Rust—OJ大作业报告

Part 1.代码结构功能

大作业代码主体部分位于main.rs文件中，其中包含少数结构体（类）的定义以及后端主体部分函数定义；其余模块包含config.rs，job.rs，user.rs，contest.rs，分别定义不同模块之间的重要结构体类。

1. **config.rs**

定义配置文件（config）中json文件格式的结构体，包含Config及其所需的Server，Problem，Language类，以及Problem类所需的ProblemType枚举类，Case，Misc类。

1. **job.rs**

定义post和get jobs时所必须json返回格式的结构体，PostJob为提交评测任务的结构体，Job为返回相应的结构体，内含JobCase类，描述评测状态的State枚举类，任务结果JobResult枚举类，测试点结果CaseResult枚举类。JobQuery类用于在get\_job时自动获取url参数并分析。

1. **user.rs**

定义User类，作为描述用户的基准类。

1. **contest.rs**

ContestRules用于在获取排行榜时获取url参数，由枚举类ScoringRule和TieBreaker构成。UserInContest为返回的排行榜内呈现的User信息，UserRanking为某User的详细信息，用于根据不同规则进行排序。Contest和ContestInfo为新建比赛信息及其包含剩余提交次数的结构体类。

1. **main.rs**

定义Error类抛出异常，Args类读取命令行参数，剩余为后端各API所需函数，注释及宏均有明确介绍。以下介绍主要函数的结构：

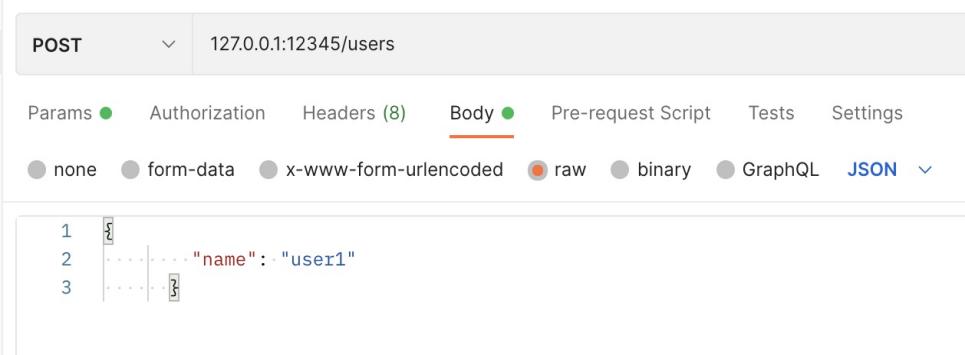
post\_jobs：合理化检测，返回值Job类初始化，编译，打包测试判断，运行，TLE判断，RE判断，根据ProblemType分多种情况进行测试（包括提高评测方式Special Judge和Dynamic Ranking），返回相应结果

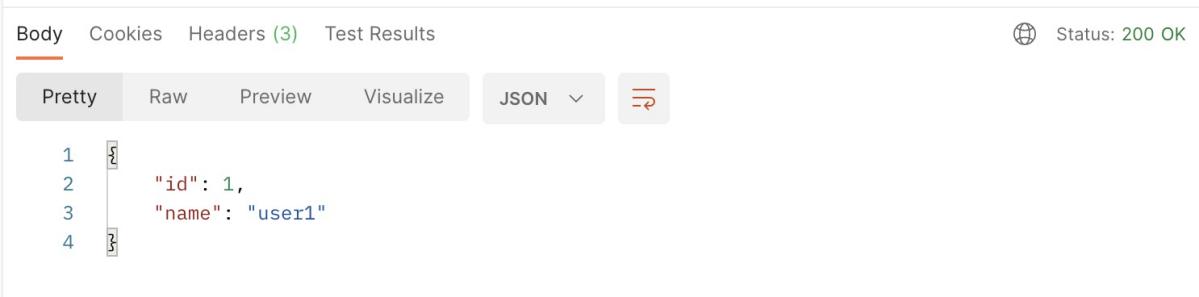
get\_ranklist：全局变量提取评测任务列表，根据Dynamic\_ranking标志计算分数，初始化排行榜，根据Jobs和ScoringRule向排行榜内添加得分以及时间信息，根据TieBreaker进行排序，返回响应结果

