

MarsOJ 项目交付文档

组号	C10
成员	徐浩博 顾洋丞 胡昌宇 闵安娜
时间	2023.1.8

修订记录

日期	修订版本	修改描述	作者
2023/1/8	V1.0	MarsOJ 项目交付文档	C10组全体

目录

1. 交付产品	4
1.1. 产品简介	4
1.2. 产品信息	4
2. 产品目标	4
2.1. 功能目标	4
2.2. 性能目标	6
2.3. 可扩展性目标	6
2.4. 用户友好性目标	6
2.5. 安全性目标	6
3. 开发组织管理	7
3.1. 过程及配置管理	7
3.2. 人员分工	8
3.3. 开发环境	8
4. 系统设计	9
4.1. 整体概述	9
4.2. 前端交互	9
4.3. 后端模块	12
4.4. 数据库设计	13
4.5. 接口规范和接口测试文档	14
5. 重难点问题	16
5.1. 前端	16
5.2. 后端	16
6. 测试总结	19
6.1. 测试方法	19
6.2. 功能测试	19
6.3. 性能测试	21
6.4. 测试具体说明	23
7. 系统部署	25
7.1. 部署方法	25
7.2. 部署流程与规范	25

1. 交付产品

1.1. 产品简介

本项目是一个 Web 前后端项目，主要为 MarsOJ 机构的学生提供线上辅助训练，主要功能为实时答题对战和错题回顾，从而实现线下教育内容的巩固和提高；与此同时，教师也能够通过该平台跟踪学生的学习情况。

结合 MarsOJ 机构名称，为了普及传播度，同时为了机构增添删改后续功能，我们将产品定名为 MarsOJ，此名称也符合编程辅导机构的命名惯例。项目的图标 Logo 结合 MarsOJ 的名称含义，设计为一个火星图标，如下图所示：



图 1.1 MarsOJ 项目 Logo 图标

1.2. 产品信息

本产品部署的服务器 IP 为 <http://82.157.17.219/>，考虑到项目涉及四人对战，故项目组提供四个测试账号：

序号	用户名	密码	身份
1	Admin	Admin123	管理员
2	user1	User123	普通用户
3	user2	User123	普通用户
4	user3	User123	普通用户

2. 产品目标

2.1. 功能目标

本项目功能目标的概览图如下：



图 2.1 项目功能目标概览图

- 学生用户端

- ✓ 注册、登录

用户可以进行注册、登录，其中密码会进行合法性检验，身份验证成功后会跳转到首页。

- ✓ 资讯浏览

可以显示信奥资讯列表，列表中显示资讯概览，并以向下滚动列表的方式查看更多；点击进入资讯后可以显示资讯详情。

- ✓ 答题 PK

答题首页显示全站积分排行榜；并可进入 PK 四人答题对战。

答题对战的题目分为单项选择题和综合选择题，每题需要在规定时间内作答，超时或答错不得分，正确则根据答题速度加分；所有参赛者得分实时显示并排名。

答题结束后应显示用户排名，并显示个人在本场比赛中所有答过的题，可以将题目加入收藏夹。

- ✓ 错题回顾

可以对个人题目收藏夹进行增删改查，并对收藏夹内的题目进行移动、复制、删除等操作。

每个收藏夹以列表的形式显示题目缩略内容，点击可以查看题目详情和答案。

- ✓ 个人页面

显示并修改个人头像、签名、显示正确率、积分等个人信息，并可以向下滚动列表的方式查看个人参与的对战信息。

- 管理员端

- ✓ 资讯管理

管理员可以对资讯进行增、删、改、查。

- ✓ 题目管理

管理员可以对题目进行批量增、删、改、查，同时提供批量上传功能。

- ✓ 对战记录管理

管理员可以列表式查看对战记录，同时提供批量下载功能和单独下载某场对战记录。

2.2. 性能目标

1) 运行稳定的情况下：

1. 一般操作（如：注册、登陆、用户管理等操作）响应时间最大不超过2秒
2. 特殊操作（如：对手匹配成功，答题结算等操作）响应时间最大不超过 5 秒。

2) 系统前端服务：

1. 前端界面的按钮（前后端接口处）的响应不超过0.5秒。
2. 平均页面跳转响应时间在2秒内。

3) 系统后端：

1. 前台提交数据给后台的处理时间不宜超过1秒。
2. 进行数据库操作的平均时间不宜超过0.5秒。

2.3. 可扩展性目标

项目应当具有可扩展能力，这要求项目架构应当分层化，模块化设计，同时应当将功能组件化，为后期维护升级和扩展功能提供便利。

同时，项目还应该有良好的接口设计，接口设计应当尽量风格统一、职责单一，团队应维护有清晰的接口说明文档，以保证后续的维护开发得以进行。

2.4. 用户友好性目标

首先是用户 UI 设计应当布局合理、美观大方、字体合适，符合大众审美。同时用户交互应当尽量做到简洁不晦涩，符合一般人的操作习惯，也符合网页通常的业界习惯，保证用户操作的便利性。

其次是应当采用清晰明了的用户语言，尽量添加更多提示框、提示语，友好引导用户完成各种功能操作，不至于找不到各个功能的入口和打开方式。

最后，导航栏应当保持主题统一，且放置在明显的位置。文字、背景、图片也尽量保持风格和大小一致，保证页面的清晰性和逻辑性。

2.5. 安全性目标

用户数据的安全性意味着需要尽可能保护用户的信息安全，不被泄露，这就要求项目对

数据库安全性等具有基本保障，同时应该通过基本的安全性测试。

用户数据安全保障一方面在防止泄露用户隐私：应尽量避免后端、数据库等直接暴露在公网，同时应对用户密码进行加密保存，避免隐私泄露。

除此之外，应当保证用户在重要操作前给予必要的提示，如管理员用户删除题目等，这样可以尽最大可能挽回用户不必要的损失。

3. 开发组织管理

3.1. 过程及配置管理

该项目主要依托 CODING 平台，通过 git 管理前端 frontend、后端 backend 和文档 doc 三个仓库，实现多人开发和版本控制。同时项目组还为前端和后端编写 Jenkinsfile，借助 CODING 平台实现持续集成/部署(CI/CD)，在 git push 时触发进行代码格式检查，后端部分还会进行 API 接口测试。具体见下图：

