<公司名称>

智慧公路养护管理系统
软件构架文档

版本 <1.1>

修订历史记录

日期	版本	说明	作者
08/11/2022	1.0		林哲显
16/11/2022	1.1	根据架构评审意见修改	裴奕博

目录

软件构架文档

1. 简介

本文档将从构架方面对智慧公路养护管理系统进行综合概述，其中使用多种不同的构架视图来描述系统的各个方面，包括用例视图技术视图、逻辑视图、部署视图。它用于记录并表述已对系统的构架方面作出的重要决策。

1.1 目的

本文档将从构架方面对智慧公路养护管理系统进行综合概述，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。

1.2 范围

本文档适用于整个智慧公路养护管理系统，包括提供给平台维护人员和内页人员的网页端，外业人员的移动端和两者共用的后端整个项目。

1.3 定义、首字母缩写词和缩略语

MVC：Model-View-Controller 架构，常用于用户界面设计的架构。

C/S 和 B/S：Client-Server 和 Browser-Server 架构。

RTK(Q)：Redux Toolkit (Query)，一个全局状态管理库和数据请求的封装。

ORM： Object Relation Mapping，把数据库中的关系数据映射成为程序中的对象。

RN：React Native，FaceBook 公司推出的跨平台的移动端框架。

1.4 参考资料

《软件架构设计：大型网站技术架构与业务架构融合之道》

《软件架构的艺术》

《架构即未来：现代企业可扩展的 Web 架构、流程和组织》

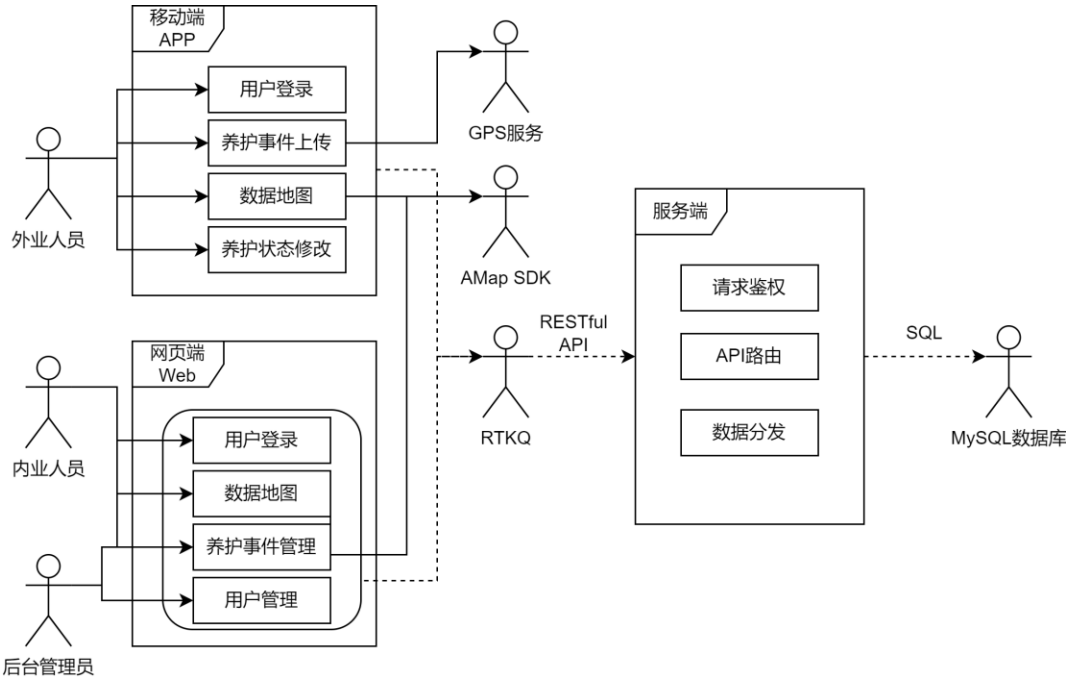
1.5 概述

本文档按照用例视图、逻辑视图、进程视图、部署视图和技术视图和数据视图构架表示方式的顺序，对本项目的软件架构进行了详细的描述。

2. 构架目标和约束

1. 跨平台。本项目的移动端需要同时在 Android 系统和 IOS 系统中实现，因此需要保证平台适配的兼容性。
