青训营h-trackpoint项目答辩报告

一、项目介绍

> h-trackpoint是一个轻量级、使用简单、高性能的前端埋点监控平台。支持不同埋点类型和上报方式，并提供了丰富的项目、事件、参数管理和数据可视化界面；
>
> 项目地址：https://github.com/hfdy0935/h-trackpoint
>
> npm地址：https://www.npmjs.com/package/h-trackpoint
>
> 预览地址：Docker上运行后http://localhost:5173

二、项目分工

> 好的团队协作可以酌情加分哟～请组长和组员做好项目分工与监督。

团队成员	主要贡献
w	SDK开发、优化、测试
z	架构设计、后端核心功能开发、前端基本增删改查、可视化
x	数据可视化、用户行为分析
h	数据可视化、测试

三、项目实现

3.1 技术选型与相关开发文档

1. 埋点SDK
    - 构建工具：Rollup，支持cjs、esm方式导出使用
    - 主要依赖：
        1. axios：用于上报请求；
        2. ua-parser-js：用于解析用户设备信息；

> 3. html2canvas：上报页面截图；
> 4. jest：测试；
> 5. typescript：类型检查；
> 6. h-trackpoint：本项目的SDK

2. 后端
   - 语言及框架：Python、FastAPI
   - 数据库：MySql
   - 中间件：Redis
   - 文件存储：MinIO
   - 第三方库：

     > tortoise-orm
     >
     > fastapi-boot
     >
     > minio
     >
     > aiomysql
     >
     > pydantic
     >
     > pyjwt
     >
     > ……

3. 前端
   - 语言及框架：Typescript、Vue、Vite
   - 主要依赖：

     > 1. h-trackpoint
     > 2. antd vue
     > 3. axios
     > 4. leaflet
     > 5. antv/g2

3.2 架构设计

3.1 场景分析

1. 数据收集需求

- 点击行为：记录用户在页面上的点击行为，包括页面连接、屏幕宽高和点击位置；
- 页面性能：测量从请求发起至页面完全加载并可供用户互动所需的时间，包括DNS查询、TCP连接、请求、响应、渲染、加载事件等行为的耗时和JS对堆内存占用百分比等；
- 页面报错：上报页面中出现的报错，并按类型分组，以根据不同页面、时段进行精确查找；
- 网络请求：记录每次请求和响应的信息，如状态码、请求方法、url、请求耗时l等，以便分析接口性能；
- PV与UV统计：记录每个事件的触发记录和时间，并分时段统计PV和UV。
- 自定义事件：允许开发者根据业务需求定义特殊的行为或状态变化作为事件进行监测，例如特定流程完成、支付、点赞、收藏等，在控制台可以实现按参数查询。

2. 需要解决的问题

- 数据准确性：确保收集到的数据真实反映了用户的实际行为及系统表现，避免因代码实现不当导致的数据偏差。
- 数据上报机制：设计高效可靠的数据上报策略，即使在网络条件不佳的情况下也能尽可能保证数据的完整性和及时性。
- 性能影响最小化：尽量减少埋点和上报对网页加载速度和运行效率的影响。

3.2 项目设计

1.1 数据库

- 表设计：

> 1. 主要的表：用户表、项目表、默认事件表、自定义事件表、客户端设备表、参数表、上报记录表以及他们的关联表；
> 2. 考虑到事件埋点的多样性，用户可以给自定义事件添加参数约束以上报具体的参数，支持的类型对应到JS中包括string、number、boolean、Array、Object；
> 3. 对应关系：每个用户对应多个项目和事件，每个事件可以对应多个项目，每个参数也可以对应多个事件，通过中间表进行关联，以增加灵活性、降低耦合度；
> 4. 项目和事件都支持停用和启用两种状态，停用时对应的上报请求会失败，项目中不同状态的事件数量和事件绑定的不同状态的项目都可以在控制台实时显示；

- 索引设计：

  > 考虑到查询频率、字段是否唯一等因素，将项目表中的用户id、上报记录表中的事件id等字段设置为索引以加速查询；

- 主要字段设计：

  > 1. 项目表：项目id、key、项目名、描述、时间、状态等；
  > 2. 客户端设备表：记录操作系统及版本、浏览器及版本、设备类型、经纬度等；
  > 3. 参数表：id、参数名、描述、类型等；
  > 4. 上报记录表：上报时间、页面url、截图url、参数等；

1.2 其他中间件

- Redis
  - 基本缓存：储存用户注册和找回密码时的验证码；
  - 布隆过滤器：

i. 考虑到自定义事件被很多用户添加，且设计为不可修改，因此将自定义事件的id添加到布隆过滤器中，在用户上报自定义事件之前进行验证，阻挡一部分不合理的上报请求；

ii. 上报截图时将截图的标识（根据页面宽高、url和点击位置编码得到）存到布隆过滤器中，每次点击后判断是否已上报，只有不存在截图才需要前端继续上报截图；

- MinIO
  - 用于存储点击事件的截图，数据库中记录；

3.3 设计详情

1.1 基础功能

- 埋点SDK

1. 用项目的id和key注册之后，自动收集设备信息并上报性能指标，且后续可以通过唯一的事件名上报该项目下的事件；
2. 自动捕获和上报项目中的报错；
3. 可以随时添加通用参数进行上报；

