项目名称：tzlOJ

文档名称：详细设计说明书

*HUSTZL*

*侯皓斐 软件2003班 U202010851*

*刘铭宸 软件2003班 U202010783*

*甘凤轩 软件2003班 U202010822*

版本：V2.1

文 档 信 息

|  |  |  |
| --- | --- | --- |
| 文件状态：  [ ] 草稿  [√ ] 正式发布  [ ] 正在修改 | 文件标识： | HUSTZL\_006 |
| 当前版本： | V2.1 |
| 作者： | 侯皓斐 |
| 发布日期： | < 2023.4.23 > |

文 档 更 改 记 录

| 版本 | 更改日期 | 更改人 | 更改原因 | 说明 |
| --- | --- | --- | --- | --- |
| V1.0 | 2023.4.15 | 侯皓斐 |  | 初版 |
| V2.0 | 2023.4.20 | 甘凤轩 | 校对 | 勘误并调整行文语句 |
| V2.1 | 2023.4.23 | 甘凤轩 | 调整格式 | 完善文档大纲，调整段落格式，修改序列编号 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1.模块结构 1](#_Toc133031087)

[1.1模块结构图 1](#_Toc133031088)

[1.2模块清单 2](#_Toc133031089)

[1.3后端结构图与模块设计 3](#_Toc133031090)

[2.前端设计 4](#_Toc133031091)

[2.1模块概述 4](#_Toc133031092)

[2.2功能与质量属性 5](#_Toc133031093)

[2.3输入输出项 6](#_Toc133031094)

[2.4数据结构 8](#_Toc133031095)

[2.5算法描述 8](#_Toc133031096)

[2.6限制条件 8](#_Toc133031097)

[2.7详细设计 9](#_Toc133031098)

[3.后端设计 17](#_Toc133031099)

[3.1模块概述 17](#_Toc133031100)

[3.2功能与质量属性 18](#_Toc133031101)

[3.3输入输出项 19](#_Toc133031102)

[3.4数据结构 21](#_Toc133031103)

[3.5算法描述 23](#_Toc133031104)

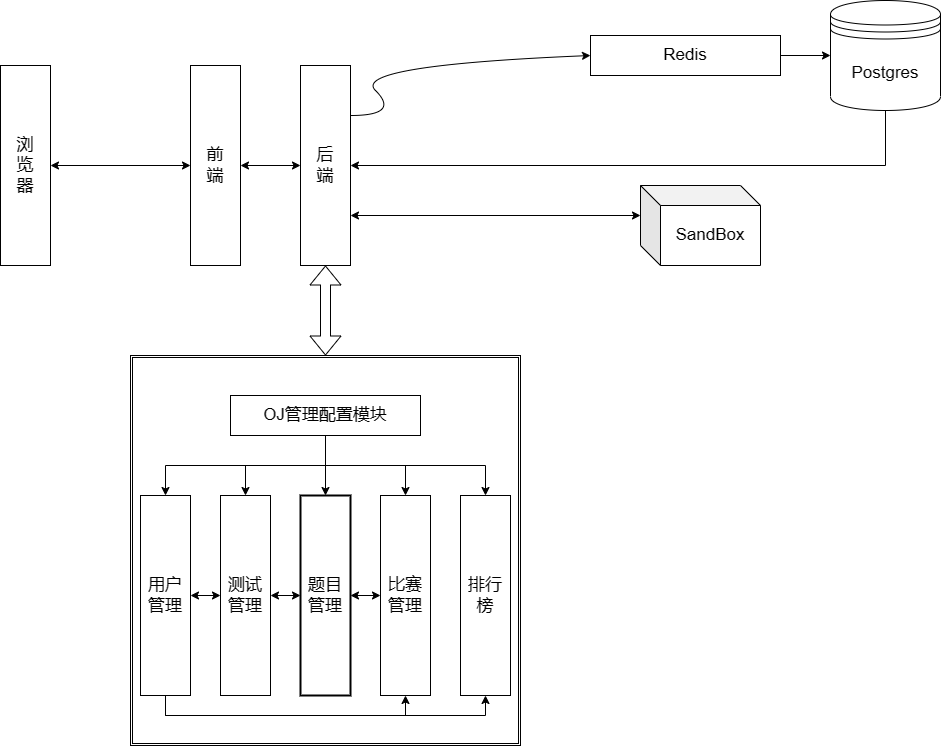
[3.6限制条件 24](#_Toc133031105)

[3.7详细设计 25](#_Toc133031106)

# 1.模块结构

## 1.1模块结构图

采用三层B/S模式，MVC模式和分层架构模式的架构图如下：



将系统分为三个层次：用户界面层、应用层、领域和基础设施层。

* 用户界面层：用户界面层是OnlineJudge系统的顶层，负责与用户进行交互。该层提供用户界面和交互逻辑，包括学生提交和管理员管理等功能。
* 应用层：应用层负责处理系统的业务逻辑和数据处理。该层包括自动评测系统、用户管理管理和比赛管理等模块。题目管理系统负责管理题目题面和测试数据等信息，测试管理负责利用SandBox自动编译和运行学生提交的代码，并生成评测报告，学生提交作业和查看提交历史等功能，用户管理负责管理账号提供用户信息，比赛管理负责比赛及其所属的题目的管理。排行榜子系统负责用户积分等信息，实现排行榜功能。
* 领域和基础设施层：领域和基础设施层负责系统的数据模型和数据访问。该层包括用户信息、成绩信息和提交历史等领域模型，以及数据访问接口和服务，而且是OnlineJudge系统的底层支撑，负责提供系统的底层服务和基础设施，特别是运行和编译程序的SandBox。我们由于约束条件按照设计我们采用了第三方API，使用docker部署和运行，我们redis，postgres均将使用官方提供的标准docker，SandBox将使用HUSTOJ的沙箱运行环境的docker，三者均直接从云端拉取。我们不再仔细开发领域与基础设施层，调用开源软件调用其API。

所以我们总体只需设计前端和后端两大模块。

## 1.2模块清单

前端子系统清单：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编号 | 模块名称 | 模块标识 | 输入 | 处理 | 输出 |
| 1 | 学生提交模块 | submission | 学生代码、题目ID、语言类型 | 检查提交格式、向后端发送评测请求 | 提交结果、评测报告 |
| 2 | 管理员管理模块 | admin | 管理员登录信息、题目信息、比赛信息 | 管理员身份验证、管理题目和比赛信息 | 成功或失败信息 |
| 3 | 用户界面模块 | user | 用户登录信息、题目信息、排行榜信息 | 显示用户信息、题目信息、排行榜信息 | 用户界面显示 |

后端子系统清单：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编号 | 模块名称 | 模块标识 | 输入 | 处理 | 输出 |
| 1 | 自动评测系统 | judge | 学生代码、题目信息、测试数据 | 编译和运行学生代码，生成评测报告 | 评测结果、评测报告 |
| 2 | 题目管理系统 | problem | 题目ID、题目信息、测试数据 | 管理题目和测试数据 | 题目和测试数据的增删改查 |
| 3 | 用户管理 | user | 用户信息、登录信息 | 管理用户信息 | 用户信息的增删改查 |
| 4 | 比赛管理 | contest | 比赛信息、题目信息 | 管理比赛和题目信息 | 比赛和题目信息的增删改查 |
| 5 | 排行榜子系统 | leaderboard | 用户积分信息、排行榜信息 | 管理用户积分信息和排行榜信息 | 排行榜信息的查询和更新 |
| 6 | 评测机管理模块 | JudgeManagement | 测评的代码，题目，数据 | 进行数据分析 | 合适的评测机 |

## 1.3后端结构图与模块设计

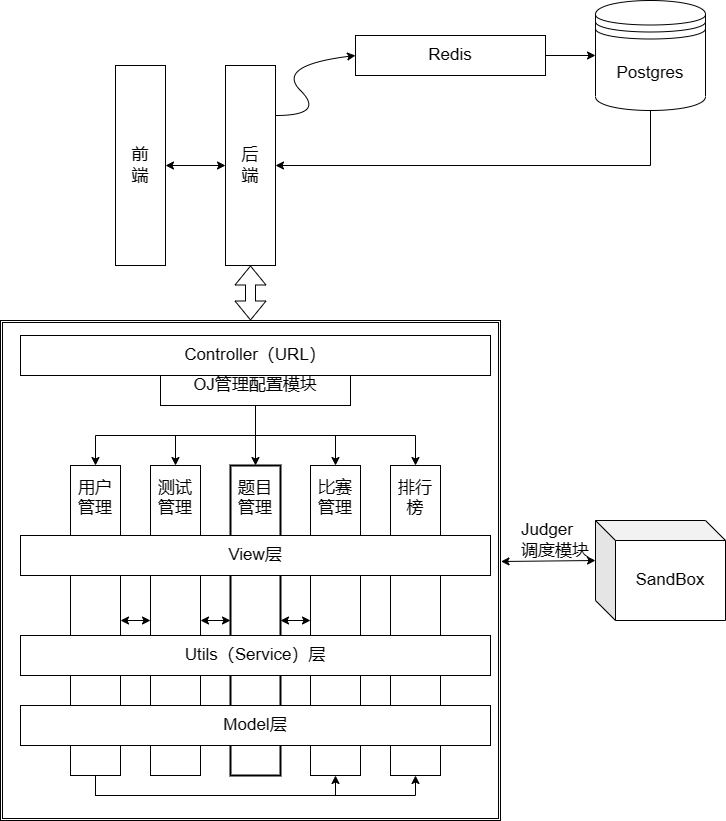
Django后端采用了MVC（Model-View-Controller）架构模式，将系统的各个功能分成了不同的层次和模块。下面是Django后端的结构分析：

* 模型层（Model）：该层负责处理系统的数据模型和数据库访问操作。系统中的每一个数据实体都对应着一个模型类，该类负责定义实体的属性和操作方法。在Django中，该层对应着models模块。
* 视图层（View）：该层负责处理系统的业务逻辑和数据处理。系统中的每一个业务实体都对应着一个视图函数，该函数负责处理实体的相关逻辑和操作。在Django中，该层对应着views模块。
* 控制器层（Controller）：该层负责处理系统的请求和响应，以及控制系统的流程和状态。系统中的每一个请求都对应着一个控制器类或函数，该类或函数负责处理请求并生成相应的响应。在Django中，该层对应着urls模块。

同时为了应对各个模块对应的请求，我们同时采用了模块化的架构，通过新加入的Service层减少混淆，使得代码可维护性和可扩展性大大提升。

而且我们为了提高测评效率，应该加入评测机管理模块，该模块主要负责管理系统中的评测机，包括评测机的添加、编辑、删除、维护等功能。我们采用心跳机制，测评机定期向后端服务器指定端口发送心跳信号，后端回应。第一次收到信号则将测评机加入列表，若3分钟后没有成功接收到心跳信号，则将其移除列表，采用合适的调度算法，将测评分布到合适的测评机上。

后端子系统图如下（模块内部结构图）：



# 2.前端设计

## 2.1模块概述

OnlineJudge系统核心设计理念在于构建一个支持众多编程语言的自动化评测平台，让学生能够在线提交代码并接收自动评测反馈。系统遵循前后端分离架构，前端负责用户界面和交互逻辑的呈现，而后端则处理业务逻辑和数据处理任务。

以下是前端模块的基本设计理念和处理流程：

1. 登录与注册模块：用户可在前端界面输入用户名和密码登录，若尚无账户，可通过注册页面创建新账号。
2. 题目列表与详情模块：前端展示所有可用编程题目，并支持筛选和排序功能。用户可查看题目详情，内容包括题目描述、输入输出样例、评测结果以及提交历史等。
3. 代码编辑与提交模块：前端提供代码编辑器，用户可在此编辑并提交代码。编辑器支持多种编程语言，同时提供自动缩进、语法高亮和自动补全等功能。
4. 评测结果与排行榜模块：前端展示评测结果页面，用户可查看提交代码的评测结果，如编译错误、运行错误、通过测试点数等。排行榜页面展示用户排名及积分等信息。

