 return true;  
 }  
  
 @Override  
 public List<Order> selectOrderByTelephone(String telephone) {  
 *//2、 获取SqlSession对象，用它来执行sql* SqlSession sqlSession = sqlSessionFactory.openSession();  
 *// 3、获取userMapper接口的代理对象（Mapper代理开发）* OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.class);  
 List<Order> orders = orderMapper.selectOrderByTelephone(telephone);  
 return orders;  
 }  
  
  
}