- 埋点数据服务

1. 数据库表结构合理，可以进行灵活的增删改查和分析，易扩展，便于管理和维护；
2. 使用索引加速查询，提升响应性能；

- 埋点平台

1. 用户登陆、注册、找回密码、修改信息等；
2. 用户对项目、自定义事件、参数、上报记录的增删改查，其中上报记录可通过上报的参数值进行过滤；
3. 数据看板分为项目统计、性能监控和用户分析三部分，使用柱状图、折线图、饼图、词云图等展示上报事件统计信息；同时还添加了地图以表示客户端设备分布，使用热力图表示不同页面用户点击位置的分布；

1.2 进阶功能

- 埋点SDK

1. 上报SDK实现了同时上报请求的最大限制、埋点合并上报队列以及离开页面前剩余事件的一次性上报；
2. 通过性能指标分析DNS解析、TCP连接、页面渲染等事件的耗时，以主力快速定位页面加载问题；

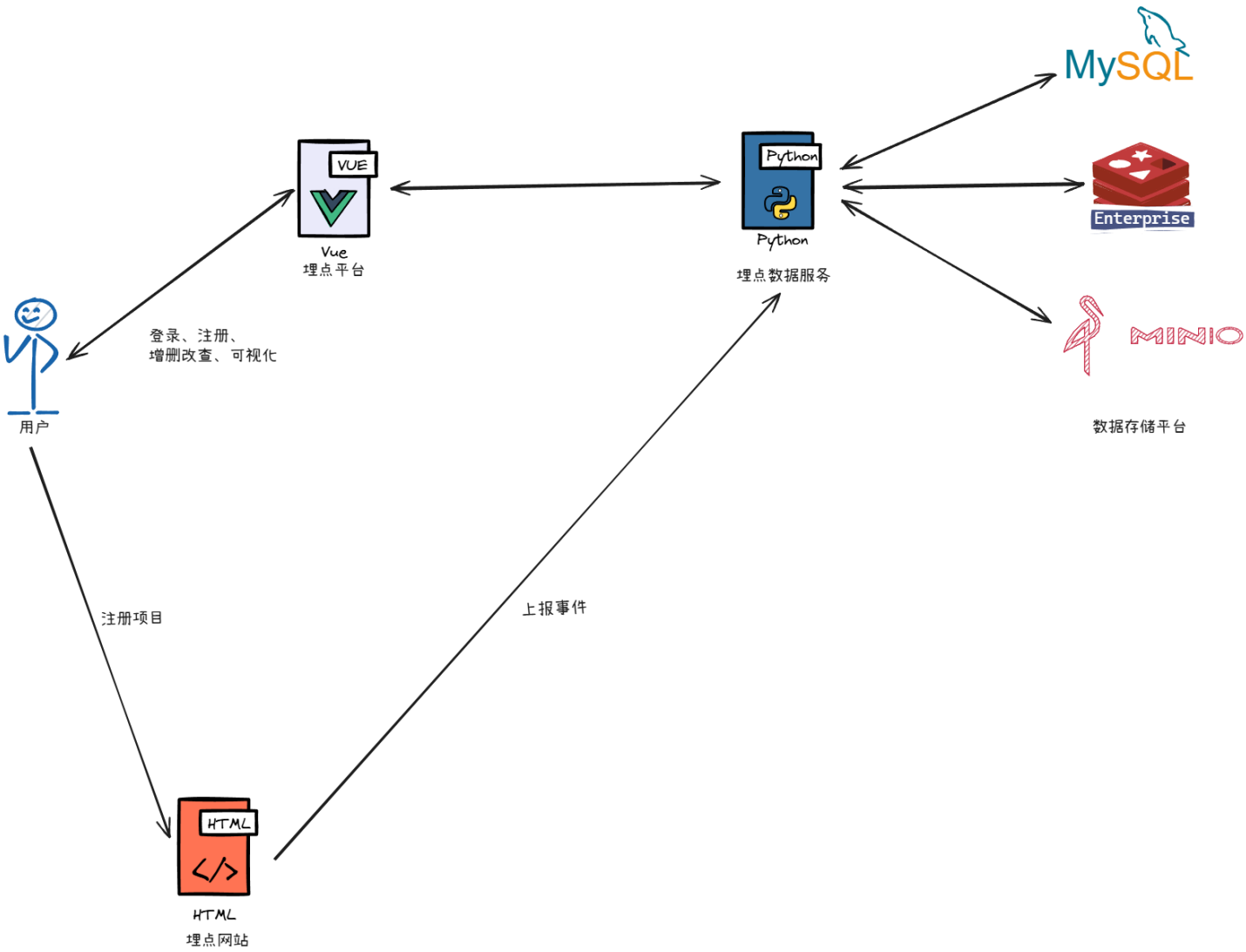
- 埋点数据服务

> 1. 对于无需截图的事件，前端上报之后后端直接返回“已提交”，使用`backgroundTasks`和事件循环进行上报，以减少上报耗时和对页面性能的影响，未来会考虑加入消息队列进行处理，以提升性能、解耦上报请求和记录入库操作；
> 2. 合理使用索引加速查询；
> 3. 使用MySQL的JSON格式储存上报的参数，避免创建大量的参数记录，同时便于查询；

