南開大學
Nankai University

南　　开　　大　　学

计 算 机 学 院

招聘网站项目文档

王　浩　2013287

王照钦　1911484

王昕炜　2113426

张麟浩　2113976

李铭辉　2011266

指导老师：李起成

2024 年 6 月 30 日

目录

一、 小组个人信息 **1**

二、 项目概况 **1**
（一） 任务目标 . . . 1
（二） 用户特点 . . . 1

三、 假定与约束 **2**

四、 需求分析 **3**
（一） 业务描述 . . . 3
1. 系统总业务流程图及其描述 . . . 3
2. 各个子业务流程图及其描述 . . . 4
（二） 数据需求 . . . 9
1. 数据需求描述 . . . 9
2. 数据流图 . . . 9
3. 数据字典 . . . 11
（三） 功能需求 . . . 12
（四） 性能/非功能需求 . . . 13

五、 系统设计 **14**
（一） 用例图 . . . 14
（二） 活动图 . . . 14
（三） 类图 . . . 17
（四） 时序图 . . . 18
（五） 状态图 . . . 18
（六） 部署图 . . . 19

六、 UI 设计 **20**

七、 系统测试 **21**
（一） 后端测试 . . . 21

八、 项目实现流程 **25**

九、 用户手册 **26**
（一） 引言 . . . 26
1. 编写目的 . . . 26
2. 背景 . . . 26
3. 定义 . . . 26
4. 参考资料 . . . 26
（二） 用途 . . . 26
1. 功能 . . . 26
2. 性能 . . . 28
3. 安全保密性 . . . 30
4. 运行环境 . . . 30

5. 使用过程 31

十、 项目部署环境具体参数 **31**

（一） 项目语言 31

（二） 项目框架 31

（三） 引用包的具体版本 32

（四） 项目安装部署流程 33

十一、 项目 GitHub 地址 **33**

十二、 项目功能截图 **33**

十三、 代码功能介绍 **33**

1. 用户管理 34

2. 工作管理 37

3. 简历和消息管理 40

4. oj 管理 42

十四、 总结 **43**

一、 小组个人信息

组长:王浩 (2013287) 组员:王照钦 (1911484) 组员:王昕炜 (2113426) 组员:张麟浩 (2113976) 组员：李铭辉 (2011266)

备注：均为 0930 李起成老师班级。

二、 项目概况

(一) 任务目标

本招聘网站项目的主要目标包括:

1. **用户注册与登录**：实现用户（求职者和招聘方）注册、登录、以及账户管理功能，包括用户信息的增删改查。
2. **职位发布与管理**：招聘方可以发布和管理职位信息，包括职位的分类、描述、要求等，并能够编辑和删除职位。
3. **求职功能**：求职者可以浏览、搜索并申请职位，并能够查看申请状态和历史记录。
4. **简历管理**：求职者可以创建、编辑简历。
5. **通知与消息**：系统提供通知功能，向用户推送招聘进展、面试安排等消息，同时提供站内消息功能，方便用户之间的交流。
6. **推荐系统**：实现职位推荐和简历推荐功能，帮助招聘方和求职者更快匹配合适的工作和人才。
7. **安全性与隐私保护**：保证用户数据的安全性和隐私保护，采用加密算法进行数据存储，防止数据泄露和滥用。
8. **用户体验**：提供良好的用户界面和用户体验，使用户能够方便快捷地完成所有操作。

(二) 用户特点

本系统的最终用户主要分为三类：求职者、招聘方和管理员。

求职者
- 注册并创建个人账号。
- 浏览和搜索职位信息。
- 投递简历和查看申请进度。
- 接收面试通知和系统推荐的职位信息。

招聘方
- 注册并创建公司账号。
- 发布和管理招聘职位。
- 浏览和筛选求职者简历。
- 通知求职者面试安排和录用信息。

管理员
- 系统的整体管理与维护。
- 管理用户账号，包括求职者和招聘方的注册申请审核、账号启用和禁用。

- 监控系统运行状态，确保系统稳定和安全。
- 审核和管理招聘信息，确保职位信息的合法性和真实性。
- 处理用户反馈和投诉，提供技术支持。
- 生成系统报告和统计数据，提供给决策层参考。

三、 假定与约束

1. 系统运行的最小寿命：在无重大改动的情况下运行 3 年。
2. 开发期限：120 天。
3. 假设相关硬件设备齐全。
4. 假设系统相关功能达到预期要求。
5. 系统必须遵守数据保护和隐私法规，支持高并发访问，确保在用户高峰期依然能够稳定运行。
6. 系统需要满足网络安全和信息安全的相关规定和要求，包括但不限于防止黑客攻击、数据泄露等方面的规定。

四、 需求分析

(一) 业务描述

1. 系统总业务流程图及其描述

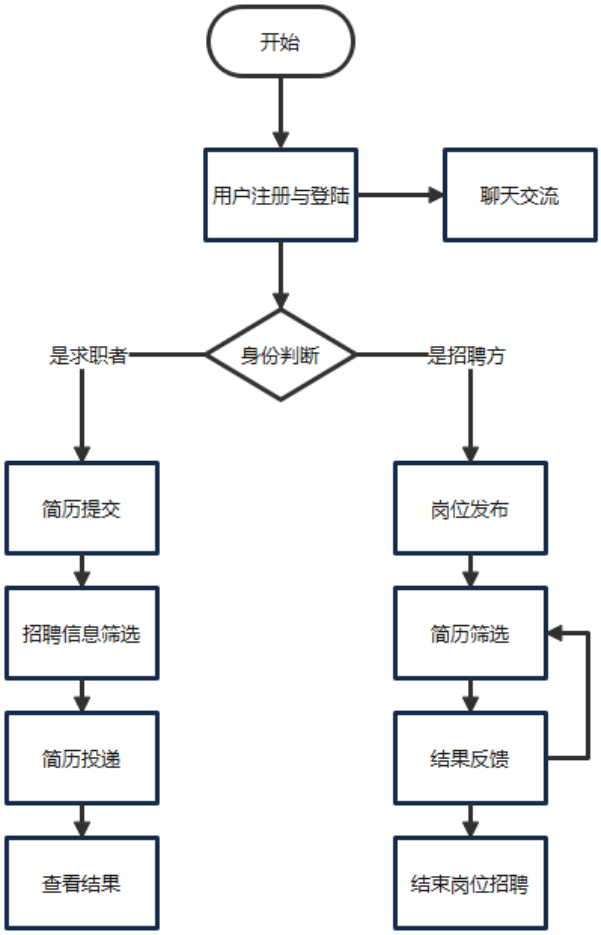

图 1: 总业务流程图

总业务流程图1概述了招聘平台的核心操作流程。用户首先在平台上注册并登录，随后根据其身份（求职者或招聘方）进入不同的操作路径。求职者可以提交简历并根据个人条件筛选并申请职位，而招聘方则负责发布岗位需求并对收到的简历进行筛选。双方通过平台进行互动，求职者投递简历后，招聘方会根据岗位要求进行评估并给出反馈。这个过程不断进行，直到招聘方找到合适的候选人并结束岗位的招聘。此外，无论用户是求职者还是招聘方，都可以在聊天系统与他人交流、分享职位信息。

