[项目LOGO]



[项目名称]

**[基于wifi定位的商城导购系统]**

软件设计说明书

**Version [Number]**

[2024.07.03]

**Written by [Yu.]**

[Team LOGO]



**All Rights Reserved**

目录

[1 摘要 1](#_Toc45225790)

[2 引言 1](#_Toc45225791)

[2.1 项目研究背景及意义（应用场景、目标人群） 1](#_Toc45225792)

[2.2 相关产品及对比 1](#_Toc45225793)

[2.3 项目价值及创新 1](#_Toc45225794)

[3 开发计划 1](#_Toc45225795)

[3.1 主要功能描述 1](#_Toc45225796)

[3.2 关键难点问题 1](#_Toc45225797)

[3.3 进度安排 1](#_Toc45225798)

[4 可行性分析 1](#_Toc45225799)

[4.1 市场可行性分析 1](#_Toc45225800)

[4.2 技术可行性分析 1](#_Toc45225801)

[5 需求分析 1](#_Toc45225802)

[5.1 数据采集及描述 1](#_Toc45225803)

[5.2 功能性需求 1](#_Toc45225804)

[5.2.1 功能模块结构图 2](#_Toc45225805)

[5.2.2 核心功能模块描述 2](#_Toc45225806)

[5.3 非功能性需求（精度、性能、易用性、可扩展性、可维护性等） 2](#_Toc45225807)

[5.4 用例图 2](#_Toc45225808)

[6 概要设计 2](#_Toc45225809)

[6.1 系统架构设计 2](#_Toc45225810)

[6.2 系统总体功能结构设计 2](#_Toc45225811)

[6.3 关键技术设计 2](#_Toc45225812)

[6.4 数据库设计 2](#_Toc45225813)

[6.5 错误/异常处理设计 2](#_Toc45225814)

[7 详细设计与实现 2](#_Toc45225815)

[7.1 xx子系统的设计与实现 2](#_Toc45225816)

[7.1.1 类图、流程图、时序图等 2](#_Toc45225817)

[7.1.2 用户界面展示 2](#_Toc45225818)

[7.2 核心算法（伪代码） 2](#_Toc45225819)

[7.3 接口设计 2](#_Toc45225820)

任务记录

|  |  |  |  |
| --- | --- | --- | --- |
| **名字** | **完成任务** | **占比（总和100%）** | **备 注** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 摘要

本项目旨在开发一款基于安卓平台的商场导购应用，利用 WiFi 定位技术为商场内的顾客提供精准的室内定位和导航服务等功能，提升购物体验。本文档详细介绍了项目的背景、开发计划、可行性分析以及需求分析。

# 引言

## 项目研究背景及意义（应用场景、目标人群）

**应用场景**：商场导购应用可用于大型购物中心，帮助顾客快速找到目标店铺、提前了解店铺信息等，同时考虑到用户体验我们加入了聊天等功能。  
**目标人群**：购物中心的顾客、商场管理人员。  
意义：通过提供高效、精准的室内导航服务，提升顾客购物体验，增加商场顾客满意度和回头率。其次，提供对应商铺的核心介绍，帮助顾客快速了解到商铺特点，节约购买时间，提高购买效率。最后通过聊天系统，方便用户进行聊天。

## 相关产品及对比

· **传统商场地图**：需要顾客自行寻找，不够直观和实时。本产品无需顾客了解整个商城布局，只需要通过检索功能就可实现定位前往。

· **蓝牙定位系统**：需要额外硬件支持，部署成本较高，本产品无需这些硬件，同时商场内广大的wifi布局使产品的得以顺利使用

· **GPS 定位**：适用的场景为室外，无法在室内有效工作，本产品服务于室内的应用场景，提升用户体验，促进消费。

· **现有室内导航应用**：部分精度不够，使用体验不佳，本产品旨在兼备用户体验和导航精度，力求做出一款实用大众满意的安卓app。

## 项目价值及创新

### 价值

**1. 提升用户体验：**

**精准定位**：本项目提供高精度的室内定位服务，帮助顾客轻松找到目标店铺等，提升用户购物体验。

**实时导航**：通过实时路径规划和导航，帮助顾客高效地完成购物，提高顾客满意度和购物体验。

**个性化服务（后期考虑）**：根据顾客的位置和历史行为数据，推送个性化的优惠信息和推荐，增加购物的乐趣和互动性。

1. **优化商场管理效率：**

面向商城管理员，当管理员登录系统能够进行：

**顾客流量分析**：通过分析顾客的移动轨迹和停留时间，帮助商场管理者了解顾客的行为习惯和热点区域，优化店铺布局和商品陈列。

**紧急情况响应**：在紧急情况下，如火灾或其他安全事件，能够迅速定位顾客位置，并提供最佳疏散路径，提高安全管理水平。

**资源优化配置**：通过数据分析，合理配置商场内的资源，如保安人员、清洁人员等，提高管理效率和服务质量。

3. **市场竞争优势：**

**差异化服务：**与传统的商场导航方式相比，提供更加智能和互动的服务，增强商场的竞争力和品牌形象。

**吸引新顾客：**通过创新的导购服务，吸引更多的新顾客，并提高现有顾客的回头率和消费频次。

### 创新

**1. 结合机器学习算法优化定位精度：**

**信号处理优化**：通过机器学习算法对WiFi信号进行处理和校正，减少信号干扰和噪声，提高定位精度。这里我们考虑使用knn，在后期会考虑高级算法如slam，粒子滤波器等。

**多源数据融合：**结合其他传感器数据（如经纬度陀螺仪等），通过数据融合算法进一步提升定位的准确性和稳定性。

**2. 减少硬件部署成本：**

利用现有WiFi基础设施：无需额外部署专用硬件设备，充分利用商场现有的WiFi基础设施，降低部署和维护成本。

**3.高效的实时定位服务：**

**实时位置更新：**通过定位算法和数据处理流程，实现高效的实时定位更新，保证导航服务的即时性和准确性。

**动态路径规划：**基于实时位置和环境数据，动态调整导航路径，提供最优路线。

**4. 智能推荐和互动：**

**个性化优惠推送：**通过分析顾客的购物习惯和当前位置，智能推送个性化的优惠信息和推荐，提高顾客的购物体验和参与度。

**社交互动功能：**增加顾客之间的位置共享和互动功能，提升用户粘性和社区氛围。

**5. 扩展功能（后期）：**

**语音导航：**增加语音导航功能，为顾客提供更加便捷和人性化的导购服务。

**虚拟导购：**结合AR技术，提供虚拟导购服务，增强用户体验和互动性。

# 开发计划

## 主要功能描述

· **位置计算**：通过 WiFi 信号强度和其他经纬度，海拔等相关指标，使用三角测量算法计算顾客位置。

· **店铺导航**：提供从当前位置到目标店铺的路径导航。

· **信息推送**：根据顾客当前位置推送附近店铺的优惠信息，商品信息。

·  **用户体验**：顾客可以对导航服务进行反馈，有聊天等功能。

## 关键难点问题

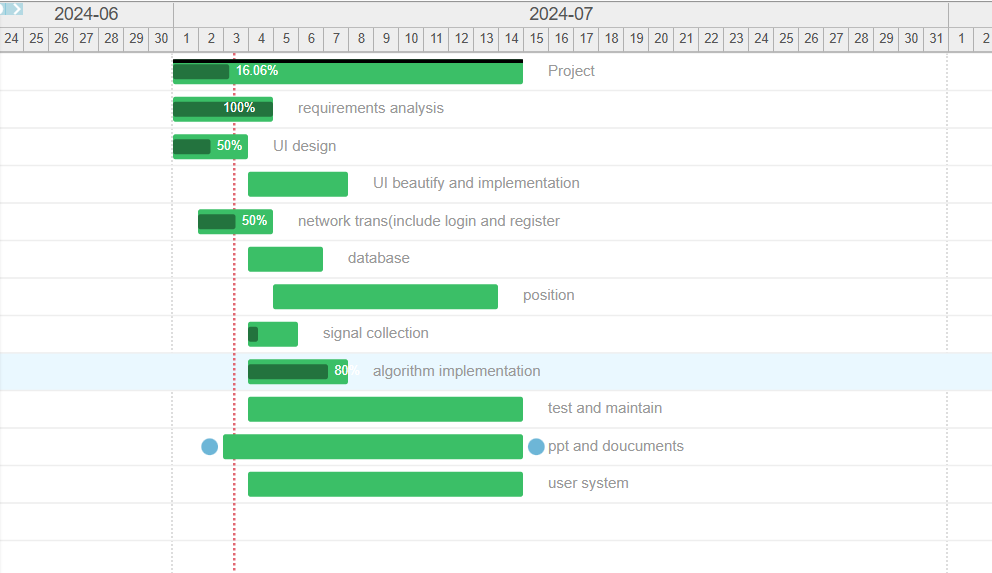
·  ****信**号干扰：**WiFi 信号在商场环境中易受干扰，影响定位精度，是否需要考虑相关因素，对于有些距离等因素无法获取的wifi信号要如何去处理。

