软件体系结构文档

文档名称：《“即行”——城市打车系统体系结构文档》

更改记录

日期	修改章节	修改类型*	修改描述	修改人	版本
2019/12/15	全部	A	编写文档初稿	王梓彧	1.0
2019/12/17	第三章	A	删除重复的图片，使用引用的方式代替	王梓彧	2.0
2019/12/20	第三章	D	删除了一些不必要的需求描述（需求描述详见之前写过的需求分析文档）	王梓彧	2.0
2019/12/20	第三章	M	对一些不准确的图和描述进行了修正	王梓彧	2.0

*修改类型分为 **A** - ADDED **M** - MODIFIED **D** – DELETED

职责明确

计划编写人员签名： 王梓彧 2017211900

计划审核人员签名：

计划批准人员签名：

目录

第一章 引言

随着城市化进程的快速发展，打车难、打车慢的问题变得日益突出，同时，移动互联网和智能终端的高速发展为利用打车软件解决该问题提供了机遇。为了解决城市交通在原有出租车系统的情况下产生的空闲时出租车调度不便和高峰时负载压力大等问题，我们设计一个城市打车系统，一方面使市民享受到便捷的打车服务，另一方面使司机能避免“空载”，更快地接单、接最近的单，实现更高的创收，同时还能根据交通的实时状态为司机规划路线，降低城市道路交通的压力。

1.1.编写目的

1. 简述阐述本系统的体系结构模式、设计风格和框架，方便用户了解
2. 实现和部署 *需求文档** 中所指定的需求目标，对实现的过程和方法进行记录和统一
3. 对日后的部署起到指导和规范化的作用，防止项目开发中后期出现蠕变或者失去方向

**需求文档详述请见《“即行”——城市打车系统需求分析文档》*

1.2.项目范围

本系统为城市打车系统，主要服务于有出行打车需求的乘客，以及有空闲车辆、希望在闲时赚取外快的司机。

1.3.定义、术语和缩写语

名称	定义
城市打车系统/系统	本文档负责的软件系统
司机	通过本系统合法检验、注册本系统并使用的司机
乘客	拥有并注册使用本系统的乘客
系统管理员	在使用本系统的管理此系统的公司中的专业人员，拥有最大权限
接客路线	从司机接到乘客发起的打车消息到乘客所在位置的路径规划
送客路线	从乘客上车位置到乘客目的地的路径规划
司机公开信息	司机的性别，姓，车牌号，车型，驾龄，星级等

1.4.参考文献

1. 《软件工程化》，王安生著，清华大学出版社，2014年4月第一版
2. 《UML和模式应用》，Craig Larman 著，机械工业出版社，原书第三版

1.5.综述

本文档共分为五个章节，至此第一章节引言部分结束。

第二章为需求简述，用于记录需要实现的需求

第三章为解决方案，用于规范本项目使用的框架、设计模式以及开发风格

第四章为场景和风险分析，用于确定在当前架构下需要明确的场景需求和高风险项

第五章为用户界面设计，在UI表现层上进行原型设计，通过原型来引导项目开发（如此可以尽快得到可执行成果）

第二章 需求简述*

**该栏目只简要记录设计过程中值得关注的需求，其他详述需求请见需求文档《“即行”——城市打车系统需求分析文档》*

2.1.关键指标

经过从各个用户类型对系统需求的粗略分析，该系统应具备以下功能：

主要参与者：

- **乘客端**：查看行程、估计行程时间、即时打车、预约打车、寻求拼车、联系司机
- **司机端：**接单路线规划、实时路况信息展示、送客路线规划、联系乘客

辅助参与者：

- **第三方支付软件**：支付

此外，本项目未来还可以进行进一步拓展，为其他行业或领域提供信息管理服务，如周边旅游景点路线/时间规划，城内/城际短途跑腿业务规划等。可将其包含于其他系统或项目，并定义相关接口。

2.2.功能性需求简述

2.2.1.用户角色需求

1. 司机：处理订单、查看路况、规划抢单路径、获取报酬
2. 乘客：请求打车、查看实时路况、查看接单司机的基本信息、完成订单之后提交反馈
3. 系统管理员：拥有最高权限，对所有数据（除了第三方提供的数据）进行操控

*2.2.2.系统功能分析**

**直至本次迭代，本文档主要关注司机端和乘客端对系统的功能需求*

> 需求描述的格式遵循：
>
> （<*触发动作*>，）在<*前提条件*>的情况下，系统应<*功能*> <*对象*>；（在<*拓展条件*>的情况下，系统也可<*功能*> <*对象*>）

> （由于是功能性需求，故而不涉及性能）

- ## 司机端

1. **注册账户**：*新司机（未注册的司机）上传自己的合法证件照片和信息进行合法性检测，在检测合法*的情况下，系统应*正确生成一个新的司机账户

2. **登录账户**：略

3. **查看实时路况**：在*司机成功登录*的情况下，系统应*在主页上持续定期更新当前的路况*

4. **预计订单密度**：*司机登录成功后，在没有正在进行的订单*的情况下，系统应*根据历史数据通过地区和时间来对订单密度进行预测，并给出未来一段时间内的订单频率比较高的地区推荐司机前去抢单*

5. **处理订单**

5.1. **接单**：*司机登录成功后，在没有正在进行的订单并且周围有乘客发起打车请求*的情况下，系统应*显示周围合理范围之内的用户发起的打车请求供司机选择*

5.2. **规划接单路线**：司机选择系统提供的打车请求，在*接单成功*的情况下，系统应*持续定期规划并显示接单路线*

5.3. **规划送客路线**：*司机出发接单后，在乘客指定的时间之前接到乘客*的情况下，系统应*持续定期规划并显示送客路线*

6. **获取报酬**：在*有订单完成并且用户完成评价*的情况下，系统应*向司机支付薪资和奖励*

7. **管理个人信息**：略

- ## 乘客端

1. **注册&登录账户**：略

2. **查看实时路况**：在*乘客成功登录*的情况下，系统应*在主页上持续定期更新当前的路况*

3. **请求打车**

3.1. **查看行程及其估计到达的时间**：在*乘客输入自己的目的地*的情况下，系统应*显示出当前路况下的推荐行程，并估计每个行程到达终点大致的时刻*

3.2. **发起打车请求**：*在系统已经获取乘客位置（自动获取或乘客输入）、乘客已经输入自己的目的地并选择确认需要的服务（服务包括即刻打车、预约打车、拼车等）*的情况下，系统应*将其的打车请求加入附近的等待队列*

3.3. **查看司机信息**：在*乘客的打车请求已经被接受*的情况下，系统应*展示出司机的公开信息，并在主页地图上持续更新司机的当前位置，还应提供联系司机的途径（在不泄露司机个人信息的情况下）*

3.4. **支付行程费用**：在*乘客确认行程结束*的情况下，系统应*提醒乘客付款；在乘客绑定有已授权无密码支付的第三方用户*的情况下，系统也可*在乘客绑定的第三方支付应用中直接扣除*