## 2.2功能与质量属性

作为一名专业的软件架构师，我将为您详细描述一个OnlineJudge系统的功能要求和关键质量属性。这个系统将包含以下功能：

|  |  |  |
| --- | --- | --- |
| 输入 | 处理 | 输出 |
| 用户名、密码 | 登录验证 | 用户登录后的页面 |
| 用户名、密码 | 注册验证，创建新账号 | 用户注册成功后的页面 |
| 搜索条件、排序方式 | 根据条件和方式筛选和排序题目，展示题目列表 | 经过筛选和排序后的题目列表页面 |
| 题目详情页中的代码编辑器，用户的代码 | 将用户的代码提交到后端进行评测，展示评测结果 | 用户在题目详情页中查看到的评测结果 |
| 题目详情页中的输入输出样例，用户的代码 | 在编辑器中编辑代码，保存到题目提交历史中 | 用户在题目详情页中查看到自己提交的代码和提交时间 |
| 用户资料中的用户名、密码、邮箱 | 编辑用户资料，将编辑后的信息保存到后端进行验证和更新 | 用户编辑后的资料，或者提示用户编辑内容不符合规定的错误信息 |
| 比赛页面中的比赛信息 | 展示比赛基本信息、排名、规则和题目列表，允许用户参与 | 用户在比赛中获得的成绩和积分，或者提示用户不能参与比赛的错误信息 |
| 用户管理、题目管理和比赛管理页面的操作 | 根据用户的操作对用户、题目或比赛进行管理 | 用户在管理页面上进行操作后的结果或者提示信息 |
| 用户积分、排名 | 根据用户的活动和积分规则计算用户的积分和排名 | 用户当前的排名和积分，或者提示用户当前无法查看排行榜的错误信息 |

OnlineJudge系统的重要质量属性为性能和安全。以下是前端质量设计的具体描述：

性能：强调前端设计中的响应时间和吞吐量，为用户提供高速提交代码和评测结果反馈，支持大量用户并发访问与测评。

安全：关注以下几个方面：

* 安全控制与物理保护：采用防火墙、加密传输、防病毒措施等保护系统安全。
* 用户身份验证：采用用户名和密码登录、双因素认证等保证用户身份安全。
* 访问权限与范围：根据用户角色和权限限制系统访问范围和操作权限，确保安全性。
* 病毒防护措施：预防系统受到病毒或恶意软件攻击。
* 数据加密：采用SSL/TLS加密传输、用户数据加密算法等保障用户数据安全。

## 2.3输入输出项

输入项：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 标识符 | 类型 | 介质 | 来源 | 描述 |
| 用户名 | username | 字符串 | 输入框 | 用户输入 | 用户在登录或注册时输入的用户名 |
| 密码 | password | 字符串 | 输入框 | 用户输入 | 用户在登录或注册时输入的密码 |
| 搜索条件 | keyword | 字符串 | 输入框 | 用户输入 | 用户在搜索题目时输入的关键词 |
| 排序方式 | sort | 字符串 | 下拉菜单 | 用户选择 | 用户在搜索题目时选择的排序方式 |
| 题目代码 | code | 字符串 | 编辑器 | 用户输入或复制粘贴 | 用户在编辑器中输入或复制粘贴的题目代码 |
| 提交记录 | submission | 字符串 | 文本框 | 后端返回 | 用户在题目详情页中查看到的提交记录 |
| 用户资料 | profile | 对象 | 输入框 | 用户输入或后端返回 | 用户在个人信息管理页面中查看和编辑的用户资料 |
| 管理操作 | operation | 字符串 | 下拉菜单 | 管理员选择 | 管理员在用户管理、题目管理和比赛管理页面中选择的管理操作类型 |
| 比赛成绩 | score | 字符串 | 文本框 | 后端返回 | 用户在比赛页面中查看到的自己的成绩 |
| 比赛ID | contestID | 字符串 | URL参数 | URL参数 | 用户在访问比赛页面时从URL中传递的比赛ID |
| 题目ID | problemID | 字符串 | URL参数 | URL参数 | 用户在访问题目详情页时从URL中传递的题目ID |

输出项：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 标识符 | 类型 | 介质 | 目的 | 描述 |
| 登录状态 | loginStatus | 字符串 | 页面上的提示框 | 用户登录成功或失败后的状态提示 | 提示用户登录是否成功或失败 |
| 注册状态 | registerStatus | 字符串 | 页面上的提示框 | 用户注册成功或失败后的状态提示 | 提示用户注册是否成功或失败 |
| 题目列表 | problemList | 对象数组 | 题目列表页面 | 显示符合搜索条件和排序方式的所有题目信息 | 包括题目ID、标题、难度、通过率等信息 |
| 题目详情 | problemDetail | 对象 | 题目详情页面 | 显示某个题目的具体信息，包括题目描述、输入输出样例、评测结果等 | 包括题目ID、标题、难度、通过率、输入输出样例、评测结果等信息 |
| 评测结果 | judgeResult | 字符串 | 页面上的提示框 | 显示用户提交代码的评测结果 | 包括编译错误、运行错误和通过测试点数量等信息 |
| 提交记录 | submissionList | 对象数组 | 题目详情页面 | 显示用户提交该题目的历史记录 | 包括提交时间、状态、用时、用空间等信息 |
| 用户资料 | profile | 对象 | 个人信息管理页面 | 显示用户个人资料信息，包括用户名、密码、邮箱等 | 用户在个人信息管理页面中查看和编辑的用户资料 |
| 比赛信息 | contestInfo | 对象 | 比赛页面 | 显示比赛的基本信息，包括比赛名称、时间、规则等 | 包括比赛ID、比赛名称、开始时间、结束时间、规则等信息 |
| 比赛排名 | contestRank | 对象数组 | 比赛页面 | 显示比赛的参与者排名和积分 | 包括用户ID、用户名、积分等信息 |
| 管理操作结果 | operationResult | 字符串 | 页面上的提示框 | 显示管理员进行的用户管理、题目管理和比赛管理操作后的结果 | 包括操作成功或失败的提示信息 |
| 排行榜信息 | rankingInfo | 对象 | 排行榜页面 | 显示所有用户的排名、用户名、积分等信息 | 包括用户ID、用户名、排名、积分等信息 |
| 用户积分、排名 | userRanking | 对象 | 页面上的提示框 | 显示用户的积分和排名 | 包括用户ID、用户名、排名、积分等信息 |

## 2.4数据结构

前端主要采用调用后端的方法获取数据，而本身需要存储的数据十分少，所以应该对不同的页面进行不同的设计，主要存储TOKEN和当前状态下提交的代码（以字符串形式）就能满足所有的数据存储需要，所以我们不再仔细设计逻辑上的数据结构。直接存储到浏览器存储池即可。通常情况下，前端数据结构与后端程序实现之间的数据交互采用统一的接口格式，比如JSON格式。前端通过调用后端的API接口，将前端页面中的数据提交给后端进行处理，并将后端处理结果返回到前端进行展示。因此，前端数据结构与程序实现之间的关系主要体现在数据交互的接口规范上，而不是具体的数据结构实现。

## 2.5算法描述

前端主要采用调用后端的方法计算数据，而本身数据计算很少，没有算法可言。

## 2.6限制条件

在开发OnlineJudge前端系统时，需要考虑以下限制条件：

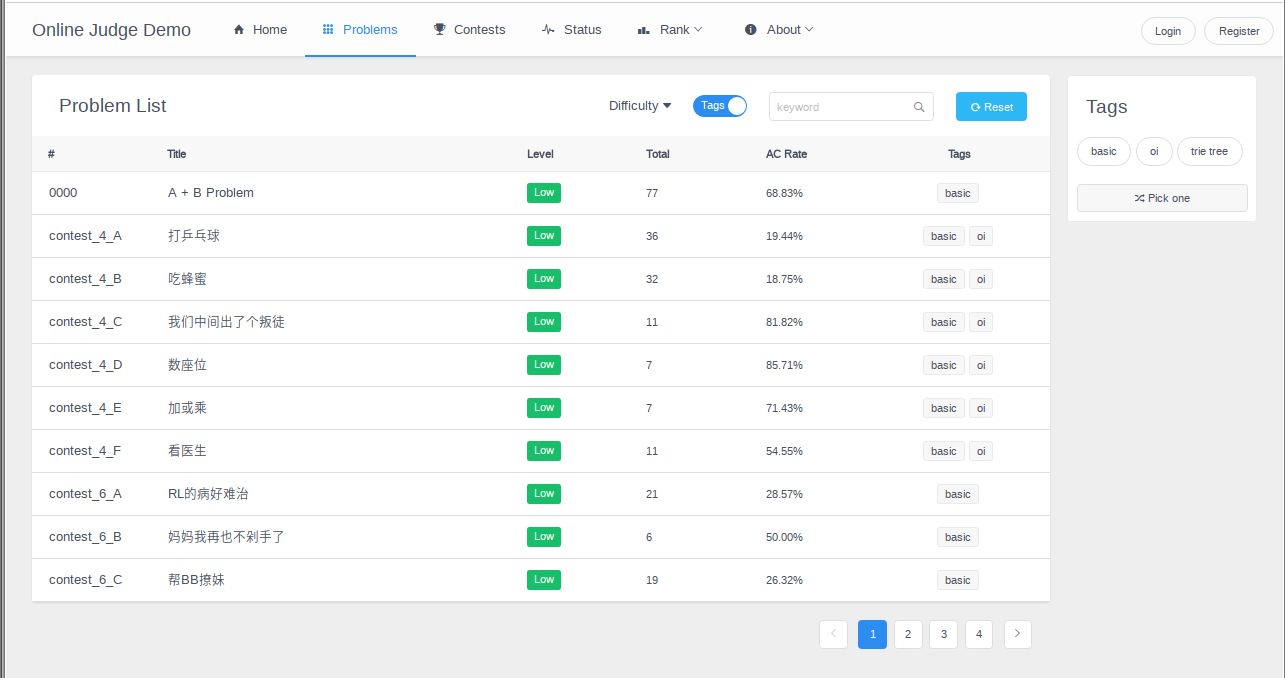
* 浏览器兼容性：确保系统在不同浏览器（如Google Chrome、Mozilla Firefox、Safari、Microsoft Edge等）和版本上保持一致的表现和功能。
* 技术选型：根据项目需求和团队技能选择合适的前端技术栈，我们只能选择Vue框架。
* 工期限制：前端开发的工期通常一个月左右。
* 经费限制：前端开发的经费在五万元到八万元之间。在有限的经费下，前端开发需要权衡不同需求的优先级，以保证开发的效率和质量。

