同济大学
TONGJI UNIVERSITY

软件工程课程设计
实习报告

项目名称	手机私家车拼车软件——拼途 APP
组　　号	第 21 组
组 员 1	
组 员 2	
组 员 3	
组 员 4	
组 员 5	
指导老师	曾国荪老师

二〇二五　年　　六　月　　十二　日

成员分工

学号	姓名	负责内容	贡献度
		需求分析阶段原型设计； 主界面、搜索界面与拼途广场相关界面代码实现； MySQL 数据库设计与 Flask 后端实现； 文档维护；	20%
		主界面与路径规划详情界面代码实现； React Native + Android 项目环境搭建与维护； 第三方高德地图 API 与 SDK 集成； 软件测试与测试文档撰写；	20%
		需求分析阶段原型设计； 软件总体结构设计与详细设计； 座驾管理代码实现； 主要文档设计与撰写；	20%
		群聊功能代码实现； 软件功能需求分析阶段的用例分析、顺序分析与状态分析； 软件测试部分的测试用例搭建； 文档维护；	20%
		个人中心代码实现； 软件功能需求分析阶段的 UML 建模分析； 软件接口文档撰写与接口管理； 文档维护；	20%

目 录

一、 项目概要 1
1.1 项目背景与系统概述 1
1.2 目标用户 1
二、 可行性分析 2
2.1 技术可行性 2
2.2 经济可行性 3
2.3 运营可行性 3
2.4 社会可行性 3
三、 需求分析 3
3.1 功能需求 3
3.1.1 用户管理功能 3
3.1.2 拼车服务功能 4
3.1.3 座驾管理功能 4
3.1.4 位置服务功能 4
3.1.5 即时通讯功能 4
3.1.6 评价系统功能 4
3.2 性能需求 5
3.2.1 规模大小 5
3.2.2 存储容量限制 5
3.2.3 执行速度与响应时间 5
3.2.4 吞吐量与并发个数 5
3.2.5 计算精度 5
3.2.6 可靠性和安全性 5
3.3 环境需求 6
3.3.1 硬件需求 6
3.3.2 软件需求 6
3.4 界面需求 6
3.4.1 界面划分及主要元素 6
3.4.2 与其他系统的交互 6
3.5 系统数据流图 6
四、 总体设计 7
4.1 体系结构设计 7
4.1.1 分层架构设计 7
4.1.2 模块间的相对关系 8
4.1.3 体系结构的特点 9
4.2 层次架构 10
4.3 过程设计 11
4.3.1 用户登录注册账号 11
4.3.2 座驾管理 11
4.3.3 拼车服务 12
4.3.4 拼车订单群聊 13
4.3.5 个人中心 14
4.4 接口设计 15
4.4.1 用户接口 15
4.4.2 外部接口 16
4.4.3 内部接口 16
4.5 数据结构设计 17
4.5.1 数据库表及其字段说明 17
4.5.2 表间关系 20
五、 UML 建模分析 21
5.1 针对进行 UML 分析建模 21
5.2 UML 建模分析 22
5.2.1 功能模型 22
5.2.2 对象模型 23
5.2.3 行为模型 24
六、 详细设计 27
6.1 系统架构设计 27
6.2 模块划分 27
6.3 模块设计 29

6.3.1 用户认证模块 29
6.3.2 发起拼车模块 30
6.3.3 参与拼车模块 32
6.3.4 订单管理模块 34
6.3.5 座驾管理模块 36
6.3.6 个人中心模块 38
6.3.7 拼车群聊模块 40
6.4 API 接口 42
6.4.1 用户认证接口 42
6.4.2 个人中心接口 43
6.4.3 座驾管理接口 43
6.4.4 订单管理接口 43
6.4.5 订单操作接口 44
6.4.6 评价接口 45
6.4.7 地图服务接口 45
七、 编码规范 46
7.1 命名规则 46
7.1.1 项目整体命名 46
7.1.2 目录命名 46
7.1.3 文件命名 47
7.2 前端编码规范 48
7.2.1 命名规范 48
7.2.2 代码格式化风格 48
7.2.3 语句与表达式 49
7.2.4 异常处理规范 50
7.3 后端编码规范 50
7.3.1 命名规范 50
7.3.2 代码格式化规范 51
7.3.3 语句与表达式 51
7.3.4 异常处理规范 52
八、 代码编写 52
8.1 技术栈架构 52
8.2 功能实现 54
8.2.1 页面导航与跳转 54
8.2.2 司机发车与乘客拼车 55
8.2.3 路径规划与订单详情 58
8.2.4 群聊与行驶全过程的操作 59
8.2.5 用户评价系统 59
8.2.6 用户数据管理 60
九、 软件测试 60
9.1 测试需求 60
9.1.1 功能性测试需求 60
9.1.2 非功能性测试需求 61
9.2 测试框架 62
9.3 单元测试 62
9.4 集成测试 70
9.5 确认测试 74
十、 效果展示 78
10.1 运行说明 78
10.2 功能展示 78
十一、 项目管理 88
十二、 心得体会 89

一、项目概要

1.1 项目背景与系统概述

在现代城市中，交通拥堵与环境污染问题日益严重，拼车作为一种新兴的出行方式，正逐渐受到大众关注和支持。拼车不仅能够有效缓解交通压力，减少道路上的车辆数量，还能够降低能源消耗与尾气排放，改善环境质量。同时，对于出行者而言，拼车有助于节省出行成本，实现资源的优化配置。

基于此背景，我们致力于开发一款手机私家车拼车软件，旨在为用户提供更加便捷、高效、环保、个性化的出行解决方案。这款软件基于智能匹配算法，根据用户输入的出发地点、目的地、出行时间等精准匹配顺路的私家车车主与乘客；同时考虑用户个性化需求，提供拼车广场供乘客自行选择，最大程度提高拼车的成功率与出行效率。

该软件具备丰富且实用的功能。在登录注册方面，通过加密技术保障用户信息安全；围绕核心的拼车服务，用户创建拼车和参与拼车灵活设定需求，快速匹配合适行程，拼友聊天便于多方实时沟通行程细节，拼车广场展示拼车发布信息，便于乘客自由选择；个性化管理方面，座驾管理便于车主完善车辆信息，合规出行，个人中心管理信息资料，满足用户多样化需求。

在系统设计方面，我们采用先进的技术架构，确保软件运行的稳定性与流畅性。界面设计秉持简洁直观的原则，操作流程简便易懂，极大提升用户体验。拼车匹配算法高效快捷，精准匹配最佳拼车组合，减少等待时间，提高出行效率、同时，软件提供行程追踪和报警服务，为用户的出行安全保驾护航。

综上所述，这款手机私家车拼车软件的开发具有重要意义。它不仅为人们的出行方式带来创新变革，让出行更加轻松便捷；同时在环保节能方面发挥积极作用，减少能源浪费与尾气排放，助力城市可持续发展。相信随着软件的推广使用，将为缓解交通拥堵，改善环境质量做出积极贡献，为广大用户创造更加绿色、高效、美好的出行体验。

1.2 目标用户

私家车拼车软件的目标用户群体可划分为以下类别，每类用户都具有鲜明的需求特征和行为模式。

- **通勤上班族：**

拼车软件的核心用户群体。此类用户通常有固定的乘车路线（家和公司之间），工作日几乎每天都需要使用拼车服务。他们最看重的是经济实惠和准时高效，通过拼车来节省出行成本，确保准时到达；往往倾向于寻找长期稳定的拼车伙伴，建立固定的拼车关系。

- **偶尔出行者：**

拼车软件的补充用户群。此类用户没有固定的拼车需求，仅在特定情况下选择拼车服务，比如周末出游、临时加班或者特殊约会等。他们的使用频率较低，出行路线和时间都不固定，更依赖平台的即时匹配功能。

环保倡议者：拼车软件的重要支持者。此类用户选择拼车主要是出于环保理念，希望通过共享出行减少碳排放。他们愿意为环保理念支付稍高的费用，是平台绿色出行理念的最佳传播者。

- **私家车主：**

拼车软件的供给端用户。此类用户作为拼车服务的关键一环，通过平台分享闲置座位，主要目的是分摊油费，降低出行成本。部分车主还希望通过拼车结识新朋友，为长途驾驶增添乐趣。

- **无车乘客：**

拼车软件的需求端用户。由于没有私家车，此类用户需要依赖拼车来完成日常出行。他们对价格极为敏感，会对比多种出行方式后选择最经济的方案；通常选择拼车往返于住所和地铁站、商业区等目的地。安全性和准时性是此类用户最关系的两个因素。

二、可行性分析

2.1 技术可行性

当前移动互联网技术已相当成熟，GPS 定位、LBS 服务和实时通信等技术在出行领域得到广泛应用。拼车软件所需的核心技术包括智能匹配算法、实时导航和在线支付系统等，这些技术在国内主流出行平台已有成功实践案例。

通过对现有平台技术架构的研究发现，基于微服务的分布式架构能够满足高并发需求，各类地图 API 和支付接口的开放也大大降低了开发难度。但项目仍面临一些技术挑战，其中最大的难点在于实现精准的路线匹配和动态定价算法，这需要大量真实出行数据进行模型训练。此外，确保用户隐私数据安全也是需要重点解决的问题。

2.2 经济可行性

作为学生课程设计项目，实际开发成本主要体现在使用校园现有计算机资源、采用开源框架和免费开发工具等方面。项目团队将主要利用课余时间进行开发，并使用公开数据集或模拟数据。虽然不涉及真实商业运营，但该项目仍具有重要的学习价值和创新意义，能够帮助学生掌握移动应用开发全流程，同时探索绿色出行解决方案的设计可行性。

2.3 运营可行性

市场调研显示大部分受访者对拼车服务持开放态度，主要关注点集中在安全性和匹配效率上。虽然共享经济理念已在国内形成共识，但拼车文化的培育仍需加强，需要通过建立完善的信用体系和采取有效的安全保障措施来提升用户信任度。项目设计考虑了功能扩展性和算法优化空间，具备理论上的可持续发展潜力。

2.4 社会可行性

该项目符合国家绿色出行政策导向，有望为缓解交通拥堵和环境污染问题提供解决方案。但需要特别注意可能引发的安全问题和社会纠纷，在隐私保护方面要严格遵守相关法律法规，对位置信息等敏感数据进行特别保护。同时，需要明确平台、车主和乘客三方的权责关系，并关注各地拼车监管政策的差异性，确保项目设计与现行法规要求相符。

三、需求分析

3.1 功能需求

3.1.1 用户管理功能

（1）用户注册：

系统应提供基于邮箱验证的注册功能，支持用户身份信息的安全存储与验证，确保注册流程的完整性与用户信息的真实性。

（2）用户登录：

系统需实现用户使用注册时的邮箱和密码进行登录。登录状态在有效期内保持，过期后需要重新登录，确保账号安全。

（3）个人信息管理：

系统应提供完整的个人信息管理模块，支持基础信息维护、证件信息上传及用户行为数据统计功能，建立完善的用户信用评价体系。

3.1.2 拼车服务功能

（1）发起拼车：

系统需支持车主创建拼车行程的功能，包括行程信息设置、车辆选择、乘客容量设定及自动费用计算等核心业务流程。

（2）参与拼车：

系统需支持乘客创建拼车需求行程的功能，行程信息设置后系统进行智能匹配。同时，系统应提供拼车订单浏览与检索功能，支持用户查看行程详情、提交参与申请及跟踪申请状态的全流程管理。

（3）订单管理：

系统需实现订单全生命周期管理功能，包括状态跟踪、行程确认及流程控制、费用结算等核心操作，确保拼车服务流程的规范性和可追溯性。

3.1.3 座驾管理功能

（1）座驾信息管理：

系统应提供车辆信息管理功能，支持多车辆的信息维护与查询，包括基础信息管理、当前车辆选择等车主服务需求。

3.1.4 位置服务功能

（1）地点管理：

系统需集成地图服务 API，实现智能地点检索、常用位置管理及地理信息存储功能，提升用户位置选择的效率和准确性。

（2）路线规划：

系统应提供基于地理信息的智能路线规划服务，支持行程距离计算、费用预估等核心功能，为拼车服务提供基础数据支持。

3.1.5 即时通讯功能

（1）群聊管理：

系统需实现基于拼车订单的即时通讯功能，支持多人会话管理、成员状态显示及身份识别等基础通讯需求。

3.1.6 评价系统功能

（1）行程评价：

系统应建立行程服务质量评价机制，同时支持私家车主和无车乘客的双向评价，支持多维度的服务评价收集与分析，为服务改进提供数据支持。

（2）用户评价：

系统需构建用户信用评价体系，实现用户互评机制及行为数据统计功能，形成可量化的用户信用档案。

3.2 性能需求

3.2.1 规模大小

系统设计容量为支持日活跃用户 10,000 人左右，同时在线用户峰值约 2,000 人，能够满足城市区域范围内的拼车需求。这个规模设计既考虑了实际应用场景的用户量级，又确保了系统在初期运营阶段的稳定性。系统架构采用可扩展设计，可根据用户增长情况灵活扩容，支持未来业务规模扩大至日活跃用户 50,000 人的水平。在用户分布方面，系统将重点覆盖城市通勤高峰时段的需求，同时兼顾非高峰时段的拼车匹配需求。

3.2.2 存储容量限制

系统预计需要存储 5000 个用户的基本信息，保留最近 3 个月的行程记录数据，同时永久保存用户评价信息。这样的存储方案既保证了数据的完整性，又避免了存储资源的过度占用。

3.2.3 执行速度与响应时间

系统要求关键操作响应时间控制在 2 秒以内，行程搜索匹配时间不超过 1 秒，地图加载时间限制在 3 秒内。这样的性能指标能够确保用户获得流畅的使用体验。

3.2.4 吞吐量与并发个数

系统架构设计需支持 100 个并发用户同时操作，这一指标充分考虑了课程设计的实际应用场景，既保证了系统的实用性，又不会过度增加开发难度。

3.2.5 计算精度

系统要求路线距离计算基于高德地图的路径规划 API，误差控制在 50 米范围内，费用计算精确到 0.01 元，时间计算精确到分钟。这些精度指标能够满足拼车服务的基本需求，同时保证计算的准确性。

3.2.6 可靠性和安全性

系统设计要达到 99%的可用性，采用数据加密存储、关键操作二次验证等安

全措施。通过以上措施，系统能够在提供稳定服务的同时，有效保护用户数据安全和隐私，防范各类安全风险，为用户创造安全可靠的拼车环境。

3.3 环境需求

3.3.1 硬件需求

系统运行需要配置 2 核 CPU、4GB 内存和 100GB 存储空间的服务器环境，客户端需支持 Android 8.0 以上系统的智能手机设备，并要求稳定的网络连接。

3.3.2 软件需求

系统开发需要使用 Windows Server 操作系统，MySQL 5.7 以上版本数据库，开发工具采用 Android Studio 和 VScode，并集成主流地图 API 和支付接口。

3.4 界面需求

3.4.1 界面划分及主要元素

系统界面包括登录注册界面、发起拼车/参与拼车主界面、拼友聊天界面、拼车广场界面、行程详情界面、座驾管理界面和个人中心界面。每个界面都设计了相应的功能区域和操作元素，确保用户操作的便捷性和直观性。

3.4.2 与其他系统的交互

系统需要与高德地图 API 对接获取路线规划服务，与支付系统对接完成交易流程，与邮箱系统对接实现验证码发送和通知功能。这些交互设计保证了系统的功能完整性。

3.5 系统数据流图

乘客、车主和系统作为数据的源头/终点，在拼车订单表、拼车需求表、拼车订单群聊、车主座驾表和个人信息表五个数据存储中心之间进行数据流操作，形成手机私家车拼车系统的数据流图。

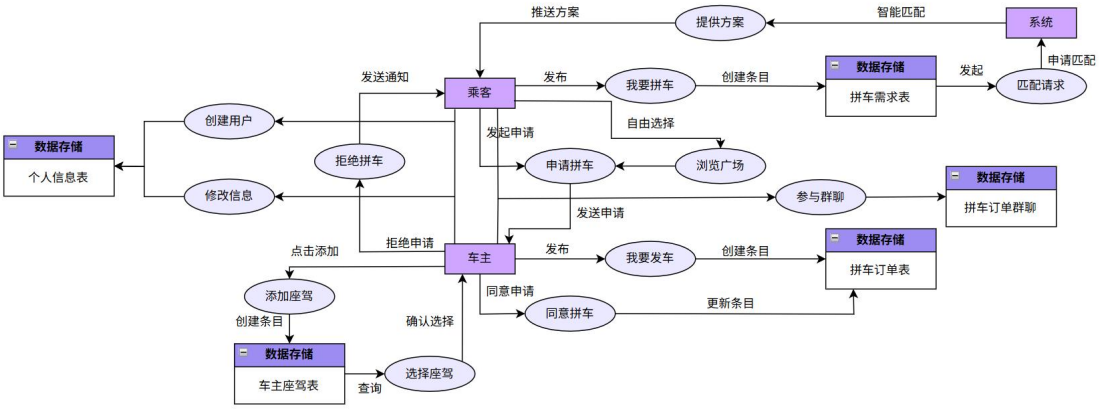

图 1 手机私家车拼车系统数据流图

四、总体设计

4.1 体系结构设计

4.1.1 分层架构设计

系统的体系结构可以分为以下主要层次，每一层次包含了与之对应的功能模块：

表现层主要负责与用户的直接交互，包括用户界面和传入模块两个主要部分。用户界面负责展示系统的各项功能，包括用户登录注册、拼车发起、拼车广场、订单详情以及个人中心等界面，为用户提供直观的操作入口。传入模块则负责处理所有来自用户的输入数据，执行表单验证以确保数据的合法性，对输入数据进行预处理如时间格式转换和坐标处理，同时还要响应用户的各种交互操作，以及处理地图相关的操作请求。

业务逻辑层是系统的核心处理层，由变换模块/加工模块和协调模块构成。变换模块/加工模块实现了系统的核心业务逻辑，包括用户认证服务、订单处理服务和匹配服务。用户认证服务处理用户的身份验证，订单处理服务负责订单的全生命周期管理，匹配服务则通过复杂的算法实现路线、时间、价格和座位的智能匹配。协调模块则负责管理整个业务流程，确保发起拼车、参与拼车和订单完成等流程的顺利进行，同时控制用户信息、订单信息和支付信息等数据在系统各模块间的正确流转。

数据访问层主要由数据库模块和第三方服务集成两部分组成。数据库模块规范管理系统的各类数据，包括用户数据、订单数据、申请数据和车辆数据等。用

户数据管理不仅包括基本信息的维护，还包括用户评分和信用记录的管理；订单数据管理负责订单信息的存储和状态更新；申请数据管理则处理拼车申请的记录和状态变更；车辆数据管理维护车主的车辆信息。第三方服务集成模块则负责与外部系统的对接，主要包括高德地图服务的集成，实现地点搜索、路线规划和距离计算等功能，以及短信服务的集成，用于验证码发送和通知消息推送。

外部接口层包括传出模块和 API 接口两个部分。传出模块负责将系统的处理结果推送给用户或外部系统，包括各类订单状态的推送（如新订单通知、申请状态更新、订单完成通知）以及界面数据的展示（如订单列表渲染、路线地图展示、用户信息展示）。API 接口则为移动应用提供了与后端系统交互的标准接口，包括用户相关 API（如注册、登录）、订单相关 API（如创建订单、订单匹配、申请拼车）、车辆相关 API（如添加车辆、获取车辆列表）以及评价相关 API（如创建评价、获取评价列表）等。

这种分层架构设计实现了系统各个功能模块的解耦，使得系统具有良好的可维护性和可扩展性。每一层都有其明确的职责，层与层之间通过定义好的接口进行通信，确保了系统的稳定性和可靠性。同时，这种设计也便于团队协作开发和后期的功能扩展。通过清晰的层次划分和模块职责定义，系统能够高效地处理拼车业务的各项需求，为用户提供流畅的拼车服务体验。

4.1.2 模块间的相对关系

在拼车系统中，各层次模块之间形成了紧密而有序的协作关系，通过标准化的接口和数据流转确保系统的正常运行。

表现层与业务逻辑层之间的交互主要通过数据的双向传递实现。当用户在拼车系统的界面上进行操作时，如发起拼车或提交拼车申请，表现层的传入模块会收集并预处理这些用户输入的数据，包括出发地点、目的地、出发时间、拼车人数等信息。这些经过初步验证的数据随后被传递给业务逻辑层进行进一步处理。业务逻辑层完成处理后，处理结果会通过传出模块返回给表现层，由表现层将结果以适当的方式呈现给用户，如显示订单创建成功提示或展示匹配到的拼车订单列表。

业务逻辑层与数据访问层之间的关系体现了系统的核心数据处理流程。业务逻辑层的协调模块承担着重要的调度职责，它根据业务需求组织和管理变换模块的执行顺序。在处理拼车业务时，协调模块会根据具体的业务场景调用相应的数据库模块进行数据操作。例如，在处理拼车申请时，需要先检查订单状态，然后创建申请记录，最后更新相关统计数据，这些操作都需要通过数据库模块来完成。同时，对于需要使用地图服务进行路线规划或距离计算的场景，协调模块会调用

第三方服务集成模块，与高德地图服务进行交互，获取所需的地理信息数据。

业务逻辑层与外部接口层之间的关系则主要体现在数据传输和服务调用方面。外部接口层的 API 接口为移动端应用提供了标准化的数据访问通道，连接着表现层和业务逻辑层。当移动端需要获取或提交数据时，会通过这些 API 接口与服务器进行通信。例如，在用户发起拼车时，移动端会调用创建订单的 API 接口，该接口会将请求数据传递给业务逻辑层进行处理，处理完成后的结果再通过 API 接口返回给移动端。这种标准化的接口设计不仅确保了表现层与业务逻辑层之间的无缝交互，还提供了良好的扩展性，便于未来功能的扩展和系统的升级维护。

通过这种清晰的模块间关系设计，拼车系统实现了高内聚、低耦合的架构特征。各个模块之间通过标准化的接口进行通信，既保证了数据流转的顺畅，又维持了模块间的相对独立性。这种设计不仅提高了系统的可维护性和可扩展性，还为系统的稳定运行提供了可靠的架构支持。同时，这种模块化的设计也便于团队协作开发，不同的开发人员可以专注于不同模块的开发，提高了开发效率。

4.1.3 体系结构的特点

拼车系统的体系结构设计体现了现代软件工程的先进理念，具有显著的模块化设计、分层架构和高可扩展性等特点。

在模块化设计方面，系统将拼车业务划分为多个功能独立的模块，如用户认证、订单处理、匹配服务等。这些模块可以独立进行开发、测试和部署，有效降低了系统的整体复杂性。例如，开发团队可以专注于匹配算法的优化，而无需过多考虑用户界面或数据存储的实现细节，这种设计大大提高了系统的可维护性和开发效率。

分层架构的设计使得系统在逻辑结构上更加清晰。表现层、业务逻辑层、数据访问层和外部接口层各司其职，每一层都有其明确的职责和边界。数据和控制流在这些层次之间有序流动，形成了高效的处理链条。例如，用户发起拼车请求时，数据从表现层传递到业务逻辑层处理，通过数据访问层存储，最后结果通过外部接口层返回。这种分层设计确保了系统的高内聚性和低耦合性，因为各层之间通过标准化的接口通信，减少了相互依赖。

系统的高可扩展性主要体现在其灵活的架构设计上。通过协调模块的统一调度和管理，系统能够轻松应对新的功能需求或业务流程的变化。例如，增加新的支付方式只需在第三方服务集成模块中添加相应接口，优化路线规划算法也可以直接在匹配服务模块中更新，而不会影响其他功能。这种灵活的可扩展性不仅支持功能扩展，还便于系统性能的优化，使系统能够持续演进，不断满足用户需求。

4.2 层次架构

手机私家车拼车系统的功能结构可以分为三个主要模块：用户创建、拼车服务和信息管理。

用户创建模块主要包含登录账号和注册账号两个基本功能，负责用户身份的创建和验证，确保用户能够安全地使用系统服务。

拼车服务模块是系统的核心功能模块，包括发起拼车、参与拼车、拼车广场和拼车群聊四个主要功能。其中，发起拼车和参与拼车功能下设多个子功能：收发申请功能用于处理拼车请求，信息填写用于录入拼车信息，费用结算处理车费相关事务，评价订单用于用户互评；拼车广场功能下设浏览广场和参与拼车两个子功能，方便用户查看和选择合适的拼车订单；拼车群聊则提供实时交流的功能，促进拼车参与者之间的沟通。

信息管理模块分为座驾管理和个人中心两部分。座驾管理包含创建座驾和选择座驾功能，方便车主管理自己的车辆信息；个人中心则包括修改信息和驾照上传功能，使用户能够维护和更新个人资料。

这种层次结构设计清晰地划分了系统的功能模块，既保证了各个功能的独立性，又通过合理的层次关系维持了功能间的有机联系，为用户提供了完整的拼车服务体验。

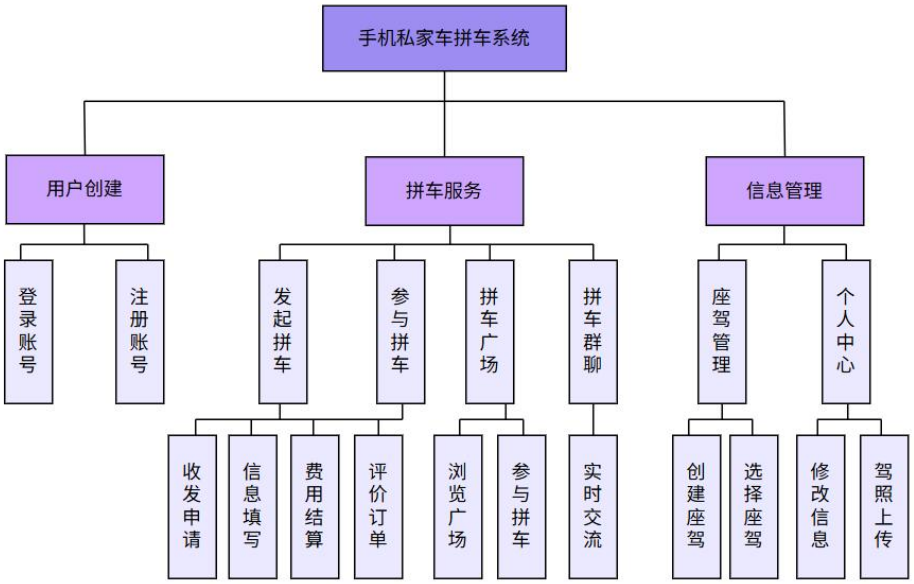

图 2 手机私家车拼车系统层次架构图

4.3 过程设计

4.3.1 用户登录注册账号

新用户首次登录进行账号注册。用户输入邮箱和密码，发起账号注册请求。系统检查邮箱是否有效，若验证通过系统向邮箱发送验证邮件，若验证失败进入错误处理。邮件发送成功后，系统生成固定格式随机用户名，对密码进行哈希加密处理，创建新的用户记录，注册成功。