2. 代码复用性。本项目的网页端和移动端有部分组件和 API 可以重用，需要提高代码的复用性以避免代码冗余。
3. 统一的 RESTful API。前端和后端的交互通过 RESTful API 实现。
4. 开发语言和框架。前端使用 React 和 React Native 实现，后端使用 Python 实现。

3. 用例视图



外业人员（移动端）：

- 用户登录：用户登录、个人信息查看和 Token 的获取。
- 数据地图：借助 AMap SDK 中间件获取当前的养护事件信息并以点标记的形式显示在地图上。
- 养护事件上传：用户填写待养护路段信息，通过 GPS 服务获取当前位置，并将数据上传至服务端。
- 养护状态修改：养护完成后，修改养护事件状态并同步数据到服务端。

内业人员（网页端）：

- 用户登录和数据地图：与外业人员相同。
- 养护事件管理：查看并确认养护事件信息，养护事件的分配和验收，自动化养护事件分类（深度学习模型调用）。

后台管理员（网页端）

- 养护事件管理：与内业人员相同。
- 用户管理：用户的新增、激活、删除和用户权限的修改。

4. 逻辑视图

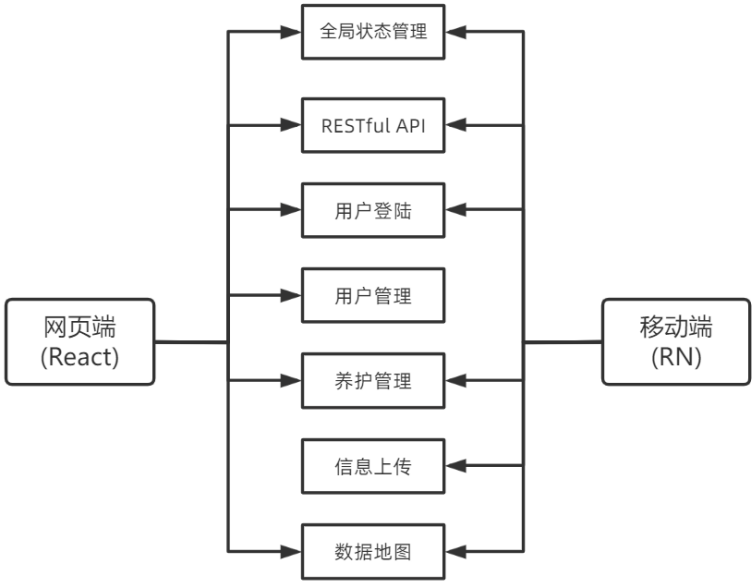
4.1 概述

设计模型被分为前端和后端两个部分，其中前端又可分为外业人员使用的移动端和内页人员和管理人员使用的网页端。两者使用前后端分离的结构实现，前端仅实现数据请求和界面的渲染，后端负责数据的存储和处理，两者通过 RESTful API 进行交互。

4.2 在构架方面具有重要意义的设计包

4.2.1 网页端和移动端

网页端和移动端共分为全局状态管理、API、用户登录、用户管理、养护管理、信息上传、数据地图等模块。其中网页端和移动端分别包含的模块如下图。其中两者具有类似功能的模块较多，但由于二者使用框架和界面的不同，部分模块需要做单独实现。项目使用 RTK 进行全局状态管理，并对部分状态进行本地存储。RESTful API 模块主要依赖 RTKQ 库解决数据请求和刷新，以及数据缓存。全局状态管理和 API 模块基本可以复用。



