霍格沃兹地图

软件架构文档

版本 <1.4>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 29/10/2020 | 1.0 | 编写软件架构文档 | 赵昱童 |
| 30/10/2020 | 1.1 | 修改软件架构文档 | 蓝煜斌 |
| 01/11/2020 | 1.2 | 整体完善 | 全体成员 |
| 07/11/2020 | 1.3 | 添加进程、页面、数据视图 | 赵昱童、蓝煜斌 |
| 08/11/2020 | 1.4 | 添加关键的质量属性的设计战术 | 熊添晨、郭黛筠 |

目录

1. 简介 5

1.1 目的 5

1.2 参考资料 5

2. 用例视图 5

2.1 管理个人信息 5

2.2 设置个性化主题 5

2.3 添加、管理好友 5

2.4 组队 5

2.5 获取景点信息 5

2.6 打卡留言评价 6

2.7 生成游记并分享 6

2.8 获取并绘制当前位置信息 6

2.9 选择目的地导航 6

2.10 维护景区信息 6

2.11 用户管理 6

2.12 用户行为统计 6

3. 逻辑视图 6

3.1 概述 6

3.2 在构架方面具有重要意义的设计包 6

**3.2.1** **Client包** 6

**3.2.2** **Server包** 8

4. 进程视图 10

4.1 Processes 10

**4.1.1** **UserApplication** 10

**4.1.2** **UserForm** 10

**4.1.3** **ScenicOperatorApplication** 10

**4.1.4** **ScenicOperatorForm** 10

**4.1.5** **AdministratorWeb** 10

**4.1.6** **AdministratorForm** 10

**4.1.7** **HogwartsServices** 10

**4.1.8** **HogwartsController** 11

4.2 Processes to Design Elements 11