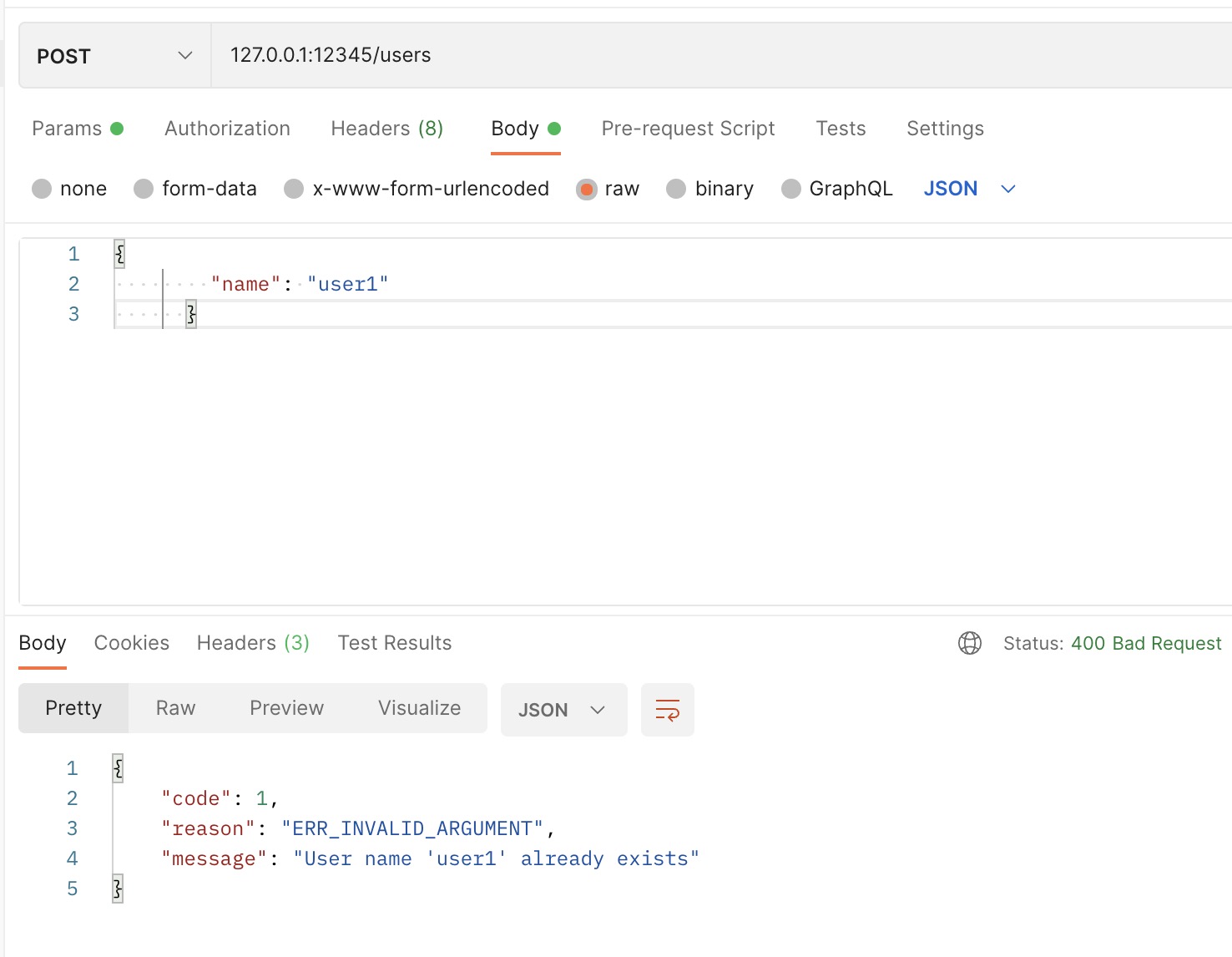
其余函数结构较为简单或有较多代码复用，故不再赘述结构。

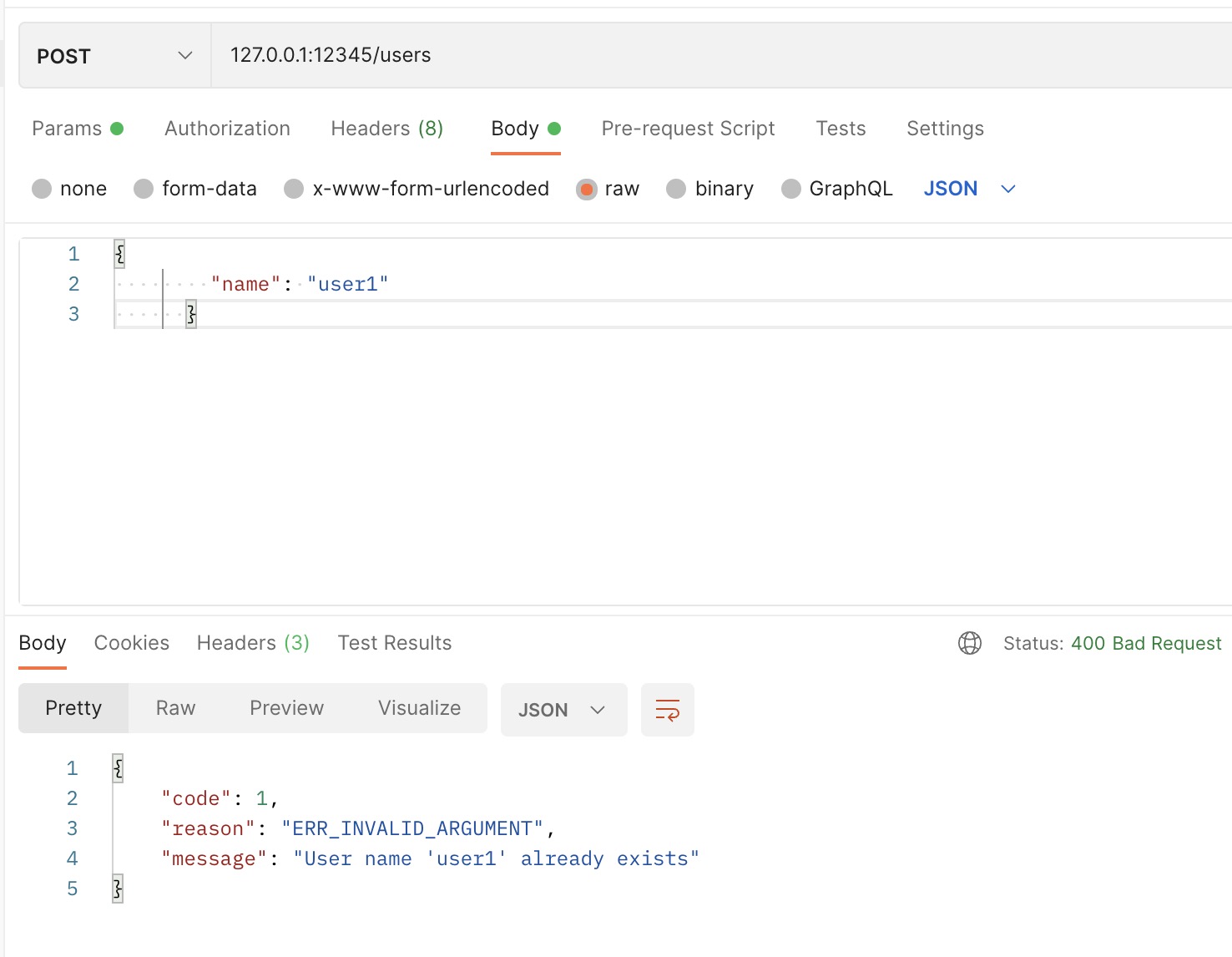