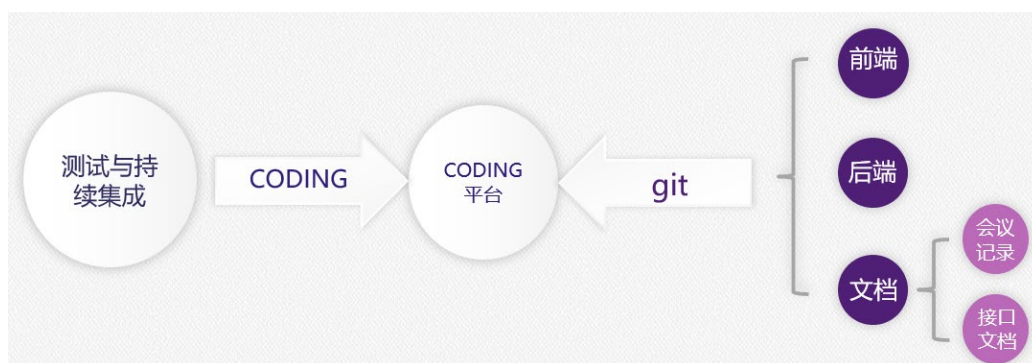


图 3.1 过程管理概览图

项目的开发过程文档主要在文档 doc 仓库中管理，主要包括以下内容：

- **会议记录：**会议记录包括项目组本学期共同线上/线下开会研讨的七次会议纪要；
- **接口文档：**接口文档包括文字版的前后端对接使用的 API 接口定义和说明；
- **原型设计与需求文档：**原型设计以墨刀为主要平台，项目组在墨刀平台上建立团队并共同管理原型设计，原型设计的文字版材料还写进了需求文档。

项目组还采用了 CODING 平台的事项管理提出 issue 发布任务、设定事项截止时间并更新协调进度，如下图所示：

ID	标题	优先级	状态	处理人	创建人	迭代	⚙
#25	🔗 个人信息和管理员接口pytest	中	已完成	25 闵安娜	17 徐浩博	个人信息页和	
#24	🔗 个人信息和管理员对应接口和数据库	中	已完成	17 徐浩博	17 徐浩博	个人信息页和	
#23	🔗 管理员页面	中	已完成	11 胡昌宇	17 徐浩博	个人信息页和	
#22	🔗 个人信息页和对战界面优化	中	已完成	S 顾洋丞	17 徐浩博	个人信息页和	
#20	🔗 收藏夹后端API pytest测试	中	已完成	25 闵安娜	17 徐浩博	收藏夹	
#19	🔗 收藏夹后端及数据库	中	已完成	17 徐浩博	17 徐浩博	收藏夹	
#18	🔗 收藏夹逻辑	中	已完成	S 顾洋丞	17 徐浩博	收藏夹	
#17	🔗 收藏夹页面	中	已完成	11 胡昌宇	17 徐浩博	收藏夹	
#15	🔗 对战接口测试	中	已完成	25 闵安娜	17 徐浩博	答题对战优化	
#14	🔗 对战逻辑优化	中	已完成	17 徐浩博	17 徐浩博	答题对战优化	
#13	🔗 对战前后页面优化	中	已完成	11 胡昌宇	17 徐浩博	答题对战优化	

图 3.2 事项发布示意图

3.2. 人员分工

- 前端

- ✓ 顾洋丞：登录注册页、结算页面、对战逻辑、个人信息页、整体页面优化
- ✓ 胡昌宇：主页、答题对战中页面、收藏夹页面、管理员页面

- 后端

- ✓ 徐浩博：账户、资讯、收藏夹的后端与数据库、对战交互的后端、过程文档管理
- ✓ 闵安娜：题目的后端和数据库、后端测试

3.3. 开发环境

- 后端

后端采用开发服务器的统一环境，服务器环境如下

OS: Ubuntu 20.04.5 LTS

开发工具: Python 3.8.10

数据库版本: MongoDB 3.6.8

- 前端

前端为项目组个人成员 PC，项目组尽可能统一了开发环境，确保在以下环境中运行正常：

OS: Windows 10、11
开发工具: node.js >= 16
浏览器版本: Microsoft Edge 108.0.1462、Chrome 108.0.5359、FireFox 108.0.2

4. 系统设计

4.1. 整体概述

本项目整体可分为前端、后端、数据库三个模块，它们之间的关系示意图如下：

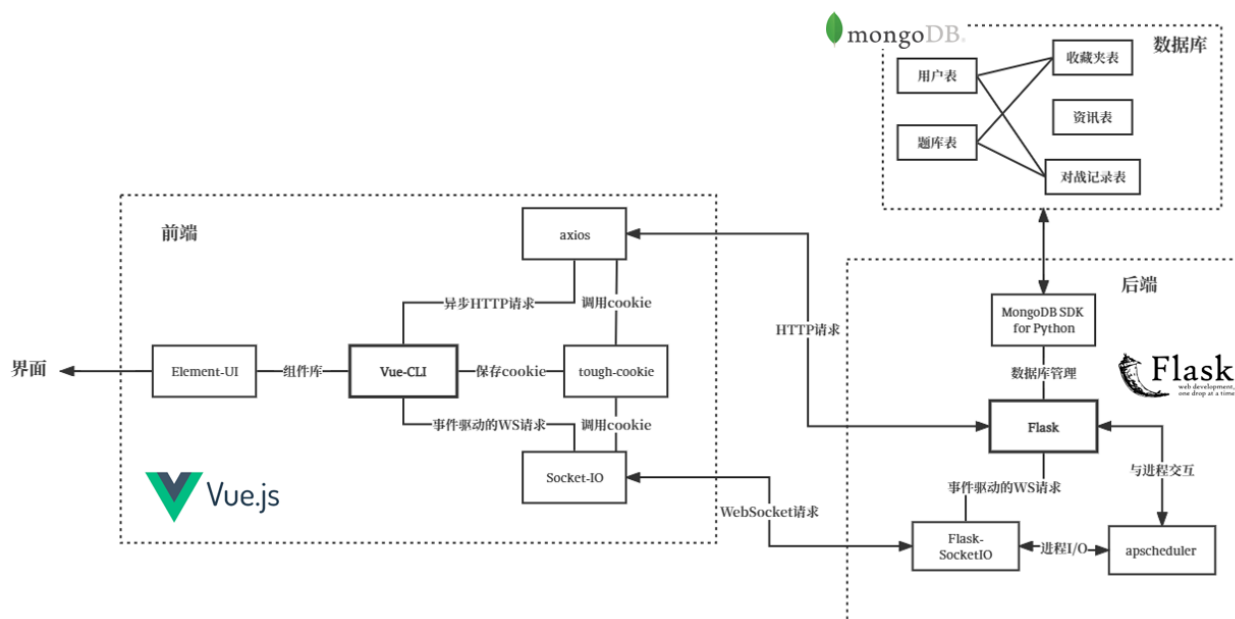


图 4.1 系统设计概览图

4.2. 前端交互

前端使用 Vue+ElementPlus 作为框架，实现了用户登录注册、资讯查看、答题对战、收藏夹、个人中心和管理员后台页面，通过与后端的交互满足用户需求。

具体而言，前端页面可以分为以下六个：

- ✓ 登录注册：验证密码合法性，身份验证成功后将会跳转到首页
- ✓ 首页资讯：首页是可以下拉的资讯栏，用户可以查看资讯的标题、来源，并可以点击查看详情；侧边栏有相关网站的链接
- ✓ 答题对战：可以查看全站的 PK 排行榜，可以查看榜上选手的信息。点击开始匹配按钮后可以参加四人对战
- ✓ 线上 PK：PK 时可以实时查看各选手排名；PK 结束会生成比赛总结，用户也可以查看错题，并自由选择将部分题目加入收藏夹
- ✓ 收藏夹：用户可以创建和删除收藏夹。收藏夹中的题目以表格形式呈现，用户可以