**4.2.1** **GenerateTravelNote&Share** 11

**4.2.2** **GroupTravel** 11

**4.2.3** **GetScenicInformation** 11

**4.2.4** **Clock/Comment/Note** 11

**4.2.5** **UserCache** 11

**4.2.6** **ScenicCache** 11

**4.2.7** **TravelNote** 11

**4.2.8** **Comment** 12

**4.2.9** **Note** 12

**4.2.10** **User** 12

**4.2.11** **Scenic** 12

5. 部署视图 12

5.1 Desktop: Administrator(电脑web端) 12

5.2 Mobile Phone: User 12

5.3 Hogwarts Server 12

5.4 AliyunOSS 12

6. 实现视图 13

7. 页面视图 14

8. 技术视图 15

9. 数据视图 16

10. 关键的质量属性的设计战术 16

10.1 性能战术 16

10.2 易用性战术 16

10.3 可修改性战术 16

软件架构文档 （简化版）

# 简介

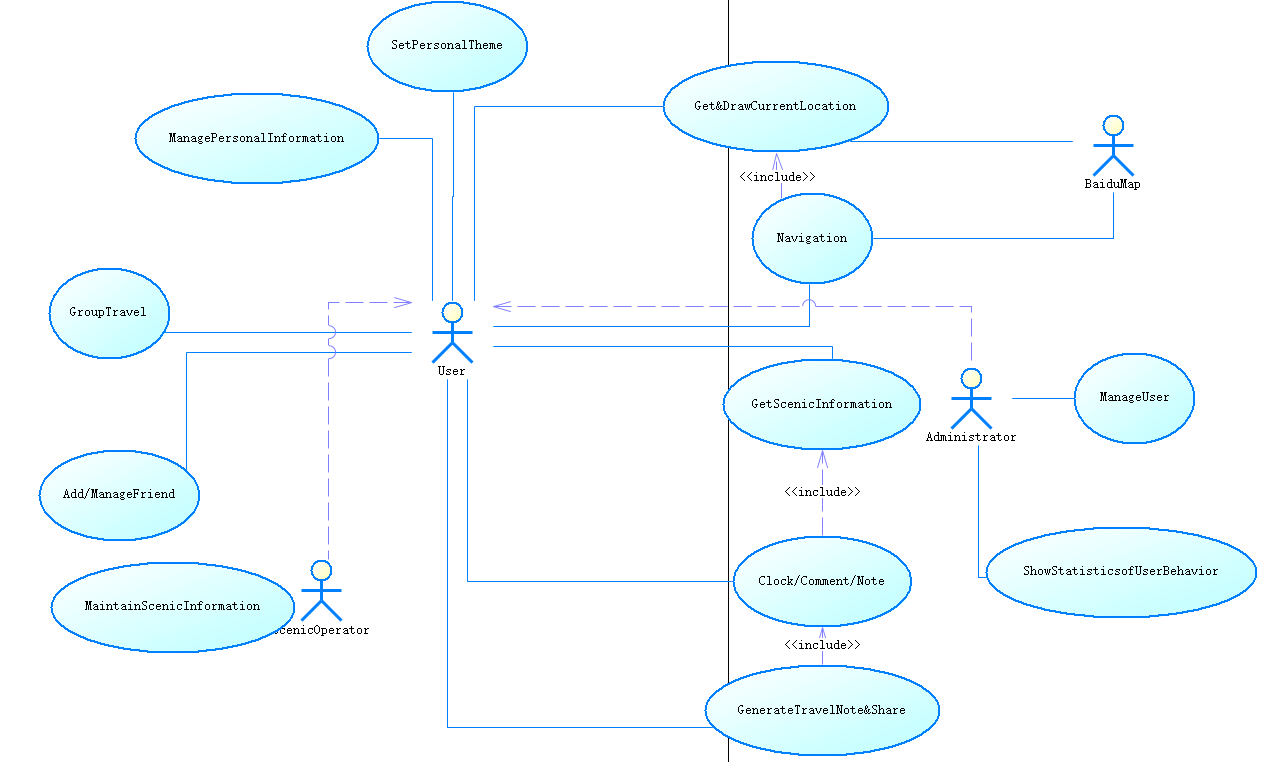
## 目的

本文档将从构架方面对系统进行综合概述，其中会使用多种不同的构架视图来描述系统的各个方面。它用于记录并表述已对系统的构架方面作出的重要决策。

## 参考资料

1. 沈备军，陈昊鹏.软件工程原理[M].北京:高等教育出版社,2013.2
2. 第一小组.软件需求规约.2020.9
3. Flutter开发文档：https://flutter.cn/docs

