華中科技大學

《软件工程》项目报告

题目：Android 外卖应用

课程名称：软件工程

专业班级：CS2304

组　　名：歪比巴卜

同组成员：学号：U202315540

姓名：曾程

学号：U202315594

姓名：贾柠泽

学号：U202315600

姓名：田知恒

学号：U202315614

姓名：周域楷

指导教师：瞿彬彬教师

报告日期：2025.12.4

计算机科学与技术学院

任 务 书

一 总体要求

1. 综合运用软件工程的思想，协同完成一个软件项目的开发，掌软件工程相关的技术和方法；
2. 组成小组进行选题，通过调研完成项目的需求分析，并详细说明小组成员的分工、项目的时间管理等方面。
3. 根据需求分析进行总体设计、详细设计、编码与测试等。

二 基本内容

根据给出的题目任选一题，自行组队，设计与开发中软件过程必须包括：

1. **问题概述、需求分析：**正确使用相关工具和方法说明所开发软件的问题定义和需求分析，比如 NABCD 模型，Microsoft Visio，StarUML 等工具 (20%)；
2. **原型系统设计、概要设计、详细设计：**主要说明所开发软件的架构、数据结构及主要算法设计，比如墨刀等工具（35%）；
3. **编码与测试：**编码规范，运用码云等平台进行版本管理，设计测试计划和测试用例（30%）；
4. **功能创新：**与众不同、特别吸引用户的创新（10%）；
5. **用户反馈：**包括用户的使用记录，照片，视频等（5%）。

目　录

1 问题定义

1.1 项目背景与意义

随着校园餐饮经济发展，外卖订餐成为在校大学生人群的刚需。当前校园外卖市场虽已形成规模，但仍存在食堂外卖服务缺失、支付手段单一等痛点。

本项目旨在打造一款外卖订餐 APP，通过整合学校餐饮资源，实现餐品搜索、下单支付等功能，为用户提供便捷的校内校外点餐体验。

1.1.1 项目 NABCD 模型

根据上述分析，我们的系统需要收集、存储并展示餐厅中的菜品信息，实现选菜、下单、支付流程和用户管理。项目的 NABCD 模型如下。

（1）N（Need 需求）

本软件的主要用户包括本校大学生、教职工等人员。在校大学生群体在餐饮方面存在着多个未被充分满足的“痛点”：

时间冲突：饭堂的用餐高峰期与下课时间高度重合，导致学生需要花费大量时间排队打饭，挤占宝贵的午休或学习时间。

空间限制：部分宿舍或教学楼距离饭堂较远，在天气恶劣（酷暑、严寒、雨雪）时，学生出门就餐的意愿较低。

支付不便：学生主要依赖校园卡消费，但校外餐厅只支持微信/支付宝。这样频繁切换支付方式是一种不便。

（2）A（Approach 做法）

我们的解决方案是开发一款专注于高校市场的聚合型外卖平台。项目 APP 基于 Android Studio 平台开发，通过 xml 文件实现前端 UI 显示，采用 Java 语言实现后端，商家菜品信息缓存在本地的 json 文件中，可联网更新。核心功能划分为首页（餐厅展示页），餐厅详情页，下单与支付流程，订单页面，“我的”页面共五个模块，可实现菜品展示、选择菜品、下单支付、订单管理、用户管理等功能。

（3）B（Benefit 好处）

我们的方案为学生、商家和平台自身都带来了明确且可量化的收益。

1. 对学生的收益：

节省时间（可量化）：平均每天节省 10-30 分钟的排队和往返时间，一周即

可节省约 1-3 小时，可用于学习、休息或社团活动。

提升便利性（可感知）：足不出户即可享用美食，尤其在恶劣天气下，体验感提升巨大。

2. 对合作商家的收益：

校内饭堂：分流高峰期压力，提升运营效率。

校外餐厅：精准触达大学生消费群体，开辟新的销售渠道，有效提升午晚餐高峰期的订单量。

3. 对平台的收益：

商业模式清晰：通过向合作商家抽取一定比例的佣金和收取少量配送费来实现盈利。

用户粘性高：深度绑定校园场景，支付方式便捷，容易形成用户习惯和口碑传播。

（4）C（Competitors 竞争）

本产品会面对美团、饿了么等外卖巨头平台的竞争，它们商家资源丰富，品牌认知度高，市场占有率高。而我们的优势在于立足校园学生群体，提供饭堂外卖服务，支持校园卡虚拟支付，丰富了外卖商家的选择。

面对饭堂现场就餐的竞争，我们花费的时间更短，提供足不出户的外卖体验，方便了大学生的饮食生活。

面对学生自建的外卖微信群的竞争，我们可以提供标准化、可靠、高效的平台服务，信息更集中、售后有保障。

（5）D（Delivery 推广）

推销校园外卖订餐平台主要考虑以下 2 个方面。

1. 产品推广：

①确定目标客户群体为大学生、教职工等在校内居住、生活、工作的人员。

②开展市场调研：了解行业需求与竞争对手（如美团、饿了么）情况，为产品定位和差异化提供准确依据。

③制定营销策略：包括线上线下宣传推广、推出针对目标客户的定制化方案等。

2. 系统支持维护：

①提供技术支持和解决方案：通过在线客服、电话支持等渠道，及时回答用

户疑问、解决问题。

②定期更新和升级：及时修复系统漏洞，持续改进系统性能和功能，提供更好的用户体验。

1.2 项目基本目标

根据上述分析，本项目旨在开发一款外卖订餐 APP，整合校内校外餐饮资源，为用户提供便捷的点餐体验。用户可以在 APP 中浏览不同类型的餐厅，查看菜单，搜索菜品，将食物添加到购物车，并进行结算。应用采用清晰的分类导航和直观的用户界面。

1.3 可行性分析

为了确保项目开展顺利，我们从技术和经济两方面分析了项目的可行性。

（1）技术可行性

本项目要求实现商家菜品展示、选择菜品、下单支付、订单管理、用户管理等功能，需要在前端完成相关页面的展示，在后端实现相关业务逻辑。同时，为了呈现出简单易用的安卓软件界面，还需要在 Android Studio 平台上进行模拟测试和安卓手机上的测试。这些需要使用的工具、技术（如 Java， Android Studio 平台上的开发）都有丰富的学习资源和文档资料。团队成员均为计算机学院本科生，具有丰富的计算机领域专业知识，如软件体系设计、前后端应用开发、软件测试与迭代等，在项目推进过程中具有良好的团队凝聚力和共同目标。

针对本项目，团队成员已有的技术积累包括 Android Studio 的配置使用、Java 语言编程、前后端开发等，具有合作设计开发项目系统并进行调试的专业能力。

目前项目已产出第一代可用系统，证明本项目具有充分的技术可行性。

（2）经济可行性

本项目在开发完成后，将先在校内进行小范围宣传推广，扩大项目的知名度，并首先在校内启动试用工作。在一段时间的试用期后，我们将收集用户反馈，对系统进行优化迭代后再次上线。

本项目盈利模式清晰，市场空间明确。如果能成功解决“校园卡支付”和“最后一公里配送”这两个核心痛点，就能构建起强大的护城河，避免与巨头正面竞争，从而获得可观的经济回报。但是需要解决与校方的合作问题，并准备好充足的资金以应对前期的市场培育和竞争。

综合上述分析，我们认为本项目的开发是必要且可行的。

1.4 人员管理和项目进度管理

通过系统地分析项目目标，本小组共同商议并制定了合理的人员分工安排与项目进度计划，通过对任务设置阶段性期限，确保项目有序推进。

项目的开发主要分为前期调研阶段、初步探索阶段、分模块实现阶段、测试阶段四个部分。

本项目组共包含 4 名成员，在不同阶段的不同任务中，详细人员分工安排如表 1-1 所示，工作量百分比各占 25%.

表 1-1 Android 外卖应用项目分工合作计划表

阶段	任务		负责人
阶段一：前期调研阶段	模型侧（技术调研）	阅读 Android 应用开发最佳实践文档，了解行业标准	田知恒
		调研移动电商应用的用户体验设计模式	周域楷
		查找适合外卖应用的数据存储方案	贾柠泽
		研究 Android UI 设计规范和 Material Design	田知恒
		阅读用户认证和权限管理相关文档	曾程
	前后端侧	进行系统的总体设计，完成需求分析、UML、系统架构图等	贾柠泽、曾程、田知恒、周域楷
		学习 Android 前端开发相关知识（Activity、Fragment、RecyclerView 等）	曾程、田知恒
		学习后端相关知识（网络请求、数据持久化、API 设计等）	周域楷、贾柠泽
		进行前端页面的初步设计和原型	曾程、田知恒
阶段二：初步探索阶段	模型侧（技术验证）	搭建 Android 项目基础框架，配置开发环境	贾柠泽
		实现简单的用户界面组件，验证 UI 设计思路	周域楷
		开发数据模型类（User、FoodItem、Order 等）	曾程
		实现基础的数据存储功能（JSON）	田知恒
		测试应用的基本功能流程，评估技术方案可行性	贾柠泽、曾程、田知恒、周域楷
	前后端侧	研究学习 RecyclerView 和适配器模式的使用	曾程
		调研网络请求框架（Retrofit、OkHttp 等）	曾程
		实现简单的数据交互功能演示	曾程
阶段三：分模块实现阶段	用户管理模块	实现用户注册登录功能	周域楷
		实现个人信息管理和修改密码功能	周域楷
		实现用户会话管理和自动登录功能	周域楷
		实现餐厅列表界面	贾柠泽

	餐厅浏览模块	实现不同类型餐厅的展示界面（汉堡、中餐、意大利餐等）	贾柠泽
		实现餐厅搜索功能	贾柠泽
	购物车模块	实现购物车功能（添加、删除、修改数量）	田知恒
		实现购物车结算功能	田知恒
		实现购物车数据的持久化存储	田知恒
	订单管理模块	实现订单创建功能	田知恒
		实现订单历史记录查询	田知恒
		实现订单详情查看功能	田知恒
		实现订单状态管理	田知恒
	公共功能模块	实现应用主题和样式统一	曾程
		实现错误处理和日志记录	周域楷
		实现应用设置功能	贾柠泽
		进行系统联调	贾柠泽
阶段四：测试阶段	测试阶段	编写单元测试和 UI 测试	周域楷
		进行功能测试，改正发现的 Bug	田知恒
		进行性能测试和优化	曾程
		进行兼容性测试（不同设备和 Android 版本）	周域楷

为了进行项目整体进度管理，我们引入甘特图对任务的具体时间进行分配，如图 1-1 所示。其中每一行代表一个子任务，括号内为该子任务的负责人。

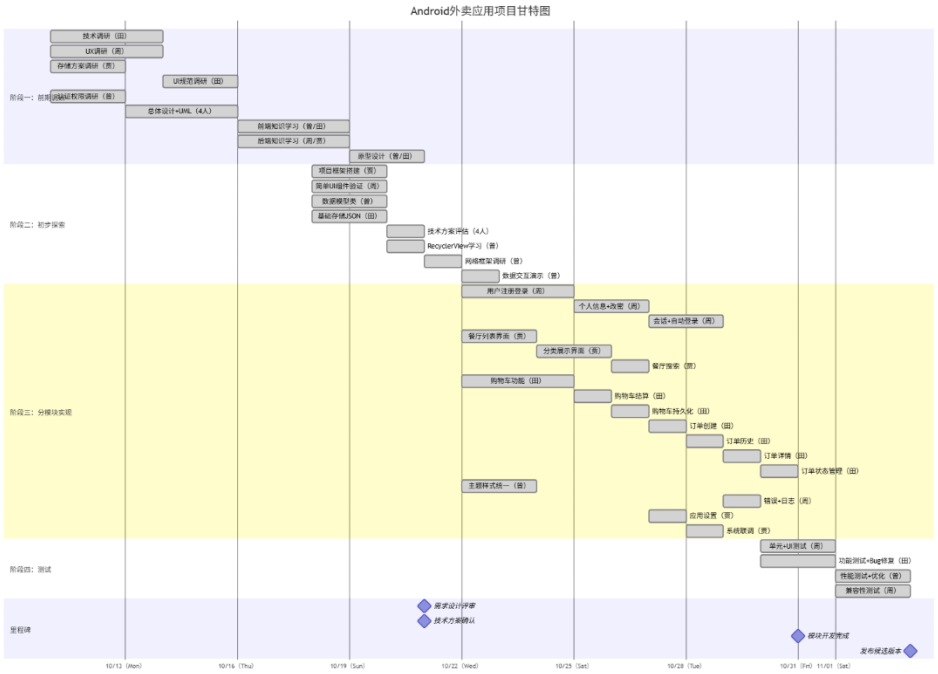

图 1-1 甘特图

2 需求分析

2.1 需求分析概述

该订餐软件的核心目标是：构建一个功能完整、用户体验流畅的线上到线下一体化餐饮服务平台。 系统需同时服务于消费者（用户） 和平台运营方，实现从菜品浏览、下单、支付到订单管理的全流程数字化。