查看、删除题目，或者在收藏夹之间移动题目。

- ✓ 个人中心：显示自己的对战记录和对战统计信息。可以在设置界面修改密码，管理员则可以进入后台。
- ✓ 后台管理：仅管理员账号可见，用来管理资讯、题库、对战记录数据，以表格形式呈现
 - 资讯管理：显示全部资讯信息，支持批量删除资讯，或者对单一资讯进行增删改查
 - 题库管理：显示题库全部题目信息，支持批量删除与上传，或者对单一题目进行增删改查
 - 对战记录：显示全部对局的时间与参赛者名次，支持下载单一对战记录或全部对战记录

其中，**答题对战**是我们的重要功能，我们将以此为例进一步介绍对战的流程：

- ✓ 答题对战界面（URL: /battle）提供了开始对战的入口，点击页面中间的“开始匹配”按钮即可申请加入匹配。**答题对战为四人对战**，系统为四名用户匹配成功后，页面自动跳转到对战中界面。页面的下方展示了实时的积分排行榜，用户可以点击其中的各行来查看其它用户的数据。

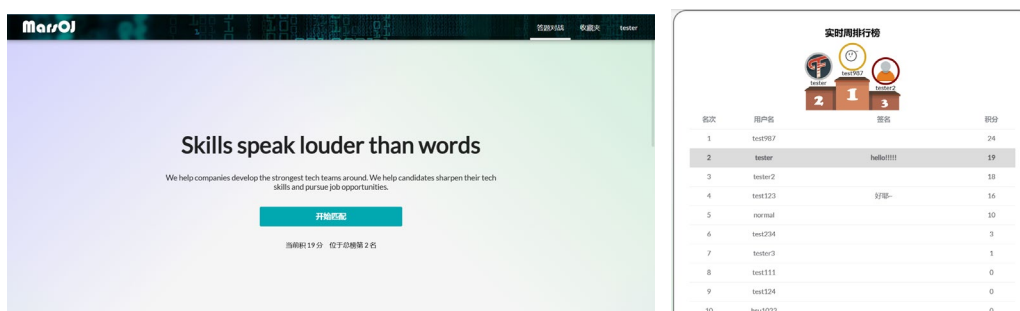


图 4.2 答题对战首页

- ✓ 比赛开始后，用户在对战中界面答题。
 - 用户需要在页面中央的答题区域内选择正确的选项（可能包括多道小题），在页面右上方的计时器超时之前，点击页面右下方的“提交”按钮提交答案；
 - 提交答案后，系统会核对答案的正误，并展示在答题区域内。此时用户需要耐心等待其他用户提交答案，或等待计时器超时；
 - 答题区域的左侧提供了所有的对局信息，包括所有用户的作答信息、实时积分、正确答案等；
 - 如果用户的排名出现变化，页面的正上方会弹出提示信息进行实时提醒；
 - 页面左侧显示了用户当前的排名。点击“当前第 x 名”可以弹出侧边栏，其中依次展示了实时排行榜和分数走势。点击排行榜中的各行可以查看用户信息。

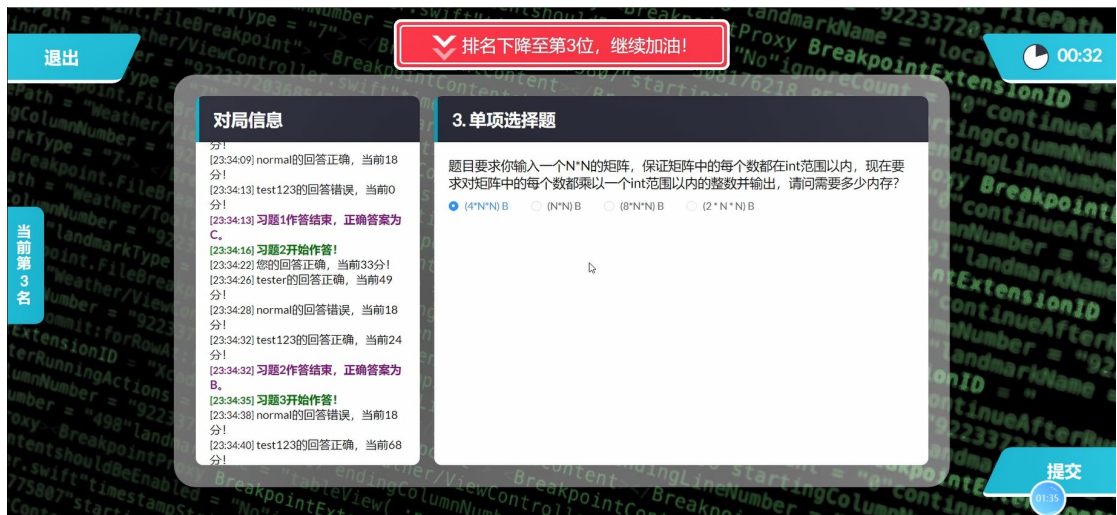


图 4.3 答题对战中页面



图 4.4 答题对战中边栏

- ✓ 所有题目作答完毕后, 系统自动跳转到对战结算界面, 展示了本次对局的所有数据, 包括比赛统计 (分数走势和总积分榜) 和题目详情。用户可以在题目详情中查看所有的习题, 并点击右上角的“收藏本题”下拉菜单, 选择收藏夹并收藏该题。



图 4.5 a) 答题结算页面——排行榜



图 4.5 b) 答题结算页面——错题回顾

4.3. 后端模块

后端以 Flask 为框架, 处理前端的 HTTP 请求和 WebSocket 请求, 并与数据库交互。

对于 HTTP 请求的项目、文件等内容, 后端通过访问数据库对数据进行增、删、改、查。

对于实时交互的功能, 如答题对战等, 后端通过 Socket.IO 进行数据通信, 并使用内存保存实时变动的数据, 同时也会访问数据库进行添加、查询等操作。

具体而言, 后端可以分为 **views**、**sockets** 和 **tests** 三个模块, 示意图如下:

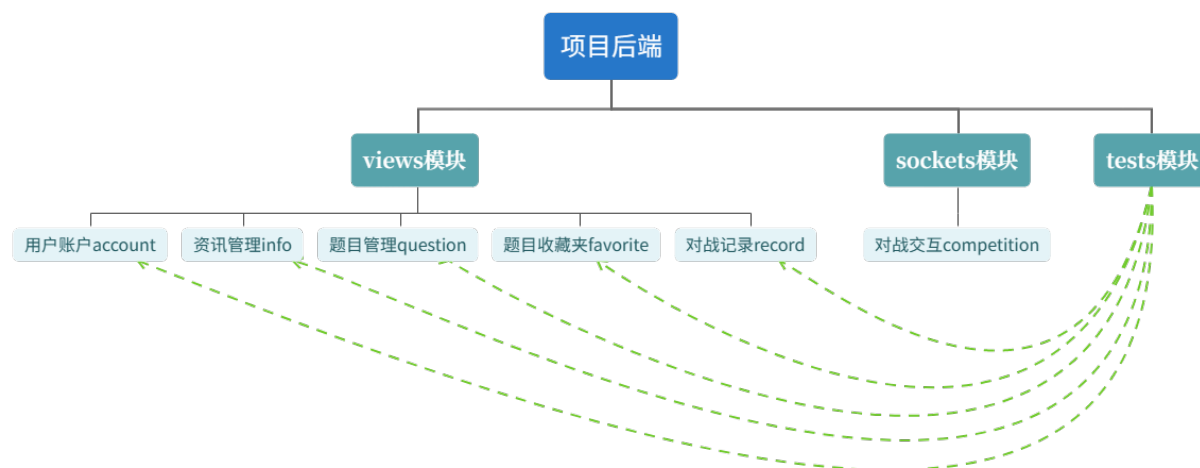


图 4.6 项目后端结构示意图

- views

后端共分 5 个 views，每个 view 以 Flask Blueprint 蓝图进行模块化，对项目的应用进行模块化拓展：

- ✓ 用户账户 account：用户注册、登录、个人信息查询和修改等，其中密码进行 sha256 加盐加密存储；
- ✓ 资讯管理 info：对于主页信息资讯的增、删、改、查；
- ✓ 题目管理 question：对于题目的增、删、改、查；
- ✓ 题目收藏夹 favorite：对于用户题目收藏夹及收藏夹内题目 id 的增、删、改、查；
- ✓ 对战记录 record：对于答题对战记录的增、删、改、查，包括用户排名、题目 id、每题选项和得分等。

- sockets

后端还包含一个 sockets 模块，用以实时对战交互：

- ✓ 对战交互 competition：接收前端的连接、退出、作答完毕等信号，进行匹配、发题、定时、判断正误等操作，并将结果发送给前端。

- tests

tests 模块用于测试，分为 6 个部分，分别对应以上模块。tests 模块采用 pytest 测试框架，对以上每个模块依次进行 API 测试。

4.4. 数据库设计

结合已有题库为 MongoDB 导出的 json 格式和甲方要求，项目使用非关系型数据库 MongoDB，共建立四张表（collection），分别如下：

- ✓ account 表：记录用户名、加密后密码、个人题目收藏夹、头像 base64 值、积分以及其他个人信息；
- ✓ info 表：记录信息资讯标题、正文、来源、日期等信息；

- ✓ question 表：记录题库中题目，包括题目类型、题干、各个小题选项等信息；
- ✓ record 表：记录答题对战记录，包括用户排名、题目 id、每题选项和得分等。

4.5. 接口规范和接口测试文档

本项目也采用 swagger api 进行接口文档的管理和部分接口测试，可直接进行测试。与本交付文档同目录下 swagger-api 接口文档静态示例.mhtml 为示例，可直接查看。

