
# 青训营前端结业项目答辩汇报文档

## 一、项目介绍

Root：一个轻量、快速的全栈前端监控系统。

项目服务地址：https://monitor.xiaotianxt.com/login

Github 地址：

- https://github.com/youth-camp-root/web-monitor-dashboard（前端）
- https://github.com/youth-camp-root/monitorSDK（埋点代码与测试应用）
- https://github.com/youth-camp-root/web-monitor-backend（后端）

## 二、项目分工

| 团队成员 | 主要贡献 |
| --- | --- |
| 郑琬仪 | 异常概览、总概览和其他页面和接口、页面异常捕获sdk、项目进度与其他事项 |
| 林佑光 | 用户细查及总概览页面、相关前后端接口、用户行为捕获sdk |
| 田雨沛 | 用户详情页面、相关前后端接口、Flask后端框架搭建、项目自动化构建部署、辅助组员完成代码 |
| 熊恺杰 | 异常详情页面、相关前后端接口、MongoDB数据库搭建、sdk打包、辅助组员完成代码 |
| 熊康伟 | 性能页面、相关前后端接口、页面性能sdk |

| | |
|---|---|
| 龚恒 | 异常详情页面、相关前后端接口、HTTP请求监控sdk及打包 |

# 三、项目实现

## 3.1 技术选型与相关开发文档

> 可以补充场景分析环节，明确要解决的问题和前提假设，比如按当前选型和架构总体预计需要xxx存储空间，xxx台服务器.....。

本项目主要是实现一个前端监控系统，使用者将本项目开发的埋点代码放置在需要被监控的前端项目中，用户的行为数据会被记录，使用者可以通过一个中台页面监控项目情况。

### 3.1.1 前端

技术选型：Vue3 + Vite + TypeScript + Arco Design

### 3.1.2 后端

Flask + MongoDB

### 3.1.3 监控SDK

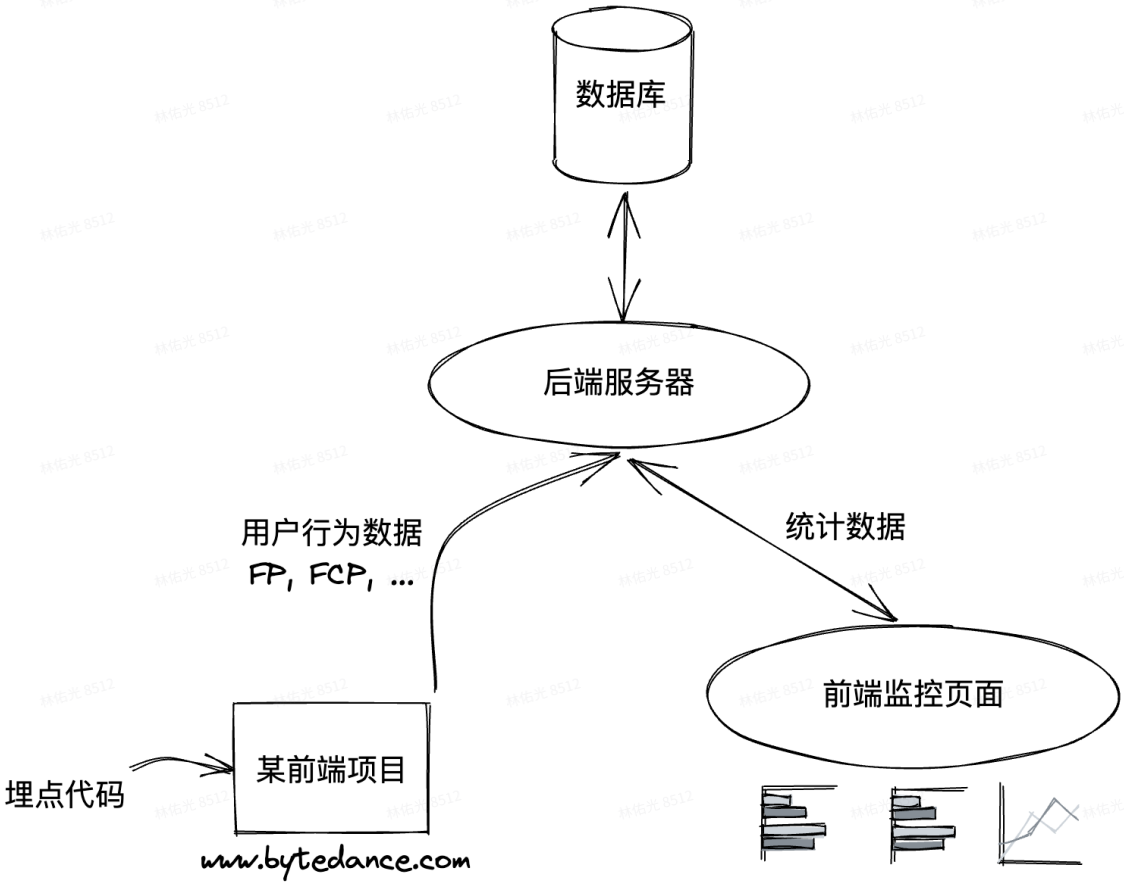
Webpack

### 3.1.4 其他

- 为提升代码的可读性和可维护性，我们在前端监控系统仓库中使用 TS + ESLint 来规范成员的代码风格；为方便统计成员每次提交的内容，使用 commitlint 来规范仓库的提交信息。
- 网站监控系统项目中，使用 flask+mongodb 搭建数据服务器并模拟数据；在监控SDK项目中，通过设置 webpack 的请求中间件来实现虚拟服务器。

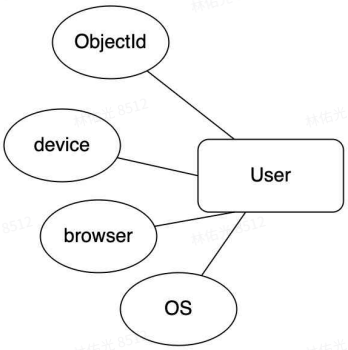
## 3.2 架构设计

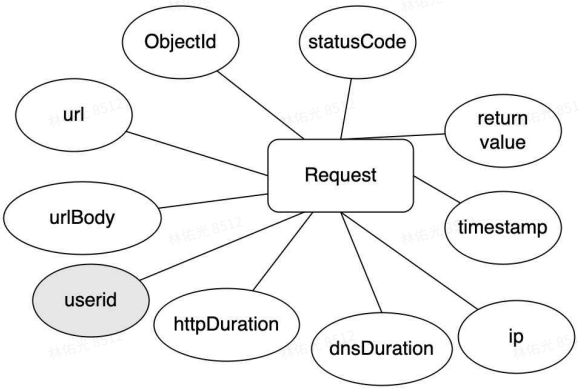
> 可以补充场景分析环节，明确要解决的问题和前提假设，比如预计0.5%的用户属于大V，粉丝很多，也会经常上传视频，当前架构的解决方案是xxx。