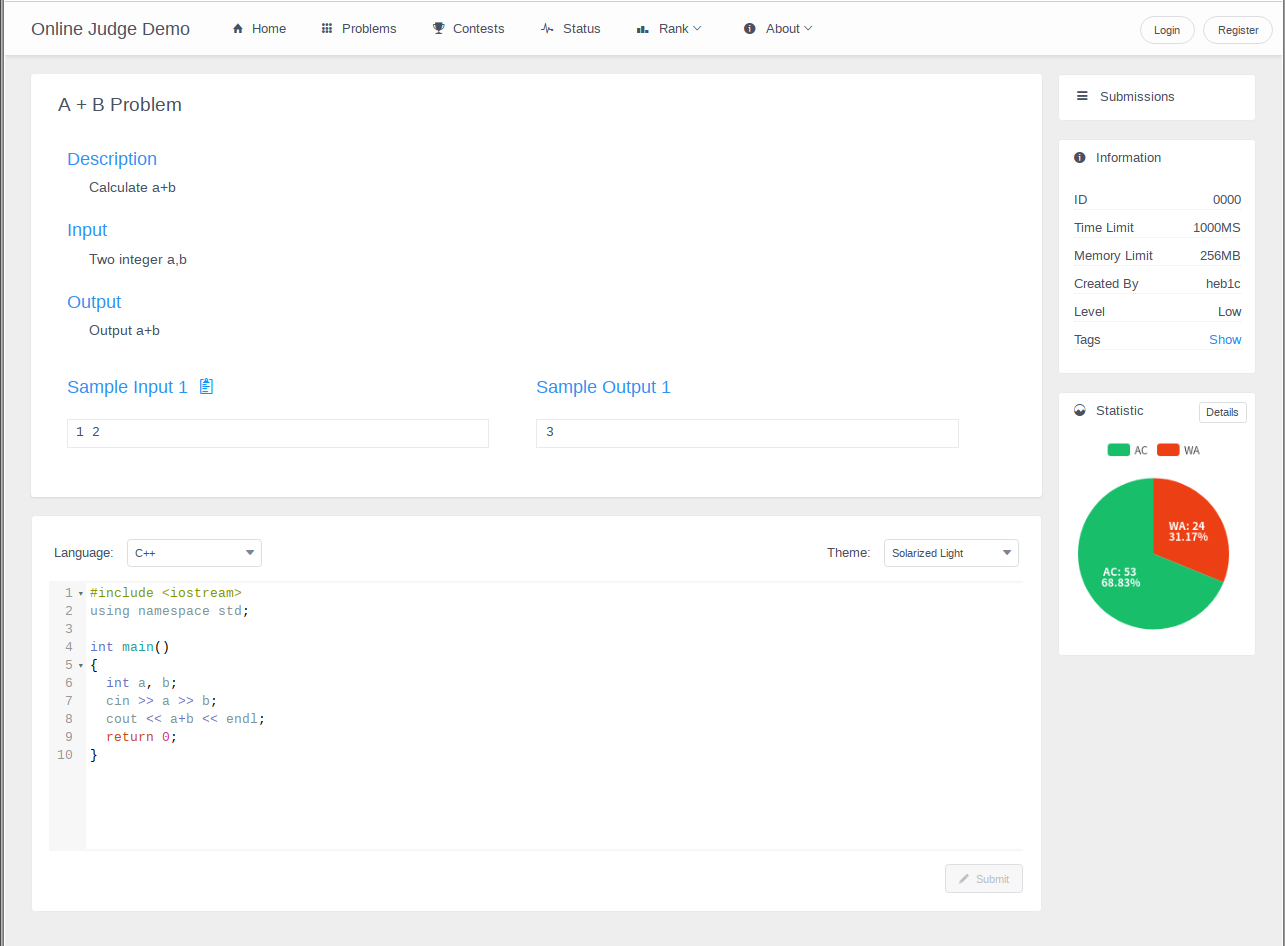
## 2.7详细设计

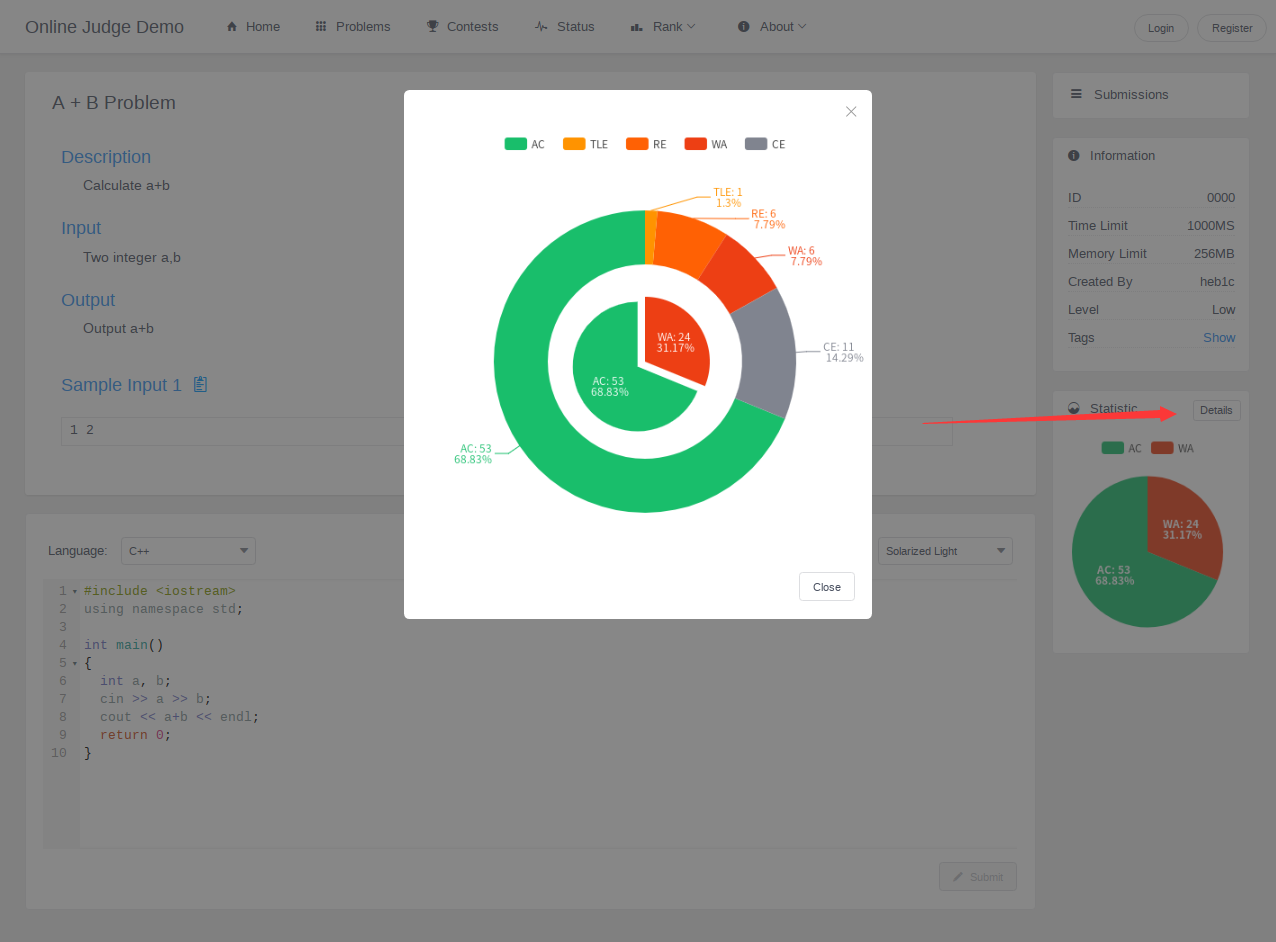
前端主要采用调用后端的方法获取数据，而本身需要存储的数据十分少，所以应该对不同的页面进行不同的设计，实现各个页面的嵌套关系以及跳转关系即可。

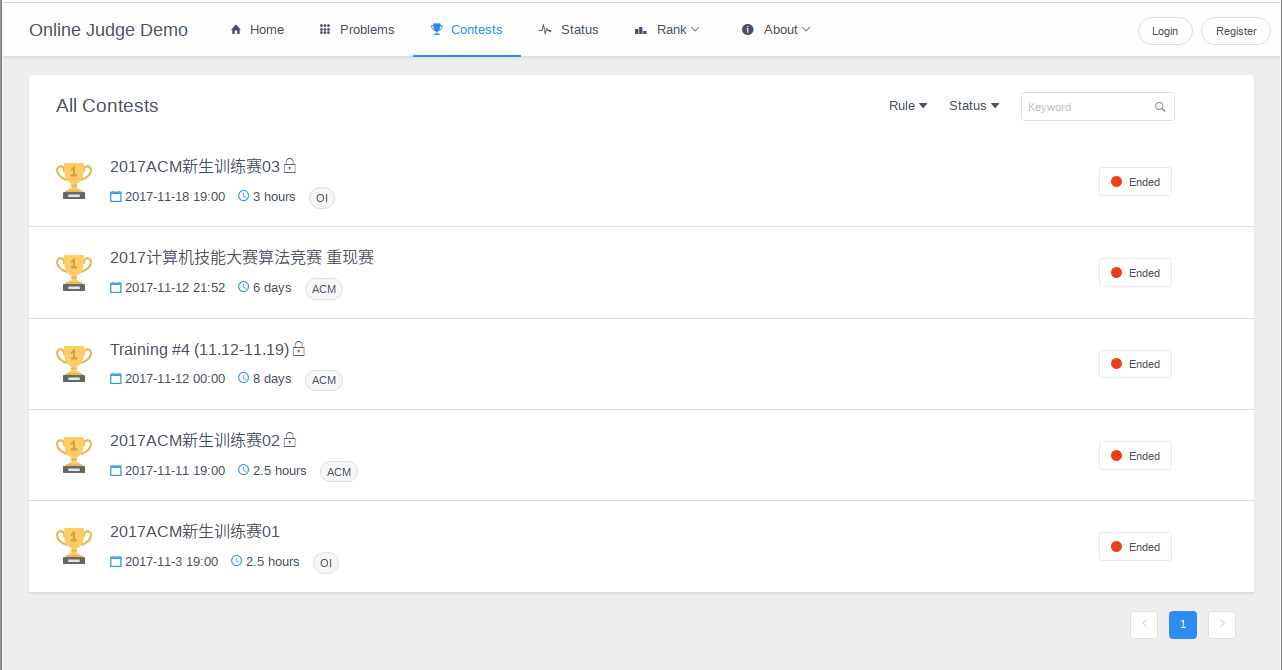
|  |  |  |
| --- | --- | --- |
| **类结构及类** | **标识符** | **功能** |
| 用户登录模块 | UserLoginPage | 负责用户登录和验证 |
| 题目列表模块 | ProblemListPage | 负责展示所有可用的编程题目列表 |
| 题目详情模块 | ProblemDetailPage | 负责展示题目详情页，包括题目描述、输入输出样例、评测结果和提交历史等信息 |
| 代码编辑模块 | CodeEditorPage | 负责提供代码编辑器，支持多种编程语言，并提供自动缩进、语法高亮和自动补全等功能 |
| 代码提交模块 | CodeSubmitPage | 负责将用户提交的代码发送到后端评测机进行评测 |
| 评测结果页面模块 | SubmissionResultPage | 负责展示评测结果页面，用户可以查看自己提交的代码的评测结果，包括编译错误、运行错误、通过测试点的数量等信息 |
| 排行榜模块 | RankListPage | 负责展示用户的排名和积分等信息 |
| 后台模块 | AdminPage | 负责进行管理员操作，增加题目，修改题目，举办比赛等 |