- 埋点平台

> - 支持不同格式图表的切换和不同筛选条件的组合；

3.3 项目流程



3.3 项目代码介绍

1.1 SDK

```
├─coverage // 测试覆盖率报告
│  └─lcov-report
│      ├─impl
```

```
4  │       ├─lib
5  │       │  ├─impl
6  │       │  └─util
7  │       └─util
8  ├─dist // 打包产物
9  │  ├─cjs
10 │  ├─esm
11 │  └─types
12 │       ├─enum
13 │       ├─impl
14 │       ├─type
15 │       └─util
16 ├─lib // 源码
17 │  ├─enum
18 │  ├─impl // 具体实现
19 │  ├─type
20 │  └─util
21 ├─node_modules
22 ├─types // 类型
23 │  ├─enum
24 │  ├─impl
25 │  ├─type
26 │  └─util
27 └─__tests__ // 测试文件
```

1.2 后端

```
1  │  .gitignore
2  │  constants.py // 常量
3  │  dependencies.py // 公共依赖
4  │  docker-compose.yml // docker compose 启动文件
5  │  enums.py //枚举
6  │  event.py // 项目启动事件
7  │  exception.py // 自定义一场
8  │  helper.py // 工具类
9  │  main.py // 启动文件
10 │  README.md
11 │  requirements.txt
12 │  utils.py
13 │
14 ├─bean // 注入Bean
15 │  │  cache.py
16 │  │  config.py
17 │
18 ├─controller // Controller
```

```
19  │   │  client.py
20  │   │  event.py
21  │   │  project.py
22  │   │  record.py
23  │   │  resource.py
24  │   │  test.py
25  │   │  user.py
26  │   │
27  │   ├─data // 数据展示的接口
28  │   │   │  performance_monitor.py
29  │   │   │  proj_stat.py
30  │   │   │  user_analysis.py
31  │
32  ├─dao // 数据库操作
33  │   │  event.py
34  │   │  project.py
35  │   │
36  │   ├─data
37  │   │   │  project_stat.py
38  │   │   │  user_analysis.py
39  │
40  ├─domain // 接收、传输、相应实例的类
41  │   │  config.py
42  │   │
43  │   ├─bo
44  │   │   │  client.py
45  │   │   │  event.py
46  │   │   │  project.py
47  │   │   │  record.py
48  │   │
49  │   ├─dto
50  │   │   │  client.py
51  │   │   │  common.py
52  │   │   │  event.py
53  │   │   │  project.py
54  │   │   │  record.py
55  │   │   │  user.py
56  │   │   │
57  │   │   ├─data
58  │   │   │   │  common.py
59  │   │   │   │  user_behavior.py
60  │   │
61  │   ├─entity // 数据库实体类
62  │   │   │  bind_param.py
63  │   │   │  client.py
64  │   │   │  constants.py
65  │   │   │  custom_event.py
```

```
66  │   │   │  default_event.py
67  │   │   │  eventbind_param.py
68  │   │   │  event_project.py
69  │   │   │  project.py
70  │   │   │  record.py
71  │   │   │  user.py
72  │   │
73  │   ├─vo
74  │   │   │  client.py
75  │   │   │  common.py
76  │   │   │  event.py
77  │   │   │  project.py
78  │   │   │  record.py
79  │   │   │  user.py
80  │   │   │
81  │   │   ├─data
82  │   │   │   │  common.py
83  │   │   │   │  performance_monitor.py
84  │   │   │   │  project_stat.py
85  │   │   │   │  user_analysis.py
86  ├─resources // 资源、配置
87  │   │  22973563910f1eda3d79254d67c215aa.png
88  │   │  config.docker.yml
89  │   │  config.yml
90  │   │  init.sql
91  │   │  regiater_email_template.html
92  ├─service // Service
93  │   │  client.py
94  │   │  email_service.py
95  │   │  event.py
96  │   │  minio.py
97  │   │  project.py
98  │   │  record.py
99  │   │  resource.py
100 │   │  user.py
```

1.3 前端

```
1  ├─dashboard // dashboard页面入口
2  ├─public
3  └─src
4      ├─api // 封装发送请求的函数
5      │  └─v1
6      │      └─data
7      ├─assets
```

```
├─components // 公共租界
│  └─header
├─constant
├─enum
├─pages // 页面
│  ├─dashboard // 控制台
│  │  ├─component
│  │  ├─composable
│  │  ├─directive
│  │  ├─layout
│  │  │  └─component
│  │  │      ├─header
│  │  │      └─side-menu
│  │  ├─router
│  │  └─views
│  │      ├─data // 数据展示页面
│  │      │  ├─performance-monitor // 性能监控
│  │      │  │  └─component
│  │      │  ├─project-overview // 项目总览
│  │      │  └─user-analysis // 用户分析
│  │      │      ├─user-behavior-plot
│  │      │      └─user-props-plot
│  │      │          └─user-distribution-plot
│  │      ├─event // 事件管理
│  │      │  └─component
│  │      │      ├─create-update-event-modal
│  │      │      └─table-body-cell
│  │      ├─main
│  │      ├─project // 项目管理
│  │      │  └─component
│  │      │      ├─create-project-modal
│  │      │      └─table-body-cell
│  │      ├─record // 上报记录管理
│  │      │  └─component
│  │      │      ├─params-filter
│  │      │      ├─plot
│  │      │      └─table
│  │      └─test // 调试SDK
│  │          └─component
│  │              ├─custom-event
│  │              └─default-event
│  │                  └─data
│  └─site // 网站入库
│      ├─layout
│      │  └─header
│      │      └─account-modal
│      ├─router
```

