一、实验题目

基于 Android 的外卖助手系统设计与实现

二、实验目的

1. 复习和巩固 Android 应用技术，加深对软件设计方法、软件设计技术和设计思想的理解，并能运用所学 Android 设计知识

2. 通过实际项目开发，掌握 Android Studio 开发环境的使用、项目结构设计、界面布局实现等技能。

3. 通过分离数据模型（Model）、视图（View）和控制器（Controller），实现代码的模块化和可维护性。

4. 掌握使用 OkHttp 进行网络请求和 JSON 数据解析的方法

5. 掌握购物车的数据结构设计、商品增减逻辑、价格计算等核心功能的实现。

三、总体设计

1.背景知识

本系统基于 Android 平台开发，采用 Java 语言编写。系统架构遵循 MVC 设计模式，结合 Android 四大组件（Activity、Service、BroadcastReceiver、ContentProvider）的特性进行设计。

开发工具：Android Studio

测试设备：Android 模拟器 (Medium API 36)

后端服务：本地 Tomcat 服务器 (http://10.0.2.2:8080)

主要涉及以下技术：

- Android基础组件：Activity、Intent、Adapter、Handler等
- 数据交互：OkHttp网络请求、Gson/JsonParse数据解析
- 数据传递：使用Serializable接口实现对象序列化传递
- 异步处理：Handler机制处理主线程与子线程通信
- UI设计：ListView、ViewPager等复杂布局实现

2.系统功能模块

- 用户管理模块

  主要功能：用户注册、登录、个人信息管理

  关键类：LoginActivity, RegisterActivity、UserCenterActivity

- 店铺展示模块

  主要功能：店铺列表、广告轮播

  关键类：ShopActivity, ShopAdapter，AdBannerAdapter

- 店铺详情模块

  主要功能：商品列表、店铺信息

  关键类：ShopDetailActivity、MenuAdapter

- 购物车模块

  主要功能：商品添加、数量修改、价格计算、清空购物车

  关键类：ShopDetailActivity、CarAdapter

- 订单管理模块

  主要功能：订单生成、支付模拟、历史订单查看

  关键类：OrderHistoryActivity、OrderHistoryAdapter

- 菜品详情模块

  主要功能：菜品信息展示

  关键类：FoodActivity、FoodBean

- 数据管理模块

  主要功能：本地数据存储、网络数据请求

  关键类：JsonParse, Constant, OrderHistoryManager，UserManager

3.设计步骤

- 需求分析阶段：明确系统功能需求

功能模块	子功能	用户场景
用户管理	登录/注册/修改信息	新用户注册 已有用户登录\修改信息
店铺浏览	列表/详情	用户选择店铺
购物车	添加/减少/清除	用户管理所选菜品
订单管理	生成/支付	用户下单支付
菜品展示	详情展示	用户查看菜品信息
历史订单	查看订单	用户查看消费历史

- 界面设计阶段：使用XML设计各界面布局

设计界面布局及交互，主要模块包括：登录注册页面控件、店铺列表页面广告轮播及导航栏、店铺详情页购物车、支付页面、个人中心页面、历史订单页面

- 数据模型设计阶段：设计实体类（Bean）和数据结构

  主要涉及 UserBean，ShopBean，FoodBean，OrderHistroyBean

- 业务逻辑实现阶段：编写各模块的业务逻辑代码

  主要涉及模块划分，算法，网络请求处理与事件处理机制

- 数据持久化阶段：实现本地数据存储功能

  SharedPreferences 封装，订单历史持久化

- 测试调试阶段：进行功能测试和性能优化

四、详细设计（含主要的数据结构、程序流程图、关键代码等）

1.数据结构设计

Bean 类：

- 用户实体类

用于存储用户的所有个人信息，包括基本身份信息和消费统计信息。该类实现了 Serializable 接口，通过 Intent 在 Activity 之间传递。除了基本的用户 ID、用户名、密码、手机号、收货地址等字段外，还包含了注册时间、最后登录时间、订单总数和总消费金额等扩展字段

```
public class UserBean implements Serializable {
    private static final long serialVersionUID = 1L;
    private int userId;                    // 用户 ID（自增主键）
    private String username;               // 用户名（唯一标识）
    private String password;               // 密码（明文存储，实际应加密）
    private String phone;                  // 手机号（11 位数字）
    private String address;                // 收货地址
    private Date registerTime;             // 注册时间
    private Date lastLoginTime;            // 最后登录时间
    private int orderCount = 0;            // 订单总数（统计用）
    private BigDecimal totalConsumption = BigDecimal.ZERO; // 总消费金额
    // 构造函数、Getter/Setter 方法、辅助方法
    public boolean validate() {
        return username != null && password != null &&
                phone != null && phone.matches("^1[0-9]{10}$");
    }
}
```