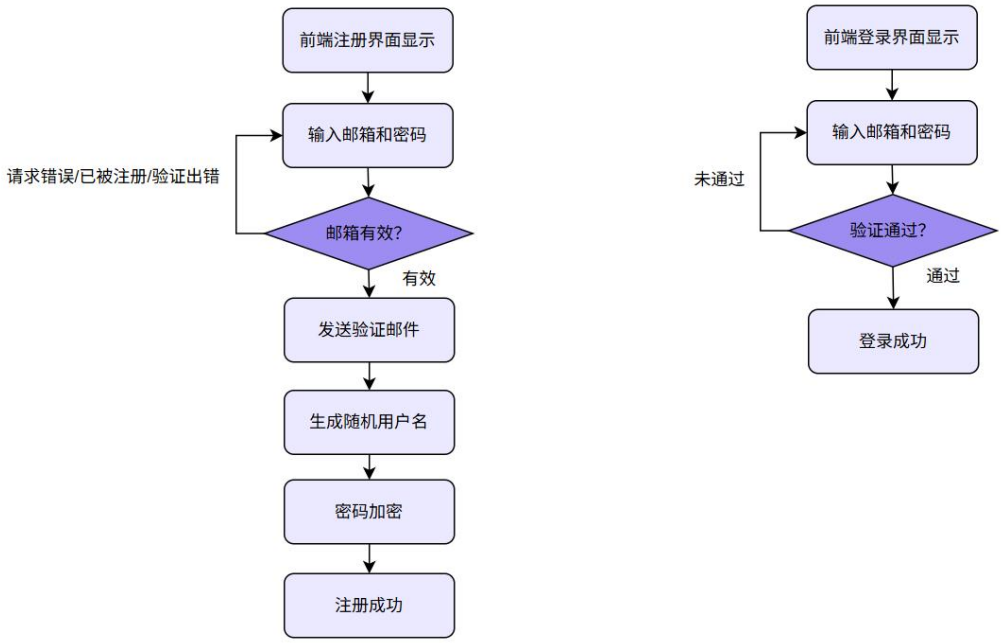

图 3 用户注册登录账号流程图

4.3.2 座驾管理

用户在座驾管理界面拥有三个可选功能：查看座驾情况，若存在座驾则展示座驾列表，否则范湖空列表；添加座驾，输入座驾信息，系统进行车牌合规性核验，通过则创建新座驾记录，否则错误处理；删除座驾，用户点击需要删除的座驾，系统验证所有权，通过则删除座驾记录，否则错误处理。

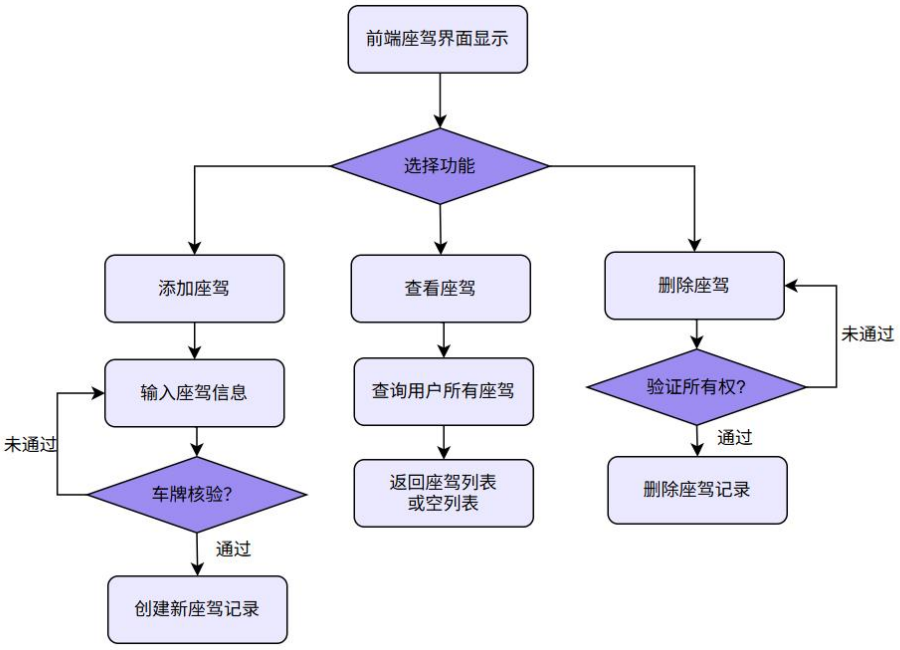

图 4 用户座驾管理流程图

4.3.3 拼车服务

主界面包含拼车和发车两个界面，用户根据需求选择。

私家车主发起拼车，填写发车信息，系统检测完整性通过后成功创建拼车订单，用户跳转至拼车广场查看订单详情，处理新的拼车申请，车主若接受申请，当前订单状态更新。车主选择确认发车，拼车订单生效。行程结束后，系统进行订单费用结算，车主收取费用，进行最后的行程体验评价。

拼车乘客发起拼车申请，填写拼车需求，系统检测完整性后进行智能匹配，若匹配成功，展示匹配订单；若乘客同意匹配，点击同意即可加入订单；乘客不同意匹配或系统未能匹配订单，乘客跳转至拼车广场自行选择心仪订单，发送申请。待车主通过申请，乘客加入订单。行程结束后，系统进行订单费用结算，乘客支付费用，进行最后的行程体验评价。

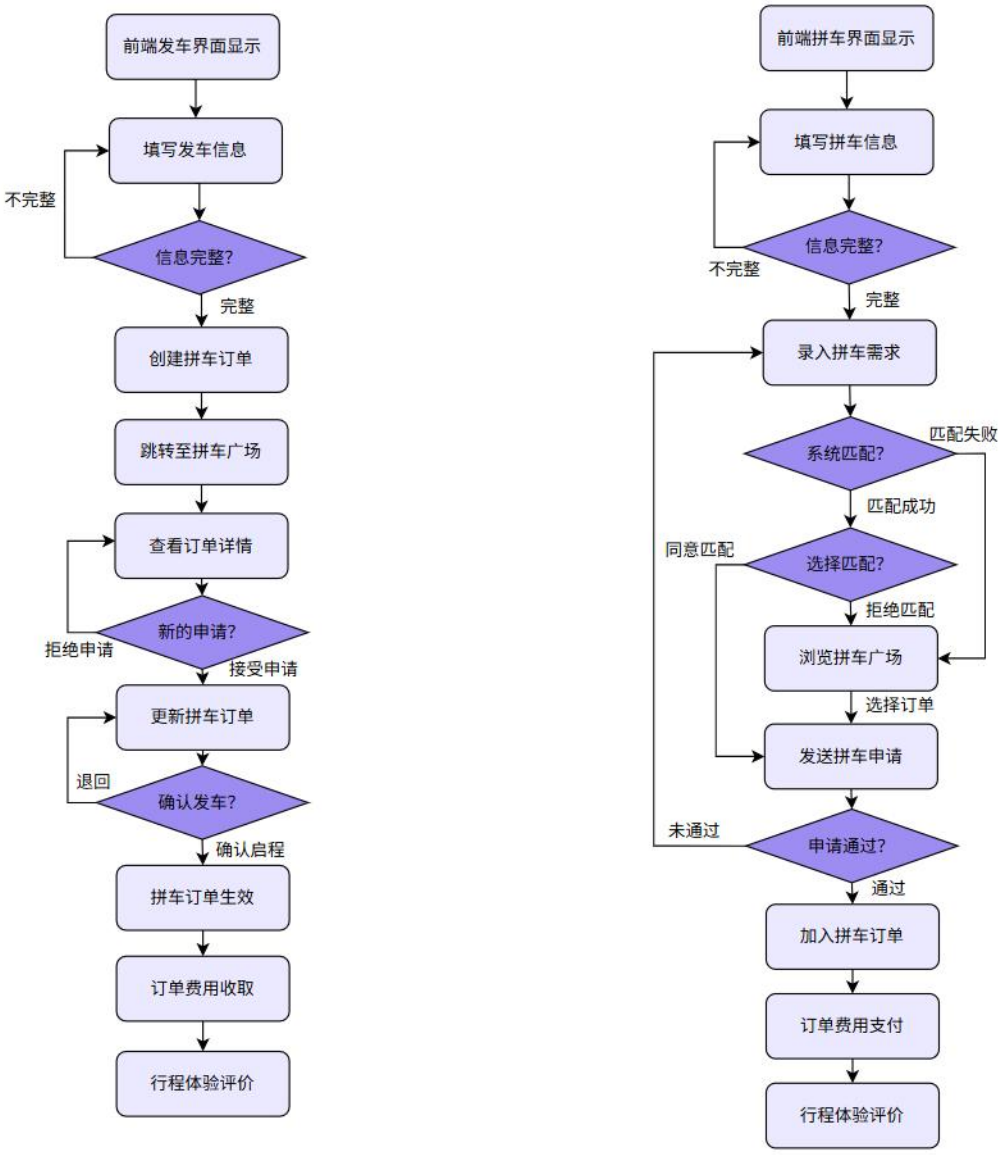

图 5 用户发起拼车（左）、参与拼车（右）流程图

4.3.4 拼车订单群聊

拼车订单群聊模块包含群聊和聊天两个界面。在群聊页面，用户可查看加入群聊的信息，如群聊名称、成员人数、在线人数，同时用户可根据需求进行清除聊天记录和退出群聊的操作。用户点击群聊卡片即可进入该群聊的聊天页面，查看群成员发送的消息记录以及发送的日期和时间，用户可点击输入框填入要发送的内容并发送消息。

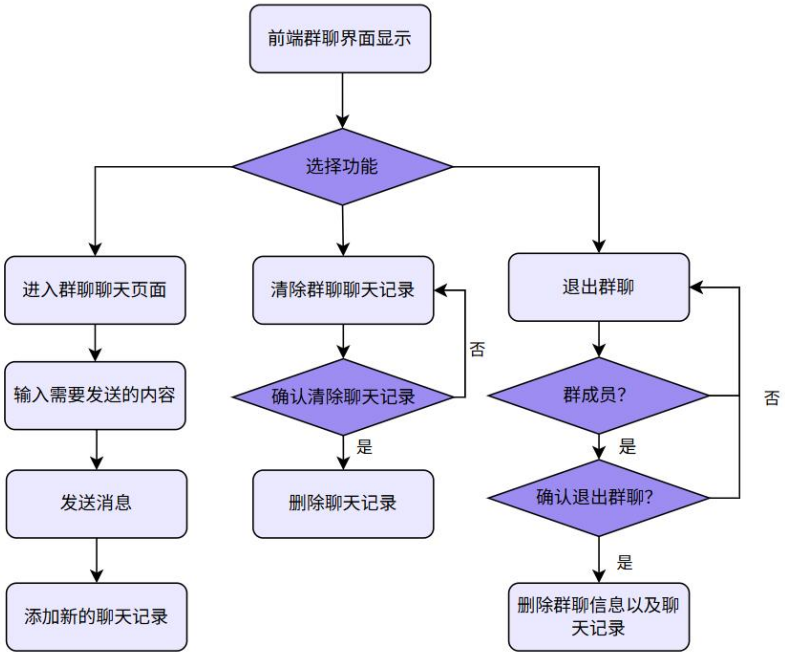

图 6 群聊管理流程图

4.3.5 个人中心

个人中心模块主要包含信息展示和信息编辑两个核心流程。在信息展示流程中，系统首先验证用户登录状态，验证通过获取数据库中的用户信息，包括基本信息和统计数据在界面展示；若验证失败则跳转登录页面，若获取信息失败则显示错误提示。在信息编辑流程中，用户可以修改基本信息或更新头像、驾照图片。当用户提交修改时，系统对文本信息进行格式验证和唯一性检查，对图片进行格式和大小验证，验证通过后更新数据库相应字段并刷新显示；若验证失败则提示具体错误并允许重新编辑。

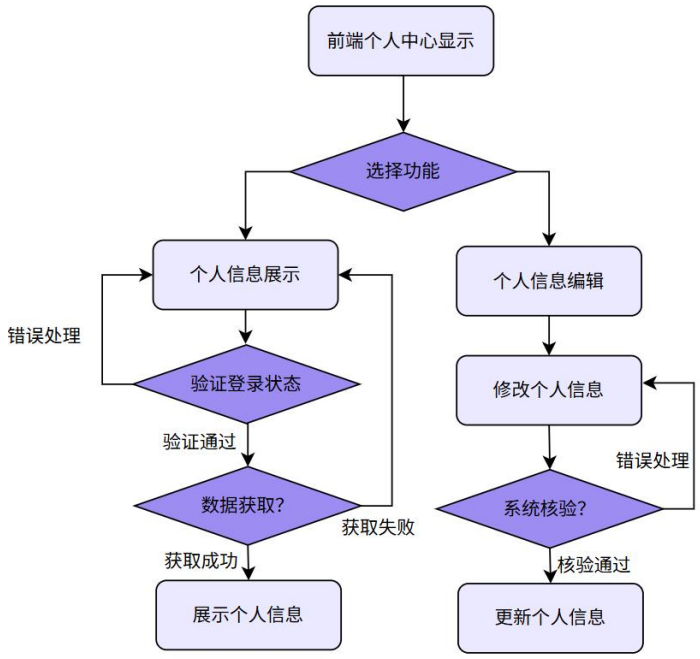

图 7 个人中心信息管理流程图

4.4 接口设计

4.4.1 用户接口

用户接口是指用户通过应用与系统交互的界面，包括前端的界面布局、功能展示、用户输入和反馈等。良好的用户接口设计能够提升用户体验，使操作直观、简单。

（1）登录与注册界面：

- 功能：用户通过邮箱和密码进行注册和登录。
- UI 元素：邮箱输入框、密码输入框、登录按钮、注册按钮、错误提示信息。

（2）拼车发起界面：

- 功能：车主可以发布拼车信息，设置起终点、出发时间、座驾、座位数等。
- UI 元素：地址选择控件、地图选点、时间选择器、座位数设置、座驾选择、费用预估显示、发布按钮。

（3）拼车广场界面：

- 功能：用户可以浏览可用拼车订单，查看订单详情，发起拼车申请。
- UI 元素：订单列表、订单详情卡片（包含路线、时间、费用等信息）、申请按钮、路线展示地图。

（4）个人中心界面：

- 功能：用户可以管理个人信息、查看历史订单、管理座驾信息。
- UI 元素：头像上传、个人信息编辑框、座驾信息管理、订单历史列表、驾照上传控件。

4.4.2 外部接口

（1）高德地图服务接口：

- 功能：提供地图展示、路线规划、距离计算等服务。
- 主要接口：地点搜索：根据关键词搜索地点；路线规划：计算最优行车路线；距离计算：计算两点间行车距离和时间；POI 查询：获取地点详细信息。

（2）支付服务接口：

- 功能：对接主流支付平台，处理拼车费用结算。
- 主要接口：创建支付订单：生成支付订单，返回支付链接；支付状态查询：查询订单支付状态；退款处理：处理用户退款请求；账单查询：查询历史支付记录。

（3）邮箱登录接口：

- 功能：支持用户使用邮箱进行账号注册和登录。
- 主要接口：邮箱验证：发送验证码到用户邮箱；密码重置：通过邮箱验证重置密码；邮箱绑定：绑定或更换登录邮箱。

4.4.3 内部接口

（1）用户模块与订单模块接口：

- 功能：处理用户创建订单、申请拼车等操作
- 主要接口：创建拼车订单：传入起终点、时间、座位数等信息，返回订单 ID；申请拼车：传入订单 ID 和申请人数，创建拼车申请记录；获取订单列表：根据用户 ID 获取相关订单信息。

（2）订单模块与匹配模块接口：

- 功能：处理订单匹配和筛选
- 主要接口：订单匹配：根据用户需求（时间、地点、人数）匹配合适订单；更新订单状态：更新订单的当前状态（等待中、进行中、已结束）；获取匹配结果：返回匹配度排序的订单列表。

（3）用户模块与评价模块接口：

- 功能：处理用户间的互评
- 主要接口：创建评价：提交用户评分和评价内容；获取评价：获取用户的历史评价信息；更新用户评分：根据新评价更新用户的综合评分。

4.5 数据结构设计

4.5.1 数据库表及其字段说明

本系统采用 Python Flask 作为后端开发框架，MySQL 作为数据库管理系统。数据结构设计旨在支持系统的核心功能，包括用户管理、拼车服务、订单管理、评价管理等。通过合理的表结构设计和表间关系的建立，系统能够高效处理大规模数据，并支持未来的功能扩展。

以下是系统的主要数据库表及其字段说明：

（1）用户表（users）：

用于存储已通过注册的用户信息。

主要字段	说明
u_id(INT, Primary Key)	用户 ID
u_name(VARCHAR, 10)	用户昵称
u_password(VARCHAR, 100)	用户密码（哈希存储）
u_tel(VARCHAR, 15)	联系电话
u_mail(VARCHAR, 50)	用户邮箱
u_avatarURL(VARCHAR, 100	用户头像图片路径
u_licenseURL(VARCHAR, 100)	驾照截图路径
u_totalOrders(INT)	总订单数
u_driveCnt(INT)	发起拼车总次数
u_passengerCnt(INT)	参与拼车总次数
u_stars(FLOAT)	用户评分
u_punctualRate(FLOAT)	用户准时度

（2）用户常用地点表（user_{id}fav_addresses）：

每个用户都有独立的常用地点表，用于存储其常用地点信息。

主要字段	说明
id (INT, Primary Key)	地点 ID
name(VARCHAR, 100)	标签名称
address(VARCHAR, 200)	详细地址
poi_id(VARCHAR, 50)	高德唯一 POI ID
lng(FLOAT)	经度
lat(FLOAT)	纬度

（3）座驾信息表（cars）：

用于存储用户的座驾信息。

主要字段	说明

id (INT, Primary Key)	车辆 ID
owner_id(INT, Foreign Key)	车主用户 ID
category Enum("小轿车","吉普车","商务车","面包车","大型车")	车型
plate(VARCHAR, 20)	车牌号
fuelConsumption(FLOAT)	每公里油耗
brand(VARCHAR, 20)	车的品牌

（4）拼车订单表（orders）：

用于记录所有拼车订单信息。

主要字段	说明
id(INT, Primary Key)	订单 ID
driver_id(INT, Foreign Key)	车主用户 ID
car_id(INT, Foreign Key)	车辆 ID
passenger_limit(INT)	拼车上限人数
passenger_cnt(INT)	目前实际拼单人数
passenger_getOn(INT)	已上车人数
planStartTime(DATETIME)	计划发车时间
startTime(DATETIME)	实际发车时间
endTime(DATETIME)	实际结束时间
start_poi_id(VARCHAR, 100)	起点的高德 POI ID
start_address(VARCHAR, 100	起点详细地址
start_lng(FLOAT)	起点经度
start_lat(FLOAT)	起点纬度
end_poi_id(VARCHAR, 100)	终点的高德 POI ID
end_address(VARCHAR, 100)	终点详细地址
end_lng(FLOAT)	终点经度
end_lat(FLOAT)	终点纬度
distance(FLOAT)	总距离
cost(FLOAT)	订单总价
status Enum("waiting","driving","ended")	订单状态

（5）乘客信息表（passengers）：

用于记录订单中的乘客信息。

主要字段	说明
id(INT, Primary Key)	记录 ID
order_id(INT, Foreign Key)	订单 ID
passenger_id(INT, Foreign Key)	乘客用户 ID
passenger_count(INT)	乘客人数
getOn_time(DATETIME)	乘客上车时间

（6）拼车申请表（applies）：

用于记录用户的拼车申请信息。

主要字段	说明
id (INT, Primary Key)	申请 ID
order_id (INT, Foreign Key)	订单 ID
passenger_id (INT, Foreign Key)	乘客用户 ID
driver_id (INT, Foreign Key)	发车人用户 ID
passenger_count (INT)	申请拼车人数
apply_time (DATETIME)	申请时间
approve_time (DATETIME)	审批时间
Status Enum("waiting","approved","rejected","ended")	申请状态

（7）用户评价表（ratings）：

用于记录用户间的评价信息。

主要字段	说明
id (INT, Primary Key)	评价 ID
submitter (INT, Foreign Key)	提交评价用户 ID
rater (INT, Foreign Key)	被评价用户 ID
order_id (INT, Foreign Key)	订单 ID
stars (INT)	评分（1-5 分）

（8）群聊表（chats）：

用于记录拼车群聊的基本信息。

主要字段	说明
id (INT, Primary Key)	群聊 ID
order_id (INT, Foreign Key)	订单 ID
Name (VARCHAR, 100)	群聊名称
member_cnt (INT)	总人数
online_cnt (INT)	在线人数

（9）群聊成员表（chatMembers）：

用于记录拼车群聊的成员信息。

主要字段	说明
id (INT, Primary Key)	群聊成员 ID
chat_id (INT, Foreign Key)	群聊 ID
member_id (INT, Foreign Key)	成员用户 ID
joined_time (DATETIME)	加入时间
is_online (BOOLEAN)	是否在线

role Enum("发起人","参与者")	成员角色

（10）群聊消息表（Message）：

每个群聊创建一张独立的消息表。

主要字段	说明
id (INT, Primary Key)	消息 ID
chat_id (INT, Foreign Key)	群聊 ID
sender_id (INT, Foreign Key)	发送人 ID
sender_role Enum("发起人","参与者")	发送人角色
content (TEXT)	消息内容
sent_at (DATETIME)	发送时间

4.5.2 表间关系

（1）用户表(users)与车辆信息表(cars)：

cars.user_id 是 users.user_id 的外键，表示一个用户可以拥有多辆认证车辆。

（2）用户表(users)与订单表(orders)：

orders.creator_id 是 users.user_id 的外键，表示订单的创建者。

orders.driver_id 是 users.user_id 的外键，表示订单的司机（如果已确定）。

（3）订单表(orders)与乘客表(passengers)：

passengers.order_id 是 orders.order_id 的外键，表示一个订单可以包含多个乘客。

passengers.user_id 是 users.user_id 的外键，表示乘客的用户身份。

（4）订单表(orders)与申请表(applies)：

applies.order_id 是 orders.order_id 的外键，表示用户对某个订单发起的申请。

applies.user_id 是 users.user_id 的外键，表示申请人的用户身份。

（5）用户表(users)与评价表(ratings)：

ratings.from_user_id 是 users.user_id 的外键，表示评价的发起者。

ratings.to_user_id 是 users.user_id 的外键，表示评价的接收者。

ratings.order_id 是 orders.order_id 的外键，表示评价关联的订单。

（6）用户表(users)与群聊表(groups)：

groups.creator_id 是 users.user_id 的外键，表示群聊的创建者。

group_members.group_id 是 groups.group_id 的外键，表示群聊成员关系。

group_members.user_id 是 users.user_id 的外键，表示群聊中的成员。

（7）群聊表(groups)与消息表(messages)：

messages.group_id 是 groups.group_id 的外键，表示消息所属的群聊。

messages.sender_id 是 users.user_id 的外键，表示消息的发送者。

（8）用户表(users)与常用地点表(common_locations)：

common_locations.user_id 是 users.user_id 的外键，表示用户保存的常用地点。

这些数据表之间形成紧密关联的数据关系网络，通过精心设计的表结构和关联关系，全面支持系统的核心业务功能。表结构之间的关联关系不仅确保了数据的完整性和一致性，更为系统的扩展性和可维护性奠定了坚实基础。每个表之间的关系都经过深入考虑，既满足了当前的功能需求，又预留了未来功能扩展的可能性。

五、UML 建模分析

5.1 针对进行 UML 分析建模

为了进一步明确手机私家车拼车软件系统的需求和功能，本项目将结合前期的需求分析内容，进行详细的 UML 分析建模。UML 统一建模语言是软件工程中用于描述、设计和记录软件系统的一种标准化语言，通过 UML 建模，可以形象化地展示系统的结构和行为，为后续的开发和实现提供清晰的蓝图。

在此次 UML 分析建模过程中，我们将从三个核心维度展开分析：功能模型（用例图）、对象模型（类图）以及行为模型（顺序图和状态图）。通过这些模型的建立，我们能够全面地描述系统的功能需求、结构设计和动态行为。

首先，通过用例图展示用户与系统之间的交互，明确用户管理、车辆管理、拼车服务、拼车群聊、评价反馈等模块的功能和流程。用例图帮助我们清晰地展示了作为私家车主和乘客的两类当前用户状态与系统之间的交互关系，以及各个功能模块的业务流程。

其次，通过类图描述系统的静态结构，展示了系统中的核心类及其关系。类图涵盖了用户管理、车辆管理、行程管理、评价管理等核心业务模块，定义了各个类的属性和方法，并通过关联、继承等关系展示了类之间的交互方式，为系统的模块化设计提供了基础。

最后，通过顺序图和状态图描述系统的动态行为。顺序图展示了用户操作的时序过程，如用户登录注册、发起拼车、参与拼车等关键业务流程的交互顺序；状态图则展示了系统中重要对象（如订单、行程）在其生命周期中的状态变化，

帮助我们理解系统的动态特性。

5.2 UML 建模分析

5.2.1 功能模型

手机私家车拼车系统不区分车主和乘客两种身份，用户可任意选择发起拼车/参与拼车，以下用例图是用户在某一时刻选择作为私家车主发起拼车或作为乘客参与拼车场景下进行绘制。

私家车主用例图展示了整个拼车系统中与私家车主相关的核心功能结构。系统以六个主要功能模块构成：用户管理、座驾管理、即时通讯、发起拼车、订单管理和评价系统。在用户管理模块中，系统支持用户完成注册、登录、个人信息管理以及驾照上传等基础功能，确保用户身份的真实性和合法性。座驾管理模块私家车主进行添加新座驾、选择当前使用座驾以及删除不再使用的座驾信息。在发起拼车模块中，私家车主可以发布行程信息，并对乘客的拼车申请进行审核，同时具备开始行程、结束行程等完整的行程管理功能。即时通讯模块为私家车主提供了与乘客沟通的平台，支持加入拼车群聊、查看乘客消息等功能，促进司乘双方的有效沟通。订单管理模块则负责处理行程相关的具体事务，包括费用结算等重要环节。评价系统模块通过行程评价和乘客评价功能，建立起完善的信用体系，促进平台良性发展。这些功能模块相互关联、协同运作，共同构建了一个完整、高效的拼车服务体系，为私家车主提供全方位的系统支持。