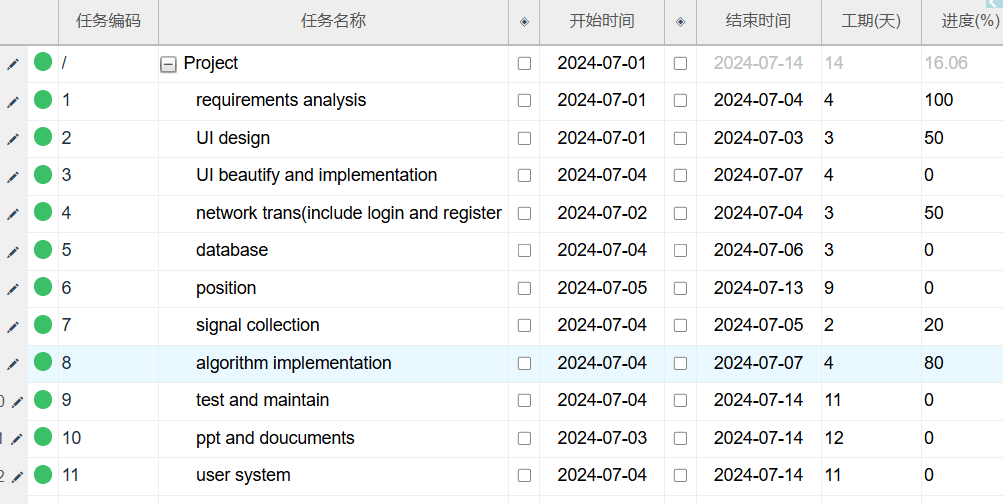
· **信号精度**：不同手机，不同时间段等因素都会影响定位的精度，如何考虑和处理这些因素，加入其他传感器信息都是需要考虑的。

· **多线程**：分配主线程，子线程等，确保多用户和安卓应用等不会卡死机等

· **实时性要求：**定位系统需要在较短时间内计算出准确位置。

## 进度安排





# 可行性分析

## 市场可行性分析

**市场需求**：

1. **大型商场数量增加**：随着城市化进程的推进，大型购物中心和商业综合体的数量不断增加。这些大型商场面积广阔、店铺众多，顾客在购物过程中经常遇到找到需求商店困难的问题
2. **室内导航需求增长**：传统的纸质地图和指示牌已经无法满足现代顾客对便捷、高效的室内导航需求。顾客希望能够通过手机应用快速、精准地找到目标位置，提高购物效率和体验。
3. **智能化和个性化服务**：现代消费者越来越倾向于个性化和智能化的购物体验。通过室内导航应用，不仅可以提供路径指引，还能根据顾客的实时位置推送个性化的优惠信息和推荐，增加购物的趣味性和便利性。

**竞争分析**：

1. **现有解决方案**：目前市场上已有一些室内导航应用，但大多数体验不佳，主要问题包括定位不准确、导航路径规划不合理、界面复杂操作不便等。
2. **改进空间**：现有应用在技术实现和用户体验上有较大的改进空间。本项目通过结合先进的 WiFi 定位技术和机器学习算法，可以显著提升定位精度和导航体验，具备明显的竞争优势。

## 技术可行性分析

**WiFi 技术成熟度**：

1. **广泛应用**：WiFi 技术已经在全球范围内得到广泛应用，几乎所有的智能手机和商场内的无线路由器都支持 WiFi 技术。这为本项目的实现提供了坚实的基础。
2. **设备普及**：商场内普遍部署有WiFi网络，顾客也习惯于使用手机连接WiFi网络进行上网和购物。这使得利用WiFi进行室内定位成为一种低成本且高效的解决方案。

**算法支持**：

1. **多种定位算法**：现有多种算法支持 WiFi 定位，包括三角测量、指纹定位等。通过结合机器学习算法，可以进一步提升定位精度和稳定性。
2. **数据融合**：除了 WiFi 信号外，还可以结合其他传感器数据（如经纬度等），通过数据融合算法提高定位准确性，尤其在复杂的商场环境中表现更加出色。

**开发工具和技术**：

**Android Studio**：

* 1. **开发环境**：Android Studio 是谷歌推荐的 Android 应用开发集成环境（IDE），提供了全面的开发工具和功能，支持高效的应用开发、测试和调试。
  2. **丰富的库和框架**：Android Studio 支持多种库和框架，可以方便地集成 WiFi 定位、地图展示、导航路径规划等功能。
  3. **广泛的社区支持**：Android Studio 拥有庞大的开发者社区，丰富的文档和教程资源，可以帮助开发团队快速解决开发过程中遇到的问题。

**Flask**：

* 1. **轻量级框架**：Flask 是一个轻量级的 Python Web 框架，适合快速开发和部署后端服务。它灵活性高，易于扩展，能够很好地支持本项目的需求。
  2. **API 服务**：通过 Flask 可以开发和部署 RESTful API，为 Android 应用提供后端支持。包括用户认证、数据存储、定位数据处理等服务。
  3. **数据处理和机器学习**：Flask 可以与各种 Python 库结合，进行数据处理和机器学习模型的训练和预测，提升定位精度和服务质量。

**Python**：

1. **数据处理和分析**：Python 拥有丰富的数据处理和分析库，可以高效地处理 WiFi 信号数据，进行统计分析和数据预处理。

**2 机器学习支持：**Python 是机器学习和人工智能领域的主流编程语言，拥有强大的机器学习库。通过这些库，可以构建和训练高效的定位算法模型，提升定位精度和性能。

1. **数据可视化：**Python 还拥有强大的数据可视化工具，，可以用于分析和展示 WiFi 信号数据和定位结果，帮助开发者理解数据特征和算法效果,这有利于帮助评判我们算法实现结果的精度等。

# 需求分析

## 数据采集及描述

### 数据采集

数据采集是WiFi定位系统的基础工作，通过采集商场内各个位置的WiFi信号强度数据，建立指纹数据库，为后续的定位算法提供基础数据支持。以下是数据采集的主要步骤和方法：

**确定采集点**：

* 1. 在商场内选定若干个采集点，这些采集点应均匀分布在整个商场内，涵盖所有可能的用户活动区域。
  2. 在每个采集点上标记具体位置（如使用商场的平面图或直接标注在地面上），以便后续采集数据和建立指纹数据库。

**WiFi 信号强度采集**：

* 1. 在每个采集点使用智能手机或其他设备，扫描周围的WiFi信号，记录每个接入点（AP）的信号强度（RSSI）。
  2. 为提高数据准确性，应在不同时间、不同角度进行多次采集，取平均值或进行数据融合，减少环境噪声和干扰的影响。
  3. 每个采集点的WiFi信号数据应包含接入点的MAC地址、信号强度、采集时间等信息。

**建立指纹数据库**：

* 1. 将每个采集点的WiFi信号强度数据与该点的实际位置（如坐标）关联，建立指纹数据库。

**其他传感器信息：**

1. **考虑经纬度，气压海拔等相关信息，加入数据当中**

### 数据描述

数据描述部分主要包括采集到的WiFi信号数据的结构和特征，为后续的算法设计和实现提供依据。

**数据结构**：

* 1. **位置数据**：每个采集点的实际位置，通常用二维坐标（x, y）表示，以适应多层商场或复杂环境。若后期加入楼层，考虑加入（x,y,z）
  2. **WiFi信号数据**：每个接入点的MAC地址和对应的信号强度（RSSI），信号强度通常为负值（dBm），表示信号的强弱。
  3. **加入其他的传感器数据信息。**