2. 各个子业务流程图及其描述

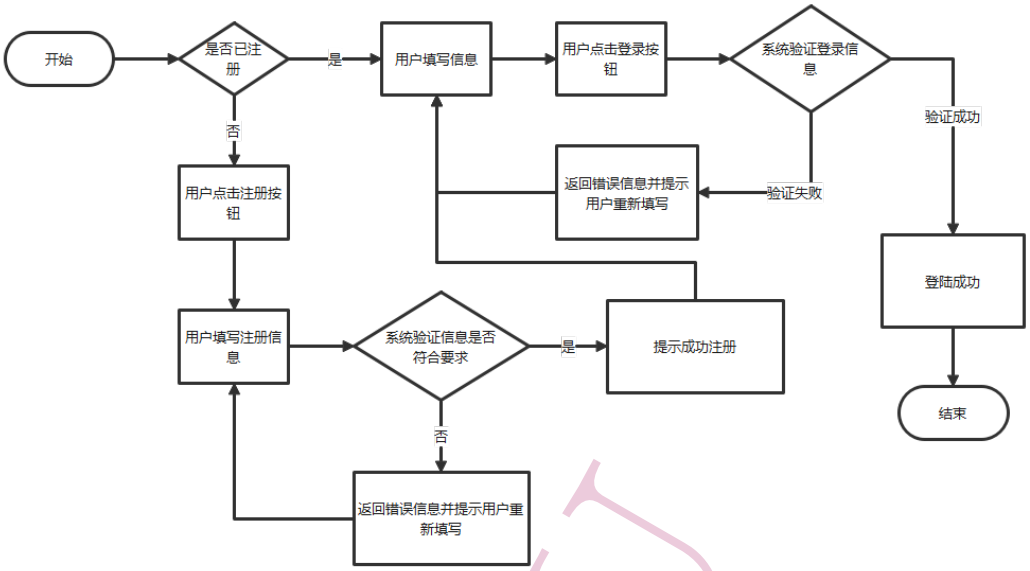

图 2: 注册登录子流程图

（1）注册登录

注册登录子流程图2展示了用户在进行登录和注册操作时的系统响应流程。当用户访问系统时，首先需要自行判断是否已经注册。如果已注册，用户将点击登录按钮并输入个人信息，系统随后验证这些信息。如果登录信息正确，用户将成功登录；如果不正确，系统会提示错误并要求用户重新填写。对于未注册的用户，他们将选择注册按钮并填写注册信息。系统将检查这些信息是否符合注册要求，如果符合，用户将被告知注册成功；如果不符合，系统同样会返回错误信息并要求用户重新填写。

（2）求职者

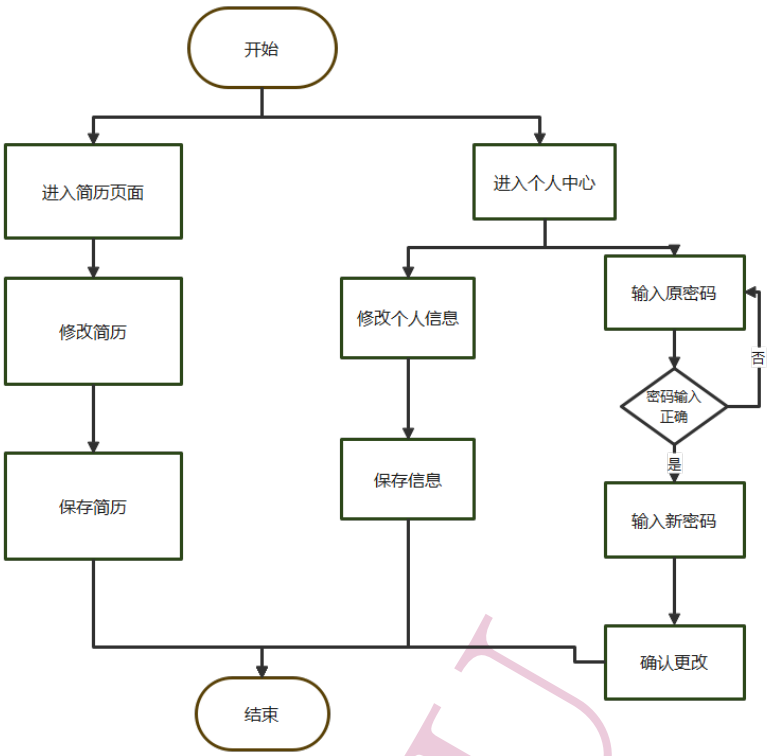

图 3: 求职者个人信息管理

个人信息管理与简历编辑 在求职者登陆后，可以进入个人中心，修改姓名、性别等个人信息。也可以进入修改密码界面，输入原密码确认成功后，可以输入新密码并确认密码，确认更改后保存新密码。此外，还可以访问简历编辑页面，编辑个人简历并保存。

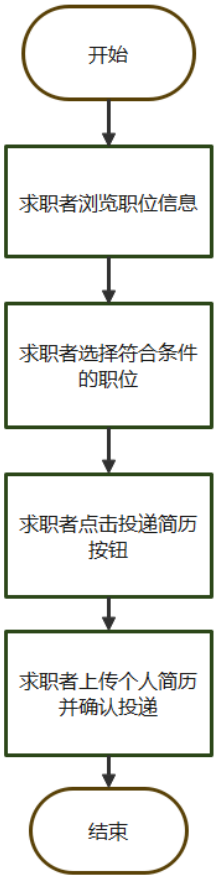

图 4: 简历投递子流程

投递简历 求职者首先浏览各种职位信息，寻找与自己资质和兴趣相匹配的岗位。一旦找到合适的职位，他们会点击投递简历的按钮，随后确认投递自己的个人简历。一旦投递完成，流程即告结束，求职者的简历便会被发送至招聘方进行审核。