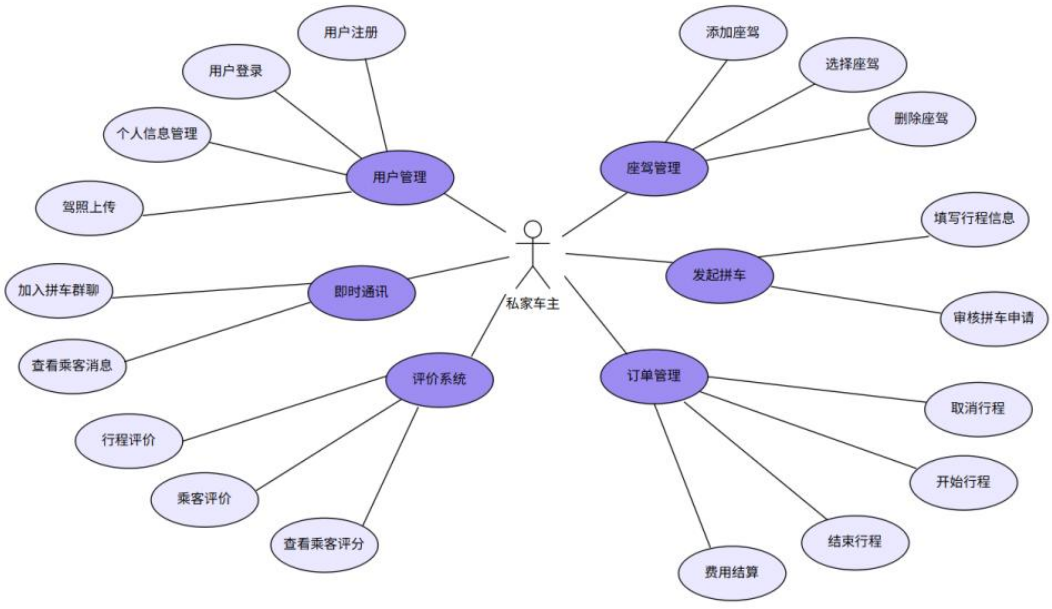

图 8 私家车车主用例图

乘客用例图展示了拼车系统中乘客角色可以进行的所有核心操作和功能。系统为乘客提供了六个主要功能模块，形成了完整的用户体验闭环。在用户管理模块中，乘客可以进行基础的账户操作，包括用户注册、登录以及个人信息管理。拼车广场模块作为乘客寻找合适拼车信息的主要平台，提供了浏览广场和选择订单的功能，方便乘客快速找到符合需求的拼车信息。参与拼车模块则支持乘客提交拼车需求、选择系统匹配的车主，以及提交拼车申请等核心业务功能。订单管理模块负责处理乘客的订单全生命周期，包括加入订单、订单拒绝、结束行程等状态变更，同时还包含费用结算等关键环节。即时通讯模块通过加入拼车群聊和查看司机消息功能，确保乘客与司机之间的顺畅沟通。评价系统模块则允许乘客对完成的行程和车主进行评价，并查看司机的评分情况，建立起完善的信用评价机制。这些功能模块通过有机整合，为乘客提供了一个便捷、安全、高效的拼车服务体验。

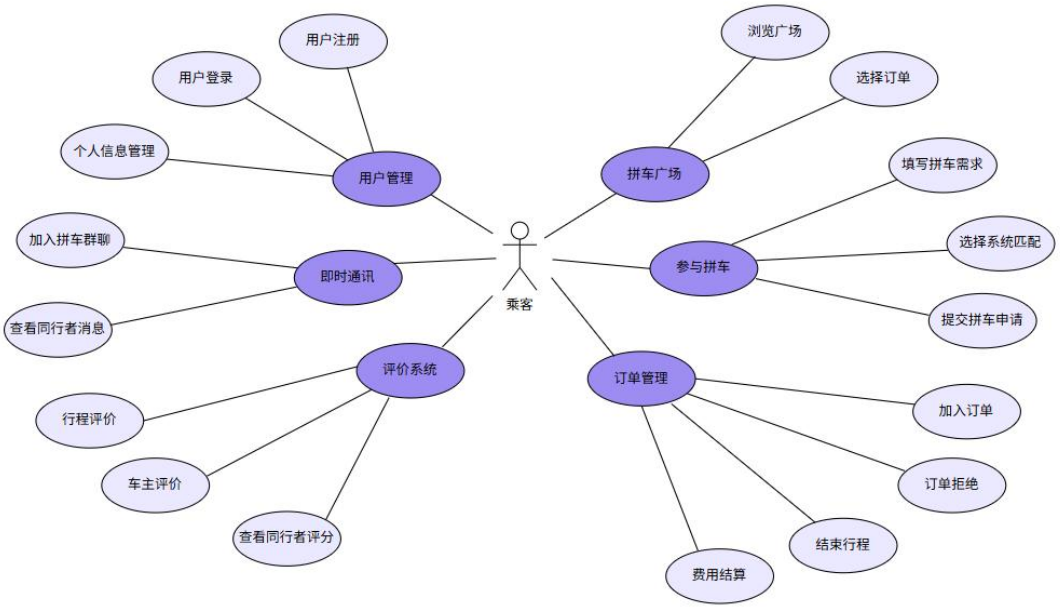

图 9 乘客用例图

5.2.2 对象模型

拼车应用系统的核心对象模型包含六个主要类：用户类作为核心，管理用户基本信息和账户操作；车辆类负责车辆信息管理；订单类处理行程信息及其生命周期；拼车申请类管理乘客的搭车请求；评价类支持用户间的互评机制；聊天类提供即时通讯功能。类之间通过关联关系紧密连接：用户与车辆是一对多关系，用户与订单是一对多关系，订单与拼车申请是一对多关系，订单与聊天是一对一关系，评价则通过双向关联连接用户与订单。这种设计既保证了数据结构的完整

性，又支持了从发布行程到完成订单的完整业务流程。

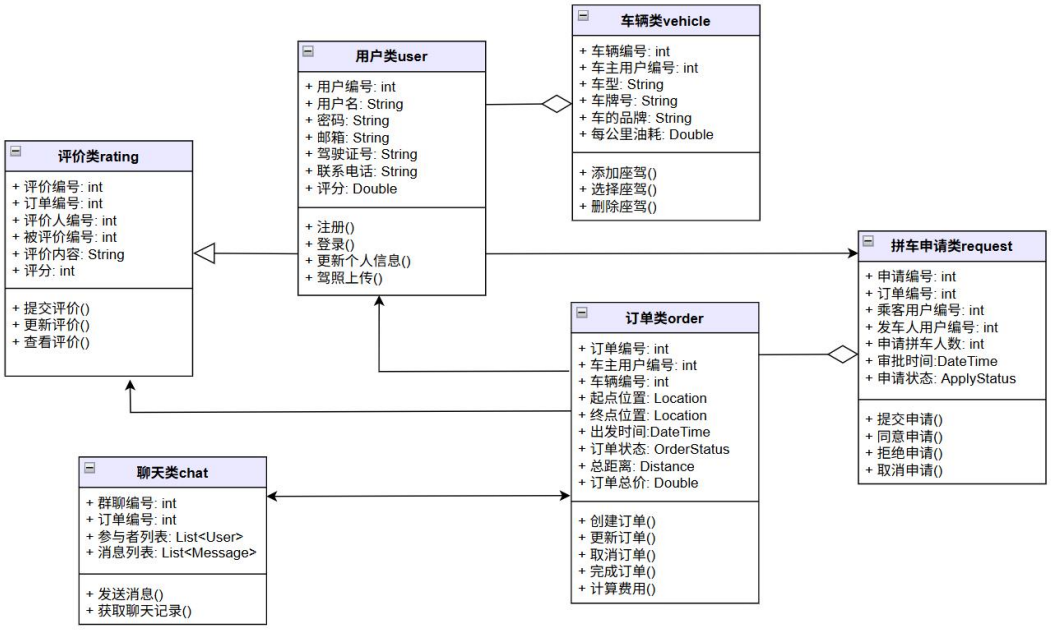

图 10 系统类图

5.2.3 行为模型

（1）用户登录注册：

用户打开软件进入欢迎界面，点击跳转用户登录界面，新用户首次登录点击进行账号注册。注册成功或已注册用户，添加信息登录账号，登录成功，进入拼车系统首页面。

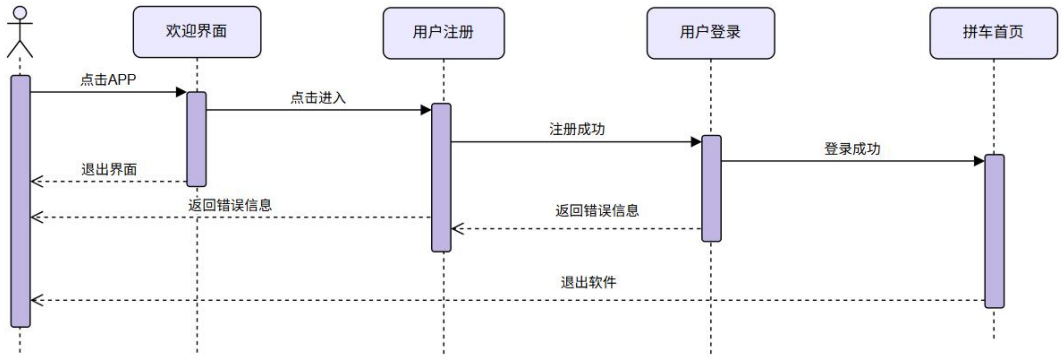

图 11 用户登录注册顺序图

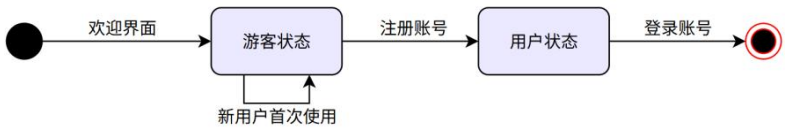

图 12 用户登录注册状态图

（2）用户发起拼车：

私家车主点击主界面发起拼车。首先填写行程信息，创建新的拼车订单，每当新的乘客申请到来时，车主审核申请，若同意申请则订单更新，拒绝申请则订单保持原状。车主点击发车，开始行程，到达目的地后点击按钮，行程结束。

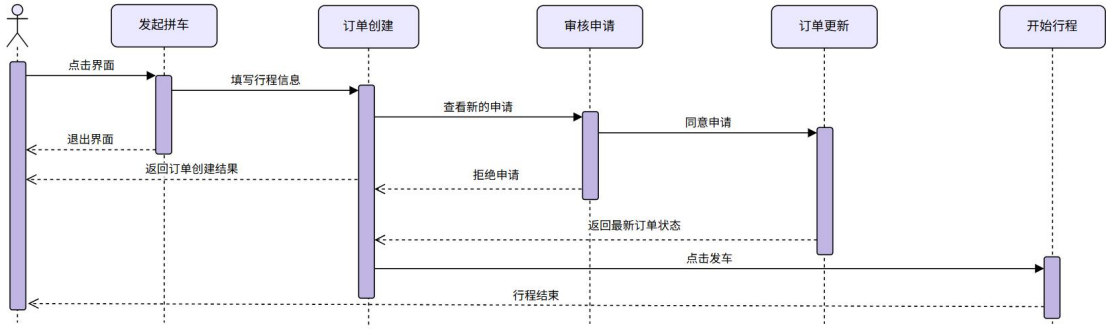

图 13 用户发起拼车顺序图

图 14 用户发起拼车状态图

（3）用户参与拼车：

乘客点击主界面参与拼车。首先填写行程信息，创建新的拼车需求，每当新的乘客需求到来时，系统开始自动匹配，若匹配成功则推送给乘客，乘客有权选择是否接受匹配，若不接受可自行去拼车广场选择心仪订单。乘客同意匹后，向车主方发起加入拼车申请，若车主同意申请，乘客加入订单，到达目的地后行程结束。

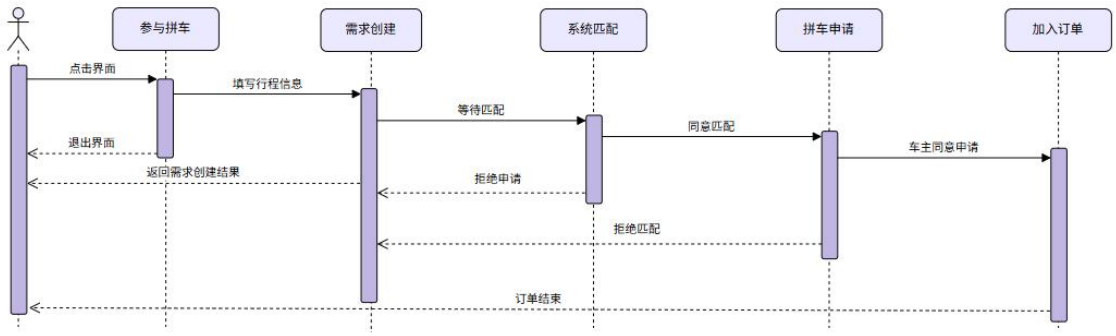

图 15 用户参与拼车顺序图

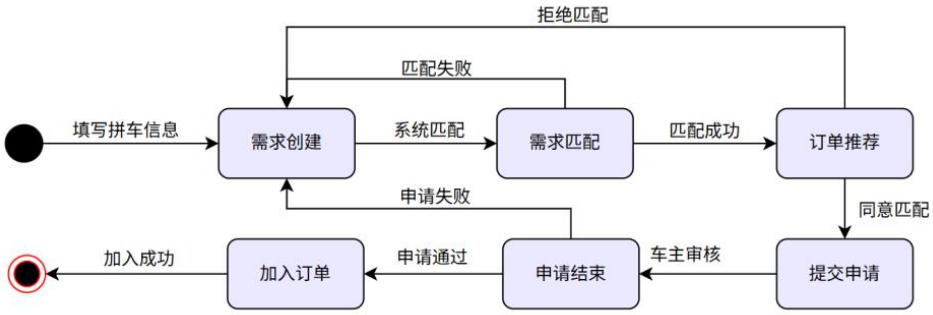

图 16 用户参与拼车状态图

（4）用户订单：

私家车主填写行程信息发布拼车邀请后，拼车订单创建。车主同意申请后，乘客加入该订单。订单中的用户点击进入拼车群聊，系统随时为用户返回群聊最新状态，车主和乘客在群聊中沟通出行事宜。车主点击行程开始，系统行程跟踪，到达目的地时，车主点击行程结束，系统进行费用结算，用户可以在此时选择退出，或者继续评价订单，对司机和同行者做出评分。

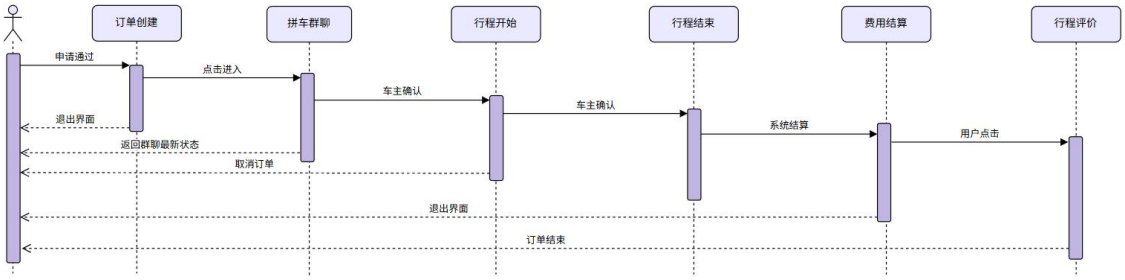

图 17 用户订单顺序图

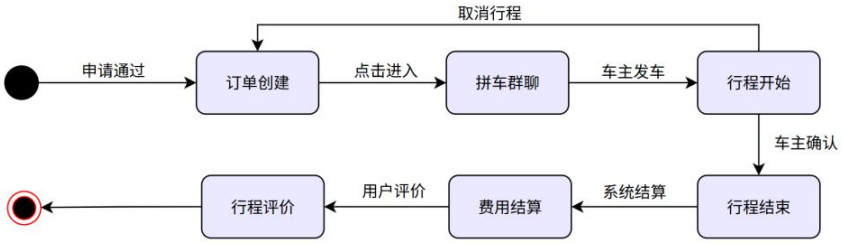

图 18 用户订单状态图

六、详细设计

6.1 系统架构设计

手机私家车拼车系统采用了现代化的前后端分离架构，前端基于 React Native 开发移动应用，后端采用 Python Flask 框架。这种架构设计不仅确保了系统的跨平台兼容性，还提供了优秀的用户体验和系统可扩展性。

前端部分采用 React Native 框架进行开发，这是一个成熟的跨平台移动应用开发框架。从项目结构可以看出，前端采用了模块化的设计思路，主要包括用户认证模块、行程管理模块、车辆管理模块、评价系统模块等。在路由管理方面，使用了 React Navigation 进行页面导航，实现页面的无缝切换。项目使用 TypeScript 作为开发语言，提供了更好的类型检查和代码提示，提高了开发效率和代码质量。

后端系统基于 Flask 框架开发，采用了完整的 MVC 架构设计。数据持久层使用 MySQL 数据库，并通过 SQLAlchemy ORM 框架实现了对象关系映射，使得数据库操作更加直观和安全。数据库设计包含了用户、车辆、订单等核心表，它们之间通过外键关系形成了完整的数据关系网络。在用户认证方面，系统使用 JWT 实现了基于令牌的身份验证机制，确保了接口调用的安全性。同时，系统还集成了邮件服务用于用户验证，以及跨域资源共享支持，方便前后端开发和调试。

在安全性方面，系统实现了多层次的安全保护措施。用户密码采用 SHA256 算法进行加密存储，敏感接口都需要通过 JWT 认证才能访问，同时系统还实现了完整的异常处理机制，能够优雅地处理各种错误情况。

在性能优化方面，系统使用了数据库连接池、合理的索引设计等技术手段，确保了系统的响应速度和稳定性。

通过这种前后端分离的架构设计，系统实现了高可用性、可扩展性和可维护性。前端的模块化设计和后端的微服务架构为未来的功能扩展提供了便利，同时完善的错误处理机制和安全措施确保了系统的稳定运行。这种架构不仅满足了当前的业务需求，也为未来的系统升级和功能扩展预留了充分的空间。

6.2 模块划分

手机私家车拼车系统整体架构被划分为前端模块、后端模块、数据库模块和

服务器模块四个主要部分，各模块的功能和职责如下：

前端模块负责与用户直接交互，处理用户输入并展示相应的界面。基于 React Native 框架开发。前端采用 JavaScript 作为开发语言，提供了更好的类型检查和代码提示。前端模块包含了用户认证功能，实现了登录和注册界面；主页模块作为核心功能的入口；行程管理模块处理发布行程和搜索行程的功能；广场模块展示可用的拼车信息；车辆管理模块用于添加和管理座驾；评价模块处理用户评分和反馈；个人中心模块负责用户信息的管理。通过这些功能模块的组合，为用户提供了完整的应用体验。

后端模块是整个系统的核心，基于 Python Flask 框架开发，负责处理前端请求、执行业务逻辑和数据库交互。后端实现了完整的用户管理功能，包括注册、登录和信息管理；订单管理模块处理订单的创建、匹配和状态更新；车辆管理模块提供车辆信息的增删改查功能；位置服务模块通过集成高德地图 API，提供了地点搜索和路径规划功能；评价系统模块处理用户评分并进行统计；即时通讯模块实现了用户间的实时消息交互；安全认证模块基于 JWT 实现了用户认证和授权。这些模块共同构成了系统的业务处理核心。

数据库模块采用 MySQL 数据库，通过 SQLAlchemy ORM 框架实现数据的持久化存储和管理。数据库设计包含了多个关键表：用户表存储用户基本信息、评分和统计数据；车辆表管理车辆信息；订单表记录拼车订单信息；乘客表记录乘客信息；申请表管理拼车申请；聊天和消息表存储即时通讯数据；评价表记录用户评价数据。数据库的设计注重数据一致性和查询性能，通过合理的索引设计和关系模型确保了数据的高效访问和安全存储。

服务器模块采用 Flask 内置的开发服务器，负责系统后端服务的部署与运行。服务器实现了完整的 HTTP 请求处理机制，支持跨域资源共享，集成了邮件服务器功能，并实现了完整的错误处理机制。服务器模块还支持异步任务处理，使用连接池管理数据库连接，通过这些特性确保了系统的稳定运行和良好的性能表现。服务器配置注重安全性和性能，通过合理的配置参数优化，确保了系统的响应速度和可靠性。

这种模块化的架构设计使得系统各个部分职责明确、耦合度低，便于开发和维护。通过使用现代化的技术栈和框架，确保了系统的可扩展性和性能表现。每个模块都可以独立开发和测试，大大提高了开发效率和系统可靠性。同时，模块间通过标准的接口进行通信，使得系统具有良好的扩展性和维护性，为未来的功能扩展和系统升级提供了坚实的基础。

6.3 模块设计

6.3.1 用户认证模块

（1）模块概述：

用户认证模块是系统的基础功能模块，负责用户的注册和登录管理。该模块采用 JWT（JSON Web Token）认证机制，确保系统安全性。模块设计注重用户体验，提供了邮箱验证功能，并实现了安全的密码存储机制。

（2）界面布局与交互设计：

- **欢迎界面：**

顶部展示应用 LOGO 和“拼途”品牌名称；中部字幕简要介绍应用，欢迎用户的到来；底部设置醒目的进入按钮，跳转进入登录页面。

- **登录界面：**

采用简洁的垂直布局设计。顶部显示系统 LOGO，中部是邮箱和密码输入框，提供新用户提示按钮，点击跳转注册界面；底部是醒目的登录按钮，点击跳转至应用首页界面。

- **注册界面：**

保持与登录界面一致的设计风格。包含邮箱和密码输入字段，提供密码确认功能。提供已注册提示按钮，点击跳转登录界面。底部是醒目的注册按钮，注册成功后自动跳转至登录界面。

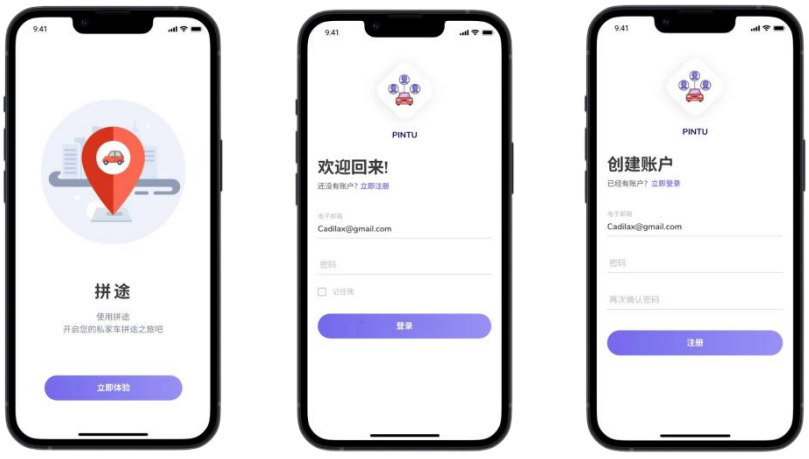

图 19 用户认证原型图

（3）表单字段设计：

用户认证模块主要涉及用户表（User）的设计。

字段名称	设计说明
用户编号	作为主键，采用整型并自动递增，确保每个用户的唯一标识。
用户名	字符长度限制，系统自动生成并确保唯一性。
密码	字符长度限制，存储经过 SHA256 加密后的密码串。
邮箱	字符长度限制，要求唯一性，用于用户登录和身份验证。
联系电话	字符长度限制，同样要求唯一。
头像 URL 和驾照 URL	用于存储相应图片的访问路径。
用户统计信息	包括总订单数、发起和参与拼车次数。采用整型，默认为零。
用户评分和准时率	采用浮点型设计，分别默认为满分和百分之百。

（4）核心功能实现：

用户认证模块的核心功能实现围绕身份验证和安全性展开。在欢迎页面，系统首先检查本地存储的 JWT Token 有效性，根据验证结果决定是否自动跳转到主页面。注册功能通过邮箱验证确保用户身份真实性，系统使用 Flask-Mail 组件发送验证邮件，并通过 SHA256 算法对用户密码进行加密存储。用户名通过 random 模块自动生成，并通过数据库查询确保唯一性。登录功能采用 JWT 认证机制，成功验证后生成包含用户 ID、用户名和邮箱信息的 Token，有效期设置为 2 小时。

系统实现了完整的错误处理机制，包括邮箱重复检查、密码强度验证等。Token 的管理采用 Bearer 认证方案，要求客户端在后续请求中通过 Authorization 头部携带 Token。模块还实现了会话管理和自动刷新 Token 机制，确保用户在使用过程中的身份认证持续有效。所有的数据库操作都通过 SQLAlchemy ORM 框架实现，确保数据操作的安全性和一致性。

6.3.2 发起拼车模块

（1）模块概述：

发起拼车模块是系统的核心业务模块，负责处理拼车行程的发布和搜索功能。该模块集成了高德地图 API，提供精确的位置服务和路径规划功能，实现了智能的拼车匹配算法。模块设计注重用户体验，通过直观的界面展示和便捷的操作流程，帮助用户快速发布和查找合适的拼车行程。

（2）界面布局与交互设计：

- **发布行程界面：**

采用表单式布局，主体显示地图场景，清晰直观。在地点选择完毕后，显示

路径规划结果；下部为填写发车信息的表单，包含起点和终点选择框、发车时间选择器、座位数量设置、座驾选择字段，底部为发起拼车按钮。每个输入项配有清晰的图标和提示文字，帮助用户理解和填写。