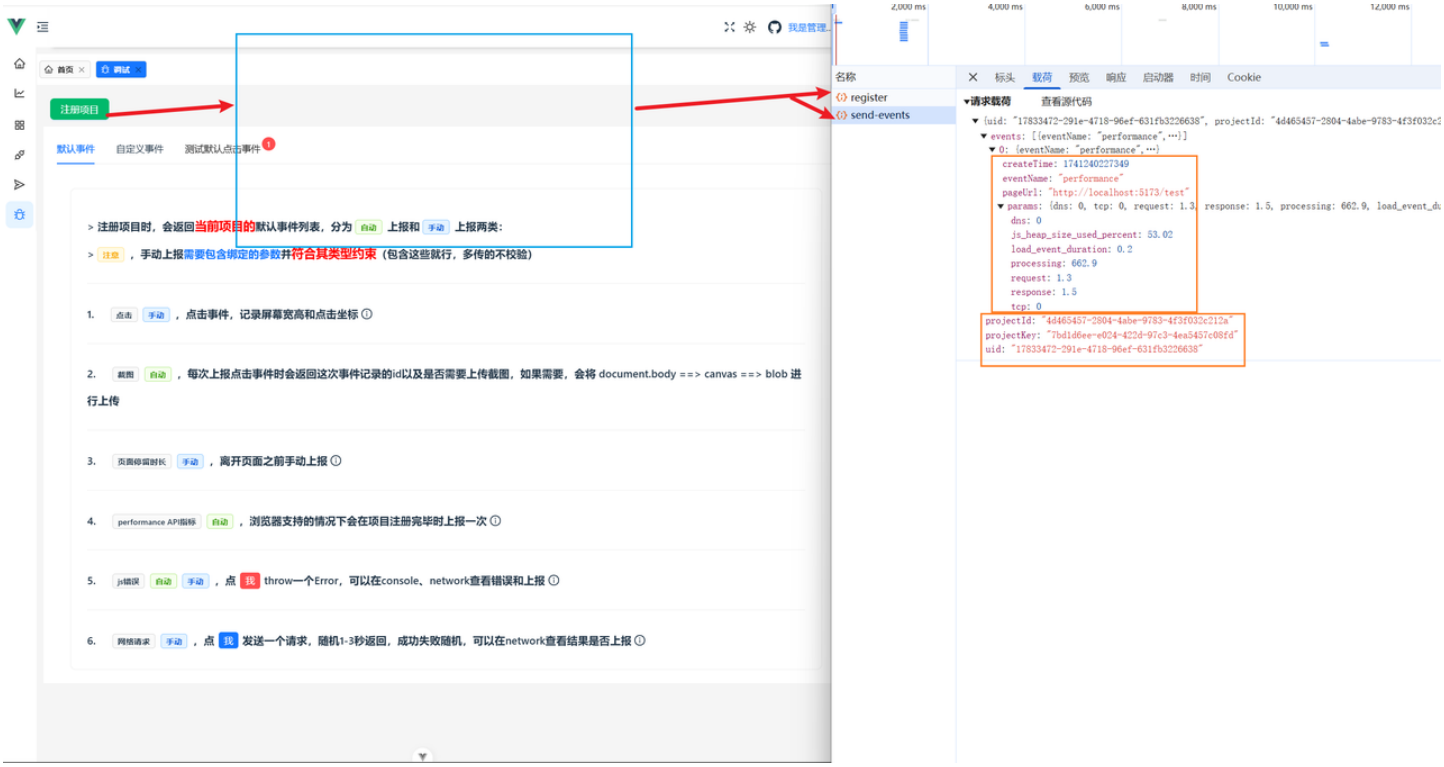
```
55      |       └─views
56      |           └─main
57      ├─store // 公共仓库
58      |   └─module
59      ├─type // 类型
60      |   └─data
61      └─util
62      └─index.html // 主页面入口
```