# 用例视图



## 管理个人信息

普通用户在个人中心查看或者修改个人信息。

## 设置个性化主题

普通用户在设置页面查看或修改个性化主题。

## 添加、管理好友

普通用户在好友列表添加、管理好友。

## 组队

普通用户在即将出行游玩时进行组队。

## 获取景点信息

普通用户选定一个景点获取景点详细信息。

## 打卡留言评价

普通用户在景点页面进行打卡或对景点评价或留言。

## 生成游记并分享

普通用户点击生成游记自动生成一篇游记可供分享。

## 获取并绘制当前位置信息

普通用户可通过百度地图API获取位置信息并在主页绘制。

## 选择目的地导航

普通用户选择一个目的地使用系统进行导航。

## 维护景区信息

景区运营方维护景区信息，新增景区信息或更新存储在数据库的景区信息。

## 用户管理

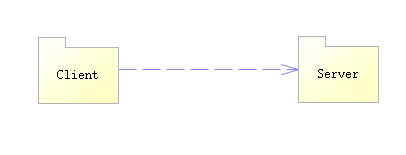
管理员查看数据库中的用户信息并对用户信息进行更新。

## 用户行为统计

管理员统计各景点各季的旅行热度、游客的旅行方式等。

# 逻辑视图

## 概述



霍格沃兹地图的逻辑视图采用了C/S的架构，由2个主要包组成：Client和Server。

## 在构架方面具有重要意义的设计包

### **Client包**

Client包是主要用于服务用户，负责完成与用户的交互任务，与Server端相连，接受用户请求，并通过网络向服务器提出请求，对数据库进行操作。

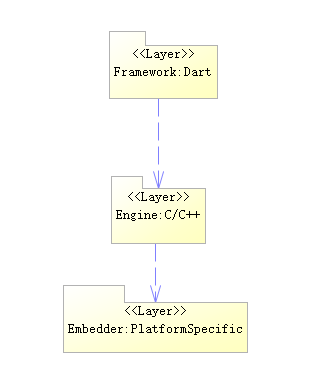
Client包由Flutter包组成。

Flutter包采用了层次结构，分为三层，由3个主要包组成：Framework: Dart、Engine: C/C++和Embedder: PlatformSpecific。

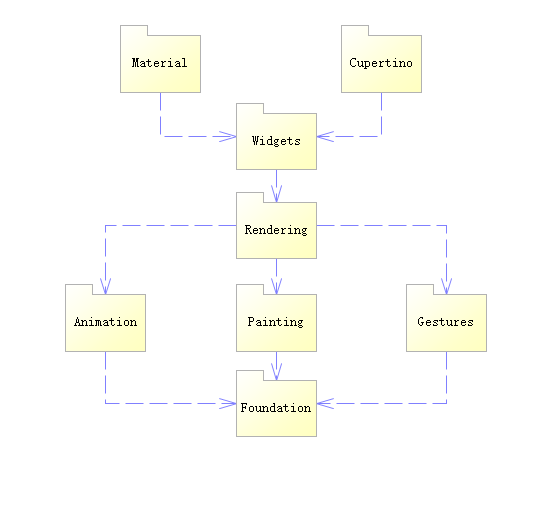
Framework层：Framework使用dart实现，包括Material Design风格的Widget, Cupertino(针对iOS)风格的Widgets，文本/图片/按钮等基础Widgets，渲染，动画，手势等。此部分的核心代码是:flutter仓库下的flutter package，以及sky\_engine仓库下的io, async, ui(dart: ui库提供了Flutter框架和引擎之间的接口)等package。

Engine层：Engine使用C++实现，主要包括:Skia, Dart和Text。Skia是开源的二维图形库，提供了适用于多种软硬件平台的通用API。

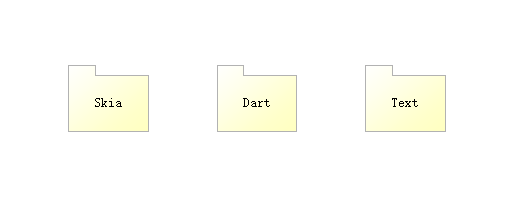
Embedder层：Embedder是一个嵌入层，即把Flutter嵌入到各个平台上去，这里做的主要工作包括渲染Surface设置,线程设置，以及插件等。从这里可以看出，Flutter的平台相关层很低，平台(如iOS)只是提供一个画布，剩余的所有渲染相关的逻辑都在Flutter内部，这就使得它具有了很好的跨端一致性。



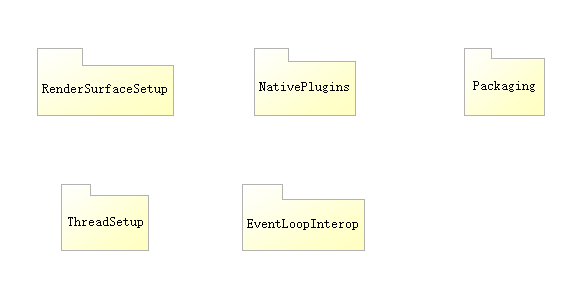
**Framework: Dart：**



**Engine: C/C++：**



**Embedder: PlatformSpecific：**



从上述架构图可以看出，渲染引擎依靠跨平台的Skia图形库来实现，依赖系统的只有图形绘制相关的接口，可以在最大程度上保证不同平台、不同设备的体验一致性，逻辑处理使用支持AOT的Dart语言，执行效率也比JavaScript高得多。

