天津大学

《软件工程综合实践》结课报告

《软件工程综合实践》结课报告

学　　院　只能与计算学部

专　　业　软件工程

年　　级　2022 级

姓　　名　张三

学　　号　3022244455

2024 年 9 月 10 日

目　　录

第一章　软件需求规格说明书

1.1　引言

1.1.1　简介

本文档的目的是详细地介绍饿了吧 APP 所包含的需求，以便客户能够确认产品的确切需求以及开发人员能够根据需求设计编码，以下叙述将结合文字描述、UML 图、ER 图等来描述饿了吧 APP 的功能、性能、用户界面、运行环境、外部接口以及针对用户操作给出的各种响应。本文档的预期读者有用户及客户、项目经理、开发人员以及跟该项目相关的其他竞争人员。

1.1.2　背景

1.1.2.1　项目背景

作为天津大学智算学部暑假到 T 软公司实习的学生。上班第一天，老板告诉我们，公司现在有个紧急项目，要求三周后上线，但是……上周末，这个项目唯一的程序员，小东，因不满公司的薪水待遇过低而离职了，现在留下了一个半成品的烂摊子，重担就这样落在了我们几个毫无开发经验的实习生身上……

T 软公司为了站在互联网的风口起飞，决定快速开发一个互联网 APP，模仿著名的互联网点外卖的 APP “饿了么”，开发一个竞品 APP “饿了吧”。此项目要求必须三周内上线，时间紧任务重，本着小步快跑、快速迭代的原则，“饿了吧”第一期仅开发下列功能：

- 商家入驻平台，管理待售商品。
- 用户注册，管理个人账号和个人信息。
- 用户点餐，管理购物车，管理送货地址，下单支付。
- 注：支付界面中点击确认后，将调用支付宝等平台完成实际支付，此功能由其他组开发，本项目组不必开发，直接返回支付成功即可。

1.1.2.2　开发路线

基于公司的技术基础和偏好，本项目后续开发将延用之前采用的前后端分离的技术路线。

- 后端：Spring Boot
- 前端：VUE 3
- 数据库：MySQL

1.1.2.3　项目现状

- 唯一的程序员小东已跑路，现在项目组只有你们几个实习生。

- 经检查，小东仅完成了用户点餐功能，包括购物车管理、送餐地址管理、下订单几个功能，其它功能并未完成，像用户、商家、商品等数据，是在数据库中直接添加的。
- 已完成的功能中，也存在不少问题，有功能上的问题，也有设计上的问题。
- 现有的材料，包括
  - 部分不规范的文档和代码
  - 公司要求小东录制的一些技术讲座视频，可以用来企业内部学习相关技术

1.1.2.4 目前项目工作内容

- 重新进行项目需求分析，达到预期目标
- 项目重构，解决原项目中存在的问题

1.1.3 定义、缩略语

术语	**解释**
购物车	用户在浏览商家菜单时选择的多个菜品和商品的临时集合，用户可以在下单之前将这些选定的商品放入购物车中。

表 1-1 术语解释

1.1.4 约束

前后端分离的设计方案，前后端使用符合 RESTful 风格的接口设计

1.2 需要分析

1.2.1 目标

饿了吧旨在为用户以及商家提供一款便捷，高效的在线外卖平台，支持用户在线快速浏览附近餐厅，下单并获得及时送餐服务，最终做到让用户享受简单且快速的订餐体验，同时提升餐厅的订单量和运营效率. 具体而言，饿了吧最终会实现以下内容:

1. 实现用户与商家的注册行为.
2. 用户和商家能够修改自身账号信息.
3. 用户能够完成从点餐、查看购物车、修改订单地址到付款的点餐流程.
4. 商家能够完成对自身商品的上架、下架、修改操作.
5. 设置管理员登录，管理商家信息.

1.2.2 涉众分析

本系统只涉及第 1.2.3,4 类涉众, 没有第 5 类涉众

序号	涉众	代表人物	待解决的问题/对系统的期望
1	顾客	张三	1. 能注册账号 2. 能管理账号和个人信息 3. 能点餐，管理购物车、送货地址，下单支付
2	商家	李四	1. 能注册账号 2. 能管理商家信息 3. 能上架、下架、管理待售商品
3	管理员	王五	1. 能管理用户数据
4	审核员	赵六	1. 审核商家上架商品
5	配送员 (骑手)	李七	1. 能获取订单信息 2. 获取商家以及用户地点进行取餐, 送餐

表 1-2　　涉众分析表

1.2.3 范围

本系统不与实际支付接口对接，不提供针对配送员的业务流程

1.3 业务分析

1.3.1 业务概念分析

1.3.1.1 概述

饿了吧要管理的事情主要分为三个部分: 用户注册, 用户下订单, 商家上架与下架商品.