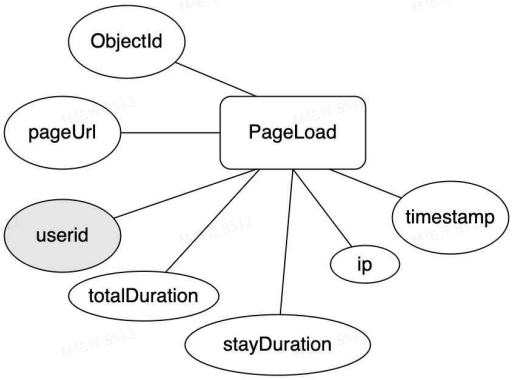


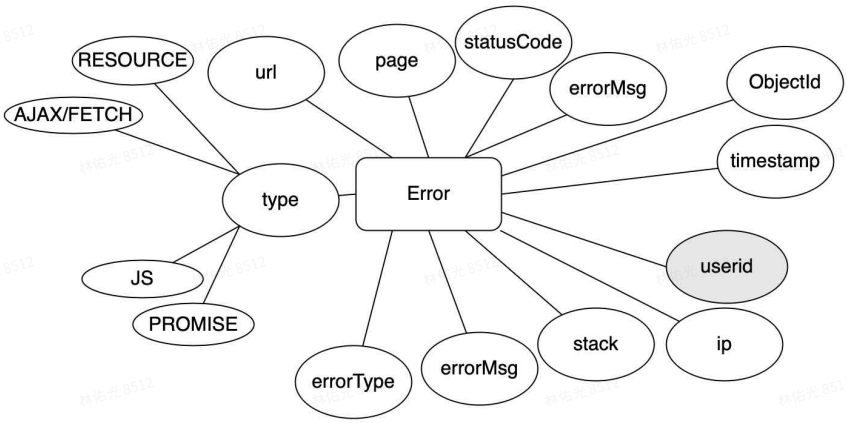

- 为了使多人协同的项目进行部署时快捷高效，我们加入了自动化部署脚本
- 为了在有限的时间内快速开发具有良好交互特性和较美观的 UI 的网页，我们选择了字节的Arco Design组件库
- 为了良好的项目和组件结构，我们根据不同需求建立文件夹（概览、异常、用户、性能）内有 `index.vue` 以及组件文件夹以存储组件，前端架构耦合度较低，便于查找和管理

### 3.2.1 数据库设计




## 3.3 项目代码介绍

- 前端监控系统：

```
web-monitor-dashboard
├──.husky
├──config
└──src
    ├──api // api接口
    ├──assets // 静态资源
    ├──components // 通用组件
    ├──config // 项目配置
    ├──directive // 自定义指令库
    ├──hooks // 前端工具
    ├──layout // 基础布局
    ├──locale // 中英文编码
    ├──mock // 数据mock
    ├──router // 路由文件
    │   ├──appMenus
    │   ├──guard
    │   └──routes
    │       └──modules // 各个页面的router
    ├──store // 数据存储
```

```
    ├──types // 数据类型定义
    ├──utils // 数据类工具
    └──views
      ├──error // 错误页面
      │   ├──issue-details // 错误详情页面
      │   └──issues // 错误概览页面
      ├──login // 登入页面
      ├──not-found // not-found 页面
      ├──overview // 概览页面
      │   ├──components
      │   └──locale
      ├──performance // 性能页面
      │   ├──api // api性能及详情页面
      │   └──page // 页面性能及详情页面
      ├──user // 用户详情页面
      ├──useraction // 用户细查页面
      └──userinfo
```

- 后端：

```
web-monitor-backend
│  requirements.txt
└──api
    │  __init__.py
    ├──mock
    │  data_generator.py
    │  mock_data_forger.py
    ├──model // 数据表定义
    │  models.py
    ├──route
    │  api.py // 其他相关接口
    │  errors.py // 错误相关后端接口
    │  performance.py // 性能相关后端接口
    │  user.py // 用户相关后端接口
    │  mock_api.py
    └──util
      data_process.py
      utils.py
```

- 监控系统SDK：

```
monitorSDK
    └──lib
```

```
        blankError.js // 白屏异常
        jsError.js // 脚本异常
        promiseError.js // Promise 脚本异常
        resource.js // 资源加载异常
        stayTime.js // 监控用户行为
        xhr.js // 监控request
```