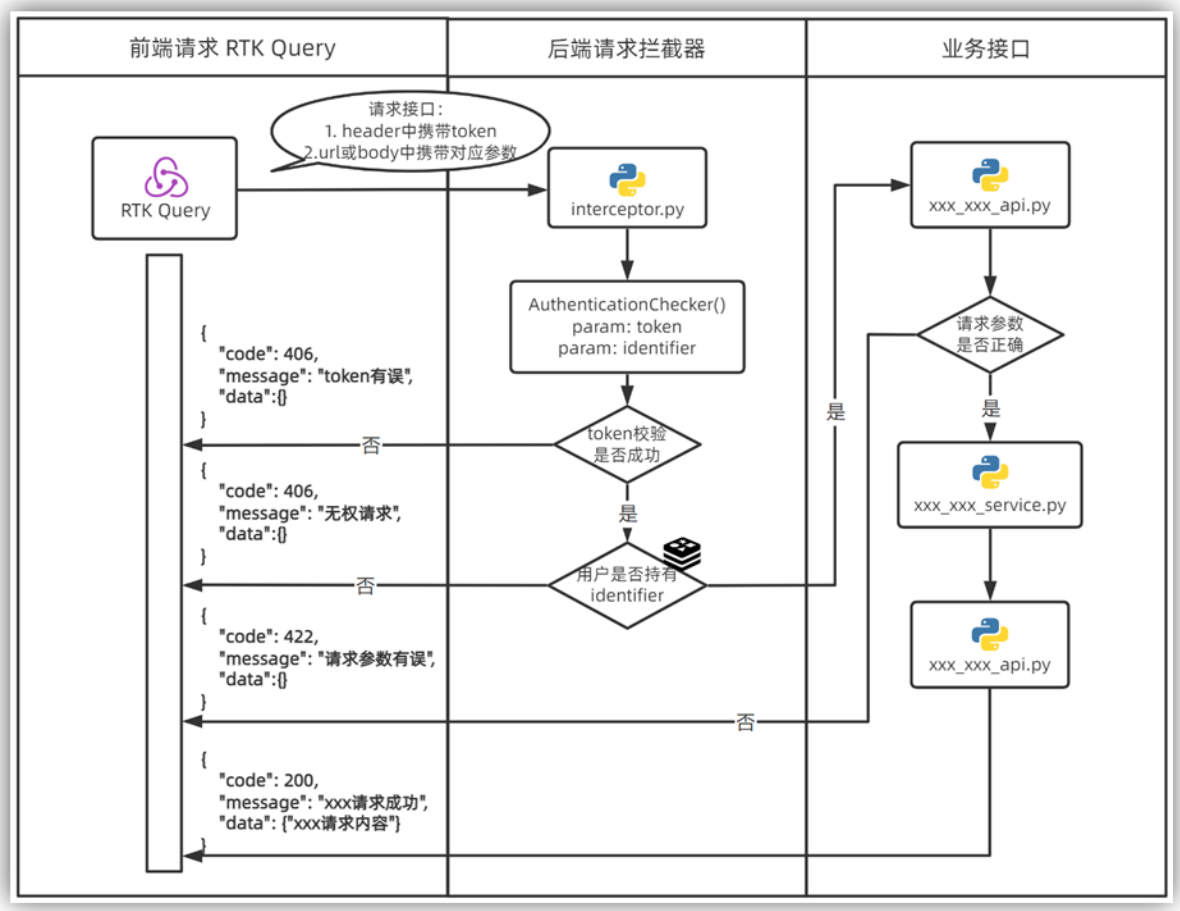
4.2.2 服务端

4.2.2.1 授权认证与权限判断模块

授权认证与权限管理是本系统后端业务逻辑层的入口模块，会对前端发送的请求进行两步校验判断：