1.3.1.2 业务概念一览

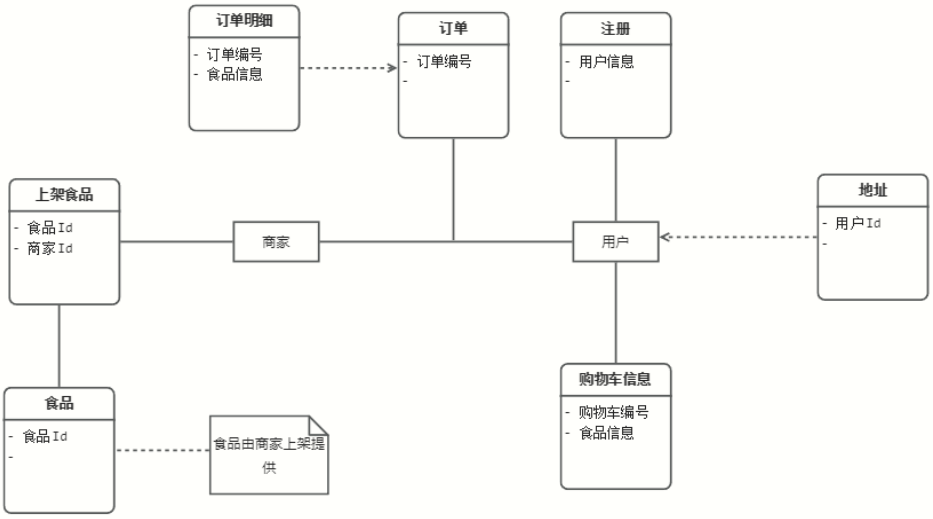

图 1-1 业务概念一览

1.3.1.3 登录用户

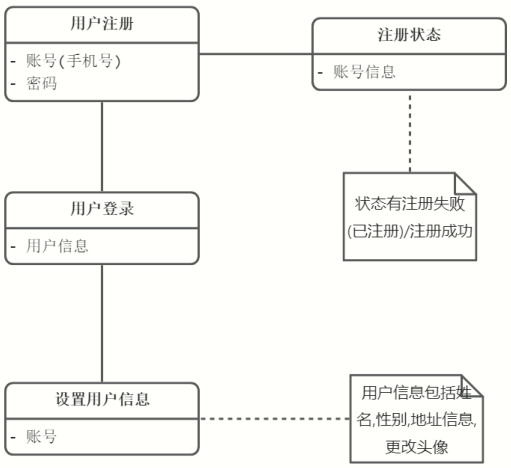

图 1-2 登录用户

1.3.1.4 顾客下单

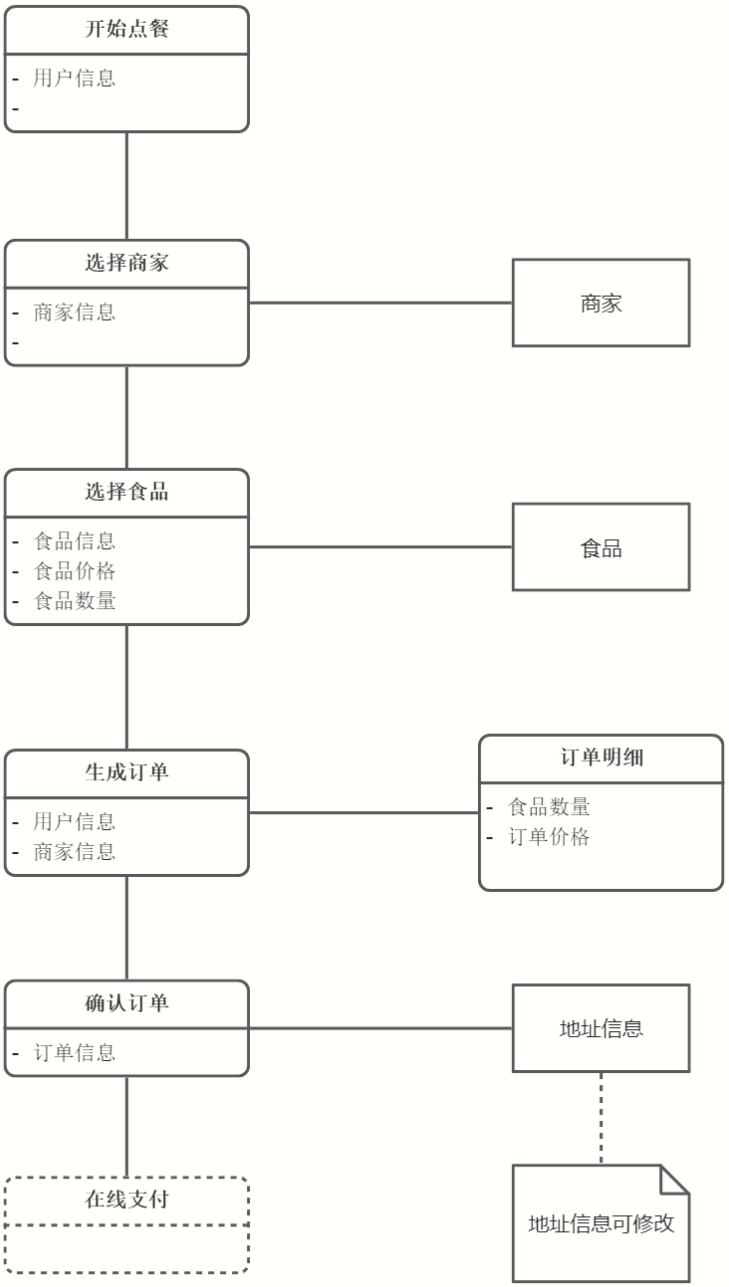

图 1-3　　顾客下单

1.3.2 商家管理商品

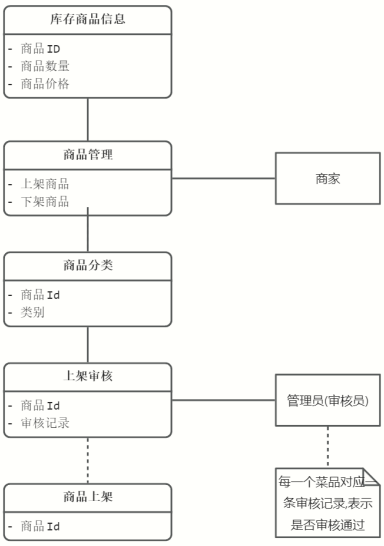

图 1-4　　商家管理商品

1.3.3 业务流程分析（UML 活动图）

1.3.3.1 用户登录流程

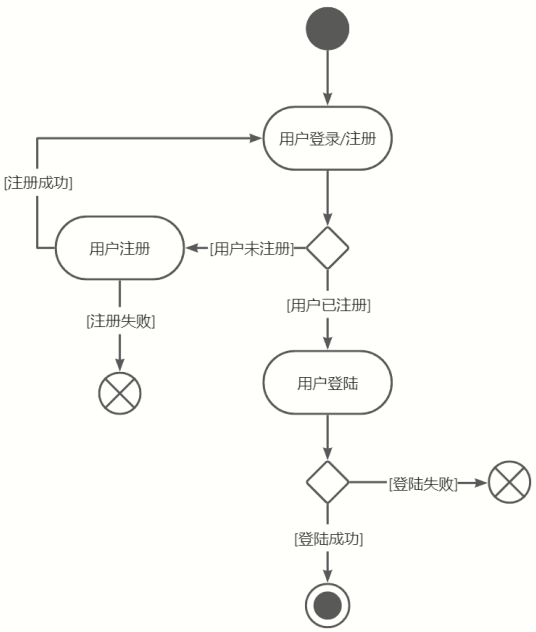

图 1-5　　用户登录流程

1.3.3.2 顾客下单流程

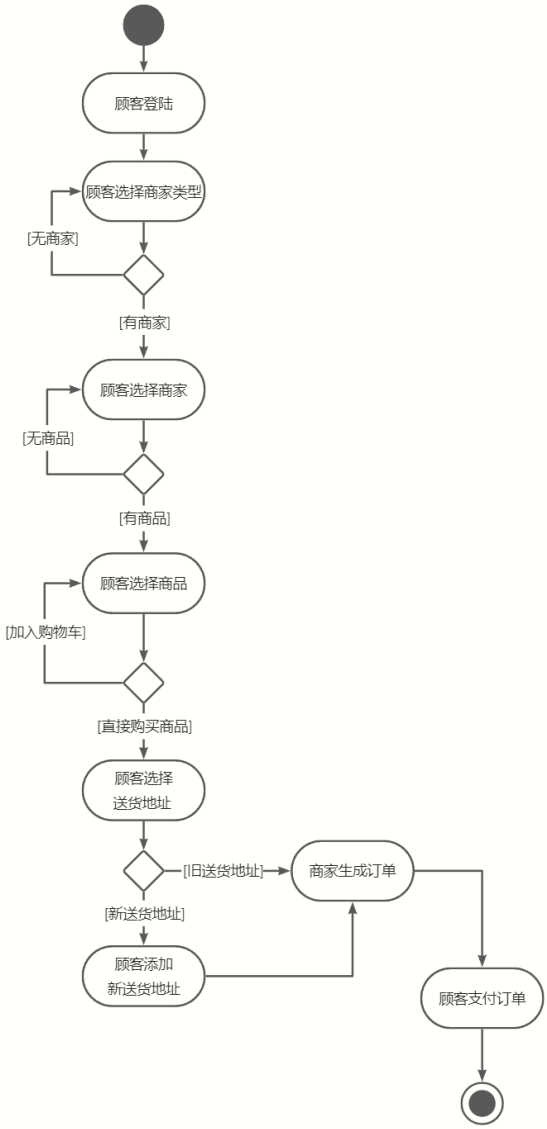

图 1-6　　顾客下单流程

1.3.3.3 商家管理商品流程

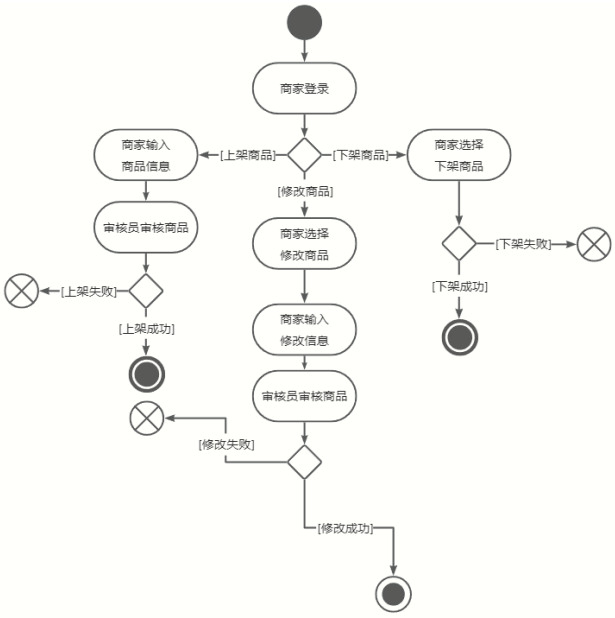

图 1-7　　商家管理商品流程

1.4 功能性需求

1.4.1 执行者分析

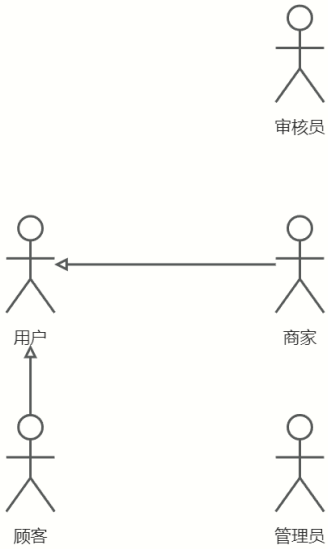

图 1-8　　执行者分析

1.4.2 总用例图

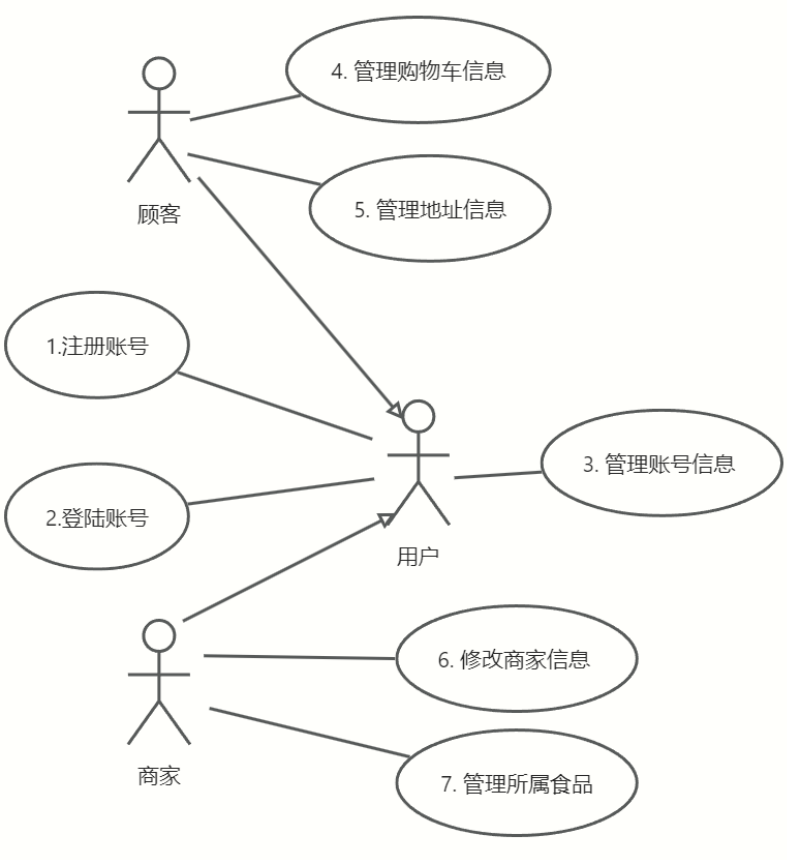

图 1-9　　总用例图

1.4.3 用户的用例

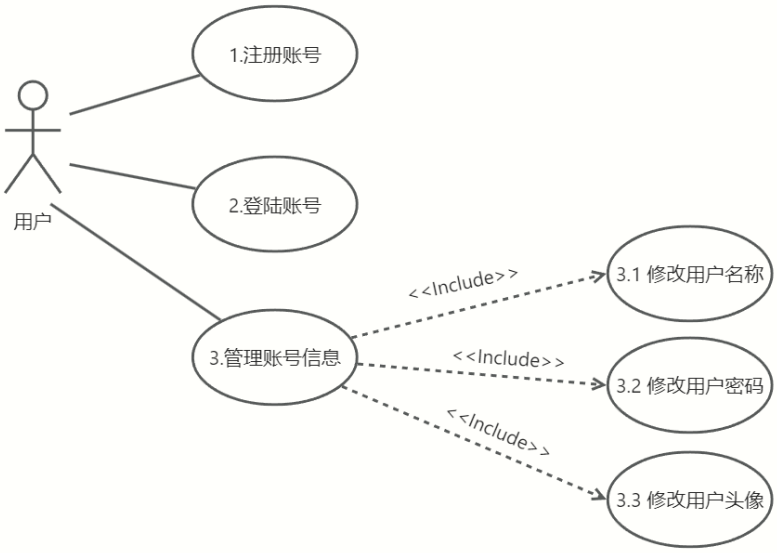

图 1-10　　用户的用例

编号	1	名称	注册账户
执行者	普通用户	优先级	高
描述	用户使用手机号进行账户注册, 设置账户密码		
前置条件	输入正确的手机号码		
基本流程	1. 输入手机号码 2. 输入密码 3. 显示注册成功信息		
结束状况	注册成功进入登录界面		
可选流程			
异常流程	1. 手机号验证不通过, 要求重新填写手机号 2. 用户已经注册显示用户已存在信息		
描述			

表 1-3

编号	2	名称	登录账户
执行者	普通用户	优先级	高
描述	用户使用手机号进行账户登录		