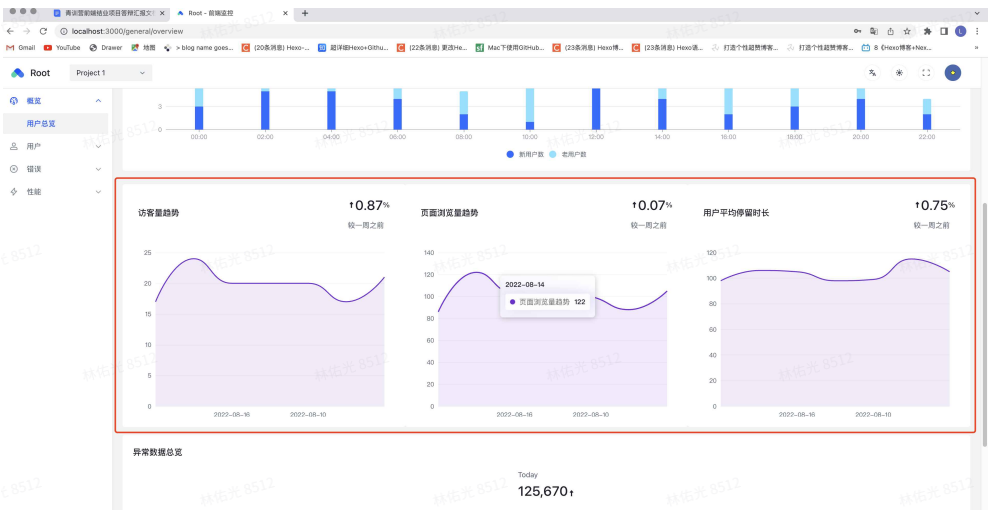
# 四、测试结果

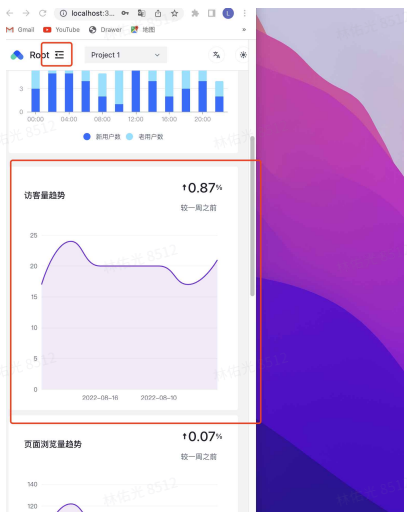
> 建议从功能测试和性能测试两部分分析，其中功能测试补充测试用例，性能测试补充性能分析报告、可优化点等内容。

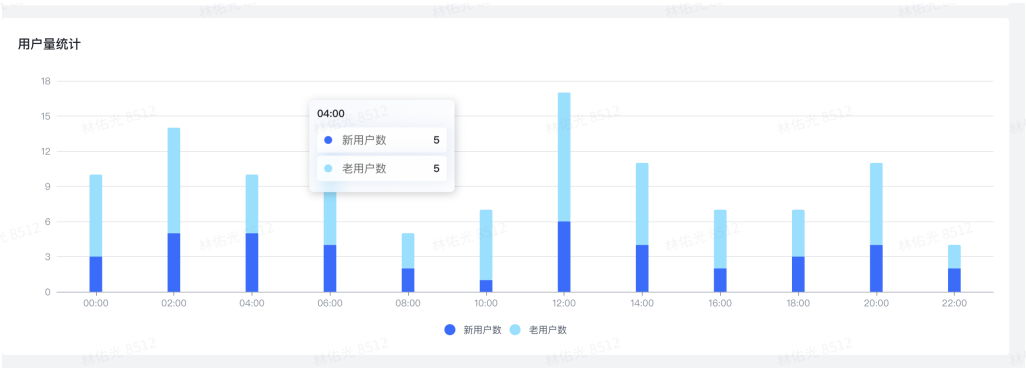
功能测试：

1. 用户登录、路由切换
2. 概览页面：流量数据、用户量统计、访客量趋势、页面浏览量趋势、用户平均停留时长、异常数据总览加载
3. 用户细查页面：用户列表显示、根据用户编号查询功能
4. 用户详情页面：用户相关信息显示
5. 错误概览页面：异常数据图标显示、异常数据列表及根据异常类型即网页查询功能
6. 页面性能页面：FP、FCP、DOM Ready、DNS渲染和页面列表
7. 接口性能页面：平均请求概览、总请求量、接口列表