### **Server包**

Server包负责数据的管理，接受Client端的请求，将数据提供给Client，Client将数据呈现给用户。

Server包采用了层次结构，分为六层，由6个主要包组成：Controller、JWT Token、Service、Dao、DB和AliyunOSS。

具体介绍如下：

Controller层：主要负责具体业务模块流程的控制，会调用Service层的接口来控制业务逻辑。Controller负责请求转发，接受页面过来的参数，传给Service处理，接到返回值，再传给前端。

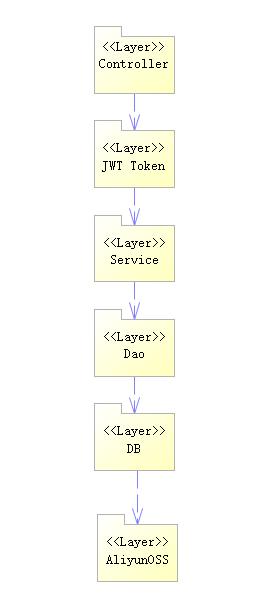
JWT Token层：用于用户的认证与授权，在前端与后端之间传递被认证的用户身份信息（Bearer Token），以便于从资源服务器获取资源。

Service层：主要负责业务模块的应用逻辑与应用设计。Service层业务实现，具体调用到已经定义的DAO的接口，封装Service层的业务逻辑有利于通用的业务逻辑的独立性和重复利用性 。

Dao (Data Access Object):数据存储对象层：主要做数据持久层的工作，负责与数据库进行联络的一些任务都封装在此

DB层：数据库层，大部分数据存储在MongoDB数据库中。

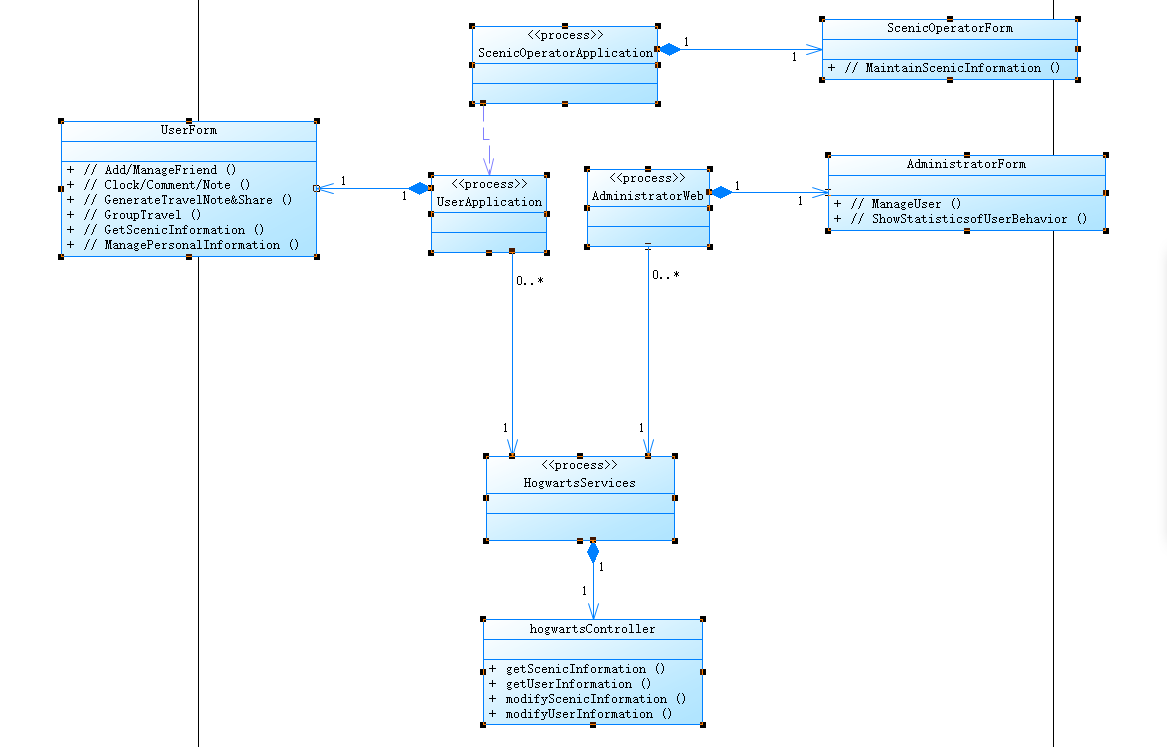
AliyunOSS层：使用阿里云对象存储服务，部分数据（如图像等非文本数据或音频数据等）存储在此层。