前置条件	1. 提交正确的手机号码 2. 提交正确的密码		
基本流程	1. 输入手机号码 2. 输入密码 3. 显示登录成功信息		
结束状况	1. 登录成功进入主界面 (注册时) 2. 登陆成功返回上一个页面		
可选流程	注册账号		
异常流程	1. 手机号验证不通过, 要求重新填写手机号 2. 用户未注册进入用户注册流程 3. 密码验证不通过		
描述			

表 1-4

编号	3.1	名称	修改用户名称
执行者	普通用户	优先级	低
描述	用户修改个人信息中的名称		
前置条件	无		
基本流程	1. 输入新用户名 2. 判断是否满足要求 3. 显示修改成功界面		
结束状况	用户界面显示修改后的用户名		
可选流程	1. 用户名称不符合要求，重新输入 2. 用户名称已经存在，显示用户名称已经被使用		
异常流程			
描述			

表 1-5

编号	3.2	名称	修改用户密码
执行者	普通用户	优先级	低
描述	用户修改个人信息中的密码		
前置条件	1. 旧密码输入正确		
基本流程	1. 输入新密码并再次输入确保无误 2. 判断是否满足密码要求 3. 显示密码修改成功界面		
结束状况	用户退出登录，需要重新输入新密码		
可选流程	1. 用户密码不符合要求，要求重新输入		
异常流程			
描述			

表 1-6

编号	3.3	名称	修改用户头像
执行者	普通用户	优先级	低
描述	用户修改个人信息中的头像		
前置条件			
基本流程	1. 导入新头像 2. 判断是否满足大小要求 3. 显示头像修改成功界面		
结束状况	用户界面显示新头像		
可选流程	1. 图片大小不符合要求，要求重新导入		
异常流程			
描述			

表 1-7

1.4.4 顾客的用例

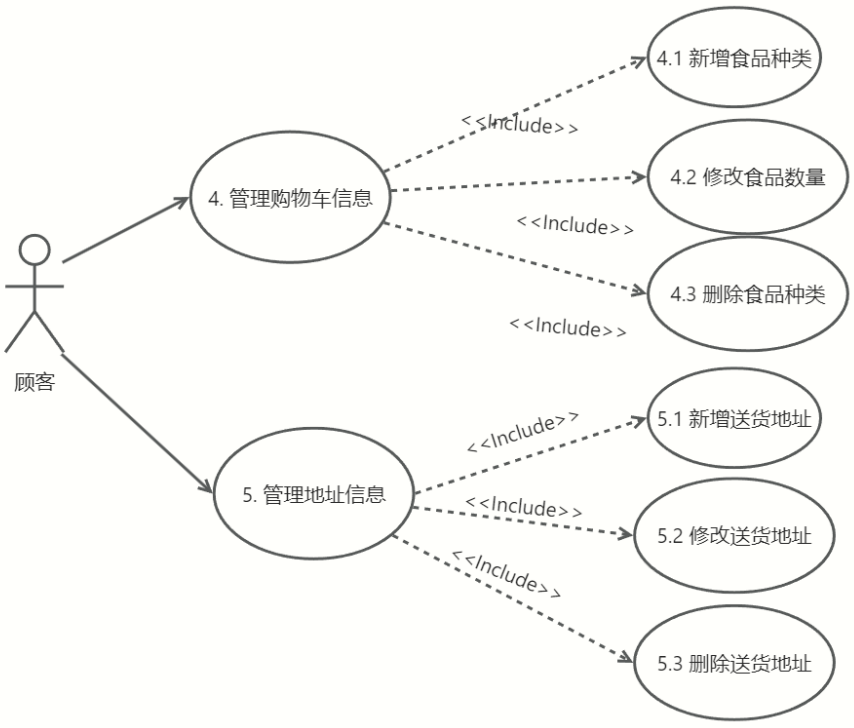

图 1-11　　顾客的用例

编号	4	名称	管理购物车信息
执行者	顾客	优先级	低
描述	用户对购物车中商品进行新增, 修改, 删除操作		
前置条件	登录用户		
基本流程	1. 向购物车中添加商品 2. 修改商品的数量 3. 统计购物车内商品的总价格		
结束状况	得到一个完整的购物车内容 超过起送费即可生成订单		
可选流程	1. 删除购物车中的商品.		
异常流程			
描述			