**数据特征**：

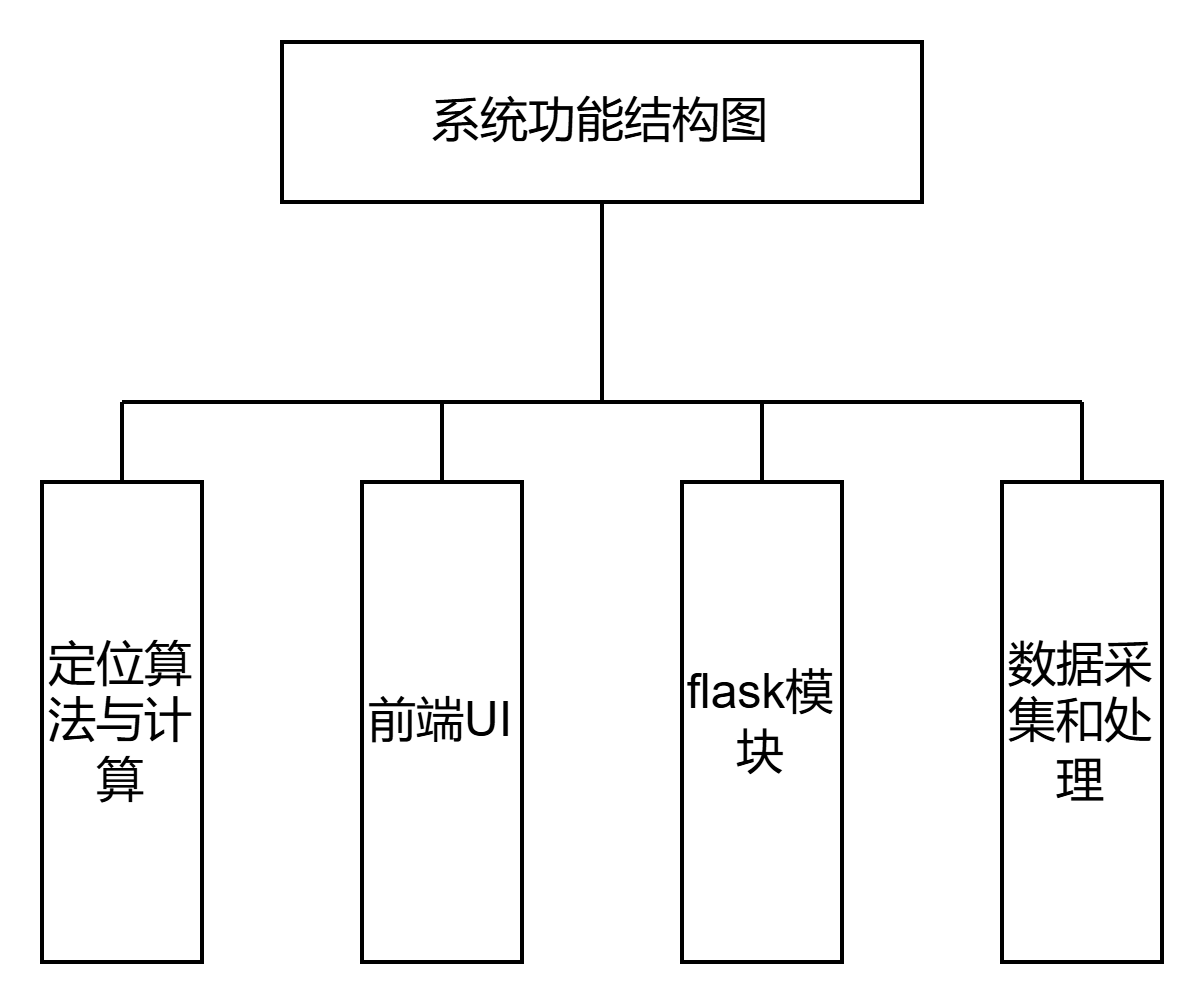
* 1. **信号强度（RSSI）**：表示接入点发出的WiFi信号到达设备时的强度，通常范围在-30 dBm 到 -100 dBm 之间。信号强度越高，值越接近于0。对于无法接受到的信息，以-256处理。
  2. **信号波动**：由于环境干扰、障碍物、设备差异等因素，WiFi信号强度会有一定的波动。通过多次采集和数据融合，可以减少这些波动对定位精度的影响。
  3. **接入点（AP）数量**：每个采集点能检测到的WiFi接入点数量不同，一般来说，接入点数量越多，定位精度越高。

**数据预处理**：

* 1. **去噪处理**：对采集到的WiFi信号数据进行滤波处理，去除异常值和噪声，提高数据质量。
  2. **标准化处理**：对信号强度数据进行标准化处理，确保数据在同一尺度上，便于后续算法处理。
  3. **缺失值处理**：处理采集过程中可能出现的缺失值，如某个接入点在某些采集点不可见，通过插值或其他方法进行补全。