Part 2.主要功能截图

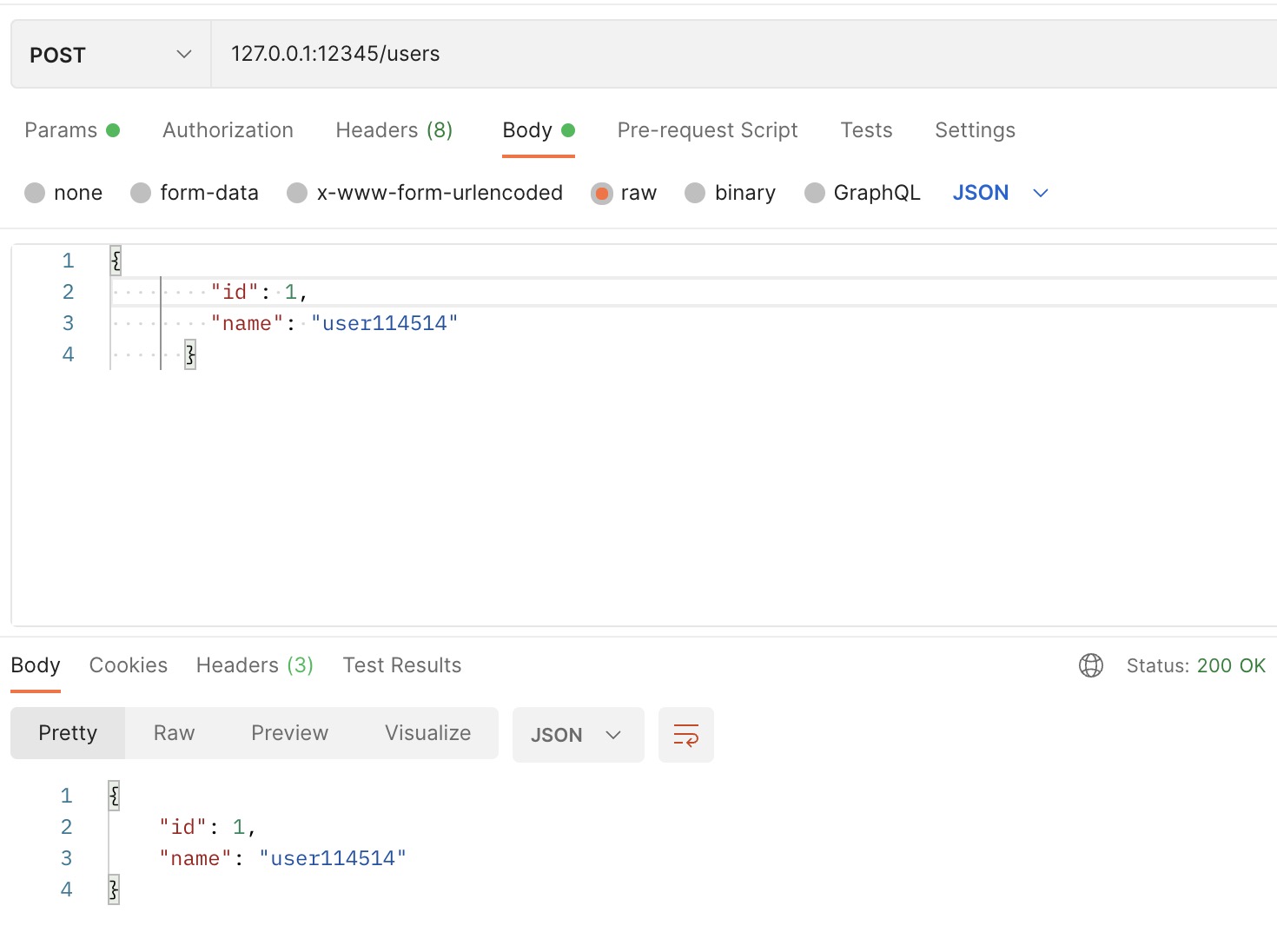
1. **基础（以下命令按顺序依次串行执行）**
2. users