- **搜索行程界面：**

集成地图组件，顶部为搜索栏，支持起终点位置搜索。右部提示按钮进行常用地点的选择和删除。

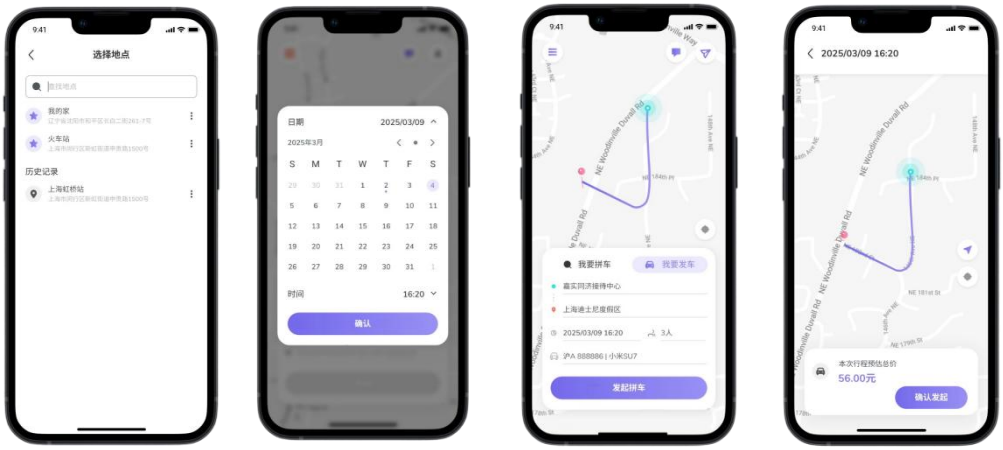

图 20 发起拼车原型图

（3）表单字段设计：

行程管理模块主要涉及订单表的设计。

字段名称	设计说明
订单编号	作为主键，采用整型自增设计。
司机编号和车辆编号	作为外键，分别关联用户表和车辆表。
乘客限制和当前乘客数	采用整型设计，用于控制拼车人数。
时间相关	包括计划发车时间、实际发车和结束时间，均采用 DateTime 类型。
位置信息	包括起终点 POI 编号，详细地址，均限制字符长度；经纬度信息采用浮点型存储。
距离和费用	使用浮点型设计。
订单状态	使用枚举类型，包含"waiting"、"driving"、"ended"三种状态。

（4）核心功能实现：

发起拼车模块的核心功能实现围绕位置服务和创建订单展开。在发布行程时，系统通过高德地图 API 的输入提示接口实现地点搜索功能，支持关键词搜索和

周边位置推荐。路径规划功能通过高德地图的驾车路径规划接口实现，可获取详细的行驶路线、距离和预计时间。

订单创建过程中，系统自动关联用户信息和车辆信息，计算预估费用，并更新用户的发车统计数据。搜索匹配功能实现了基于时间和位置的智能匹配算法，考虑发车时间前后 30 分钟范围内的订单，并计算起终点位置的匹配程度，按照匹配分数对结果进行排序。

6.3.3 参与拼车模块

（1）模块概述：

参与拼车模块是系统为乘客用户提供的核心功能模块，包含拼车申请、系统智能匹配和拼车广场浏览等功能。该模块通过智能匹配算法帮助用户快速找到合适的拼车行程，并提供直观的广场浏览界面，让用户可以自主选择心仪的拼车订单。模块设计注重用户体验和匹配效率，实现了便捷的拼车申请流程。

（2）界面布局与交互设计：

- **拼车申请界面：**

与发起拼车界面保持一致。采用表单式布局，主体显示地图场景，清晰直观。在地点选择完毕后，显示路径规划结果；下部为填写发车信息的表单，包含起点和终点选择框、发车时间选择器、参与拼车人数设置、理想价格选择字段，底部为参与拼车按钮。每个输入项配有清晰的图标和提示文字，帮助用户理解和填写。

- **系统匹配界面：**

集成了地图组件，顶部为起终点输入框，中部显示地图和推荐路线，底部展示匹配度最高的拼车订单列表。

- **拼车广场界面：**

采用信息流布局，顶部为搜索栏和筛选选项，主体部分以卡片形式展示可用的拼车信息，每张卡片包括起终点、时间、价格、车主信息等关键数据，支持下拉刷新和上拉加载更多。

图 21 参与拼车原型图

（3）表单字段设计：

参与拼车模块主要涉及申请表和乘客表的设计。

申请表设计如下：

字段名称	设计说明
申请编号	作为主键，采用整型自增设计。
订单编号、乘客编号和司机编号	作为外键，关联相应的订单和用户信息。
申请人数	采用整型设计，默认为 1。
申请时间和审批时间	使用 DateTime 类型，其中申请时间默认为当前时间。
申请状态	使用枚举类型，包含"waiting"、"approved"、"rejected"、"ended"四种状态。

乘客表设计如下：

字段名称	设计说明
乘客编号	作为主键，采用整型自增设计。
订单编号	作为外键，关联订单和用户信息。
乘客人数	采用整型设计，默认为 1。
上车时间	使用 DateTime 类型，用于记录实际上车时间和计算准时率。

（4）核心功能设计：

参与拼车模块的核心功能实现围绕需求填写、订单匹配和广场浏览三个方面

展开。在需求填写环节，系统通过高德地图 API 的输入提示接口实现地点搜索和选择功能，用户可以通过关键词搜索或从常用地址中选择起终点位置。时间选择支持日期和具体时间的设定，系统会根据选择的时间范围进行后续的订单匹配。乘车人数的设置会影响最终的费用计算和座位匹配。

订单匹配功能采用了多维度的智能匹配算法。系统首先在时间维度上筛选目标时间前后 30 分钟范围内的订单，然后从空间维度计算匹配程度：完全匹配、位置相近、单点匹配得分依次递减。系统会按照匹配分数对结果进行排序，并结合价格范围进行二次筛选，最终向用户推荐最适合的拼车订单。

拼车广场浏览功能实现了订单信息的实时展示和智能排序。系统会自动过滤已过期的订单，只展示状态为"waiting"的有效订单。每个订单卡片展示关键信息，包括起终点地址、发车时间、每人费用、车主信息等。系统支持按发车时间排序，并提供了订单详情的快速查看功能。在订单详情页面，用户可以查看完整的行程信息、车主历史订单数据和评价信息，帮助用户做出更好的选择。所有数据的获取和展示都通过异步加载实现，确保界面响应的流畅性，同时系统会定期更新订单状态，保证信息的实时性。

6.3.4 订单管理模块

（1）模块概述：

订单管理模块是系统的核心业务流程模块，负责管理整个拼车行程的生命周期，包括乘客申请、车主审核、行程开始和结束等完整流程。该模块通过清晰的状态管理和交互设计，确保车主和乘客能够顺畅地完成整个拼车过程。模块设计注重实时性和可靠性，实现了完整的订单状态追踪和用户互动机制。

（2）界面布局与交互设计：

- **乘客申请界面：**

采用列表式布局，以时间倒序展示用户的所有拼车申请记录。每条记录以卡片形式展示，包含起终点信息、申请日期时间和当前状态（申请中、已通过、未通过、已结束）。卡片采用不同的状态信息颜色标识，帮助用户快速识别申请进展。用户可以通过点击卡片查看详细信息。

车主行程界面根据行程状态分为三个阶段：

- **行程开始前界面：**

采用乘客管理布局。顶部显示行程卡片，包含起终点、时间、价格等关键信息；中部为乘客列表区域，分为“待上车”和“已上车”两个部分，每个乘客卡片显示头像，支持点击确认上车；底部为发车按钮，根据上车状态动态激活。

- **行程进行中界面：**

采用实时状态展示布局，顶部显示行程卡片；中部嵌入地图组件显示实时位置和路线；下部显示所有乘客信息和温馨提示。

➢ **行程结束界面：**

顶部显示行程数据；中部显示所有乘客信息；底部为结束行程按钮。

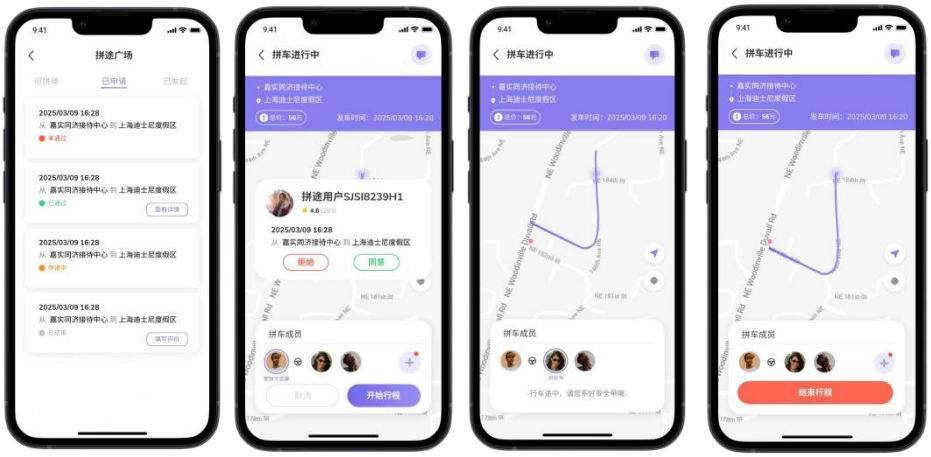

图 22 订单管理原型图

（3）表单字段设计：

行程管理模块主要涉及申请表和订单表的设计。

申请表字段设计如下：

字段名称	设计说明
申请编号	作为主键，采用整型自增设计。
订单编号、乘客编号和司机编号	作为外键，关联相应的订单和用户信息。
申请人数	采用整型设计，默认为 1。
申请时间和审批时间	使用 DateTime 类型。
申请状态	使用枚举类型，包含"waiting"、"approved"、"rejected"、"ended"四种状态。

订单表字段设计如下：

字段名称	设计说明
订单状态	使用枚举类型，包含"waiting"、"driving"、"ended"三种状态。
乘客相关	包括限制人数、实际人数和已上车人数。
时间相关	包括计划发车时间、实际发车时间和结束时间。

位置信息	包括起终点的 POI 编号、详细地址和经纬度信息。

（4）核心功能实现：

行程管理模块的核心功能实现围绕状态管理和实时更新展开。

在乘客申请记录管理中，系统通过关联查询实现申请历史的展示，包括订单基本信息和申请状态。系统会定期更新申请状态，确保用户能够及时了解申请进展。对于已通过的申请，系统自动创建乘客记录，更新订单的实际乘客数量。

在车主行程管理中，系统实现了完整的行程状态流转机制。行程开始前，系统支持车主逐个确认乘客上车状态，记录上车时间并计算准时率。系统会验证所有已通过审核的乘客是否全部上车，只有在满足条件时才允许开始行程。行程开始后，系统记录实际发车时间，更新订单状态为"driving"，并开始计时。行程进行中，系统持续更新行驶时间和距离信息。行程结束时，系统自动计算总时长和油耗（基于车辆的油耗系数和总距离），更新订单状态为"ended"，同时将所有相关申请状态更新为"ended"。系统还会更新所有参与者的统计数据，包括订单总数、准时率等。所有状态变更都通过事务管理确保数据一致性，并实现了完整的异常处理机制。

整个过程中，系统通过状态管理和数据同步确保所有参与者能够实时了解行程状态的变化。模块采用合理的数据缓存策略和查询优化，确保了在处理大量并发订单时的系统性能和响应速度。同时，通过完善的错误处理机制，保证了行程管理过程的稳定性和可靠性。

6.3.5 座驾管理模块

（1）模块概述：

座驾管理模块是面向私家车主的重要功能模块，负责管理用户的车辆信息。该模块提供车辆信息的添加、删除和选择功能，为拼车行程的发起提供必要的车辆支持。模块设计注重数据的准确性和操作的便捷性，通过简洁的界面和清晰的流程，帮助用户高效管理自己的座驾信息。

（2）界面布局与交互设计：

- **我的座驾界面：**

采用列表式布局，顶部显示"我的座驾"标题和添加按钮。主体部分以卡片形式展示用户的所有车辆信息，每张卡片包含车型图标、车牌号、品牌和油耗信息。每张卡片右侧设有删除按钮，点击触发删除确认弹窗。界面支持下拉刷新，保证数据的实时性。

➢ **添加座驾界面：**

采用表单式布局，顶部为导航栏，显示"添加座驾"标题；主体部分为车辆信息输入表单，包含车型选择（下拉选择框，提供"小轿车"、"吉普车"、"商务车"、"面包车"、"大型车"等选项）、车牌号输入框、每公里油耗输入框和品牌输入框。每个输入项配有清晰的标签和提示文字，底部为提交按钮。

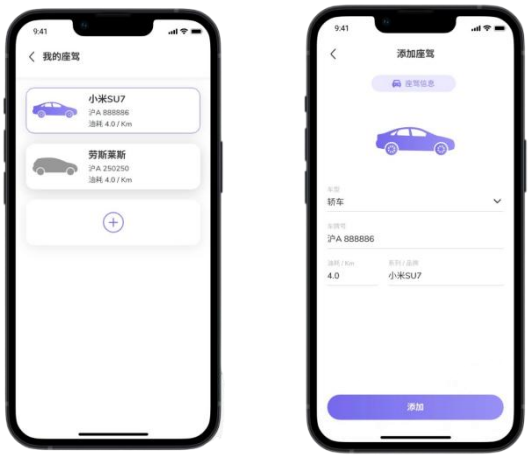

图 23 座驾管理原型图

（3） 表单字段设计：

座驾管理模块主要涉及车辆表的设计。

字段名称	设计说明
车辆编号	作为主键，采用整型自增设计。
车主编号	作为外键，关联用户表的用户编号。
车型	使用枚举类型，包含"小轿车"、"吉普车"、"商务车"、"面包车"、"大型车"五种类型，默认为"小轿车"。
车牌号	设计为限制长度的字符串，要求唯一性。
百公里油耗	采用浮点型设计，用于计算行程费用。
品牌	设计为限制长度的可选字段，用于记录车辆品牌信息。

（4） 核心功能实现：

座驾管理模块的核心功能实现围绕车辆信息的增删查展开。

在添加座驾功能中，系统首先对输入数据进行验证，包括车牌号格式检查、油耗数值验证和车牌号唯一性检查。验证通过后，系统创建新的车辆记录，关联当前用户编号。如果发现重复的车牌号，系统会返回适当的错误提示。

座驾列表功能通过查询当前用户编号关联的所有车辆记录实现，按添加时间倒序排列。系统支持分页加载，确保在车辆数量较多时的展示效率。删除功能实现了权限验证机制，确保用户只能删除自己的车辆信息。删除操作通过事务管理确保数据一致性，同时系统会检查该车辆是否关联了正在进行的订单，如果存在则不允许删除。

在发起拼车时的座驾选择功能中，系统会预先加载用户的所有可用车辆信息，并缓存选择结果。选中的车辆信息会自动关联到订单创建过程中，其油耗数据用于计算预估费用（基于规划路线的总距离）。系统还实现了车辆信息的本地缓存机制，减少重复请求，提升用户体验。

模块采用统一的错误处理机制，对各种异常情况提供友好的用户提示。同时，通过合理的缓存策略和查询优化，确保了模块在高并发场景下的性能表现。

6.3.6 个人中心模块

（1） 模块概述：

个人中心模块是用户管理个人信息和查看使用数据的核心功能区域。该模块提供用户基本信息的展示和编辑功能。模块设计注重信息的完整性和隐私保护，通过直观的数据展示和便捷的信息管理，帮助用户了解自己在平台的使用情况并维护个人资料。

（2） 界面布局与交互设计：

- **个人主页界面：**

采用分区块的布局设计，顶部为用户基本信息区，展示头像、用户名；中部为数据统计区，展示用户的订单总数、发起拼车次数、参与拼车次数、评分指数等统计数据；底部为详细信息区，包含联系方式、驾照信息等个人资料。界面整体采用下拉刷新机制。

- **信息编辑界面：**

采用分类表单式布局，顶部为导航栏，显示"编辑资料"标题。主体部分按照信息类别分为基本信息、联系方式、认证信息三个部分。每个部分包含相应的输入表单，配有清晰的标签和输入提示。头像和驾照图片支持预览和更新功能。底部为保存按钮。

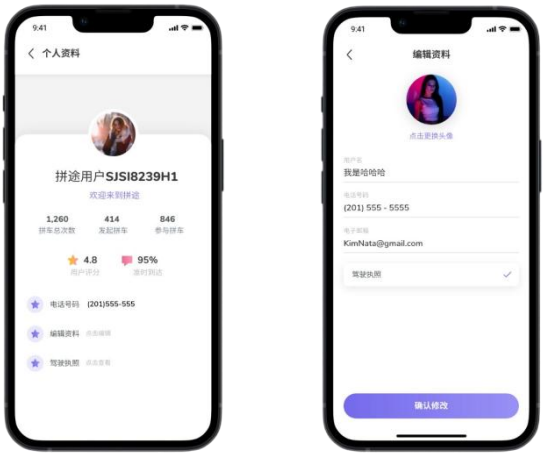

图 24 个人中心原型图

（3） 表单字段设计：

个人中心模块主要基于用户表设计。

字段名称	设计说明
用户编号	作为主键，采用整型自增设计。
用户名	字符串，限制字符长度，唯一。
密码	字符串，限制字符长度。
联系电话	字符串，限制字符长度，唯一。
邮箱	字符串，限制字符长度，唯一。
订单总数	整型，默认为 0。
发起和参与拼车总次数	整型，默认为 0。
用户评分	浮点型，默认为满分 5 分。
准时率	浮点型，默认为 100%。

（4） 核心功能实现：

个人中心模块的核心功能实现围绕用户信息的展示、编辑和统计数据的计算展开。

信息展示功能通过用户认证机制获取当前登录用户编号，查询并返回用户完整信息。系统对敏感信息（如联系方式）进行脱敏处理，确保信息安全。统计数据的展示通过实时计算用户的各项统计指标实现，包括订单总数、发起和参与拼车次数、评分和准时率等。

信息编辑功能实现了完整的数据验证机制，包括必填字段检查、格式验证（如

电话号码、邮箱格式）和唯一性检查（防止用户名、手机号、邮箱重复）。系统支持单个字段的更新，避免不必要的全量更新。头像和驾照图片的更新采用异步上传机制，支持图片预览和裁剪功能。评分和准时率的计算采用加权平均算法，确保数据的准确性和公平性。系统实现了订单完成后的自动统计功能，及时更新用户的各项统计数据。模块还包含数据缓存机制，减少频繁的数据库查询，提升访问性能。

整个模块采用统一的错误处理机制，对各类异常情况提供友好的用户提示。同时，通过合理的权限控制确保用户只能访问和修改自己的信息。模块设计考虑了未来的扩展性，预留了增加新的个人信息字段和统计指标的接口。

6.3.7 拼车群聊模块

（1） 模块概述：

拼车群聊模块是行程协同的核心通信模块，负责为同一订单的车主与乘客提供实时沟通渠道。该模块基于订单动态创建封闭式聊天群组，支持用户间实时交流，解决行程中的即时协调问题（如上车点变更、延误通知等），保障了拼车场景的沟通效率。在本模块中，每当司机成功发起一单订单，就将创建一个对应于该订单的群聊。此后，每当司机同意新乘客的加入申请，该乘客都将加入本订单对应的群聊。此外，群聊模块还应具备聊天基本的清除历史记录、退出群聊等基本功能。

（2） 界面布局与交互设计：

- **群聊界面：**

采用列表式布局，顶部显示"我的群聊"标题。主体部分以卡片形式展示用户加入的所有群聊信息，每张卡片包含群聊图标、群聊名称、成员人数和在线人数等信息。每张卡片右侧设有“更多操作”按钮，点击弹出清除历史记录和退出群聊选项卡片。

- **聊天界面：**

采用气泡布局，顶部显示群聊图标、群聊名称、在线人数信息。主体部分以气泡形式自底向上展示用户所发送的消息内容，气泡上面展示发送消息的用户的用户名和头像，气泡下面展示发送时间，用户发送的消息从右向左展示，其他人发送的消息从左向右展示。页面底部设有输入框和发送按钮，支持用户输入要发送的内容并发送到群聊。

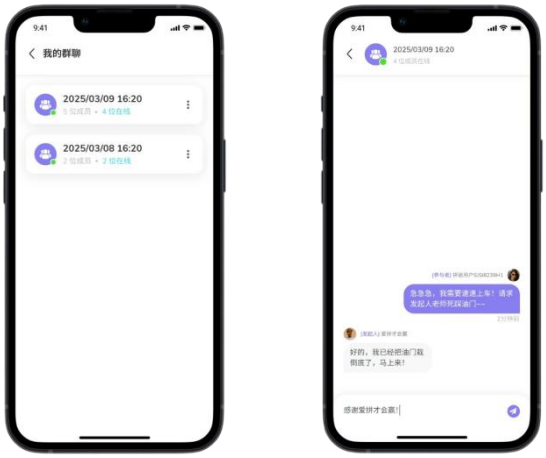

图 25 拼车群聊原型图

（3） 表单字段设计：

群聊模块主要涉及群聊表、群成员表、群聊消息表的设计。

群聊表设计如下：

字段名称	设计说明
群聊编号	作为主键，采用整型自增设计。
群聊名称	字符串，限制字符长度，唯一。
成员人数	整型，默认为 0
在线人数	整型，默认为 0。

群成员表设计如下：

字段名称	设计说明
编号	作为主键，采用整型自增设计。
群聊编号	作为外键，关联群聊表的群聊编号
成员编号	作为外键，关联用户表的用户编号
加入时间	date 类型，不为空
是否在线	bool 类型，在线为 1，反之为 0
成员角色	使用枚举类型，包含"发起人"、"参与者"两种类型

群聊消息表设计如下：

字段名称	设计说明
消息编号	作为主键，采用整型自增设计。

群聊编号	作为外键，关联群聊表的群聊编号
发送者编号	作为外键，关联用户表的用户编号
发送者角色	作为外键，关联群成员表的成员角色
内容	text 类型，不为空
发送时间	date 类型，不为空

（4） 核心功能实现：

拼车群聊模块的核心功能实现围绕群聊和消息的增删展开。

车主发布订单后，系统自动获取订单信息创建群聊，并设车主为群主，若有用户参与拼单并得到车主的同意后，系统自动将用户拉进群聊。

用户群聊列表功能首先会获取用户的信息，若未获取到用户信息，则报错返回到登陆页面，反之则通过查询当前用户关联的群聊信息，按照加入时间倒序排列。系统支持页面滚动，确保在群聊较多时也可充分展示群聊信息。清除历史记录功能可删除用户所记录的该群聊历史消息信息。退出群聊功能实现了权限验证机制，确保用户只能退出自己所加入的群聊。

发送消息功能会验证用户的信息，确认是否是群成员，若不是，则发送消息失败，反之则通过存储消息内容，记录发送时间、发送人信息、群聊信息实现消息记录的保存。

聊天消息展示功能通过查询与群聊相关的消息记录，按照发送时间正序排列，同时系统会验证消息的发送者和账号用户是否一致，若一致则表明是该账号发送的信息，将消息自右向左展示，反之则自左向右展示，以此来区分自己和其他人发送的消息。系统支持页面滚动，确保可充分展示历史聊天记录。

6.4 API 接口

本系统采用 RESTful API 设计风格，主要包含以下接口模块：

6.4.1 用户认证接口

接口名称：**用户注册**	API 路径：**/users/register**	请求类型：**POST**
功能	新用户注册账号	
请求参数	邮箱、密码	
响应	注册成功状态及提示信息	

接口名称：**用户登录**	API 路径：**/users/login**	请求类型：**POST**
功能	用户登录并获取认证令牌	
请求参数	邮箱、密码	
响应	登录状态、认证令牌	

6.4.2 个人中心接口

接口名称：**获取用户信息**	API 路径：**/users/info**	请求类型：**POST**
功能	获取用户基本信息和统计数据	
请求参数	用户 ID（可选，默认获取当前登录用户）	
响应	用户详细信息（包括基本信息、统计数据）	

接口名称：**更新用户信息**	API 路径：**/users/edit**	请求类型：**POST**
功能	修改用户基本信息	
请求参数	用户名、电话、邮箱（可选）	
响应	更新状态及提示信息	

6.4.3 座驾管理接口

接口名称：**添加座驾**	API 路径：**/cars/add**	请求类型：**POST**
功能	添加新的车辆座驾信息	
请求参数	车型、车牌号、油耗、品牌	
响应	添加状态及提示信息	

接口名称：**获取座驾列表**	API 路径：**/cars/list**	请求类型：**GET**
功能	获取用户的所有车辆信息	
请求参数	携带身份令牌访问	
响应	车辆信息列表	

接口名称：**删除座驾**	API 路径：**/cars/delete/{car_id}**	请求类型：**DELETE**
功能	删除指定车辆信息	
请求参数	车辆 ID	
响应	删除状态及提示信息	

6.4.4 订单管理接口

接口名称：**创建订单**	API 路径：**/orders/create**	请求类型：**POST**
功能	创建新的拼车订单	

请求参数	车辆 ID、乘客限制、计划时间、起终点信息、距离、费用
响应	创建状态及提示信息

接口名称：**订单匹配**	API 路径：**/orders/match**	请求类型：**POST**
功能	根据条件匹配合适的拼车订单	
请求参数	起终点位置、乘客数、价格范围、出发时间	
响应	匹配到的订单列表	

接口名称：**获取订单信息**	API 路径：**/orders/fetch**	请求类型：**POST**
功能	获取订单列表及详细信息	
请求参数	订单状态、订单 ID 列表	
响应	订单详细信息列表	

接口名称：**获取已申请订单**	API 路径：**/users/fetch_applied**	请求类型：**GET**
功能	获取用户作为乘客申请的订单列表	
请求参数	携带身份令牌访问	
响应	申请订单列表及状态	

接口名称：**获取已创建订单**	API 路径：**/orders/fetch_created**	请求类型：**GET**
功能	新用户注册账号	
请求参数	携带身份令牌访问	
响应	创建订单列表及状态	

6.4.5 订单操作接口

接口名称：**申请拼车**	API 路径：**/orders/apply/{order_id}**	请求类型：**POST**
功能	乘客申请加入拼车订单	
请求参数	订单 ID、乘客数量	
响应	申请状态及提示信息	

接口名称：**审批申请**	API 路径：**/orders/{order_id}/applies/{user_id}/approve**	请求类型：**POST**
功能	司机审批乘客的拼车申请	
请求参数	订单 ID、申请用户 ID	
响应	审批状态及提示信息	

接口名称：**确认上车**	API 路径：**/orders/{order_id}/passengers/{user_id}/get_on**	请求类型：**POST**
功能	确认乘客上车状态	
请求参数	订单 ID、乘客 ID	
响应	确认状态及提示信息	

接口名称：**开始行程**	API 路径：**/orders/{order_id}/drive**	请求类型：**POST**
功能	司机确认开始行程	
请求参数	订单 ID	
响应	行程状态及提示信息	

接口名称：**结束行程**	API 路径：**/orders/{order_id}/end**	请求类型：**POST**
功能	新用户注册账号	
请求参数	邮箱、密码	
响应	注册成功状态及提示信息	

6.4.6 评价接口

接口名称：**获取评价信息**	API 路径：**/orders/{order_id}/review**	请求类型：**POST**
功能	获取订单相关的评价信息	
请求参数	订单 ID	
响应	订单评价详情及成员信息	

接口名称：**提交评价**	API 路径：**/orders/{order_id}/rate/{u_id}**	请求类型：**POST**
功能	对订单中的其他成员进行评价	
请求参数	订单 ID、被评价用户 ID、评级打分	
响应	订单状态及提示信息	