## 功能性需求

### 功能模块结构图



### 核心功能模块描述

· **数据采集与处理模块**：

**WiFi 信号强度采集**：通过 Android 设备的 WiFiManager 类，实时采集用户周围的 WiFi 信号强度 (RSSI) 数据。

**数据预处理**：对采集到的信号数据进行预处理，如滤波、去噪和标准化处理，保证数据质量。

· **定位算法与计算模块**：

**指纹定位**：通过预先采集的大量 WiFi 信号强度数据，建立指纹数据库。利用机器学习算法（如 KNN），将实时采集的 WiFi 信号与指纹数据库进行匹配，计算用户当前位置。

**数据融合**：结合其他传感器数据，通过数据融合算法提高定位精度。

· **flask模块**：

**服务接口**：提供 RESTful API 接口，接收客户端上传的 WiFi 信号数据，返回计算出的用户位置。

**数据存储（考虑）**：将用户的历史定位数据和相关信息存储在数据库中，以供后续分析和优化。

· UI**模块**：

**地图显示**：在 Android 应用中，显示商场地图和用户实时位置。

**路径规划**：根据用户的当前位置和目标位置，提供最优路径规划和导航服务。

**信息推送**：根据用户位置，推送商场内的促销信息和推荐商品，提高用户体验。

## 非功能性需求（精度、性能、易用性、可扩展性、可维护性等）

· **精度**：

**定位精度**：系统应确保定位误差在 2米以内，保证用户在商场内能够准确找到目标位置。

**数据处理精度**：通过高精度的算法和数据处理方法，提高定位计算的准确性。

· **性能**：

**响应时间**：系统应在接收到 WiFi 信号数据后，能够在指定时间内返回定位结果，保证用户体验的实时性。

**处理能力**：系统应能够处理高并发请求，支持大量用户同时使用定位服务。

· **易用性**：

**用户界面友好**：前端应用应提供简洁、直观的用户界面，方便用户操作和查看定位结果。

**交互流畅**：应用应具备流畅的交互体验，保证用户在使用过程中的顺畅操作。

· **可扩展性**：

**功能扩展**：系统应具备良好的可扩展性，便于后续增加新的功能模块，如增加新的定位技术或服务。

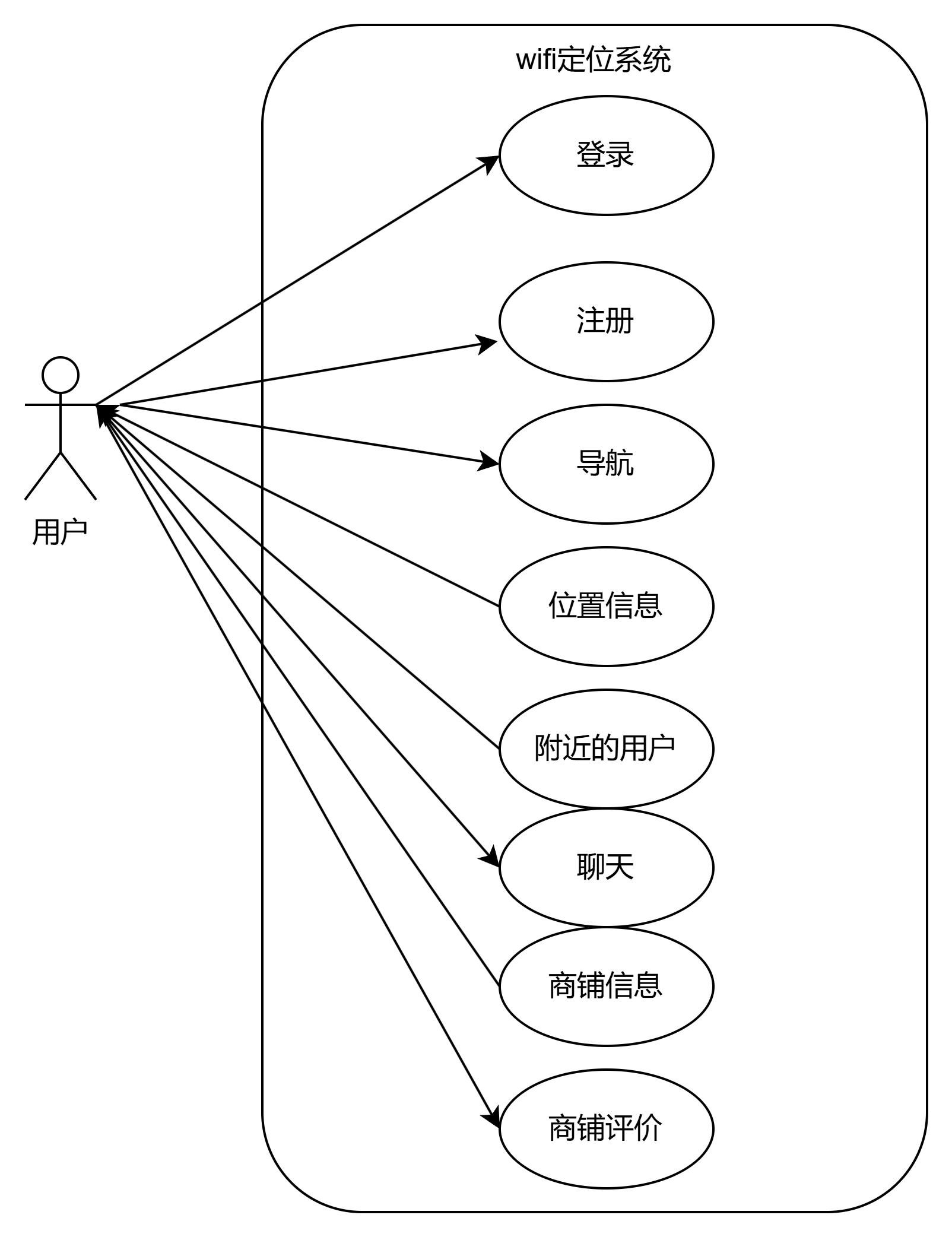
**负载扩展**：系统应具备良好的负载扩展能力，支持通过增加服务器和优化架构提升处理能力。

· **可维护性**：

**代码质量**：系统应遵循良好的编码规范和设计模式，保证代码的可读性和可维护性。

**日志记录**：系统应具备完善的日志记录功能，便于监控和排查问题，提升维护效率。

## 用例图



# 概要设计

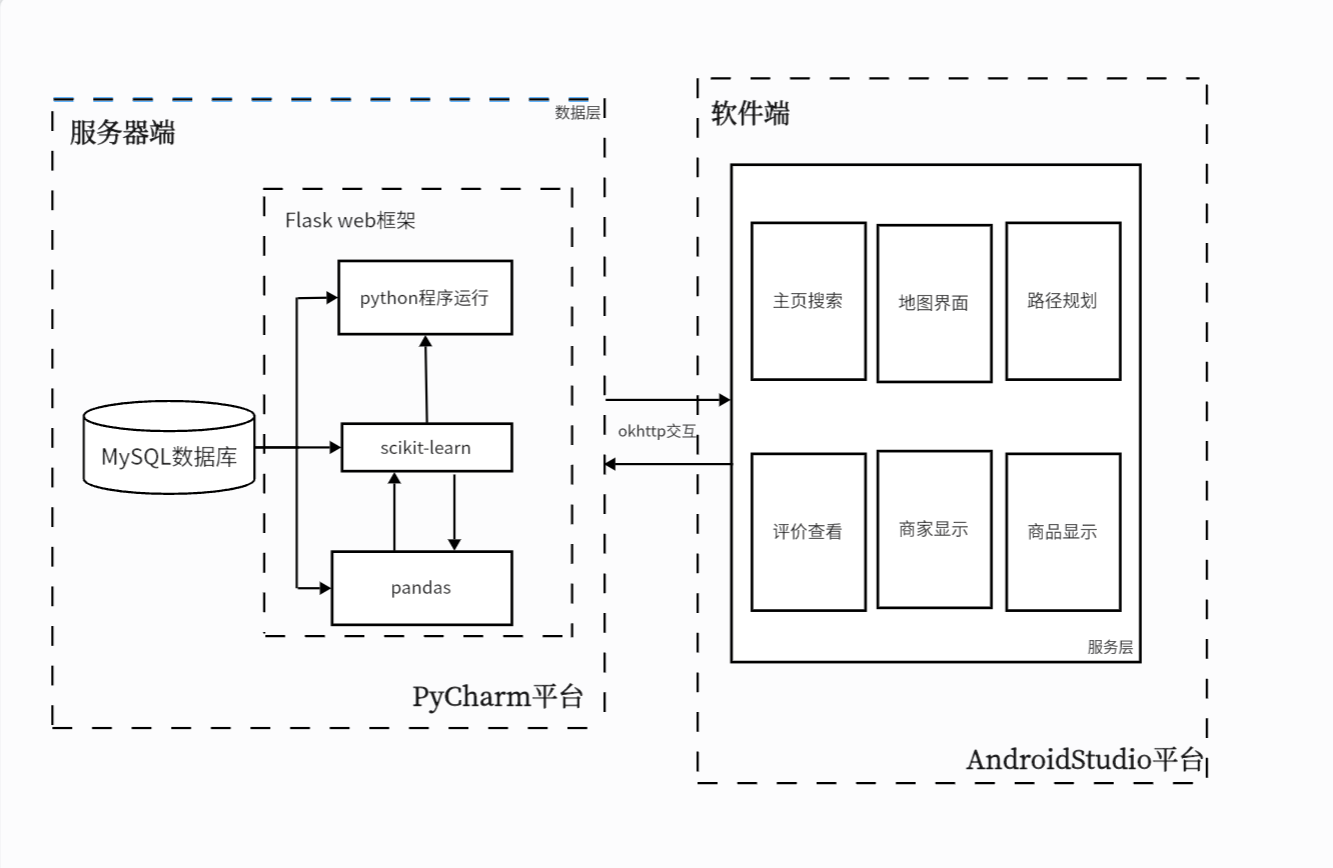
## 系统架构设计

**数据层**：该层主要负责数据的存储和管理，核心组件是“MySQL数据库”。MySQL数据库以其稳定性和高效性，为系统提供数据存储和查询的能力，确保数据的安全性和一致性。

**服务器端**：服务器端是系统的核心处理部分，包含多个关键组件。其中，“Flask web框架”用于构建网站的Web应用程序，提供Web服务的核心功能。同时，“scikit-learn”可能用于进行机器学习或数据分析任务，为系统提供智能决策和数据洞察的能力。此外，“pandas”作为Python的数据处理库，在服务器端进行数据预处理和分析工作。而“okhttp交互”则可能是一个HTTP客户端库，用于处理与客户端之间的网络通信。从整体架构来看，服务器端各组件共同构成了服务层，为客户端提供稳定可靠的服务。

**软件端**：软件端是用户直接交互的界面层，包含了多个功能模块，如“主页搜索”、“路径规划”、“地图界面”、“商品显示”等。这些模块通过调用服务器端的接口获取数据，并将处理结果展示给用户

**客户端：**客户端是用户接入系统的入口，支持多种开发平台和工具。“PyCharm平台”支持使用Python语言进行开发和调试，为我们开发提供了极大的便利。同时，“AndroidStudio平台”支持Android应用的开发，使得系统能够跨平台运行，满足不同用户的需求。