用户接口即为图形化界面，我们从各方面展示我们定稿的系统设计UI如下：

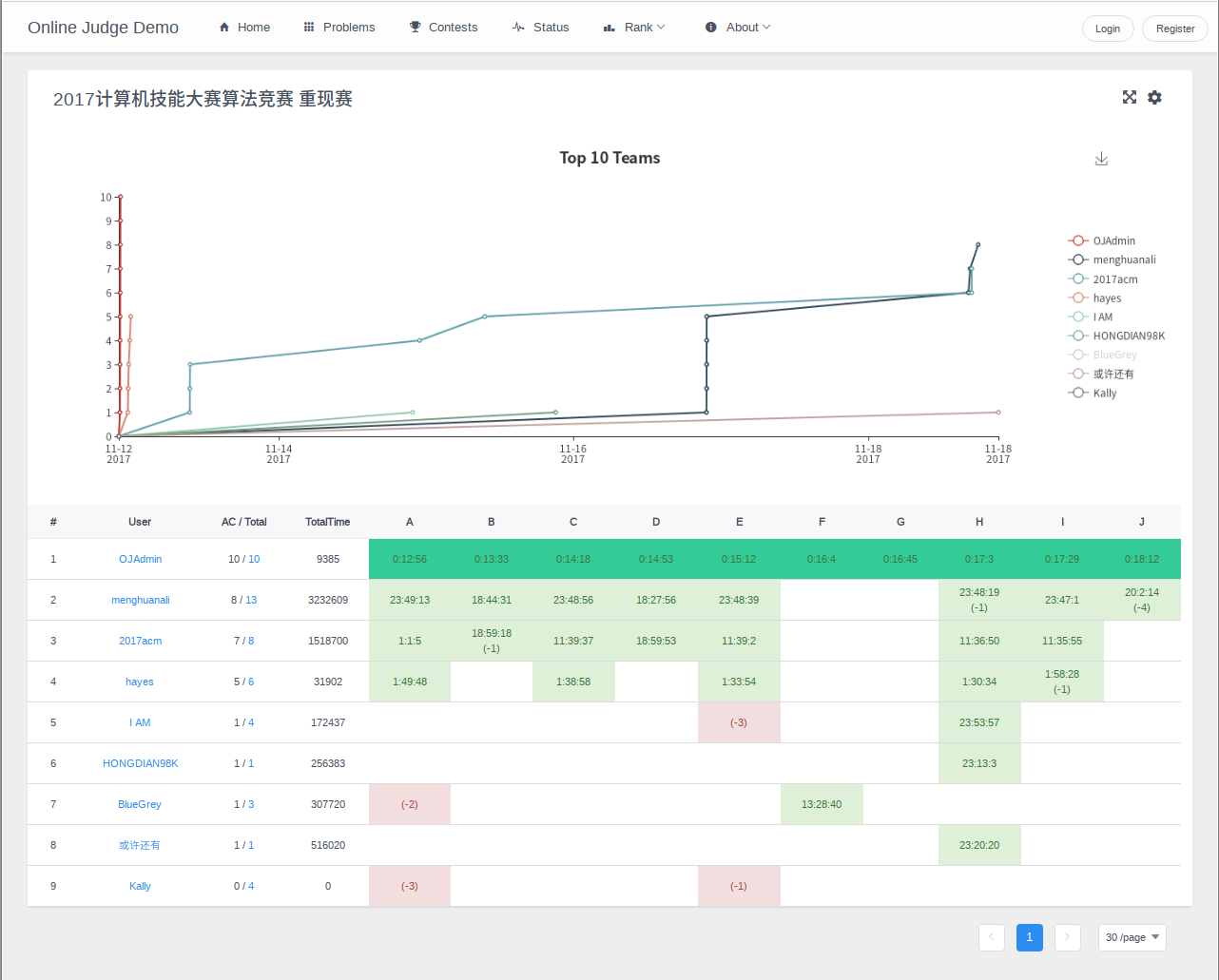
[](https://user-images.githubusercontent.com/20637881/33372506-402022e4-d539-11e7-8e64-6656f8ceb75a.png)

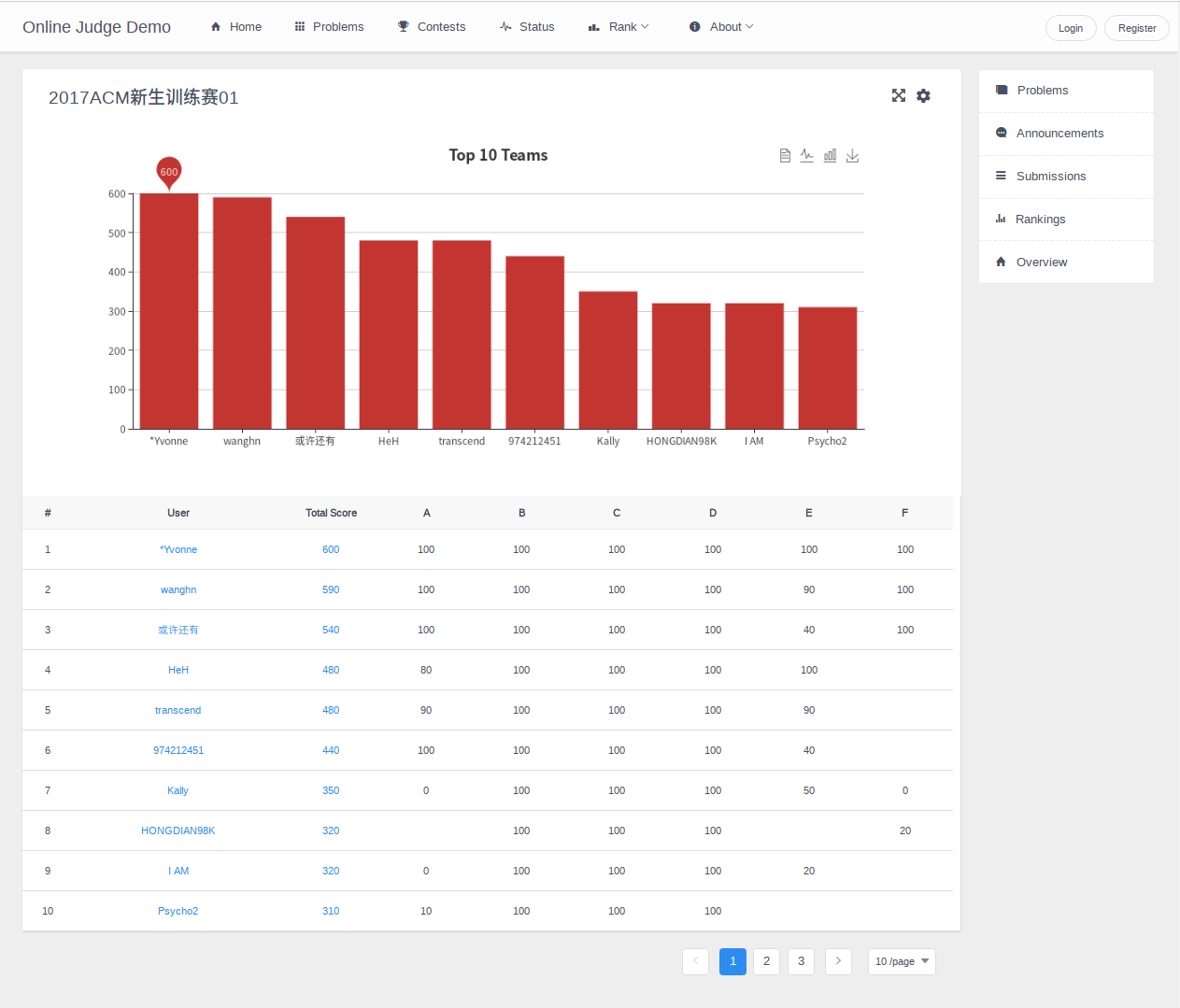
[](https://user-images.githubusercontent.com/20637881/33372507-4061a782-d539-11e7-8835-076ddae6b529.png)

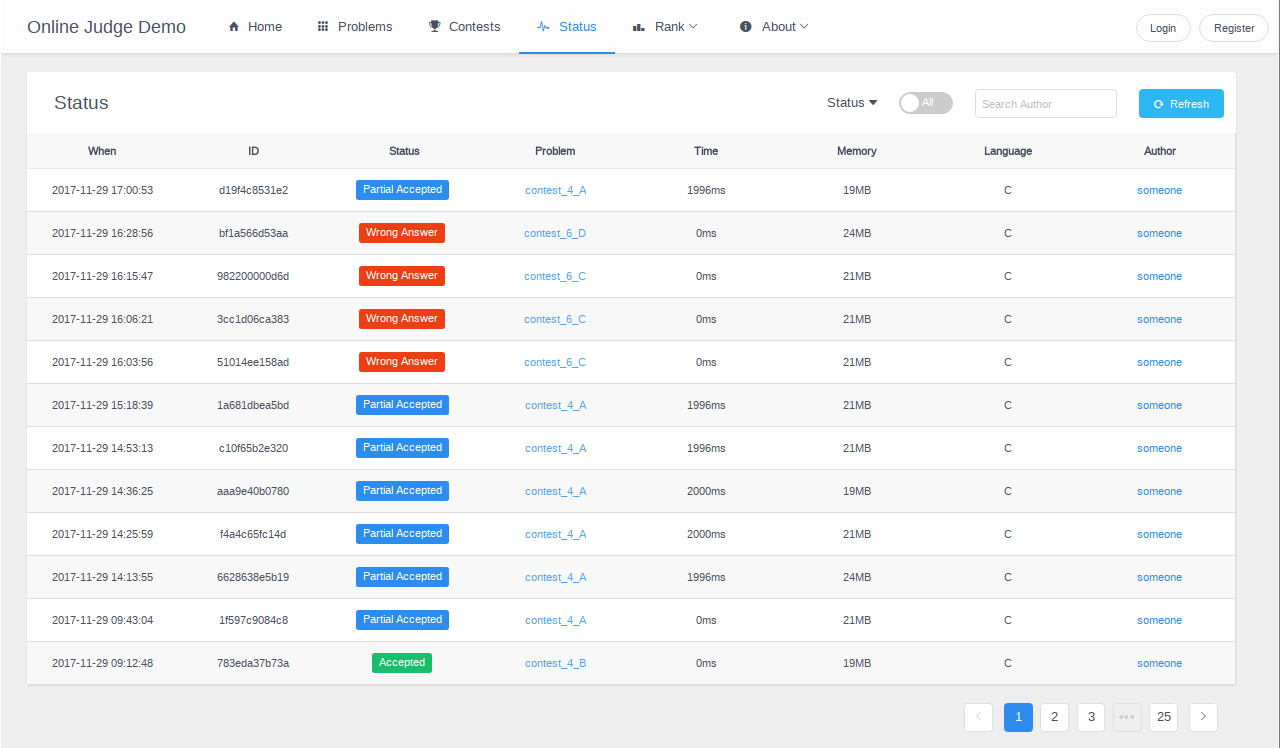
[](https://user-images.githubusercontent.com/20637881/33372508-40a0c6ce-d539-11e7-8d5e-024541b76750.png)

[](https://user-images.githubusercontent.com/20637881/33372509-40d880dc-d539-11e7-9eba-1f08dcb6b9a0.png)