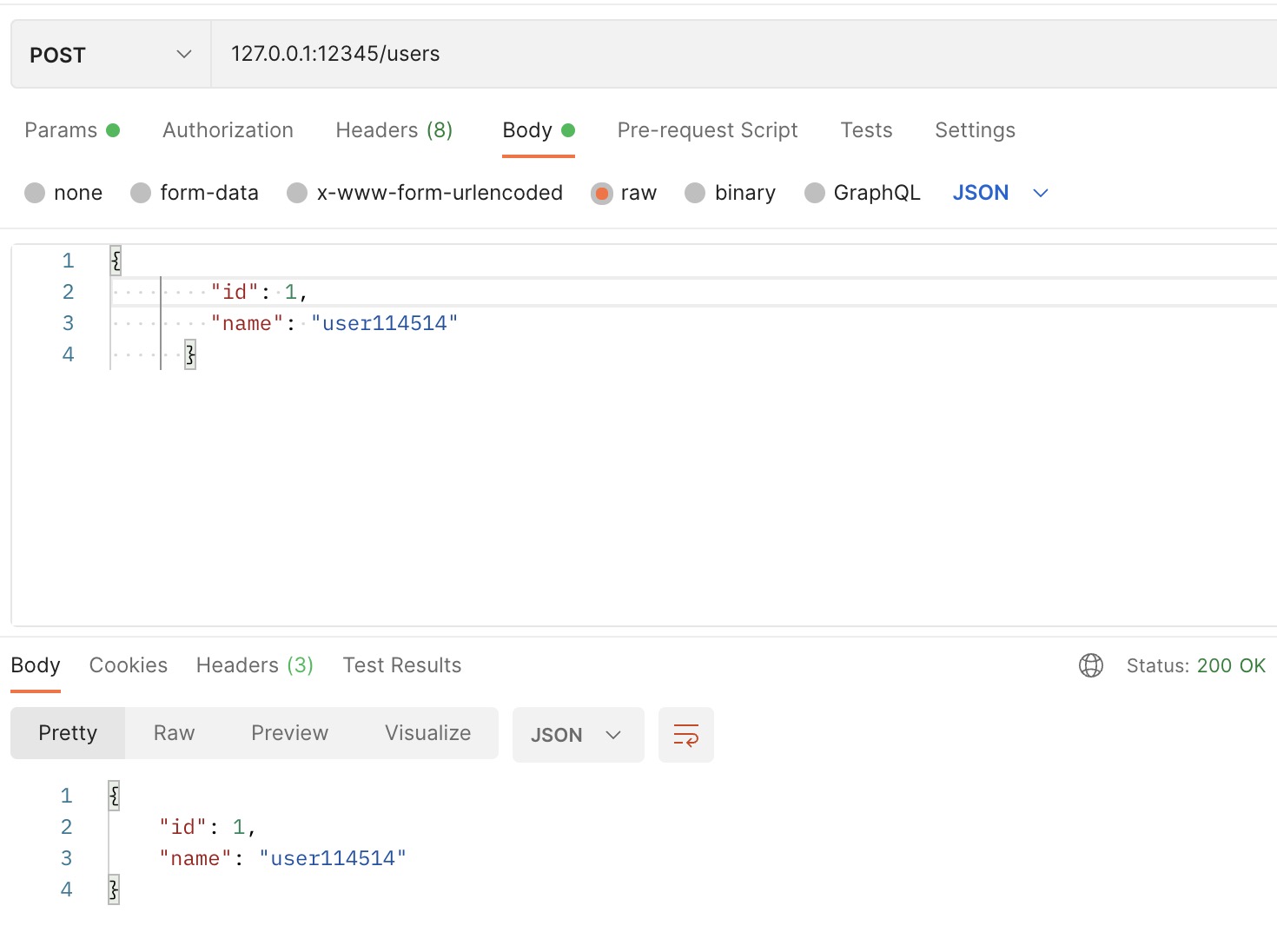


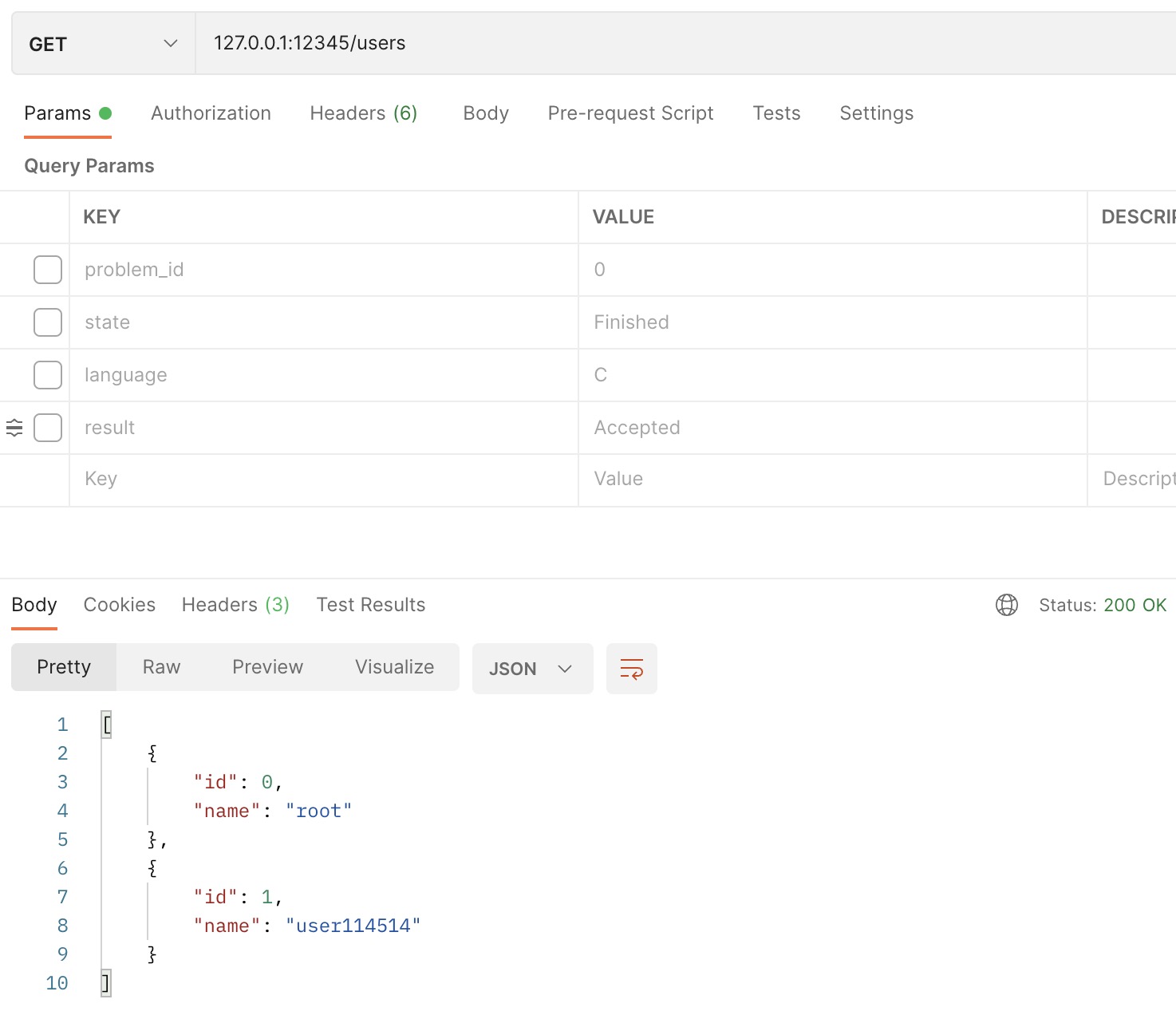


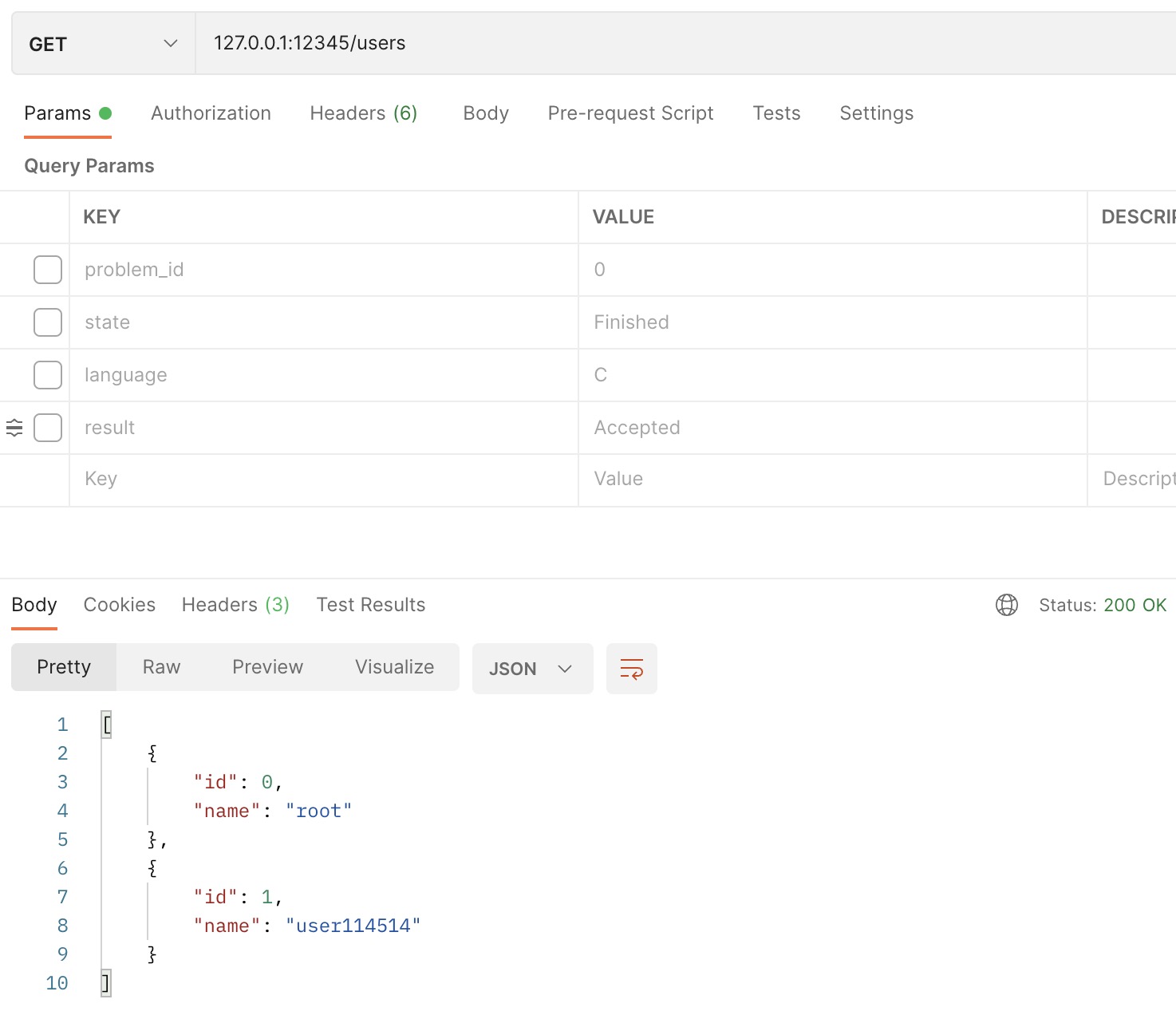