表 1-8

编号	5.1	名称	新增送货地址
执行者	顾客	优先级	低
描述	顾客新增地址列表中的送货地址		
前置条件			
基本流程	1. 点击地址界面的新建用户地址 2. 依次填入地址信息满足正常地址规范 3. 显示新建地址成功界面		
结束状况	用户界面出现新地址的简略版本		
可选流程	1. 用户输入的地址不符合正常的地址规范，需要重新输入 2. 地址中的必填信息未填写，需要写入		
异常流程			
描述			

表 1-9

编号	5.2	名称	修改送货地址
执行者	顾客	优先级	低
描述	顾客修改地址列表中的送货地址		
前置条件			
基本流程	1. 点击地址界面的用户地址 2. 修改地址信息满足正常地址规范 3. 显示修改地址成功界面		
结束状况	用户界面出现修改后的新地址的简略版本		
可选流程	1. 用户输入的地址不符合正常的地址规范，需要重新输入 2. 地址中的必填信息未填写，需要写入		
异常流程			
描述			

表 1-10

编号	5.3	名称	删除送货地址
执行者	顾客	优先级	低
描述	顾客删除地址列表中的送货地址		
前置条件	1. 地址列表中有已经存在的地址		
基本流程	1. 点击地址界面的用户地址，进入用户的地址列表界面 2. 选中需要删除的地址，点击删除，询问是否删除 3. 显示删除地址成功界面		
结束状况	用户的地址列表界面原地址消失		
可选流程	1. 误触点到删除，询问是否删除时点击否，返回用户地址列表界面		
异常流程			
描述			

表 1-11

1.4.5 商家的用例

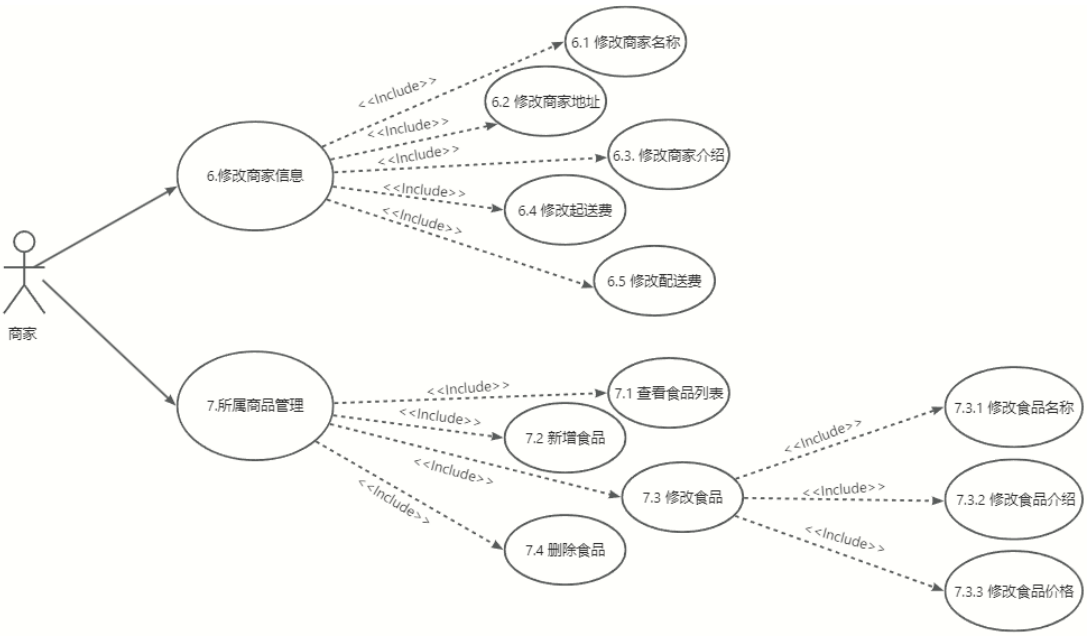

图 1-12 商家的用例

编号	6.1	名称	修改商家名称
执行者	商家	优先级	低
描述	商家修改商家信息中的名称		
前置条件	无		
基本流程	1. 输入新商家名 2. 判断是否满足要求 3. 显示修改成功界面		
结束状况	商家界面显示修改后的商家名		
可选流程	1. 用户名称不符合要求，重新输入 2. 用户名称已经存在，显示用户名称已经被使用		
异常流程			
描述			

表 1-12

编号	6.2	名称	修改商家地址
执行者	商家	优先级	低
描述	商家修改商家信息中的地址		
前置条件	无		
基本流程	1. 输入商家的新地址 2. 判断是否满足正常的地址规范 3. 显示地址修改成功界面		
结束状况	商家界面显示修改后的商家地址		
可选流程	1. 商家地址不符合要求，重新输入		
异常流程			
描述			

表 1-13

编号	6.3	名称	修改商家介绍
执行者	商家	优先级	低
描述	商家修改商家信息中的介绍		
前置条件	无		
基本流程	1. 输入商家的新介绍 2. 判断是否满足要求（字数要求，敏感词） 3. 显示修改成功界面		
结束状况	商家界面显示修改后的商家介绍		
可选流程	1. 商家介绍不符合要求，重新输入		
异常流程			
描述			

表 1-14

编号	6.4	名称	修改商家起送费
执行者	商家	优先级	低
描述	商家修改商家信息中的起送费		
前置条件	无		
基本流程	1. 输入商家的新起送费 2. 判断是否满足要求（选择：免起送费，固定金额） 3. 显示修改成功界面		
结束状况	商家下单界面显示修改后的新起送费 只要满足订单金额大于起送费时，订单才允许跳转到支付界面		
可选流程	1. 商家起送费不符合要求，重新输入		
异常流程			
描述			

表 1-15

编号	6.5	名称	修改商家配送费
执行者	商家	优先级	低
描述	商家修改商家信息中的配送费		
前置条件	无		
基本流程	1. 输入商家新的配送费 2. 判断是否满足要求（金额要求） 3. 显示修改成功界面		
结束状况	商家下单界面显示修改后的配送费		
可选流程	1. 商家配送费不符合要求，重新输入		
异常流程			
描述			

表 1-16

1.4.6 其他功能性需求

在” 我的” 页面通过” 成为商家” 功能对用户权限进行修改, 成为商家之后在” 我的” 页面会显示” 店铺管理” 功能, 进入” 店铺管理” 功能页面即可执行商家相关功能.

1.5 非功能性需求

1.5.1 系统架构要求

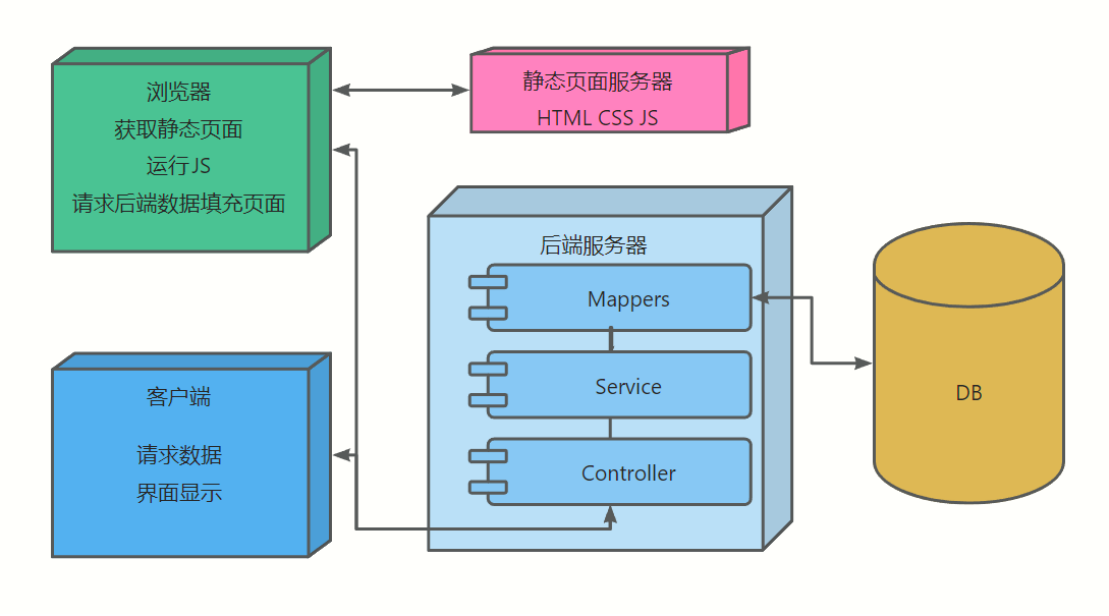

图 1-13 系统架构

1.5.1.1 微服务架构

服务拆分：饿了吧的业务逻辑复杂，包括用户管理、商户管理、订单处理、支付系统等多个模块。微服务架构将这些模块拆分为独立的服务，便于各自独立开发、部署和维护。不仅提高了开发效率，还增强了系统的可扩展性和容错能力。