6.4.7 地图服务接口

接口名称：**地点搜索**	API 路径：**/search/AutoComplete**	请求类型：**POST**
功能	搜索地点信息并给出模糊搜索提示	
请求参数	关键词、当前位置坐标	
响应	高德 API 匹配的地点列表	

接口名称：**路径规划**	API 路径：**/route_planning**	请求类型：**POST**

功能	规划起终点间的行驶路径
请求参数	起终点经纬度坐标及 POI 信息
响应	路径详情、距离、时间、价格估算

所有接口都需要进行用户认证（除注册登录接口外），通过请求头中的 Authorization 字段携带 JWT 令牌实现。接口返回统一的 JSON 格式，包含状态码、消息提示和数据内容。错误处理采用 HTTP 标准状态码，并附带详细的错误信息。

七、编码规范

7.1 命名规则

7.1.1 项目整体命名

- 项目名称：PinTu-APP
- 命名规则：采用大写字母开头，中间使用连字符分隔

7.1.2 目录命名

项目的目录结构采用功能模块化的组织方式：

```
PinTu-APP
|
|--src
|  |--screens
|  |  |--component
|  |  |--profile.js
|  |  |--register.js
|  |  |--review.js
|  |  |--search.js
|  |  |--square.js
|  |  |--square_detail.js
|  |  |--welcome.js
|  |  |--add_vehicle.js
|  |  |--edit_profile.js
|  |  |--home.js
|  |  |--login.js
|  |  |--my_vehicle.js
|  |  |--my_groups.js
```

```
|  |  |--group_chat.js
|  |
|  |--assets
|     |--avatar.png
|     |--icon-back.png
|     |--Icon-Chat.png
|     |--icon-profile-rating.png
|     |--icon-profile-star.png
|     |--logo.png
|     |--......
|
|--post-end.py
|--requirements.txt
```

项目主要包含：

- **前端部分（src 目录）：**
  - screens 目录：包含所有页面组件;
  - assets 目录：包含所有静态资源文件;
- **后端部分：**
  - post-end.py：主程序文件;
  - requirements.txt：Python 依赖配置文件;

这种目录组织方式清晰地展示了项目的结构，便于理解项目的组成部分和文件组织方式。

7.1.3 文件命名

文件命名遵循其所属技术栈的最佳实践：前端文件采用小写字母命名，多词使用下划线连接。

- 页面组件：profile.js、square_detail.js、add_vehicle.js;
- 功能组件：存放在 component 目录下;
- 静态资源：avatar.png、icon-back.png;
- 后端文件采用小写字母配合连字符命名;
- 后端程序文件：post-end.py;

这种命名方式保持了文件名的一致性和可读性，同时也便于在不同操作系统中使用。

7.2 前端编码规范

7.2.1 命名规范

在 React Native 开发中，采用规范的命名约定：

- 组件名使用大驼峰命名法，如 ProfileScreen；
- 变量和状态使用小驼峰命名法，如 userInfo、setUserInfo；
- 样式对象属性使用小驼峰命名法，如 headerCurtain、buttonText；

这种命名方式符合 JavaScript 社区通用做法，提高了代码的可读性和维护性。

```
// 组件名：大驼峰命名
const ProfileScreen = ({navigation, route}) => { }

// 变量名：小驼峰命名
const [userInfo, setUserInfo] = useState({});

// 样式对象：小驼峰命名
const styles = StyleSheet.create({
  container: { },
  headerCurtain: { },
  backButtonContainer: { }
});
```

7.2.2 代码格式化风格

代码格式保持统一的组织结构：

- 导入声明分组整理，React 相关库优先导入；
- 组件内部结构清晰，状态定义、副作用、方法实现和渲染逻辑分块组织；
- 样式定义统一放置在组件底部；

这种格式化风格使代码结构清晰，便于阅读和维护。

```
// 导入声明分组
import React, {useState, useEffect} from 'react';
import {View, Text, Image} from 'react-native';
import AsyncStorage from '@react-native-async-storage/async-storage
';
// 组件定义
const ComponentName = ({props}) => {
```

```
  // 状态定义
  const [state, setState] = useState();
  // 副作用
  useEffect(() => {
    // 实现
  }, []);

  // 返回 JSX
  return (
    <View>
      <Text>Content</Text>
    </View>
  );
};
```

7.2.3 语句与表达式

代码编写遵循现代 JavaScript 的最佳实践：

- 使用 async/await 处理异步操作;
- 使用箭头函数定义方法;
- 使用解构赋值简化代码;
- 条件渲染使用三元运算符或&&运算符;

```
// 异步函数使用 async/await
const fetchUserInfo = async () => {
    try {
      const token = await AsyncStorage.getItem('token');
      // 实现
    } catch (error) {
      // 错误处理
    }
  };
  // 条件渲染使用三元运算符或&&
  {view === 'myself' && (
    <TouchableOpacity>
      // 内容
    </TouchableOpacity>
  )}
```

7.2.4 异常处理规范

异常处理采用 try-catch 结构，针对不同类型的错误提供友好的用户提示：

- 网络请求错误统一处理;
- 使用 Alert.alert 提供用户反馈;
- 在控制台记录详细错误信息;

```
try {
    const response = await fetch(`${BASE_URL}/users/info`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${token}`,
      }
    });
    if(response.ok) {
      const data = await response.json();
    } else {
      Alert.alert('失败', '获取个人信息失败，请稍后再试~');
    }
  } catch (error) {
    console.error('Error fetching user info:', error);
  }
```

7.3 后端编码规范

7.3.1 命名规范

Python 后端采用清晰的命名规则：

- 类名使用大驼峰命名法，如 User、Order;
- 函数和变量使用小写字母加下划线，如 hash_password、db_session;
- 常量使用大写字母加下划线，如 JWT_SECRET;

```
# 类名：大驼峰命名
class User(Base):
    __tablename__ = "users"

# 函数名：小写字母，下划线连接
```

```
def hash_password(password):
    return hashlib.sha256(password.encode("utf-8")).hexdigest()

# 变量名：小写字母，下划线连接
db_session_class = sessionmaker(bind=engine)
```

7.3.2 代码格式化规范

后端代码保持统一的格式化风格：

- 导入语句按标准库、第三方库、本地模块顺序排列；
- 类和函数定义之间保留两个空行；
- 相关的功能代码组织在一起；

这种格式化风格提高了代码的可读性和可维护性。

```
# 导入分组
from flask import Flask, request
from sqlalchemy import Column, Integer
from datetime import datetime

# 常量定义
JWT_SECRET = "pintu12345"
JWT_EXPIRATION_DELTA = timedelta(hours=2)

# 类定义
class Order(Base):
    __tablename__ = "orders"
    id = Column(Integer, primary_key=True)
```

7.3.3 语句与表达式

代码编写遵循 Python 的惯用做法：

- 使用装饰器定义路由和中间件；
- 使用列表推导式和生成器表达式；
- 合理使用条件表达式和循环结构；

```
# 路由定义
@app.route("/users/register", methods=["POST"])
def register():
```

```
    if request.method == "POST":
        data = request.json
        email = data.get("mail")

        # 条件判断
        if existing_user:
            return jsonify({"status":"failed"})
```

7.3.4 异常处理规范

采用结构化的异常处理机制：

- 使用 try-except 捕获具体异常类型；
- 数据库操作错误单独处理；
- 提供清晰的错误信息和 HTTP 状态码；
- 发生异常时进行事务回滚；

```
try:
    db_session.add(new_user)
    db_session.commit()
except SQLAlchemyError as e:
    db_session.rollback()
    return jsonify({
        "status": "failed",
        "message": f"数据库错误: {str(e)}"
    }), HTTPStatus.INTERNAL_SERVER_ERROR
except Exception as e:
    return jsonify({
        "status": "failed",
        "message": f"服务器错误: {str(e)}"
    }), HTTPStatus.INTERNAL_SERVER_ERROR
```

八、代码编写

8.1 技术栈架构

“拼途手机私家车拼车软件系统”采用前后端分离的开发方式进行代码编写，项目完整的架构如下图所示：

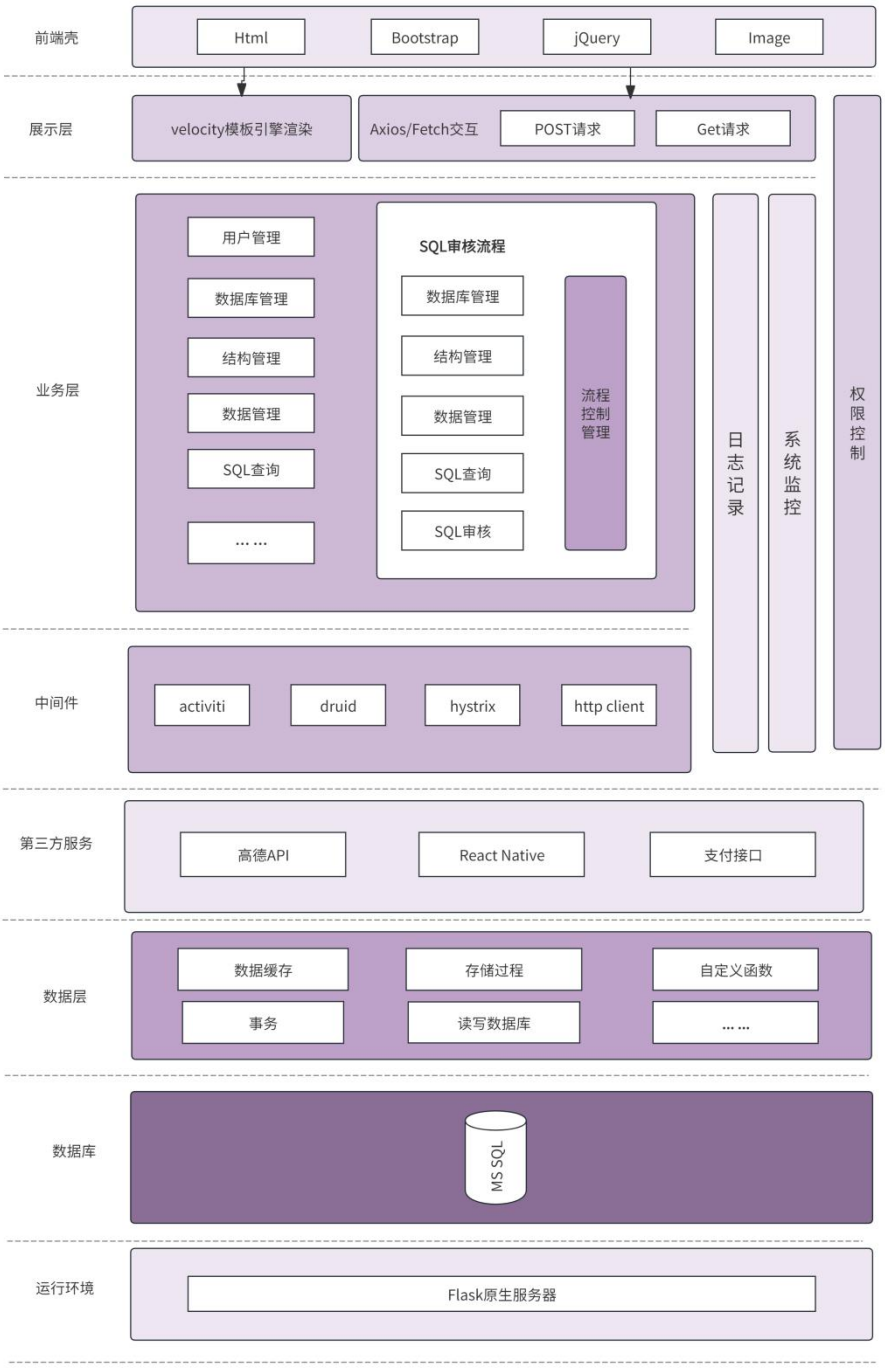

图 26 系统完整技术栈架构图

在项目和代码管理方面，小组成员全程使用 Github 管理全部代码，通过合理运用分支管理（如 develop、feature 分支），到代码的提交（commit）、推送（push），再到合并请求（Pull Request）和代码审查（Code Review）等工具，历时 3 个月，完成了拼途 PINTU-APP 的软件开发。

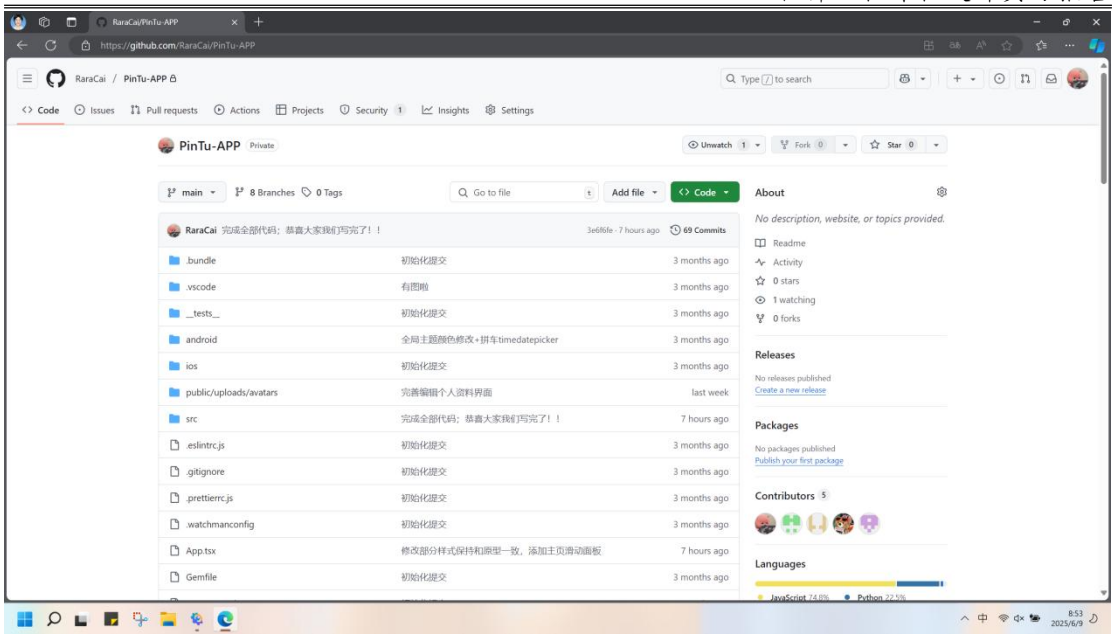

图 27 本项目的 Github 网址

8.2 功能实现

按照需求分析阶段的原型设计方案，我们对原型中提及的所有功能都进行了代码实现。总体而言，所有页面及相关功能都按照：前端处理用户数据并发起网络请求、服务器转发网络请求、后端接受请求实现数据库操作并返回 JSON 响应、前端处理响应的流程进行。

下面将详述本系统部分核心功能的具体代码实现和重点函数说明：

8.2.1 页面导航与跳转

整个软件系统的页面跳转依赖于 App.tsx 中声明的全局导航栈（Stack Navigation），该导航栈基于 React Native 的原生系统。在具体代码编写上，通过全局声明 NavigationContainer 中的元素来实现页面定义：

```
<Stack.Screen
      name="Square"
      component={SquareScreen}
      options={{
        headerShown: true,
        title: '拼途广场',
        headerTitleAlign: 'center',
        headerStyle: {
          elevation: 0,
```

```
            shadowOpacity: 0,
          },
        }}
      />
```

在每个具体的.js 页面中，当需要进行页面跳转时，直接通过在 script 部分调用定义的 name 实现：

```
// 跳转到主页面
navigation.navigate('Home');
```

在本软件系统中，所有的详情页面都有固定的上游页面，因此当需要返回时，可以直接使用内置的 goBack()方法将页面弹出栈顶，回到上层：

```
// 回到上一个页面
navigation.goBack();
```

当页面堆栈层级过多、或一次完整的拼车订单已结束时，需要将当前的导航堆栈清空，并将页面定位回初始页面：

```
navigation.reset({
    index: 0,
        routes: [{ name: 'Home' }],
})
```

8.2.2 司机发车与乘客拼车

司机发车与乘客拼车的功能主要在 home.js 文件中完成，该文件作为父组件，分别使用 import 方法调用了 tabContent.js、Map.js、RouteInfoDisplay.js 等文件作为子组件。其中，tabContent 用于处理用户填入的页面数据，Map 用于定位用户位置以及显示路劲规划结果，RouteInfoDisplay 用于显示本订单详情信息以及地图规划数据。tabContent.js 链接了 search.js 完成地点模糊搜索、常用地点添加和历史记录列表管理等。

在这种父子组件嵌套调用的代码实现方式中，需要有效处理回调和数据传递。此处，我们以 tabContent.js（子组件）与 home.js（父组件）为例，详述具体的代码实现（本软件系统中的其他组件封装与调用的实例同理）。

在 home.js 的 script 部分定义了拼车相关的父组件数据如下：

```
    const [userLocation, setUserLocation] = useState(null);
    const [routePlanningData, setRoutePlanningData] = useState(null);
    const [plannedRouteInfo, setPlannedRouteInfo] = useState({
      startName: '',
      startLocation: '',
      startID: '',
      endName: '',
      endLocation: '',
      endID: '',
    });
    const [selectedDepartureTime, setSelectedDepartureTime] = useState
('');
    const [activeTab, setActiveTab] = useState('');
    const [car, setCar] = useState('');
    const [passengerLimit, setPassengerLimit] = useState('');
    const mapRef = useRef(null);
```

在调用子组件 tabContent 时需要通过回传的方式实现数据的传递，之后这些数据将在 tabContent 中完成各种操作。tabContent 定义的 onRoutedPlanned 方法即为处理回调的方法，当内部数据处理完成将发出回调信号，父组件在收到信号后将继续执行在本文件中定义的 handleRoutePlanned 方法，完成后续操作：

```
    <TabContent
        userLocation={userLocation}
        onRoutePlanned={handleRoutePlanned}
    />
```

在 TabContent 内部的 script 部分定义了同步或异步的函数用于处理前端各种用户数据。同步函数多用于处理前端展示的数据，而异步函数（比同步函数增加一个 async 异步声明）需要向后端服务器发送请求、与数据库中的条目进行增删改查的交互。例如，此处展示的是前端用户已经在 tabContent 中填写完毕订单详

情，处理司机创建订单请求的异步函数（仅展示部分代码）：

```
const handleDepartSubmit = async () => {
  // ...前问涉及 token 验证与数据合法性检查，省略
  try {
        const requestBody = {
          origin: startLocation.location,
          destination: endLocation.location,
          originID: startLocation.id,
          destinationID: endLocation.id,
        };

        const response = await fetch(`${BASE_URL}/route_planning`, {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
            Authorization: `Bearer ${token}`,
      },
          body: JSON.stringify(requestBody),
        });
      // ...后续省略
  };
```

上述请求将经过服务器转发，发送到后端被 Python Flask 代码处理。在后端的接口代码中，我们将首先解析请求体数据，从中获取需要的字段。然后使用此字段与数据库进行交互。此处展示后端与创建订单相关的接口（仅展示部分代码）：

```
@app.route("/orders/create", methods=["POST"])
@token_required
def create_order():
    # token 身份验证
    driver_id = request.user["u_id"]
    # 从 JSON 数据中获取所需字段
    data = request.get_json()
    car_id = data.get('car_id') # 车辆 ID，外键关联 cars 表
    passenger_limit = data.get('passenger_limit')
    # 其他字段的获取与上面同理，省略
    # 数据合法性检验，省略
    # 创建新订单
```

```
        new_order = Order(
            driver_id=driver_id,
            car_id=car_id,
           # 其他字段的赋值与上面同理，省略
        )
        db_session.add(new_order)
        # ...后续代码省略
```

本系统的所有前后端交互都与本例类似，只是在前端的数据结构定义与后端接口的具体实现上存在不同。因此，在后文中我们将不再详述前后端数据交互及相关的网络请求发送，以及所有于此过程相关或类似的函数实现。

8.2.3 路径规划与订单详情

本软件系统的路径规划基于第三方平台高德开放 API，通过在前端的 search.js 界面确定起终点的 POI ID（高德地图规定的唯一标识，用于标志地图中的标志性建筑与地点等）或经纬度坐标，使用网络请求的方式将这些位置信息发送至后端，后端接口将发起第三方 API 调用，并将路径规划的结果按 JSON 格式返回。

```
# 高德地图 API 配置
    amap_url = "https://restapi.amap.com/v5/direction/driving"
    params = {
        "key": gd_key,
        "origin": origin,
        "destination": destination,
        "origin_id":originID,
        "destination_id":destinationID,
        "show_fields": "polyline,cost"
    }
    try:
        # 调用高德路径规划 API
        response = requests.get(amap_url, params=params)
        response.raise_for_status()
        data = response.json()
        # 合法性检查及 JSON 数据返回，省略
```

有关订单详情的展示，其实现方式与 tabContent 组件的工作原理相似。Search_detail.js 作为父组件，调用 OrderInfoDisplay.js 和 RouteInfoDisplay.js 等子组件，

数据与方法通过回传回调等方式触发，此处不再重负赘述。

8.2.4 群聊与行驶全过程的操作

群聊功能与行驶全过程的审批、上车、启动和终止行程依赖于多个用户之间的协作。通过之前所说的前后端交互形式，对数据库中 Passenger、Order、Apply 等多个数据表进行联合修改，从而实现了完整的功能流。

在本部分内容中，司机端与乘客端将在软件上看到不同的界面。其实现方式是在页面挂载时（使用 React Native 原生的 UseEffect 方法监听）向后端索取状态（status），并依据状态字段的值在前端进行个性化的渲染：

```
// 页面挂载获取本订单详情
    useEffect(() => {
        fetchOrderDetails(order_id);
    },[order_id]);
```

前端的个性化显示是依靠三目表达式实现的，根据后端回传的数据给页面数据赋值，进而再根据页面数据的值判断显示何种内容：

```
{ order.role === 'driver' && (
    <View style={{marginBottom: 20}}>
        { newApplies.length > 0 ? (
            // 司机端有新拼车申请时的内容，省略
             ) : (
            // 司机端无新拼车申请时的内容，省略
       )}
    </View>
 )}
```

8.2.5 用户评价系统

本软件的用户评价系统主要再 review.js 文件中完成，页面每次挂载时（使用 UseEffect 方法监听）判断当前用户是否已经对本订单评价完成，并个性化地展示出评价结果或评分框。若用户尚未打分，则在用户打分后将数据传送给后端，修改数据库中相关的 Rating 表等字段。

8.2.6 用户数据管理

本软件的用户数据管理主要分为用户个人基本信息数据和座驾信息数据管理两大部分。其中，个人基本信息管理主要在 profile.js 和 edit_profile.js 两个文件中实现，座驾信息管理主要在 my_vehicle.js 和 add_vehicle.js 两个文件中实现。

上述功能的实现原理都与 8.2.2 和 8.2.3 中所述相同，主要涉及后端数据库中 User 和 Car 两个数据表的增删改查。

九、软件测试

9.1 测试需求

本次测试围绕“拼途手机私家车拼车软件系统”基于 React Native 技术栈、JavaScript 语言特性、高德地图 API 集成且覆盖基本功能、满足稳定高效安全用户需求及良好体验的核心需求展开。

9.1.1 功能性测试需求

功能性测试旨在验证软件的各项功能是否按照需求规格说明书正确实现，并且在各种正常和异常情况下都能按预期运行。

- **用户身份与访问控制：**

（1）验证新用户（区分为乘客和司机角色）能够通过设定的方式（邮箱账号）完成账户创建流程，包括邮箱邮件校验、密码设置等。

（2）验证已注册用户能够按流程成功登录系统，并能顺利退出登录。

（3）验证针对司机角色的特定身份认证流程（如个人信息、驾驶执照、车辆信息的提交与审核状态反馈）能够正确执行。

（4）验证用户能够查看和编辑其个人基本资料（如昵称、头像等）。

- **拼车行程的发起与响应：**

（1）验证乘客能够明确指定出行需求，包括准确设定起点和终点（通过手动输入或地图交互方式）、选择出行时间、乘车人数以及期望的拼车类型。

（2）验证司机用户能够发布可提供服务的行程信息，包括设定可接单区域、时间、允许拼车人数等。

（3）验证系统能够根据用户需求进行有效的订单匹配和展示。

- **行程信息的检索与管理：**

（1）验证乘客能够根据设定的条件（如地点、时间）搜索到符合条件的拼车单。

（2）验证系统能够清晰、准确地展示行程详情，包括但不限于路线信息、车辆信

息、司机信息、预估费用、同行者信息。

（3）验证用户（乘客和司机）能够查看其当前和历史行程记录。

- **拼车过程的执行与交互：**

（1）验证司机在接受订单后，能够获取乘客信息及规划路线提供服务。

（2）验证行程进行中，关键状态（如司机已出发、已到达约定地点、接到乘客、行程结束）能够被正确记录和通知相关方。

- **地图服务与位置功能的集成（基于高德地图 API）：**

（1）验证系统能够准确获取并展示用户当前的地理位置。

（2）验证地图界面的基本交互（如加载、缩放、拖动）流畅且准确，标记点（如起点、终点）显示清晰无误。

（3）验证通过关键词搜索地点能够正常工作，并返回准确结果。

（4）验证系统能根据起点和终点信息，利用高德地图 API 规划出合理的行驶路线，并在地图上进行可视化展示，包含预计距离、时间和费用。

- **用户间的沟通与反馈：**

（1）验证在行程确认后或特定场景下，乘客与司机之间存在有效的沟通渠道。

（2）验证行程完成后，用户（乘客和司机）能够对本次服务进行评价，评价信息能被正确提交和展示（在适当的聚合后）。

- **费用计算与支付处理（若适用）：**

  验证系统能根据既定规则（如里程、时长、车型等）准确预估拼车费用。

- **系统通知与提示：**

（1）验证关键的业务节点和状态变更（如订单确认、司机接单、行程开始/结束等）能通过应用内通知或推送等方式及时告知用户。

（2）验证系统重要提示信息能够有效传达给目标用户。

- **边界条件与异常情况处理：**

（1）验证系统对用户输入的有效性进行校验（如申请拼车人数、时间选择的合理性、车牌号等），并对无效输入给出清晰的提示。

（2）验证在网络连接不稳定或中断、服务器响应超时、第三方服务（如高德地图 API、支付接口）不可用等异常情况下，系统能有优雅的处理机制（如错误提示、重试机制、功能降级等），避免应用崩溃或数据混乱。