上述分层架构实现了“高内聚，低耦合”的特性，使得同一层只需要负责与上下层进行交互，降低了设计的复杂度。同时使用Spring框架，这是一个轻量级的控制反转(IoC)和面向切面(AOP)的容器框架，也是面向接口的编程，由容器控制程序之间的依赖关系，而非传统实现中，由程序代码直接操控。（依赖）控制权由应用代码中转到了外部容器，控制权的转移，是所谓反转。依赖注入，即组件之间的依赖关系由容器在运行期决定，即由容器动态地将某种依赖关系注入到组件之中，也能起到解耦的作用。

# 进程视图

## Processes



### **UserApplication**

管理用户功能，包括用户页面处理与业务流程的协调。对于每一个普通用户都有一个此进程的实例。

### **UserForm**

控制UserApplication的界面，控制用户使用霍格沃兹地图的接口。

### **ScenicOperatorApplication**

管理景区运营方功能，（与系统管理员区分开来，景区运营方相当于特殊用户）包括景区运营方页面处理与业务流程的协调。对于每一个景区运营方都有一个此进程的实例。

### **ScenicOperatorForm**

控制ScenicOperatorApplication的界面，控制景区运营方使用霍格沃兹地图的接口。

### **AdministratorWeb**

管理管理员功能，包括用户页面处理与业务流程的协调。对于每一个管理员都有一个此进程的实例。

### **AdministratorForm**

控制AdministratorWeb的界面，控制管理员使用霍格沃兹地图的接口。

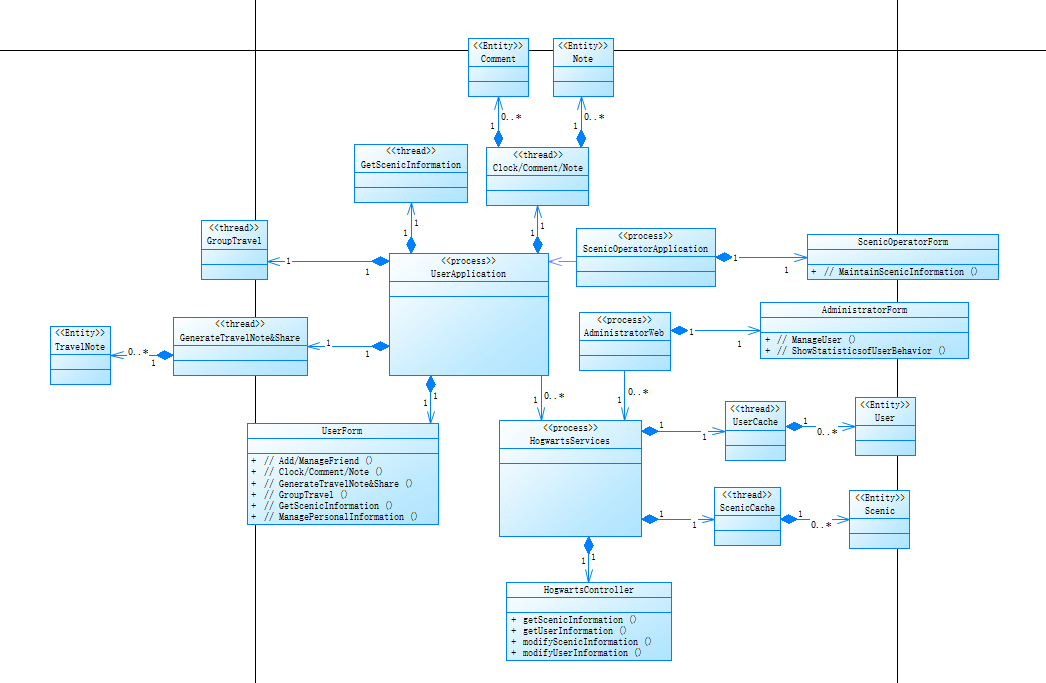