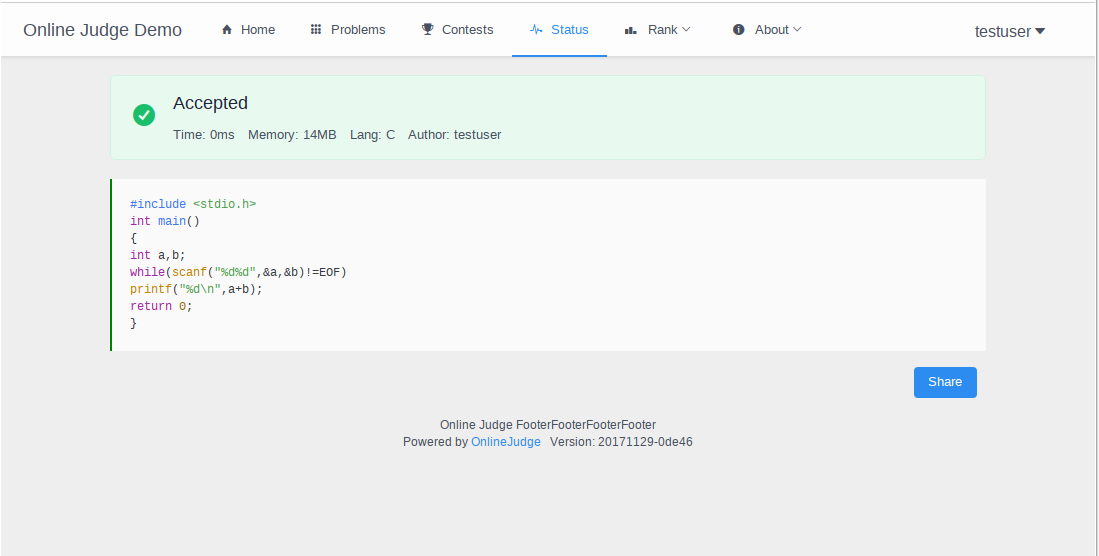
排行榜模块：

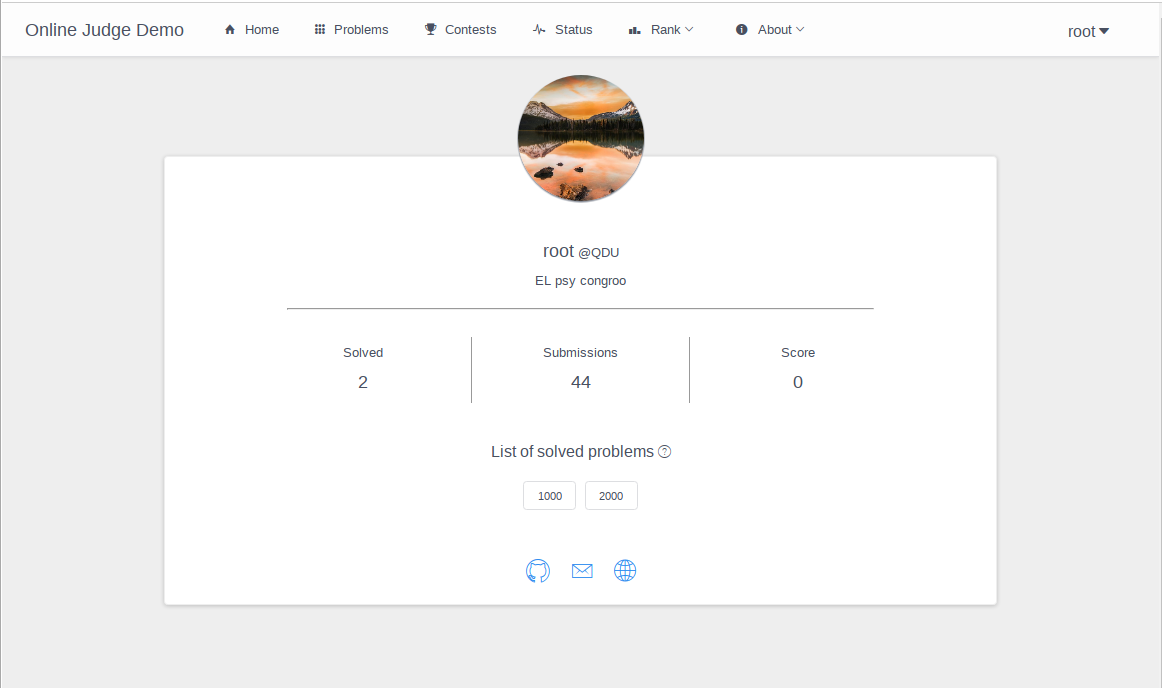
[](https://user-images.githubusercontent.com/20637881/33372510-41117f68-d539-11e7-9947-70e60bad3cf2.png)

[](https://user-images.githubusercontent.com/20637881/33372511-41d406fa-d539-11e7-9947-7a2a088785b0.png)

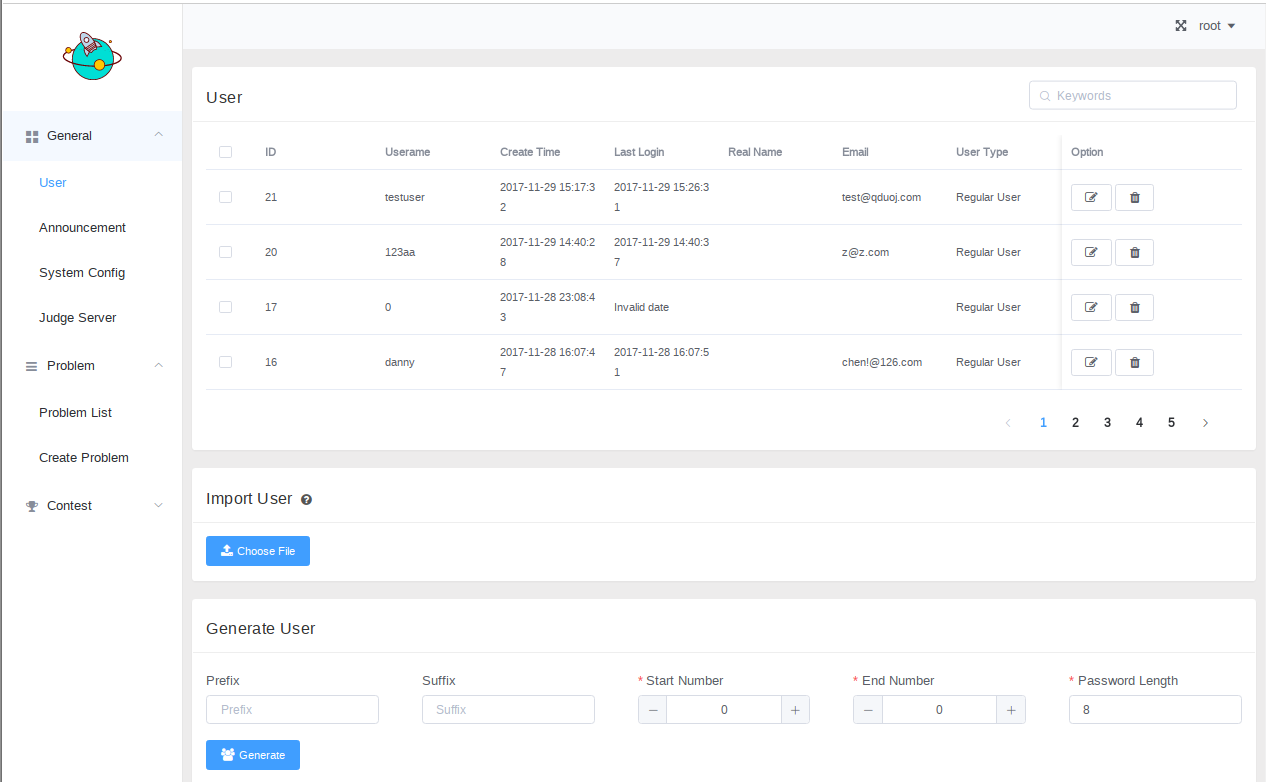
[](https://user-images.githubusercontent.com/20637881/33372512-420ba240-d539-11e7-8645-594cac4a0b78.png)