经过前期的头脑风暴讨论，团队确定了项目开发的总体方向，思维导图 Mindmap 如图 2-1 所示。

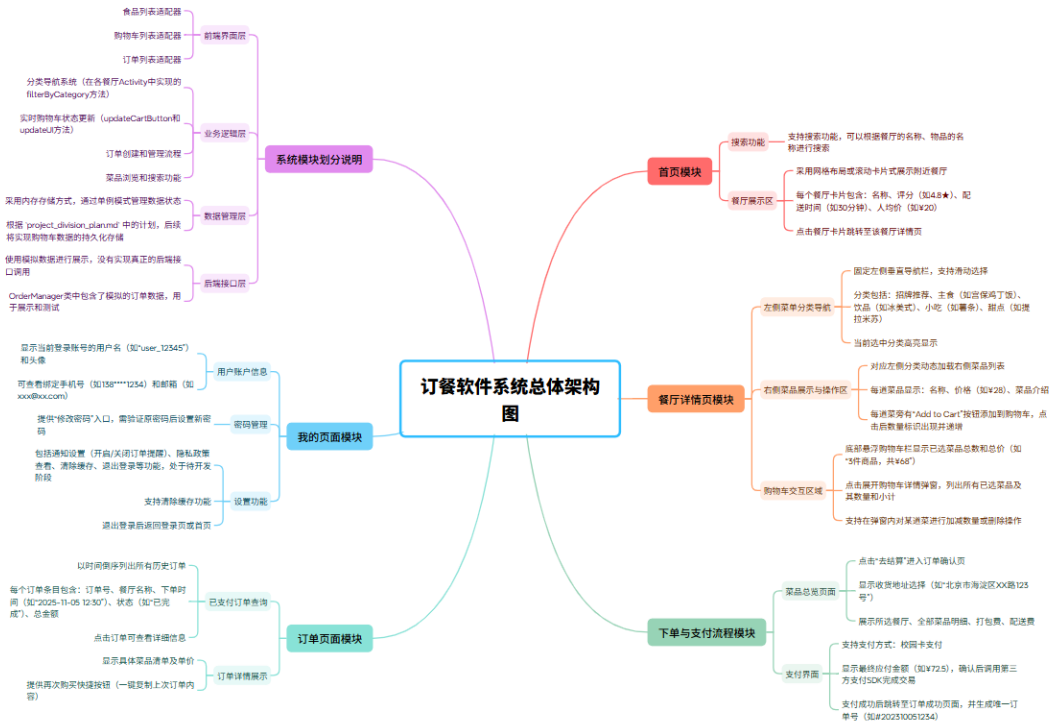

图 2-1 订单软件总体结构图

详细细节如图 2-2 到图 2-7 所示。

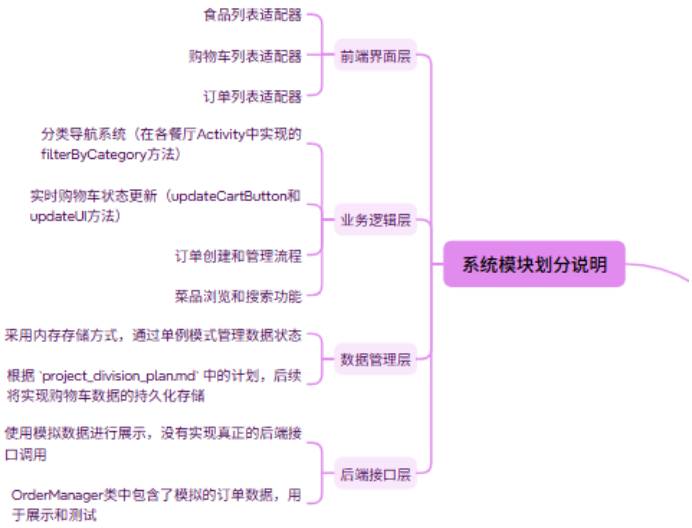

图 2-2 系统模块划分图

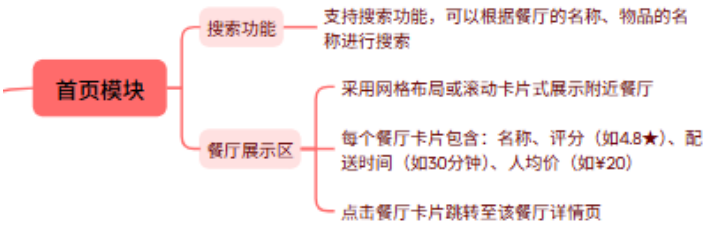

图 2-3 首页模块图

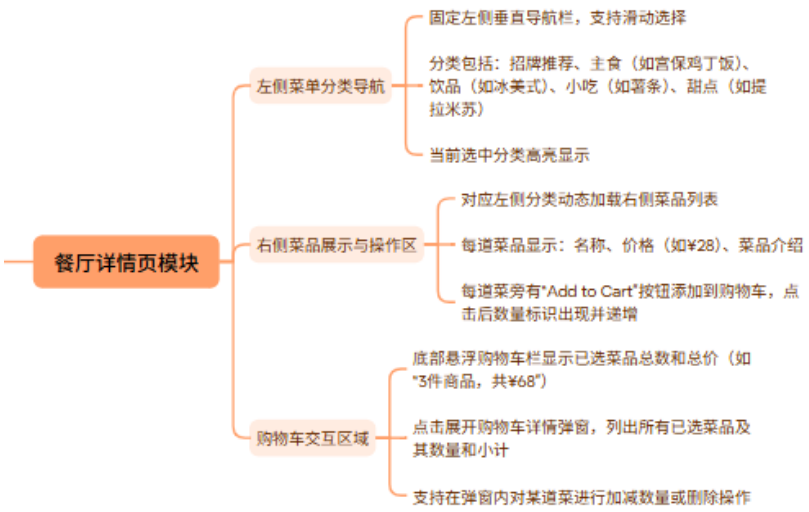

图 2-4 餐厅详情模块图

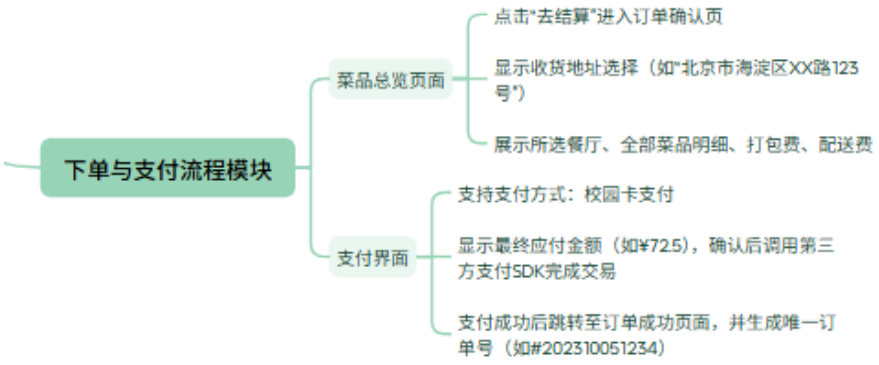

图 2-5 下单与支付流程图

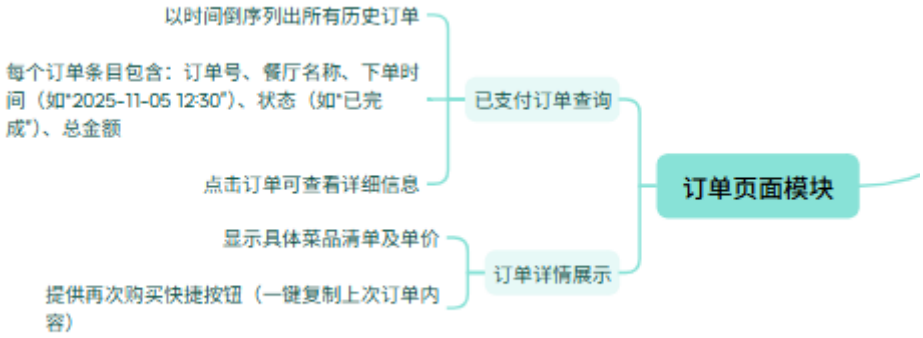

图 2-6 订单页面模块图

图 2-7 我的页面模块图

2.2 UML 相关需求分析图

UML 相关需求图总体图如图 2-8 所示。

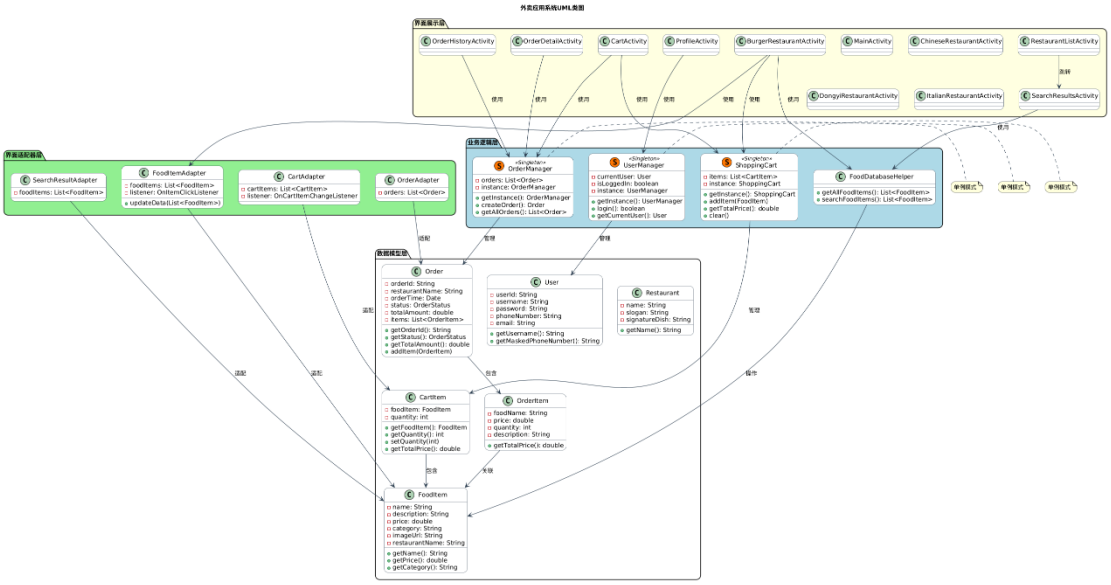

图 2-8 总体 UML 图

每个板块的详细信息如图 2-9 到图 2-12 所示。

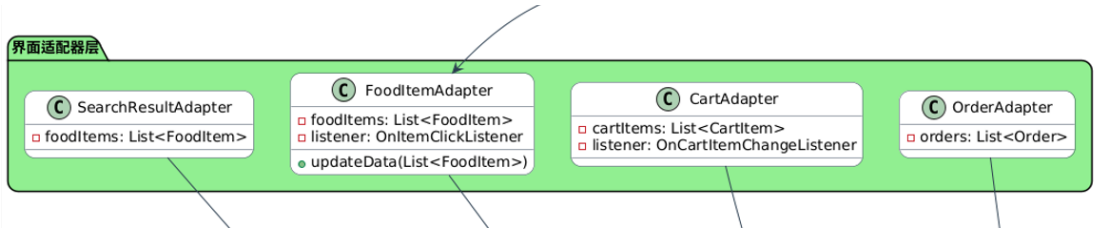

图 2-9 界面适配器层 UML 图

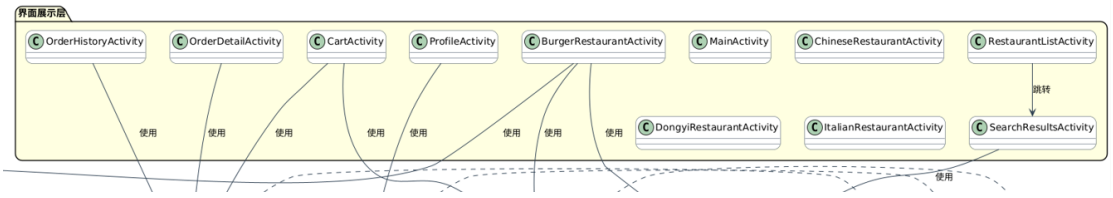

图 2-10 界面显示层 UML 图

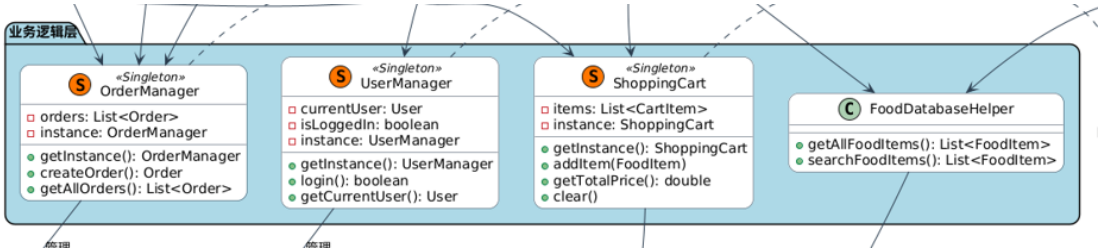

图 2-11 业务逻辑层 UML 图

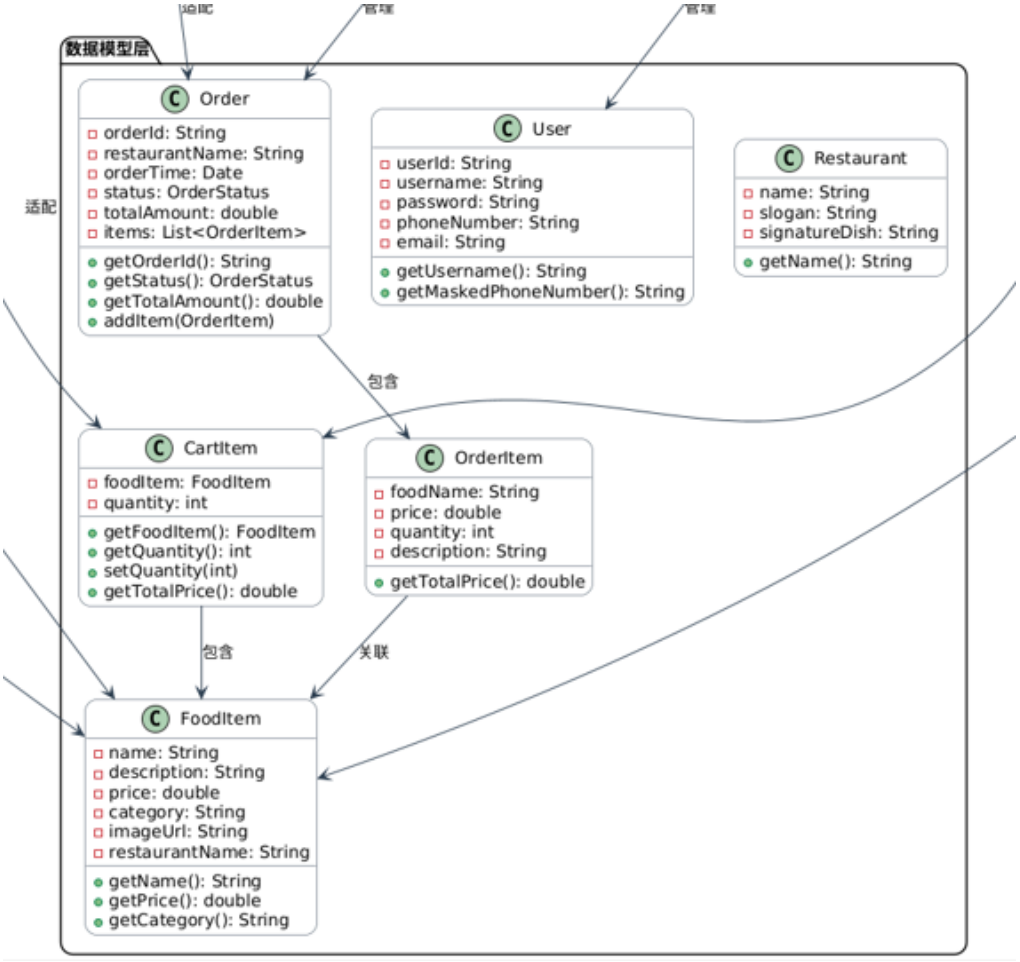

图 2-12 数据模型层 UML 图

2.3 原型系统设计

在正式进行开发工作前，我们运用 Calicat 工具设计了项目的原型系统（链接： https://www.calicat.cn/design/1994963620184604672 ）， 主要分为“餐厅列表页”、“餐厅详情页”、“购物车页面”、“订单历史页面”、四个部分， 以便于更准确直观地说明主要功能和用户交互界面，如图 2-13 所示。