3.5. **评星和评价司机**：在*乘客支付完行程费用*的情况下，系统应*提供使乘客能对司机进行评星评价的页面*

4. **管理个人信息**：略
5. **查看个人历史行程**：略

2.3.各相关方对体系结构的要求

1. 用户：

   功能需求见本文档2.2.1

2. 开发技术人员：

   系统有明确的开发需求，客户不增添和本系统无关的任意功能，不添加不可能实现的任意功能，不在职责之外添加任务，保证规定时间之内任务能够完成。

3. 客户：

   项目需要保证在规定的合同约定的时间内能定时定量地完成进度，保证进度不拖拉，同时在财政预算的方面没有多余的预算。

4. 项目经理：

   保证任务能够及时并合理地进行分解，分发给每一个开发团队独立工作。保证每个团队都能按质按量按时地完成任务。

5. 政府或交通路况数据提供方：

   要保证数据的安全，不向得外出售数据，不得将数据用于恶意操作

系统有明确的开发需求，客户不增添和本系统无关的任意功能，不添加不可能实现的任意功能，不在职责之外添加任务，保证规定时间之内任务能够完成。

3.客户：

项目需要保证在规定的合同约定的时间内能定时定量地完成进度，保证进度不拖拉，同时在财政预算的方面没有多余的预算。

4.项目经理：

保证任务能够及时并合理地进行分解，分发给每一个开发团队独立工作。保证每个团队都能按质按量按时地完成任务。

2.4.约束条件

1. 常用的信息管理系统采用 B/S 开发模式，系统应采用应用开发的基本技术。
2. Web 服务器部署于 Linux 系统之上的系统一般运行稳定性高。
3. 服务器系统硬件的配置能支持服务器高效稳定的运行。
4. 为了系统将来的可扩展性，系统在硬件的使用上需尽可能减少针对性。整个 系统也应尽量减少各模块间的调用，尽量做到松耦合。
5. 数据文件、系统配置文件应当安全可靠的存储。
6. 系统的数据需要具有足够的可靠性，才会保证系统正常运行。

2.5.非功能需求

2.5.1.性能

1. 响应时间：正常负载下，系统响应时间应小于2秒钟。
2. 吞吐量：单位时间（按1秒钟计）内吞吐量应不低于10000
3. 并发用户数：系统支持并发用户数最少为30000。

2.5.2.可靠性

1. 采用面向对象的迭代开发方法；
2. 选用统一的开发工具（例如NetBeans），这一考量主要在于编译器之间有可能发生不兼容的问题；

3. 采用结构化的程序设计方法，最好满足GRASP原则；
4. 经过严格的测试，要求做到语句覆盖100%，条件覆盖100%；
5. 为最有可能出现的风险设置必要的错误处理和错误陷阱；
6. 选择典型的单位试运行。

2.5.3.易用性

- 系统应当具有在线帮助界面等部分，方便用户学习操作。
- 系统应当足够简洁明了，使大部分用户能够自己学会使用本系统。

2.5.4.密安性

- 系统应当能够保证用户信息不泄露，系统配置文件和数据库存储文件应该进行加密处理。

2.5.5.可维护性

- 系统应该能够容易诊断出存在的缺陷和失效原因，容易识别出待修改部分的可能性或能力。
- 全部的源代码都应由版本控制工具 (如source safe或cvs)进行管理，能展示完整的代码树。
- 开发人员应当记录开发过程日志，以便备份追踪。
- 系统应当能够保证开发过程的代码、设计和文档容易修改，代码应当结构清晰且有较详细的注释，设计文档详细明确。

2.5.6.可移植性

- 系统应当能够保证在 Windows、Linux 等多平台上容易安装和部署。
- 系统应当能够和其他软件共存于一个平台上，存在冲突的软件不超过。系统应当能够容易地被卸载，也容易被更高版本的系统替换。

第三章 解决方案

3.1.相关的体系结构模式

分层架构模式中的组件被组织成水平层，每个层执行应用中的一个具体角色(比如，表示逻辑或业务逻辑)。尽管分层架构模式不会具体要求模式中必须存在的层的数目和类型，但大多数分层架构由四个标准层：表示(presentation)，业务(business)，持久(persistence)和数据库(database)。

层次架构模式的每层有一个具体角色和职责。比如，一个表示层用于处理所有用户接口和浏览通信逻辑，而一个业务层会用于执行具体的和要求(request)相关的业务规则。架构的每个层会形成满足具体业务要求的一个抽象。举个例子，表示层不必知道或不必关心如何得到用户的数据，而只需要以特定的格式展示屏幕上的信息。同样地，业务层不必关心如何格式化用户数据来展示在屏幕上，甚至用户数据从哪里来；只需从持久层获得数据，用业务逻辑处理数据（比如，计算值或聚集数据），最后把信息传到表示层。

分层架构模式最重要的特性之一就是组件的关注分离(separation of concerns)。一个层中的组件只处理和该层相关的逻辑。比如，表示层中的组件只处理表示逻辑，而业务层中的组件只处理业务逻辑。这种组件分类使构建有效角色和职责模型变得容易，也有益于开发，测试，掌控和维护，这是因为这个架构有着定义明确的组件接口和限制的组件视野。

更多关于分层架构的内容，详见《UML和模式应用》中提到的GRASP开发模式

3.1.1.概念级体系结构

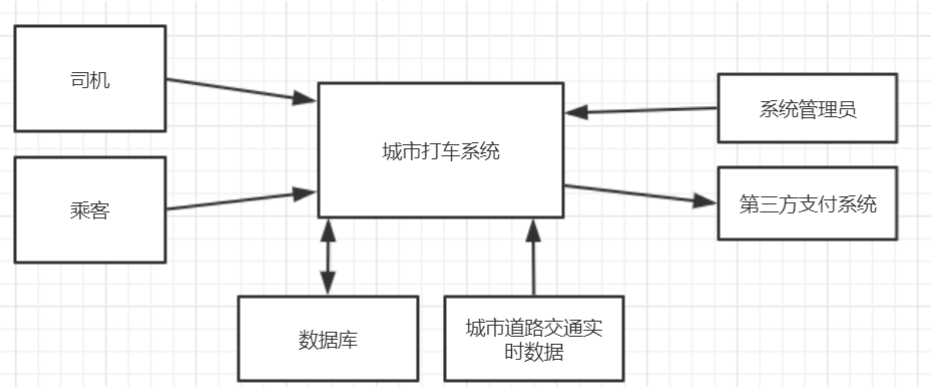
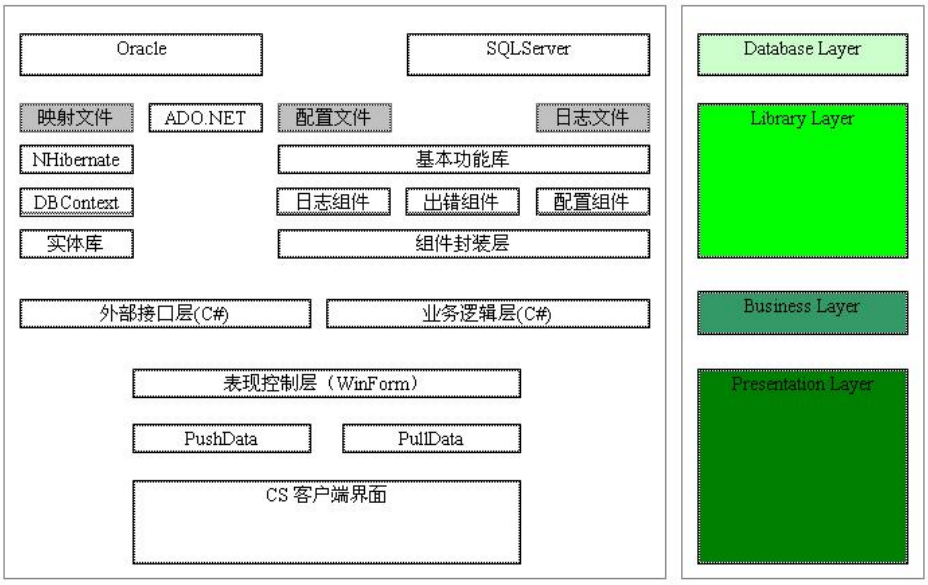