四、测试结果

- **控制台自带了一个上报测试页面，可直接用于测试所有默认和自定义事件；**

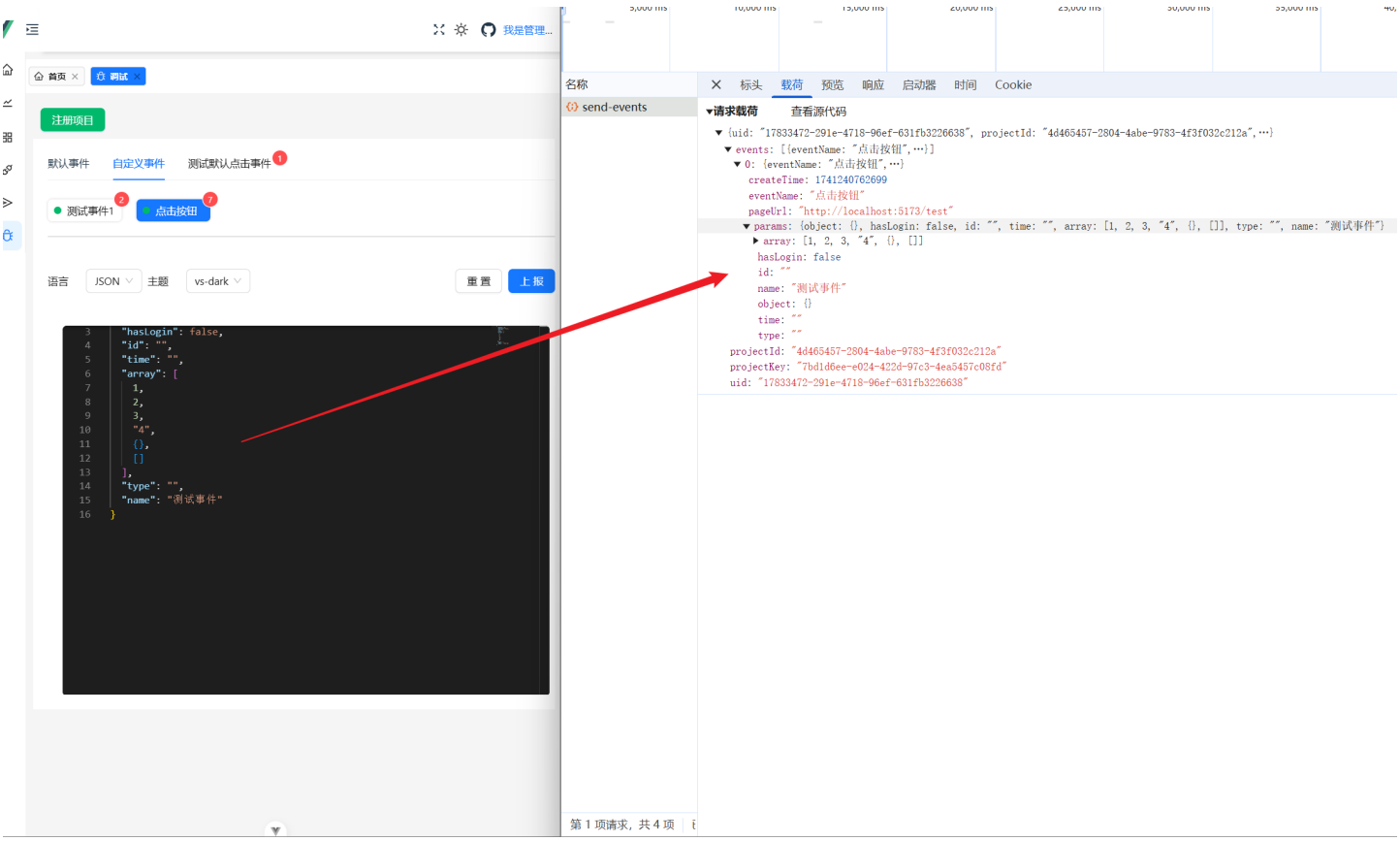
注册

输入项目id和key，验证通过之后会自动发送一次性能事件，如下图：



请求合并上报

上报自定义事件

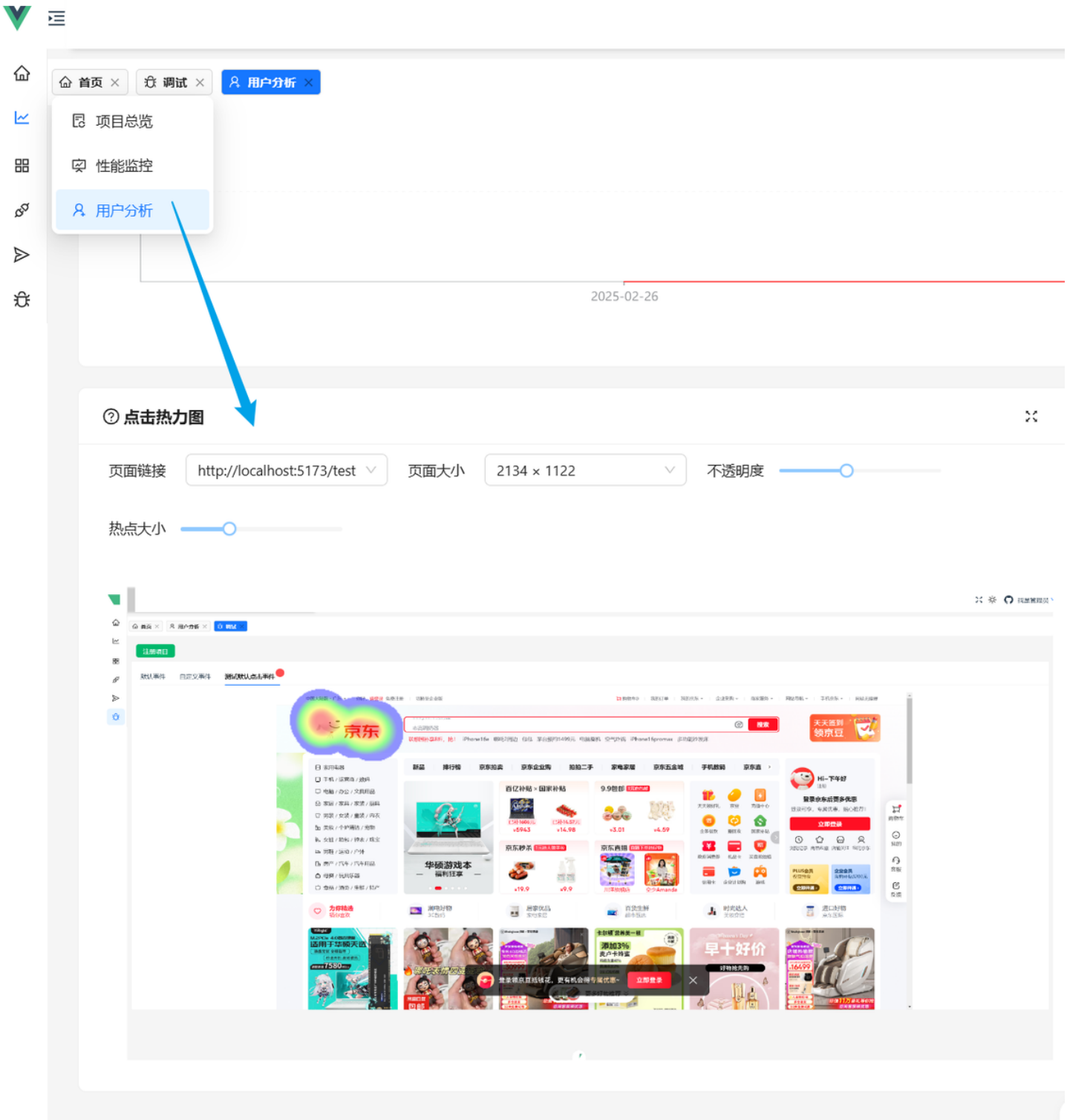


可以选json/yaml格式，切换不同主题

测试点击事件

> 点击logo位置，第一次上报了截图，之后后端把截图id存到布隆过滤器中，之后的请求没有上报截图；

查看点击热力图，符合之前点的位置，可以自定义不透明度和热点大小；



- **使用jest对SDK进行测试**

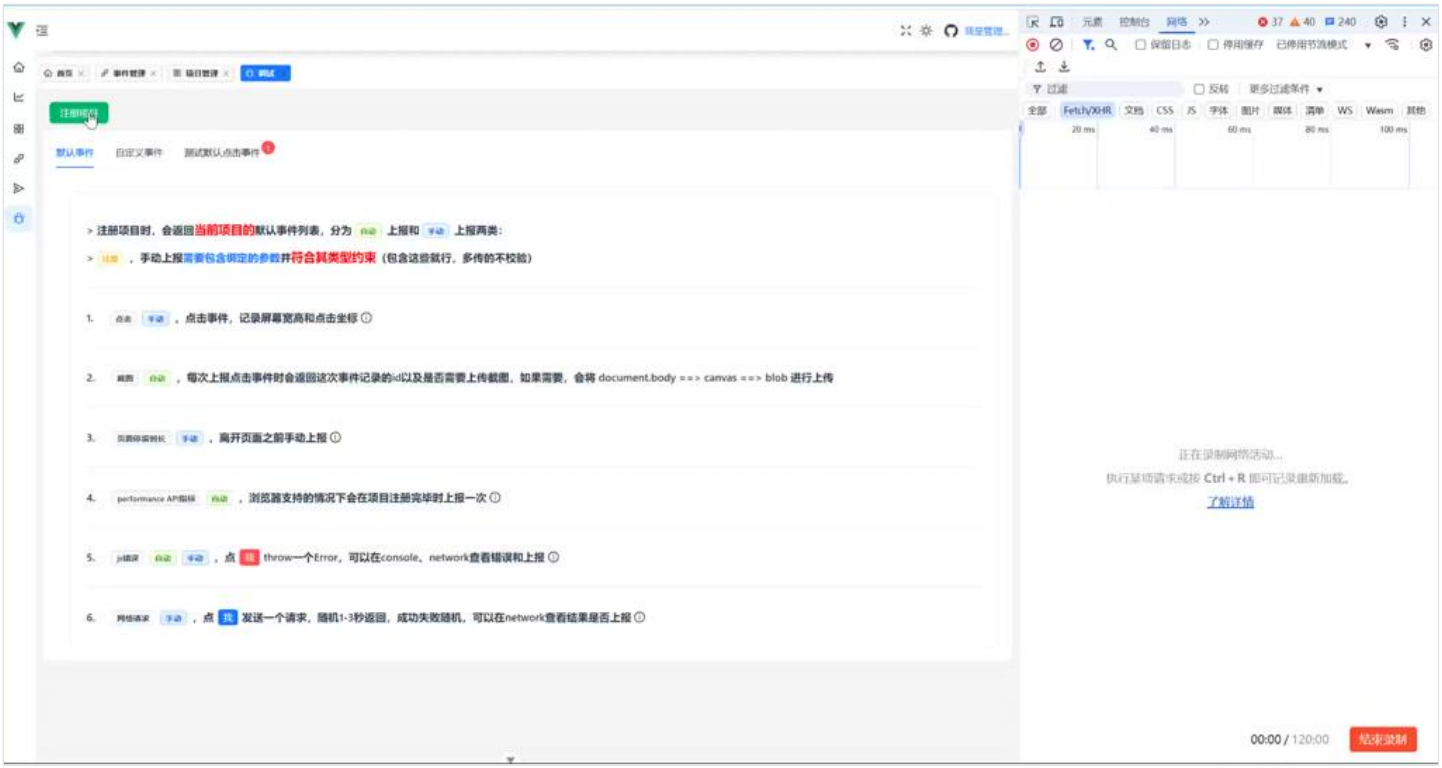
```
-------------------|---------|----------|---------|---------|-------------------
File               | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------|---------|----------|---------|---------|-------------------
All files          |      80 |    80.95 |      70 |   79.54 |
 lib               |     100 |      100 |     100 |     100 |
  constants.ts     |     100 |      100 |     100 |     100 |
 lib/impl          |   63.63 |       60 |      40 |   63.63 |
  http.ts          |   63.63 |       60 |      40 |   63.63 | 38,53-59
 lib/util          |      95 |     87.5 |     100 |   94.73 |
  event-register.ts |     100 |     87.5 |     100 |     100 | 19
  register.ts      |   93.75 |     87.5 |     100 |   93.33 | 16
-------------------|---------|----------|---------|---------|-------------------

Test Suites: 3 passed, 3 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        2.361 s
```