1.5.2 接口 API

business

- businesses/orderType/orderTypeId
  - 请求方法:GET
  - 参数：orderTypeId
  - 返回值：business 数组
  - 功能：根据点餐分类编号查询分类商家信息
  - 后端函数名:listBusinessByOrderTypeId
- businesses/businessId
  - 请求方法:GET
  - 参数:businessId
  - 返回值:businessId 对象
  - 功能: 根据商家编号查询商家信息

  - 后端函数名:getBusinessById
- businesses
  - 请求方法:POST
  - 参数:businessName、businessImg、orderTypeId
  - 返回值:int(返回行数)
  - 功能: 向商家表中添加一条字段
  - 后端函数名:r egisterBusiness
- businesses
  - 请求方法:PUT
  - 参数:businessId、businessName、businessAddress、businessExplain、businessImg、orderTypeId、starPrice、deliveryPrice
  - 返回值:int(返回影响的行数)
  - 功能: 向商家表中更新一条记录
  - 后端函数名:updateBusiness
- 5. businesses
  - 请求方法:DELETE
  - 参数:businessId
  - 返回值:int(删除成功)
  - 功能: 根据商家编号删除一条商家记录（将 delTag 设为 0）
  - 后端函数名:deleteBusiness

food

- foods/business/businessId
  - 请求方法:GET
  - 参数:businessId
  - 返回值:food 数组
  - 功能: 根据商家编号查询所属食品信息
  - 后端函数名:listFoodByBusinessId
- foods/foodId
  - 请求方法:GET
  - 参数:foodId
  - 返回值:food 对象
  - 功能: 根据食品编号查询食品信息
  - 后端函数名:getFoodById
- foods
  - 请求方法:POST

  - 参数:foodName、foodExplain、foodImg、foodPrice、businessId、quantity
  - 返回值:int
  - 功能: 向食品表中添加一条记录
  - 后端函数名:addfood
- foods/foodId
  - 请求方法:PUT
  - 参数:businessId、foodId、foodName、foodExplain、foodImg、foodPrice、quantity
  - 返回值:int
  - 功能: 根据商家编号、食品编号更新一条食品记录（检测商家是否匹配，更新前拷贝）
  - 后端函数名:updateFood
- foods
  - 请求方法:PATCH
  - 参数:businessId、foodId、soldOut
  - 返回值:int
  - 功能: 根据商家编号、食品编号更新食品是否售罄
  - 后端函数名:setFood
- foods/foodId
  - 请求方法:DELETE
  - 参数:businessId、foodId
  - 返回值:int
  - 功能: 根据商家编号、食品编号删除一条食品记录（将 delTag 设为 0，检测商家是否匹配，删除前拷贝）
  - 后端函数名:removeFood

cart

- carts/user/userId/businessId
  - 请求方法:GET
  - 参数:userId、businessId（可选）
  - 返回值:cart 数组（多对一：所属商家信息、所属食品信息）
  - 功能: 据用户编号和商家编号，查询此用户购物车中某个商家的所有购物车信息
  - 后端函数名:listCart
- carts

  - 请求方法:POST
  - 参数:userId、businessId、foodId
  - 返回值:int
  - 功能: 向购物车表中添加一条记录
  - 后端函数名:saveCart
- carts
  - 请求方法:PUT
  - 参数:userId、businessId、foodId、quantity
  - 返回值:int
  - 功能: 根据用户编号、商家编号、食品编号更新数量
  - 后端函数名:updateCart
- carts
  - 请求方法:DELETE
  - 参数:userId、businessId、foodId（可选）
  - 返回值:int
  - 功能: 根据用户编号、商家编号、食品编号删除购物车表中的一条食品记录
  - 根据用户编号、商家编号删除购物车表中的多条记录
  - 后端函数名:removeCart

deliveryAddress

- delivery-addresses/user/userId
  - 请求方法:GET
  - 参数:userId
  - 返回值:deliveryAddress 数组
  - 功能: 根据用户编号查询所属送货地址
  - 后端函数名:listDeliveryAddressByUserId
- delivery-addresses/daId
  - 请求方法:GET
  - 参数:daId
  - 返回值:deliveryAddress 对象
  - 功能: 根据送货地址编号查询送货地址
  - 后端函数名:getDeliveryAddressById
- delivery-addresses
  - 请求方法:POST
  - 参数:contactName、contactSex、contactTel、address、userId

  - 返回值:int
  - 功能: 向送货地址表中添加一条记录
  - 后端函数名:saveDeliveryAddress
- delivery-addresses
  - 请求方法:PUT
  - 参数:daId、contactName、contactSex、contactTel、address、userId
  - 返回值:int
  - 功能: 根据送货地址编号更新送货地址信息（更新前拷贝）
  - 后端函数名:updateDeliveryAddress
- delivery-addresses/daId
  - 请求方法:DELETE
  - 参数:daId
  - 返回值:int
  - 功能: 根据送货地址编号删除一条记录（删除前拷贝）
  - 后端函数名:removeDeliveryAddress

order

- orders/user/userId
  - 请求方法:GET
  - 参数:userId
  - 返回值:orders 数组（包括多对一：business；一对多：订单明细信息）
  - 功能: 根据用户编号查询此用户的所有订单信息
  - 后端函数名:listOrdersByUserId
- orders/orderId
  - 请求方法:GET
  - 参数:userId
  - 返回值:orders 数组（包括多对一：business；一对多：订单明细信息）
  - 功能: 根据订单编号查询订单信息，包括所属商家信息，和此订单的所有订单明细信息
  - 后端函数名:getOrdersById
- orders
  - 请求方法:POST
  - 参数:userId、businessId、daId、orderTotal
  - 返回值:int

  - 功能:

    1. 用户编号、商家编号、订单总金额、送货地址编号向订单表中添加一条记录, 并获取自动生成的订单编号

    2. 根据用户编号、商家编号从购物车表中查询所有数据，批量添加到订单明细表中

    3. 根据用户编号、商家编号删除购物车表中的数据。
  - 后端函数名:createOrders

user

- users/login
  - 请求方法:POST
  - 参数:userId、password
  - 返回值:user 对象
  - 功能: 根据用户编号与密码查询用户信息
  - 后端函数名:getUserByIdByPass
- users/userId
  - 请求方法:GET
  - 参数:userId
  - 返回值:int
  - 功能: 根据用户编号查询用户表返回的行数
  - 后端函数名:getUserById
- users/register
  - 请求方法:POST
  - 参数:userId、password、userName、userSex
  - 返回值:int
  - 功能: 向用户表中添加一条记录
  - 后端函数名:saveUser
- users
  - 请求方法:PUT
  - 参数:userId、password、userName、userSex、userImg（可选）
  - 返回值:int
  - 功能: 根据用户编号与密码更改用户信息
  - 后端函数名:updateUser
- users
  - 请求方法:DELETE
  - 参数:userId、password

  - 返回值:int
  - 功能: 根据用户编号与密码删除用户（将 delTag 设为 0）
  - 后端函数名:deleteUser

key

- public-key
  - 请求方法:GET
  - 参数:
  - 返回值:string
  - 功能: 根据后端的私钥生成公钥返回到前端
  - 后端函数名:getPublicKey

captcha

- captcha
  - 请求方法:GET
  - 参数:session
  - 返回值:byte[]
  - 功能: 生成验证码图片发送到前端
  - 后端函数名:getCaptcha
- captcha
  - 请求方法:POST
  - 参数:session,captchaInput
  - 返回值:ture or false
  - 功能: 对前端发来的验证码进行验证
  - 后端函数名:verifyCaptcha

1.5.3 安全性

饿了吧在前后端采用多种方案共同实现对于用户敏感信息的保护:

- 在传输密码时, 使用 HTTP POST 进行传输, 不在 URL 中传输敏感数据
- 实现登录时, 对密码的传输首先在前端接收后端发送的公钥, 在前端对密码进行加密, 前端传到后端的内容为密文, 密文在后端进行解密, 保证传输过程的密码安全性
- 用户注册时, 后端将会对密码进行哈希加密处理之后再存入数据库. 保存在数据库中的密码为哈希值, 有效防止数据库被入侵后导致用户密码信息的泄露风险

另外, 我们在需要登录信息才能访问的页面发送的请求中附带 token, 在前端, 我们在创建好的 axios 实例 axiosInstance 上，挂载两个拦截器:

- 请求拦截器：
  - 前端向后端发送请求之前拦截；
  - 在 header 添加 token，若有请求错误，在控制台打出
- 响应拦截器：
  - 后端向前端发送响应，前端接到之后拦截；
  - 处理响应状态码错误 401，500 等
  - 网络错误，重试三次，重试失败，跳转回主页；
  - 重试次数超过最大限制 3，跳转回主页

后端接到请求后，进入 controller 层之前拦截；登录成功后得到 token 并发回前端，后所有请求都需加上 token；被拦截的请求发回 500 错误码；拦截涉及用户信息的请求，有关拦截过滤的内容在 WebMvcConfig 文件配置，需精细到请求方法的在 TokenInterceptor 配置；

同时，为了预防恶意脚本使用大量相同请求对于服务器的攻击，我们设计了验证码人机验证的接口，主要是在用户下订单流程中，同一订单中商品超过一定数量就会触发人机验证，通过从后端接收随机生成的，经过多重模糊化处理的多位数字图片进行人机验证，用户根据验证码图片的内容填写验证码，验证失败则拒绝改次对服务器的访问，从而保证了服务器的安全.

1.5.4 性能

无

1.5.5 界面

饿了吧的界面风格强调简洁、现代和用户友好，界面流的设计目的是让用户能够快速、顺畅地完成点餐和支付。首页设计则突出了搜索、分类导航、订单和用户信息管理，确保用户能够快速找到所需的内容。商家后台的报表功能详细且直观，帮助商家有效管理和分析经营数据，优化服务质量。

1.5.5.1 界面风格

在界面风格的设计上，饿了吧秉持简洁、现代、友好的设计思路，遵循以下几个特点：

- 颜色搭配：主色调为蓝色和白色，蓝色作为品牌的标志性颜色，用于导航栏、按钮等高亮位置，白色背景确保信息清晰易读。辅助色（如橙色、红色）用作重要信息提示，增强视觉对比。
- 扁平化设计：饿了吧采用扁平化设计语言，图标简洁、信息模块化，整体界面更加简约和直观。
- 图标和插图：界面中使用卡通风格的图标和插图，增强了友好度.
- 响应式设计：饿了吧的设计能够适应各种设备屏幕，从手机到平板和电

脑，界面元素会根据屏幕尺寸动态调整，保证了用户体验的一致性。

1.5.5.2 界面流

饿了吧的界面流设计注重用户体验的顺畅性和任务的高效完成, 在页面路由的设计上力争以最高效的流程实现业务操作

1.5.5.3 首页

饿了吧的首页设计简洁直观，旨在让用户快速找到所需的餐厅或商品。

1.5.5.4 报表信息

饿了吧计划提供功能丰富的后台系统，方便商家查看订单数据、经营情况和各类分析报表。

第二章　项目设计方案

2.1　数据库设计

在原有的饿了吧 V1.0 的数据库结构基础上, 饿了吧 V2.0 对数据库进行了如下修改:

1. business 表添加 delTag 字段
2. food 表添加 delTag 字段
3. deliveryaddress 表添加 delTag 字段
4. delTag 表示删除标记, 其中 0: 正常，1: 删除
5. food 表添加 soudOut 字段
6. user .password 成员的长度修改为 70
7. user 表添加 authorization 成员，0: 管理员，1: 普通用户，2: 商家用户
8. user 表添加 businessId 成员, 无符号, 添加外键, 对 business 表 businessId 的引用

2.1.1　DB 一览表

NO	表名称	中文名	说明
1	business	商家表	存储所有商家信息
2	food	食品表	存储每个商家拥有得食品得的所有信息
3	cart	购物车表	存储每个用户的购物车中的食品信息
4	deliveryaddress	送货地址表	存储每个用户的所有送货地址信息
5	orders	订单表	存储每个用户的所有订单信息
6	orderdetailet	订单明细吧表	存储每个订单中的所有食品信息
7	user	用户表	存储所有用户的信息

表 2-1　数据库一览表

2.1.2　表结构

约束类型标识：PK：primary key 主键 FK：foreign key 外键 NN：not null 非空 UQ：unique 唯一索引 AI：auto increment 自增长列

删除标记: 数据类型 int, 正常:0 删除:1

点餐分类：1：美食、2：早餐、3：跑腿代购、4：汉堡披萨、5：甜品饮品、6：速食简餐、7：地方小吃、8：米粉面馆、9：包子粥铺、10：炸鸡炸串

2.1.2.1 business(商家表)

NO	字段名	数据类型	size	默认值	约束	说明
1	businessId	int			PK AI NN	商家编号
2	businessName	varchar	40		NN	商家名称
3	businessAddress	varchar	50			商家地址
4	businessExplain	varchar	40			商家介绍
5	businessImg	mediumtext			NN	商家图片
6	orderTypeId	int			NN	点餐分类
7	starPrice	decimal	(5,2)	0.00		起送费
8	deliveryPrice	decimal	(5,2)	0.00		配送费
9	remarks	varchar	40			备注
10	delTag	int		0	NN	删除标记

表 2-2　　商家表

2.1.2.2 food(食品表)

NO	字段名	数据类型	size	默认值	约束	说明
1	food	int			PK AI NN	食品编号
2	foodName	varchar	30		NN	食品名称
3	foodExplain	varchar	30		NN	食品介绍
4	foodImg	mediumtext			NN	食品图片
5	foodPrice	decimal	(5,2)		NN	食品价格
6	businessId	int			FK NN	所属商家编号
7	remarks	varchar	40			备注
8	soldOut	int			NN	是否售罄
9	delTag	int		0	NN	删除标记

表 2-3　　食品表

2.1.2.3 orders（订单表）

NO	字段名	数据类型	size	默认值	约束	说明
1	orderId	int			PK AI NN	订单编号
2	userId	varchar	20		FK NN	所属用户编号
3	businessId	int			FK NN	所属商家编号
4	orderDate	varchar	20		NN	订购日期
5	orderTotal	decimal	(7,2)	0.00	NN	订单总价
6	daId	int			FK NN	所属送货地址编号
7	orderState	int		0	NN	订单状态（0：未支付；1：已支付）

表 2-4　　订单表

2.1.2.4 orderdetailet（订单明细表）

NO	字段名	数据类型	size	默认值	约束	说明
1	odId	int			PK AI NN	订单明细编号
2	orderId	int			FK NN	所属订单编号
3	foodId	int			FK NN	所属食品编号
4	quantity	int			NN	数量

表 2-5　　订单明细表

2.1.2.5 user（用户表）

NO	字段名	数据类型	size	默认值	约束	说明
1	userId	varchar	20		PK NN	用户编号
2	password	varchar	70		NN	密码
3	userName	varchar	20		NN	用户名称
4	userSex	int			NN	用户性别（1：男；0：女）
5	userImg	mediumtext				用户头像
6	delTag	int			NN	删除标记
7	authorization	int			NN	用户权限
8	businessId	int			PK	商家 id

表 2-6　　用户表

2.2 系统架构设计

2.2.1 应用架构

饿了吧的应用架构可以按照功能模块分为多个层次，包括客户端层、应用服务层、业务逻辑层、数据存储层。这几层架构各自分工明确，能够支持高并发、高性能、扩展性和灵活的业务需求。