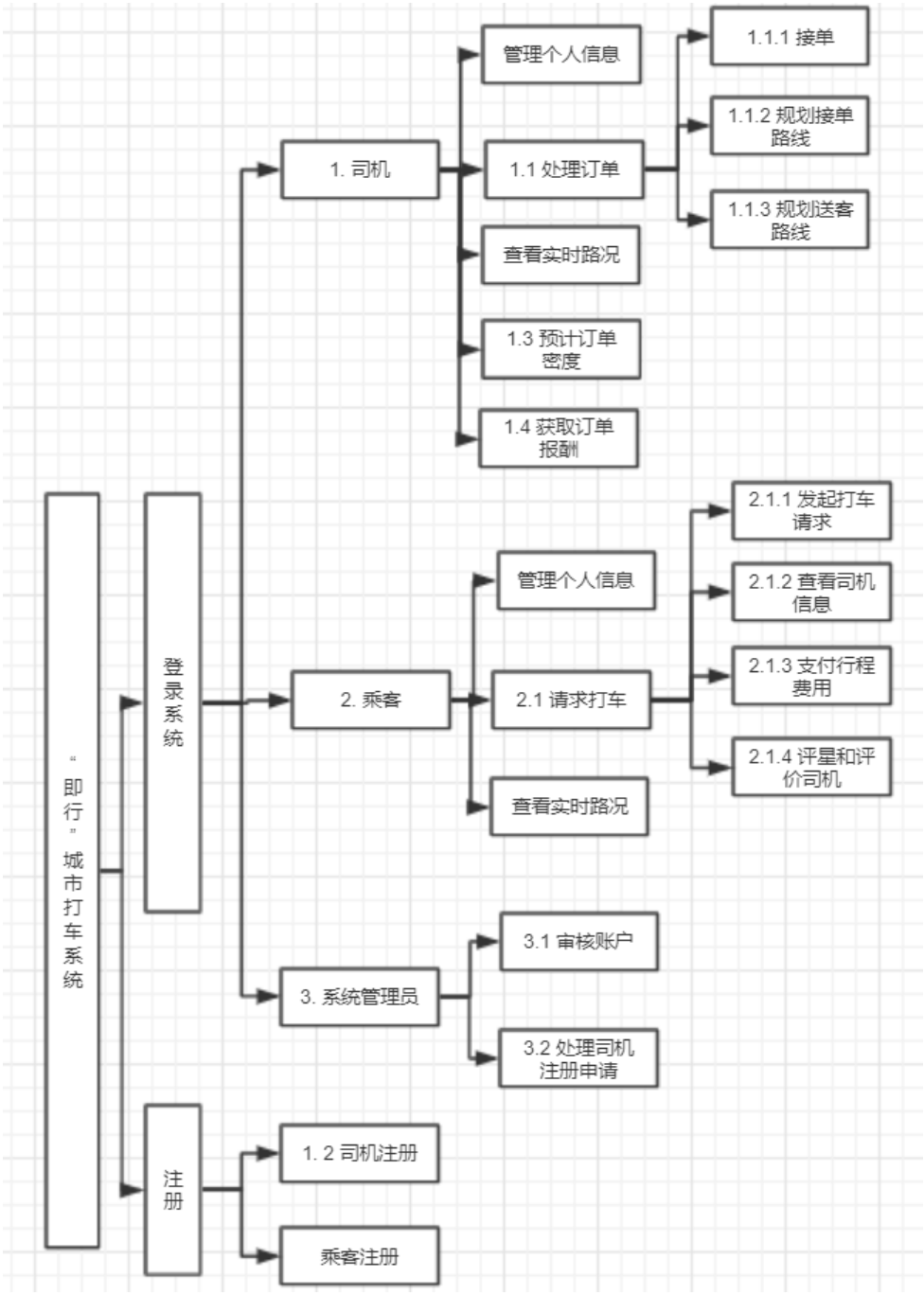
图3.1.1-1

3.1.2.分层级体系结构



3.1.3.模块级体系结构

系统的主要功能结构图见下图



3.2.体系结构概述

3.2.1.C/S三层体系结构简介

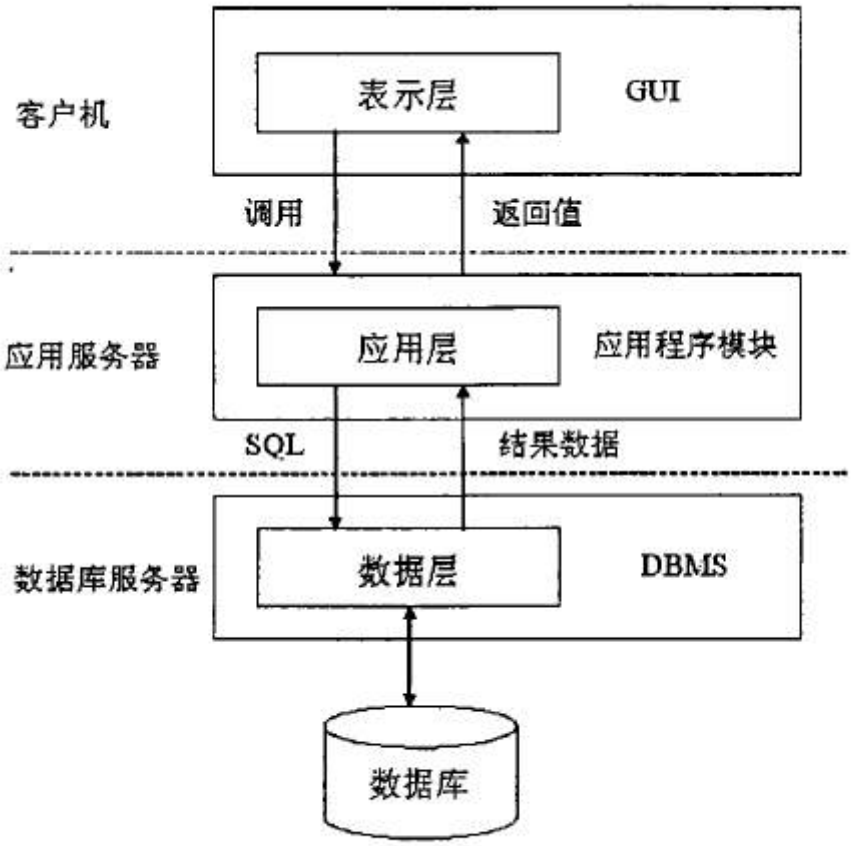


图 2-3 三层 C/S 结构

1. 表示层 表示层是应用的用户接口部分,它担负着用户与应用间的对话功能。它用于检查用户从键盘等输入的数据,显示应用输出的数据。为使用户能直观地进行操作,一般要使用图形用户接口(GUI),操作简单、易学易用。在变更用户接口时,只需改写显示控制和数据检查程序,而不影响其他两层。检查的内容也只限于数据的形式和值的范围,不包括有关业务本身的处理逻辑。 图形界面的结构是不固定的,这便于以后能灵活地进行变更。例如,在一个窗口中不是放入几个功能,而是按功能分割窗口,以便使每个窗口的功能简洁单纯。在这层的程序开发中主要是使用可视化编程工具。
2. 功能层 功能层相当于应用的本体,它是将具体的业务处理逻辑地编入程序中。例如,在制作订购合同的时要计算合同金额,按照定好的格式配置数据、打印订购合同,而处理所需的数据则要从表示层或数据层取得。表示层和功能层之间的数据交往要尽可能简洁。例如,用户检索数据时,要设法将有关检索要求的信息一次传送给功能层(参见图2),而由功能层处理过的检索结果数据也一次传送给表示层。在应用设计中,一定要避免"进行一次业务处理,在表示层和功能层间进行多几次数据交换"的笨拙设计。 通常,在功能层中包含有:确认用户对应用和数据库存取权限的功能以及记录系统处理日志的功能。这层的程序多半是用可视化编程工具开发的,也有使用COBOL和C语言的。
3. 数据层 数据层就是DBMS,负责管理对数据库数据的读写。DBMS必须能迅速执行大量数据的更新和检索。现在的主流是关系数据库管理系统(RDBMS)。因此,一般从功能层传送到数据层的要求大都使用SQL语言。