9.1.2 非功能性测试需求

- **性能测试：**

  应用启动速度、页面加载速度、地图交互响应时间等。

- **兼容性测试：**

在不同的 Android/iOS 版本或设备模拟器/真机上进行测试（如果条件允许）。

- **可用性测试：**

  用户体验是否良好，操作是否便捷。

9.2 测试框架

根据本项目的技术栈（React Native/JavaScript/高德地图 API/Python Flask），主要采取以下测试技术及相关工具/库：

核心框架与语言	React Native (基于 JavaScript/TypeScript)
地图服务	高德地图开放平台 SDK
API 通信	Fetch API / Axios
单元/组件测试工具	Jest, React Testing Library
调试与环境支持	React Native Debugger, Flipper, Node.js

说明：在快速迭代和构建过程中，我们可能优先确保核心集成路径的稳定，单元测试按需进行，并可配置在 CI/CD 流程中选择性执行。

9.3 单元测试

单元测试将按照单个.js 文件进行，每个.js 文件视为一个单元逐个进行测试。此处为节省篇幅，我们仅展示部分单元测试中重要功能的的详细测试流程，其他单元测试同理。

（1）RouteInfoDisplay.js（路线信息展示组件）：

辅助函数 (formatDistance, formatDuration, formatTime) 单元测试:

test_formatDistance_various_inputs	测试不同距离输入（有效数字，null, undefined, 非数字字符串）的输出是否符合预期。
test_formatDuration_various_inputs	测试不同时长输入（秒，分钟，小时，null, undefined）的输出是否符合预期。
test_formatTime_various_inputs	测试不同 Date 对象输入（有效日期，无效日期，null）的输出是否符合预期。

组件渲染单元测试 (使用 React Testing Library, 传入 props):

test_RouteInfoDisplay_renders_correctly_with_valid_data	测试当传入有的 routeData, startName, endName, departureDateTime, activeTab 时，组件是否正确显示所有信息（司机占位信息，日期，路线，距离，耗时，出发时间，预计到达时间，价格，按钮文本）。

test_RouteInfoDisplay_handles_missing_route_data	测试当 routeData 或其内部 paths 为 null 或空时，距离、耗时、价格等是否显示为占位符（如 '--' 或 '? km'）。
test_RouteInfoDisplay_handles_missing_departure_time	测试当 departureDateTime 缺失时，出发时间和预计到达时间是否显示为占位符。
test_RouteInfoDisplay_cancel_button_calls_onCancel	测试点击取消按钮时，是否调用 onCancel prop。
test_RouteInfoDisplay_confirm_button_calls_onConfirm	测试点击确认按钮时，是否调用 onConfirm prop。
test_RouteInfoDisplay_button_text_changes_based_on_activeTab	测试取消和确认按钮的文本是否根据 activeTab (拼车/发车) 的不同而正确显示。
test_RouteInfoDisplay_price_label_changes_based_on_activeTab	测试价格标签（订单总价/人均价格）是否根据 activeTab 正确显示。

（2）Map.js（地图组件）：

parsePolyline() 辅助函数单元测试：

test_parsePolyline_valid_string	测试输入有效的 polyline 字符串，是否返回正确的经纬度坐标数组。
test_parsePolyline_empty_string	测试输入空字符串，是否返回空数组。
test_parsePolyline_null_or_undefined_input	测试输入 null 或 undefined，是否返回空数组。
test_parsePolyline_string_with_invalid_points	测试 polyline 字符串中包含无效坐标点（如格式错误或非数字）时，是否能正确过滤掉无效点。

组件渲染单元测试 (使用 React Testing Library, 关注 Marker 和 Polyline 的条件渲染):

test_MapScreen_renders_MapView	测试 MapView 组件是否被渲染。

test_MapScreen_renders_my_location_marker_if_location_exists	测试当 location prop 有效时，是否渲染当前位置的 Marker。
test_MapScreen_renders_route_polyline_if_routeCoordinates_exist	测试当 routeCoordinates 数组不为空时，是否渲染 Polyline。
test_MapScreen_renders_start_marker_if_startMarkerPos_exists	测试当 startMarkerPos 有效时，是否渲染起点 Marker。
test_MapScreen_renders_end_marker_if_endMarkerPos_exists	测试当 endMarkerPos 有效时，是否渲染终点 Marker。

（3）tabContent.js（主界面底部 Tab 内容组件）：

handleConfirm() (日期时间选择器确认回调) 函数单元测试:

test_handleConfirm_formats_datetime_correctly_and_updates_state	测试选择日期时间后，selectedDateTime state 是否被格式化为 "YYYY/MM/DD HH:mm" 并正确更新，以及 isDateTimePickerVisible 是否变为 false。

handleNavigateToSearch() 和 HandleNavigateToMyVehicle() 导航函数单元测试 (mock navigation):

test_handleNavigateToSearch_calls_navigation_with_correct_params	测试调用 handleNavigateToSearch 时，是否使用正确的参数（'Search', type, userLocation, onSelectLocation 回调）调用 navigation.navigate。
test_HandleNavigateToMyVehicle_calls_navigation_with_correct_params	测试调用 HandleNavigateToMyVehicle 时，是否使用正确的参数（'MyVehicle', onSelectCar 回调）调用 navigation.navigate。

handleDepartSubmit() 函数单元测试：

test_handleDepartSubmit_missing_basic_info	测试起点、终点或出发时间未选择时，是否弹出提示。
test_handleDepartSubmit_invalid_coordinates	测试起点或终点坐标格式无效时，是否弹出提示。
test_handleDepartSubmit_missing_driver_specific_info	测试乘客数上限或座驾未选择时，是否弹出提示。
test_handleDepartSubmit_invalid_max_passengers	测试乘客数上限无效（非数字，<=0 或 >10）时，是否弹出提示。

test_handleDepartSubmit_no_token	未登录处理。
test_handleDepartSubmit_successful_route_planning_and_calls_onRoutePlanned	测试路径规划 API 调用成功后，是否正确调用 onRoutePlanned 回调并传入所有必要参数。
test_handleDepartSubmit_route_planning_api_error_handling	测试路径规划 API 各种错误状态码的处理。

（4）register.js（用户注册界面）：

handleRegister() 函数单元测试:

test_handleRegister_passwords_not_match	测试当两次输入的密码不一致时，是否正确弹出 Alert 提示，并且 isSubmitting 状态未改变，没有发起网络请求。
test_handleRegister_successful_registration	测试当输入有效且密码一致时，是否正确发起注册 API 请求，请求体是否正确构建，成功响应后是否弹出成功提示，并且导航到登录页，isSubmitting 状态是否正确更新。
test_handleRegister_api_error_response	测试当注册 API 返回错误状态码或错误信息时，是否正确弹出错误提示，并且 isSubmitting 状态是否正确恢复。
test_handleRegister_network_error	测试当网络请求失败时（例如，fetch 抛出异常），是否正确弹出网络错误提示，并且 isSubmitting 状态是否正确恢复。
test_handleRegister_button_submission_state	测试在调用 handleRegister 后，isSubmitting 状态是否立即变为 true，并在请求完成后（无论成功或失败）恢复为 false。

（5）home.js（主界面）：

requestLocation() 函数单元测试 (需要 mock PermissionsAndroid 和 Geolocation):

test_requestLocation_permission_granted_android	测试 Android 平台权限授予后，是否调用 Geolocation.getCurrentPosition 并正确设置 userLocation。
test_requestLocation_permission_denied_android	测试 Android 平台权限拒绝后，是否不调用 Geolocation.getCurrentPosition。
test_requestLocation_permission_granted_ios	测试 iOS 平台权限授予后，是否调用 Geolocation.getCurrentPosition。

test_requestLocation_geolocation_error	测试 Geolocation.getCurrentPosition 返回错误时的处理。
test_requestLocation_moves_camera_if_no_route_data	测试在没有路线规划数据时，定位成功后是否移动地图相机。

handleConfirmCarpool() (发车逻辑) 函数单元测试:

test_handleConfirmCarpool_depart_no_token	测试未登录时处理。
test_handleConfirmCarpool_depart_successful_creation	测试发车 API 调用，请求体是否正确构建，成功后是否弹出提示，重置状态并导航到广场。
test_handleConfirmCarpool_depart_api_error	测试 API 错误处理。
test_handleConfirmCarpool_depart_network_error	测试网络错误处理。

组件渲染与交互单元测试 (使用 React Testing Library, mock 子组件):

test_HomeScreen_renders_loading_state_initially	测试初始 userLocation 为 null 时是否显示加载指示器。
test_HomeScreen_renders_map_and_tabContent_when_location_available_and_no_route	测试当 userLocation 有效且 routePlanningData 为 null 时，是否渲染地图和 TabContent。
test_HomeScreen_renders_route_info_when_route_planned	测试当 routePlanningData 有效时，是否渲染地图和 RouteInfoDisplay，并隐藏 TabContent。
test_HomeScreen_top_buttons_and_recenter_button_visibility	测试顶部按钮和定位按钮是否根据 userLocation 和 routePlanningData 的状态正确显示/隐藏。
test_HomeScreen_route_header_visibility	测试路线规划页头是否根据 routePlanningData 的状态正确显示/隐藏。

（6）search.js（地点搜索界面）：

fetchSuggestions() 函数单元测试:

test_fetchSuggestions_no_token	测试当 AsyncStorage 中没有 token 时，是否弹出未登录提示，suggestions state 是否为空数组。
test_fetchSuggestions_invalid_input	测试当 userLocation 或 currentQuery 无效时，是否不发起网络请求，suggestions state 是否为空数组。
test_fetchSuggestions_successful_fetch	测试当输入有效时，是否正确发起搜索建议 API 请求，请求体是否正确构建，成功响应后 suggestions s

	tate 是否正确更新。
test_fetchSuggestions_api_error_response	测试当搜索建议 API 返回错误时，suggestions state 是否为空数组。
test_fetchSuggestions_network_error	测试当网络请求失败时，是否弹出网络错误提示，suggestions state 是否为空数组。

handleSelectSuggestion() 函数单元测试:

test_handleSelectSuggestion_updates_history_correctly	测试选择一个建议项后，历史记录 history state 是否被正确更新（新项在顶部，旧项存在则移到顶部，总数不超过 10）。
test_handleSelectSuggestion_saves_history_to_AsyncStorage	测试选择建议项后，更新后的历史记录是否被正确保存到 AsyncStorage。
test_handleSelectSuggestion_calls_onSelectLocation_and_navigates_back	测试选择建议项后，是否正确调用 onSelectLocation 回调函数，并执行返回上一页的导航操作。

（7）sqaure.js（拼车广场界面）：

handleConfirmCarpool() 函数单元测试:

test_handleConfirmCarpool_no_token	测试未登录时是否提示并导航到登录页。
test_handleConfirmCarpool_successful_application	测试是否正确发起申请拼车 API 请求，请求体是否包含正确的 order_id 和 passengerCnt，成功响应（status 'success'）后是否弹出成功提示，关闭输入弹窗。
test_handleConfirmCarpool_repeated_application	测试 API 返回 'repeated' 状态时，是否弹出重复申请提示。
test_handleConfirmCarpool_same_user_application	测试 API 返回 'same' 状态时，是否弹出无需申请自己订单的提示。
test_handleConfirmCarpool_api_error_response	测试 API 返回其他错误时，是否弹出失败提示。
test_handleConfirmCarpool_network_error	测试网络请求失败时，是否弹出网络错误提示。

组件渲染与交互单元测试 (使用 React Testing Library):

test_SquareScreen_renders_tabs_correctly	测试三个 Tab（可拼单，已申请，已发起）是否正确渲染。

test_SquareScreen_tab_switching_loads_correct_data	测试切换 Tab 时，是否调用对应的 fetch 函数并显示相应的加载状态。
test_SquareScreen_renderListItem_displays_order_info	测试 renderListItem 是否正确渲染订单信息（司机信息，路线，时间，价格等）。
test_SquareScreen_renderListItem_confirm_button_opens_modal	测试点击“确认参与”按钮时，showInputModal 是否为 true，selectedOrderId 是否设置。
test_SquareScreen_passenger_count_modal_functions_correctly	测试人数确认弹窗的输入框是否能更新 inputPassengerCnt，确认和取消按钮是否能正确触发 handleConfirmCarpool 和关闭弹窗。
test_SquareScreen_renderAppliedListItem_displays_status_and_actions	测试 renderAppliedListItem 是否根据订单状态显示正确的文本颜色和操作按钮（如“查看详情”，“填写评价”）。
test_SquareScreen_renderCreatedListItem_displays_status_and_actions	测试 renderCreatedListItem 是否根据订单状态显示正确的文本颜色和操作按钮。
test_SquareScreen_empty_list_component_renders_when_no_data	测试当订单列表为空时，是否显示“没有可显示的订单”等提示文本。

（8）square_detail.js（拼车详情界面）：

fetchOrderDetails() 函数单元测试：

test_fetchOrderDetails_no_token	测试未登录时处理。
test_fetchOrderDetails_successful_fetch	测试获取订单详情 API 调用，成功后 orderDetails state 是否更新。
test_fetchOrderDetails_api_error	测试 API 错误处理。
test_fetchOrderDetails_network_error	测试网络错误处理。

useEffect 钩子逻辑单元测试 (模拟生命周期和依赖变化)：

test_useEffect_fetches_order_details_on_mount	测试组件挂载时是否调用 fetchOrderDetails。
test_useEffect_calls_route_planning_when_order_details_available	测试当 orderDetails 更新后，是否调用 handleRoutePlanning。
test_useEffect_sets_correct_header_title_based_on_type_and_status	测试根据 type 和 orderDetails.order_status，headerTitle 是否被正确设置。

（9）add_vehicle.js（添加座驾界面）：

validatePlate() 函数单元测试:

test_validatePlate_valid_traditional_plate	测试传统车牌号（如 "京 A12345"）是否返回 true。
test_validatePlate_valid_new_energy_plate	测试新能源车牌号（如 "京 A123456"）是否返回 true。
test_validatePlate_invalid_province_abbreviation	测试无效省份简称的车牌号是否返回 false。
test_validatePlate_invalid_format_too_short	测试格式错误（过短）的车牌号是否返回 false。
test_validatePlate_invalid_format_too_long	测试格式错误（过长）的车牌号是否返回 false。
test_validatePlate_invalid_characters	测试包含无效字符的车牌号是否返回 false。
test_validatePlate_plate_with_spaces	测试带空格的车牌号在去除空格后是否能正确验证。

组件渲染与交互单元测试 (使用 React Testing Library):

test_AddVehicleScreen_renders_correctly	测试界面是否渲染所有输入字段（车型选择器，车牌号，油耗，品牌）和添加按钮。
test_AddVehicleScreen_category_picker_updates_state	测试选择不同车型时，category state 是否正确更新。
test_AddVehicleScreen_plate_input_updates_state	测试车牌号输入框更新 plate state。
test_AddVehicleScreen_fuel_consumption_input_updates_state_and_filters_input	测试油耗输入框更新 fuelConsumption state，并验证其是否只接受数字和小数点，且最多两位小数。
test_AddVehicleScreen_brand_input_updates_state	测试品牌输入框更新 brand state。
test_AddVehicleScreen_add_button_disabled_when_submitting	测试 isSubmitting 为 true 时添加按钮的禁用状态。

（10）my_vehicle.js（我的座驾列表界面）：

openActionModal() 和 confirmSelection() 函数单元测试:

test_openActionModal_sets_state_correctly	测试调用 openActionModal 后，selectedCar 和 modalVisible state 是否正确设置。
test_confirmSelection_calls_onSelectCar_and_navigates_back	测试调用 confirmSelection 后，是否调用 onSelectCar 回调（如果存在），关闭模态框并返回。

组件渲染与交互单元测试 (使用 React Testing Library):

test_MyVehicleScreen_renders_loading_state	测试 loading 为 true 时显示加载状态。
test_MyVehicleScreen_renders_empty_state_when_no_cars	测试 cars 数组为空时显示“暂无座驾信息”。
test_MyVehicleScreen_renders_car_list_correctly	测试当 cars 数组有数据时，是否正确渲染车辆列表项。
test_MyVehicleScreen_car_item_press_opens_modal	测试点击车辆列表项时，是否调用 openActionModal。
test_MyVehicleScreen_add_button_navigates_to_AddVehicleScreen	测试点击“新增车辆”按钮是否导航到 AddVehicle 页面。
test_MyVehicleScreen_modal_options_function_correctly	测试模态框中的“确认选择”和“删除座驾”按钮是否能正确触发相应函数。

9.4 集成测试

集成测试的重点在于验证不同模块（组件、服务、API 调用等）协同工作时的正确性。在我们的 React Native 应用中，主要包括如下几个方面：

- **导航流程测试**：验证用户在不同屏幕间的跳转是否符合预期。
- **组件间交互与数据传递**：验证父子组件、兄弟组件或通过导航传递的数据是否正确。
- **前端与后端 API 集成**：验证前端发起的 API 请求以及对响应的处理是否正确，确保数据流的完整性。
- **第三方服务集成**：例如地图服务加载、定位、路线规划等功能的正确性。
- **状态管理与持久化**：验证如 AsyncStorage 的使用是否正确，以及全局状态（如果使用 Context API 或 Redux 等）在不同组件间的同步。

在本次课程设计的集成测试部分中，我们分别针对：用户注册与登录流程、地点搜索与选择流程、用户发车与拼车流程、查看拼车详情与地图路线流程、车辆管理流程进行了不同场景下的集成测试。此处，我们选择最为核心的“用户发车流程”“用户拼车流程”以及“查看拼车详情与地图路线流程”3 个场景，详述集成测试的具体流程，前文提到的其他场景下的集成测试同理：

- **场景 1: 用户发车流程 (依赖用户已登录)**

（1）test_Driver_Create_Carpool_Order_Flow：

测试司机用户从 HomeScreen 填写发车信息，查看路线规划，确认并发起拼车订单的完整流程。详细步骤如下：

① 确保用户已登录并导航到 HomeScreen。

② 在 TabContent 切换到“我要发车” Tab。

③ 选择起点、终点、上车时间、填写乘客数上限、选择座驾 (涉及导航到 MyVehicleScreen 并返回选择)。

④ 模拟点击“发起拼车”按钮 (在 TabContent 中)。

⑤ 验证:

- handleDepartSubmit 是否被调用。
- 是否向后端 /route_planning 发起 POST 请求，请求体包含正确的起点终点信息。
- 路径规划成功后，onRoutePlanned 回调是否被触发，HomeScreen 的 routePlanningData, plannedRouteInfo, selectedDepartureTime, car, passengerLimit, activeTab state 是否正确更新。
- 界面是否切换到显示 RouteInfoDisplay 组件。

⑥ 在 RouteInfoDisplay 组件中，检查显示的路线信息、时间、价格是否与规划结果一致。

⑦ 模拟点击“发起订单”按钮 (在 RouteInfoDisplay 中)。

⑧ 验证:

- HomeScreen 的 handleConfirmCarpool (发车逻辑) 是否被调用。
- 是否向后端 /orders/create 发起 POST 请求，请求体包含所有正确的订单信息（车辆 ID，乘客上限，计划时间，起终点名称、ID、坐标，距离，价格等）。
- 订单创建成功后，是否弹出“发车成功”提示。
- HomeScreen 的路线规划相关 state 是否被重置。
- 是否导航到 SquareScreen。

本过程的关键集成点如下：

HomeScreen (TabContent) 表单交互 -> MyVehicleScreen 导航与数据返回 -> /route_planning API 集成 -> RouteInfoDisplay 组件数据展示与交互 -> /orders/create API 集成 -> SquareScreen 导航。

- **场景 2: 用户拼车流程 (依赖用户已登录)**

（1）test_Passenger_Join_Carpool_Order_Flow：

测试乘客用户从 HomeScreen 填写拼车需求，系统进行订单匹配，导航到 SquareScreen 查看匹配结果，选择订单并申请加入的流程。详细步骤如下：

① 确保用户已登录并导航到 HomeScreen。

② 在 TabContent 切换到“我要拼车” Tab。

③ 选择起点、终点、上车时间、填写上车人数、调整价格区间。

④ 模拟点击“参与拼车”按钮 (在 TabContent 中)。

⑤ 验证:

- handleCarpoolSubmit 是否被调用。
- 是否向后端 /orders/match 发起 POST 请求，请求体包含所有正确的拼车需求信息。
- 订单匹配成功后，是否使用后端返回的 status 和 orders 数据导航到 SquareScreen。

⑥ 在 SquareScreen (假设显示“可拼单” Tab)，验证订单列表是否根据匹配结果正确渲染。

⑦ 模拟用户点击一个订单的“确认参与”按钮。

⑧ 验证人数确认弹窗 (showInputModal) 是否显示。

⑨ 在弹窗中确认人数后点击“确认”。

⑩ 验证:

- SquareScreen 的 handleConfirmCarpool 是否被调用。
- 是否向后端 /orders/apply/{order_id} 发起 POST 请求，请求体包含正确的 passengerCnt。
- 申请成功后，是否弹出“申请成功”提示，并关闭弹窗。
- （可选）切换到“已申请” Tab，验证该订单状态是否为“申请中”。

本过程的关键集成点如下：

HomeScreen (TabContent) 表单交互 -> /orders/match API 集成 -> SquareScreen 导航与数据展示 -> SquareScreen 订单交互与 /orders/apply/{order_id} API 集成。

（2）test_Passenger_View_Applied_And_Created_Orders:

测试用户在 SquareScreen 中切换到“已申请”和“已发起” Tab 时，是否能正确加载并显示相应的订单列表。详细步骤如下：

① 确保用户已登录并有一些已申请或已发起的订单。

② 导航到 SquareScreen。

③ 模拟点击“已申请” Tab。

④ 验证:

- fetchAppliedOrders 是否被调用，并向后端 /orders/fetch_applied 发起 GET 请求。

- 订单列表是否根据后端返回数据正确渲染，订单状态和操作按钮是否正确。

⑤ 模拟点击“已发起” Tab。

⑥ 验证:

- fetchCreatedOrders 是否被调用，并向后端 /orders/fetch_created 发起 GET 请求。
- 订单列表是否根据后端返回数据正确渲染，订单状态和操作按钮是否正确。

本过程的关键集成点如下：

SquareScreen Tab 切换逻辑与对应的后端 API 集成 (/orders/fetch_applied, /orders/fetch_created)。

- **场景 3: 查看拼车详情与地图路线 (依赖用户已登录)**

（1）test_View_Carpool_Order_Detail_And_Route_On_Map：

测试用户从 SquareScreen 点击订单的“查看路线”或“查看详情”后，能否在 SquareDetailScreen 正确显示订单详情、地图路线以及相关操作。详细步骤如下：

查看路线的详细步骤：

① 确保用户已登录，导航到 SquareScreen 的“可拼单” Tab。

② 模拟点击一个订单的“查看路线”按钮。

③ 验证导航到 SquareDetailScreen，并传递了正确的 order_id 和 type='route'。

④ 在 SquareDetailScreen 中:

⑤ 验证:

- fetchOrderDetails 是否被调用以获取订单详情。
- 获取订单详情后，handleRoutePlanning 是否被调用以获取路线规划数据。
- headerTitle 是否为“查看路线”。
- MapScreen 组件是否正确加载并显示了规划的路线、起点和终点标记。
- RouteInfoDisplay 组件是否显示了正确的路线信息、时间、价格等。
- “暂不参与”和“参与拼车”按钮是否可见。

⑥ 模拟点击“参与拼车”按钮，并完成人数确认。

⑦ 验证:

- 是否成功向后端 /orders/apply/{order_id} 发起请求。
- 成功后是否返回 SquareScreen。

查看订单详情的步骤：

① 确保用户已登录，导航到 SquareScreen 的“已发起”或“已申请”（且已通过）的订单。

② 模拟点击订单的“查看详情”按钮。

③ 验证导航到 SquareDetailScreen，并传递了正确的 order_id 和 type='order'。

④ 在 SquareDetailScreen 中:

⑤ 验证:

- fetchOrderDetails 和 handleRoutePlanning 是否被调用。
- headerTitle 是否为“拼车进行中”或“拼车已结束”。
- OrderInfoDisplay 是否显示订单的起终点、价格、发车时间。
- MapScreen 是否显示路线。
- OperationCard 是否加载并显示了订单成员信息。
- 如果 headerTitle 为“拼车进行中”，群聊图标是否显示。

本过程的关键集成点如下：

SquareScreen -> SquareDetailScreen 导航与参数传递，SquareDetailScreen 内部多个 API 调用 (/orders/{order_id}, /route_planning, /orders/{order_id}/members) 的协同工作，MapScreen, RouteInfoDisplay, OrderInfoDisplay, OperationCard 组件

9.5 确认测试

本次确认测试的目的是从最终用户的角度验证整个系统是否满足了最初定义的需求和业务目标。它通常是在一个接近生产的环境中进行的，关注的是用户能否顺利完成其核心任务，以及系统是否按预期工作。在进行确认测试之前，需进行如下的测试环境准备：

设备	至少一台 Android 真机或高质量模拟器/仿真器
网络	模拟不同网络条件（良好 Wi-Fi, 4G, 较差网络）
后端	确保后端 API 服务稳定运行，并有可用的测试数据（例如，一些已注册用户，一些已发布的拼车订单）
高德地图服务	确保高德地图 API Key 有效且服务可用
测试账户	准备多个测试账户，分别扮演乘客和司机角色

在本次确认测试中，我们通过构建不同的用户故事，集成测试各个板块的功

能，从而完成了对整个手机私家车拼车软件的集成测试。

- **用户故事 1：作为一名乘客，我希望能方便地搜索和筛选拼车行程，找到合适的拼车伙伴。**

  AT_Passenger_SearchAndFilterTrips （乘客搜索拼车行程）：

➢ 验收标准：

- 乘客可以在主界面 (HomeScreen 的 TabContent) 输入起点、终点、期望出发时间、人数和可接受的价格范围。
- 点击起点/终点输入框可以进入地点搜索界面 (SearchScreen)。
- 在 SearchScreen 中可以输入关键词获得地点建议，并能选择一个地点返回。
- SearchScreen 能正确显示和管理搜索历史及常用地点。
- 填写完拼车需求后，点击“参与拼车”能触发订单匹配，并跳转到拼车广场 (SquareScreen) 显示匹配结果。
- SquareScreen 能根据匹配度（如果后端支持）或默认排序显示订单列表。

➢ 测试步骤：

- 以乘客账户登录，进入 HomeScreen。
- 在“我要拼车” Tab，点击起点输入框，进入 SearchScreen。
- 在 SearchScreen 搜索并选择一个起点（如“同济大学嘉定校区”）。（预期：返回 HomeScreen，起点更新）
- 重复前 2 个选择一个终点（如“上海虹桥火车站”）。
- 选择一个期望的出发时间。
- 输入上车人数（如“1”）。
- 调整价格区间。
- 点击“参与拼车”按钮。（预期：导航到 SquareScreen，显示相关订单）
- 返回 SearchScreen，验证刚才选择的地点已加入历史记录。