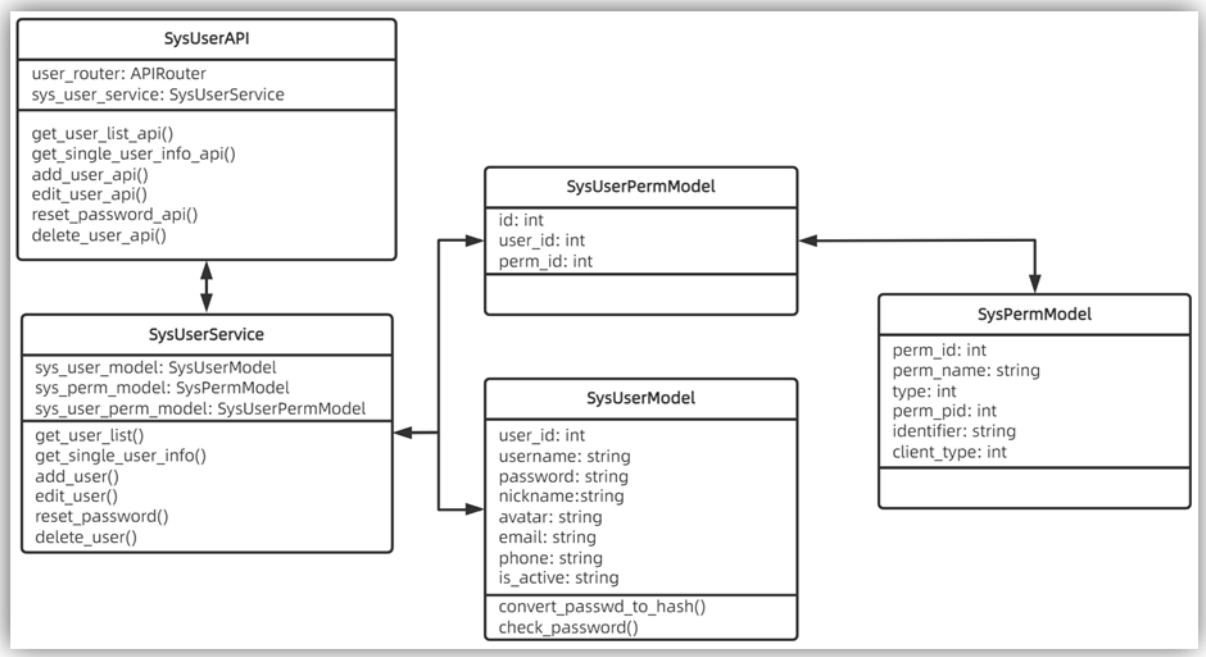
1. 请求是否携带 token，并且 token 中的负载是否为系统用户的 username；
2. 该系统用户是否拥有请求该接口的权限