（3）招聘方

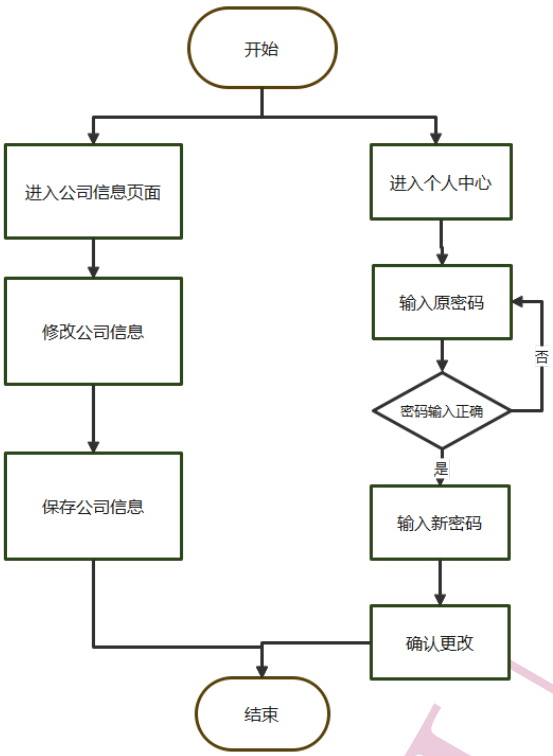

图 5: 招聘方个人信息管理

个人信息管理 在招聘方登陆后，可以进入个人中心，修改密码，输入原密码确认成功后，可以输入新密码并确认密码，确认更改后保存新密码。此外，还可以访问公司信息编辑页面，编辑公司信息并保存。

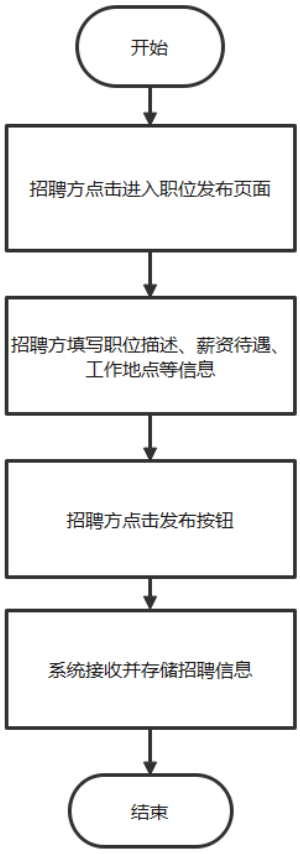

图 6: 招聘信息发布子流程

职位发布 招聘方首先进入职位发布页面，然后详细填写包括职位描述、薪资待遇和工作地点在内的关键信息。点击发布按钮后，系统便会自动接收并存储这些信息。

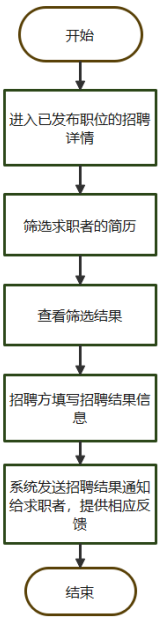

图 7: 招聘结果反馈子流程

招聘结果反馈 招聘方选择要查看详情的已发布职位，然后查看求职者的简历，然后填写相应的招聘结果反馈信息，例如录取、拒绝、进一步面试安排等。系统随后自动发送这些信息给求职者，并提供反馈，确保求职者及时了解申请结果。

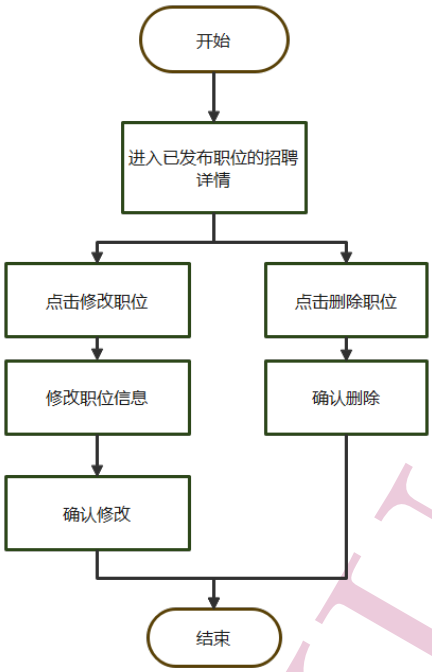

图 8: 已发布职位管理

管理已发布职位 招聘方首先进入系统查看已发布的职位列表，然后选择需要操作的职位。如果需要修改职位信息，点击修改按钮，进入编辑页面进行编辑，并在完成后确认修改。如果招聘方决定删除某个职位，则点击删除按钮，系统会弹出确认对话框以避免误操作。一旦招聘方确认删除或修改，流程结束，职位信息将相应地更新或从系统中移除。

(二) 数据需求

1. 数据需求描述

1. 用户（包括求职者、招聘方）信息，用于满足用户浏览和修改个人信息的需求。
2. 职位信息，用于用户查看职位。
3. 求职信息，用于满足求职者投递简历、招聘方查看求职者简历的需求。
4. 聊天记录信息，用于记录用户间交流信息。

2. 数据流图

用户信息数据流图如下:

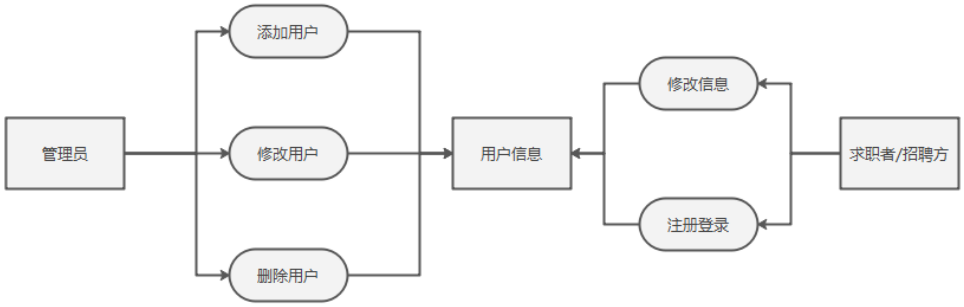

图 9: 用户信息数据流图

职位信息数据流图如下:

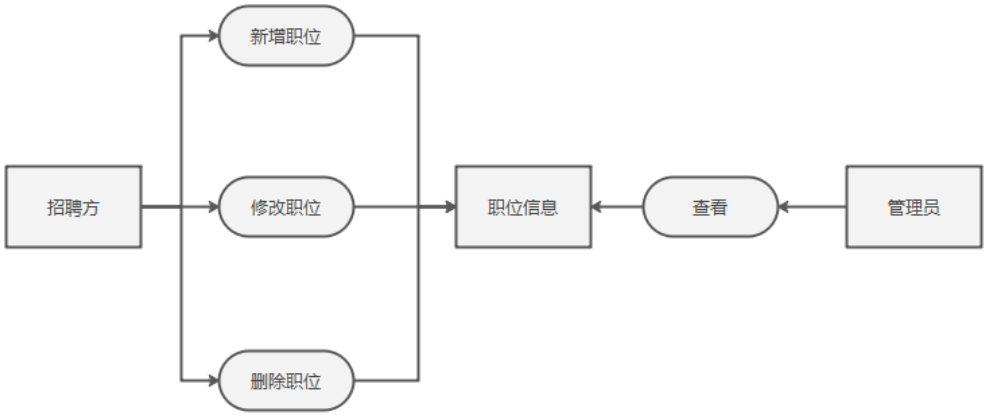

图 10: 职位信息数据流图

求职信息数据流图如下:

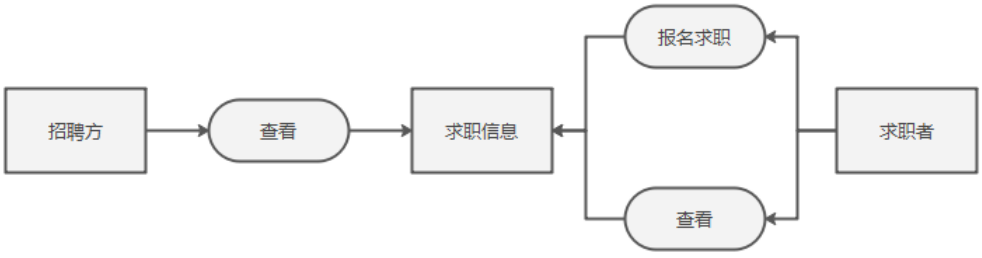

图 11: 求职信息数据流图

聊天记录信息数据流图如下:

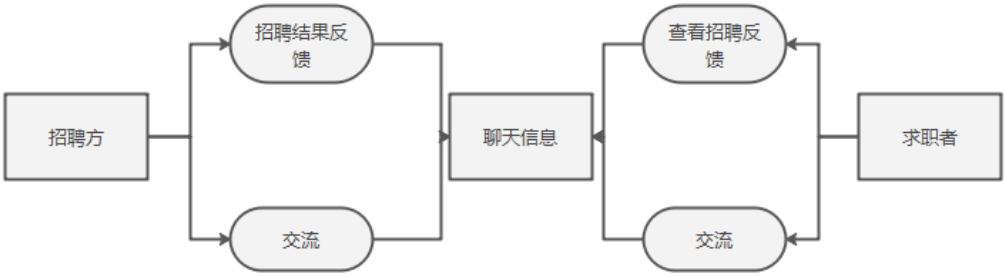

图 12: 聊天记录信息数据流图

3. 数据字典

(1) 求职者信息表

字段名	描述	数据类型和约束
求职者 ID	唯一标识符，自增长	-
求职者姓名	字符串，最大长度为 50	-
性别	字符串，最大长度为 10	-
用户	与 django 自带用户模型一一对应	

(2) 招聘方信息表

字段名	描述	数据类型和约束
招聘方 ID	唯一标识符，自增长	-
招聘方姓名	字符串，最大长度为 50	-
性别	字符串，最大长度为 10	-
部门	字符串，最大长度为 50	
公司	字符串，最大长度为 50	外键
用户	与 django 自带用户模型一一对应	

(3) 公司信息表

字段名	描述	数据类型和约束
公司 ID	唯一标识符，自增长	-
公司名称	字符串，最大长度为 50	-
行业	字符串，最大长度为 10	-
联系方式	字符串，最大长度为...	-

(4) 管理员信息表

字段名	描述	数据类型和约束
管理员 ID	唯一标识符，自增长	-
管理员姓名	字符串，最大长度为 50	-
性别	字符串，最大长度为 10	-
联系方式	字符串，最大长度为...	-

(5) 职位信息表

字段名	描述	数据类型和约束
职位 ID	唯一标识符，自增长	-
职位名称	字符串，最大长度为 50	
职位描述	字符串，最大长度为 50	
职位要求	字符串，最大长度为 50	
所属公司	外键，标志公司	
工作地点	字符串，最大长度为 50	
薪资	整数	
发布日期	时间	

(6) 求职信息表

字段名	描述	数据类型和约束
报名 ID	唯一标识符，自增长	-
求职者 ID	外键，关联求职者信息表中的求职者 ID	-
职位 ID	外键，关联职位信息表中的职位 ID	-
申请日期	时间	
申请状态	分为已申请，面试中，已录用，已拒绝	

(7) 聊天信息记录表

字段名	描述	数据类型和约束
消息 ID	唯一标识符，自增长	-
发送者 ID	外键，关联发送者 ID	-
接受者 ID	外键，关联接收者 ID	-
消息内容	text	
发送日期	时间	
接收状态	分为已读和未读	

(8) 个人简历表

字段名	描述	数据类型和约束
简历 ID	唯一标识符，自增长	-
求职者 ID	外键，关联求职者 ID	-
教育经历	字符串，最大长度为 50	-
工作经历	字符串，最大长度为 50	-
技能	字符串，最大长度为 50	-
项目经历	字符串，最大长度为 50	-
认证和证书	字符串，最大长度为 50	-

(三) 功能需求

功能划分为以下四个部分:

1. **用户管理模块**
   - **用户注册和登录：**求职者、招聘方和管理员可以通过系统注册账号，并填写个人信息。注册完成后可通过账号和密码登录系统。
   - **个人信息管理：**用户可以查看和修改自己的个人信息，包括姓名、联系方式等。
   - **用户权限管理：**管理员可以设置不同用户的权限，确保各类用户只能访问和操作相应的功能。
2. **求职者功能模块**
   - **职位浏览和搜索：**求职者可以浏览和搜索招聘信息，按职位类别、公司、地点等条件筛选职位。
   - **简历管理：**求职者可以在线创建、编辑和管理简历，上传附件等。
   - **职位申请：**求职者可以投递简历至感兴趣的职位，并查看申请状态。
   - **职位推荐：**系统根据求职者的简历和求职意向，推荐适合的职位信息。
3. **招聘方功能模块**
   - **职位发布和管理：**招聘方可以发布招聘信息，编辑职位描述，设定招聘要求等。
   - **简历筛选和管理：**招聘方可以浏览和筛选求职者的简历，标记关注简历并管理面试安排。
   - **面试通知和反馈：**招聘方可以发送面试通知，记录面试反馈，并通知求职者录用结果。
4. **管理员功能模块**
   - **用户账号管理：**管理员可以审核求职者和招聘方的注册申请，启用或禁用账号。
   - **系统监控和维护：**管理员可以监控系统运行状态，进行维护和更新，确保系统稳定和安全。
   - **内容审核：**管理员审核职位信息，确保招聘信息的合法性和真实性。
   - **数据统计和报告：**管理员生成系统使用报告和统计数据，为决策提供参考。

（四） 性能/非功能需求

1. **准确性**：系统应能够支持大量用户同时在线，且响应时间不超过 3 秒。它应能迅速处理聊天消息传递、报名结果反馈，同时保持 24 小时的高稳定性运行。
2. **及时性**：系统应迅速响应用户操作，大多数操作的响应时间应保持在 3 秒以内。
3. **可用性**：系统应易于理解并操作，具有友好的用户界面。它应能自适应不同设备和屏幕尺寸，并保证在各种设备上正常运行。此外，系统应具有高可用性和可靠性，防止系统崩溃或数据丢失。
4. **安全性**：用户密码在输入时应显示为加密状态，并使用 pbkdf2_sha256 算法进行加密传输和存储，以确保数据的保密性和完整性。系统应具备用户身份验证和授权机制，防止未授权访问。用户职责应明确区分，以防止越权操作，确保数据安全。系统还应具备防护恶意攻击和数据泄露的能力，以维护系统安全。

5. **可靠性**：系统应具备数据备份和恢复功能，以防数据丢失。应有故障自动检测和恢复机制，确保系统在故障时能迅速恢复。同时，系统应有日志记录和审计功能，以追踪和监控系统操作。
6. **易维护性**：系统应具备可扩展性，方便添加新功能和模块。应有可配置性，便于进行系统配置和参数调整。代码结构和文档应易于维护，以便于系统升级和维护。
7. **标准性**：软件产品应严格依据软件标准进行测试，以确保产品符合质量标准。
8. **先进性**：系统设计应采用前沿的架构和技术，以满足当前和未来一段时间的系统需求。