- **用户故事 2：作为一名司机，我希望能方便地发布我的拼车行程，包括设置路线、出发时间、可载乘客数和车辆信息。**

  AT_Driver_PublishTrip （司机发布拼车行程）：

➢ 验收标准:

- 司机可以在主界面 (HomeScreen 的 TabContent) 切换到“我要发车”模式。
- 司机可以输入起点、终点、计划出发时间、乘客数上限。
- 司机可以从“我的座驾”中选择一辆已添加的车辆，或添加新车辆。
- 点击“发起拼车”后，系统能规划路线并在 RouteInfoDisplay 中展示预计

的距离、耗时、总价。

- 司机确认路线信息后，可以成功“发起订单”。

- 发布成功后，司机收到成功提示，并能在 SquareScreen 的“已发起” Tab 中查看到自己发布的订单。

➢ 测试步骤:

- 以司机账户登录（确保该账户已添加至少一辆车，或现场添加）。

- 在 HomeScreen 切换到“我要发车” Tab。

- 选择起点、终点、出发时间。

- 输入乘客数上限（如“3”）。

- 点击座驾选择框，导航到 MyVehicleScreen，选择一辆车后返回。（或先导航到 AddVehicleScreen 添加车辆后再选择）

- 点击“发起拼车”按钮。（预期：显示 RouteInfoDisplay，包含路线规划信息）

- 在 RouteInfoDisplay 确认信息无误后，点击“发起订单”。（预期：发车成功提示，导航到 SquareScreen）

- 在 SquareScreen 切换到“已发起” Tab。（预期：能看到刚才发布的订单，状态为“进行中”）

- **用户故事 3: 作为一名司机，我希望能管理我发布的行程，查看乘客的申请，并能接受或拒绝申请。**

AT_Driver_ManageApplications （司机管理乘客申请）：

➢ 验收标准：

- 当有乘客申请加入司机发布的行程时，司机能收到通知（本期可能简化为在订单详情中查看）。

- 司机可以在其“已发起”订单的详情页 (SquareDetailScreen，type='order') 中看到申请列表（在 OperationCard 中显示新申请者头像）。

- 司机可以接受或拒绝乘客的申请。

- 接受申请后，该乘客成为正式乘员，订单的已确认人数更新。

- 拒绝申请后，该乘客申请状态变为“未通过”。

- 乘客端能看到其申请状态的变化（“已通过”或“未通过”）。

➢ 测试步骤 (需要两个账户配合，一个司机，一个乘客)：

- 司机账户发布一个拼车订单 (如 AT_Driver_PublishTrip 中所述)。

- 乘客账户找到该订单并申请加入 (如 AT_Passenger_ViewTripDetailAndApply 中所述)。

- 司机账户进入 SquareScreen 的“已发起” Tab，找到该订单，点击“查看详情”。

- 在 SquareDetailScreen 的 OperationCard 中验证是否能看到乘客头像。
- 乘客账户刷新“已申请”列表或订单详情。
- 司机账户再次查看订单详情。
- 司机拒绝另一个乘客的申请，验证乘客端状态。

- **用户故事 4：作为一名用户，我希望能管理我的车辆信息 (如果我是司机)。**

  AT_User_ManageVehicles （用户管理车辆）：

➢ 验收标准：

- 用户可以进入“我的座驾”页面 (MyVehicleScreen)。
- 用户可以看到已添加的车辆列表。
- 用户可以成功添加新的车辆信息，包括车型、车牌号、油耗和品牌。
- 车牌号和油耗有基本的格式验证。
- 用户可以删除已添加的车辆。

➢ 测试步骤：

- 以一个账户登录，导航到 MyVehicleScreen 。
- 点击“新增车辆”按钮，进入 AddVehicleScreen。
- 输入有效的车辆信息，点击“添加”。
- 尝试添加车辆时不输入车牌号或输入无效格式的车牌号/油耗。
- 在 MyVehicleScreen 选择一辆已添加的车辆，在弹出的操作中选择“删除座驾“。

- **用户故事 5: 作为一名用户，我希望能对已完成的行程进行评价 (司机评价乘客，乘客评价司机)。**

  AT_User_RateTrip （用户评价行程）：

➢ 验收标准：

- 当一个拼车行程结束后（订单状态变为“已结束”）。
- 司机和乘客都可以在相应的订单列表项（如 SquareScreen 的“已发起”或“已申请”）中找到“填写评价”的入口。
- 用户可以对对方进行评分和文字评价。
- 提交评价后，评价内容被保存。
- 用户的平均评分能被更新。

➢ 测试步骤 (高度依赖评价功能的具体实现)：

- 确保一个订单已完成。
- 乘客账户找到该已结束订单，点击“填写评价”。
- (进入评价页面) 给出评分和评价内容，提交。
- 司机账户找到该已结束订单，点击“填写评价”。

- (进入评价页面) 给出评分和评价内容，提交。
- 查看司机/乘客的个人资料，验证评分是否有所体现。

十、效果展示

10.1 运行说明

项目运行需要确保配置 React Native 和 Android 原生的运行环境。从终端打开项目根目录，首先使用“yarn install”命令安装本软件运行所需的全部依赖工具。在所有依赖安装成功后，使用“yarn android”启动 Android 项目，等待一段加载时间后，可以看到模拟器启动打开。

```
D:\Desktop\PinTu-APP>yarn android
yarn run v1.22.22
$ react-native run-android
warn Package react-native-amap3d contains invalid configuration: "dependency.platforms.ios.project" is not allowed
. Please verify it's properly linked using "npx react-native config" command and contact the package maintainers a
bout this.
* daemon not running; starting now at tcp:5037
* daemon started successfully
info Launching emulator...
info Successfully launched emulator.
info Installing the app...
Starting a Gradle Daemon, 1 incompatible and 1 stopped Daemons could not be reused, use --status for details
<=============> 100% CONFIGURING [46s]
> Resolve dependencies of :app:debugCompileClasspath > Resolve dependencies of :app:debugRuntimeClasspath
```