## 系统总体功能结构设计

**定位功能模块**：

**用户个性化服务**：此功能可能用户的历史行为、偏好等信息，提供个性化的定位服务，如推荐附近的商店、景点等。

**数据收集与处理**：该子功能负责收集用户的定位数据，并进行处理和分析，以便为后续的个性化服务提供数据支持。

**算法设计与实现**：在定位功能模块中，可能涉及到复杂的算法设计，如路径规划、位置预测等，这些算法的实现对于提高定位服务的准确性和效率至关重要。

**评论模块**：

**评价展示**：此功能用于展示用户对商店、商品或服务的评价信息，帮助其他用户做出更明智的选择。

**评论功能**：提供用户发表评论的接口，允许用户分享自己的购物体验、使用感受等。

**信息同步**：可能涉及将评论信息同步到其他平台或数据库，以确保信息的一致性和可访问性。

**商店信息模块**：

**商家信息展示**：展示商店的基本信息，如名称、地址、联系方式等，方便用户了解商店的详细情况。

**商品信息展示**：展示商店内商品的信息，包括价格、规格、图片等，帮助用户做出购买决策。

**前瞻信息展示：**可能包括商店的促销活动、新品发布等前瞻信息，以吸引用户的关注和购买欲望。



## 关键技术设计

### 前端RecycleView

· **RecyclerView 和 Adapter**：

使用 RecyclerView 来展示静态和动态数据列表。

自定义 StaticRvAdapter 和 DynamicRVAdapter 分别用于静态和动态数据的适配和展示。

在 StaticRvAdapter 中，通过 ViewHolder 来绑定静态数据项的布局和数据。

在 DynamicRVAdapter 中，通过 ViewHolder 来绑定动态数据项的布局和数据。

· **ViewHolder 模式**：

使用 ViewHolder 模式来提高 RecyclerView 的性能，通过重用视图来减少不必要的视图创建和绑定。

· **数据回收与复用**：

在适配器中正确实现 onBindViewHolder 方法来绑定数据。

onViewRecycled 方法中进行必要的清理操作

· **数据模型类**：

StaticRvModel 和 DynamicRVModel 类分别表示静态和动态数据的模型。

每个模型类包含必要的字段（例如图片资源、文本内容等）和方法（如获取数据的方法）。

· **布局文件**：

定义 activity\_shop2.xml 作为 ShopActivity 的布局文件，包括 RecyclerView、BottomNavigationView 和悬浮按钮等控件。

· **交互和事件处理**：

底部导航栏的点击事件处理，通过 Intent 启动不同的活动。

悬浮按钮点击事件处理，弹出评分输入对话框。

### Http协议网络通信实现：

采用了okhttp库实现客户端http请求的发送,okhttp首先创建了一个用于发送请求的客户端(OkHttpClient)。为了实现多用户功能，我们需要在服务器建立session缓存用于存储用户的登录信息，这样当每次发送请求时，服务器根据请求头的Cookie内容匹配到具体的session。okhttp提供了用于管理Cookie的接口(CookieJar)，只要在OkHttpClient实例化时设置CookieJar即可实现对Cookie的管理。我们实现了接口，使cookie可以长久化的存储在用户端，使多账户功能可以实现。

然后是请求的定义，我们的请求主要为GET请求和POST请求，创建请求体需要设置请求访问的目标URL,如果时POST请求还需要定义好请求体RequestBody。

当这些准备工作完成时，我们就可以开始发送请求了，发送请求需要根据Request从OkHttpClient获取Call对象，然后可以根据需要选择同步或异步的发送请求，同步则直接调用Call对象的execute()方法，他会阻塞线程直到服务端做出回应。如果想要避免线程阻塞可以采用异步方法，这个方法会将Call加入一个等待队列，并附加上对应的回调函数使得请求的发送与接受可以异步的发生。

在服务端我们使用了flask后端框架，它可以很简单的根据路由信息调用对应的接口，处理信息并返回结果。

### 定位算法实现

#### 数据处理部分

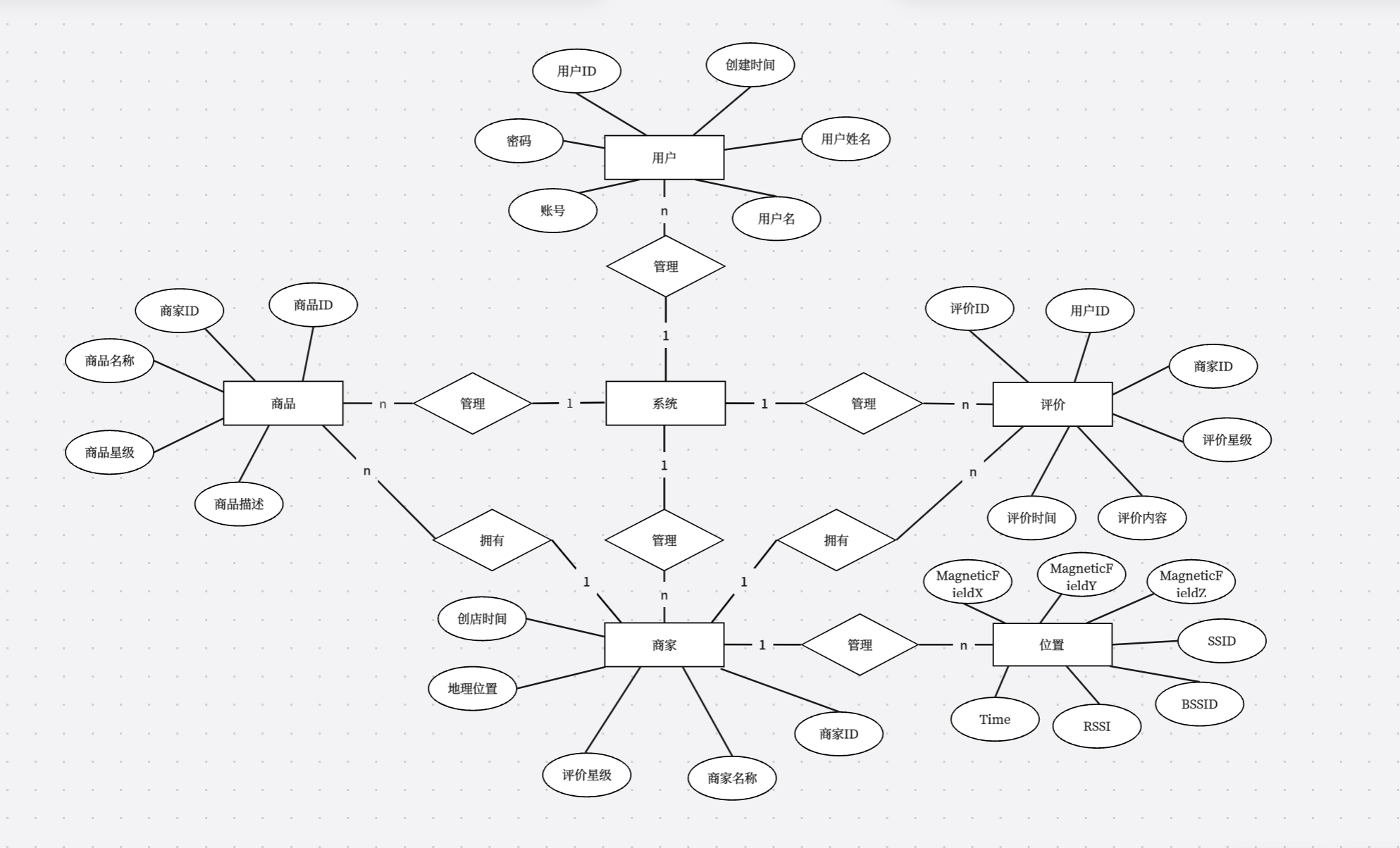
当服务器接收到用户端发来的/locator请求时，读取数据库中数据，使用pandas库中的DataFram方法转换为格式化数据并过滤异常数据，然后使用groupby方法按照Time和Position分组，对分组后的每一组数据内部，将每条数据BSSID对应的RSSI值转换为字典，存储在direction\_vectors中，键时（Time,Position)元组，值是该组的RSSi向量，之后将app传来的数据转换为Pandas Series对象。