3.2.2.体系结构需求

1. 城市打车管理系统采用三层C/S结构，通过表现层的方式提供人机交互的界面，方便移动端用户使用
2. 采用VPN网络系统支撑平台运行，平台为应用系统提供包括：用户访问控制、信息加密、身份认证等安全方面的服务，全面保证系统安全。
3. 系统具有高可靠性，保证联网用户的在线率及成绩或评分规则准确无误的上传。
4. 系统具有高稳定性，可以高并发运行，保证服务器在处理大量信息时不死机。
5. 系统保证可扩展性。系统可继续开发为为企业提供运动管理、成员信息管理，要求整个系统能在不间断使用的情况下完成系统的升级。我们的产品在设计中，主服务器及网络设备采用模块化结构，硬件平台可以积木式拼装。平台产品提供良好的业务类型扩展性和业务规模扩展性，保证系统能快速方便地引入新的硬件和软件系统，可以随服务内容和业务量的增加动态部署计算机以提高系统处理能力。

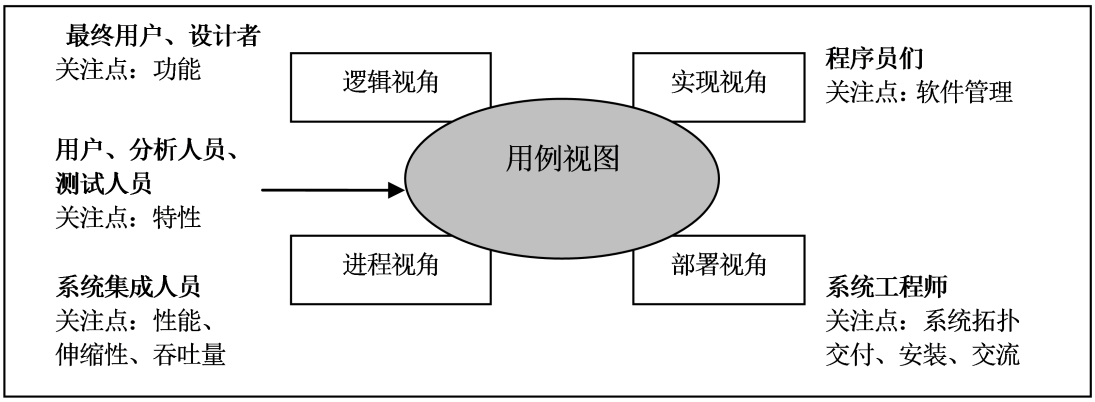
6. 系统支持开放性与标准化原则。采用开放技术标准，便于与基于不同开发技术实现的各种内外部系统互联互通，另一方面，在产品供应商和技术服务商的选择上也提供了更大的余地。
7. 系统实时运行过程中对数据进行备份，保证数据的安全性和有效性，同时实现系统运行时联网数据导入导出，不影响系统的实时运行。
8. 系统支持用户实时接入。联网用户的接入不影响系统的实时运行。

3.2.3.开发风格约定

1. 尽量简化项目,使项目易于管理。应尽快建起一个初始系统,并尽早投入运行。当项目规模较大时,可以将其分割成由更小开发组担负的子项目。
2. 要把精力花在设计上。首先要彻底弄清"需求" ,然后建立一个原型,以便测试设计中的"薄弱"环节。后来增加的特性或部件要保证与系统结构兼容。
3. 要奉行"拿来主义"。近来,可供选购的市售C/S产品很多,要坚持"能买就买,为我所用"的原则。必要时,买来后可对系统加以修改,其中既包括基础部件也包括应用。
4. 严格遵守业界标准。
5. 采用TP监控器或对象事务处理管理器 (Object Transaction Manager ,OTM)。
6. 要循序渐进。及时得到用户的反馈;保证项目各部分的良好衔接;及早解决接口问题,以保证项目进展协调
7. 在应用开发过程中,不可忽视系统管理。
8. 坚持"边分析,边设计;边编码,边测试"的原则。反复测试,包括用户信任测试、基准测试、系统测试、性能测试、系统集成测试、坚固性测试、服务交付测试等。
9. 制定合理的工程进度。
10. 制定完善的系统拓展计划,包括用户的培训和技术支持、高效的硬软件装载、已有数据和系统的平滑迁移。

3.3.结构化视图

- 本说明书使用 Philippe Kruchten 提出的从不同角度勾画系统的蓝图，即"4+1"视图模 型。该模型从 4 个角度(逻辑、实现、进程和部署)指出不同的相关利益方关心的事情，外 加从使用者的角度对用例进行观察，分析其影响系统的上下文和商业目标情况