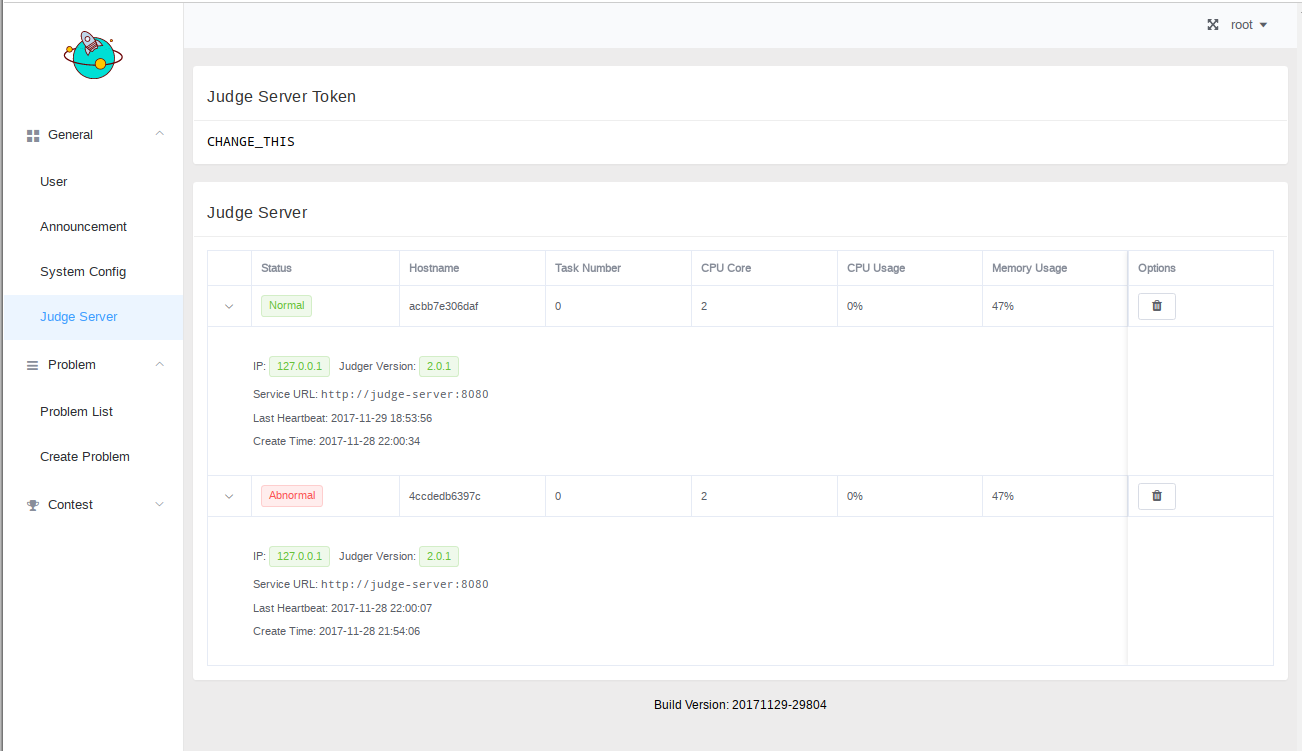
测评模块：

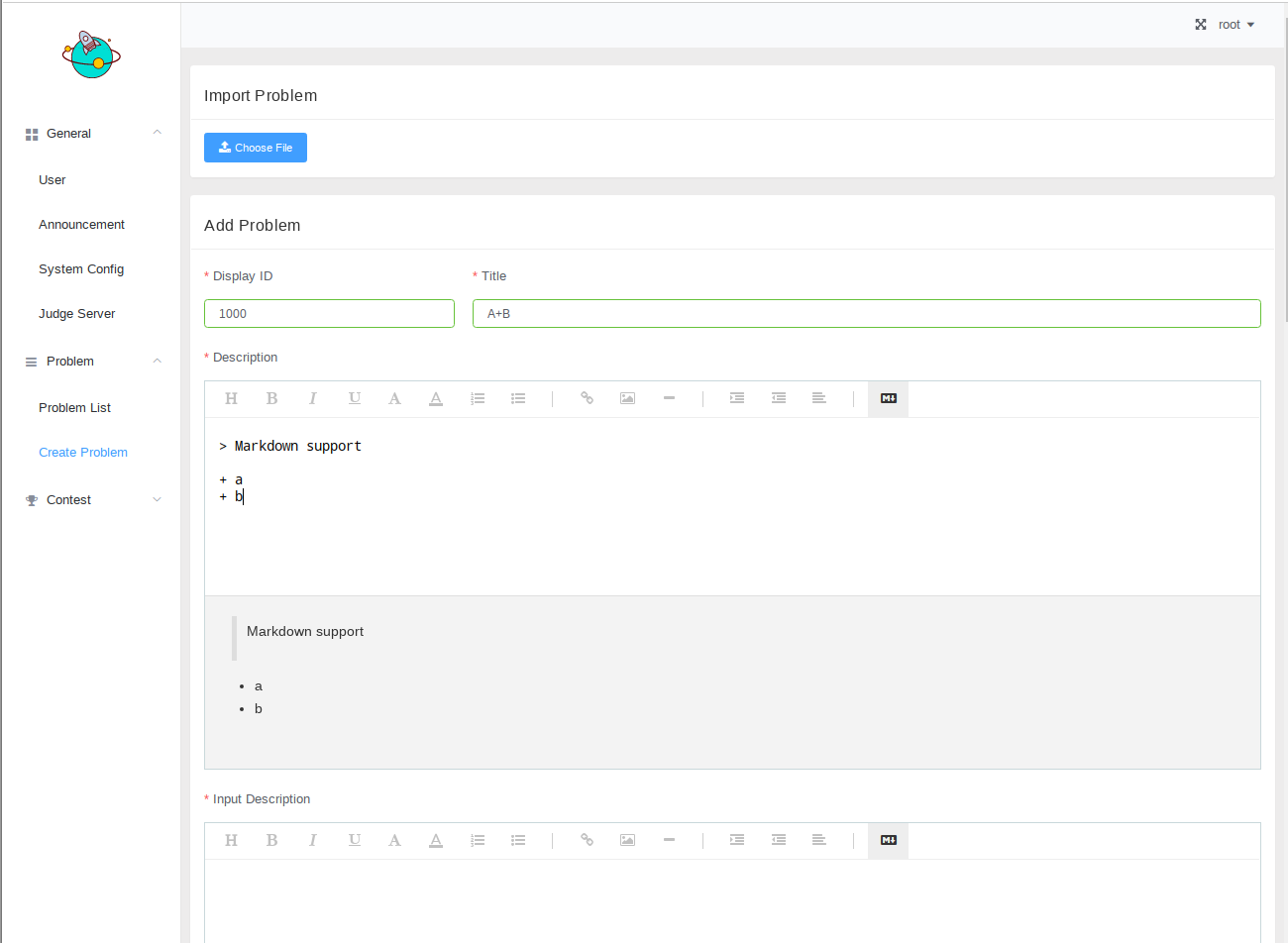
[](https://user-images.githubusercontent.com/20637881/33365523-787bd0ea-d523-11e7-953f-dacbf7a506df.png)

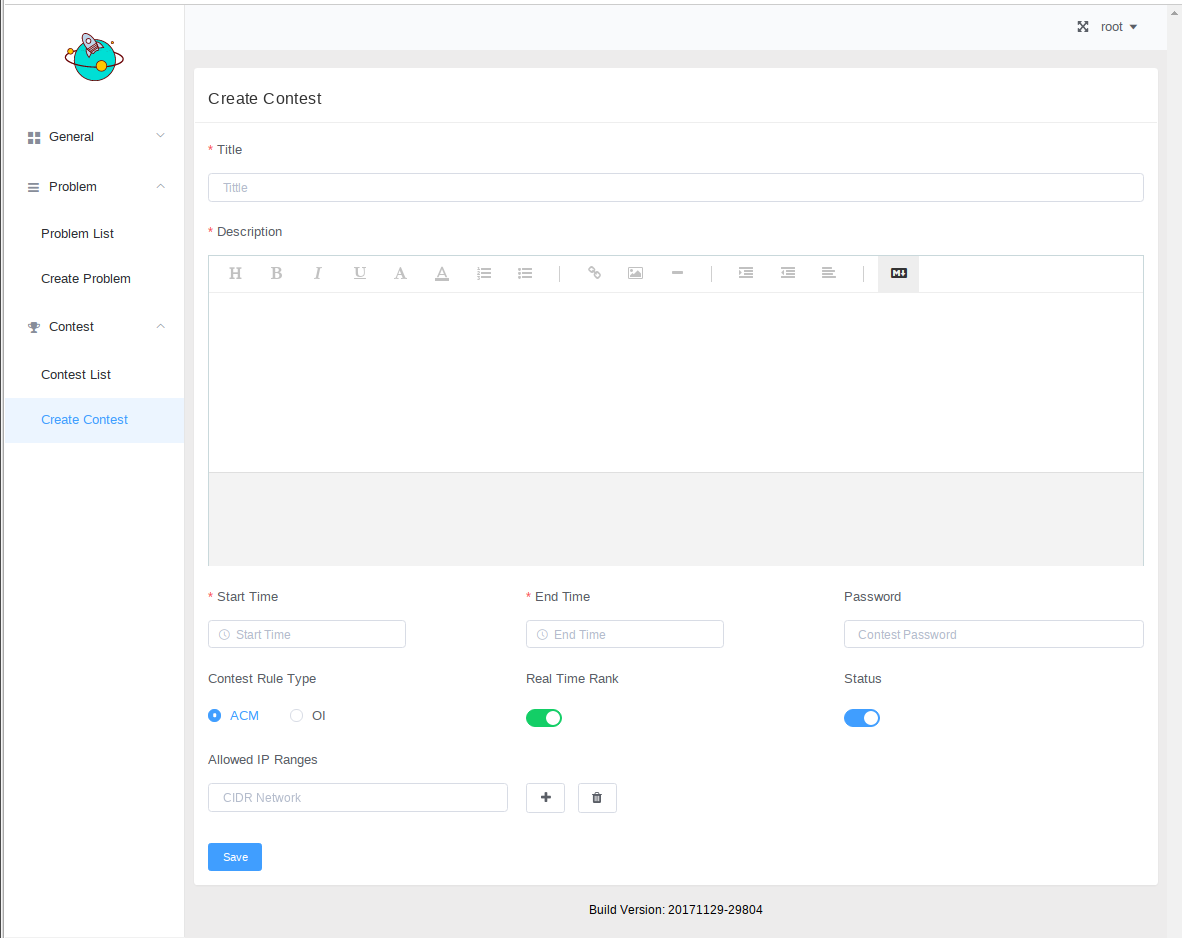
[](https://user-images.githubusercontent.com/20637881/33365521-7842d808-d523-11e7-84c1-2e2aa0079f32.png)

后台管理模块：

[](https://user-images.githubusercontent.com/20637881/33372516-42c34fda-d539-11e7-9f4e-5109477f83be.png)

[](https://user-images.githubusercontent.com/20637881/33372517-42faef9e-d539-11e7-9f17-df9be3583900.png)

[](https://user-images.githubusercontent.com/20637881/33372513-42472162-d539-11e7-8659-5497bf52dbea.png)

[](https://user-images.githubusercontent.com/20637881/33372514-428ab922-d539-11e7-8f68-da55dedf3ad3.png)

# 3.后端设计

## 3.1模块概述

OnlineJudge后端的基本设计思想是实现一个高效稳定、安全可靠的在线评测系统，主要包括三个方面的设计思想：

1. 模块化设计：将整个后端系统分解为多个独立的子系统，每个子系统负责不同的功能和业务逻辑，以实现系统的高内聚低耦合。
2. 面向对象设计：采用面向对象的设计方法，将系统中的各个对象进行抽象和封装，以实现系统的可扩展性和维护性。
3. 高并发处理：针对在线评测系统高并发的特点，采用分布式架构和异步处理等技术手段，以实现系统的高并发处理和高性能。

后端处理模块大致如下：

1. 用户管理模块：负责用户身份认证和授权管理，包括用户登录、注册、身份认证、权限管理等功能。
2. 题目管理模块：负责管理系统中的所有编程题目，包括题目添加、编辑、删除、搜索、排序等功能。
3. 提交管理模块：负责管理用户提交的编程代码，包括代码编译、运行、测试、评测等功能。
4. 排名管理模块：负责管理用户的排名和积分等信息，以实现排行榜功能。
5. 比赛管理模块：负责管理系统中的比赛，包括比赛添加、编辑、删除、管理等功能。
6. 数据库管理模块：负责管理系统中所有数据的存储和访问，包括用户数据、题目数据、提交数据、答案数据等。
7. 评测机管理模块：负责管理系统中的评测机，包括评测机的添加、编辑、删除、维护等功能。

以上各个模块之间相互关联，通过数据交换和接口调用等方式进行协作和协同工作，最终实现整个OnlineJudge系统的功能和业务逻辑。

## 3.2功能与质量属性

前端和后端通过数据交换和接口调用等方式进行协作和协同工作，最终实现整个OnlineJudge系统的功能和业务逻辑。所以功能需求不在此赘述，必须考虑的质量属性列举如下：

1. 性能：
   1. 吞吐量：OnlineJudge系统的吞吐量要求高，需要能够支持同时多个用户的访问和请求。为此，系统采用分布式架构和负载均衡等技术手段，以实现系统的高并发处理和高吞吐量。
   2. 并发速度：OnlineJudge系统需要能够快速响应用户的请求，包括题目查询、提交、评测等操作。为此，系统采用异步处理、缓存优化、数据库索引等技术手段，以实现系统的高并发处理和高性能。
2. 安全：
   1. 安全控制和物理保护措施：OnlineJudge系统采用多层安全措施，包括网络安全、主机安全、应用程序安全等方面的措施，以保障系统的安全性。同时，系统还采用物理保护措施，包括数据备份、灾备恢复、设备监控等方面的措施，以防止硬件故障和数据丢失等问题。
   2. 用户身份鉴别机制：OnlineJudge系统采用严格的用户身份鉴别机制，包括密码强度检测、双因素身份认证、访问控制等措施，以确保用户身份的合法性和数据的安全性。
   3. 用户对系统的访问权限和范围：OnlineJudge系统采用细粒度的访问控制策略，以确保用户只能访问其有权限访问的数据和功能。同时，系统还采用审计和日志记录等措施，以追踪和监控用户的操作行为。
   4. 病毒的防治措施：OnlineJudge系统采用病毒扫描、漏洞扫描等安全措施，以防止系统受到病毒、木马等恶意软件的攻击和侵害。
3. 可靠性：OnlineJudge系统需要保证高可靠性，即在面对各种异常情况下，系统仍能够保持正常的运行和服务。为此，系统采用备份和容灾等措施，以确保系统的可用性和可靠性。
4. 可扩展性：OnlineJudge系统需要具备良好的可扩展性，即在需要增加或减少系统的规模时，能够快速、便捷地进行系统扩展或缩减。为此，系统采用分布式架构、容器化技术、自动化部署等措施，以提高系统的可扩展性和灵活性。
5. 可维护性：OnlineJudge系统需要具备良好的可维护性，即在系统出现故障或需要进行更新和维护时，能够快速、便捷地进行修复和升级。为此，系统采用模块化设计等措施，以提高系统的可维护性和可扩展性。