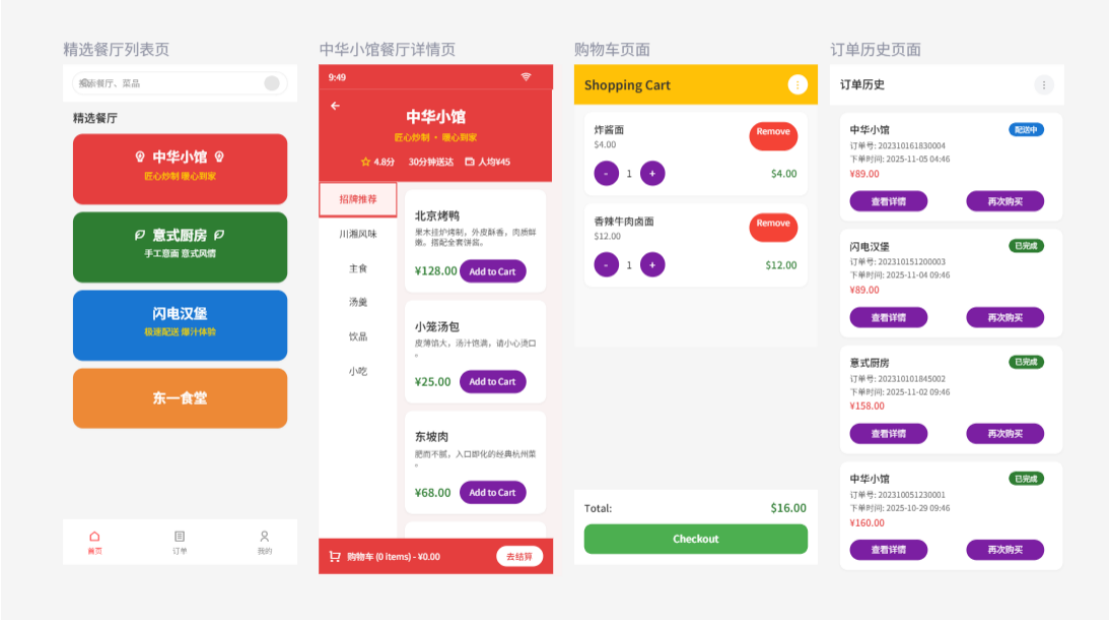

图 2-13 总体原型设计图

图 2-14 餐厅列表页原型图

图 2-15 餐厅详情页原型图

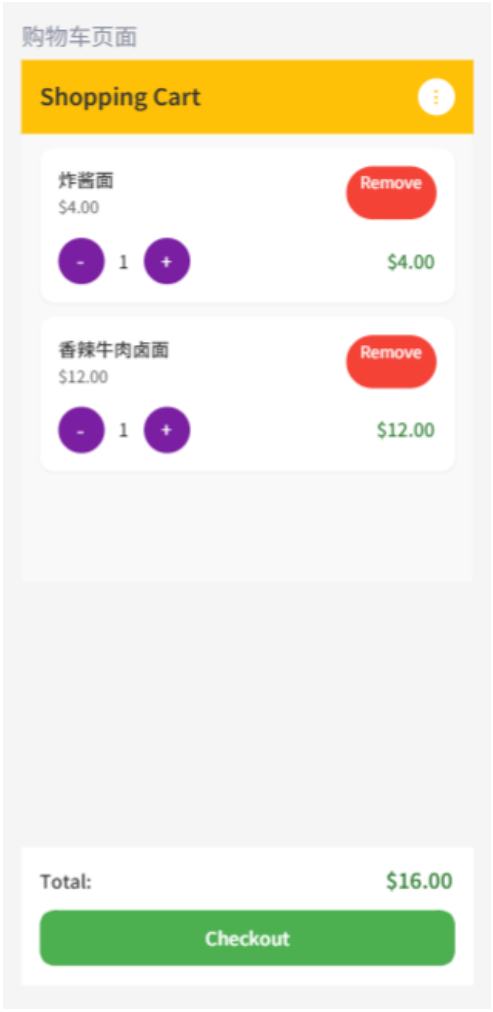

图 2-16 购物车页面原型图

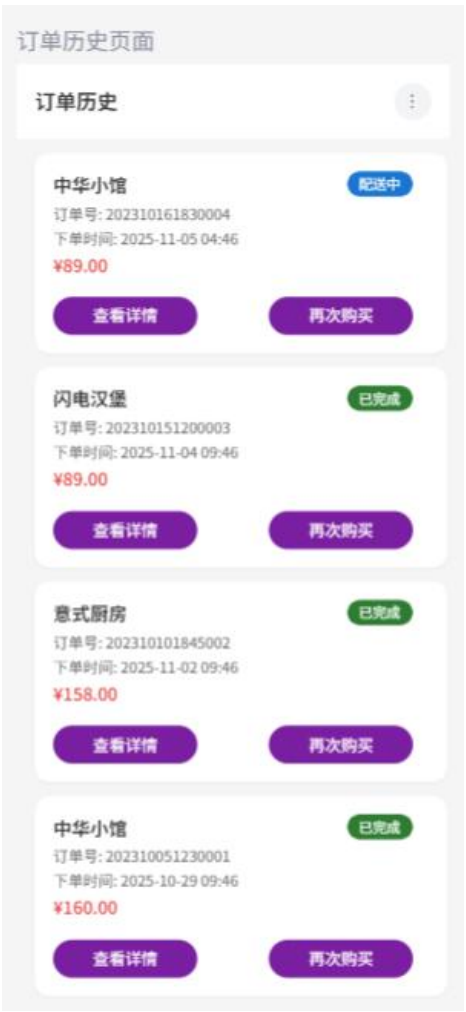

图 2-17 订单历史页面原型图

3 概要设计和详细设计

3.1 系统结构

（1）总述

本系统是一个完整的Android平台外卖点餐应用系统，采用本地化数据存储与内存状态管理相结合的架构设计。系统实现了从餐厅浏览、菜品选择、购物车管理到订单生成的全流程功能，同时提供了用户管理、搜索查询、个人中心等辅助功能模块。从前端到全栈的演进需要系统性地引入后端服务，同时保持客户端的优秀用户体验。建议采用渐进式重构策略，逐步引入网络层和服务端能力，最终实现一个现代化的云原生外卖平台。

功能模块：

1.用户管理模块：处理用户登录、注册、个人信息管理、密码修改等功能

2.餐厅与菜品模块：展示四大餐厅（中华小馆、意式厨房、闪电汉堡、东一食堂）的菜品信息，支持分类浏览

3.购物车模块：管理用户选中的菜品，实现添加、删除、修改数量、实时计算总价等功能

4.订单管理模块：生成订单、管理订单状态、查看历史订单、支持再次购买

5.搜索模块：实现跨餐厅的全局菜品搜索功能

6.个人中心模块：展示用户信息、管理缓存、提供设置选项

（2）系统架构图

系统采取典型的分层架构设计，各层职责清晰。

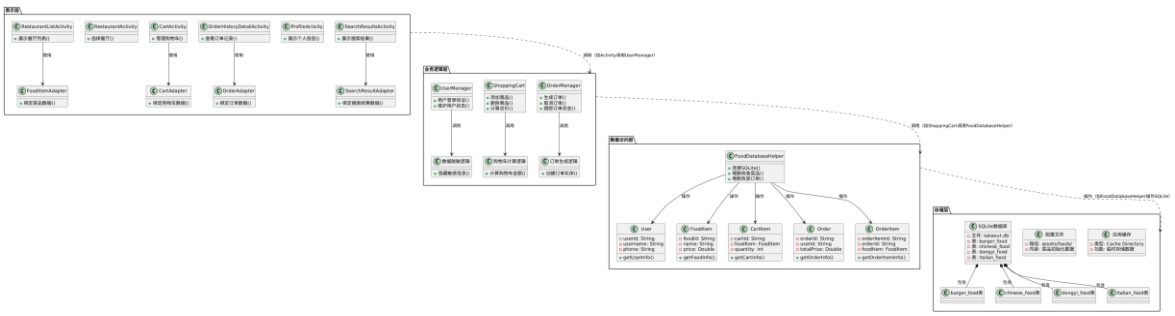

图 3-1 总体 UML 图

接口设计

1.接口设计原则遵循：

单一职责原则：每个接口都有明确的职责边界。

接口隔离原则：接口划分精细，避免臃肿。

依赖倒置原则：高层模块不依赖低层模块，都依赖于抽象。

开闭原则：对扩展开放，对修改封闭。

2. 接口特性：

类型安全：使用泛型确保类型安全。

可扩展性：接口设计考虑了未来的功能扩展。

兼容性：向后兼容，新版本接口兼容旧版本。

文档完整性：每个接口方法都有完整的文档说明。

异常处理：定义了明确的异常处理机制。

具体接口如下：

1.用户管理接口

```
public interface IUserManager {
    boolean login(String username, String password);                    // 用户登录
    boolean register(String username, String password,                  // 用户注册
                     String phoneNumber, String email);
    boolean changePassword(String oldPassword, String newPassword); // 修改密码
    User getCurrentUser();                                              // 获取当前用户
    void logout();                                                      // 退出登录
}
```

2.菜品数据接口

```
public interface IFoodRepository {
    List<FoodItem> getAllFoodItems(String tableName);                   // 获取所有菜品
    List<FoodItem> getFoodItemsByCategory(String tableName,     // 按分类获取菜品
                                          String category);
    List<FoodItem> searchFoodItems(String keyword);                     // 搜索菜品
    void refreshDataFromJson(String tableName,                          // 从 JSON 刷新数据
                             String jsonFilePath);
}
```

3.购物车接口

```
public interface IShoppingCart {
    void addItem(FoodItem foodItem);                                    // 添加菜品
    void removeItem(FoodItem foodItem);                                 // 删除菜品
    void updateQuantity(FoodItem foodItem, int quantity);       // 更新数量
    List<CartItem> getItems();                                          // 获取所有项
    double getTotalPrice();                                             // 计算总价
    int getItemCount();                                                 // 获取商品总数
    void clear();                                                       // 清空购物车
}
```

4. 订单管理接口

```
public interface IOrderManager {
    Order createOrder(String restaurantName,                    // 创建订单
                      List<CartItem> cartItems);
    List<Order> getAllOrders();                                  // 获取所有订单
    Order getOrderById(String orderId);                          // 根据 ID 获取订单
    void updateOrderStatus(String orderId,                       // 更新订单状态
                           Order.OrderStatus status);
    boolean cancelOrder(String orderId);                         // 取消订单
}
```

本系统的接口设计遵循了软件工程的高内聚低耦合原则，通过清晰的接口划分，实现了模块间的解耦，包括用户管理、菜品数据访问、购物车管理和订单管理。这些接口由相应的单例类实现，保证了数据的一致性和模块的可测试性。同时，我们为未来的网络扩展设计了RESTful API，便于后续功能拓展。

3.2 类图等

本系统采用基于面向对象设计原则的类结构，通过清晰的职责分离和单一职责原则构建了一套完整的类体系。系统类图主要由以下五个层次构成：

1. 实体类层：定义系统中的核心数据模型（User、FoodItem、Order等）

2. 管理类层：单例模式实现的核心业务管理器（UserManager、ShoppingCart等）

3. 数据访问层：处理数据持久化的辅助类（FoodDatabaseHelper）

4. 界面控制层：Android Activity和Fragment控制器

5. 适配器层：RecyclerView适配器，负责数据展示

本系统的类图设计充分体现了面向对象设计原则，通过清晰的类层次结构和合理的关系设计，构建了一个高内聚、低耦合的软件架构。主要特点包括：

1. 职责分离：每个类都有明确的单一职责

2. 模式应用：合理应用单例、适配器、回调等设计模式

3. 扩展性：类结构设计考虑了未来的功能扩展

4. 可维护性：类关系清晰，便于理解和维护

通过这样的类图设计，系统不仅满足了当前的功能需求，也为未来的演进奠定了良好的架构基础。在实际开发中，可以根据需要进一步优化和调整类结构，以适应不断变化的需求。图3-2清晰地表示了各层之间的关系。