### **HogwartsServices**

接受UserApplication和AdministrationWeb的请求，返回对应的数据。

### **HogwartsController**

控制HogwartsServices对数据库数据的访问和操作。

## Processes to Design Elements



### **GenerateTravelNote&Share**

普通用户点击生成游记自动生成一篇游记可供分享。

### **GroupTravel**

普通用户在即将出行游玩时进行组队。

### **GetScenicInformation**

普通用户选定一个景点获取景点详细信息。

### **Clock/Comment/Note**

普通用户在景点页面进行打卡或对景点评价或留言。

### **UserCache**

用于从数据库中获取用户数据。

### **ScenicCache**

用于从数据库中获取景区数据。

### **TravelNote**

用户生成的游记。

### **Comment**

用户对景点的评分。

### **Note**

用户在景区或景点的留言。

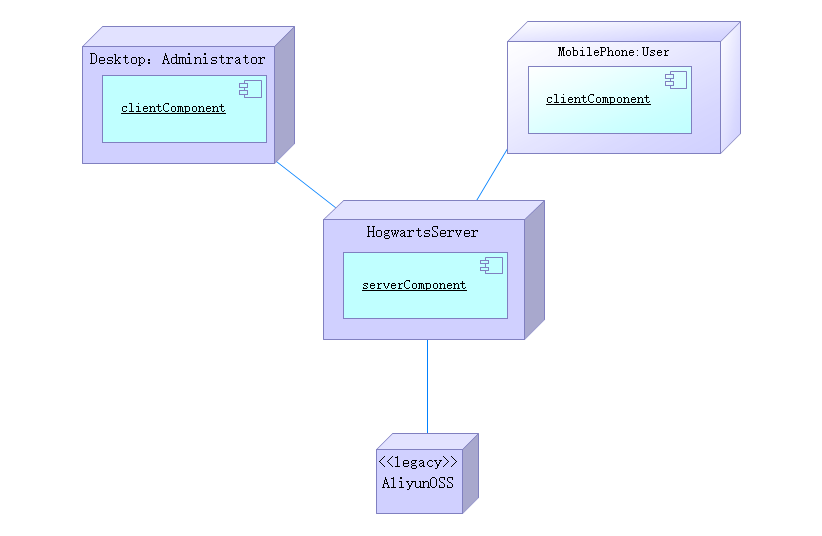
### **User**

数据库中存储的用户数据。

### **Scenic**

数据库中存储的景区数据。

# 部署视图



## Desktop: Administrator(电脑web端)

管理员用来管理用户和景点信息、统计用户行为的客户端。（使用电脑网页端）