## 3.3输入输出项

输入项如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 标识符 | 类型 | 介质 | 来源 | 描述 |
| 用户名 | username | 字符串 | 请求参数/表单数据 | 用户提交 | 用户的账号名 |
| 密码 | password | 字符串 | 请求参数/表单数据 | 用户提交 | 用户的账号密码 |
| 题目ID | problem\_id | 整数 | URL参数/请求参数 | 用户提交 | 题目的唯一标识符 |
| 题目名称 | problem\_name | 字符串 | 请求参数/表单数据 | 管理员/教师 | 题目的名称 |
| 题目描述 | problem\_description | 字符串 | 请求参数/表单数据 | 管理员/教师 | 题目的详细描述 |
| 题目输入格式 | input\_format | 字符串 | 请求参数/表单数据 | 管理员/教师 | 题目输入数据的格式描述 |
| 题目输出格式 | output\_format | 字符串 | 请求参数/表单数据 | 管理员/教师 | 题目输出数据的格式描述 |
| 样例输入 | sample\_input | 字符串 | 请求参数/表单数据 | 管理员/教师 | 题目的样例输入数据 |
| 样例输出 | sample\_output | 字符串 | 请求参数/表单数据 | 管理员/教师 | 题目的样例输出数据 |
| 时间限制 | time\_limit | 整数 | 请求参数/表单数据 | 管理员/教师 | 题目的时间限制 |
| 内存限制 | memory\_limit | 整数 | 请求参数/表单数据 | 管理员/教师 | 题目的内存限制 |
| 难度等级 | difficulty\_level | 字符串 | 请求参数/表单数据 | 管理员/教师 | 题目的难度等级 |
| 提交ID | submission\_id | 整数 | URL参数/请求参数 | 用户提交 | 提交的唯一标识符 |
| 题目ID | problem\_id | 整数 | 请求参数/表单数据 | 用户提交 | 提交的题目ID |
| 用户ID | user\_id | 整数 | 请求参数/表单数据 | 用户提交 | 提交的用户ID |
| 代码 | code | 字符串 | 请求参数/表单数据 | 用户提交 | 用户提交的代码 |
| 总分 | total\_score | 浮点数 | 数据库记录 | 系统内部 | 用户获得的总分数 |
| 比赛ID | contest\_id | 整数 | URL参数/请求参数 | 管理员/教师 | 比赛的唯一标识符 |
| 比赛名称 | contest\_name | 字符串 | 请求参数/表单数据 | 管理员/教师 | 比赛的名称 |
| 评测机ID | judge\_id | 整数 | 请求参数/表单数据 | 管理员/教师 | 评测机的唯一标识符 |
| 评测机名称 | judge\_name | 字符串 | 请求参数/表单数据 | 管理员/教师 | 评测机的名称 |
| 评测机IP | judge\_ip | 字符串 | 请求参数/表单数据 | 管理员/教师 | 评测机的IP地址 |
| 评测机端口 | judge\_port | 整数 | 请求参数/表单数据 | 管理员/教师 | 评测机的端口号 |
| 评测机状态 | judge\_status | 字符串 | 数据库记录 | 系统内部 | 评测机的状态，如空闲、繁忙等 |
| 评测机描述 | judge\_description | 字符串 | 请求参数/表单数据 |  |  |

输出项如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 标识符 | 类型 | 介质 | 目的 | 描述 |
| 用户ID | user\_id | 整数 | 数据库记录 | 用户验证 | 用户的唯一标识符 |
| 用户名 | username | 字符串 | 数据库记录 | 用户验证 | 用户的账号名 |
| 用户角色 | user\_role | 字符串 | 数据库记录 | 用户验证 | 用户的角色，如学生、教师、管理员等 |
| 认证令牌 | auth\_token | 字符串 | HTTP响应头/数据库记录 | 用户认证 | 用户成功登录后返回的认证令牌 |
| 题目列表 | problem\_list | 列表 | HTTP响应正文 | 题目管理 | 系统中所有的题目列表 |
| 题目详情 | problem\_detail | 字典 | HTTP响应正文 | 题目管理 | 单个题目的详细信息，包括名称、描述、输入输出格式、样例数据、时间限制、内存限制、难度等级等 |
| 提交详情 | submission\_detail | 字典 | HTTP响应正文 | 提交管理 | 单个提交的详细信息，包括提交ID、用户ID、题目ID、提交时间、编译结果、运行结果等 |
| 排名列表 | rank\_list | 列表 | HTTP响应正文 | 排名管理 | 系统中用户的排名列表，按照通过次数和总分数排序 |
| 用户排名 | user\_rank | 字典 | HTTP响应正文 | 排名管理 | 单个用户的排名信息，包括排名、通过次数、总分数等 |
| 比赛列表 | contest\_list | 列表 | HTTP响应正文 | 比赛管理 | 系统中所有比赛的列表 |
| 比赛详情 | contest\_detail | 字典 | HTTP响应正文 | 比赛管理 | 单个比赛的详细信息，包括名称、开始时间、结束时间、规则等 |
| 评测机列表 | judge\_list | 列表 | HTTP响应正文 | 评测机管理 | 系统中所有评测机的列表 |
| 评测机详情 | judge\_detail | 字典 | HTTP响应正文 | 评测机管理 | 单个评测机的详细信息，包括名称、IP地址、端口号、状态等 |

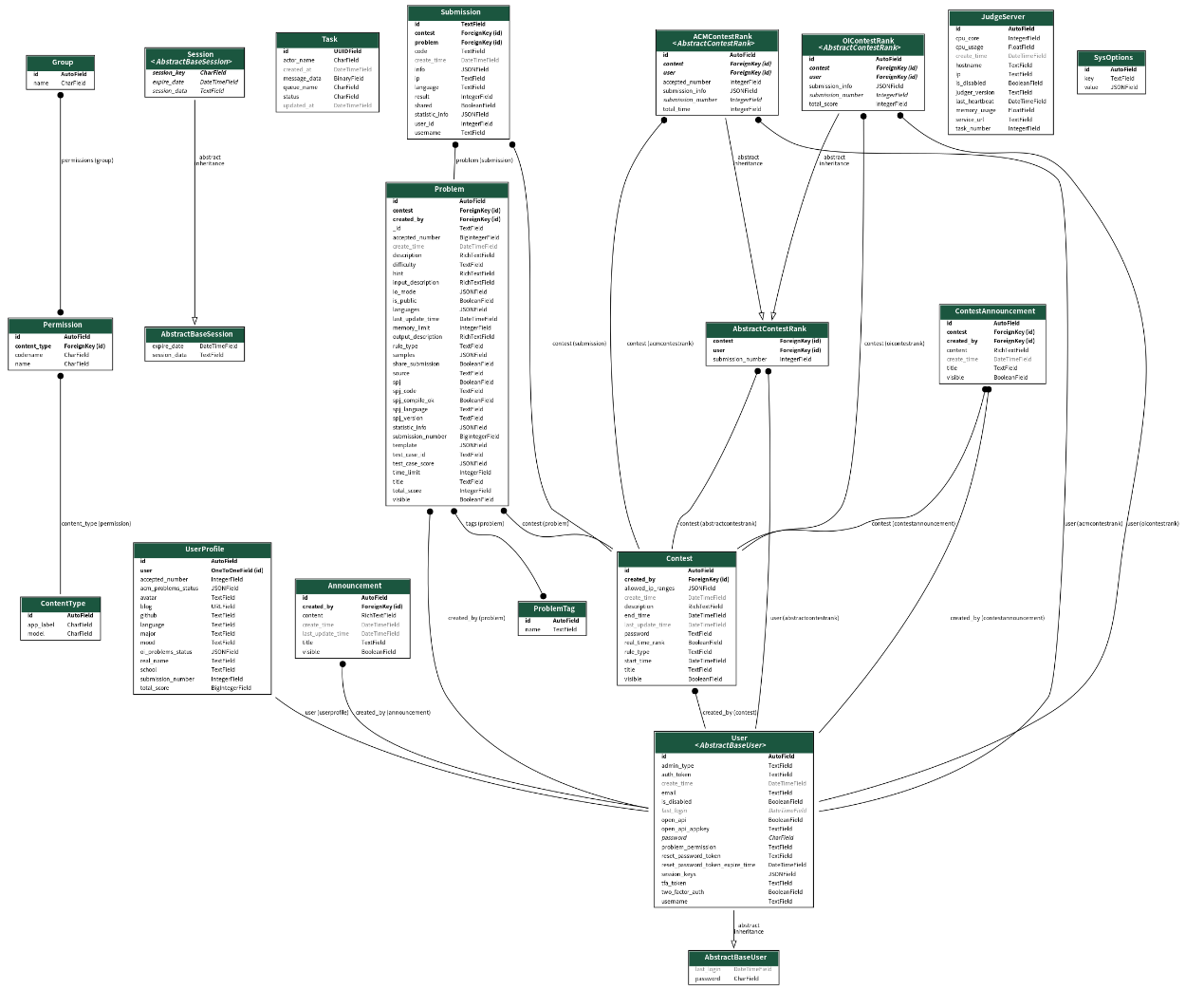
## 3.4数据结构

OnlineJudge后端数据结构主要包括用户数据、题目数据、提交数据、答案数据和比赛数据等。

1. 用户数据结构：用户数据结构包括用户ID、用户名、密码、邮箱、角色等信息。其中，用户ID是唯一的标识符，用户名和密码用于身份认证和授权，邮箱用于找回密码和接收通知等，角色用于区分不同用户的权限和功能等。
2. 题目数据结构：题目数据结构包括题目ID、题目标题、题目描述、输入格式、输出格式、样例输入、样例输出、时间限制、空间限制、难度等信息。其中，题目ID是唯一的标识符，题目标题和描述用于展示题目的基本信息和要求，输入格式和输出格式用于指导用户编写代码和进行测试，样例输入和样例输出用于验证用户代码的正确性，时间限制和空间限制用于评测用户代码的性能，难度用于指导用户选择适当的题目进行练习和挑战。
3. 提交数据结构：提交数据结构包括提交ID、用户ID、题目ID、提交时间、提交代码、提交状态等信息。其中，提交ID是唯一的标识符，用户ID和题目ID用于标识用户和题目，提交时间用于记录用户提交代码的时间，提交代码用于保存用户提交的代码内容，提交状态用于记录代码的评测状态和结果等。
4. 答案数据结构：答案数据结构包括题目ID、测试数据、标准输出等信息。其中，题目ID用于标识题目，测试数据用于进行代码测试和评测，标准输出用于验证用户代码的正确性。
5. 比赛数据结构：比赛数据结构包括比赛ID、比赛标题、比赛描述、开始时间、结束时间、题目列表等信息。其中，比赛ID是唯一的标识符，比赛标题和描述用于展示比赛的基本信息和要求，开始时间和结束时间用于控制比赛的开始和结束时间，题目列表用于指导用户选择比赛题目和进行比赛。

以上数据结构在OnlineJudge后端中起到重要的作用，通过各种机制和接口实现了系统的各种功能和特性，为用户提供了高效、可靠、可扩展和易维护的服务和支持。

详细的数据结构设计见下图：



## 3.5算法描述

评测机调度算法是在线评测系统的重要组成部分之一。它的主要功能是根据评测机的状态和负载情况，将待评测的代码分配给空闲的评测机进行评测，以最大化系统的吞吐量和响应速度。

1. 系统中每个评测机都有一个状态，包括空闲、繁忙、维护中等。当评测机空闲时，它可以接受系统分配的任务进行评测。
2. 当有新的代码需要评测时，系统将代码放入待评测队列中，并检查空闲的评测机列表，如果存在空闲的评测机，则将代码分配给其中一个评测机进行评测。
3. 如果所有评测机都处于繁忙状态，则系统将在待评测队列中等待一段时间，等待有评测机空闲后再分配任务。
4. 当有评测机完成评测任务后，系统会将评测结果保存，并将该评测机状态改为空闲状态。
5. 系统应该能够自适应地调整待评测队列和评测机的负载，以尽可能地提高系统的吞吐量和响应速度。
6. 为了确保评测机的负载均衡，系统应该将代码按照一定的规则分配给评测机进行评测，例如按照代码的类型、难度、提交时间等等进行分配。系统应该根据自身的需求和情况来选择最合适的分配规则，以最大化系统的吞吐量和响应速度。同时，在实际应用中，系统应该能够动态地调整代码分配规则，并支持用户自定义分配规则。
   1. 按照代码类型进行分配：系统可以根据代码的类型，例如ACM/ICPC或OI，将代码分配给不同的评测机进行评测。因为不同的代码类型可能需要不同的评测环境和测试数据，所以根据代码类型进行分配可以更好地利用系统资源，提高系统的效率。
   2. 按照代码难度进行分配：系统可以根据题目的难度级别，将代码分配给具有相应评测能力的评测机进行评测。对于一些难度较高的题目，需要使用更加强大的评测机进行评测，这样可以更好地保证评测的质量和准确性。
   3. 按照提交时间进行分配：系统可以根据代码的提交时间，将代码分配给最近一次空闲的评测机进行评测。这种分配方式可以保证评测机的利用率，同时也能够更快地响应用户的请求。
   4. 按照用户ID进行分配：系统可以根据用户的ID，将用户提交的代码分配给该用户上一次使用的评测机进行评测。这种分配方式可以提高评测的速度，同时也能够更好地满足用户的需求。
7. 为了保证系统的可靠性和稳定性，系统应该支持故障恢复和监控，包括评测机状态的监控、评测任务的失败重试、评测机的自动重启等。