图 28 软件前端启动方法

在同一项目根目录下，输入“python post-end.py”启用后端服务器，等待 1~2 秒后，服务器启动并正常运行，可以开始使用软件。

```
D:\Desktop\PinTu-APP>python post-end.py
 * Serving Flask app "post-end" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 191-687-689
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

图 29 软件后端启动方法

10.2 功能展示

（1）登录注册功能：

软件初次加载将显示欢迎页面，单击“立即体验”进行登录注册。在注册界面输入邮箱账号，后端将检查该邮箱的合法性，若邮箱不合法软件将给出相关提

示（如左下图所示）；邮箱合法则后端向该邮箱发送确认邮件，用户注册成功，系统自动跳转登录页面。

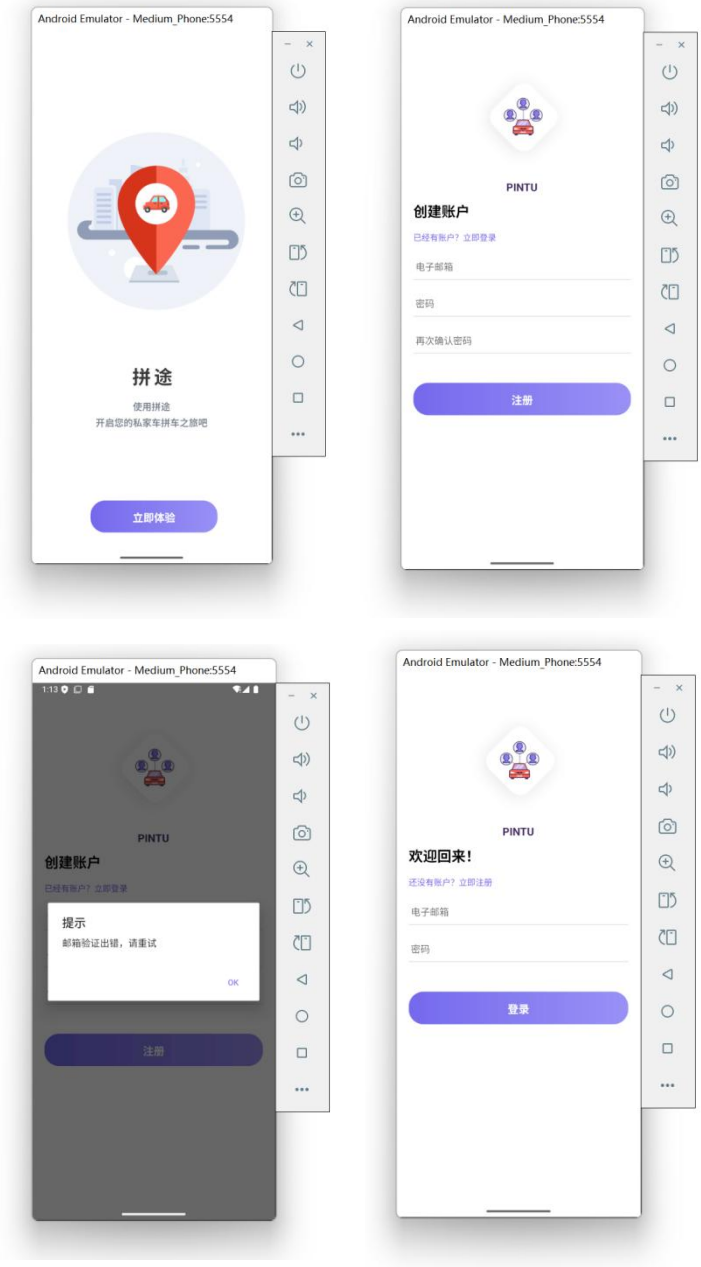

图 30 登录注册页面功能展示

（2）发车功能：

登录后将直接进入主页面，在该页面会先询问用户获取当前位置，高德地图实时定位并渲染在界面底部。下方悬浮操作面板可供用户选择“我要发车”和“我要拼车”。选择“我要拼车”按钮，填入表单信息。

“选择起点”和“选择终点”两个表单项被点击后，将自动跳转搜索界面。

用户可以搜索地点名称，搜索结果将按照距离用户当前位置的距离升序排列。直接点击搜索结果，该数据将自动回传到主界面并完成表单的填入。

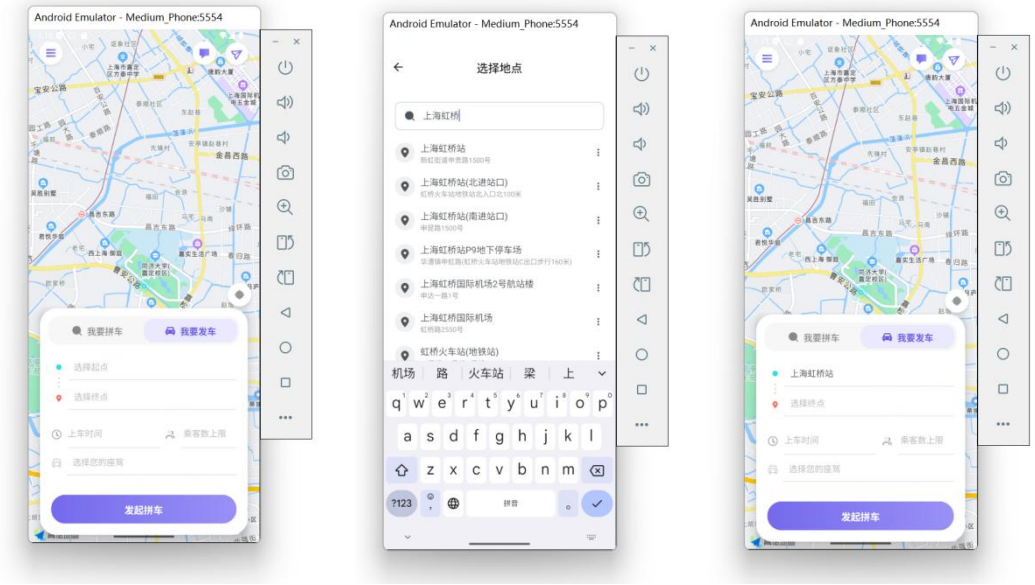

图 31 发车起终点搜索和确认功能

在搜索界面用户可以查看并删除历史记录，同时可以添加或移除常用列表：

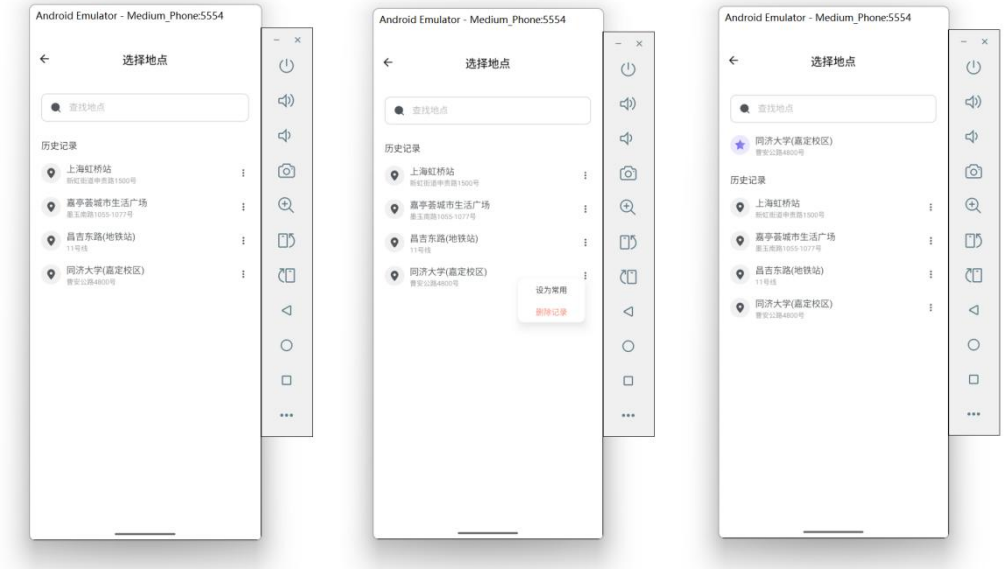

图 32 搜索页面功能展示

继续填写表单，完成发车时间选择、拼车乘客数上限的设置和座驾选择：

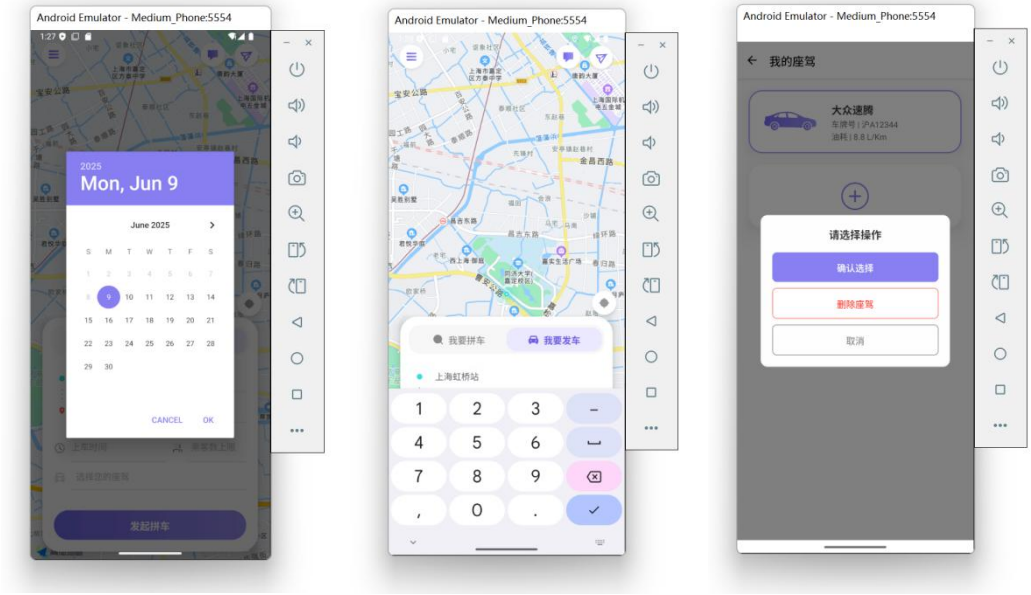

图 33 发车表单其他选项设置

在所有订单选项都被设置完成后，司机用户可以点击最下方的“发起拼车”按钮。若有任意表单项被设置或不合法，都将弹窗提示；若所有表单项合法，系统将自动为本订单进行路径规划、计算预估行程时间和订单价格。

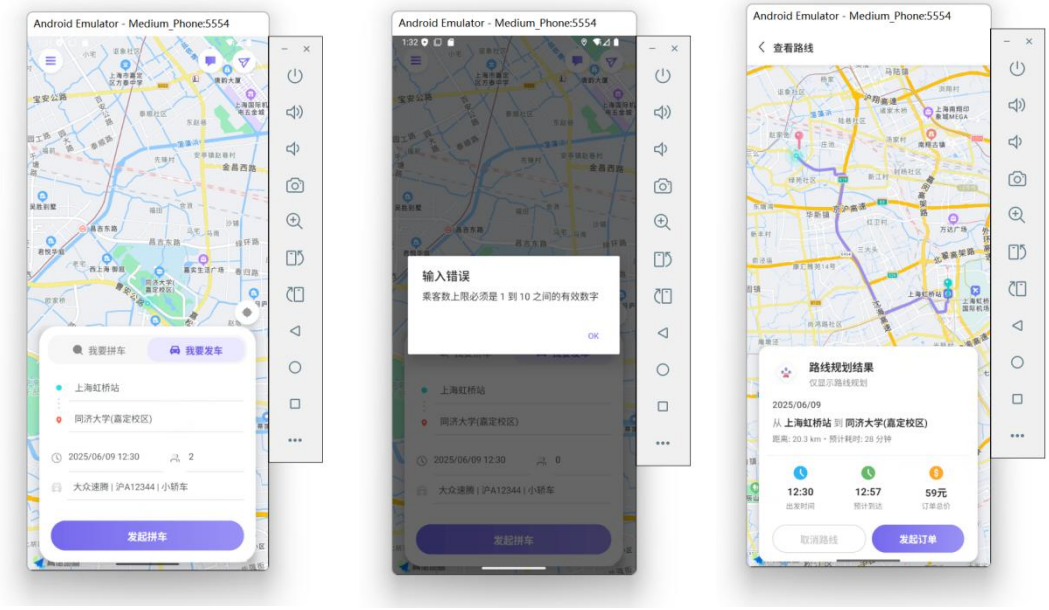

图 34 发车表单检查与路径规划显示功能

此后，若司机后悔想要取消路线，可以单击“取消路线”按钮，软件将回退到初始主页面；若单击“发起订单”，该订单将被创建并显示到“拼途广场”等待乘客申请加入。

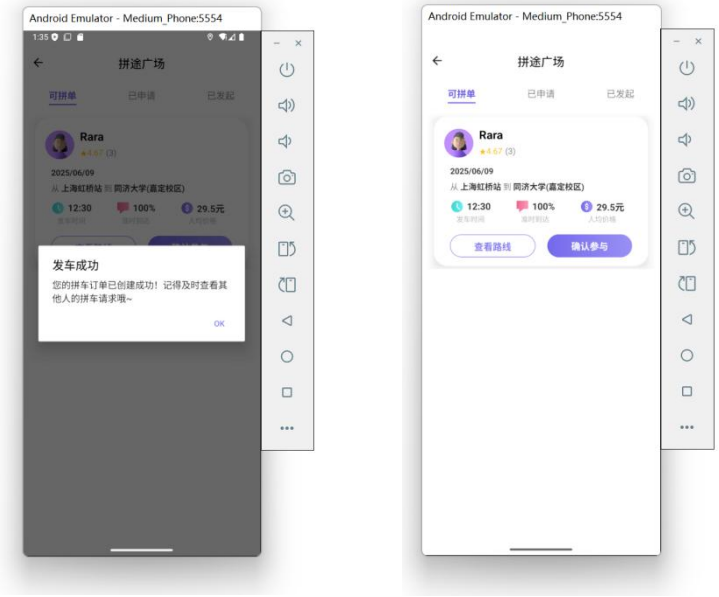

图 35 订单创建功能

（3）拼车功能：

拼车功能的前半部分流程与发车功能一致，用户可以通过在主页面输入起终点、期望上车时间和期望价格区间来发起订单查询（此处我们填入的表单项和发车中的不完全一致，略有偏差，目的是为了验证后端订单智能匹配的功能），若有匹配订单，软件系统将直接跳转“拼途广场”将合适的订单展示给用户：

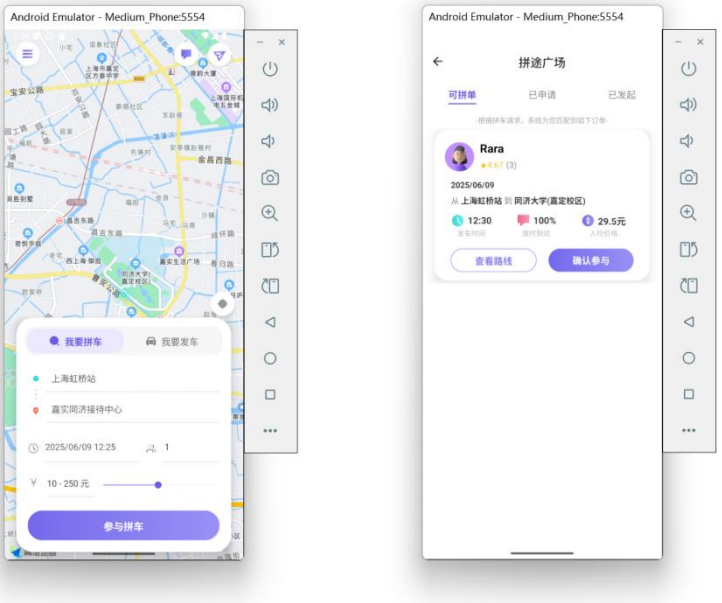

图 36 拼车订单搜索与匹配功能

（4）申请拼车功能：

进入“拼途广场”可以查看所有可拼单。单击“查看路线”，可以看到本订单的详细信息；单击“参与拼车”，输入订单人数后点击确认，正式申请本单。

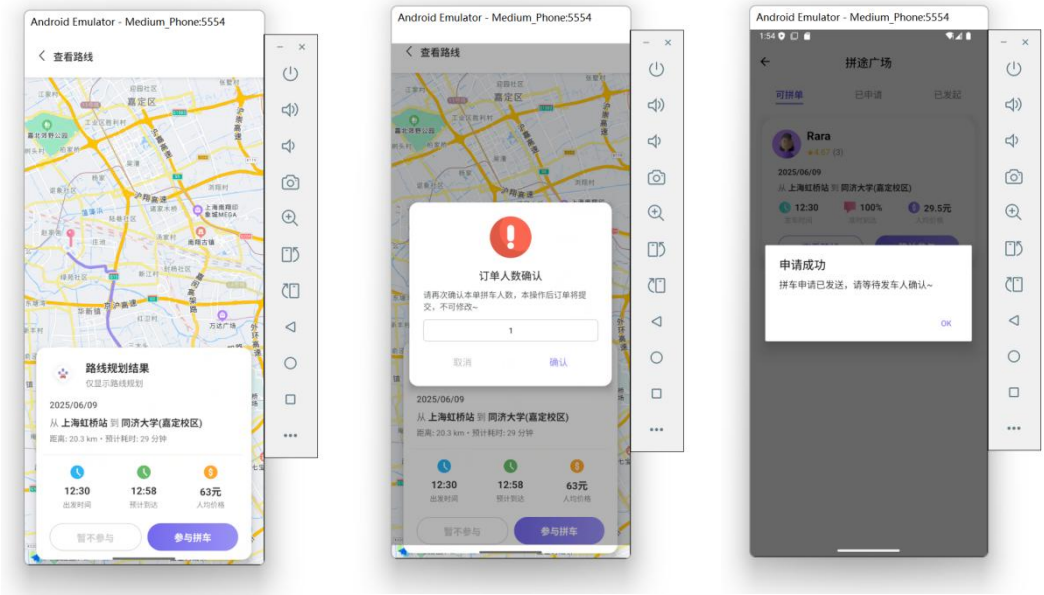

图 37 乘客申请拼单功能

（5）审批与行程控制功能：

在“拼途广场”的另外两个“已申请”和“已发起”tab 中，有我作为乘客申请的订单状态，和我作为司机创建的订单状态，并可以分别进入查看订单详情。

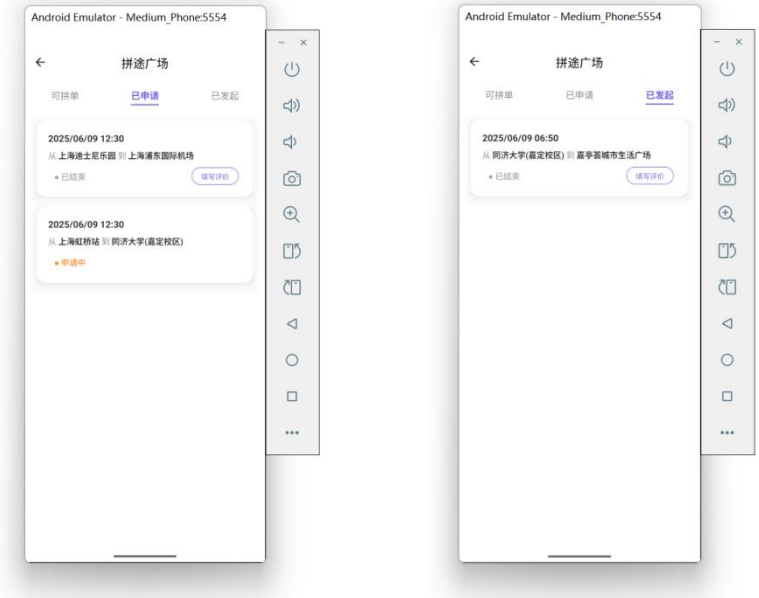

图 38 “已申请”和“已发起”订单查看功能

作为司机，单击自己“已发起”的订单卡片详情，可以查看当前订单是否有乘客用户申请（若有新乘客申请，右侧的紫色加号上方将出现红色点），并进行相关审批。

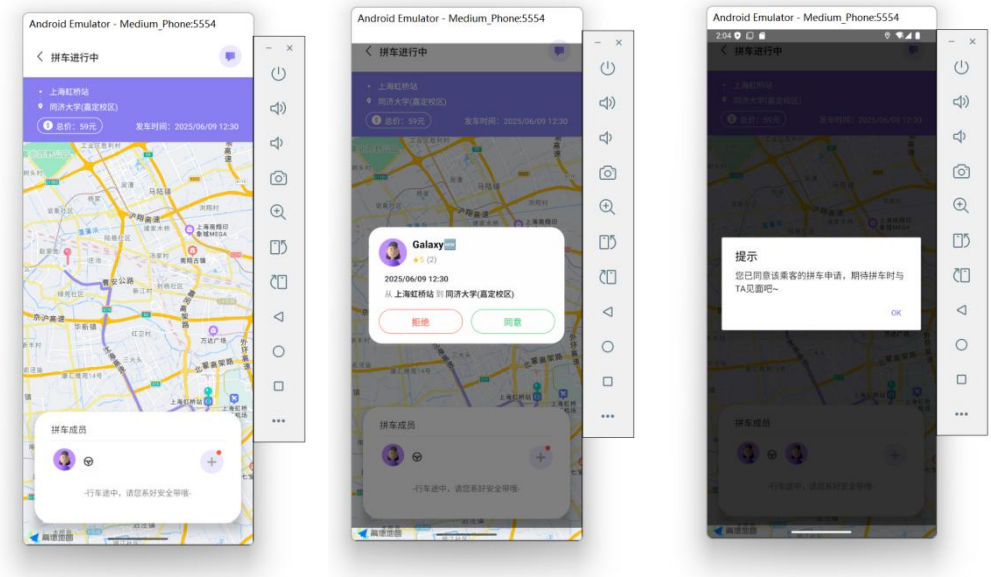

图 39 司机审批乘客拼车申请功能

完成审批后，本订单所有成员被正式确定。待正式行程开始时，司机可以进行行程管理的相关控制，包括单击某一乘客的头像确认该乘客已上车（此操作将用于计算乘客的准时率，纳入用户评分系统），以及单击自己的头像确认行程开始和结束（此操作将用于计算司机的准时率，同样纳入用户评分系统）。

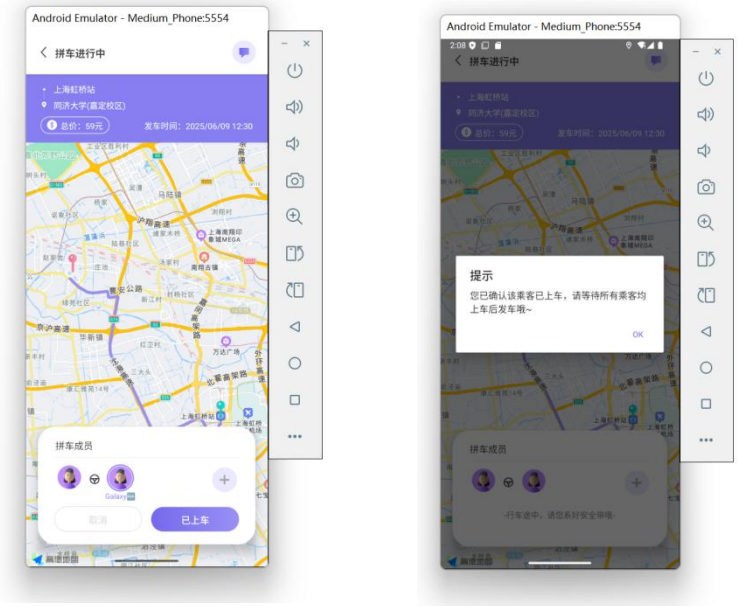

图 40 司机确认乘客上车功能

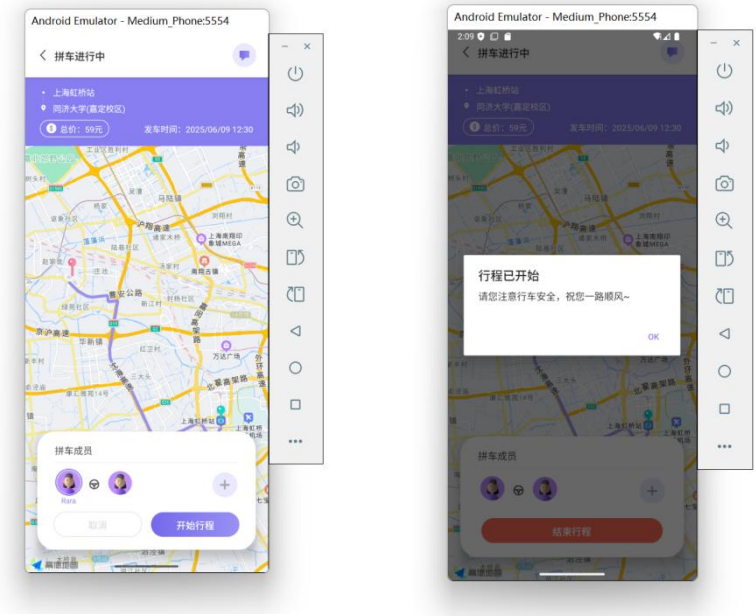

图 41 司机确认行程开始功能

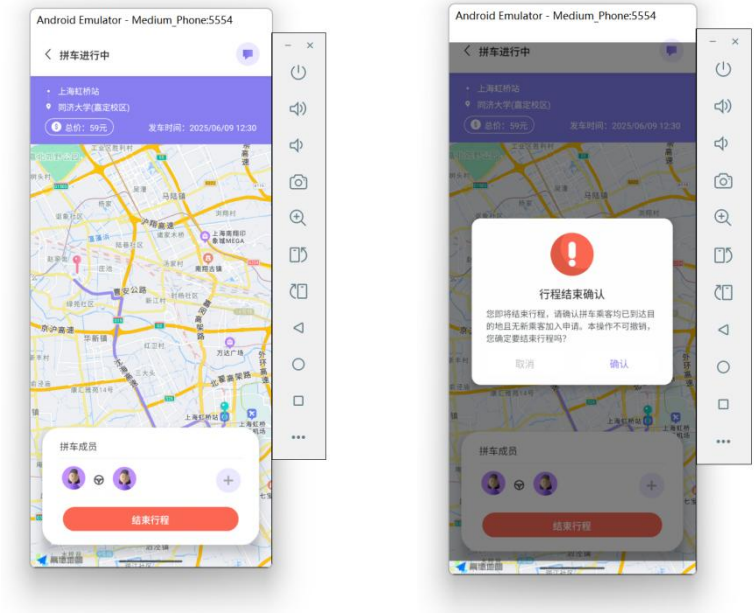

图 42 司机确认行程结束功能

作为乘客，在本页面没有过多的操作，只可以通过单击每个用户的头像查看其用户名和个人主页。

（6）评分功能：

在订单结束之后，司机视角将自动跳转评分页面，为除了自己以外的所有乘客用户打分，此操作将同样纳入用户评分系统。

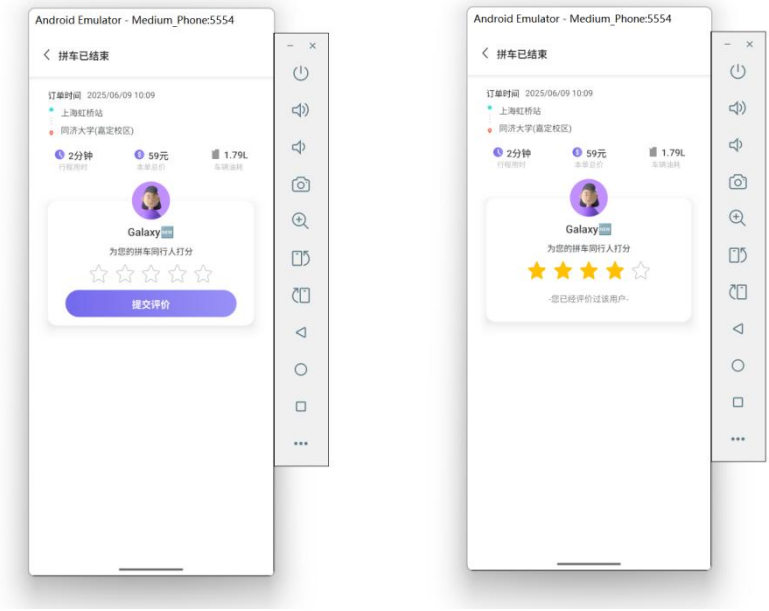

图 43 用户评分功能

作为乘客，可以通过从“拼途广场”的“已申请”tab 下查找到已结束的订单，从左侧进入“填写评价”页面，同样可以进行此操作，用户也将为除了自己以外的其他乘客（拼车同行人）和司机用户（拼车发起人）打分。

（7）群聊功能：

从软件的主界面右上角图标按钮、或单击左侧的抽屉式菜单栏、或在行程进行中从行驶详情的右上角图标入口，均可以进去我的群聊界面。

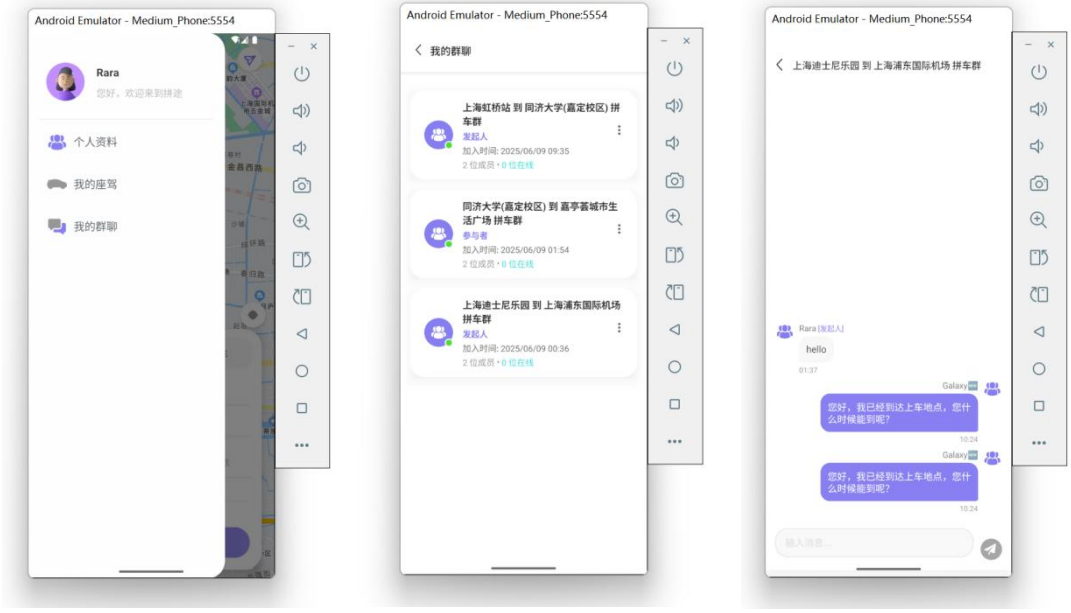

图 44 群聊多入口与消息发送功能

此外，在我的群聊主界面，可以进行清空聊天记录和删除群聊的功能：

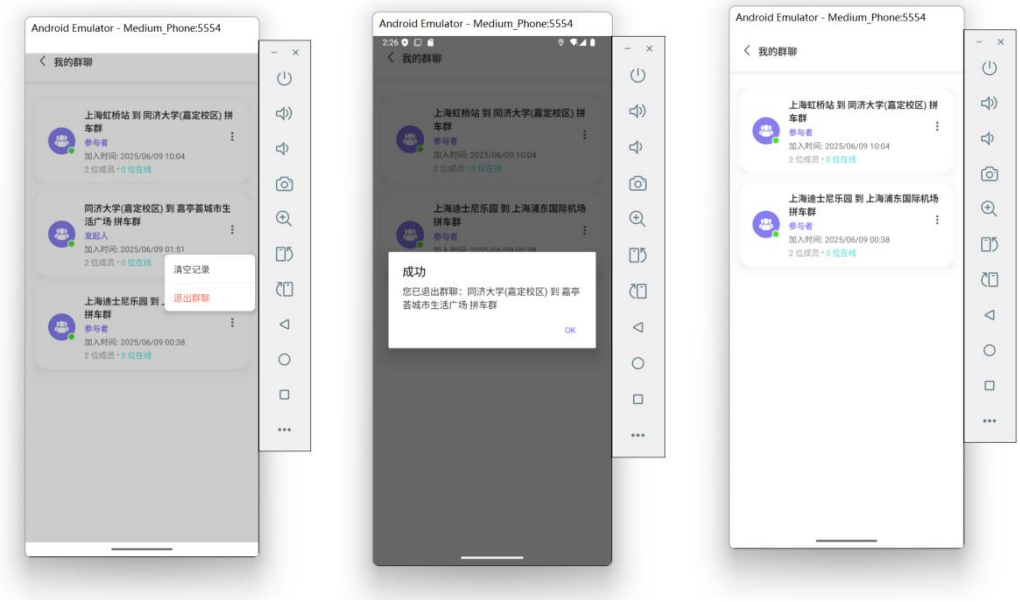

图 45 清空群聊聊天记录和删除群聊功能

（8）座驾管理功能：

从主界面左侧抽屉式菜单还可以进入“座驾中心”进行自己的座驾管理，包括增加和删除座驾，编辑座驾的车牌号、油耗、车型、品牌等多种信息。本模块同样包括丰富的合法性检查，避免脏数据的注入。

例如，单击底部加号添加座驾，将按正则表达式的方法匹配车牌号的合法性和唯一性，若不合法将弹窗提示无法添加，系统拒绝本次操作。

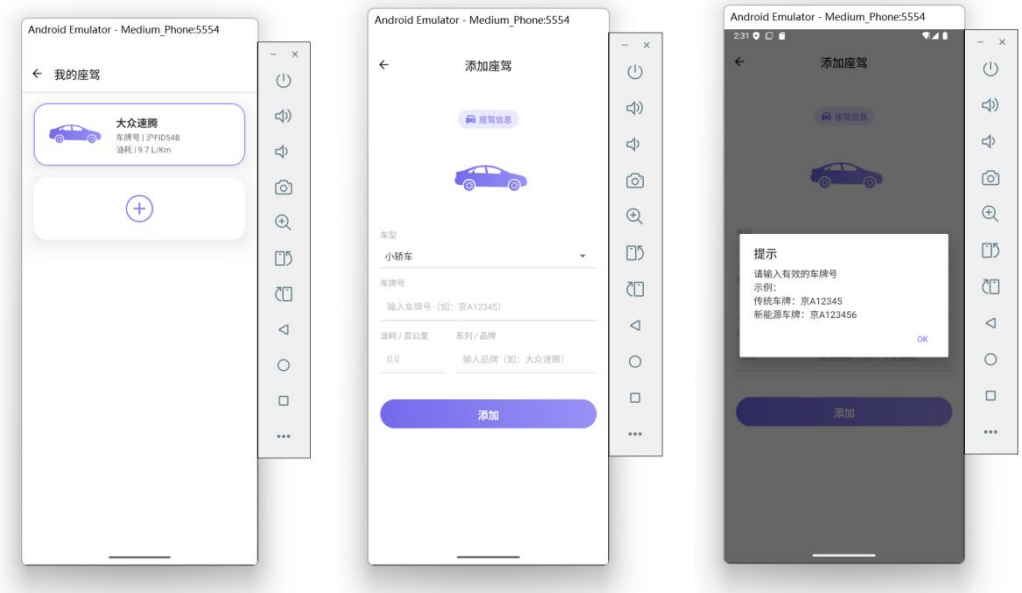

图 46 座驾中心车辆管理功能

（9）个人中心功能：

从主界面左侧抽屉式菜单还可以进入“个人中心”对个人资料进行管理，包括用户昵称、电话、邮箱的修改，以及用户头像的更改和驾驶执照图片的上传。

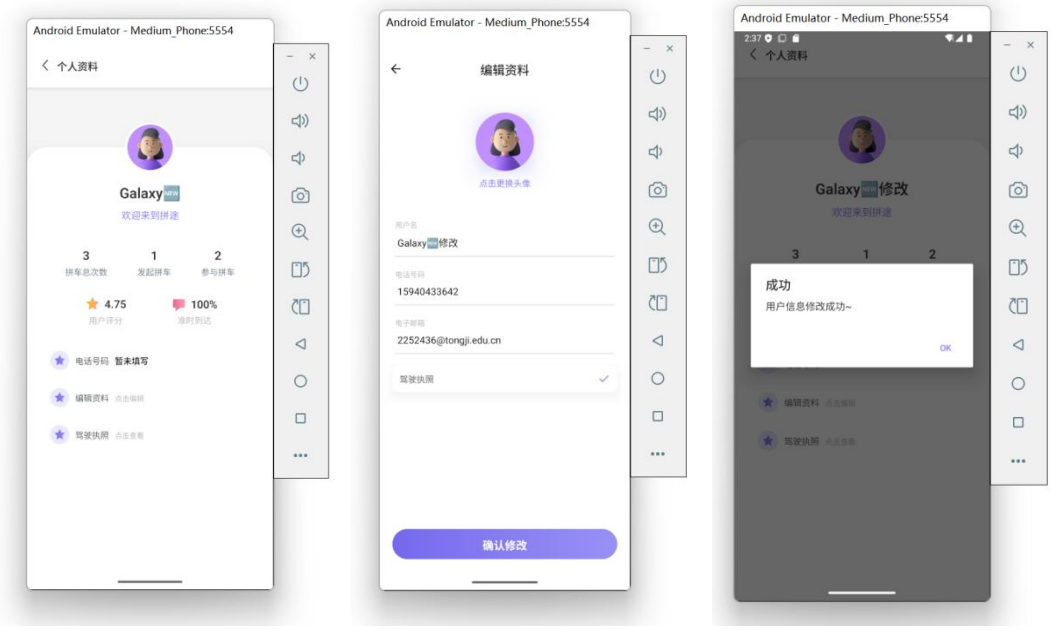

图 47 用户信息管理功能

十一、项目管理

在本次课程设计中，小组成员一直秉持着团结高效的合作精神，积极使用课堂所学的项目管理和团队管理开展项目，确保项目在每个阶段按时交付。下图所示为本项目的甘特图，记录整个软件项目从开始到结束的生命周期：

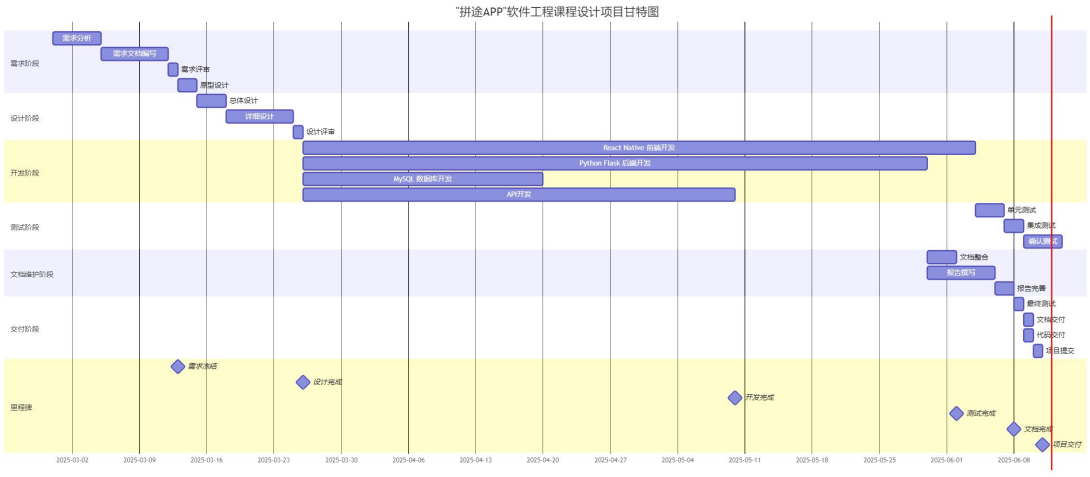

图 48 本软件项目管理甘特图

十二、心得体会

-

在本次项目中，我主要负责软件在需求分析部分的原型设计、部分主界面前端实现，以及包括搜索、拼途广场、订单详情、用户评价和拼车全流程功能的前后端代码实现，同时还承担了部分文档的编写与维护工作。通过和小组成员们的通力合作，我很高兴自己能够有这样一次从需求分析到项目落地的软件开发流程体验。我们在项目开发过程中，一直力求按照上学期《软件工程》这门课程所讲述的原理和方法进行，以期完成一个符合专业规范的软件。在完成软件的过程中，我学习并掌握了 React Native 框架和 API 集成方法，并懂得了如何让高效利用 git 工具管理代码仓库，这些都使我在专业技能上受益匪浅。

此外，小组成员们积极采用线上和线下的方式沟通开会，让我感受到了在一个优秀的团队中协作是多么的高效和有成就感。感谢我的组员们，感谢我的拼途路上有这样 4 位可爱的同行人。

-

在“拼途”APP 项目中，我从需求分析、原型绘制；到项目框架环境配置、按照软件生命周期流程进行项目开发；到座驾功能前后端代码编写、实习报告撰写全流程积极参与，走过一段规范而默契的团队开发旅程，收获颇丰。

学期初，团队首次集结，在线下会议中快速敲定了项目开发流程，提出先进行需求分析和原型设计。我通过网络资料搜集对手机私家车拼车系统进行了详尽的需求分析，然后和团队伙伴共同完成了原型设计。这一环节让我体会到“原型设计”的重要性，原型可以作为 UI 设计的模板和交互跳转的演示，在团队中很好的指导后续项目开发。最终我们的代码编写成功实现了对原型的样式复现，打造风格统一、清晰简洁的界面设计。

接下来，我们遵循上学期软件工程理论课程所学，按照软件的生命周期进行项目开发，依次进行总体设计、详细设计、编码和测试。团队采用模块化分工，按照功能板块分别认领任务，大家绘制流程图进行功能梳理；绘制用例图、顺序图等进行 UML 建模分析；团队共同制定编码规范；各自完成对应功能的前后端代码编写；最后各由一位同学负责项目代码整合、软件测试工作和实习报告整合。我负责了实习报告的整合，回顾开发全流程，深深体会到规范的团队开发带来的效率和质量上的双重优势。

至此，我们的“拼途”项目合作进入尾声，感谢与队友的相遇，让我获得了

很好的前后端代码编写和团队合作开发体验！

● :

在本次项目中，我主要负责了部分主界面的 UI 实现、核心的路径规划与地图定位功能的集成，以及点击“参与拼车”后的交互界面设计与开发。此外，我还承担了整个项目的软件测试工作。这些任务让我对一个完整软件从设计、开发到测试上线的全流程有了更深刻的理解。

主界面和交互界面的开发让我更加熟练地运用 React Native 进行跨平台应用构建。如何设计出既美观又易用的用户界面，如何处理组件间的状态共享与通信，都是我在实践中不断摸索和学习的。特别是路径规划和地图定位功能的实现，涉及到与高德地图 API 的深度集成。从 API 的申请、SDK 的引入，到异步请求的处理、地图数据的解析与展示，再到用户位置的实时获取与路线的动态规划，每一步都充满了挑战。我学会了查阅官方文档、调试 API 接口、处理各种边界情况和异常，这个过程极大地锻炼了我解决实际问题的能力。

软件测试部分则让我深刻体会到“质量是设计和构建出来的，也是测试出来的”。我从测试需求分析入手，设计了包括单元测试、集成测试和系统测试在内的多层次测试用例。在单元测试阶段，我学习并实践了使用 Jest 对 JavaScript 逻辑和 React Native 组件进行测试，关注代码的语句和分支覆盖；在集成测试中，重点验证了地图 API 调用、组件间交互的正确性；系统测试则模拟真实用户场景，对应用的整体功能和用户体验进行了全面检验。这个过程不仅帮助团队发现并修复了许多潜在的 Bug，也让我对软件质量保障的重要性有了更切身的体会。

总而言之，这次课程设计是一次宝贵的实践经历。它不仅让我将所学知识应用于实际，提升了技术硬实力，更重要的是培养了我的工程思维、团队协作能力和面对挑战解决问题的勇气。这些经验和感悟，无疑将对我未来的学习和工作产生深远的影响。

● :

在本次项目中，我主要负责拼车群聊前后端的代码实现以及对报告群聊部分内容的撰写、以及相关的需求分析和文档维护工作。通过这次团队合作，我收益颇丰，团队成员都很优秀，为人热心并且自愿承担更多的任务，虽然我负责的内容较少，但组长仍然耐心地帮助我解决运行、开发过程中遇到的各种问题，让我能够顺利写出我负责的内容。同时这也是我第一次参与了完整软件从零开始地设计开发任务，这对于以前的我是遥不可及的，但在一个优秀的团队中，这些艰难的任务也能够顺利完成，或许这就是团队的魅力。此外我对于一个项目如何创建

到团队分工的过程都有了更深刻的理解，并且团队的各个成员如何各司其职而又不会互相冲突也有了实质性的理解。

总之，通过这个大作业，不仅让我学习到了实践性的专业知识和解决问题的思考角度，更是让我学会了一个团队如何沟通、分工、协作，我坚信这对于未来的我弥足珍贵。

- ：

在本次“手机私家车拼车软件系统”的的开发过程中，我主要杜泽个人信息界面，实现了一个功能较为完备的用户信息管理模块。该模块涵盖了头像、昵称、等字段的展示与编辑，是提升用户体验的重要一环。

前端部分采用了 React Native 结合 TypeScript 的技术栈，在开发过程中我加深了对组件化设计和类型安全的理解，能够更加规范地构建界面结构，提升了项目的可维护性与代码质量。同时，通过处理本地资源（如默认头像）与远程数据（如 region.json）的动态绑定，我进一步掌握了前后端数据交互的流程，并在用户输入方面注重细节优化，以提升整体交互体验。

后端方面，我使用 Python 编写了用于接收与提交用户信息的脚本，深入了解了 HTTP 请求的处理机制与接口定义规范，也强化了我对数据安全性与接口容错性的思考与实践。

除了功能实现，环境配置也是本次项目中让我印象深刻的一部分。从在 Android Studio 中创建并运行模拟器，到在 PyCharm 中配置 Python 环境、启动后端服务，再到前端的本地启动调试，这一过程涉及多个工具与平台的协同。虽然配置环境耗费了不少时间，但也极大提升了我对开发环境搭建与跨平台联调的实际能力。

此外，通过这个界面功能的实现，我也意识到一个良好的代码结构和注释规范对于团队协作与后期维护的必要性。整个开发过程让我更加注重代码的可读性与可扩展性，也让我对完整项目开发流程有了更系统的理解。

整体而言，这次开发经历不仅让我提升了编程技能，更增强了实际项目部署、调试与运行的全流程能力。我深刻体会到，一个好的产品不仅需要良好的功能设计，还需要稳定的运行环境和流畅的用户体验。在今后的学习与实践中，我将继续在技术深度与工程完整性之间不断探索与成长。