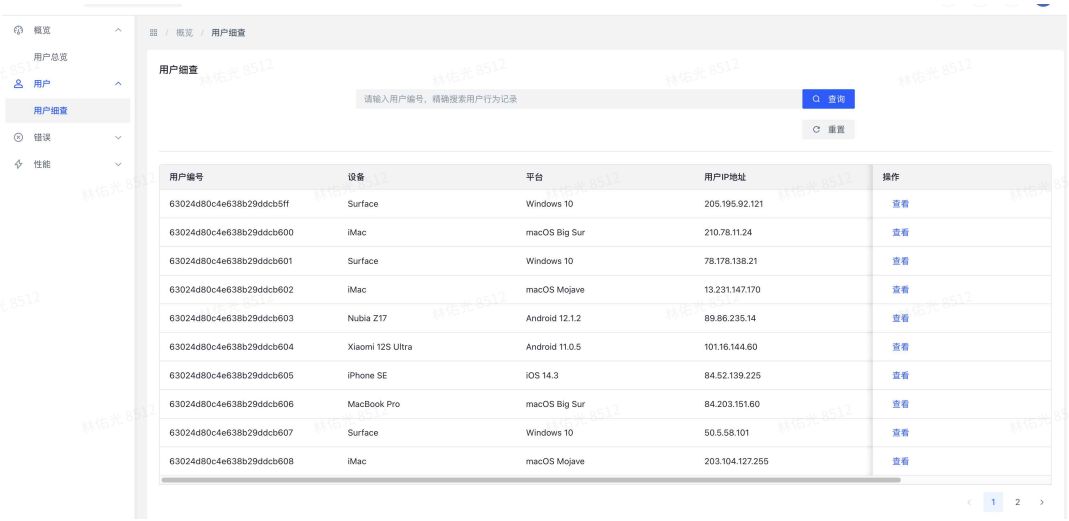
## 自适应测试

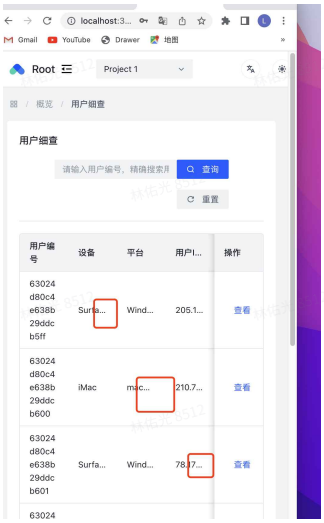


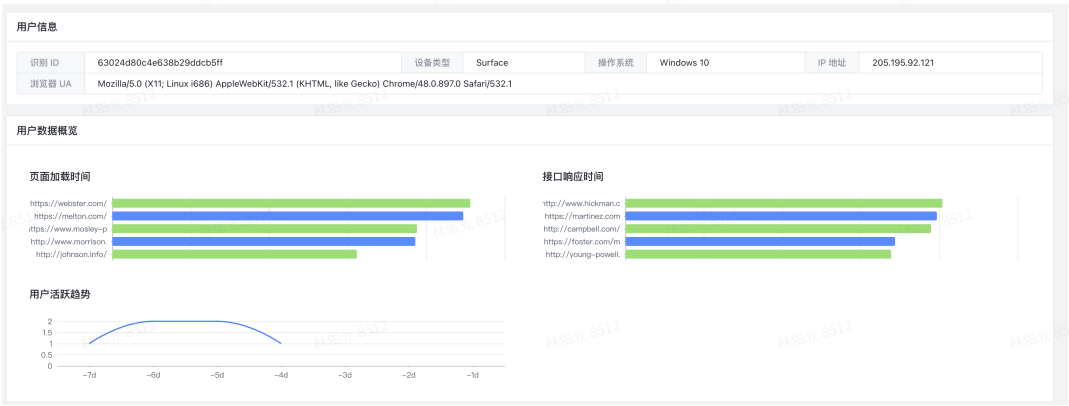











# 五、Demo 演示视频 （必填）

仅支持进入文档后查看

# SDK讲解


SDK讲解.mp4

## 六、项目总结与反思

1. 目前仍存在的问题
    a. 数据加载速度较慢
    b. 由于对 typescript 使用上的不熟悉，类型声明考虑不周；因人力和时间关系，已完成的通用可配置组件较少
    c. 缺少强制代码评审，再加上不同成员对代码熟练程度和风格不一致，导致页面的代码结构差异较大，代码质量可以改善。
2. 已识别出的优化项
    a. 针对服务器性能较弱的问题，应该定时整理日志的统计信息，在请求时直接返回，而不是每次都遍历全部日志进行统计。
    b. 项目的前端页面结构和功能不够完善，其中包括：缺乏切换监控不同项目、图表中可以选择更多不同的时间维度
    c. 数据结构设计存在改进空间，性能页面需要从多个表中获取不同的数据
3. 架构演进的可能性
4. 项目过程中的反思与总结

a. 大部分同学对ts、python、github操作不是很熟悉

b. 直接使用Arco Design Pro的代码为框架，但由于对代码的框架不是很熟悉导致出现问题

c. 后期没有实时跟踪成员的进度，导致不能在成员出现代码困难第一时间发现

d. 总结：经过这次项目，对前端监控的认识加深了很多，同时也进一步需要根据不同组员的能力进行良好和清晰的分工的重要性，尽管我们在很早就开始大项目，但是由于并没有很好组织代码提交进度，也面对了很多问题。无论如何，在大家的共同努力下依然完成了这次的大项目，辛苦大家了！