若两步校验中抛出异常，则对应返回前端 4XX 错误码以及对应信息，若两步校验通过，则调用接口对应的实现方法，执行相应的业务流程。具体的运行流程如下。

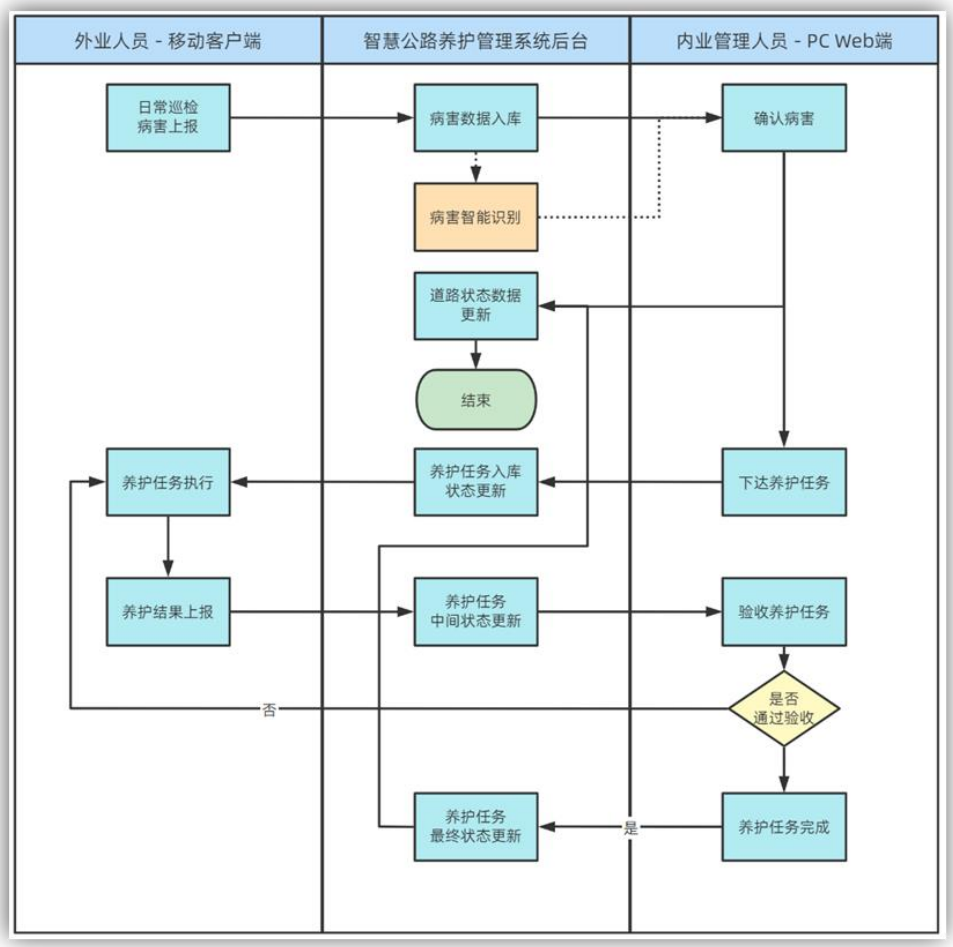


4.2.2.2 用户管理模块

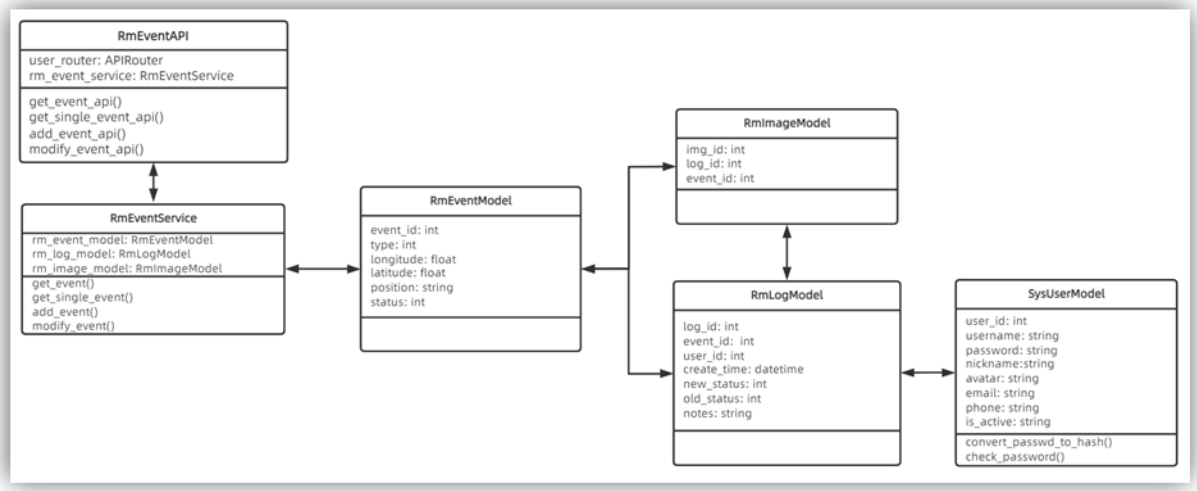
用户管理模块采用分层设计，SysUserAPI 对象获取请求，根据请求路径调用对应的实现方法，若实现方法中未抛出异常，则由 SysUserAPI 创建 response 对象，返回给前端；SysUserService 对象中的各个方面，对应 SysUserAPI 的具体业务逻辑，实现方法中涉及数据库的操作由 ORM 框架提供的对象来完成操作，若在业务逻辑执行中产生异常，则直接在 SysUserService 对象中抛出 HTTP 异常，返回给前端；用户管理涉及到用户表、用户权限表、权限表，分别由 SysUserModel、SysUserPermModel、SysPermModel 实现，这三个对象继承 peewee ORM 框架的基类，使得其拥有操作数据库的方法。用户管理模块的类与方法关系如下图所示。