接口主要分为 5 类，均符合 restful api：

1. 用户账号部分
2. 收藏夹部分
3. 资讯部分
4. 题目题库部分
5. 查看记录部分

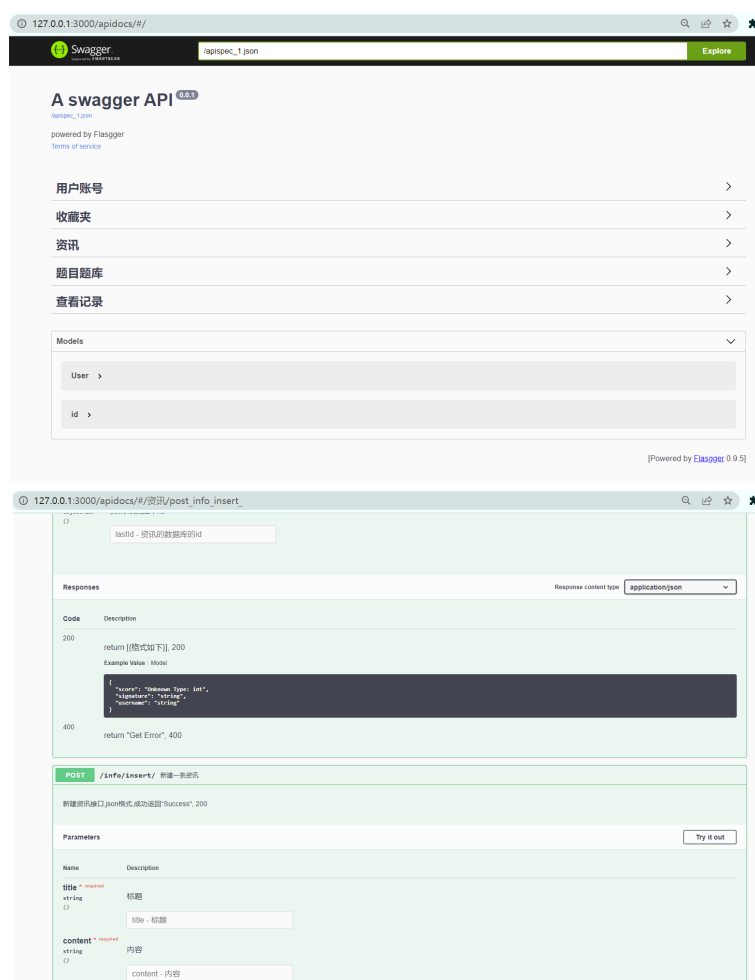


图 4.7 a) swagger 显示 API 界面

在本地打开接口文档的方法是：

1. 进入后端 backend 文件夹，运行 flask：
 - (1) cd backend

(2) flask run -p 3000 【或者其它端口】

2. 打开浏览器，进入 <http://127.0.0.1:3000/apidocs>

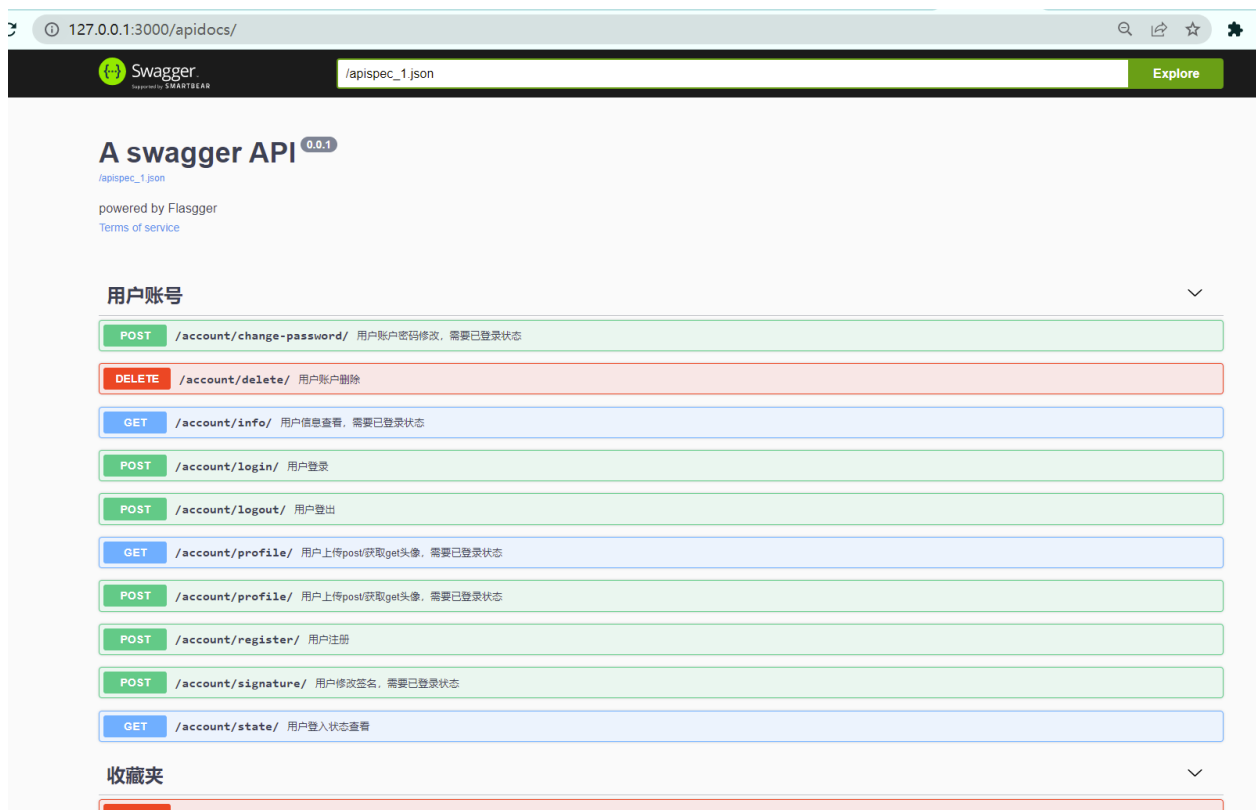


图 4.7 b) swagger 显示 API 界面

注：另外，本地使用 pip 安装 flasgger 后运行可能会遇到如下问题：

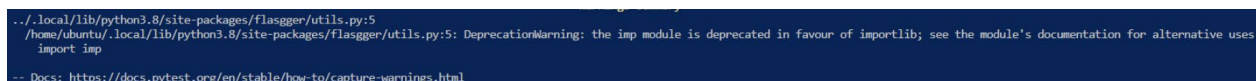


图 4.8 a) 报错界面及修改方法

遇到这个问题时，请进入报错相应文件和对对应行，将第三行 `import imp` 改成 `import importlib`。

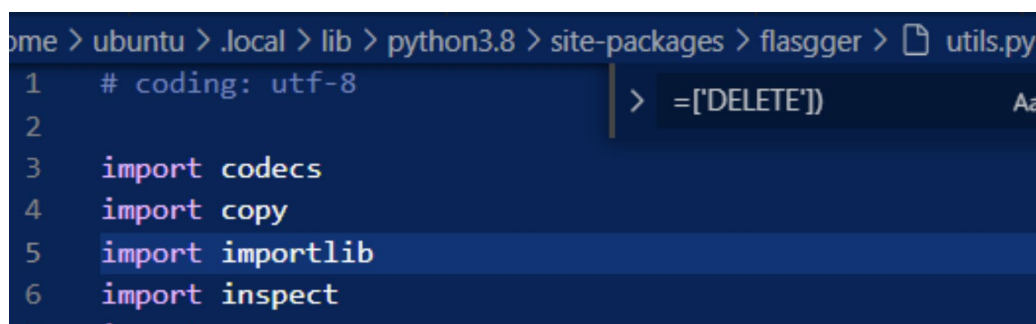


图 4.8 b) 报错界面及修改方法

5. 重难点问题

5.1. 前端

5.1.1. 设计美观简洁、风格统一、用户友好的界面

由于同学们都缺乏前端设计的经验，一开始设计出来的界面较为简陋，难以让用户产生使用欲望。因此我们参考了部分同类型网站（如 LibreOJ）的设计风格；此外我们还借鉴了 github、百词斩等应用的设计元素，并基于我们网站自己的答题 PK 系统进行融合，最终在实用性与美观性上达到统一。

5.1.2. 使用 Vuex 维护状态信息

网页中的组件不仅要保存当前页面的信息，还需要维护、访问一些全局信息，如用户登录状态、PK 对战状态等，因此需要在组件之外单独管理状态。由于本项目是单页应用，没有必要将状态信息保存到浏览器的本地存储中，最好采用封装性较好的全局变量。我们选择了 Vuex 状态管理模式，在状态仓库 store 中维护了 auth（登录状态）、competition（对战）的两个状态模块，提供响应式的状态管理。如果在组件中需要修改状态，只需要提交 Action 信息，Vuex 会调用对应的函数，由此实现状态修改。

5.1.3. cookies 保存登录信息

虽然网页已经采用 Vuex 维护了登录状态，但每次页面重载时状态会丢失。同时，一般的 axios 无法保存或处理后端提供的 session 信息。因此我们采用了 axios-cookiejar-support 和 tough-cookie 组件包装了 axios 对象，并设置对应的参数来支持 session 的存储。这样在打开页面时会先读取本地存储，在视图之间跳转时只需要维护 store 中的状态信息，这样能使得用户的使用更便捷。

5.1.4. 使用 WebSocket 进行赛时通信

在线竞技对于通信的实时性与并发能力有较高的要求，因此我们选择 WebSocket 作为比赛时与后端的通信方式。由于比赛过程中有许多信号需要处理，因此我们自己制定的 socket 通信规则较为复杂，前后端同学一起讨论了很久，并经过多次修改才得到了能够正确处理比赛流程的通信方案。此外在前端实现方面，由于 socket 连接横跨赛前、赛中和赛后三个页面，如何正确同步 socket 信息，保持连接也是重点问题。我们使用 BattleService 类实现 socket 通信，并使用 store 包装 BattleService 类做全局存储，解决了跨页面的 socket 通信问题。

5.2. 后端

5.2.1. 对战 PK 状态、线程众多

对战 PK 时后端涉及多用户、多状态、多线程问题。为此，我们开设三个内存池：waiting_pool、user_pool、competition_pool，分别代表等待匹配的用户池、正在比赛的用户状态池和比赛状态池，后端可发送的 signal 如下：