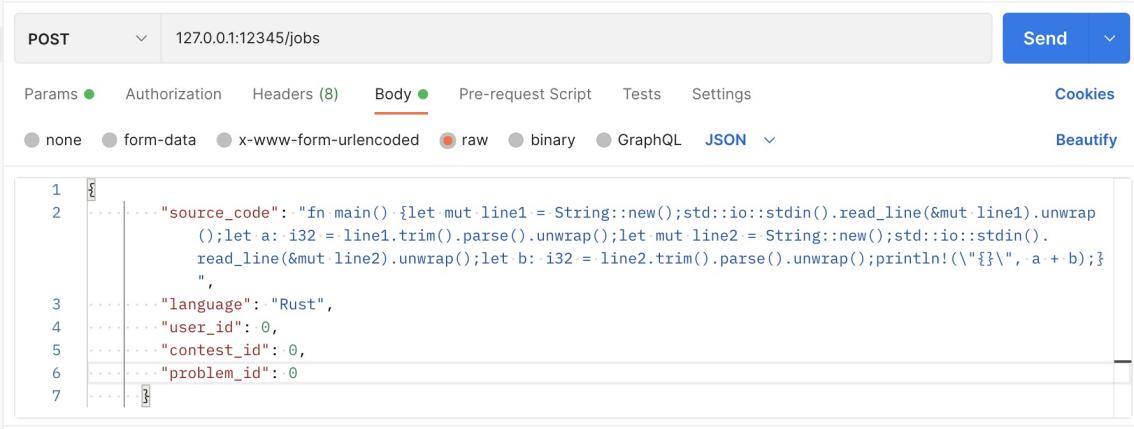








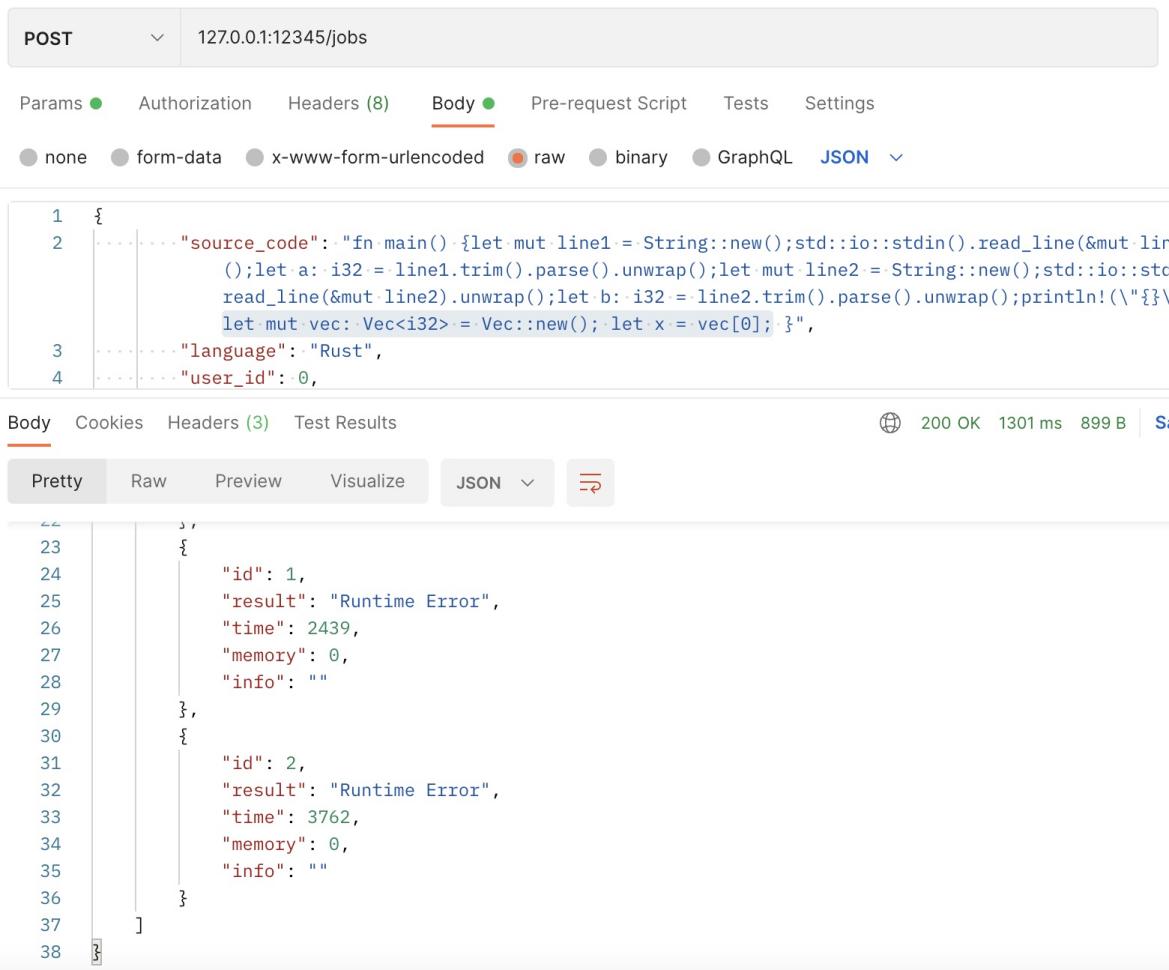