SysUserAPI
user_router: APIRouter
sys_user_service: SysUserService
get_user_list_api()
get_single_user_info_api()
add_user_api()
edit_user_api()
reset_password_api()
delete_user_api()
SysUserPermModel
id: int
user_id: int
perm_id: int
SysPermModel
perm_id: int
perm_name: string
type: int
perm_pid: int
identifier: string
client_type: int
SysUserService
sys_user_model: SysUserModel
sys_perm_model: SysPermModel
sys_user_perm_model: SysUserPermModel
get_user_list()
get_single_user_info()
add_user()
edit_user()
reset_password()
delete_user()
SysUserModel
user_id: int
username: string
password: string
nickname:string
avatar: string
email: string
phone: string
is_active: string
convert_passwd_to_hash()
check_password()

4.2.2.3 养护管理模块



养护管理模块采用分层设计，并且参考业务流程进行设计。RmEventAPI 对象获取前端请求，根据请求路径调用对应的实现方法，若实现方法中未抛出异常，则由 RmEventAPI 创建 response 对象，返回给前端；RmEventAPI 提供的 add_event_api()方法，对应业务流程中的外业人员的日常巡检病害上报流程，modify_event_api()方法通过传入参数的不同，可以分别实现内业人员进行确认病害、下达养护任务，实现外业人员执行养护任务等。get_event_api()、get_single_event_api()用于养护管理界面,包括事件列表、数据地图大屏等，为这些功能提供数据。API 层的实现逻辑由下层的 RmEventService 对象获取，实现方法中涉及数据库的操作由 ORM 框架提供的对象来完成操作，若在业务逻辑执行中产生异常，则直接在 RmEventService 对象中抛出 HTTP 异常，返回给前端。