## Mobile Phone: User

普通用户通过手机客户端进行定位、导游、组队出行、打卡评价、生成游记分享。景区运营方通过 手机客户端进行景区信息管理。

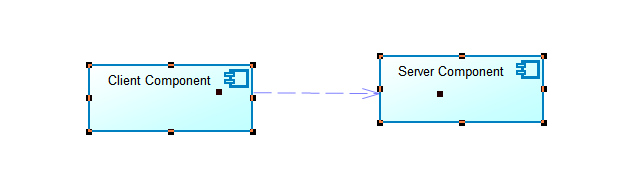
## Hogwarts Server

霍格沃兹服务器是霍格沃兹地图的服务器，普通用户、管理员、景区运营方均连接到该服务器获取数据。

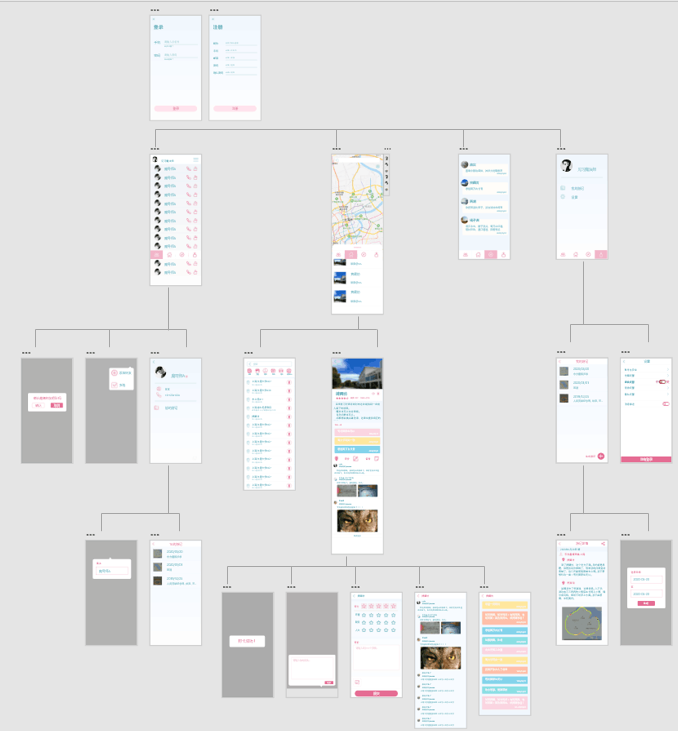
## AliyunOSS

霍格沃兹服务器将部分数据存储到阿里云OSS对象存储服务中。

# 实现视图

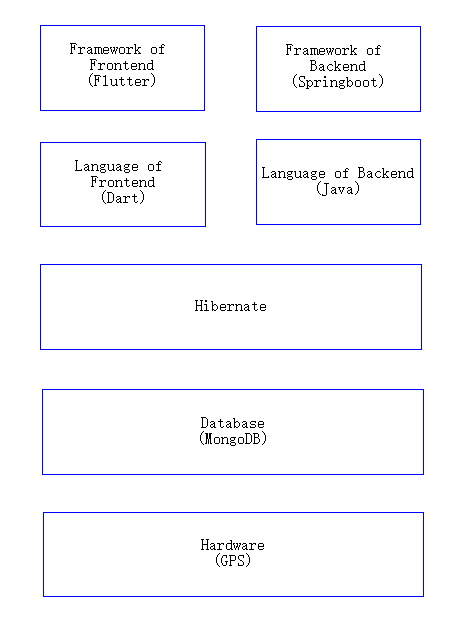


# 页面视图



页面视图如上所示，展现了各个页面之间的跳转以及对应关系，为前端页面的开发提供参考。

# 技术视图



底层需要硬件提供GPS服务。

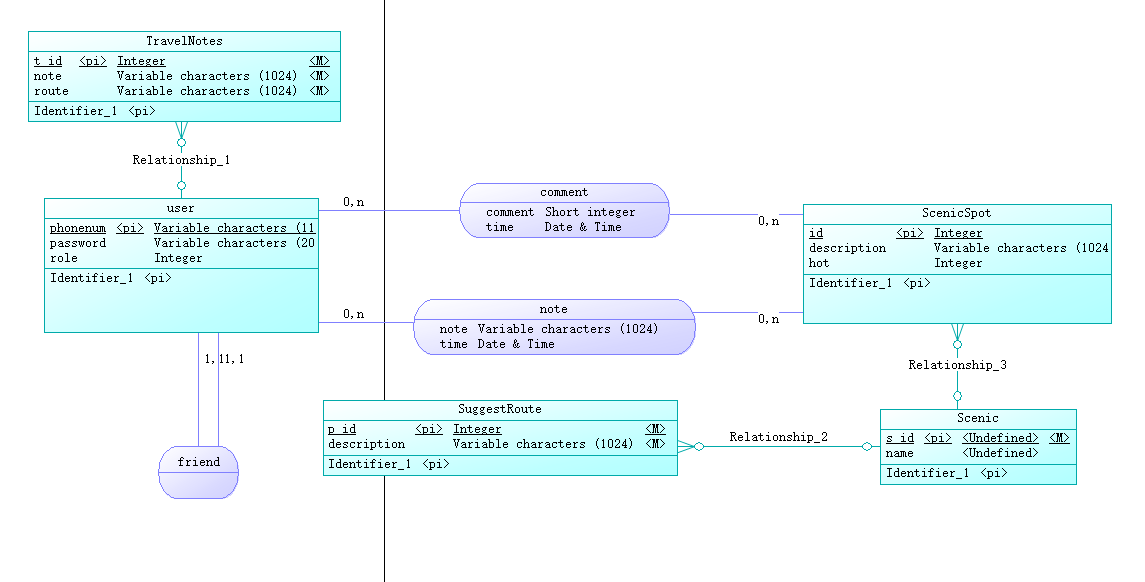
数据库需要使用MongoDB数据库。

使用Hibernate进行对象关系映射，Hibernate对JDBC进行了非常轻量级的对象封装，它将POJO与数据库表建立映射关系，是一个全自动的ORM框架，hibernate可以自动生成SQL语句，自动执行，使得后端使用Java可以使用对象编程思维来操纵数据库。

前端使用Dart语言进行开发，使用Flutter跨平台框架。使用Flutter框架可以使用一套代码实现对于IOS、Android、Web三个平台的支持。

后端使用Java语言进行开发，使用Spring Boot后端框架。

# 数据视图



数据视图如上，表明了存储在MongoDB中的数据表的对应关系。为数据库的设计与实现提供了参考。