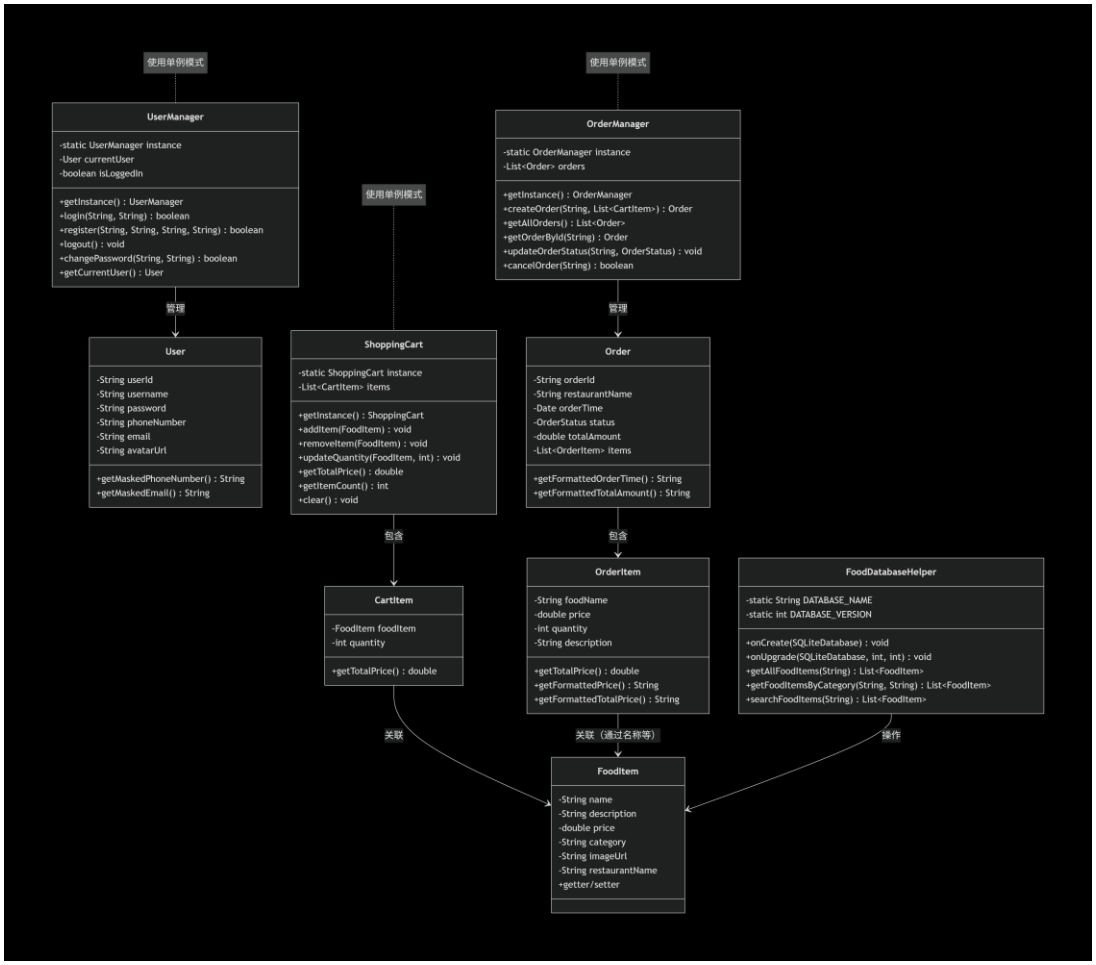

图 3-2 详细 UML 图

3.3 关键数据结构定义

（1）主要数据结构定义

1. 用户数据结构（User）

数据用途：存储和管理用户的个人信息，包括登录凭证，联系方式和个性化设置。

数据项	数据类型	长度/范围	约束条件	说明
userId	String	1-20字符	主键，唯一标识	用户唯一识别码，如"wangbian"
username	String	2-50字符	非空，允许中英文	用户显示名称，如"王扁"
password	String	6-20字符	非空	登录密码（实际应加密存储）
phoneNumber	String	11位	中国大陆手机号格式	用户手机号码，用于联系和验证
email	String	5-100字符	标准邮箱格式	用户电子邮箱
avatarUrl	String	0-500字符	可选，默认值	用户头像图片URL地址

图 3-3 用户数据结构图

2.菜品数据结构（Fooditem）

数据用途：描述餐厅中的菜品信息，包括基本属性，价格和分类信息。

数据项	数据类型	长度/范围	约束条件	说明
name	String	1-100字符	非空，主键之一	菜品名称，如"北京烤鸭"
description	String	0-500字符	可选	菜品详细描述
price	double	0.01-9999.99	非负，两位小数	菜品单价（元）
category	String	1-50字符	非空	菜品分类，如"招牌推荐"
imageUrl	String	0-500字符	可选	菜品展示图片URL
restaurantName	String	1-50字符	非空	所属餐厅名称

图 3-4 菜品数据结构图

数据库表映射：

```
CREATE TABLE food_items (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    description TEXT NOT NULL,
    price REAL NOT NULL CHECK(price >= 0),
    category TEXT NOT NULL,
    image_url TEXT DEFAULT '',
    restaurant_name TEXT NOT NULL
);
```

3.购物车项数据结构（Cartltem）

数据用途：表示用户在购物车中选择的菜品及其数量，用于订单计算。

数据项	数据类型	长度/范围	约束条件	说明
foodItem	FoodItem	对象引用	非空	关联的菜品对象
quantity	int	1-99	正整数	购买数量

图 3-5 购物车数据结构图

剩余的订单数据结构，订单项数据结构，餐厅数据结构由于篇幅限制不于此展示，后面介绍数据关联设计。

（2）数据关联详细说明

1.用户与订单关联（一对多）

```
用户(User) → 订单(Order)：一对多关系
public class Order {
    // 建议添加字段
    private String userId;  // 关联用户 ID
    // 通过 userId 获取用户的所有订单
    public static List<Order> getOrdersByUserId(String userId) {
        // 从 OrderManager 中筛选
    }
}
```

2. 订单与订单项关联(一对多)

```
// 菜品在系统中的多重关联关系：
1. 菜品 → 购物车项：一对多（一个菜品可在多个购物车中）
   FoodItem → CartItem
2. 菜品 → 订单项：一对多（一个菜品可在多个订单中）
   FoodItem → OrderItem
3. 菜品 → 餐厅：多对一（一个菜品属于一个餐厅）
   FoodItem → Restaurant
// 关联管理：
public class FoodDatabaseHelper {
    // 通过餐厅名称获取所有菜品
    public List<FoodItem> getFoodItemsByRestaurant(String restaurantName) {
        // SQL 查询：SELECT * FROM food_items WHERE restaurant_name = ?
    }
    // 通过菜品名称获取所属餐厅
    public String getRestaurantByFoodName(String foodName) {
        // SQL 查询：SELECT restaurant_name FROM food_items WHERE name = ?
    }
}
```

3.菜品与多实体的关联

```
// 结构关联：一个订单包含多个订单项
订单(Order) → 订单项(OrderItem)：聚合关系
// 关联实现：
public class Order {
    private List<OrderItem> items;  // 聚合订单项

    // 添加订单项
    public void addItem(OrderItem item) {
        items.add(item);
        // 更新订单总金额
        this.totalAmount += item.getTotalPrice();
    }
    // 获取订单项统计
    public int getTotalItemCount() {
        int count = 0;
        for (OrderItem item : items) {
            count += item.getQuantity();
        }
        return count;
    }
}
```

4.购物车与购物车项的关联

```
// 购物车管理购物车项的完整生命周期
public class ShoppingCart {
    private List<CartItem> items;  // 聚合购物车项
    // 关联操作：
    // 1. 添加关联
    public void addItem(FoodItem foodItem) {
        // 检查是否已存在
        for (CartItem item : items) {
            if (item.getFoodItem().getName().equals(foodItem.getName())) {
                item.setQuantity(item.getQuantity() + 1);
                return;
            }
        }
        // 创建新的关联
        items.add(new CartItem(foodItem, 1));
    }
    // 2. 更新关联
    public void updateQuantity(FoodItem foodItem, int newQuantity) {
        // 更新数量关联
    }
    // 3. 移除关联
    public void removeItem(FoodItem foodItem) {
        // 移除关联关系
    }
}
```

本系统的数据结构设计充分考虑了外卖点餐业务的需求特点，具有以下优势：

业务贴合度高：数据结构紧密贴合外卖点餐的业务流程

扩展性强：通过合理的关联设计支持未来功能扩展

性能优化：通过索引、缓存等机制提升数据访问效率

数据完整性：通过约束和验证确保数据质量

用户体验优化：数据结构支持快速响应和流畅交互

这些数据结构为系统的稳定运行和良好用户体验提供了坚实的基础，同时也为未来的功能演进预留了充足的空间。

3.4 关键算法设计

（1）订单号生成算法

算法设计目标：生成全局唯一的订单标识符。保证订单号的可读性和有序性防止订单号重复和冲突。

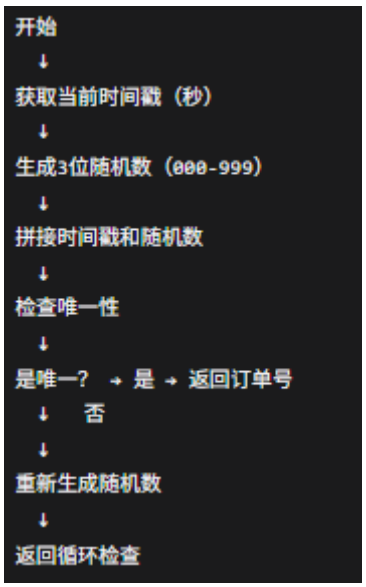

图3-6 订单生成算法图

（2）菜品搜索算法

算法设计目标：支持多字段模糊搜索，提供搜索结果排序和过滤，保证搜索效率和准确性。

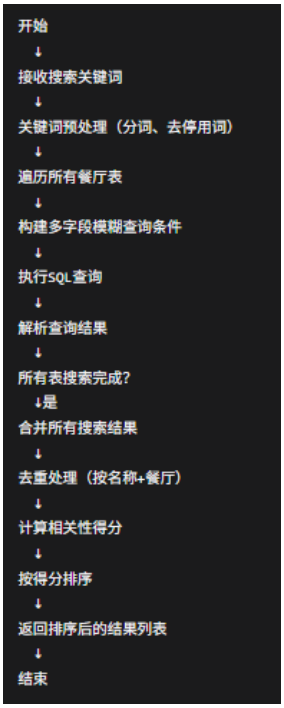

图3-7 搜索算法图

（3）订单状态流转验证算法

算法设计目标：确保订单状态转换的合法性，防止非法状态转换，提供状态转换的历史记录。

状态转换规则表

当前状态	允许转换的状态	说明
PENDING（待付款）	PAID, CANCELLED	可以支付或取消
PAID（已付款）	PREPARING, CANCELLED	可以开始制作或取消
PREPARING（制作中）	DELIVERING, CANCELLED	可以开始配送或取消
DELIVERING（配送中）	COMPLETED	只能完成订单
COMPLETED（已完成）	无	终态，不能转换
CANCELLED（已取消）	无	终态，不能转换

图3-8 状态转化规则图

图3-9 订单状态流转验证算法图

算法设计总结：

实用性：所有算法都针对实际业务需求设计。

效率性：考虑时间复杂度和空间复杂度，保证系统性能。

可扩展性：算法设计考虑未来功能扩展。

健壮性：包含异常处理和边界条件检查。

可维护性：代码结构清晰，注释完整。

这些算法共同构成了系统的核心处理逻辑，确保了外卖点餐应用的高效、稳定运行。在实际应用中，可以根据具体需求对这些算法进行进一步优化和调整。

3.5 数据管理说明

（1）多级存储架构

本系统采用四级数据结构，根据数据的使用频率，重要性，访问速度要求，将数据存储在不同的介质和位置：

图3-10存储架构图

（2）各存储层级详细说明

1.内存存储

存储内容：

用户登录状态和会话信息。

当前用户的购物车内容。

应用运行时临时数据。

页面间传递的数据对象。

生命周期管理：

应用启动时初始化。

应用运行时维护。

应用退出或用户注销时清空。

支持异常恢复（如应用崩溃后恢复购物车）。

代码实现：

```
public class SessionManager {
    private static SessionManager instance;
    private User currentUser;                    // 当前登录用户
    private ShoppingCart shoppingCart;           // 购物车状态
    private Map<String, Object> sessionData;   // 会话数据
    // 临时数据缓存（15 分钟有效期）
    private Map<String, CacheEntry> tempCache;
    public void saveToSession(String key, Object value) {
        sessionData.put(key, value);
        // 设置 15 分钟过期
            CacheEntry entry = new CacheEntry(value,
              System.currentTimeMillis() + 15 * 60 * 1000);
        tempCache.put(key, entry);
    }
```

```
    public Object getFromSession(String key) {
        CacheEntry entry = tempCache.get(key);
        if (entry != null && entry.isExpired()) {
            tempCache.remove(key);
            sessionData.remove(key);
            return null;
        }
        return sessionData.get(key);
    }
}
```

2.SQLite 数据库存储（Persistent Storage）

数据库设计：

```
-- 主数据库：takeout.db，版本 11
CREATE TABLE IF NOT EXISTS {table_name} (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,                          -- 菜品名称
    description TEXT NOT NULL,                   -- 菜品描述
    price REAL NOT NULL CHECK(price >= 0), -- 菜品价格（非负）
    category TEXT NOT NULL,                      -- 菜品分类
    image_url TEXT DEFAULT '',                   -- 菜品图片 URL
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- 创建时间
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- 更新时间

    -- 约束：同一餐厅内菜品名称唯一
    CONSTRAINT unique_food_name UNIQUE (name)
);
-- 索引优化（提升查询性能）
CREATE INDEX IF NOT EXISTS idx_category ON {table_name}(category);
CREATE INDEX IF NOT EXISTS idx_price ON {table_name}(price);
CREATE INDEX IF NOT EXISTS idx_created ON {table_name}(created_at);
```

数据库性能优化：

索引优化：为经常查询的字段创建索引。

查询优化：使用预编译语句，避免 N+1 查询。

连接管理：合理管理数据库连接，避免泄露。

事务管理：批量操作使用事务，保证原子性。

3.JSON 配置文件存储（Configuration Storage）

JSON 配置文件存储在应用的 Assets 目录（app/src/main/assets/）中，随 APK 打包发布。Assets 目录中的文件在安装时会被原样拷贝到设备上，只能读取不能写入，除非在运行时拷贝到可写目录