- 店铺实体类

封装了外卖店铺的所有信息，包括店铺的基本属性、配送信息和关联的菜品列表。每个店铺有唯一的 ID 和名称，以及月售数量、起送价格、配送费用等经营参数。特色标签、配送时间、店铺公告等信息。还包含了店铺图片和广告图的 URL 地址，以及通过列表形式存储的该店铺所有菜品信息。业务方法包括判断店铺是否营业和计算配送费用等。

```
public class ShopBean implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;                        // 店铺 ID
    private String shopName;               // 店铺名称
    private int saleNum;                   // 月售数量
    private BigDecimal offerPrice;         // 起送价格
```

```
    private BigDecimal distributionCost;// 配送费用
    private String feature;                // 特色标签
    private String time;                   // 配送时间
    private String banner;                 // 广告图 URL
    private String shopPic;                // 店铺图片 URL
    private String shopNotice;             // 店铺公告
    private List<FoodBean> foodList;       // 菜品列表
    // 业务方法
    public boolean isOpen() {
        return true;
    }
    public BigDecimal calculateDeliveryFee(BigDecimal orderAmount) {
        // 可根据订单金额计算不同配送费
        return distributionCost;
    }
}
```

- 菜品实体类

代表店铺中的单个食品项目，包含了菜品的所有展示和销售信息。包含基菜品 ID、名称、价格、图片 URL、标签和月售描述等营销信息。购物车数量字段，用于记录该菜品在购物车中的当前数量。

```
public class FoodBean implements Serializable {
    private static final long serialVersionUID = 1L;
    private int foodId;                    // 菜品 ID
    private String foodName;               // 菜品名称
    private String popularity;             // 人气标签
    private String saleNum;                // 月售描述
    private BigDecimal price;              // 单价
    private int count = 0;                 // 购物车中的数量
    private String foodPic;                // 菜品图片 URL
    //订单历史保存
    public FoodBean deepCopy() {
        FoodBean copy = new FoodBean();
        copy.setFoodId(this.foodId);
        copy.setFoodName(this.foodName);
        copy.setPrice(new BigDecimal(this.price.toString()));
        copy.setCount(this.count);
        copy.setFoodPic(this.foodPic);
        return copy;
```

```
    }
    // 计算单个菜品总价
    public BigDecimal getTotalPrice() {
        return price.multiply(BigDecimal.valueOf(count));
    }
}
```

- 订单历史实体类

用于持久化存储用户已完成订单的完整信息。每个订单有唯一的订单 ID，采用时间戳加随机数的组合方式生成，确保全局唯一性。订单信息包括关联的用户 ID、店铺名称、下单时间、总金额和订单状态等数据。

```
public class OrderHistoryBean implements Serializable {
    private static final long serialVersionUID = 1L;
    private String orderId;                 // 订单号（UUID 生成）
    private int userId;                     // 用户 ID
    private String shopName;                // 店铺名称
    private Date orderTime;                 // 下单时间
    private BigDecimal totalAmount;         // 订单总金额
    private String status = "已完成";       // 订单状态
    private List<FoodBean> foodList;        // 菜品列表（深拷贝）
    private String address;                 // 配送地址
    private BigDecimal distributionCost;// 配送费
    public String getFormattedTime() {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");
        return sdf.format(orderTime);
    }
    public int getTotalItemCount() {
        int total = 0;
        for (FoodBean food : foodList) {
            total += food.getCount();
        }
        return total;
    }
}
```