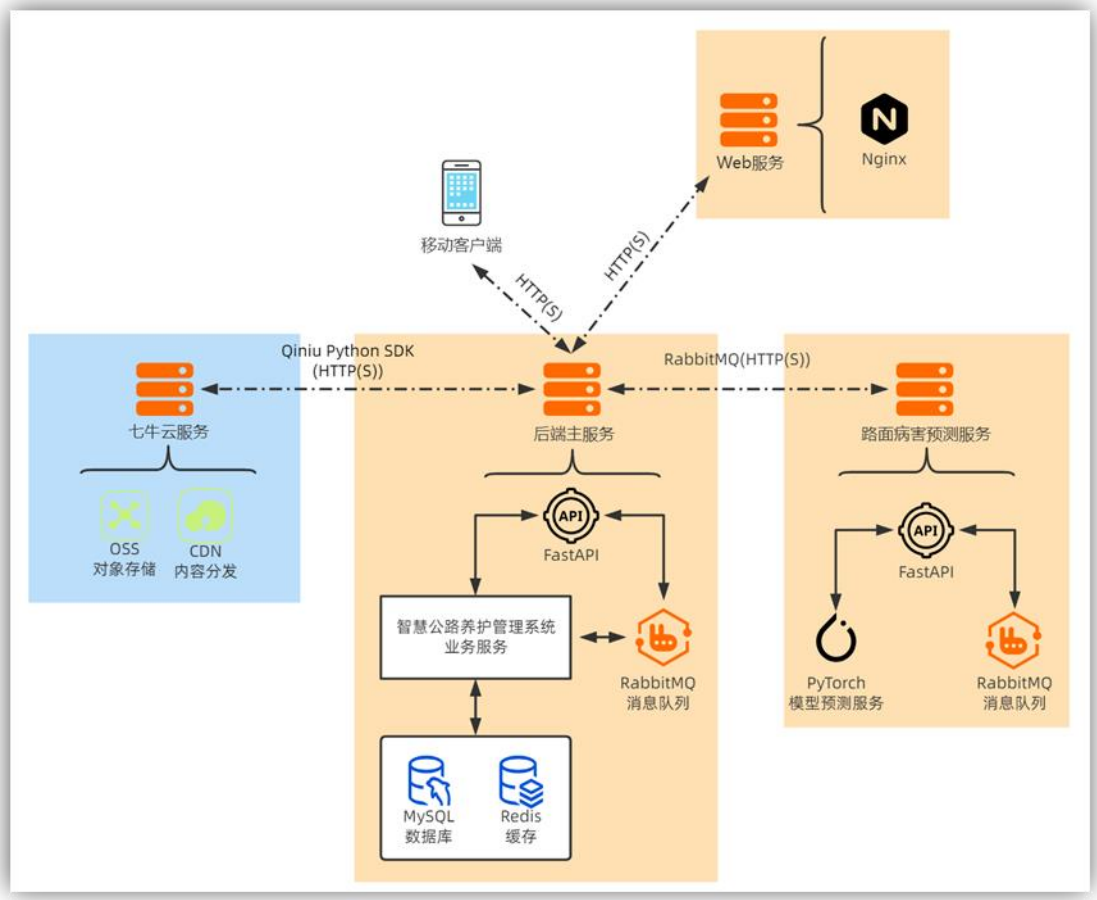


5. 进程视图

在选用开发框架时使用了 React，React Native 和 FastAPI 等框架，各类异步调用、多进程和线程的调度分配已在框架内部实现，因此不再需要专门的进程视图。

6. 部署视图

智慧公路养护管理系统使用 C/S 和 B/S 架构混合，并采用分布式部署方式，将管理系统前端网页、管理系统后端业务主服务、道路病害预测服务部署在多台服务器中，服务器之间采用 HTTP(S)进行交互。本项目涉及到外部的存储服务以及 CDN 服务，使用七牛云提供的 Python SDK 进行相应功能的调用。


Web服务
Nginx
移动客户端
HTTP(S)
HTTP(S)
Qiniu Python SDK
(HTTP(S))
RabbitMQ(HTTP(S))
七牛云服务
后端主服务
路面病害预测服务
OSS
对象存储
CDN
内容分发
API
FastAPI
API
FastAPI
智慧公路养护管理系统
业务服务
RabbitMQ
消息队列
PyTorch
模型预测服务
RabbitMQ
消息队列
MySQL
数据库
Redis
缓存