五、 系统设计

(一) 用例图

用例图如下图所示

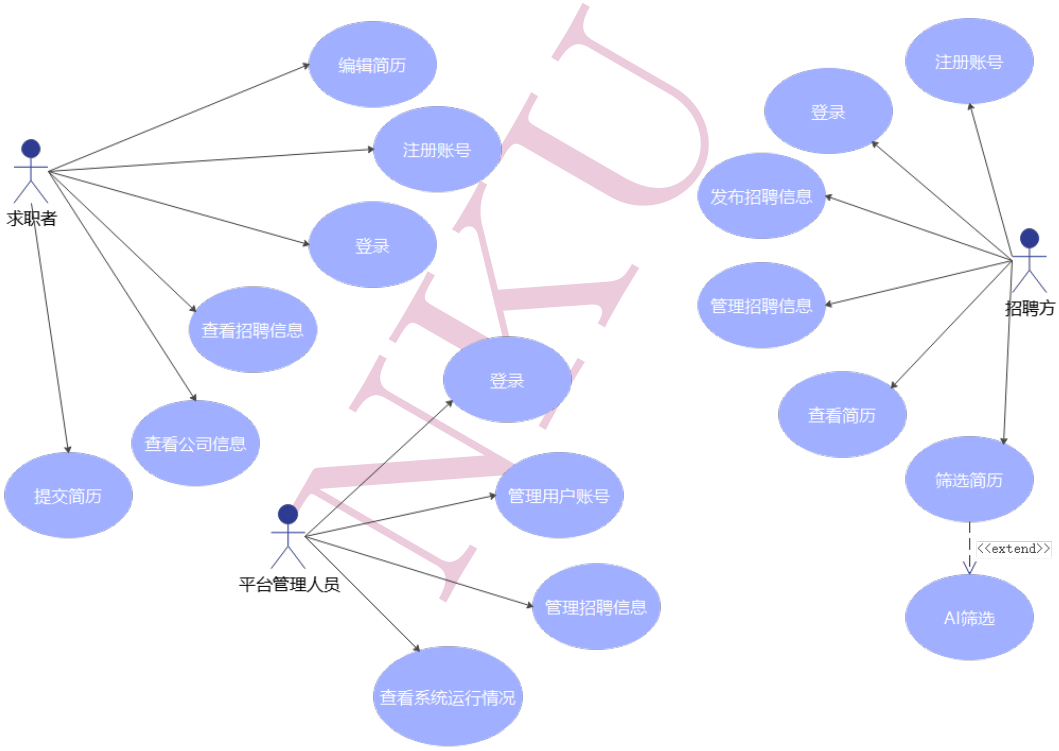

图 13: 用例图

(二) 活动图

这一部分，我们用活动图描述两类核心操作：招聘方的招聘操作与求职者的求职操作。其中，求职者的求职操作包括：求职者填写简历；求职者查看招聘信息；求职者投递简历。

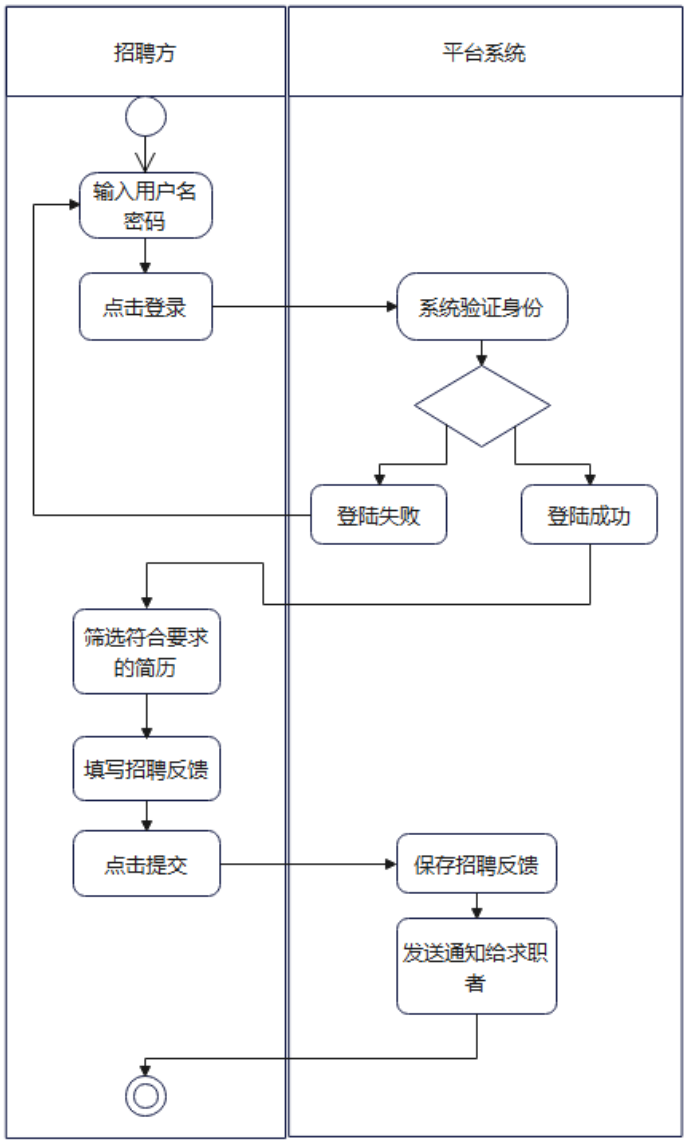

图 14: 招聘方招聘活动图

通过活动图可以得到招聘方招聘的过程如下:

1. **招聘方登录：**首先，招聘方需要在平台系统上进行登录操作。
   - 输入用户名和密码。
   - 点击登录后，系统会进行身份验证。
2. **系统验证身份：**根据招聘方提供的用户名和密码，系统会进行身份验证。
   - 如果验证失败，则会显示“登陆失败”。
   - 如果验证成功，则会显示“登陆成功”。
3. **筛选简历：**一旦登录成功，招聘方可以进行简历的筛选工作。
   - 招聘方需要筛选出符合职位要求的简历。
4. **填写招聘反馈：**筛选简历后，招聘方需要填写对候选人的招聘反馈。

- 这可能包括对候选人的评价、面试结果等信息。

5. **提交和保存招聘反馈：**填写完招聘反馈后，招聘方需要点击提交。
   - 提交后，系统会保存这些招聘反馈。
6. **发送通知给求职者：**最后，系统会根据招聘方的反馈，发送相应的通知给求职者。
   - 这可能包括面试结果、录用通知或其他相关信息。

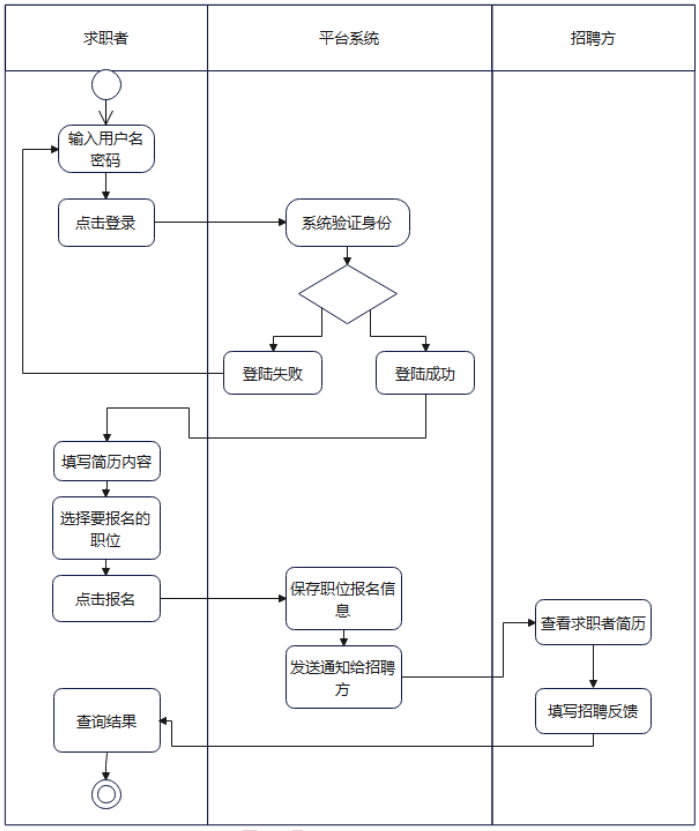

图 15: 求职者求职活动图

通过活动图可以得到求职者求职的过程如下：

1. **求职者登录：**
   - 首先，求职者需要输入用户名和密码。
   - 接着，求职者点击登录按钮。
2. **平台系统验证：**
   - 系统会验证求职者的身份。
   - 如果身份验证失败，求职者将无法登录。
   - 如果身份验证成功，求职者将登陆成功。
3. 登陆成功后，**求职者操作：**
   - 填写简历内容。

- 选择想要报名的职位。
- 点击报名按钮。

4. **平台系统响应：**
   - 保存职位报名信息。
   - 发送通知给招聘方。
5. **招聘方反馈：**
   - 招聘方可以查询求职者的报名信息和简历。
   - 招聘方填写招聘反馈。