3.3.1.逻辑视角（静态类图）*

**直至本次迭代，以下设计类图只关心乘客和司机部分，并且随着实现的深入会有变更发生。*

**以下设计类图采用属性文本表示法来表达类之间的引用关系，因而未直接标明连接*

**以下设计类图已以分包的形式给出，故不再额外画包图*

下图3.5-1为**类型包**的内容，用于格式化记录信息

图3.5-1

下图3.5-2为**角色包**的内容，用于标定参与使用系统的主要角色

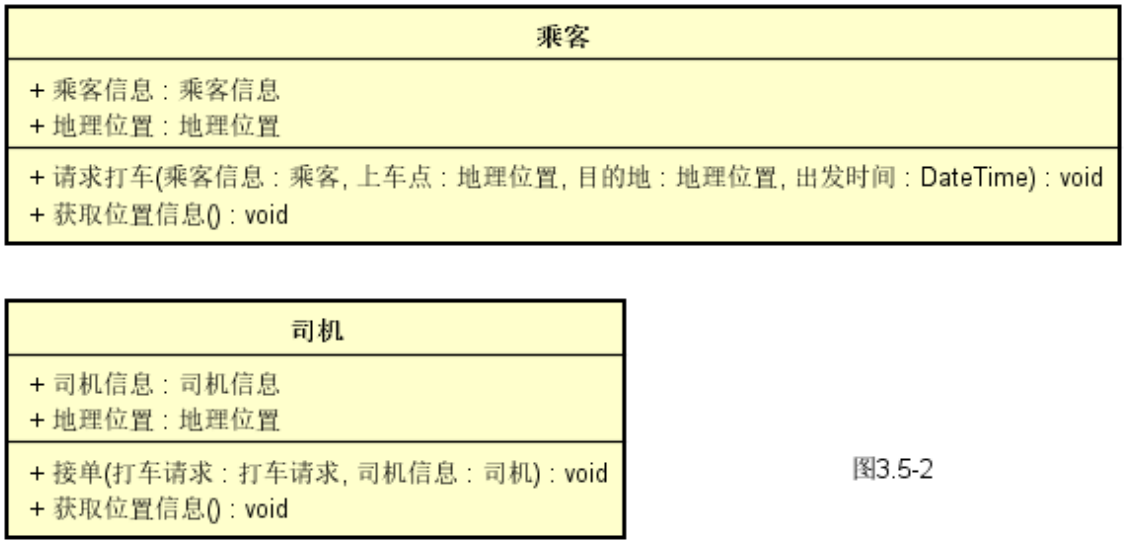


图3.5-2

下图3.5-3为系统运行时所产生的**数据包**的内容，用于记录系统中活动的数据元

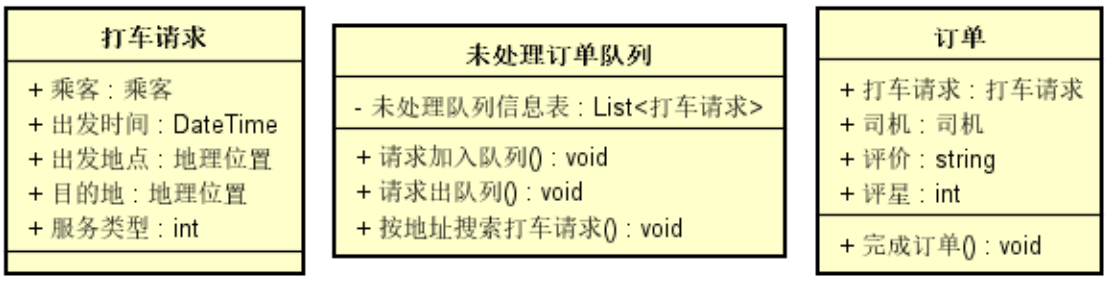


图3.5-3

下图3.5-4为系统所产生的持续性数据**记录包**的内容，类似于数据库

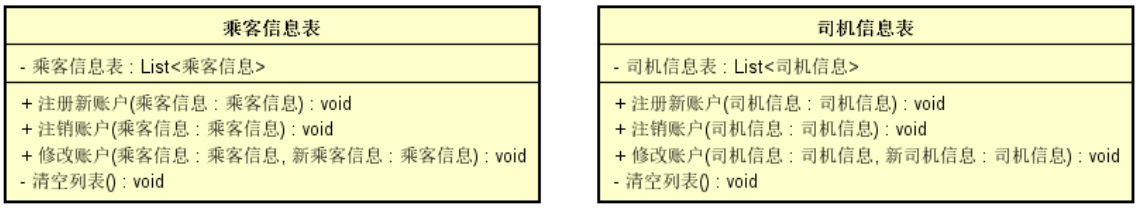
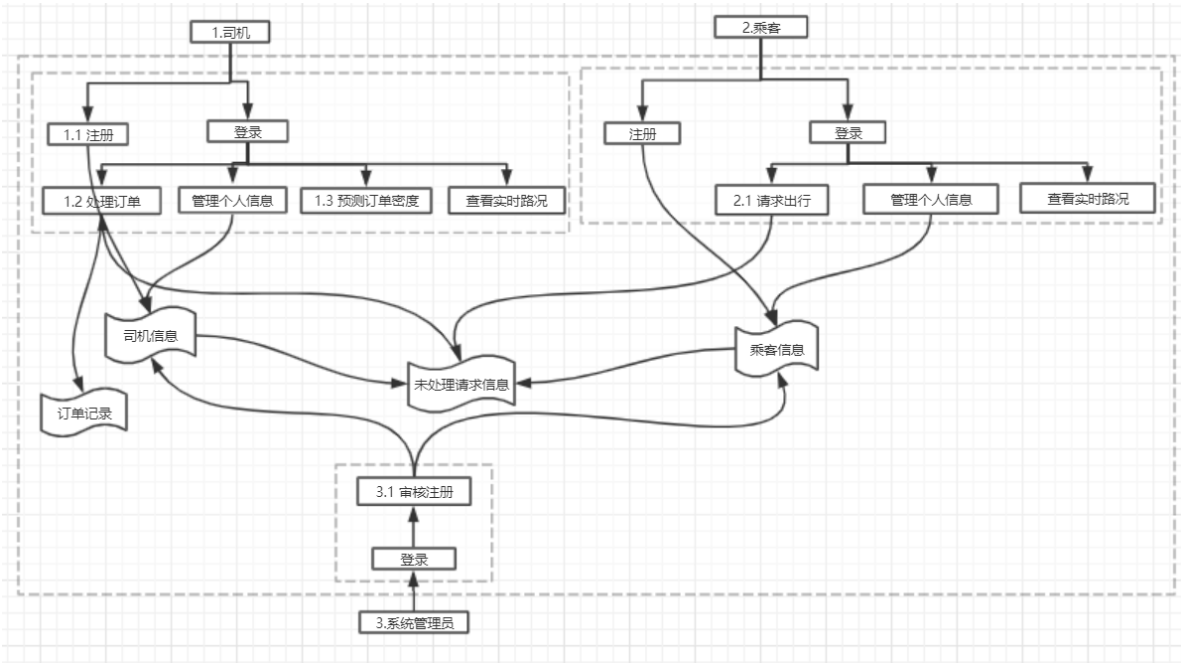


图3.5-4

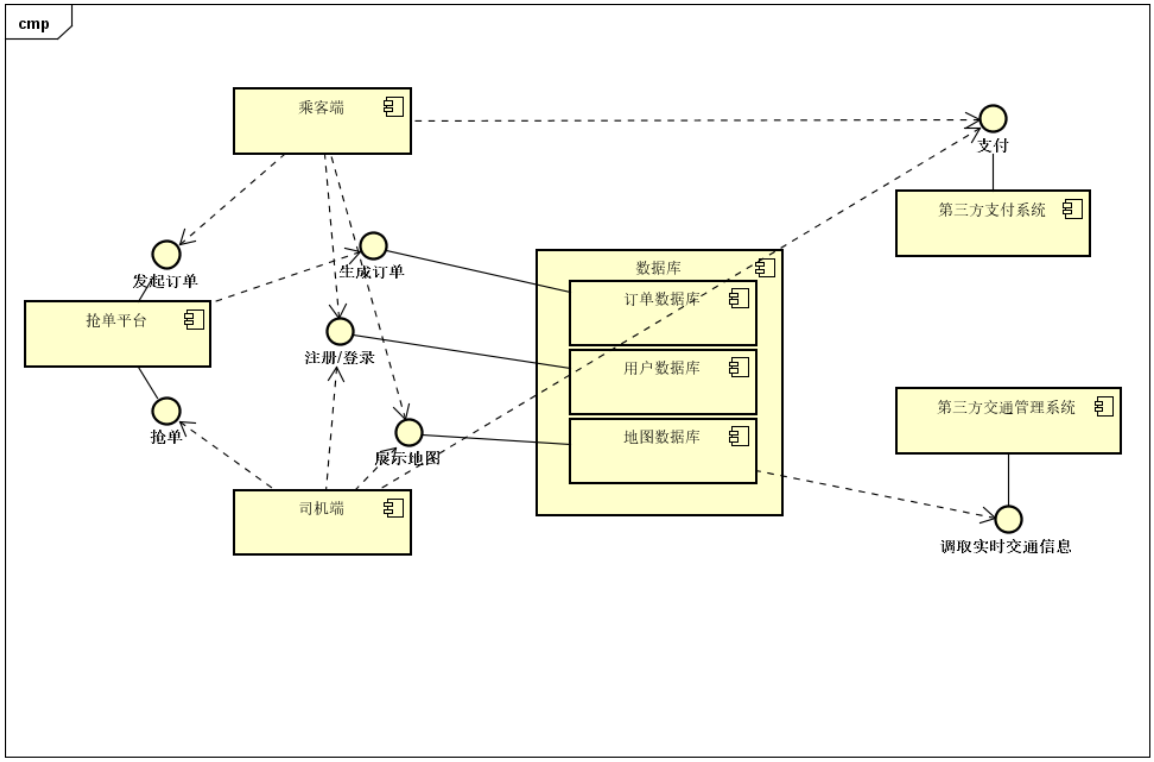
3.3.2.进程视角

- 进程视角主要描述在概念级的软件体系结构下，系统运行态的情况。描述系统在执行时，包括哪些进程（包括线程、进程、进程组），以及它们之间是如何进行通信的、如何进行消息传递、接口如何。
- 下图中内部的三个虚线方框是三个子系统的进程，中间展示的是他们之间的相互运作关系