#### 计算部分

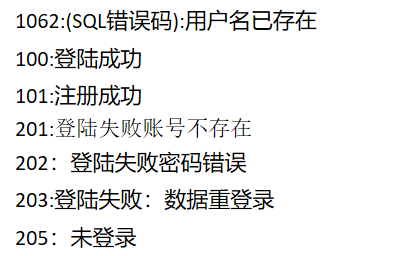
初始化空字典'similarities，遍历每个组，使用align方法对齐组内数据和客户端数据，将缺失值补0处理，计算组内数据和客户端数据两组向量之间的余弦相似度，储存在'similarities'中，键是（Time,Position)元组，值是相似度值

接下来使用knn思想，选取相似度最高的前三个组，统计这三组中的Position出现次数，将出现次数最高的Position作为判断的位置，返回给客户端。

## 数据库设计



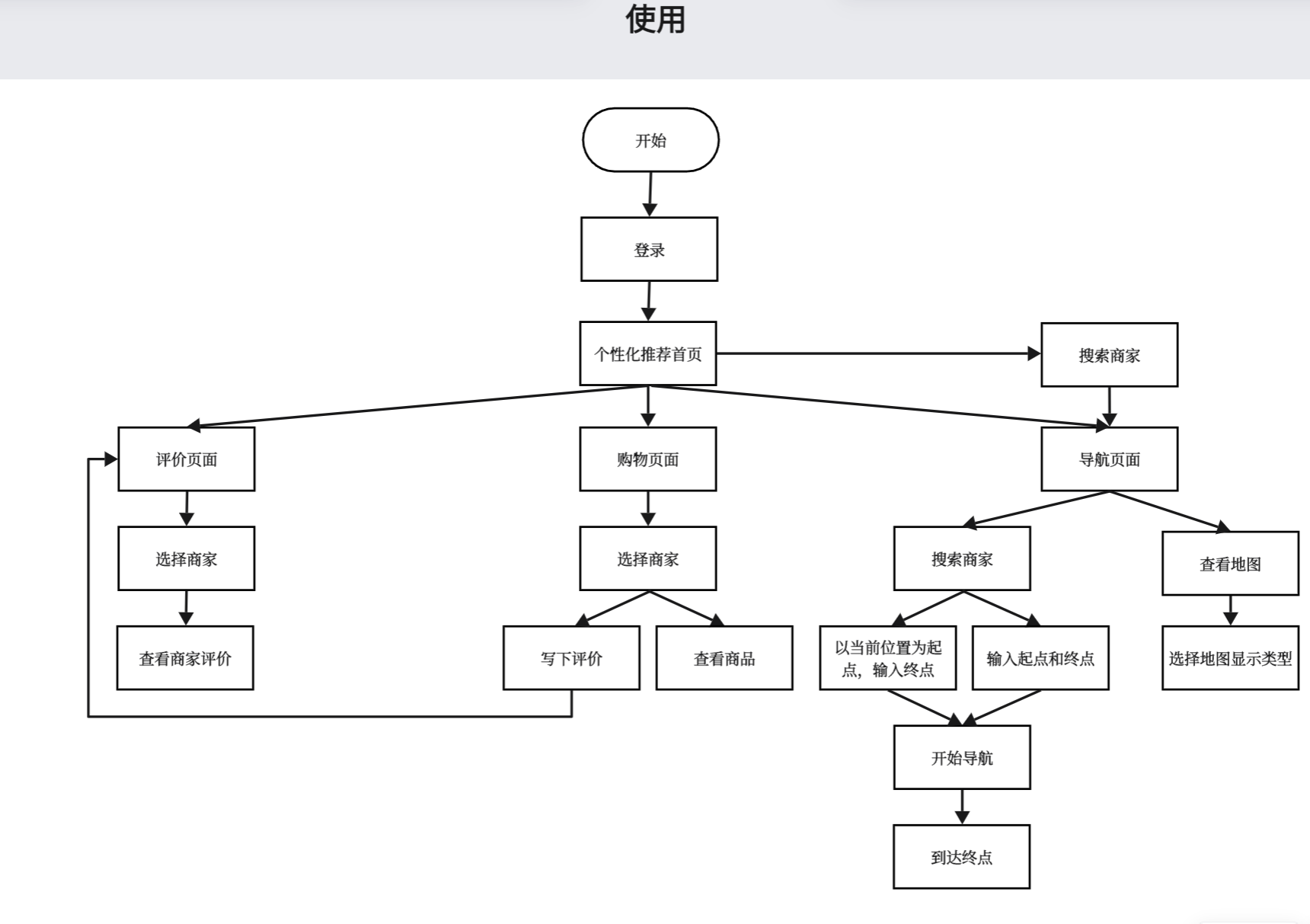
## 错误/异常处理设计



1. 定义了服务端返回值错误代码，分别对应了注册成功/失败、登录账号不存在、登陆密码错误等等。
2. 为各中异常抛出都做了详细的日志记录，以及错误追踪。
3. 自定义错误类型，便于处理更多的错误情景。