（三） 类图

(此小节存在过时内容，待进一步交流后更新！)
根据需求，绘制类图如下所示

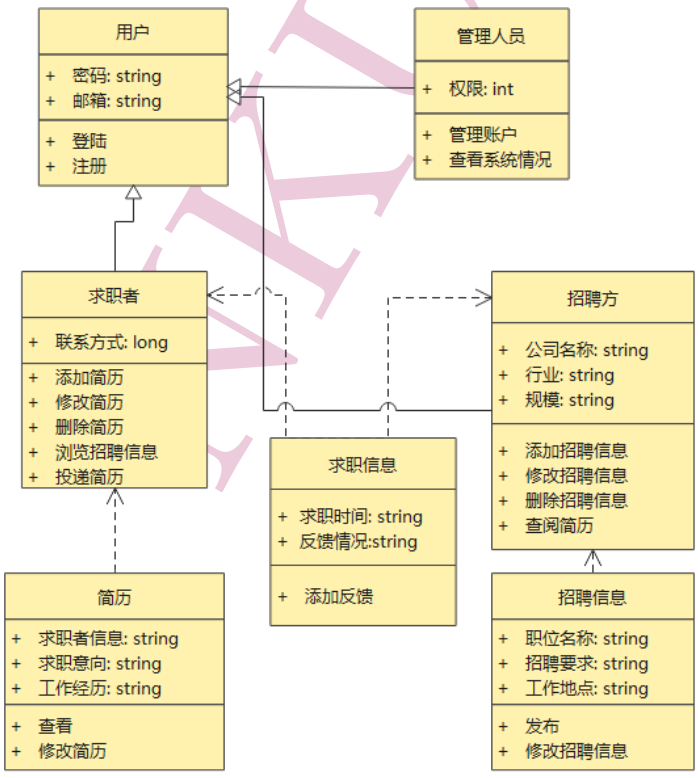

图 16: 类图

（四） 时序图

我们为招聘方的职位管理功能绘制时序图。这张时序图描述了招聘方用户在使用招聘系统时，进行职位管理的一系列操作流程：

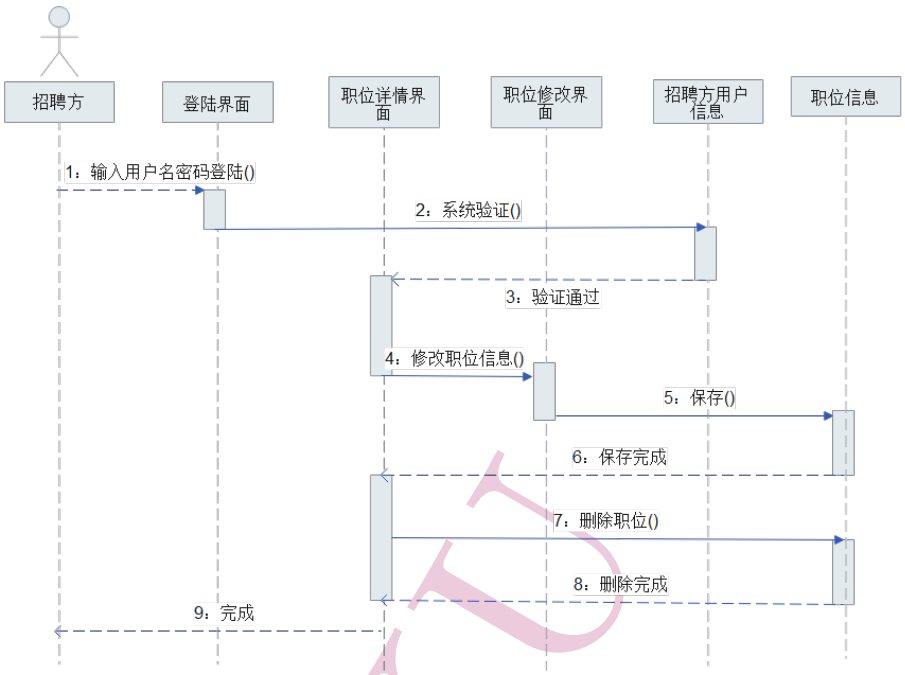

图 17: 招聘方职位管理操作流程时序图

以下是时序图中描述的操作步骤：

1. **登录系统：**招聘方首先登录系统。
2. **系统验证：**系统对招聘方登录信息进行校验。
3. **职位管理：**招聘方进入职位管理界面，进行职位信息的查看、修改或删除。
4. **保存修改：**招聘方修改职位信息后，点击保存。
5. **完成操作：**系统显示操作完成提示，职位信息更新或职位删除成功。

（五） 状态图

由于招聘、求职过程都围绕着职位展开，因此我们为职位绘制状态图，它描述了一个职位从创建到删除的整个生命周期，如下所示：

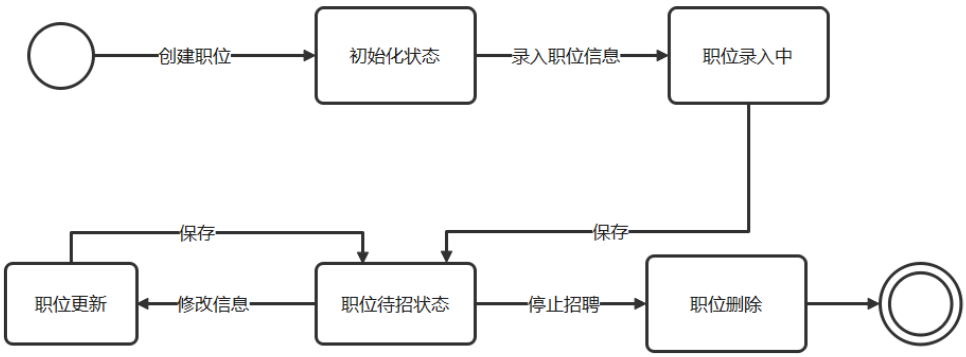

图 18: 职位生命周期状态图

以下是状态图中描述的职位生命周期的主要步骤:

1. **创建职位:** 这是职位生命周期的起始点，表示一个新职位的创建过程开始。
2. **初始化状态:** 在创建职位之后，系统会进入一个初始化状态，可能用于设置一些基本的参数或默认值。
3. **录入职位信息:** 在初始化之后，需要录入具体的职位信息，如职位名称、要求、薪资范围等。
4. **职位录入中:** 表示正在录入职位信息的过程。
5. **保存:** 录入完职位信息后，可以保存这些信息，使之成为正式的职位记录。
6. **职位更新:** 当需要对职位信息进行更改时，可以进入职位更新状态，允许修改已有的职位信息。
7. **修改信息:** 在职位更新状态下，可以修改职位的详细信息。
8. **职位待招状态:** 表示职位已经准备好，正在等待合适的候选人申请。
9. **删除职位:** 如果职位不再需要招聘或已经招到合适的人选，可以进入删除职位状态，准备将职位从系统中移除。