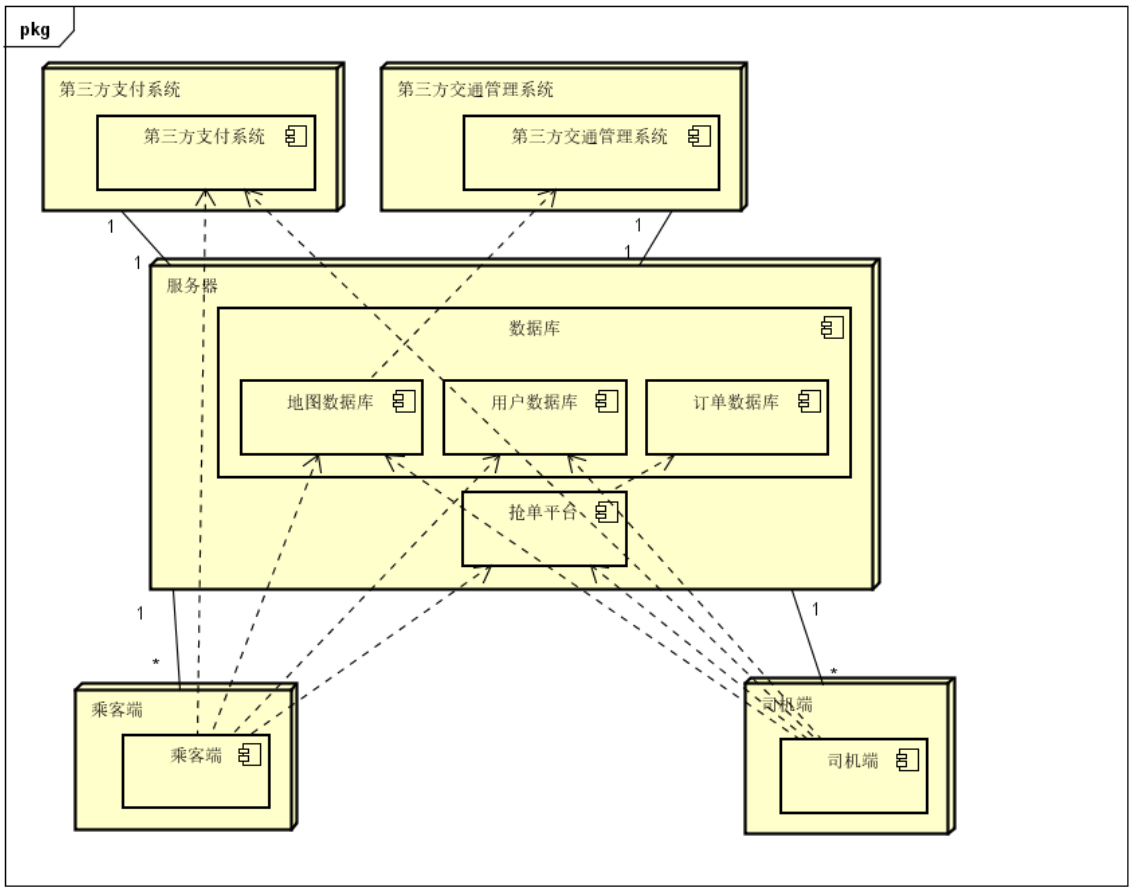
3.3.3.实现视角

- 实现视角，又叫开发视图（Development View），描述了在开发环境中软件的静态组 织结构，即关注软件开发环境下实际模块的组织，服务于软件编程人员。将软件打包成小 的程序块（程序库或子系统），它们可以由一位或几位开发人员来开发。子系统可以组织 成分层结构，每个层为上一层提供良好定义的接口。



3.3.4.部署视角

从系统软硬件物理配置的角度，描述系统的网络逻辑拓扑结构。模型包括各个物理节点的硬件与软件配置，网络的逻辑拓扑结构，节点间的交互和通讯关系等。同时还表达了进 程视图中的各个进程具体分配到物理节点的映射关系。