7. 技术视图

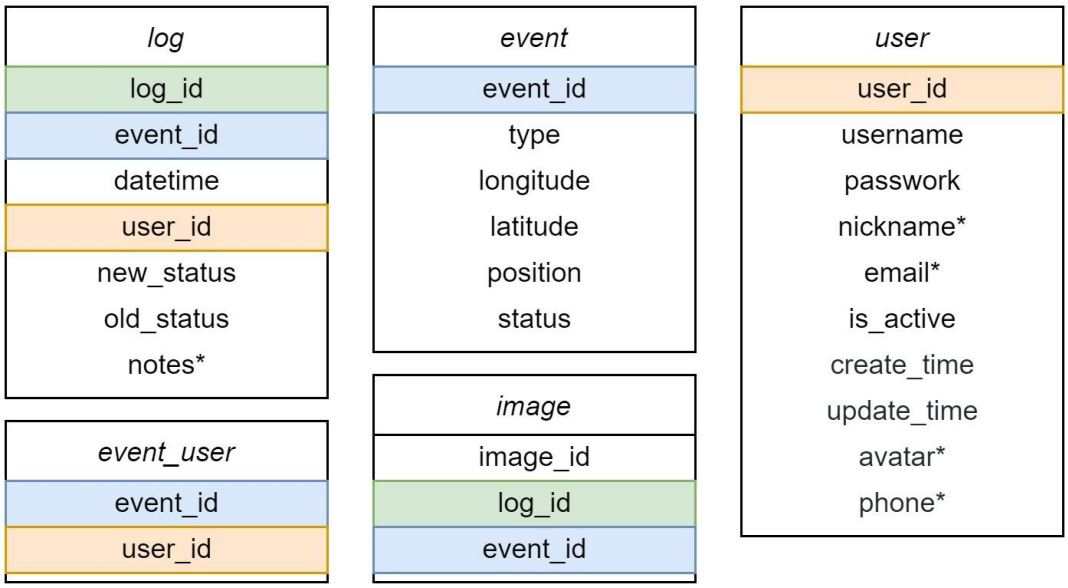
技术视图从编程语言、数据存储依赖、业务框架、中间件、部署依赖、用户界面实现 6 个层面规定了智慧公路养护管理系统的技术架构。

智慧公路养护管理系统采用前后端分离的架构进行开发，其中前端使用 TypeScript 语言开发，Web 界面使用 React 框架开发，UI 元素主要使用 Ant Design UI 库，打包后的静态资源使用 Nginx 部署。App 端使用 React Native 框架进行开发，UI 元素主要使用 React-Native-Element UI 库，打包后的客户端以安装包的形式导出。前后端数据交互使用 RTK Query，并使用 AMap 中间件用户地图数据的渲染和展示。

后端工程使用 Python 语言开发，使用 FastAPI 高性能 Web 框架构建 API，涉及数据库 CRUD 的业务逻辑，统一使用 peewee ORM 框架完成。项目涉及到深度学习相关功能，使用 PyTorch 深度学习框架实现业务功能。在本文档的规划时期，后端将采用 MQ 架构设计，业务主服务与深度学习预测服务分别部署，两端的交互使用 RabbitMQ，具体可参考部署视图。项目生产运行时产生的大量图片数据，使用 OSS 存储，需调用时使用 CDN 服务，目前已购买七牛云相关服务。

8. 数据视图

根据项目需要，我们将关系型数据库中的数据表各字段设计如下图。其中高亮的字段为外键，带“*”号的字段可以为空。



* 表示该项数据可为空

9. 大小和性能

在实现的常用功能中，1000 个并发用户请求，最长响应时间为 3 秒。在进行较长时间的异步请求时，提供进度条或者正在加载的信息。

10. 质量

10.1 性能设计

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。在设计关系型数据库时，尽量满足常见的设计范式，减少数据冗余，提高数据查询效率。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。同时我们根据各类接口调用的频率，调整了部分数据表的结构，从而对常用接口的调用进行了特别的优化，达到性能的提升。

10.2 易用性设计

在界面设计上，我们根据目前流行的 UI 设计原则，为用户提供尽量简洁和符合用户操作逻辑的页面和便捷的操作。如在设定地名字符串时提供自动补全，在地图上以点标记的形式提示数据信息，通过 DashBoard 的形式概览当前的各类状态等。

10.3 可修改性设计

在前端，我们提供了统一的 API 和全局状态管理，可以对网页端和移动端的数据获取和存储逻辑进行统一的修改。对于用户界面的设计，我们也采用组件化的设计思想，可以用于组件的复用。在后端，我们使用 Peewee 中间件，屏蔽了底层数据库种类和结构的差异，可以很方便的切换所需的数据库类型。此外我们还在后端使用了分层结构，具有较强的可扩展性。