# 关键的质量属性的设计战术

## 性能战术

根据本项目的非功能性需求中的1000个并发用户，最长响应时间为3秒（不包括网络延迟），所以我们首先聚焦于性能战术。

我们采用缓存来保存组队信息以及个人位置信息这些修改频繁的信息，在一段长时间未更新后再写入数据库并从缓存中移除，防止由于数据库的写入而延长了请求的响应时间。

其次我们采用二分查找法来完成通过队伍号码查找队伍信息，以减少大量队伍存在时队伍信息查找较为缓慢的问题。

最后如果性能依旧无法达到要求，我们将采用分表的方式，限制每张表的大小，以此减少从大表中查找和读出数据的时间。同时可以将热门属性与冷门属性分离，从而快速取出某些属性，达到性能的提升。

## 易用性战术

我们将用户接口分离，限制可能出现的用户接口的频繁变化所造成的影响。

在用户帮助上，我们采用第一次使用时进行新手引导，以及后续在软件中查看带操作步骤图解的帮助文档来帮助用户使用我们的软件。  
 在界面设计上，我们秉承以人为本的原则为用户提供尽量简洁的页面和便捷的操作。采用柔和的对比色调防止视觉疲劳，支持在发现页面上进行景点详情页的跳转。搜索时，在焦点自动定位到输入框的同时，提示历史搜索记录方便用户使用。自动生成指定时间段的游记也为用户提供了便捷的服务。

## 可修改性战术

首先我们对于后端进行分层（在前面3.2.2server有具体介绍），并将接口与实现分离，从而使得不同层解耦，以减小系统中模块修改引起的成本变化。

同时我们将一些基本不变的基础信息采用公有化的模块来处理，对于重要的关键信息采用私有的模块来处理，模块之间采用抽象化的共有接口来进行联系，这样在修改某个模块时就能防止代码污染扩散。