3.3.5.用例视角

- 司机的用例视角：

  （下图是司机角色的用例图，本文档只为司机需要被关注和记录的需求做用例分析，所以图中并未体现系统所有的功能）

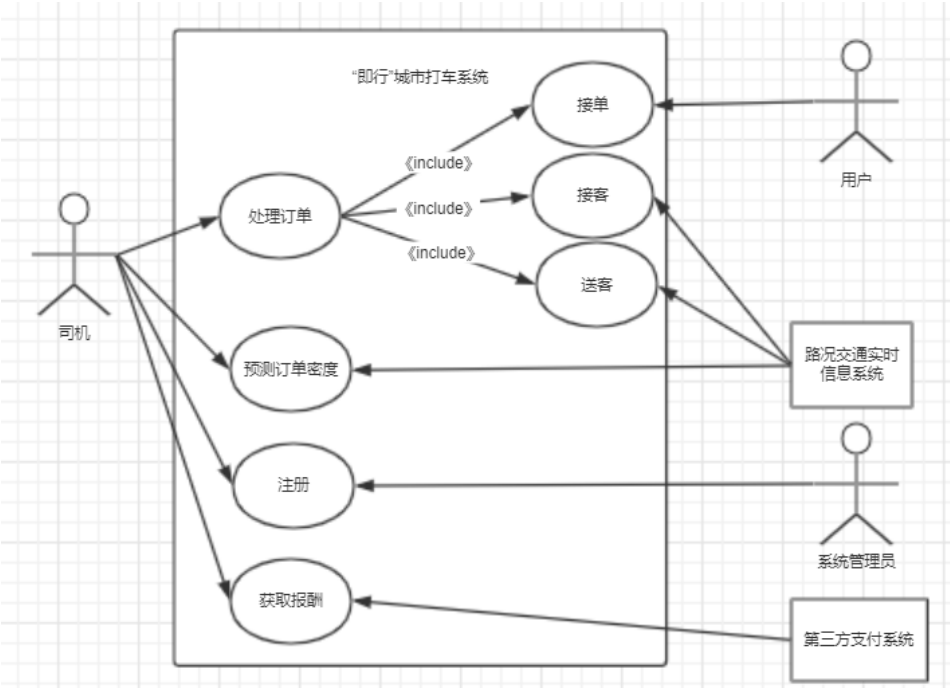


图3.3.1.1.1-1

- 乘客的用例视角：

  （下图是乘客角色的用例图，本文档只为乘客需要被关注和记录的需求做用例分析，所以图中并未体现系统所有的功能）

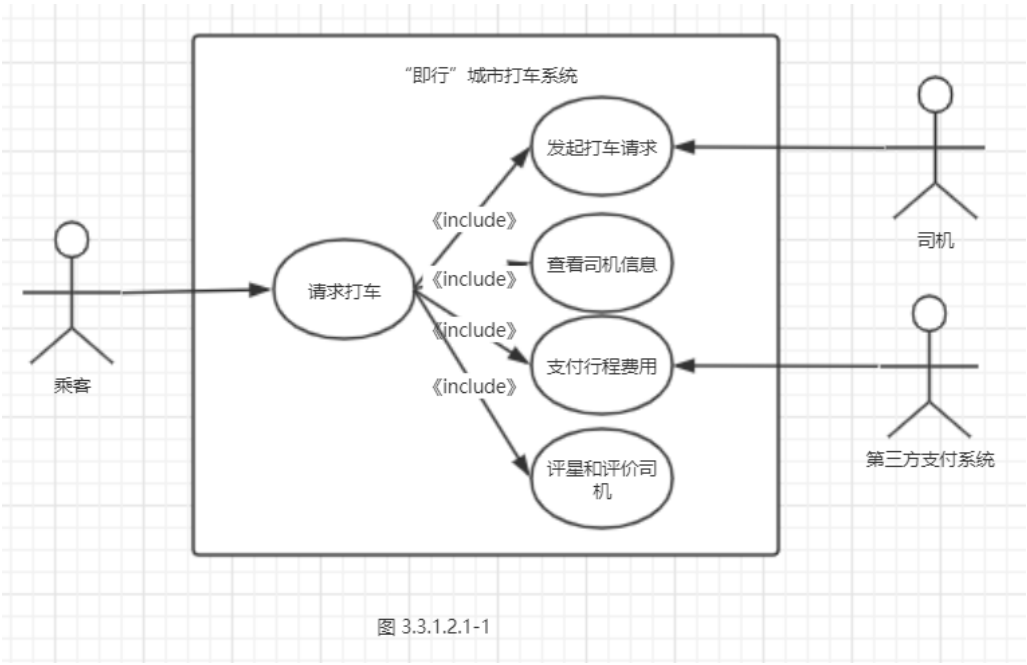


图 3.3.1.2.1-1

第四章 场景和风险分析

4.1.场景分析

4.1.1.用例场景

1. 当发生数据异常时，系统要通知到所有在线用户，并在屏幕上用红色的字体显示出来。该场景代表了用户期望的可靠性。
2. 学生期望可以看到自己的总体育成绩以及各部分复杂成绩。该场景代表了用户期望系统易于使用，即易用性。
3. 当用户登录系统时需要进行身份验证，而不是任何人都可以登录。该场景代表了用户所期望的安全性。
4. 系统管理员期望授予一类用户以权限，例如授予所有的学生以查询成绩的权限等。该场景代表了用户期望的易用性。
5. 当用户将图形界面放大时，屏幕要在1秒内重新显示出来。该场景代表了系统性能要求。

4.1.2.增长性场景

预期未来系统修改时可能发生的场景。

1. 通过扩充现有数据库表的规模，把检索时间降低到平均1s以内。
2. 当系统不再局限于app，而能用于网页web或者微程序时，该系统能照样运行良好，检索速度快

4.1.3.探索性场景

推动系统封装和降低工作压力的场景。

1. 系统能够从Windows平台更换到Linux平台。
2. 改进系统的可使用性，使其从98%提升到99.9999%。
3. 正常情况下，当一半服务器宕机时，不影响整个系统的可使用性。

4.3.风险

在项目进行过程中可能发生的事件，这些事件将会对项目按预期时间，资源和预算完成产生重大影响。

1. 规模风险

   需求是否相当稳定并得到了充分的了解。

   项目规模是固定不变还是在不断扩展。

   项目开发的时间范围是否太短、不够灵活。

2. 技术风险

   是否有足够的异常处理。

   是否考虑到模块的生命周期。

   需求中的事务量是否合理。

   数据量是否合理？当前可用的框架是否能够保存这些数据？

   对于与其他系统（包括企业以外的系统）的接口是否存在外部依赖性？是否存在必需的接口或必须创建它们。

   是否存在极不灵活的可用性和安全性需求（例如“系统必须永远不出现故障”）。

   系统的用户是否对正在开发的系统类型没有经验。

应用程序的大小或复杂性，或者技术的新颖性是否导致了风险的增加。

是否存在对国家语言支持的需求。

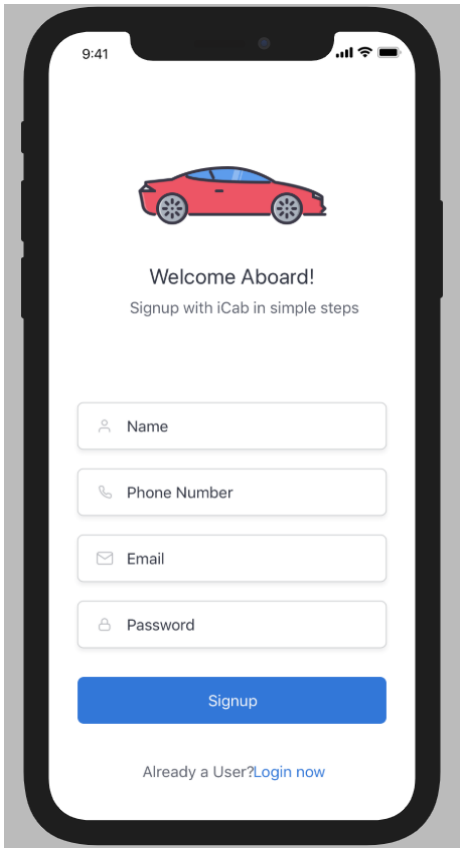
第五章 用户界面设计

1. **前后端满足GRASP原则**：采用BCE模型，将前后端分离开来，使得后端的改动不会影响前端界面，使用户有统一感

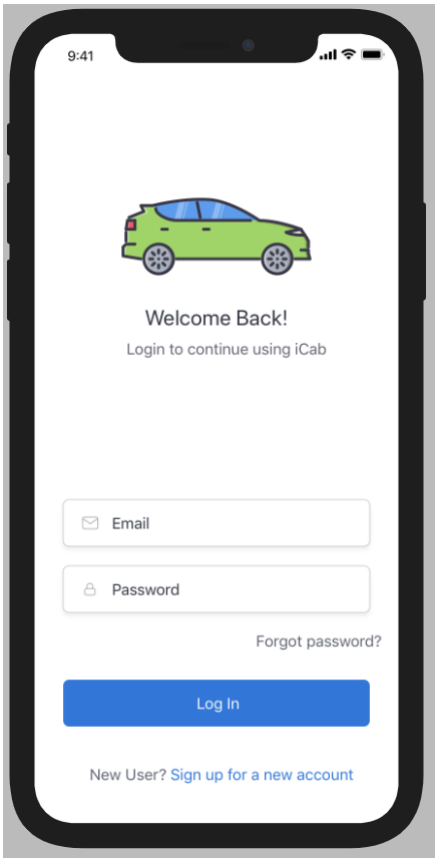
2. **满足功能性需求的要求**：用例中要求界面的主页能展示实时路况地图，有个人主页界面等