- prepare
 - 匹配成功，发送所有参赛者的个人信息
- problem
 - 发送一道题目，包含除答案、解析外的所有信息
- answer
 - 一人作答某题完毕后，将他的对战分数群发所有人进行更新
- next
 - 四人均作答结束，或定时器超时后，发送题目的正确结果，同时一个字段用于提示该题是否为最后一题
- result
 - 发送本场比赛的最终结果，包括排名、每个人的得分、每个人的每题作答正误情况等

后端接收前端的以下 signal 确定对应 action:

- 开始匹配 pair
 - 用户点击前端“开始匹配”按钮，前端向后端发送 pair 信号
 - 接收到选手开始匹配请求，检查选手是否在 user_pool 内，如不在则将选手加入 waiting_pool 等待匹配，同时修改 user_pool 状态为等待匹配
 - 检查 waiting_pool 内等待的选手数是否满 4 人，满 4 人则向所有选手发送开始比赛信号，进入阶段 3
 - 把 4 人从 waiting_pool 中删除，并在 competition_pool 内添加本局比赛所有选手信息、随机题目信息等，competition_pool 内的一个 data 就相当于某局比赛开出的一个线程数据；向所有 4 名选手发送 prepare 信号，告知选手彼此个人信息
- 开始比赛 start
 - 前端接收 prepare 信号，就绪后向后端发送 start 信号
 - 检查所有参赛选手是否均 start，若均准备好后向所有选手发送第一题，信号名称为 problem，内部包括第一题除答案的所有信息，并利用 apscheduler 开启一个定时线程，为该题设定截止提交时间，timeout 的回调函数为向所有用户发送 next
- 提交一道题结果 finish
 - 用户在前端提交一道题答案，以 finish 信号发送后端
 - 后端首先通过 competition_pool 内的数据判断提交题目是否过期（题号是否小于已发出题目），如未过期，则进行正误判断，并将判断结果和当前用户总分

告知所有用户，信号名为 `answer`

- 更新 `competition_pool` 内的 `data`，若四人均作答结束，则直接群发 `next` 信号，并取消定时器；设定新定时器，等待三秒以供前端显示题目正确结果，回调函数为群发下一题 `problem` 信号
- 确认等待比赛最终结果 `result`
 - 后端发送题目 `problem` 信号时告知前端上一道题为最后一道题后，前端结束页面跳转后发送 `result` 信号
 - 后端接收 `result` 信号后将全盘比赛的所有信息发送 `result` 信号告知所有用户
 - 后端将 `competition_pool` 内的本局对战结果通过调用数据库接口存储下来，并在 `waiting_pool`、`user_pool`、`competition_pool` 内对相关数据进行查找、修改和删除
- 取消匹配/退出游戏 `cancel`
 - 在 `waiting_pool`、`user_pool`、`competition_pool` 内对相关数据进行查找、修改和删除

除此之外，后端还设定了严密线程锁避免冲突访问数据。首先是全局的线程锁，对于修改 `waiting_pool` 和 `user_pool` 数据的临界代码区域，用全局线程锁进行保护。其次是每场比赛均有线程锁，即 `competition_pool` 的每个 `data` 内部均有自己的线程锁，这是避免并发状态下排队等待全局锁释放的情况出现。在线程锁和详细定制的前后端交互流程之下，避免了可能的冲突和错误。

下图按照类似有限状态机的形式简单描述了后端的各个状态和动作：

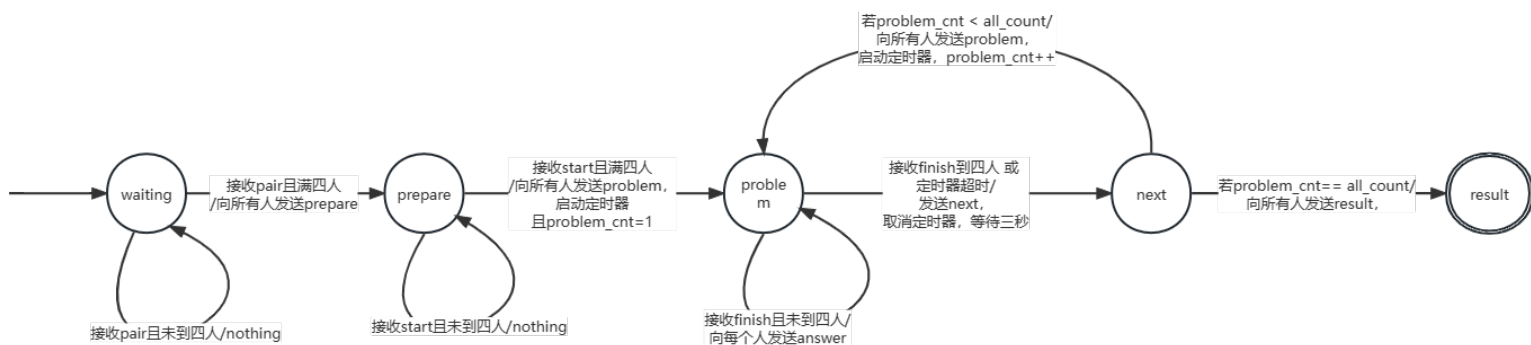


图 5.1 答题对战流程的类 FSM 图

5.2.2. 为前端提供的接口众多，难以模块化

后端涉及的接口有近 50 个之多，若不模块化管理，很难达成可扩展性目标，代码可读性和可复用性也将大大下降。为此，项目组将接口分为各个部分，并采用 `Blueprint` 将每一部分的接口模块化；同时为每一模块设定不同的 `URL` 前缀，每一模块拥有不同的 `URL` 前

缀，也为前端调用接口提供了便利。

6. 测试总结

6.1. 测试方法

1. 对于后端接口：使用 `yapi` 进行手动和自动测试，`CI/CD`, `pytest` 进行测试。将测例进行等价类划分，使用了白盒测试的方法。此外，所有 `restful api` 可在接口测试文档（见 4.5 节）网页端进行直接测试。

2. 对于前端：使用模拟器和实机测试软件在不同设备、系统、缩放比、分辨率上的表现形式，确保鲁棒性强。进行互操作性测试。

3. 对复杂的逻辑行为，使用场景法，手动将前后端一起进行测试：进行功能测试、适合性测试、准确性测试、内容测试、表单测试。在平时会在每次 `debug` 后及时进行回归测试，确保不会引入新 `bug`。（从原测试集中选择子集）

4. 使用 `W3C Link Checker` 进行连接测试，删去死链。

5. 引入对程序不熟悉的用户进行应用体验，进行探索性测试，杜绝意想不到的 `bug`，测试程序的易用性。

6. 电脑端进行兼容性测试，包括多种浏览器，多种版本的系统（`IOS11`、`HarmonyOS 2`，`Android 11`，`Windows 10`，`Mac`）等。确保功能、显示在多个平台均能正常运行。

7. 使用 `Jmeter` 进行性能测试。

6.2. 功能测试

6.2.1. 单元测试

本项目采用自动化单元测试。通过 `CI/CD`，在前、后端仓库每次 `push` 到 `master` 分支时进行自动化测试，除了代码规范性检查（前端 `ESLint`、后端 `flake8`）外，还对项目后端进行自动化单元测试。