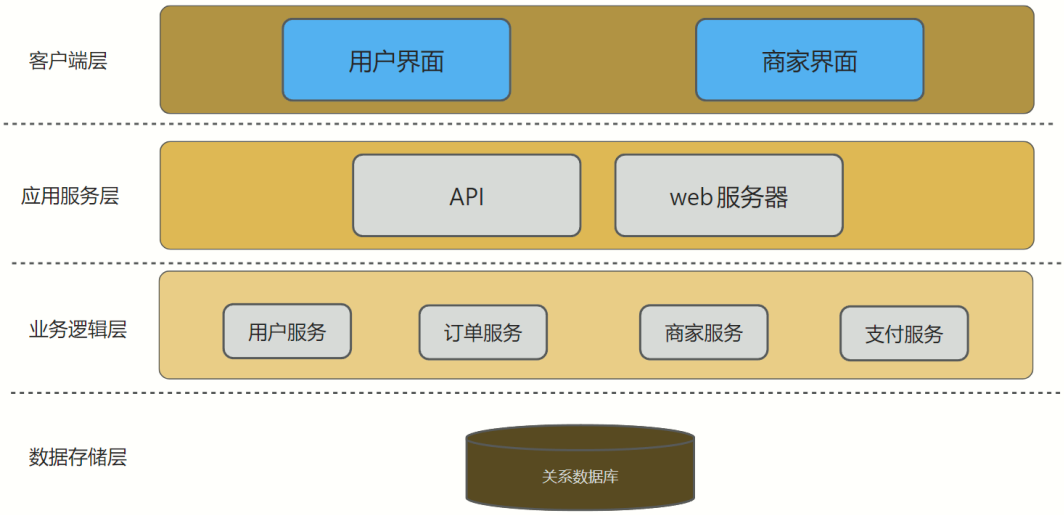

图 2-1 应用架构

2.2.1.1 客户端层

- 用户界面：普通用户界面，包含浏览商家、下单、支付、查看订单状态、评价等功能。
- 商家界面：商家界面，用于管理订单、商品信息、促销活动等。

2.2.2 应用服务层

- Web 服务器：处理客户端发来的 HTTP 请求。请求从用户端到 Web 服务器，然后分发给相应的后端服务。
- 负载均衡器：Nginx，用于将请求分发到多个服务实例，确保高并发和高可用性。

2.2.3 业务逻辑层

- 用户服务：负责用户注册、登录、个人信息管理、地址管理等功能。
- 订单服务：负责订单创建、支付、取消、状态跟踪等业务逻辑。
- 商家服务：管理商家的信息、商品列表等。

- 支付服务 (仅模拟实现)：集成第三方支付（如支付宝、微信支付），处理支付请求和对账服务。

2.2.3.1 数据存储层

- 关系型数据库：饿了吧利用 MySQL，用于存储结构化数据，如用户信息、订单数据、商家和商品信息。
- 缓存系统：如 Redis，用于缓存热数据，提升访问速度，减少数据库压力。

2.2.4 后端 Spring Boot 三层架构

饿了吧的后端采用 Spring Boot 三层结构，其应用逻辑分为三个主要层次，分别是表示层（Controller 层）、业务逻辑层（Service 层）和数据访问层（DAO 层）。

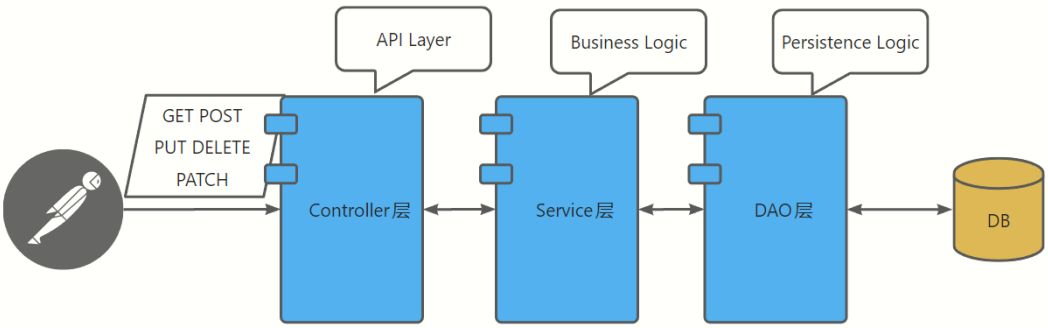

图 2-2 Spring Boot 三层架构

表示层负责接收和返回用户请求，业务逻辑层负责核心业务处理，数据访问层则负责与数据库的交互。饿了吧通过这种三层架构，保证代码的高复用性和可测试性。

2.3 用户界面设计

饿了吧的用户界面）和用户体验设计围绕简洁、高效、用户友好等核心原则，旨在为用户提供流畅的点餐、支付、配送跟踪等体验。通过清晰的布局、流畅的导航和实时反馈，饿了吧有效提升了用户点餐、支付、追踪订单的整体体验，符合时代下快节奏的生活方式和需求。

2.3.1 首页设计

- **布局简洁：**饿了吧的首页设计以卡片式布局为主，突出重点信息，如热门商家和推荐菜品. 首页的顶部包含一个搜索栏，便于用户快速搜索餐厅。
- **分类导航：**首页常包含美食、超市、生鲜等多种品类的快速导航，帮助用户迅速找到所需的服务. 见图2-3。

UI/UX **特点：**

- 清晰的导航：首页导航图标简洁、易理解，分类清晰。
- 信息优先级明确：通过视觉层次（如字体大小、颜色对比）突出重点信息，如限时优惠、热门餐厅等。
- 操作便捷：通过搜索栏，用户可以快速找到想要的餐厅。

2.3.2 餐厅列表页

- **餐厅展示**：用户选择某一类目后，会进入餐厅列表页面。每个餐厅的卡片展示餐厅名称、配送费、起送价等核心信息，方便用户比较选择。见图2-3。

UI/UX **特点：**

- 统一的视觉风格：所有餐厅卡片设计风格一致，布局统一，信息一目了然。

2.3.3 餐厅详情页

- **菜单展示**：点击餐厅进入详情页，用户可以看到详细的菜单分类，菜品按类型排列，并附有图片、价格等信息。
- **购物车浮动按钮**：当用户添加菜品到购物车后，屏幕底部会出现购物车的浮动按钮，实时显示已选商品和价格，方便用户查看和修改订单。见图2-3。

UI/UX **特点：**

- 清晰的菜单分类：菜单按照菜品类型分类，用户可以快速找到想要的餐品。
- 直观的购物车设计：浮动购物车让用户随时查看订单，避免迷失在复杂的菜品选择中。

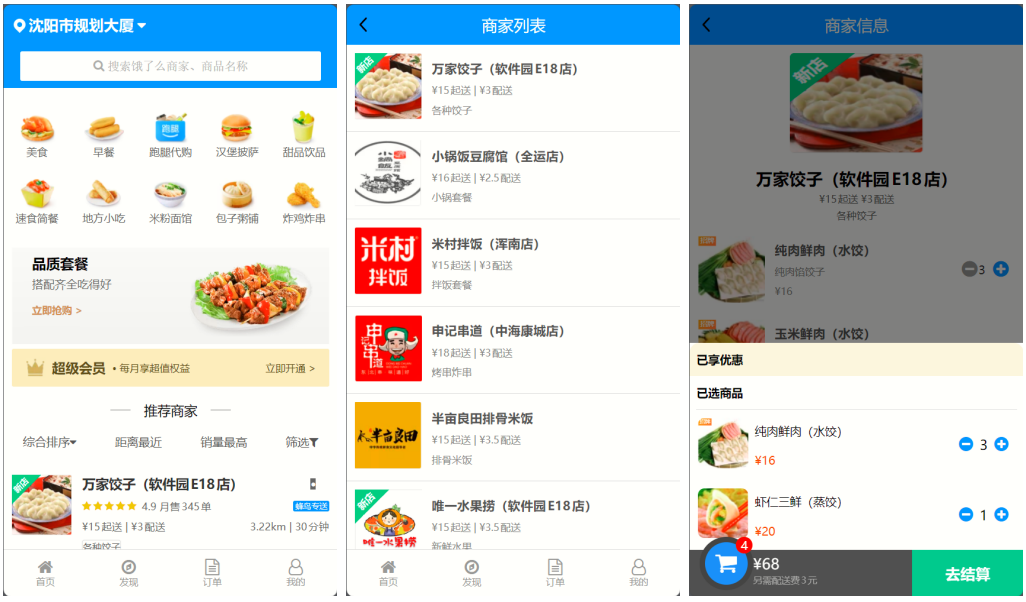

图 2-3　　用户界面

2.3.4 订单页

- **未支付订单**: 用户可以清晰的看到当前未支付的订单, 并通过去支付跳转到订单支付页面.
- **已支付订单**: 用户已支付订单的信息会罗列在这里方便查看. 见图2-4.