1. jobs（均使用oj/config.json作为配置文件）

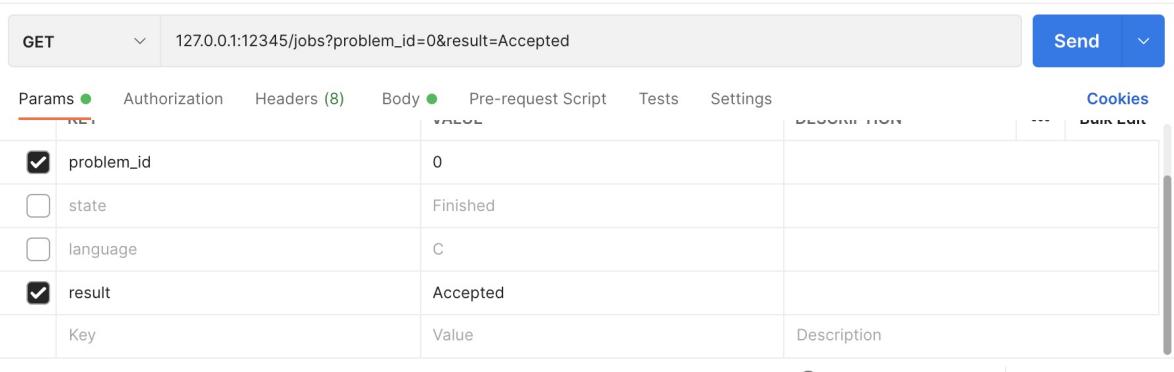


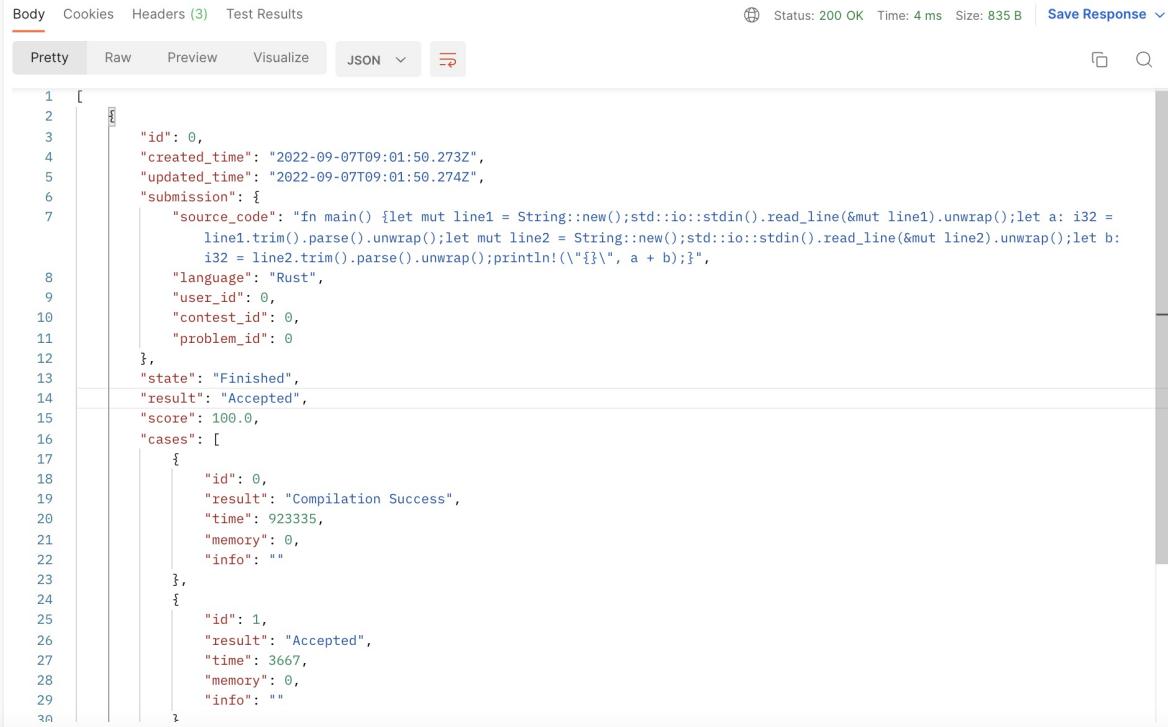