文件结构设计：

```
assets/
├── foods/                          # 菜品数据目录
│   ├── chinese_foods.json          # 中餐菜品配置
│   ├── burger_foods.json           # 汉堡菜品配置
│   ├── dongyi_foods.json           # 东一食堂菜品配置
│   └── italian_foods.json          # 意式菜品配置
├── config/                         # 应用配置目录
│   ├── restaurants.json            # 餐厅配置
│   └── themes.json                 # 主题配置
└── i18n/                           # 国际化目录（扩展）
```

优点：

版本控制友好：文本文件便于 Git 等版本管理工具管理。

易于维护：非技术人员也可修改配置，无需重新编译代码。

结构灵活：JSON 支持嵌套、数组等复杂结构。

跨平台兼容：JSON 是通用数据交换格式，便于多平台共享。

限制：

Assets 只读：运行时不能修改 Assets 中的文件。

性能一般：JSON 解析比二进制格式慢。

安全性较低：配置内容在 APK 中可见，不能存储敏感信息。

无数据类型校验：运行时才能发现格式错误。

适用场景总结：

应用初始化数据：如默认菜品、餐厅配置

UI 配置和主题：如颜色方案、字体大小

功能开关和参数：如实验性功能、业务参数

多语言文本资源：如界面文本、错误提示

4.应用缓存存储

应用缓存存储在设备的内置存储缓存目录（/data/data/<package_name>/cache/）或外部存储缓存目录中。缓存目录中的文件系统会自动管理，当存储空间不足时优先清理，用户也可手动清除。

```
cache/
├── images/                    # 图片缓存
│   ├── restaurant_icons/      # 餐厅图标
│   ├── food_images/           # 菜品图片
│   └── user_avatars/          # 用户头像
├── json/                      # 数据缓存
│   ├── search_results/        # 搜索结果缓存
│   └── food_lists/            # 菜品列表缓存
└── temp/                      # 临时文件
    ├── downloads/             # 下载文件
    └── uploads/               # 上传文件
```

存储策略：按内容类型和用途分类存储。

清理策略：LRU 算法，定期清理过期文件。

文件命名：MD5 哈希命名，避免文件名冲突。

优点：

性能提升：减少网络请求和数据库查询，提升响应速度。

离线支持：缓存数据支持离线访问基本功能。

节省流量：减少重复数据下载，节省用户流量。

负载均衡：减轻服务器和数据库压力。

限制：

数据一致性问题：缓存数据可能过期，与实际数据不一致。

存储空间占用：缓存文件占用设备存储空间。

缓存管理复杂：需要合理的缓存策略和清理机制。

安全隐患：缓存可能泄露敏感信息。

适用场景总结：

网络资源缓存：如图片、CSS、JS 等静态资源。

API 响应缓存：如菜品列表、搜索结果的缓存。

计算密集型结果缓存：如搜索结果排序、推荐算法结果。

临时数据存储：如表单草稿、未提交的数据。

4 实现与测试

4.1 实现环境与代码管理

（1）硬件环境配置：

设备规格：

处理器：Intel Core i7-12700H

内存：16GB DDR4 3200MHz

存储：512GB NVMe SSD

显示器：1920×1080

测试设备：

OnePlus 11 (Android 14)

处理器：骁龙8

开发系统：Windows 11

目标系统：Android 8.0 (API 26) 至 Android 14 (API 33)

虚拟环境：依托于Android Studio的虚拟机Google Pixel9

（2）代码管理

使用git作为版本控制软件，依托于github平台进行在线代码托管，代码仓库地址为https://github.com/ElysiaTT/takeout.git，开发过程中依托于jetbrains IDE中提供的code with me功能共同修改代码文件，体现在平台上由田知恒同学的账号提交。

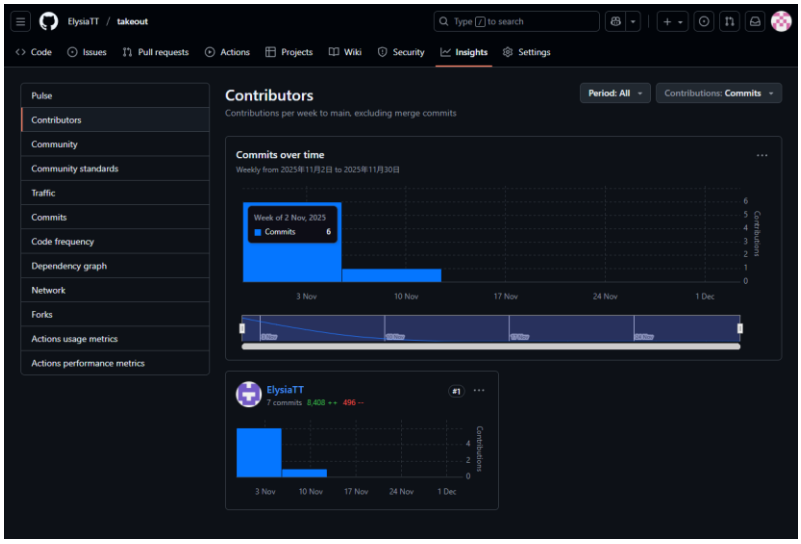

图4-1 代码管理图

下图是仓库项目概览。

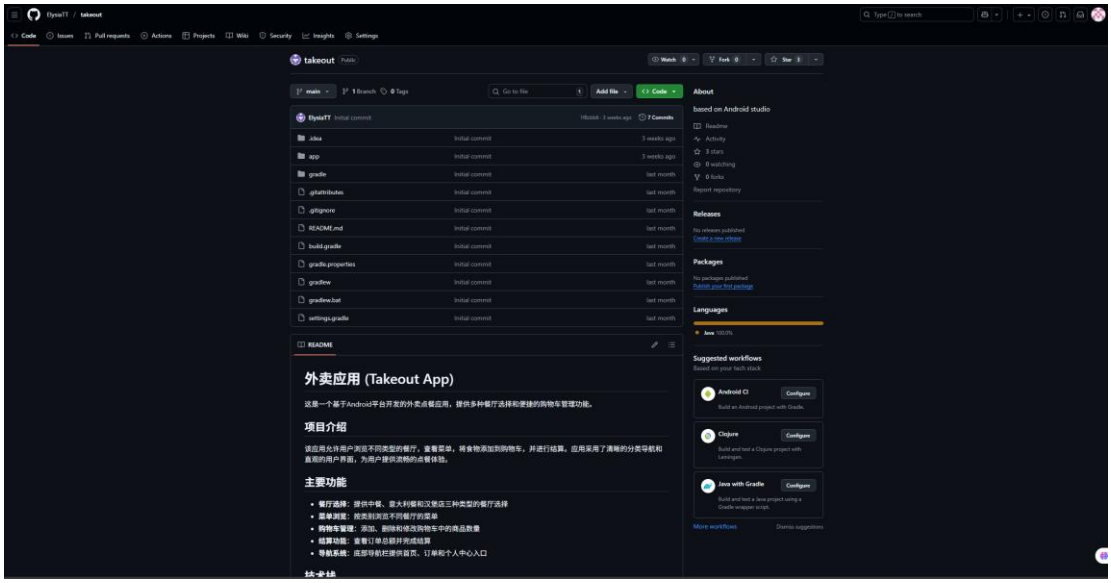

图4-2仓库项目概览图

4.2 关键函数说明

（1）核心业务函数

1. 函数名称：OrderManager.createOrder(String restaurantName, List<CartItem> cartItems)

功能说明：核心订单创建函数，将购物车内容转换为正式订单

实现逻辑：

生成唯一订单号（时间戳+随机数）；

计算购物车总金额；

创建 Order 对象并设置初始状态为"已付款"；

将 CartItem 转换为 OrderItem 添加到订单；

保存订单到订单历史。

调用关系：CartActivity.checkout() → OrderManager.createOrder()

涉及模块：购物车模块、订单模块、支付模块

2. 函数名称：ShoppingCart.addItem(FoodItem foodItem)

功能说明：向购物车添加商品，支持自动合并相同菜品

实现逻辑：

遍历购物车检查是否已有相同菜品

如存在则增加数量，否则创建新购物车项

更新购物车总价和商品数量

触发 UI 更新事件

调用关系：FoodItemAdapter.onAddToCart() → ShoppingCart.addItem()

涉及模块：菜品展示模块、购物车模块、UI 更新模块

（2）用户管理函数

1. 函数名称：UserManager.login(String username, String password)

功能说明：用户登录验证函数

实现逻辑：

验证用户名密码格式

检查用户凭证（当前为硬编码验证）

设置登录状态和当前用户

初始化用户相关数据（购物车、订单）

调用关系：LoginActivity.onLoginClick() → UserManager.login()

涉及模块：认证模块、用户状态管理、数据初始化

2. 函数名称：UserManager.changePassword(String oldPassword, String newPassword)

功能说明：修改用户密码

实现逻辑：

验证旧密码是否正确

验证新密码复杂度

更新用户密码

返回操作结果

调用关系：

ChangePasswordActivity.onConfirm() → UserManager.changePassword()

涉及模块：安全模块、用户配置管理

（3）数据访问函数

1. 函数名称：FoodDatabaseHelper.searchFoodItems(String keyword)

功能说明：全局菜品搜索函数

实现逻辑：

关键词预处理（分词、去停用词）

在所有餐厅表中执行模糊查询

合并搜索结果并按相关性排序

返回去重后的搜索结果列表

调用关系：

SearchResultsActivity.performSearch()→FoodDatabaseHelper.searchFoodItems()

涉及模块：搜索模块、数据库模块、数据排序模块

2. 函数名称：FoodDatabaseHelper.getFoodItemsByCategory(String tableName, String category)

功能说明：按分类获取菜品

实现逻辑：

根据餐厅类型选择对应数据表

执行 SQL 分类查询

将查询结果转换为 FoodItem 对象列表

缓存查询结果提高性能

调用关系：

Activity.filterByCategory()→FoodDatabaseHelper.getFoodItemsByCategory()

涉及模块：菜品展示模块、分类筛选、数据库查询

（4）UI 交互函数

1. 购物车操作函数

CartAdapter.onQuantityChanged(CartItem item, int newQuantity)

功能说明：处理购物车商品数量变化

实现逻辑：

验证新数量的有效性（>0）

更新购物车中对应商品数量

重新计算购物车总价

刷新购物车 UI 显示

调用关系：CartAdapter 按钮点击 → ShoppingCart.updateQuantity()

涉及模块：购物车 UI、购物车业务逻辑、价格计算

2. 餐厅分类函数

ChineseRestaurantActivity.filterByCategory(String category)

功能说明：餐厅内菜品分类筛选

实现逻辑：

根据分类名称从数据库获取菜品

更新当前菜品列表

刷新 RecyclerView 显示

更新分类导航 UI 状态

调用关系：分类导航点击 → filterByCategory() → FoodDatabaseHelper 查询

涉及模块：分类导航、菜品展示、数据筛选

（3）数据处理函数

1. 数据脱敏函数

User.getMaskedPhoneNumber()

功能说明：手机号脱敏显示

实现逻辑：

验证手机号格式（11 位）

保留前 3 位和后 4 位

中间 4 位替换为"****"

返回格式化字符串

调用关系：ProfileActivity 显示用户信息时自动调用

涉及模块：隐私保护、数据格式化

2. 价格计算函数

CartItem.getTotalPrice()

功能说明：计算购物车单项总价

实现逻辑：

获取菜品单价

获取购买数量

计算单价×数量

返回计算结果（保留两位小数）

调用关系：购物车总价计算时调用

涉及模块：价格计算、购物车统计

（4）函数间调用关系图

```
启动流程:
MainActivity.onCreate()
    ├── setupFoodItems()          // 初始化菜品数据
    ├── setupRecyclerView()       // 设置列表展示
    └── updateCartButton()        // 更新购物车按钮

用户操作流程:
餐厅Activity.onAddToCartClick()
    ├── ShoppingCart.addItem()    // 添加到购物车
    ├── updateCartButton()        // 更新UI显示
    └── 显示Toast提示

购物车管理流程:
CartActivity.onQuantityChanged()
    ├── ShoppingCart.updateQuantity()  // 更新数量
    ├── calculateCartTotal()           // 重新计算总价
    └── CartAdapter.notifyItemChanged() // 刷新UI

下单结算流程:
CartActivity.checkout()
    ├── OrderManager.createOrder()     // 创建订单
    ├── ShoppingCart.clear()           // 清空购物车
    ├── 显示订单成功提示
    └── finish()返回主页

搜索流程:
SearchResultsActivity.performSearch()
    ├── FoodDatabaseHelper.searchFoodItems() // 执行搜索
    ├── 处理搜索结果
    └── SearchResultAdapter更新显示

个人中心流程:
ProfileActivity.onResume()
    ├── UserManager.getCurrentUser()   // 获取用户信息
    ├── 加载用户信息到UI
    └── updateCacheSize()              // 更新缓存显示
```

图4-3 函数调用关系图

（5）关键函数分类表

函数类别	核心函数	主要功能	调用频率
订单相关	createOrder()	创建新订单	低
	getAllOrders()	获取订单历史	中
购物车相关	addItem()	添加商品	高
	updateQuantity()	修改数量	中
	getTotalPrice()	计算总价	高
用户相关	login()	用户登录	低
	changePassword()	修改密码	低
数据查询	searchFoodItems()	菜品搜索	中
	getFoodItemsByCategory()	分类查询	高
UI交互	filterByCategory()	分类筛选	高
	onQuantityChanged()	数量修改	中

图4-4关键函数分类图

（6）函数设计特点

1. 单一职责原则

每个函数只负责一个明确的业务功能

函数命名清晰反映其功能意图

函数长度控制在合理范围内

2. 可复用性设计

通用函数独立封装（如价格计算、数据脱敏）

业务逻辑函数可在多个模块调用

参数设计支持多种使用场景

3. 错误处理机制

关键函数包含参数验证

数据库操作有异常处理

用户输入有格式校验

4. 性能考虑

高频操作函数优化算法复杂度

数据库查询使用缓存机制

批量操作减少数据库交互次数

通过这些关键函数的协同工作，系统实现了完整的业务逻辑流程，从用户登录、菜品浏览、购物车管理到订单生成，每个环节都有对应的函数提供支持。函数间通过清晰的调用关系和数据传递，确保了系统的稳定运行和良好的用户体验。