UI/UX **特点：**

- 清晰的订单支付情况概览: 用户能够快速了解到订单信息, 并对订单进行管理.

2.3.5 订单确认页

- **订单概览**：用户在确认订单页可以看到已选菜品、配送费等详细信息，页面设计简洁、条理清晰，帮助用户快速确认订单。
- **地址和支付方式**：用户可以选择配送地址、备注信息和支付方式，并支持多种支付方式，如支付宝、微信支付等。见图2-4.

UI/UX **特点：**

- 简化的订单流程：用户可以在同一页面完成订单核对、支付方式选择等多个步骤，减少操作步骤。

2.3.6 个人中心页

- **简洁的用户信息展示**：个人中心包括用户头像、用户昵称等信息，用户可以快速查看账户状态。
- **资料管理和地址管理**：用户可以管理配送地址、编辑个人资料等。见图2-4.

UI/UX **特点：**

- 模块化设计：信息和功能通过模块化方式展示，减少复杂度。
- 易于管理：用户可以轻松找到常用功能，如地址管理等。

2.4 安全设计

2.4.1 密码等隐私数据安全

2.4.1.1 Ras 加密

在前后端分离的系统中，密码的安全性至关重要。饿了吧作为典型的前后端分离式项目, 为了在传输过程中保护用户的密码，饿了吧利用 RSA 加密算法在前端对密码进行加密，后端接收到加密后的密码后进行解密。

RSA 加密作为非对称加密的典型，加密和解密使用不同的密钥，这种方式可以避免私钥在网络传输中的泄露风险。在饿了吧中, 利用 Rsa 对用户密码进行加密的过程如下:

1. 首先 OpenSSL 生成私钥, 并将私钥保存到配置文件中
2. 后端调用工具类利用私钥生成对应的公钥

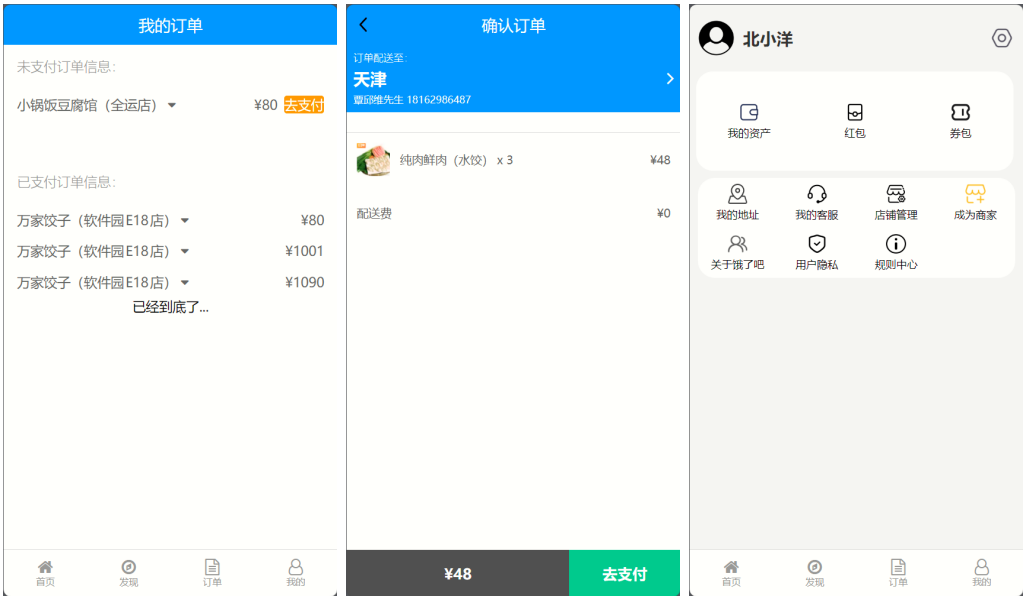

图 2-4　　订单以及个人中心

3. 前端申请注册请求时，后端通过 public-key 接口向前端发送生成的公钥
4. 前端利用接收到的公钥对明文密码进行加密形成密文，于是在前后端之间传输的内容时密文
5. 后端接收到前端发来的密文之后通过私钥对密文进行解密得到明文密码

2.4.2　加盐加密

为了预防数据库遭受攻击之后用户的密码信息泄露，提高密码存储的安全性，我们在后端将用户密码存储进数据库时，在原始密码的基础上，添加一个随机生成的值（称为“盐”或 salt），然后再对这个组合进行哈希运算。将得到的哈希值存储进数据库中，其中盐的目的是增加密码的复杂性和唯一性，使得即使用户的密码相同，生成的哈希值也会不同。从而有效保证了密码安全.

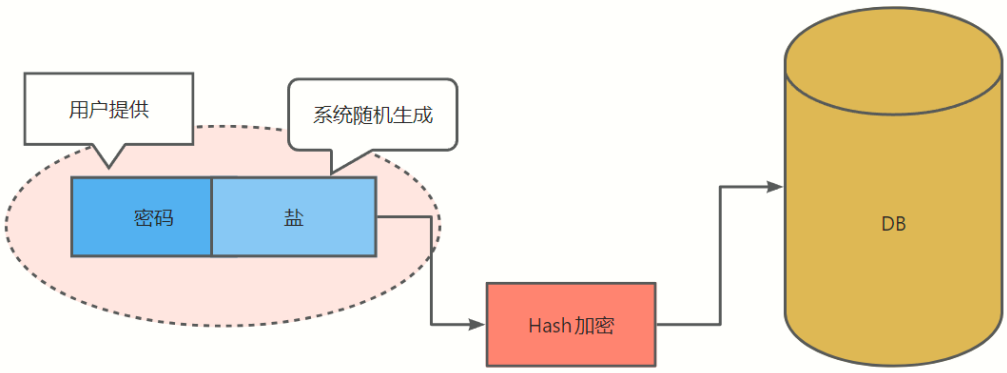

图 2-5　　加盐加密示意图

2.4.3 身份认证

饿了吧作为一款前后端分离的系统, 为了实现跨域传递的同时实现身份验证机制, 饿了吧采用 Token（令牌）原理与拦截器共同实现对用户的身份验证, 具体实现方法如下:

前端:

在创建好的 axios 实例 axiosInstance 上，挂载两个拦截器

- 请求拦截器:
  - 前端向后端发送请求之前拦截;
  - 在 header 添加 token，若有请求错误，在控制台打出
- 响应拦截器:
  - 后端向前端发送响应，前端接到之后拦截;
  - 处理响应状态码错误 401，500 等
  - 网络错误，重试三次，重试失败，跳转回主页;
  - 重试次数超过最大限制 3，跳转回主页

后端:

- 接到请求后，进入 controller 层之前拦截;
- 登录成功后得到 token 并发回前端，后所有请求都需加上 token;
- 被拦截的请求发回 500 错误码;
- 注: 拦截涉及用户信息的请求，仅需拦截 url 的在 WebMvcConfig 文件配置，需精细到请求方法的在 TokenInterceptor 配置;

在饿了吧中使用 token 认证, 允许用户在登录后，通过携带 Token 来进行身份认证，而不需要在每次请求时都重新输入用户名和密码。这种方式提高了用户体验，同时保证了系统的安全性。而 Token 和拦截器的结合则进一步提供了一种高效、安全、用户体验友好的认证方式。

2.4.4 防护策略

饿了吧为防止自动化程序或机器人对系统进行恶意操作。提升系统的安全性，使用文字验证码对短时间内进行大量相同请求进行人机验证跳转, 在饿了吧业务流程中, 主要应用于用户点餐流程中购物车添加过程, 当用户往购物车中添加的同一种食品超过一定数量时, 系统将提示人机验证, 防止机器人或者自动脚本恶意攻击. 具体的实现过程如下:

- 当前端检测到购物车中同一种商品的数量达到设定值, 时, 将会请求后端发送文字验证码图片
- 文字验证码在后端又系统随机生成, 并通过简单的模糊卷积核以及添加噪点的方式对验证码进行模糊化处理并发送到前端
- 验证码在前端显示, 并提醒用户进行验证

- 用户填写的内容发送到后端, 后端进行验证, 验证通过则前端可以继续操作, 否则拒绝前端往购物车中添加食品的请求