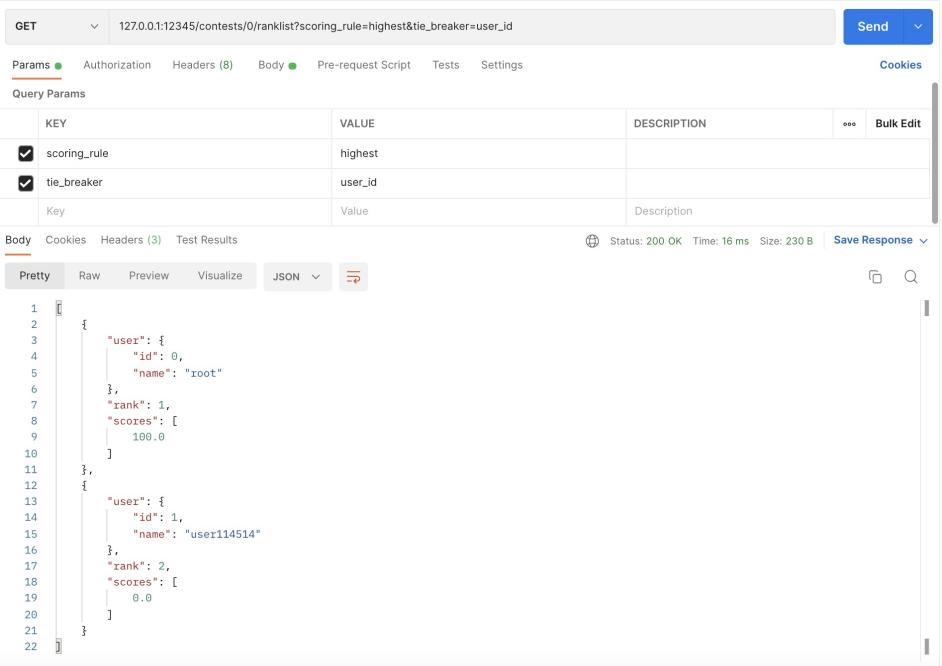
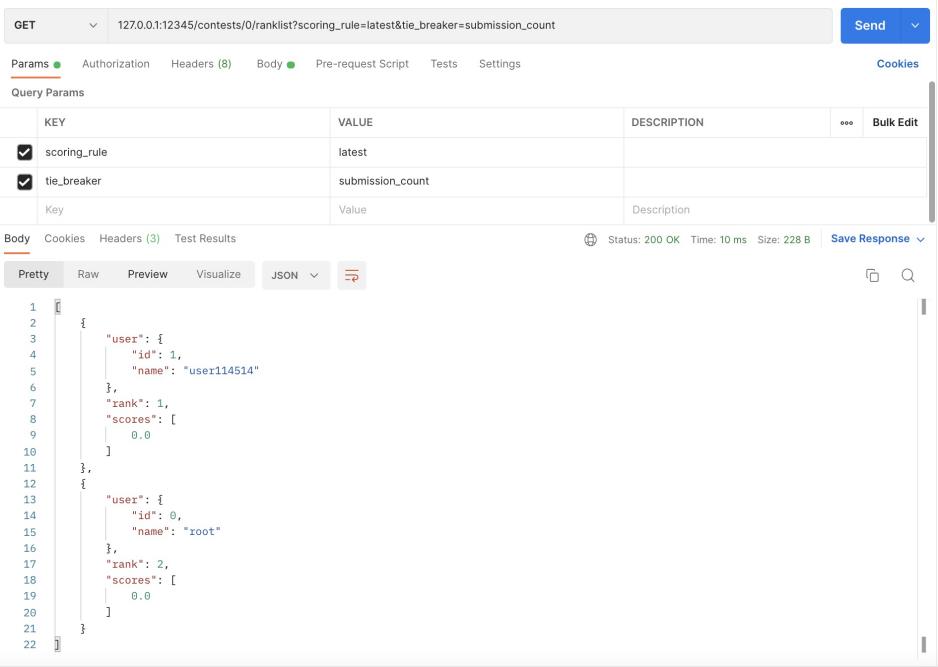