4.3 测试计划和测试用例

（1）采用分层测试策略，包括单元测试、集成测试和UI测试：

单元测试：针对核心函数进行测试

集成测试：测试模块间的交互

UI测试：测试用户界面和交互流程

（2）测试环境

测试设备：Android模拟器

测试工具：Android Studio自带的测试框架

测试数据：预定义的测试用户和菜品数据

（3）关键模块测试用例

模块1：购物车管理模块

测试目标：　验证购物车的添加、删除、数量修改和计算功能

预置条件：用户已登录，购物车为空

测试步骤：

在餐厅页面点击"炸酱面"的"加入购物车"按钮

再次点击"香辣牛肉卤面"的"加入购物车"按钮

预期结果：

购物车显示2项商品

"炸酱面"数量为1

"香辣牛肉卤面"数量为1

总价正确计算

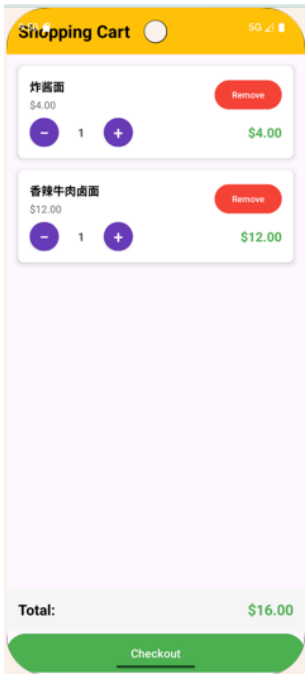

图4-5 购物车管理模块测试图

模块2：订单管理模块

测试目标：验证订单生成、查看和状态管理功能

预置条件：购物车中有商品

测试步骤：

进入购物车页面

点击"结算"按钮

查看订单确认页面

预期结果：

生成唯一的订单号

订单总价与购物车总价一致

购物车被清空

显示订单成功提示

图4-6 订单管理模块测试图

模块3：搜索功能模块

测试目标： 验证全局搜索功能的准确性和性能

预置条件：数据库中有菜品数据

测试步骤：

在搜索框输入"汉堡"

查看搜索结果

点击搜索结果中的"添加"按钮

预期结果：

显示所有包含"汉堡"的菜品

搜索结果包含菜品名称、价格、餐厅信息

可以正常添加到购物车

图4-7 搜索功能模块测试图

4.4 结果分析

经过全面的测试，系统各项功能基本满足需求规格说明书中的要求：

功能完整性：所有核心功能（餐厅浏览、菜品选择、购物车管理、订单生成）均实现并通过测试

性能表现：在模拟器上运行流畅，页面加载时间<1秒，搜索响应时间<1秒

稳定性：经过边界条件测试和异常操作测试，系统未出现崩溃情况

用户体验：界面交互流畅，反馈及时，符合用户操作习惯

5 总结

5.1 用户反馈

本项目开发完成后，我们将项目系统介绍给一些同学进行使用，使用反馈照片如下。

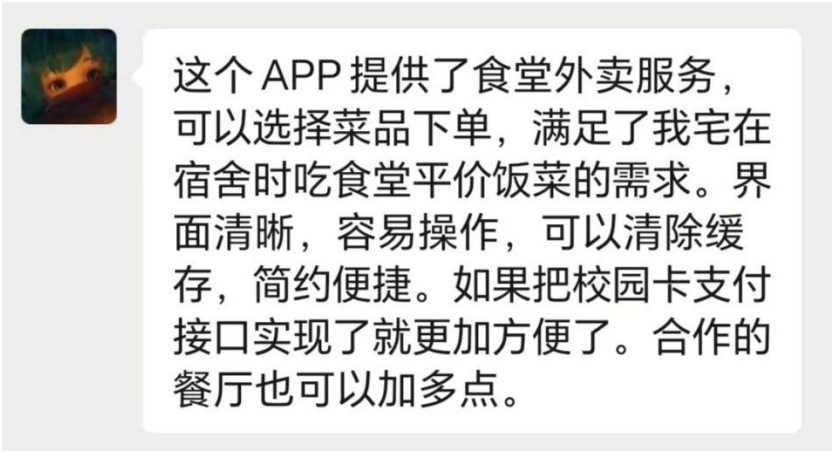

图 5-1 使用反馈图 1

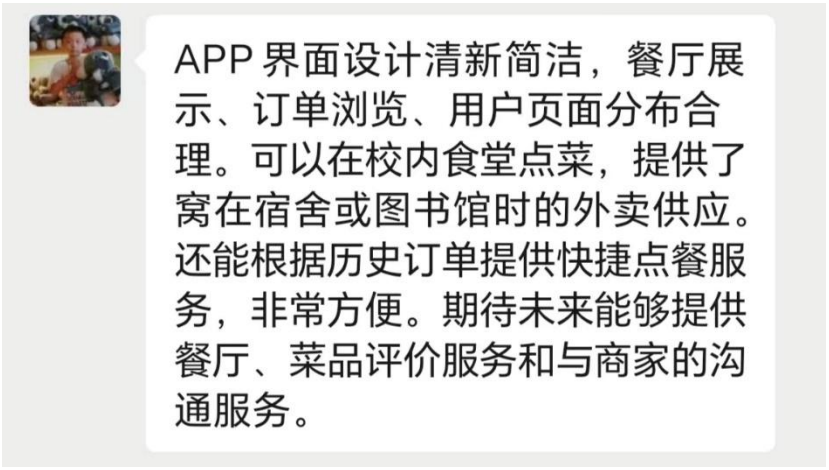

图 5-2 使用反馈图 2

5.2 全文总结

（1）周域楷同学

在软件工程项目的实现中，我主要负责进行用户管理模块的前端界面、后端逻辑的编写和调试，以及软件测试编写的相关工作。

具体工作如下：

①调研移动电商应用的用户体验设计模式

②实现简单的用户界面组件，验证 UI 设计思路

③在用户管理模块实现用户注册登录、个人信息管理和修改密码、用户会话管理和自动登录功能

④编写单元测试和 UI 测试

⑤进行兼容性测试（不同设备和 Android 版本）

（2）曾程同学

本项目完整实现了一个基于Android平台的外卖点餐系统，严格遵循软件工程的生命周期模型，完成了从问题定义、需求分析、系统设计、编码实现到测试验收的全过程。主要工作总结如下：

1. 系统分析与设计方面：运用NABCD模型明确了系统定位与用户需求；采用UML工具绘制了用例图、类图、时序图等，厘清了系统边界与内部结构；设计了清晰的分层架构（表现层、业务逻辑层、数据访问层、存储层），并运用单例模式、适配器模式等有效管理状态与数据。

2. 核心功能实现方面：成功实现了四大餐厅（中餐、意式、汉堡、东一食堂）的菜品浏览与分类筛选、完整的购物车管理（增删改查、实时计价）、订单生命周期的管理（创建、状态流转、历史查询）、跨餐厅全局搜索、用户个人信息管理及缓存清理等功能。系统采用SQLite进行菜品数据持久化，并结合JSON文件进行初始化，确保了数据的灵活性与可维护性。

3. 工程实践与质量控制方面：制定了编码规范，并在开发过程中遵循；对核心模块如购物车计算、订单生成、数据搜索等设计了详细的测试用例，进行了功能测试与边界测试，确保了系统的稳定性与可靠性。通过模块化设计，系统具备了良好的可扩展性，为后续引入网络层与服务端打下了坚实基础。

总而言之，本项目不仅交付了一个功能完整、体验良好的软件产品，更是一次对软件工程全流程的深刻实践，涵盖了需求把握、架构设计、代码实现与项目管理的综合能力锻炼。

（3）田知恒同学

1. 核心业务模块开发：独立完成了购物车模块与订单管理模块的全栈式逻辑实现。不仅实现了商品的增删改查与金额结算，还构建了完整的订单状态流转体系（创建、查看详情、历史记录查询），确保了交易流程的闭环。

2. 数据存储与持久化架构：负责应用的数据层建设。针对移动端数据易丢失的问题，对比了SQLite与Room的优劣，最终采用SQLite数据库技术实现了数据的持久化存储；同时设计了User、FoodItem、Order等核心数据模型，保证了数据结构的高效性与稳定性。

3. UI规范研究与前端实现：在项目前期深入研究了Android UI设计规范与Material Design，并将这些标准应用到前端页面的初步设计中。特别是在订单列表与详情页的UI实现上，通过统一的视觉规范，确保了界面风格的美观与交互的一致性。

4. 系统测试与质量保障：在项目后期主要负责功能测试工作，对应用进行了全流程的黑盒测试。通过模拟真实用户场景，发现并修复了多个逻辑漏洞，极大地提升了系统的健壮性。

（4）贾柠泽同学

在本次Android外卖应用项目中，我主要负责数据存储方案设计、系统架构搭建、餐厅浏览模块实现以及系统联调等关键任务。通过参与项目的全过程，我从技术调研到代码实现，积累了丰富的实践经验，现将个人工作总结如下。

1. 前期调研与系统设计

在项目初期，我重点参与了技术调研和系统设计工作。首先，我负责数据存储方案调研，通过查阅Android开发文档和行业案例，对比了SQLite、JSON文件存储和网络API等多种方案。基于项目需求（如菜品数据需本地化、支持快速搜索），我最终推荐使用SQLite数据库结合JSON初始化数据的混合方案，这体现在代码的FoodDatabaseHelper类中。其次，我参与了系统总体设计，与团队成员合作完成了需求分析、UML图绘制和系统架构设计。我梳理了模块依赖关系，如餐厅浏览模块需调用数据层（FoodDatabaseHelper）和实体类（FoodItem），购物车模块依赖ShoppingCart单例模式。通过设计清晰的接口规范，我确保了各模块解耦，提高了代码可维护性。同时，我学习了后端知识如网络请求和API设计，为

后续可能的扩展预留了空间（如订单同步功能）。这一阶段的调研和设计工作，帮助我深入理解了移动应用架构的最佳实践。

2. 项目框架搭建与可行性验证

在初步探索阶段，我负责搭建Android项目基础框架。我使用Android Studio创建了项目结构，配置了Gradle依赖，并建立了包分层（如com.example.test下按功能划分activity、adapter等）。我还编写了基础类如FoodItem，定义菜品属性（名称、价格、分类等），为数据模型统一了标准。在框架测试中，我验证了数据库操作的可行性：通过FoodDatabaseHelper的getAllFoodItems方法成功检索菜品数据，并模拟了购物车添加流程（ShoppingCart的addItem方法）。这些工作确保了核心流程（如数据加载→界面展示→用户交互）的顺畅，评估了技术方案的可靠性，为分模块开发扫清了障碍。

3. 餐厅浏览模块的实现与系统联调

本阶段是我工作的重点，我独立实现了餐厅浏览模块，包括餐厅列表、分类展示和搜索功能。首先，我开发了RestaurantListActivity，使用CardView展示四家餐厅（中华小馆、意式厨房等），并设置点击事件跳转到对应餐厅界面。通过优化UI布局（如搜索框和底部导航栏），我提升了用户交互体验。其次，我实现了多种餐厅的展示界面，如BurgerRestaurantActivity、ChineseRestaurantActivity等。这些Activity均采用RecyclerView展示菜品列表，并集成分类导航（如经典汉堡、特色汉堡）。我通过FoodItemAdapter处理菜品数据绑定，并确保与购物车的交互（如点击添加按钮调用ShoppingCart.addItem）。在搜索功能中，我设计了SearchResultsActivity 和 SearchResultAdapter ， 通 过 FoodDatabaseHelper 的searchFoodItems方法实现跨餐厅关键词搜索。例如，用户输入“汉堡”时，系统会检索所有餐厅的匹配菜品，并显示餐厅来源（如“闪电汉堡”）。这一功能增强了应用的实用性。此外，我负责系统联调，协调各模块集成：测试了从餐厅浏览到购物车结算的完整流程，修复了数据不一致问题（如菜品价格加载异常）。

4. 测试与优化支持

虽然测试阶段主要由其他成员负责，我仍参与了部分功能测试，协助周域楷进行单元测试，并针对餐厅模块进行了兼容性检查（如不同屏幕尺寸下的布局适配）。通过测试，我发现并修复了分类筛选时数据刷新的问题，确保了模块稳定性。

6 体会

（1）周域楷同学

分工：用户管理模块编写

体会：用户管理模块需实现用户注册登录、个人信息管理、修改密码等功能。在实现修改密码功能时，在 UserManager 类中先实现了基本的基本的验证旧密码后设置新密码的密码修改方法，并在该方法中使用了 UI 控制进行前端输入验证。但是这样会使得 UserManager 类中混入 UI 层的处理，变得复杂，且不一定能移植到其他环境。于是新建了 ChangePassowrdActivity 类，在这里实现了前端输入验证。然后再在 ChangePassowrdActivity 类中添加新密码不能等于旧密码、两次输入的新密码相同、新密码长度大于或等于 6 的安全性验证，并提供了相应的前端反馈。在这个过程中，我认识到将业务逻辑和 UI 控制分离可以使得代码调试中 bug 定位更容易，提高了代码的可移植性。

（2）曾程同学

分工：负责项目整体架构设计、核心业务逻辑开发（用户管理、购物车、订单模块）、数据库设计与管理，并协调完成各餐厅页面的集成与测试。

体会：

本次《软件工程》课程项目是一次从理论到实践的宝贵跃迁。我最大的体会是，一个成功的软件系统，其价值不仅在于代码的运行，更在于前期周密的规划和持续的过程管理。

首先，在架构设计阶段，我深刻理解了“高内聚、低耦合”的重要性。最初，我曾考虑为每个餐厅编写高度定制的代码，但很快意识到这将导致巨大的维护成本。通过抽象出共通的 FoodItemAdapter、IRestaurantActivity 接口以及 FoodDatabaseHelper，我们建立了一套可复用的框架，使得新增一个餐厅变得异常高效。这次经历让我明白，好的设计往往不是一开始就完美，而是在不断抽象和重构中涌现出来的。