数据存储结构：

- SharedPreferences存储结构

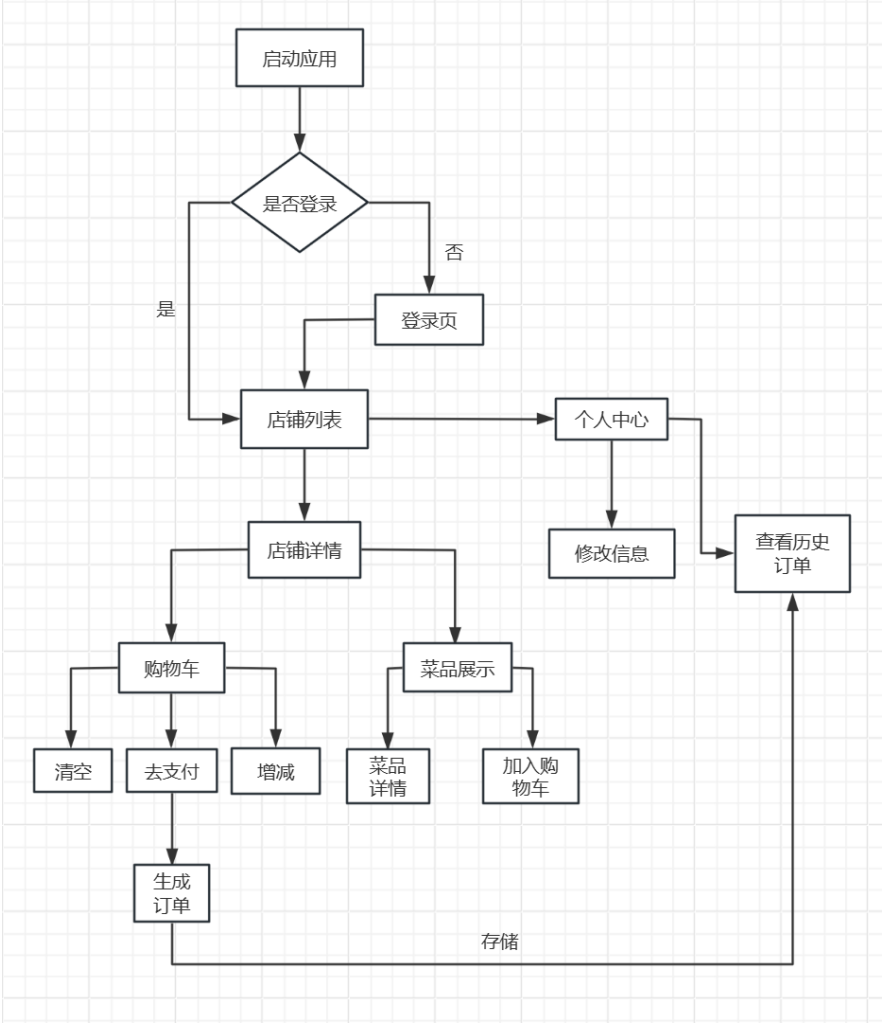
采用 Android 的 SharedPreferences 作为主要的数据持久化方案，将复杂对象通过 Gson 库序列化为 JSON 字符串进行存储。用户数据存储在 user_prefs.xml 文件中，包含两个主要键值对：users 键存储所有注册用户的 JSON 数组，current_user 键存储当前登录用户的 JSON

对象。订单历史数据存储在 order_history.xml 文件中，包含 orders 键存储所有订单的 JSON 数组

- 内存数据存储

购物车数据主要存储在内存中，通过 ShoppingCart 类进行管理。该类使用 ArrayList 存储购物车中的菜品对象，同时维护商品小计、订单总计和商品总数等统计信息。所有购物车操作都通过同步方法实现，确保在多线程环境下的数据一致性。

2.程序流程图



3.关键算法与逻辑实现

购物车管理：采用基于 HashMap 的快速查找算法，确保商品操作的效率。系统维护两个数据结构：HashMap 以菜品 ID 为键快速查找菜品对象，ArrayList 保持菜品顺序用于界面显示。当用户添加商品时，算法首先通过 HashMap 检查菜品是否已在购物车中，时间复杂度为 O(1)。

```
private Map<Integer, FoodBean> itemMap = new HashMap<>();
private List<FoodBean> itemList = new ArrayList<>();
public void addItem(FoodBean food) {
    int foodId = food.getFoodId();
```

```
            if (itemMap.containsKey(foodId)) {
                // 商品已存在，更新数量
                FoodBean existing = itemMap.get(foodId);
                existing.setCount(existing.getCount() + food.getCount());
            } else {
                // 商品不存在，添加新商品
                FoodBean copy = food.deepCopy();
                itemMap.put(foodId, copy);
                itemList.add(copy);
            }
            recalculateTotals();
        }
        public void removeItem(int foodId) {
            if (itemMap.containsKey(foodId)) {
                FoodBean item = itemMap.get(foodId);
                itemList.remove(item);
                itemMap.remove(foodId);
                recalculateTotals();
            }
        }
        private void recalculateTotals() {
            BigDecimal total = BigDecimal.ZERO;
            int count = 0;
    for (FoodBean item : itemList) {
                BigDecimal itemTotal = item.getPrice()
                        .multiply(BigDecimal.valueOf(item.getCount()));
                total = total.add(itemTotal);
                count += item.getCount();
            }
            // 更新统计信息
            this.totalMoney = total;
            this.totalCount = count;
        }
    }
```