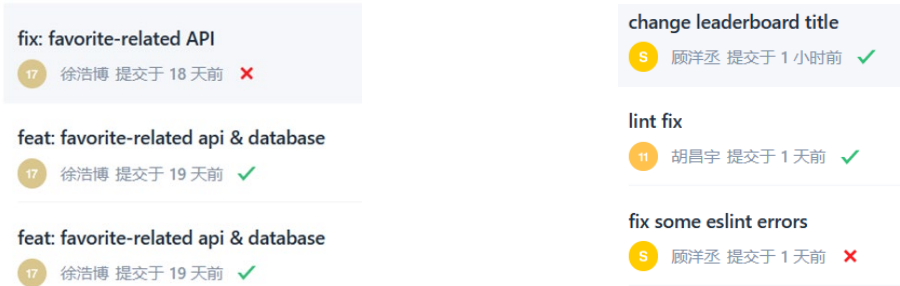
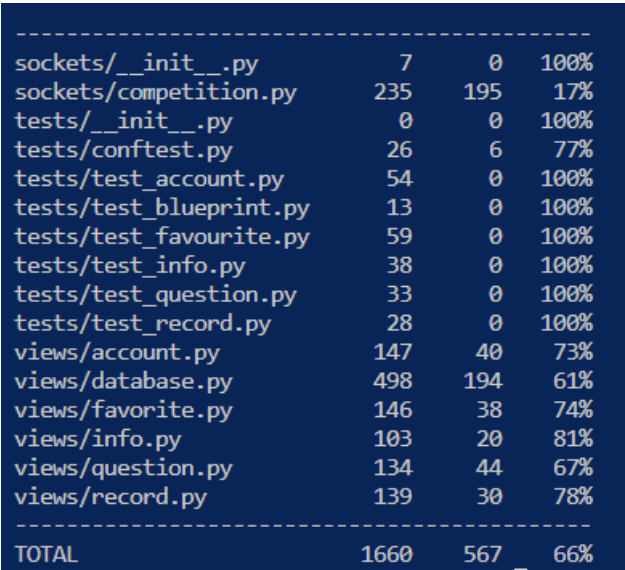


图 6.1 CI/CD 示意图

单元测试采用 pytest 框架，主要测试后端提供给前端的接口 API 的正确性，对于每个 API 验证返回内容是否符合预期，以此检查 API 是否有内部缺陷。具体测试见 6.4。pytest 在 backend/tests/*_py 中。



sockets/__init__.py	7	0	100%
sockets/competition.py	235	195	17%
tests/__init__.py	0	0	100%
tests/conftest.py	26	6	77%
tests/test_account.py	54	0	100%
tests/test_blueprint.py	13	0	100%
tests/test_favourite.py	59	0	100%
tests/test_info.py	38	0	100%
tests/test_question.py	33	0	100%
tests/test_record.py	28	0	100%
views/account.py	147	40	73%
views/database.py	498	194	61%
views/favorite.py	146	38	74%
views/info.py	103	20	81%
views/question.py	134	44	67%
views/record.py	139	30	78%
TOTAL	1660	567	66%

图 6.2 后端代码覆盖率

可以看到除了 websocket 接口，HTTP 接口的代码覆盖率基本达到了 70%左右，单元测试符合交付标准。

6.2.2. 前后端场景法测试

对以下功能进行了场景法测试：【可见交付视频】

1. 登录：
 - (1) 用户登录（固定）
 - (2) 注册：用户注册（固定）
2. 首页：
 - (1) 首页：展示资讯、网站资源（固定）
 - (2) 向下滚动以加载更多资讯
 - (3) 点击卡片可查看资讯详情
 - (4) 可跳转到其他 IO 网站
3. 个人中心：
 - (1) 个人中心：查看个人对战记录、统计数据、编辑个人资料、修改密码（固定）
 - (2) 查看对战记录
 - (3) 在设置界面修改密码
4. 管理员：
 - (1) 后台管理：仅管理员账号可见，用来管理咨询、题库、对战记录数据（固定）
 - (2) 资讯管理页面：单条咨询的查看、修改、删除
 - (3) 资讯管理页面：咨询的批量删除

- (4) 资讯管理页面：添加新资讯
- (5) 题库管理页面：单一题目的查看、修改、删除
- (6) 题库管理页面：题目批量删除与上传
- (7) 题库管理页面：自定义添加新题目
- (8) 对战记录页面：单一记录下载
- (9) 对战记录页面：全部记录下载

5. 对战：

- (1) 答题对战：4 人同时在线竞技，答题越快越准的人得分越高
- (2) 对战实时排行榜：可查看优秀选手的个人信息
- (3) 点击开始匹配，寻找对手
- (4) 匹配成功，进入答题界面
- (5) 规定时间内答题，选手答题情况实时更新
- (6) 超时未作答，视为错答，进入下一题
- (7) 对战结束：选手答题情况分析、排行榜
- (8) 题目回顾，可以自由选择将对战中的题目加入收藏夹

6. 收藏夹：

- (1) 收藏夹：收藏、回顾错题
- (2) 创建新收藏夹
- (3) 查看、移动、删除题目
- (4) 删除收藏夹

6.3. 性能测试

6.3.1. 性能测试概述

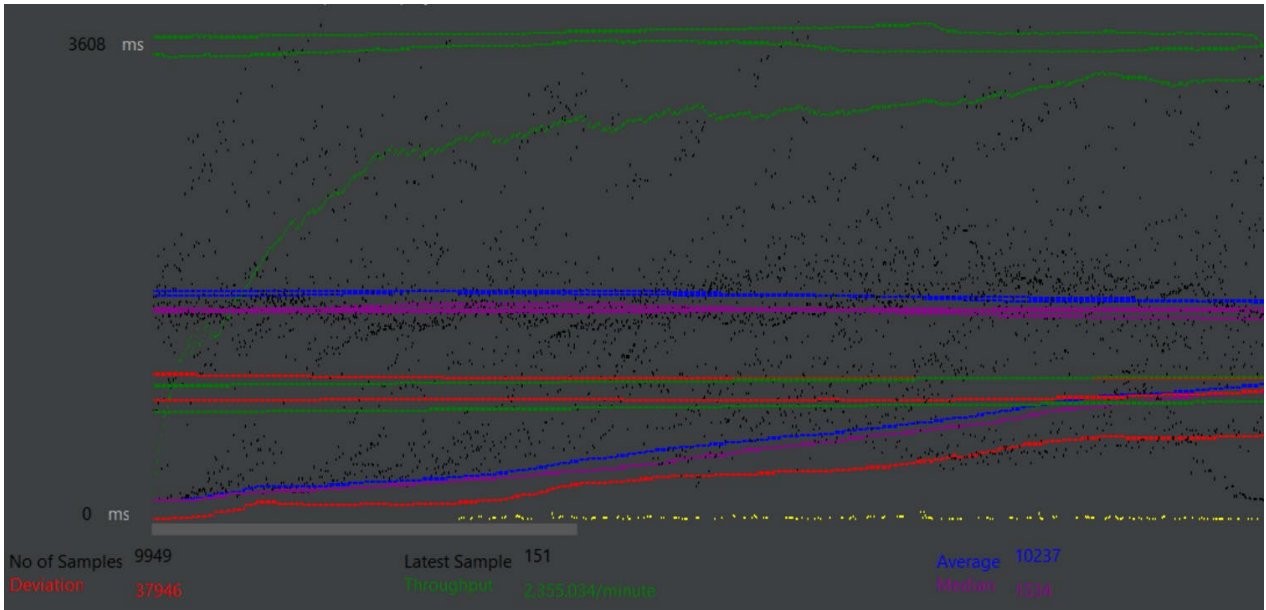
项目组在单元测试基础上进行性能测试。测试接口选择获取全部活动接口。使用 JMeter 进行负载和压力测试。