其次，在数据管理方面，我遇到了本地状态与持久化存储协同的挑战。例如，购物车需要内存级的快速响应，而订单历史需要持久保存。通过采用 ShoppingCart 单例管理会话状态，配合 SQLite 存储持久数据的策略，我学会了根据数据特性选择合适的存储媒介。同时，为了解决 JSON 数据更新后界面不刷

新的问题，我深入研究了 Android 的数据库连接生命周期，最终通过优化 FoodDatabaseHelper 的数据加载流程解决了该问题，这让我对数据流和 UI 刷新机制有了更深刻的认识。

最后，在工程素养上，本次项目强化了我的模块化思维和调试能力。将系统划分为相对独立的模块后，不仅便于并行开发，也使得单元测试成为可能。在调试一个复杂的订单状态流转 bug 时，我学会了使用系统化的日志记录和分步验证法，而不是盲目地修改代码。这个过程虽然曲折，但极大地提升了我解决复杂问题的信心和能力。

回顾整个项目，最大的收获不是完成了一个应用，而是学会了如何像一个软件工程师一样思考：在需求不确定时如何通过原型验证，在技术选型时如何权衡利弊，在遇到瓶颈时如何拆解问题。这些经验，远比代码本身更加珍贵。

（3）田知恒

分工：购物车模块、订单管理模块、数据存储（SQLite/Room）、UI 规范研究及功能测试。

体会：

在本次 Android 外卖应用开发项目中，我承担了从底层数据存储到上层核心业务逻辑的全栈式开发任务。这段经历让我对软件工程有了深刻的理解，主要体现在以下几点：

1. 技术选型与数据持久化的攻坚

在项目初期，我深刻体会到了技术方案选型的重要性。在实现“数据存储”功能时，我最初尝试直接使用原生 SQLite，但发现处理对象与关系映射（ORM）非常繁琐且容易出错。通过调研，我引入了 Android Jetpack 组件中的 Room 数据库框架。这不仅解决了购物车数据在 APP 重启后丢失的问题，还通过 TypeConverter 成功解决了“订单”与“商品列表”的一对多存储难题。这一过程让我明白，合理使用成熟的框架能极大地提高开发效率和代码质量。

2. 设计规范对用户体验的驱动作用

作为负责“UI 设计规范研究”的成员，我认识到用户体验是产品成功的关键。在开发购物车和订单列表时，我没有仅仅满足于“功能实现”，而是严格遵循 Material Design 规范，优化了列表项的间距、卡片阴影以及点击反馈。例如，在购物车数量修改功能中，我通过局部刷新替代全局刷新，避免了界面闪烁。这让我体会到，优秀的前端开发必须具备设计思维，技术实现应当服务于极致的用

户体验。

3. 复杂业务逻辑的解耦与设计模式应用

购物车和订单模块是系统中交互最频繁、逻辑最复杂的部分。为了解决数据共享问题，我应用了单例模来管理购物车状态，成功将数据逻辑与 UI 界面分离。这种“高内聚、低耦合”的设计使得代码结构更加清晰，也便于后期的维护和扩展。

4. 团队协作与全流程视野

通过参与从需求分析、UI 设计、代码实现到最终测试的全过程，我跳出了单一的程序员视角，建立了全流程的产品视野。我意识到，一个成功的软件项目不仅需要高质量的代码，更需要团队成员之间在接口定义、进度同步和 Bug 反馈上的紧密协作。

总而言之，这次项目不仅锻炼了我的 Android 编码能力，更培养了我面对技术难题时查阅文档、独立解决问题的能力，为我今后的软件开发之路打下了坚实的基础。

（4）贾柠泽

分工：数据存储方案设计、系统框架搭建、餐厅浏览模块实现、系统联调等工作。

体会：回顾整个开发过程，我遇到了诸多技术挑战和挫折，但通过不断学习和实践，最终找到了解决方案，并从中获得了宝贵的经验。以下是我基于个人分工的总结，重点描述挫折、解决方法和体会。

1.Gradle 依赖冲突

问题：在集成 RecyclerView 和 SQLite 支持时，Gradle 同步频繁失败，报错提示版本不兼容。例如，RecyclerView 库与编译 SDK 版本冲突，导致项目无法构建。我花了大量时间排查依赖关系，一度影响开发进度。

解决方法：我学习了 Gradle 的依赖管理机制，通过分析 build.gradle 文件，统一了团队的环境配置。我使用 Android Studio 的依赖分析工具，排除冲突的传递依赖，并锁定库版本。此外，我为项目创建了标准化的模块结构，将数据模型类放在独立包中，便于团队协作。最终，框架稳定支持了后续模块开发。

2. RecyclerView 滚动卡顿

问题：在 RestaurantListActivity 中，我使用 RecyclerView 展示餐厅卡片，但初始实现中，每次滚动都会重新绑定数据，导致列表滚动时卡顿。尤其是

图片加载和点击事件处理不当，进一步降低了流畅度。

解决方法：我研究了 RecyclerView 的优化技巧。首先，我引入了 ViewHolder 模式（如 FoodItemAdapter.FoodViewHolder），避免频繁调用 findViewById。其次，我为适配器实现 updateData 方法，使用 DiffUtil 计算数据差异，仅更新变化的项目，而不是全局刷新。对于搜索功能，我在 SearchResultsActivity（文档 23）中使用了异步任务处理搜索逻辑，防止主线程阻塞。此外，我通过日志调试发现分类筛选时数据源未及时更新，便添加了 notifyDataSetChanged 调用，确保 UI 同步。

通过本项目，我不仅完成了分工任务，还克服了诸多挫折。最大的收获是问题解决能力的提升：从遇到困难时的焦虑，到冷静分析、查找资料、实验验证，最终找到方案。技术上，我掌握了 Android 数据存储、UI 优化和系统设计；软技能上，我学会了团队协作和项目管理。未来，我计划深入学习性能优化，以构建更稳健的应用。这次经历让我坚信，挫折是成长的催化剂，在软件开发中，持续学习和实践是克服挑战的关键。