1. ranklist——user114514一次未提交却夺得第一名（喜）



1. **提高**
2. 多比赛

参见自动化测试（要素齐全）

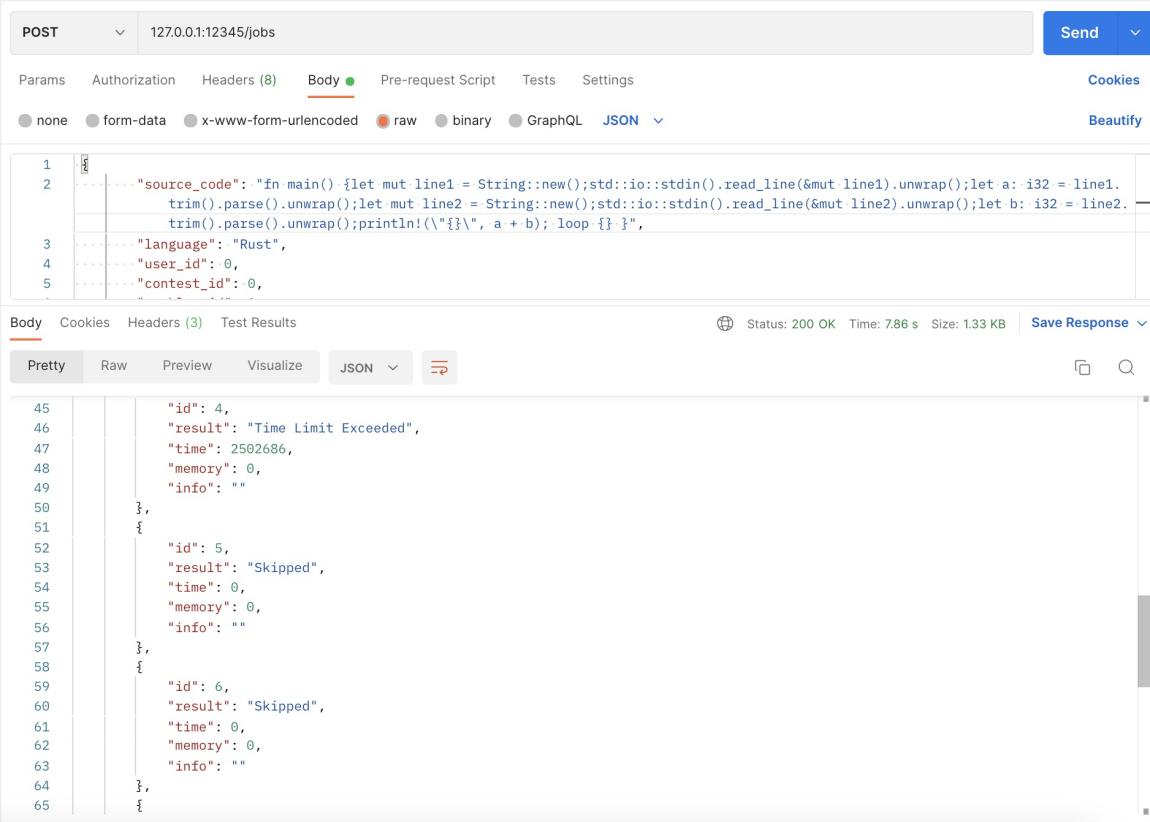
1. 持久化存储

src文件目录下jobs.json，users.json，contests.json三个文件分别持久化存储评测任务，用户以及比赛列表。

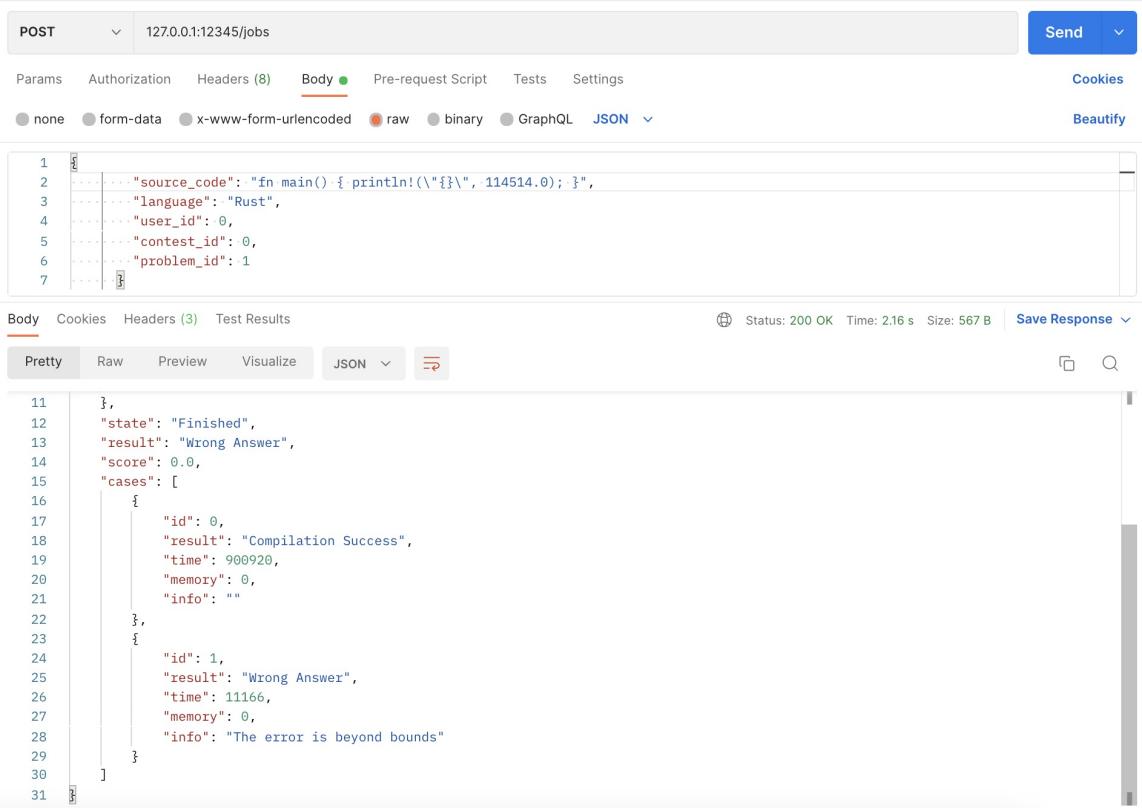


（例：上述基础部分测试服务器关闭后存储在users.json中的持久化数据）

1. 打包测试（使用自动测试的config文件）



1. Special Judge（使用自动测试的config文件）



1. 竞争得分

参见自动化测试结果（开摆）

Part 3.提高要求实现

多比赛功能的实现较为基础，其ranklist的实现同全局排行榜有较多代码复用。

使用Json文件持久化存储，每次Post指令后将Jobs，Users，Contests从全局变量中存入列表再写入文件中，下一次程序启动时如果有flush-data参数则清空文件（写入空列表 [] ）否则将文件内容转存至全局变量中，实现持久化。 

打包测试——在Post\_jobs中增加是否打包判断，若是则对每一个batch的测试点依次判断，若出现错误则中止该batch的评测，否则同非打包测试类似。

Special Judge——在match ProblemType中增加spj匹配，通过Command指令将比对结果输出至src/feedback.out文件，使用.lines().map()提取第一行（评测结果）以及第二行（更多信息），根据结果修改Jobs中内容。

竞争得分——在match ProblemType中增加Dynamic\_ranking匹配，对于通过的点只增加非竞争得分，在get\_ranklist中遍历所有涉及该题目的Jobs，并根据其运行时间计算竞争得分，然后在按照url参数排序。

Part 4.感想

第一次开展简单的后端项目编写，因为先前有Django后端基础，所以上手较快，不到一周时间完成了大部分基础提高内容，明显的感觉到对Rust语言的掌握程度有了显著提高。但在一些涉及到所有权的问题上往往通过Clone复制暴力解决，一些异常处理也较为失当。Rust常被冠以其所有权规范和异常处理见长，还需在更加深入的编程实践中加以体会。

最后感谢四周以来老师、助教的谆谆教诲与答疑解惑，程序设计训练课程即将告一段落，希望在将来的学习生活中再会！