# 详细设计与实现

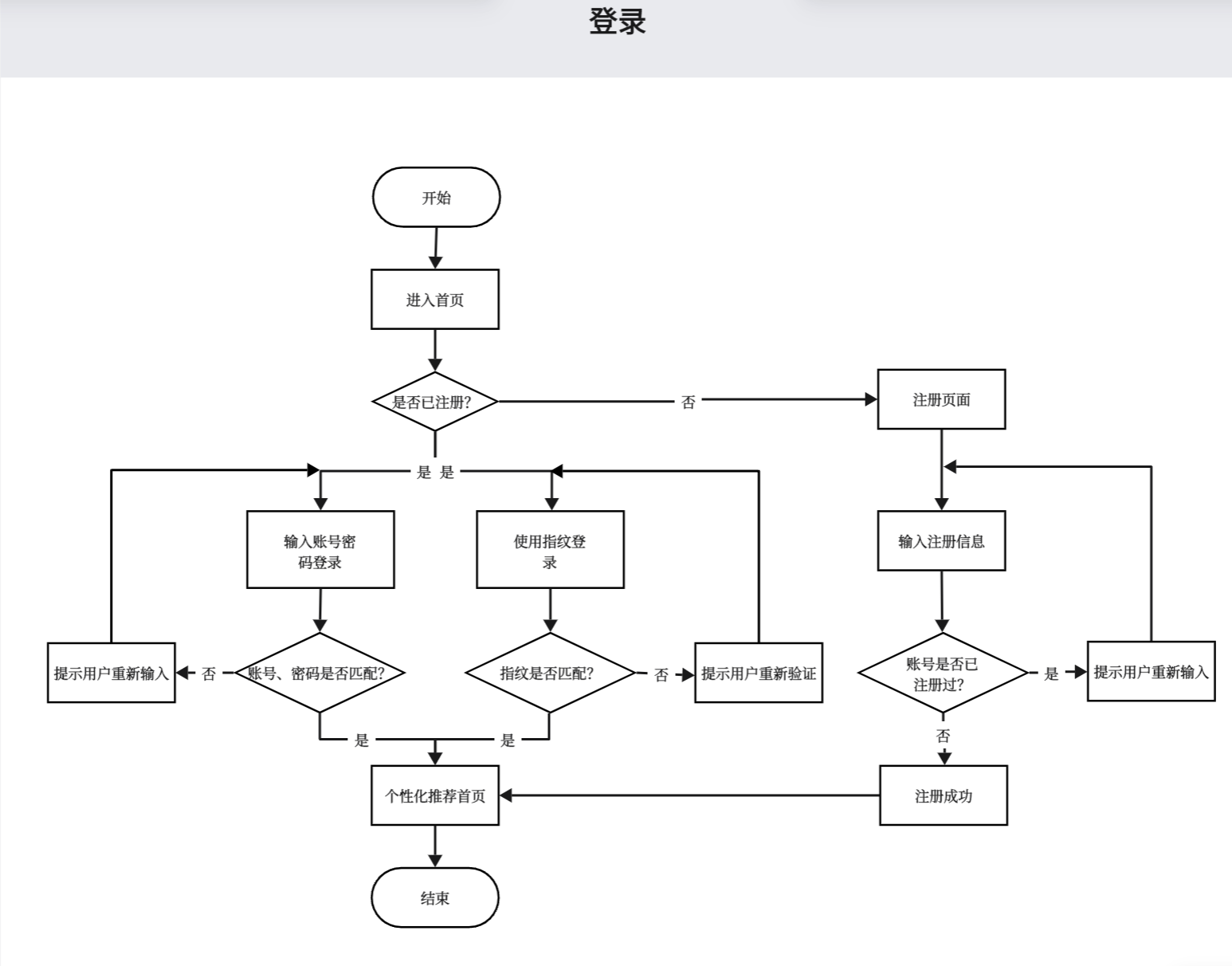
系统总体流程图



## 登录注册子系统的设计与实现

### 类图、流程图、时序图等

### 登录注册



### 用户界面展示

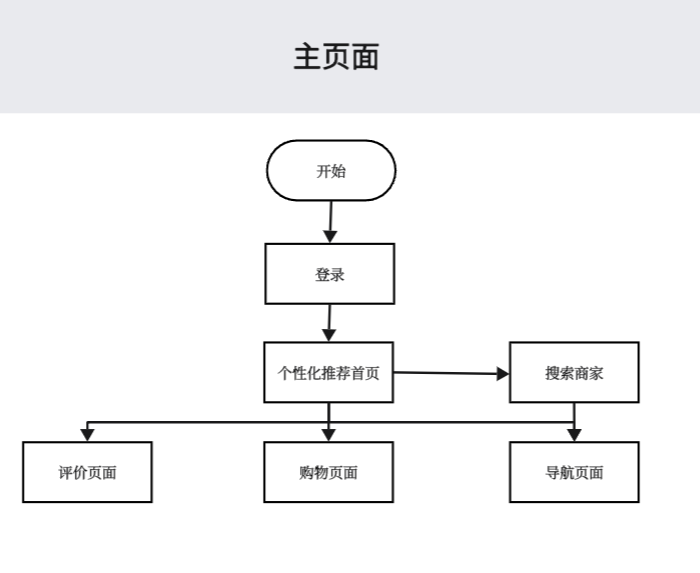
特别的，用户可以选择通过指纹识别进行登录功能，他会识别每个指纹，与手机。

本地存储的指纹去匹配，如果相互匹配则成功登录。在这里我们加入了记住密码，验证登录的相关功能，提升了用户体验，对指纹识别也加入了验证是否登录过的功能。



## 主界面子系统的设计与实现

### 流程图等



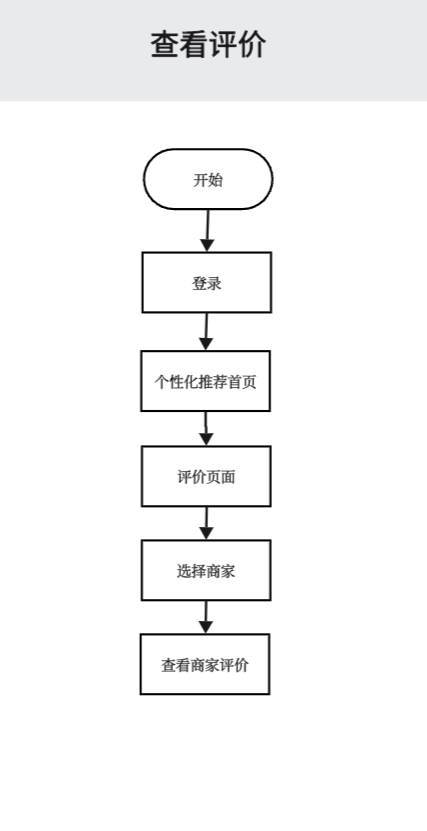
### 用户界面展示

在主界面用户可以直接进行搜索开始导航，也可以根据下放的服务提示前往对应的服务功能模块



## 评论子系统的设计与实现

### 类图、流程图、时序图等

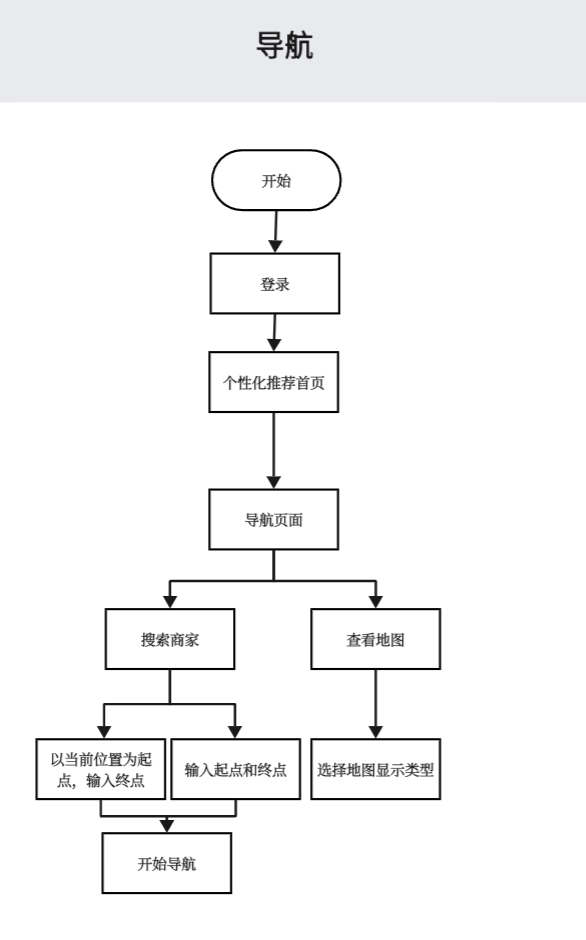


### 7.3.2用户界面展示



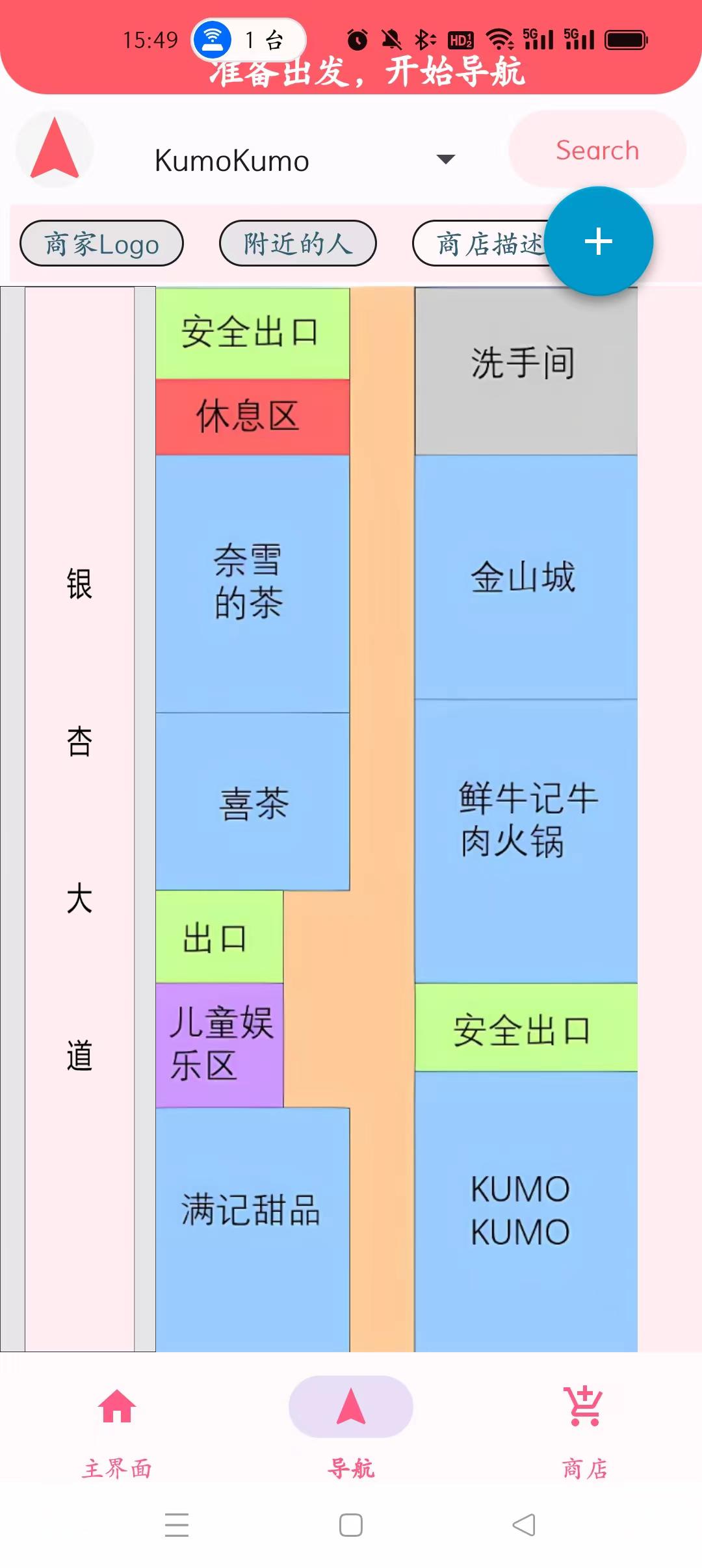
## 导航子系统的设计与实现

### 类图、流程图、时序图等



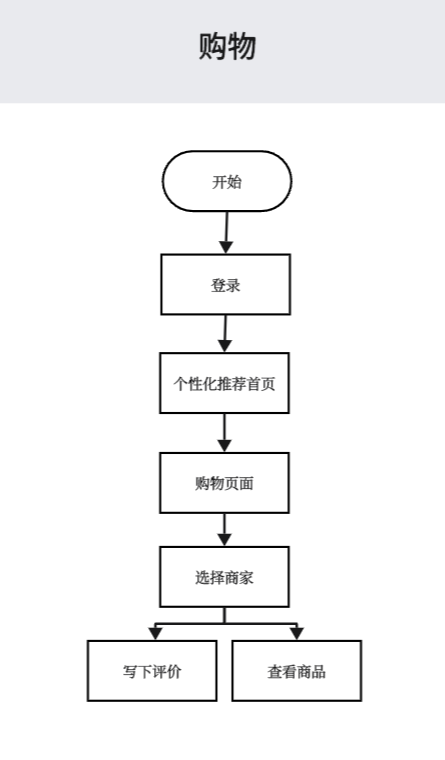
### 用户界面展示

用户可以进行导航，点击右上角的加号可以实现指定路径的导航的功能，同时提供了个性化服务，用户可以选择不同的导航图片



## 商家子系统的设计与实现

### 类图、流程图、时序图等



### 用户界面展示

通过点击不同商家，在此界面用户可以查看商家对应的商品。右上角的加号支持用户进行评论功能，在不同的商家下会实现对应商家的评价。



## 核心算法（伪代码）

#### 7.6.1基于RecycleView的界面展示

静态数据适配器 StaticRvAdapter

Class StaticRvAdapter Inherits RecyclerView.Adapter

List items

Context context

Constructor(items, context)

this.items = items

this.context = context

Method onCreateViewHolder(parent, viewType) Returns StaticViewHolder

View view = Inflate view from context

Return new StaticViewHolder(view)

Method onBindViewHolder(holder, position)

StaticRvModel item = items[position]

holder.imageView.SetImageResource(item.image)

holder.textView.SetText(item.text)

Method getItemCount() Returns Integer

Return items.size

Method onViewRecycled(holder)

Clear or reset view components if necessary

Class StaticViewHolder Inherits RecyclerView.ViewHolder

ImageView imageView

TextView textView

Constructor(itemView)

super(itemView)

imageView = itemView.FindViewById(R.id.imageView)

textView = itemView.FindViewById(R.id.textView)

动态数据适配器 DynamicRVAdapter

Class DynamicRVAdapter Inherits RecyclerView.Adapter

List items

Context context

Constructor(items, context)

this.items = items

this.context = context

Method onCreateViewHolder(parent, viewType) Returns DynamicViewHolder

View view = Inflate view from context

Return new DynamicViewHolder(view)

Method onBindViewHolder(holder, position)

DynamicRVModel item = items[position]

holder.textView.SetText(item.text)

holder.ratingBar.SetRating(item.rating)

Method getItemCount() Returns Integer

Return items.size

Method onViewRecycled(holder)

holder.ratingBar.SetRating(0)

Method setItems(newItems)

items = newItems

NotifyDataSetChanged()

Class DynamicViewHolder Inherits RecyclerView.ViewHolder

TextView textView

RatingBar ratingBar

Constructor(itemView)

super(itemView)

textView = itemView.FindViewById(R.id.textView)

ratingBar = itemView.FindViewById(R.id.ratingBar)

#### 7.6.2定位算法

function connect\_to\_mysql(host, user, password, database):

try:

connection = establish\_mysql\_connection(host, user, password, database)

return connection

except connection\_error:

print\_error\_message("Could not connect to MySQL")

return None

function fetch\_training\_data(connection):

try:

cursor = create\_cursor(connection)

query = "SELECT Time, Position, BSSID, RSSI FROM data"

execute\_query(cursor, query)

training\_data = []

for each row in cursor:

add\_row\_to\_training\_data(training\_data, row)

return training\_data

except query\_error:

print\_error\_message("Error fetching training data")

return None

finally:

close\_cursor\_if\_exists(cursor)

function find\_most\_similar\_position(train\_host, train\_user, train\_password, train\_database, input\_data):

connection = connect\_to\_mysql(train\_host, train\_user, train\_password, train\_database)

if connection is None:

return None, None

try:

training\_data = fetch\_training\_data(connection)

if training\_data is None:

return None, None

df = create\_dataframe(training\_data)

df = filter\_dataframe(df, 'Position')

grouped = group\_dataframe(df, ['Time', 'Position'])

direction\_vectors = {}

for each (time, direction), group in grouped:

vector = create\_vector\_from\_group(group)

direction\_vectors[(time, direction)] = vector

input\_vector = create\_input\_vector(input\_data)

similarities = {}

for each (time, direction), vector in direction\_vectors:

aligned\_vectors = align\_vectors(vector, input\_vector)

if aligned\_vectors are valid:

sim = calculate\_cosine\_similarity(aligned\_vectors)

similarities[(time, direction)] = sim

top\_k\_directions = get\_top\_k\_directions(similarities, 3)

direction\_counts = count\_directions(top\_k\_directions)

most\_common\_direction = get\_most\_common\_direction(direction\_counts)

print\_most\_similar\_position(most\_common\_direction)

return most\_common\_direction

except query\_error:

print\_error\_message("Error during processing")

return None

finally:

close\_connection\_if\_exists(connection)

## 接口设计

接口设计

1. 登录:路径： /login/login，类型POST，接收的值：表单，"username"和"password"，返回的值：登录成功："100"、密码错误："202"、账号不存在："201"。
2. 注册：路径：/login/register，类型POST,接收的值：表单，"username"和"password"，返回的值：成功："101"、失败(用户名重复)"1062"。
3. 登录验证：/is\_login\_in，类型GET，返回的值：已登录；"100"、未登录："205"。
4. 登出账号：/login/logout,类型PSOT,返回的值：成功:"1"。
5. 评论上传：/comment，类型POST,接收的值：表单，"sid"(代表商家)、"comment"(评论内容)和"stars"(评分)，返回的值：成功："100"、错误："206"。
6. 用户头像上传：/user/upload\_ic,类型POST,接收的值：文件信息、二进制文件，返回的值：成功："100"、错误："204"。
7. 获取用户头像：/user/get\_uic,类型POST，接受的值：表单，"uid"目标用户的，返回的值：目标用户的头像。
8. 设置用户名字：/user/get\_name,类型GET，返回的值：目标用户的名字。
9. 修改用户名字：/user/update\_name,类型POST，接受的值：表单，"name"目标用户的新名字，返回的值：成功："100"、失败："204"、数据库错误:数据库错误信息。
10. 获取评论：/get\_comment，类型POST，接收的值：表单，"sid"(代表商家)，返回的值：(json)[{"name":用户名，"comment":评论内容,"stars":评价的星级}、etc]
11. 定位：/locator，类型POST，接收的值：用户端收集到的WiFi指纹信息，返回的值：(json){"status":状态，"most\_similar\_position":最相似的位置，“coornidate”:这个位置的坐标}