附录

```
public class SearchResultsActivity extends AppCompatActivity {

    private Button backButton;
    private TextView searchKeywordText;
    private RecyclerView resultsRecyclerView;
    private LinearLayout emptyView;
    private SearchResultAdapter searchResultAdapter;
    private List<FoodItem> searchResults;
    private FoodDatabaseHelper databaseHelper;
    private ShoppingCart shoppingCart;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_search_results);

        // 初始化视图
        backButton = findViewById(R.id.backButton);
        searchKeywordText = findViewById(R.id.searchKeywordText);
        resultsRecyclerView = findViewById(R.id.resultsRecyclerView);
        emptyView = findViewById(R.id.emptyView);

        // 初始化数据库和购物车
        databaseHelper = new FoodDatabaseHelper(this);
        shoppingCart = ShoppingCart.getInstance();

        // 获取搜索关键词
        String keyword = getIntent().getStringExtra("keyword");
        if (keyword != null && !keyword.isEmpty()) {
```

```
            searchKeywordText.setText("搜索：" + keyword);
            performSearch(keyword);
        } else {
            searchKeywordText.setText("搜索结果");
            showEmptyView();
        }

        // 返回按钮
        backButton.setOnClickListener(v -> finish());

        // 设置 RecyclerView
        resultsRecyclerView.setLayoutManager(new LinearLayoutManager(this));
    }

    private void performSearch(String keyword) {
        // 在数据库中搜索
        searchResults = databaseHelper.searchFoodItems(keyword);

        if (searchResults != null && !searchResults.isEmpty()) {
            // 显示搜索结果
            searchResultAdapter = new SearchResultAdapter(searchResults, foodItem -> {
                // 添加到购物车
                shoppingCart.addItem(foodItem);
                Toast.makeText(SearchResultsActivity.this,
                        foodItem.getName() + " 已加入购物车！",
                        Toast.LENGTH_SHORT).show();
            });
            resultsRecyclerView.setAdapter(searchResultAdapter);
            resultsRecyclerView.setVisibility(View.VISIBLE);
            emptyView.setVisibility(View.GONE);
        } else {
```

```
            // 显示空状态
            showEmptyView();
        }
    }

    private void showEmptyView() {
        resultsRecyclerView.setVisibility(View.GONE);
        emptyView.setVisibility(View.VISIBLE);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (databaseHelper != null) {
            databaseHelper.close();
        }
    }
}



public class SearchResultAdapter extends
RecyclerView.Adapter<SearchResultAdapter.ViewHolder> {

    private List<FoodItem> foodItems;
    private OnItemClickListener onItemClickListener;

    public interface OnItemClickListener {
        void onItemClick(FoodItem foodItem);
    }

    public SearchResultAdapter(List<FoodItem> foodItems, OnItemClickListener listener) {
```

```
        this.foodItems = foodItems;
        this.onItemClickListener = listener;
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_search_result, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        FoodItem foodItem = foodItems.get(position);

        // 设置餐厅名称
        holder.restaurantNameText.setText(foodItem.getRestaurantName());

        // 设置菜品信息
        holder.foodNameText.setText(foodItem.getName());
        holder.categoryText.setText(foodItem.getCategory());
        holder.descriptionText.setText(foodItem.getDescription());
        holder.priceText.setText(String.format(Locale.US, "¥%.2f", foodItem.getPrice()));

        // 添加按钮点击事件
        holder.addButton.setOnClickListener(v -> {
            if (onItemClickListener != null) {
                onItemClickListener.onItemClick(foodItem);
            }
        });
```

```
    }

    @Override
    public int getItemCount() {
        return foodItems != null ? foodItems.size() : 0;
    }

    public void updateData(List<FoodItem> newFoodItems) {
        this.foodItems = newFoodItems;
        notifyDataSetChanged();
    }

    static class ViewHolder extends RecyclerView.ViewHolder {
        TextView restaurantNameText;
        TextView foodNameText;
        TextView categoryText;
        TextView descriptionText;
        TextView priceText;
        Button addButton;

        ViewHolder(View itemView) {
            super(itemView);
            restaurantNameText = itemView.findViewById(R.id.restaurantNameText);
            foodNameText = itemView.findViewById(R.id.foodNameText);
            categoryText = itemView.findViewById(R.id.categoryText);
            descriptionText = itemView.findViewById(R.id.descriptionText);
            priceText = itemView.findViewById(R.id.priceText);
            addButton = itemView.findViewById(R.id.addButton);
        }
    }
}
```

```
public class RestaurantListActivity extends AppCompatActivity {

    private EditText searchBar;
    private CardView chineseRestaurantCard;
    private CardView italianRestaurantCard;
    private CardView burgerRestaurantCard;
    private CardView dongyiRestaurantCard;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_restaurant_list);

        searchBar = findViewById(R.id.searchBar);
        chineseRestaurantCard = findViewById(R.id.chineseRestaurantCard);
        italianRestaurantCard = findViewById(R.id.italianRestaurantCard);
        burgerRestaurantCard = findViewById(R.id.burgerRestaurantCard);
        dongyiRestaurantCard = findViewById(R.id.dongyiRestaurantCard);

        // 设置搜索框监听
        setupSearchBar();

        // 设置餐厅卡片点击事件
        chineseRestaurantCard.setOnClickListener(v -> {
            Intent intent = new Intent(RestaurantListActivity.this,
ChineseRestaurantActivity.class);
            startActivity(intent);
        });
```

```
        italianRestaurantCard.setOnClickListener(v -> {
            Intent intent = new Intent(RestaurantListActivity.this,
ItalianRestaurantActivity.class);
            startActivity(intent);
        });

        burgerRestaurantCard.setOnClickListener(v -> {
            Intent intent = new Intent(RestaurantListActivity.this,
BurgerRestaurantActivity.class);
            startActivity(intent);
        }); // <-- 添加了这个 });

        dongyiRestaurantCard.setOnClickListener(v -> {
            Intent intent = new Intent(RestaurantListActivity.this,
DongyiRestaurantActivity.class);
            startActivity(intent);
        });

        // 底部导航栏
        findViewById(R.id.navHome).setOnClickListener(v -> {
            Toast.makeText(this, "首页", Toast.LENGTH_SHORT).show();
        });

        findViewById(R.id.navOrders).setOnClickListener(v -> {
            Intent intent = new Intent(RestaurantListActivity.this,
OrderHistoryActivity.class);
            startActivity(intent);
        });

        findViewById(R.id.navProfile).setOnClickListener(v -> {
            Intent intent = new Intent(RestaurantListActivity.this, ProfileActivity.class);
```

```
            startActivity(intent);
        });
    }

    private void setupSearchBar() {
        // 监听搜索框的回车键
        searchBar.setOnEditorActionListener((v, actionId, event) -> {
            if (actionId == EditorInfo.IME_ACTION_SEARCH ||
                    (event != null && event.getKeyCode() == KeyEvent.KEYCODE_ENTER &&
event.getAction() == KeyEvent.ACTION_DOWN)) {
                performSearch();
                return true;
            }
            return false;
        });

        // 也可以设置图标点击事件（如果需要）
        searchBar.setOnClickListener(v -> {
            // 当用户点击搜索框时的处理
        });
    }

    private void performSearch() {
        String keyword = searchBar.getText().toString().trim();

        if (keyword.isEmpty()) {
            Toast.makeText(this, "请输入搜索关键词", Toast.LENGTH_SHORT).show();
            return;
        }

        // 跳转到搜索结果页面
```

```
        Intent intent = new Intent(RestaurantListActivity.this,
SearchResultsActivity.class);
        intent.putExtra("keyword", keyword);
        startActivity(intent);
    }
}



public class Restaurant {
    private String name;
    private String slogan;
    private String signatureDish;
    private String backgroundColor;
    private String theme;
    private int iconResId;

    public Restaurant(String name, String slogan, String signatureDish, String
backgroundColor, String theme) {
        this.name = name;
        this.slogan = slogan;
        this.signatureDish = signatureDish;
        this.backgroundColor = backgroundColor;
        this.theme = theme;
    }

    public String getName() {
        return name;
    }

    public String getSlogan() {
        return slogan;
```

```
    }

    public String getSignatureDish() {
        return signatureDish;
    }

    public String getBackgroundColor() {
        return backgroundColor;
    }

    public String getTheme() {
        return theme;
    }
}

public class ProfileActivity extends AppCompatActivity {

    private TextView usernameText;
    private TextView userIdText;
    private TextView phoneText;
    private TextView emailText;
    private TextView cacheSize;
    private LinearLayout changePasswordLayout;
    private LinearLayout orderHistoryLayout;
    private LinearLayout clearCacheLayout;

    private UserManager userManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_profile);

    // 设置标题
    if (getSupportActionBar() != null) {
        getSupportActionBar().setTitle("我的");
    }

    userManager = UserManager.getInstance();

    initViews();
    loadUserInfo();
    setupClickListeners();
}

/**
 * 初始化视图
 */
private void initViews() {
    usernameText = findViewById(R.id.usernameText);
    userIdText = findViewById(R.id.userIdText);
    phoneText = findViewById(R.id.phoneText);
    emailText = findViewById(R.id.emailText);
    cacheSize = findViewById(R.id.cacheSize);
    changePasswordLayout = findViewById(R.id.changePasswordLayout);
    orderHistoryLayout = findViewById(R.id.orderHistoryLayout);
    clearCacheLayout = findViewById(R.id.clearCacheLayout);
}

/**
 * 加载用户信息
```

```
     */
    private void loadUserInfo() {
        User currentUser = userManager.getCurrentUser();
        if (currentUser != null) {
            usernameText.setText(currentUser.getUsername());
            userIdText.setText("ID: " + currentUser.getUserId());
            phoneText.setText(currentUser.getMaskedPhoneNumber());
            emailText.setText(currentUser.getMaskedEmail());
        }

        // 计算缓存大小
        updateCacheSize();
    }

    /**
     * 设置点击监听器
     */
    private void setupClickListeners() {
        // 修改密码
        changePasswordLayout.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(ProfileActivity.this,
ChangePasswordActivity.class);
                startActivity(intent);
            }
        });

        // 订单历史
        orderHistoryLayout.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                Intent intent = new Intent(ProfileActivity.this,
OrderHistoryActivity.class);
                startActivity(intent);
            }
        });


        // 清除缓存
        clearCacheLayout.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                showClearCacheDialog();
            }
        });

        // 退出登录
        findViewById(R.id.logoutButton).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                showLogoutDialog();
            }
        });
    }

    /**
     * 更新缓存大小显示
     */
    private void updateCacheSize() {
        try {
            File cacheDir = getCacheDir();
```

```
            long size = getDirSize(cacheDir);
            double mb = size / (1024.0 * 1024.0);
            cacheSize.setText(String.format("%.1f MB", mb));
        } catch (Exception e) {
            cacheSize.setText("0.0 MB");
        }
    }

    /**
     * 计算目录大小
     */
    private long getDirSize(File dir) {
        if (dir == null || !dir.exists()) {
            return 0;
        }

        long size = 0;
        File[] files = dir.listFiles();
        if (files != null) {
            for (File file : files) {
                if (file.isDirectory()) {
                    size += getDirSize(file);
                } else {
                    size += file.length();
                }
            }
        }
        return size;
    }

    /**
```

```
 * 显示清除缓存确认对话框
 */
private void showClearCacheDialog() {
    new AlertDialog.Builder(this)
            .setTitle("清除缓存")
            .setMessage("确定要清除应用缓存吗？")
            .setPositiveButton("确定", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    clearCache();
                }
            })
            .setNegativeButton("取消", null)
            .show();
}

/**
 * 清除缓存
 */
private void clearCache() {
    try {
        File cacheDir = getCacheDir();
        deleteDir(cacheDir);
        updateCacheSize();
        Toast.makeText(this, "缓存已清除", Toast.LENGTH_SHORT).show();
    } catch (Exception e) {
        Toast.makeText(this, "清除缓存失败", Toast.LENGTH_SHORT).show();
    }
}

/**
```

```
 * 删除目录
 */
private boolean deleteDir(File dir) {
    if (dir != null && dir.isDirectory()) {
        String[] children = dir.list();
        if (children != null) {
            for (String child : children) {
                boolean success = deleteDir(new File(dir, child));
                if (!success) {
                    return false;
                }
            }
        }
    }
    return dir != null && dir.delete();
}

/**
 * 显示退出登录确认对话框
 */
private void showLogoutDialog() {
    new AlertDialog.Builder(this)
            .setTitle("退出登录")
            .setMessage("确定要退出登录吗？")
            .setPositiveButton("确定", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    logout();
                }
            })
            .setNegativeButton("取消", null)
```

```
                .show();
    }

    /**
     * 退出登录
     */
    private void logout() {
        // 清除用户状态
        userManager.logout();

        Toast.makeText(this, "已退出登录", Toast.LENGTH_SHORT).show();

        // 返回主页
        finish();
    }

    @Override
    protected void onResume() {
        super.onResume();
        // 页面重新显示时刷新用户信息
        loadUserInfo();
    }
}



public class OrderManager {
    private static OrderManager instance;
    private List<Order> orders;

    private OrderManager() {
        this.orders = new ArrayList<>();
```

```
    // 初始化一些测试订单
    initTestOrders();
}

/**
 * 获取 OrderManager 单例
 */
public static OrderManager getInstance() {
    if (instance == null) {
        instance = new OrderManager();
    }
    return instance;
}

/**
 * 初始化测试订单
 */
private void initTestOrders() {
    // 创建一些历史订单用于测试

    // 订单 1 - 中华小馆
    Order order1 = new Order(
            "202310051230001",
            "中华小馆",
            new Date(System.currentTimeMillis() - 7 * 24 * 60 * 60 * 1000), // 7 天前
            Order.OrderStatus.COMPLETED,
            160.00
    );
    order1.addItem(new OrderItem("北京烤鸭", 128.00, 1, "经典北京烤鸭，皮脆肉嫩"));
    order1.addItem(new OrderItem("酸辣汤", 16.00, 2, "开胃酸辣汤"));
    orders.add(order1);
```

```
// 订单 2 - 意式厨房
Order order2 = new Order(
        "202310101845002",
        "意式厨房",
        new Date(System.currentTimeMillis() - 3 * 24 * 60 * 60 * 1000), // 3 天前
        Order.OrderStatus.COMPLETED,
        158.00
);
order2.addItem(new OrderItem("玛格丽特披萨", 78.00, 1, "经典玛格丽特披萨"));
order2.addItem(new OrderItem("提拉米苏", 42.00, 1, "正宗意式提拉米苏"));
order2.addItem(new OrderItem("凯撒沙拉", 38.00, 1, "新鲜凯撒沙拉"));
orders.add(order2);

// 订单 3 - 闪电汉堡
Order order3 = new Order(
        "202310151200003",
        "闪电汉堡",
        new Date(System.currentTimeMillis() - 1 * 24 * 60 * 60 * 1000), // 1 天前
        Order.OrderStatus.COMPLETED,
        89.00
);
order3.addItem(new OrderItem("经典芝士牛肉汉堡", 45.00, 1, "经典美式汉堡"));
order3.addItem(new OrderItem("薯条", 18.00, 1, "黄金薯条"));
order3.addItem(new OrderItem("可乐", 8.00, 1, "冰爽可乐"));
order3.addItem(new OrderItem("巧克力奶昔", 18.00, 1, "浓郁巧克力奶昔"));
orders.add(order3);

// 订单 4 - 中华小馆（最近的订单）
Order order4 = new Order(
        "202310161830004",
```

```
            "中华小馆",
            new Date(System.currentTimeMillis() - 5 * 60 * 60 * 1000), // 5小时前
            Order.OrderStatus.DELIVERING,
            89.00
    );
    order4.addItem(new OrderItem("宫保鸡丁", 48.00, 1, "经典川菜"));
    order4.addItem(new OrderItem("扬州炒饭", 25.00, 1, "正宗扬州炒饭"));
    order4.addItem(new OrderItem("酸梅汤", 8.00, 2, "清凉解暑"));
    orders.add(order4);
}

/**
 * 创建新订单
 */
public Order createOrder(String restaurantName, List<CartItem> cartItems) {
    // 生成订单号（时间戳 + 随机数）
    String orderId = String.format("%d%03d",
        System.currentTimeMillis() / 1000,
        (int)(Math.random() * 1000));

    // 计算总金额
    double totalAmount = 0;
    for (CartItem item : cartItems) {
        totalAmount += item.getTotalPrice();
    }

    // 创建订单
    Order order = new Order(
        orderId,
        restaurantName,
        new Date(),
```

```
            Order.OrderStatus.PAID,
            totalAmount
        );

        // 添加订单项
        for (CartItem cartItem : cartItems) {
            OrderItem orderItem = new OrderItem(cartItem);
            order.addItem(orderItem);
        }

        // 保存订单
        orders.add(order);

        return order;
    }

    /**
     * 获取所有订单（按时间倒序）
     */
    public List<Order> getAllOrders() {
        // 创建副本并排序
        List<Order> sortedOrders = new ArrayList<>(orders);
        Collections.sort(sortedOrders, new Comparator<Order>() {
            @Override
            public int compare(Order o1, Order o2) {
                // 时间倒序（最新的在前）
                return o2.getOrderTime().compareTo(o1.getOrderTime());
            }
        });
        return sortedOrders;
    }
```

```
/**
 * 根据订单 ID 获取订单
 */
public Order getOrderById(String orderId) {
    for (Order order : orders) {
        if (order.getOrderId().equals(orderId)) {
            return order;
        }
    }
    return null;
}

/**
 * 更新订单状态
 */
public void updateOrderStatus(String orderId, Order.OrderStatus status) {
    Order order = getOrderById(orderId);
    if (order != null) {
        order.setStatus(status);
    }
}

/**
 * 取消订单
 */
public boolean cancelOrder(String orderId) {
    Order order = getOrderById(orderId);
    if (order != null && order.getStatus() == Order.OrderStatus.PENDING) {
        order.setStatus(Order.OrderStatus.CANCELLED);
        return true;
```

```
        }
        return false;
    }

    /**
     * 清空所有订单
     */
    public void clearOrders() {
        orders.clear();
    }

    /**
     * 获取订单数量
     */
    public int getOrderCount() {
        return orders.size();
    }
}
```

```
public class OrderItem {
    private String foodName;        // 菜品名称
    private double price;           // 单价
    private int quantity;           // 数量
    private String description;     // 描述

    public OrderItem(String foodName, double price, int quantity, String description) {
        this.foodName = foodName;
        this.price = price;
        this.quantity = quantity;
```

```
        this.description = description;
    }

    // 从 CartItem 创建 OrderItem 的便捷构造函数
    public OrderItem(CartItem cartItem) {
        this.foodName = cartItem.getFoodItem().getName();
        this.price = cartItem.getFoodItem().getPrice();
        this.quantity = cartItem.getQuantity();
        this.description = cartItem.getFoodItem().getDescription();
    }

    // Getters
    public String getFoodName() {
        return foodName;
    }

    public double getPrice() {
        return price;
    }

    public int getQuantity() {
        return quantity;
    }

    public String getDescription() {
        return description;
    }

    // Setters
    public void setQuantity(int quantity) {
        this.quantity = quantity;
```

```
    }

    /**
     * 获取该订单项的总价
     * @return 单价 * 数量
     */
    public double getTotalPrice() {
        return price * quantity;
    }

    /**
     * 获取格式化的单价
     * @return 格式化单价字符串（例如：¥32.00）
     */
    public String getFormattedPrice() {
        return String.format(Locale.getDefault(), "¥%.2f", price);
    }

    /**
     * 获取格式化的总价
     * @return 格式化总价字符串（例如：¥64.00）
     */
    public String getFormattedTotalPrice() {
        return String.format(Locale.getDefault(), "¥%.2f", getTotalPrice());
    }
}



public class Order {
    private String orderId;              // 订单号
```

```
    private String restaurantName;       // 餐厅名称
    private Date orderTime;              // 下单时间
    private OrderStatus status;          // 订单状态
    private double totalAmount;          // 总金额
    private List<OrderItem> items;       // 订单项列表

    public enum OrderStatus {
        PENDING("待付款"),
        PAID("已付款"),
        PREPARING("制作中"),
        DELIVERING("配送中"),
        COMPLETED("已完成"),
        CANCELLED("已取消");

        private final String displayName;

        OrderStatus(String displayName) {
            this.displayName = displayName;
        }

        public String getDisplayName() {
            return displayName;
        }
    }

    public Order(String orderId, String restaurantName, Date orderTime, OrderStatus status,
double totalAmount) {
        this.orderId = orderId;
        this.restaurantName = restaurantName;
        this.orderTime = orderTime;
        this.status = status;
```

```
        this.totalAmount = totalAmount;
        this.items = new ArrayList<>();
    }

    // Getters
    public String getOrderId() {
        return orderId;
    }

    public String getRestaurantName() {
        return restaurantName;
    }

    public Date getOrderTime() {
        return orderTime;
    }

    public OrderStatus getStatus() {
        return status;
    }

    public double getTotalAmount() {
        return totalAmount;
    }

    public List<OrderItem> getItems() {
        return items;
    }

    // Setters
    public void setStatus(OrderStatus status) {
```

```
        this.status = status;
    }

    public void addItem(OrderItem item) {
        this.items.add(item);
    }

    public void setItems(List<OrderItem> items) {
        this.items = items;
    }

    /**
     * 获取格式化的下单时间
     * @return 格式化时间字符串（例如：2023-10-05 12:30）
     */
    public String getFormattedOrderTime() {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm",
Locale.getDefault());
        return sdf.format(orderTime);
    }

    /**
     * 获取格式化的总金额
     * @return 格式化金额字符串（例如：¥128.00）
     */
    public String getFormattedTotalAmount() {
        return String.format(Locale.getDefault(), "¥%.2f", totalAmount);
    }
}
```