## 3.6限制条件

后端开发的限制条件主要包括以下几个方面：

* 工期限制：后端开发时间需要1-2个月左右。
* 经费限制：后端开发的经费30万元左右，不得超过35万元。
* 技术限制：后端开发所使用的技术和框架应该能够满足项目需求，并且需要考虑到技术的成熟度和稳定性等因素，更要考虑到项目组和架构师熟悉的框架，我们不得不选用Django+Redis+Postgres的组合。
* 安全限制：在线评测系统后端开发需要考虑到安全性问题，例如用户认证和授权、数据加密和防止SQL注入等问题。这些安全问题需要花费一定的时间和资源来解决，并且需要进行充分的测试和验证，以确保系统的安全性和稳定性。

## 3.7详细设计

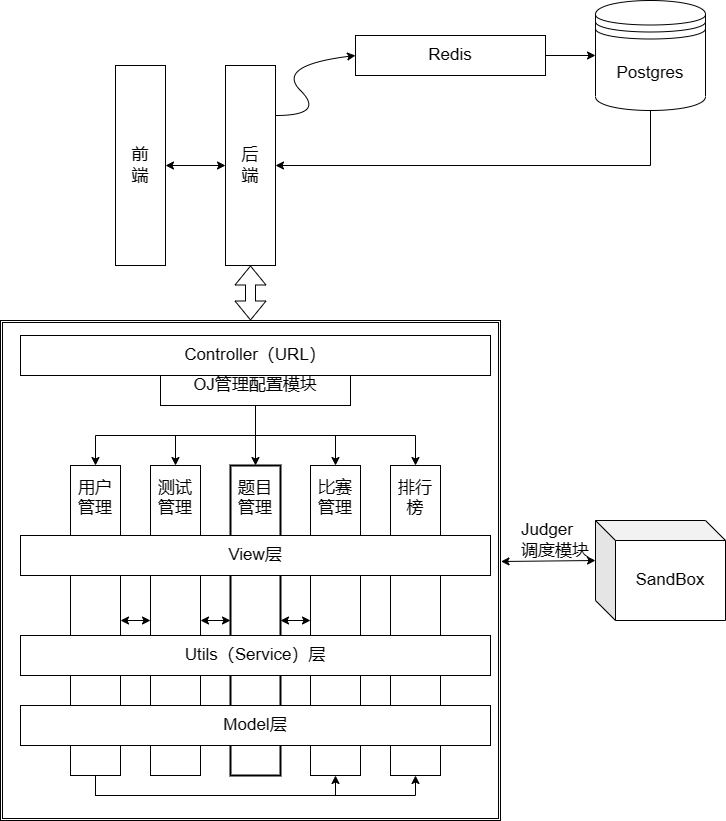
关系数据库的具体设计详见3.4节。

Django后端采用了MVC（Model-View-Controller）架构模式，将系统的各个功能分成了不同的层次和模块。下面是Django后端的结构分析：

* 模型层（Model）：该层负责处理系统的数据模型和数据库访问操作。系统中的每一个数据实体都对应着一个模型类，该类负责定义实体的属性和操作方法。在Django中，该层对应着models模块。
* 视图层（View）：该层负责处理系统的业务逻辑和数据处理。系统中的每一个业务实体都对应着一个视图函数，该函数负责处理实体的相关逻辑和操作。在Django中，该层对应着views模块。
* 控制器层（Controller）：该层负责处理系统的请求和响应，以及控制系统的流程和状态。系统中的每一个请求都对应着一个控制器类或函数，该类或函数负责处理请求并生成相应的响应。在Django中，该层对应着urls模块。

同时为了应对各个模块对应的请求，我们同时采用了模块化的架构，通过新加入的Service层减少混淆，使得代码可维护性和可扩展性大大提升。

而且我们为了提高测评效率，应该加入评测机管理模块，该模块主要负责管理系统中的评测机，包括评测机的添加、编辑、删除、维护等功能。我们采用心跳机制，测评机定期向后端服务器指定端口发送心跳信号，后端回应。第一次收到信号则将测评机加入列表，若3分钟后没有成功接收到心跳信号，则将其移除列表，采用合适的调度算法，将测评分布到合适的测评机上。



我们在此细化前端与后端交互的接口：

我们在此按模块进行细化：

1. Utils：urlpatterns = [

url(r"^upload\_image/?$", SimditorImageUploadAPIView.as\_view(), name="upload\_image"),

url(r"^upload\_file/?$", SimditorFileUploadAPIView.as\_view(), name="upload\_file")

]

1. Submission：
   1. Admin：urlpatterns = [

url(r"^submission/rejudge?$", SubmissionRejudgeAPI.as\_view(), name="submission\_rejudge\_api"),

]

* 1. OJ：urlpatterns = [

url(r"^submission/?$", SubmissionAPI.as\_view(), name="submission\_api"),

url(r"^submissions/?$", SubmissionListAPI.as\_view(), name="submission\_list\_api"),

url(r"^submission\_exists/?$", SubmissionExistsAPI.as\_view(), name="submission\_exists"),

url(r"^contest\_submissions/?$", ContestSubmissionListAPI.as\_view(), name="contest\_submission\_list\_api"),

]

1. Problem：
   1. Admin：urlpatterns = [

url(r"^test\_case/?$", TestCaseAPI.as\_view(), name="test\_case\_api"),

url(r"^compile\_spj/?$", CompileSPJAPI.as\_view(), name="compile\_spj"),

url(r"^problem/?$", ProblemAPI.as\_view(), name="problem\_admin\_api"),

url(r"^contest/problem/?$", ContestProblemAPI.as\_view(), name="contest\_problem\_admin\_api"),

url(r"^contest\_problem/make\_public/?$", MakeContestProblemPublicAPIView.as\_view(), name="make\_public\_api"),

url(r"^contest/add\_problem\_from\_public/?$", AddContestProblemAPI.as\_view(), name="add\_contest\_problem\_from\_public\_api"),

url(r"^export\_problem/?$", ExportProblemAPI.as\_view(), name="export\_problem\_api"),

url(r"^import\_problem/?$", ImportProblemAPI.as\_view(), name="import\_problem\_api"),

url(r"^import\_fps/?$", FPSProblemImport.as\_view(), name="fps\_problem\_api"),

]

* 1. OJ：urlpatterns = [

url(r"^problem/tags/?$", ProblemTagAPI.as\_view(), name="problem\_tag\_list\_api"),

url(r"^problem/?$", ProblemAPI.as\_view(), name="problem\_api"),

url(r"^pickone/?$", PickOneAPI.as\_view(), name="pick\_one\_api"),

url(r"^contest/problem/?$", ContestProblemAPI.as\_view(), name="contest\_problem\_api"),

]

1. OJ：urlpatterns = [

url(r"^api/", include("account.urls.oj")),

url(r"^api/admin/", include("account.urls.admin")),

url(r"^api/", include("announcement.urls.oj")),

url(r"^api/admin/", include("announcement.urls.admin")),

url(r"^api/", include("conf.urls.oj")),

url(r"^api/admin/", include("conf.urls.admin")),

url(r"^api/", include("problem.urls.oj")),

url(r"^api/admin/", include("problem.urls.admin")),

url(r"^api/", include("contest.urls.oj")),

url(r"^api/admin/", include("contest.urls.admin")),

url(r"^api/", include("submission.urls.oj")),

url(r"^api/admin/", include("submission.urls.admin")),

url(r"^api/admin/", include("utils.urls")),

]

1. Contest：
   1. Admin：urlpatterns = [

url(r"^contest/?$", ContestAPI.as\_view(), name="contest\_admin\_api"),

url(r"^contest/announcement/?$", ContestAnnouncementAPI.as\_view(), name="contest\_announcement\_admin\_api"),

url(r"^contest/acm\_helper/?$", ACMContestHelper.as\_view(), name="acm\_contest\_helper"),

url(r"^download\_submissions/?$", DownloadContestSubmissions.as\_view(), name="acm\_contest\_helper"),

]

* 1. OJ：urlpatterns = [

url(r"^contests/?$", ContestListAPI.as\_view(), name="contest\_list\_api"),

url(r"^contest/?$", ContestAPI.as\_view(), name="contest\_api"),

url(r"^contest/password/?$", ContestPasswordVerifyAPI.as\_view(), name="contest\_password\_api"),

url(r"^contest/announcement/?$", ContestAnnouncementListAPI.as\_view(), name="contest\_announcement\_api"),

url(r"^contest/access/?$", ContestAccessAPI.as\_view(), name="contest\_access\_api"),

url(r"^contest\_rank/?$", ContestRankAPI.as\_view(), name="contest\_rank\_api"),

]

1. Account：
   1. Admin：urlpatterns = [

url(r"^user/?$", UserAdminAPI.as\_view(), name="user\_admin\_api"),

url(r"^generate\_user/?$", GenerateUserAPI.as\_view(), name="generate\_user\_api"),

]

* 1. OJ：urlpatterns = [

url(r"^login/?$", UserLoginAPI.as\_view(), name="user\_login\_api"),

url(r"^logout/?$", UserLogoutAPI.as\_view(), name="user\_logout\_api"),

url(r"^register/?$", UserRegisterAPI.as\_view(), name="user\_register\_api"),

url(r"^change\_password/?$", UserChangePasswordAPI.as\_view(), name="user\_change\_password\_api"),

url(r"^change\_email/?$", UserChangeEmailAPI.as\_view(), name="user\_change\_email\_api"),

url(r"^apply\_reset\_password/?$", ApplyResetPasswordAPI.as\_view(), name="apply\_reset\_password\_api"),

url(r"^reset\_password/?$", ResetPasswordAPI.as\_view(), name="reset\_password\_api"),

url(r"^captcha/?$", CaptchaAPIView.as\_view(), name="show\_captcha"),

url(r"^check\_username\_or\_email", UsernameOrEmailCheck.as\_view(), name="check\_username\_or\_email"),

url(r"^profile/?$", UserProfileAPI.as\_view(), name="user\_profile\_api"),

url(r"^profile/fresh\_display\_id", ProfileProblemDisplayIDRefreshAPI.as\_view(), name="display\_id\_fresh"),

url(r"^upload\_avatar/?$", AvatarUploadAPI.as\_view(), name="avatar\_upload\_api"),

url(r"^tfa\_required/?$", CheckTFARequiredAPI.as\_view(), name="tfa\_required\_check"),

url(r"^two\_factor\_auth/?$", TwoFactorAuthAPI.as\_view(), name="two\_factor\_auth\_api"),

url(r"^user\_rank/?$", UserRankAPI.as\_view(), name="user\_rank\_api"),

url(r"^sessions/?$", SessionManagementAPI.as\_view(), name="session\_management\_api"),

url(r"^open\_api\_appkey/?$", OpenAPIAppkeyAPI.as\_view(), name="open\_api\_appkey\_api"),

url(r"^sso?$", SSOAPI.as\_view(), name="sso\_api")

]