# 七、其他补充资料（选填）

## 7.1 用户行为

**PV**：用户浏览量

**UV**：独立访客数

**用户停留时长**：用户在页面的停留时长

**获取方法：**

通过监听事件，获取用户跳转页面的行为，然后将跳转页面，用户停留时间等参数发给后端，这个时候已经获取到页面停留时间，同时可以获取pv，因为我们这里简化一下，浏览量只包括浏览页面的话，那每跳转一个页面就是一个浏览量，pv就+1，然后uv就只要对当天的统计的pv采集到的userid进行去重就可以拿到，或者直接到数据库里进行获取就好。

**监听页面跳转方法**：主要分为两种情况，单页面与多页面应用

**单页面SPA**：分为Hash模式，History模式

Hash：基于浏览器的hashchange事件，地址变化时，通过window.location.hash 获取地址上的hash值；并通过构造Router类，配置routes对象设置hash值与对应的组件内容。

History：基于HTML5新增的pushState()和replaceState()两个api，以及浏览器的popstate事件，地址变化时，通过window.location.pathname找到对应的组件。并通过构造Router类，配置routes对象设置pathname值与对应的组件内容

**多页面MPA**：

主要基于 `onpageshow` (页面显示的时候)和 `onpagehide` (页面隐藏的时候)这两个API获取用户停留的时长。只要在 `onpageshow` 初始时间值，在 `onpagehide` 的时间求出差值，然后上传到后台就行。但是目前没有考虑用户直接关闭浏览器的情况。

**登录获取用户信息：**

在安装SDK时，会先判断 `localStorage` 里是否有用户ID，如果没有，会将请求页面发送给后端，然后后端获取用户的操作系统，设备信息以及IP信息后创建一个新用户并将新用户的ID返回给前端，前端获取到用户ID后将其存放于 `localStorage` 中。如果一开始就有用户ID，就直接从 `localStorage` 中获取。