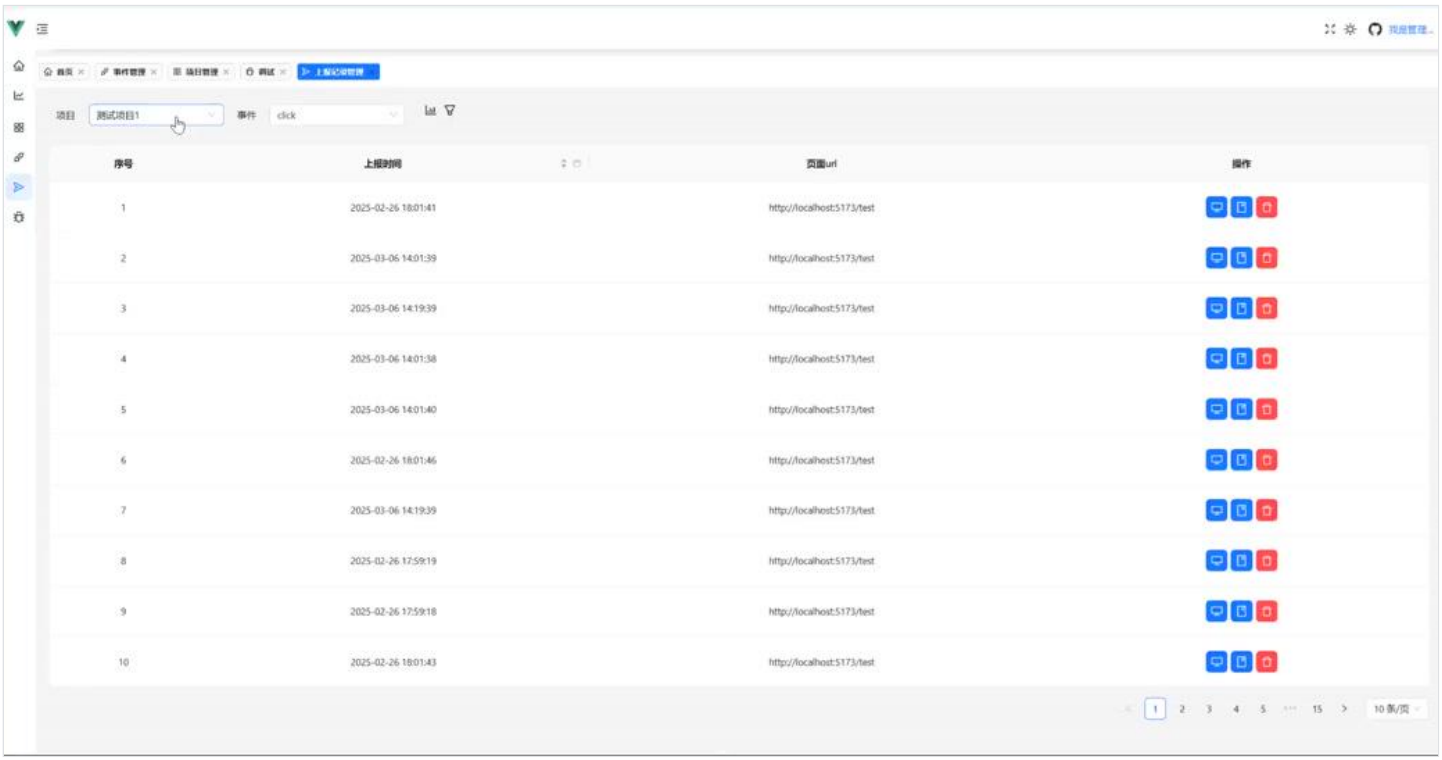
五、Demo 演示视频 （必填）

1. 项目、事件的增删改查和同步

2. 测试事件上报



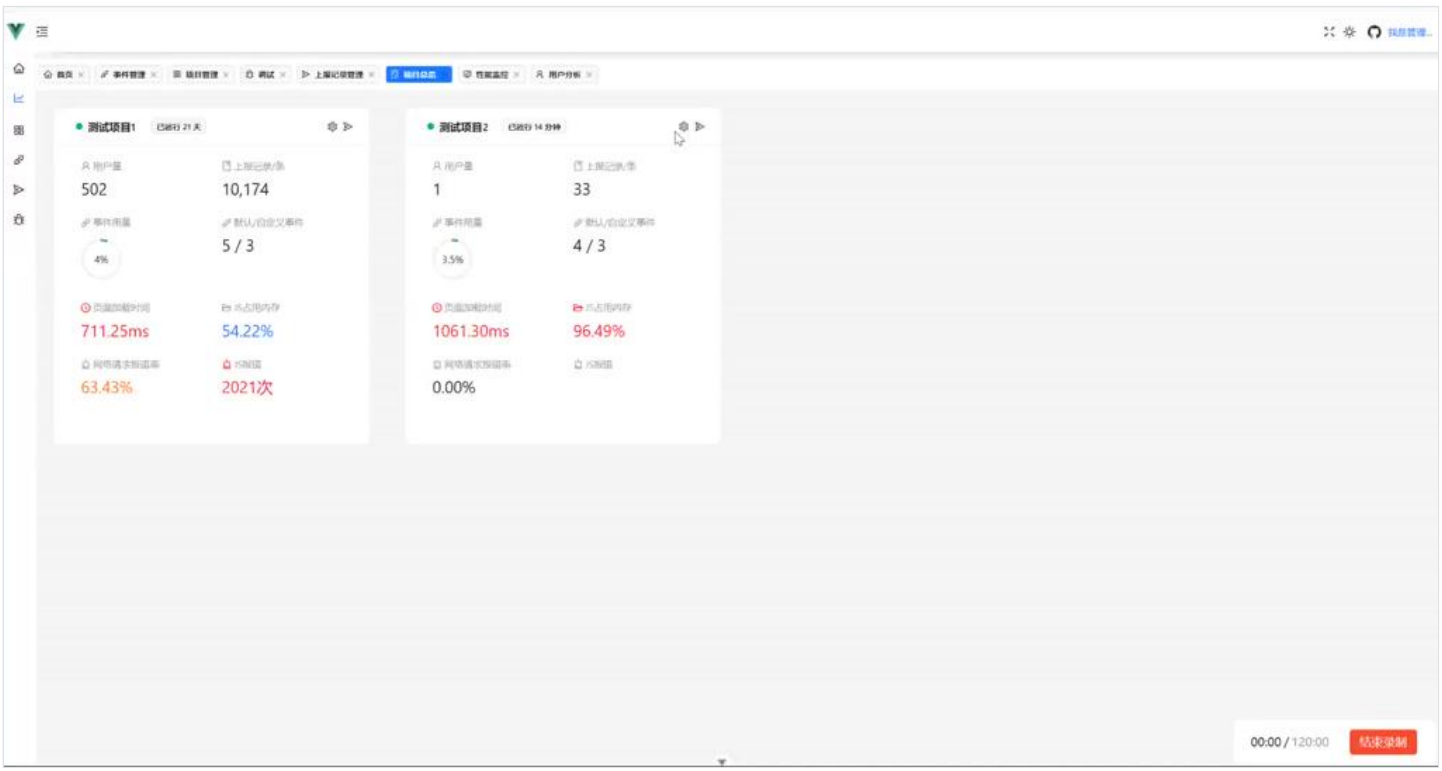
3. 上报记录管理——以刚才上报的事件为例



4. 数据可视化

- 模拟数据

- 刚创建的项目采集的数据



六、项目总结与反思

1. 目前仍存在的问题

> - 面对大量用户和高并发上报请求时可能的性能问题；
> - 没有实时对数据进行清洗归档，每次获取都是从数据库查询上报记录简单计算再返回；
> - 统计数据考虑不全面，还可以对错误记录做更详细的可视化；
> - 管理端还没写，现在只能从数据库改字段；

2. 已识别出的优化项

> - 对上报数据进行处理后进行储存，且定期更新，用户获取时不用再查数据库，减轻数据库压力；
> - 使用消息队列处理无需截图事件的上报记录，提高性能；
> - html2canvas在处理部分页面时可能会出现变形等情况，未来可以在这方面探究一下，或者让用户手动上报截图；

3. 项目过程中的反思与总结

> - 在启动一个项目之前，搭一个科学合理的框架、技术选型非常重要，不但能提高效率，也能减轻负担；
> - 在上面的框架下，逐步完成功能，比如从最简单的登录注册、增删改查开始，走通SDK上报流程，再考虑合并上报、限流、消息队列、缓存等优化手段；
> - 团队中不同人的代码风格、调包习惯需要积极协调，及时沟通在任何项目中都必不可少；

七、其他补充资料（选填）

主页截图：

字节跳动青训营项目

嗯有点像最简版的，至少颜色上👀👀