下面举出一个良好用户界面的例子*（直至本次迭代，暂且只完成了乘客端的界面原型实现）*：

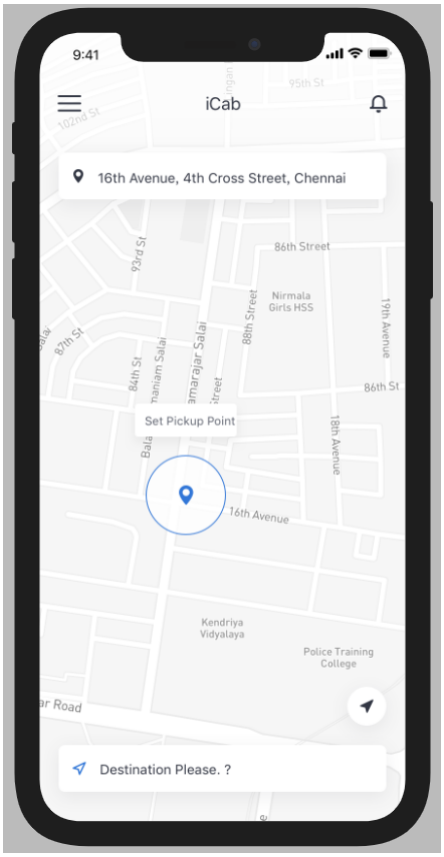
- 注册界面：



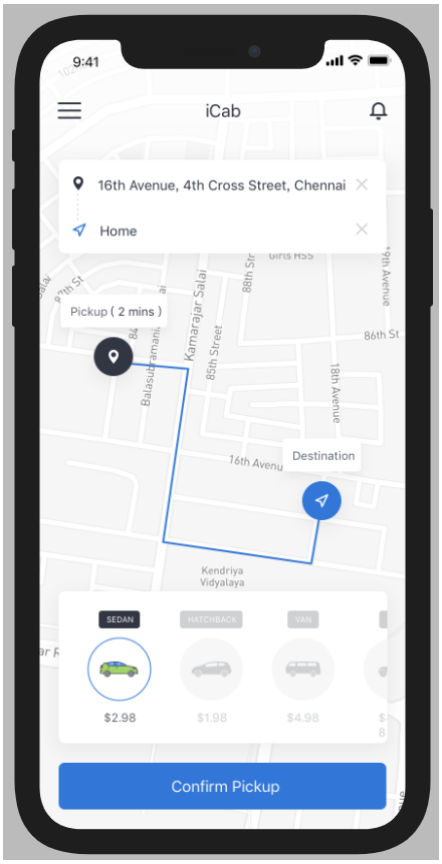
- 登录界面



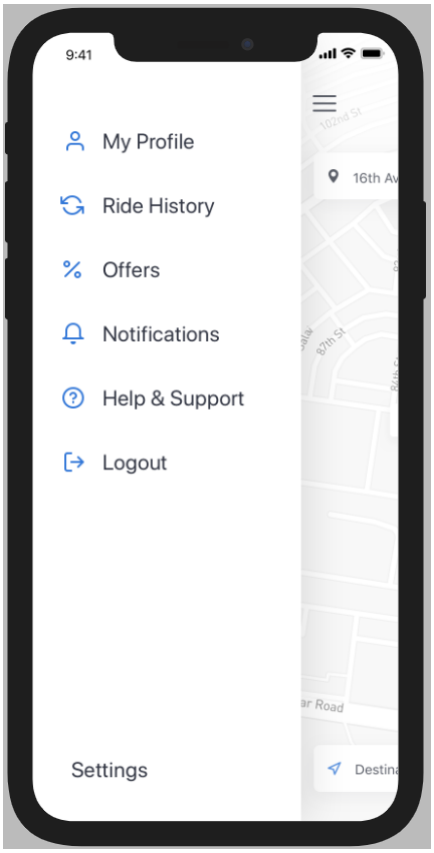
- 查看地图界面



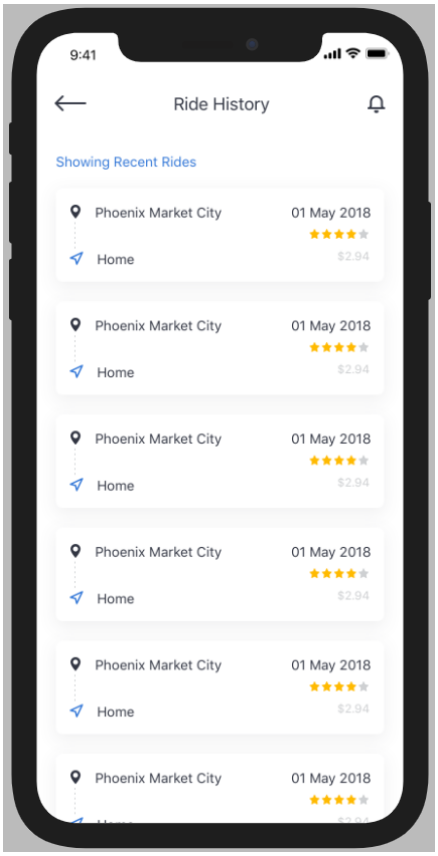
- 发起打车请求界面



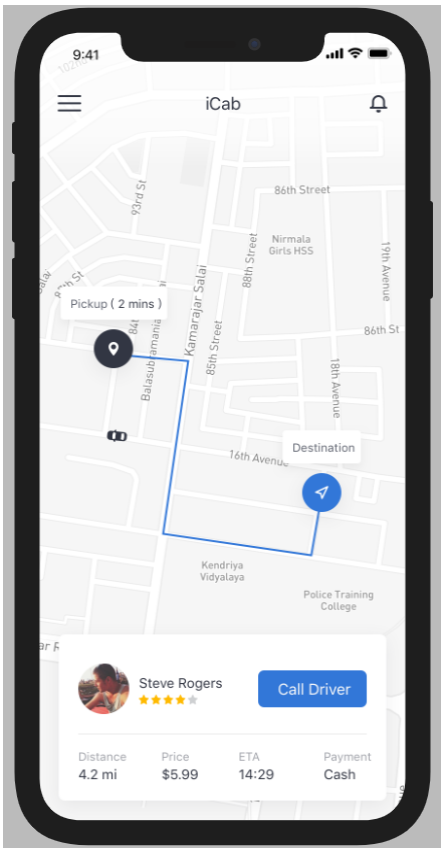
- 良好的界面转换——侧边栏



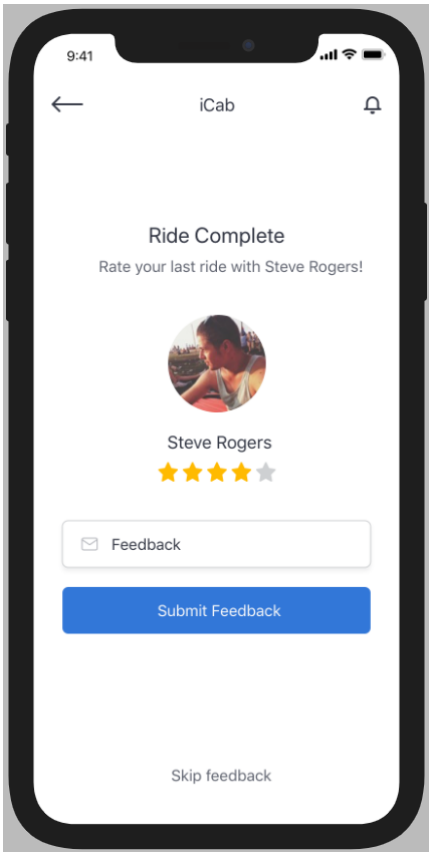
- 行程历史界面



- 打车请求被响应界面，能查看到司机基本信息



- 完成订单后的反馈界面


9:41
iCab
Ride Complete
Rate your last ride with Steve Rogers!
Steve Rogers
Feedback
Submit Feedback
Skip feedback