订单 ID 生成：采用类雪花算法的简化实现，保证 ID 的全局唯一性和时间有序性。订单 ID 格式为"ORD"前缀加 13 位时间戳加 4 位随机数加 4 位序列号。当在同一毫秒内生成多个订单 ID 时，序列号递增以确保唯一性。

```
    public class OrderIdGenerator {
        private static final String PREFIX = "ORD";
```

```
    private static long lastTimestamp = 0;
    private static int sequence = 0;
    public static synchronized String generateOrderId() {
        long currentTimestamp = System.currentTimeMillis();
        // 如果同一毫秒内生成多个 ID，增加序列号
        if (currentTimestamp == lastTimestamp) {
            sequence++;
        } else {
            sequence = 0;
            lastTimestamp = currentTimestamp;
        }
        int random = (int)(Math.random() * 10000);
        // 组合成订单 ID
        return String.format("%s%d%04d%04d",
            PREFIX,                 // 前缀
            currentTimestamp, // 时间戳
            random,                  // 随机数
            sequence);              // 序列号
    }
    public static long parseTimestamp(String orderId) {
        if (orderId == null || !orderId.startsWith(PREFIX)) {
            return 0;
        }
        try {
            // 提取时间戳部分
            String timestampStr = orderId.substring(PREFIX.length(), PREFIX.length() +
13);
            return Long.parseLong(timestampStr);
        } catch (Exception e) {
            return 0;
        }
    }
}
```

五、实验结果与分析

1.环境与配置

开发工具：Android Studio 2023.1.1

SDK：API 36 (Android 16)

虚拟机：MediumPhone API 36 模拟器

2. 实验结果

登录注册页面

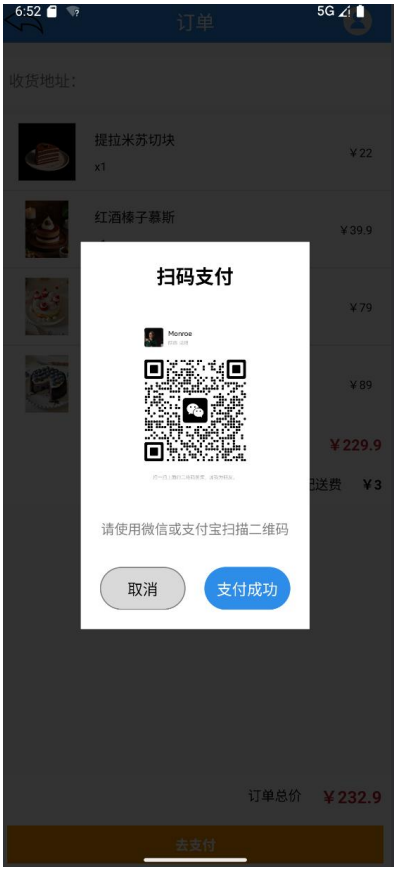
店铺列表

店铺详情页

购物车列表

结算页面

支付页面



管理页面

历史订单

六、小结与心得体会

1. 项目中遇到的问题

- Android 版本迁移问题

项目使用 AndroidX，布局中不能使用 旧版 support 包，应该将 support 全改为 AndroidX

- 使用Androidstudio内置虚拟机，应该将内网接口改为虚拟机地址 http://10.0.2.2:8080/order
- 运行后不显示店铺页面，只显示“hello android“是因为默认启动MainActivity，需要将启动项修改为shop activity
- ShopActivity 使用了 AppCompatActivity，但没有为其设置兼容的 Theme.AppCompat 主题

2. 心得体会

通过这个完整的项目实践，我最大的收获不仅仅是技术能力的提升，更是软件开发思维方式的转变：

从“能运行”到“好用的”思维转变：最初我只关心功能是否能实现，但随着项目的深入，我开始思考用户体验、代码质量、可维护性等问题。比如支付流程的改进，就是这种思维转变的直接体现。

解决问题的能力比技术本身更重要：在这次开发中，我遇到的每一个问题都是学习的机会。从最初的遇到问题就搜索答案，到现在能够分析问题本质、理解系统原理、提出解决方案，这种能力的提升远比掌握某个具体技术更有价值。

细节决定成败：Android开发中有很多细节决定成败的情况。一个Activity未注册、一个类型转换错误、一个线程问题，都可能导致应用崩溃。这让我养成了细心、严谨的开发习惯。