6.3.2. 性能调优过程

本项目经历若干性能调优过程，下面选取/question/count/接口为例进行简要介绍。

该接口是统计数据库内所有题目的总数。初始对/question/count/接口进行性能测试时，结果如下：

并发量	平均返回时间/ms	50%返回时间/ms	90%返回时间/ms	95%返回时间/ms	错误率
200	1498	1452	2167	2625	5.60%
300	2149	2117	3163	4141	3.38%
500	10237	1534	3015	155886	8.97%



可以看到当并发量达到 500 时，错误率达到将近 10%，这样的性能会严重影响整个项目的性能，为此，我们需要进行性能调优。

我们初步猜测性能瓶颈可能在于数据库部分接口的实现。我们采用的数据库为 MongoDB，是一种非关系型数据库。由于对该数据库并不熟悉，且为了开发的便捷，后端最初实现此 API 时，采用的是数据库提供的条件搜索 API，删除所有条件后，进行无条件搜索，最后利用 Python 的内置 len 函数对搜索成功后的返回结果数量进行统计。

获知该接口存在性能问题后，项目组仔细阅读了 MongoDB 的文档，发现可以使用 count_all() API 进行数量统计，返回数值，而并不必须返回所有查询到的结果。因此，我们增加了数据库提供的接口并修改了后端代码，最后测试的结果如下：

并发量	平均返回时间/ms	50%返回时间/ms	90%返回时间/ms	95%返回时间/ms	错误率
200	1210	90	643	1534	0.37%
300	1849	110	985	1703	0.87%
400	2540	131	1034	1855	0.98%
500	2871	150	1349	2134	2.10%
1000	5476	191	1520	3155	2.90%

可以看到性能已经获得了大幅度改进，该接口已经基本符合性能要求。

6.3.3. 性能测试结论

经测试，项目的 10 个重要接口在并发量达到 500 时，错误率都在 2.5% 以下，平均返回时间在 2.5s 左右。考虑到在实际甲方业务的情境中，并发量很难达到 500，最高并发量在 100-200 左右，因此项目的性能符合交付标准。

6.4. 测试具体说明

测试用例设计

对于测试用例，基本上采用了等价类划分的形式进行测试用例的测试。

以下为按接口设计的测试用例：

对于所有变量，进行了输入过长以及输入类型不合法的测试，为简洁，没有在等价类划分中单独列出。此外，所有 `restful api` 可在接口测试文档（见 4.5 节）网页端进行直接测试。

用户模块

部分关于错误信息的测试例如下所示，详细测试见 `backend/tests/test_*.py`，可在 `swagger api` 中进行测试。

登录接口

测试的变量	等价类划分
Username,password	一般未注册用户
	一般注册过的用户

修改用户信息接口

测试的变量	等价类划分
用户密码	不存在的用户
	存在的用户
用户头像	非法输入

资讯接口

添加/更新/删除/查看资讯接口

测试的变量	等价类划分
资讯 id	是否存在
资讯来源	是否合法
资讯时间	合法

收藏夹接口

查看/删除/移动收藏夹接口

测试的变量	等价类划分
用户 id	不存在的用户
	存在的用户
	是否登录

默认收藏夹	是否默认
目标收藏夹与源收藏夹	是否存在
	是否相同

下载记录接口

测试的变量	等价类划分
用户 id	不存在的用户
	存在的用户
是否管理员	权限
对战 id	是否存在
	是否合法

题目模块

添加/删除接口

测试的变量	等价类划分
子题目	非法输入
题目难度	符合范围
题目分类	符合范围

文件接口

测试的变量	等价类划分
文件中题目信息	是否存在
	是否合法

答题竞赛

测试的变量	等价类划分
答题是否正确	正确
	错误
得分曲线	更新
答题超时	超时
	没有超时
竞赛者中途退出	退出
	没有退出

对于所有接口，均进行了可靠性测试（即传递不合法的变量、不需要的变量等），均会

返回错误信息，不会崩溃。

经过多次完备的测试，认为本程序目前无重大缺陷，核心功能在多种操作系统上运行正常。认为本软件可以正常使用。而功能性、可靠性、易用性、兼容性、性能均符合预期要求，且无较大的缺陷，可以认为本产品有较好的内部和外部质量。

7. 系统部署

7.1. 部署方法

该项目部署采用 Docker 的方式，前端、后端、数据库分别运行在一个 Docker 容器里，并采用 Docker-Compose 编排所有容器。

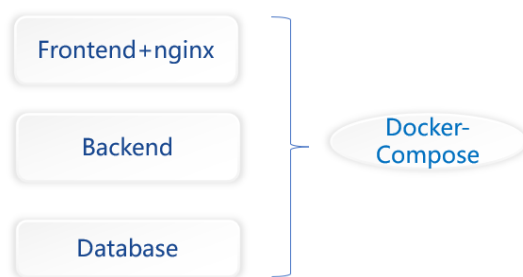


图 7.1 部署方法示意图

部署时首先利用 git 拉取前端和后端仓库，并利用 Dockerfile 创建前后端和数据库三个 Docker 镜像，最后用 Docker-Compose 编排并以网桥相连。

7.2. 部署流程与规范

- 安装 Docker 与 Docker Compose

已经有 WSL2 的 Windows 可以直接安装 Docker Desktop : [Install Docker Desktop on Windows | Docker Documentation](#); 其它平台可以参考: [Install Docker Engine | Docker Documentation](#);

- 拉取部署仓库 [git@e.coding.net:marsoj/marsoj/deploy.git](https://e.coding.net/marsoj/marsoj/deploy.git)，修改根目录下的.env 文件，将 VITE_APP_BASE_URL 改为部署的服务器的 IP;
- 运行脚本 deploy.sh 或手动拉取子模块仓库 `git submodule update --remote` 并运行 `docker compose up -d`，服务将运行在 80 端口。