(六) 部署图

部署图详细展示了系统在物理层面的运行架构，包括硬件的布局以及软件如何在硬件上进行部署。

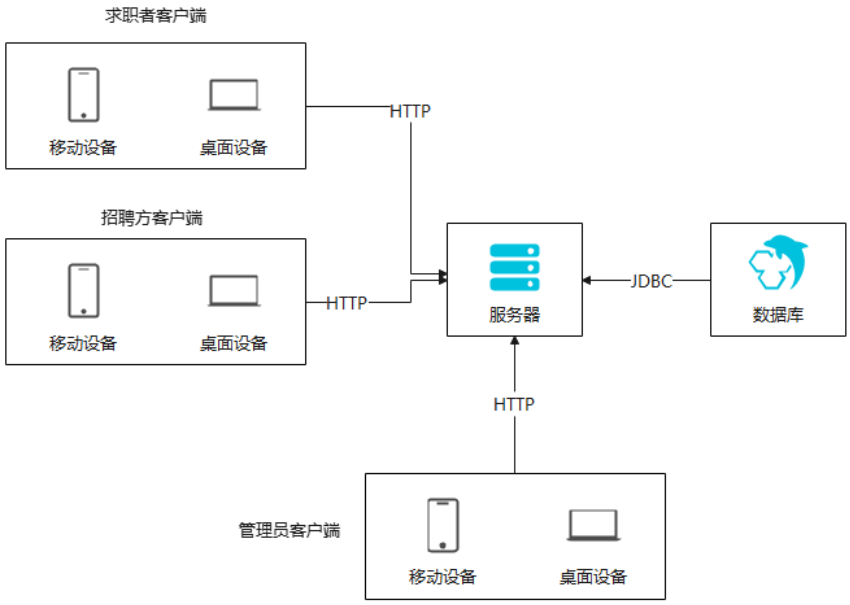

图 19: 部署图

本系统采用集中式架构设计，从图中可以清晰地识别出硬件的配置和软件部署的详细流程: - 求职者、招聘方和管理员端均能通过网页浏览器，利用 HTTP 协议通过互联网访问到 Web 服务器；- Web 服务器通过 JDBC 接口与数据库进行交互，实现数据的存储和检索；同时，通过输入输出操作与 IO 设备进行数据交换。

六、 UI 设计

在交互设计中，需要实现页面之间跳转的设计。在访问网站的时候，首先访问登录注册页面。用户根据自己的角色选择对应的登陆注册界面进行注册、登录，并根据自己登录的角色跳转到对应的首页。求职者身份会跳转到求职者首页，在求职者首页包含职位市场、个人简历、聊天消息等页面入口，在页面内部会包含相应的具体功能。招聘方身份会跳转到招聘方首页，在招聘方首页包含职位发布、公司信息、聊天消息的页面入口，在页面内部会包含相应的具体功能。管理员身份会跳转到管理员首页，管理员包含人员管理（管理求职者和招聘方）、内容审核管理功能页面用于管理职位内容。

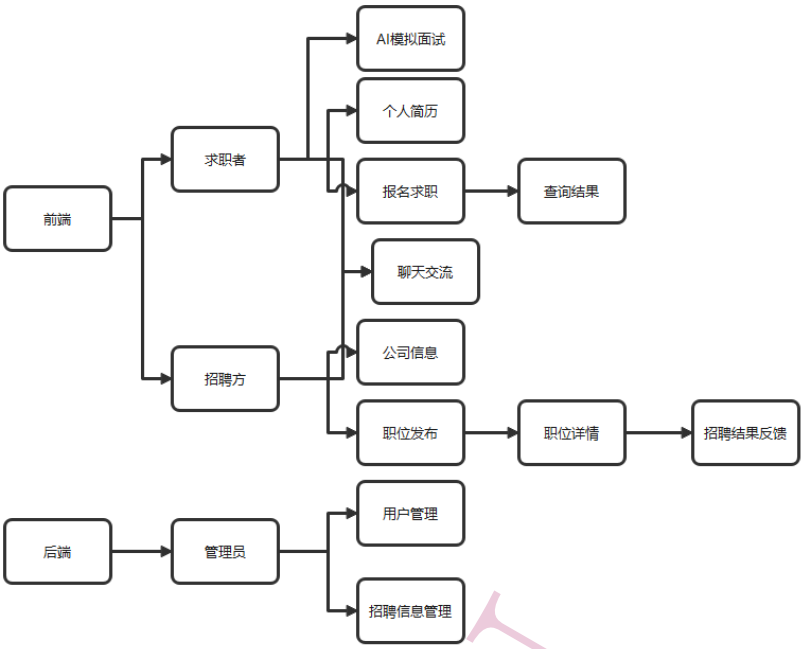

图 20: UI 交互设计

七、 系统测试

(一) 后端测试

后端使用 Django 框架，分别对 job、resume、question、user 等部分进行了测试，每个部分主要对 view 视图、model 数据库模型以及 xadmin 管理员相关的文件进行测试。

1. view 视图测试

针对视图相关的测试项目主要包括类中参数的参数、视图的状态和函数的功能。对于部分函数，在测试函数功能时还要考虑 request 进行传参的过程，我们使用 Mock 方式来模拟这一过程。以 user 的 JobseekerViewSetTest 类的测试为例，参数测试过程如下:

```
    class JobseekerViewSetTest(TestCase):
    @classmethod
    def setUpTestData(cls):
        # 创建求职者用于测试
        Jobseeker.objects.create(name='Test Jobseeker', user=User.objects.
            create_user(username='jobseeker', password='test_password'))
    def test_get_jobseekers(self):
        # 测试获取所有求职者信息
        client = APIClient()
        client.login(username='jobseeker', password='test_password')
        response = client.get(reverse('jobseeker-list'))
        self.assertEqual(response.status_code, 200)
class HRViewSetTest(TestCase):
```

```
    @classmethod
    def setUpTestData(cls):
        # 创建HR用于测试
        company = Company.objects.create(name='Test Company', industry='IT',
            location='Test Location')
        HR.objects.create(name='Test HR', user=User.objects.create_user(
            username='hr', password='test_password'), company=company)
    def test_get_hrs(self):
        # 测试获取所有HR信息
        client = APIClient()
        client.login(username='hr', password='test_password')
        response = client.get(reverse('hr-list'))
        self.assertEqual(response.status_code, 200)
```

在这里，我们主要对对应视图类中设置的参数进行了测试，检测参数的设置是否符合预期。以 user 的 CustomBackend 类的测试为例，函数功能测试过程如下:

```
    class CustomBackendTest(TestCase):
    @classmethod
    def setUpTestData(cls):
        # 创建用户用于测试
        User.objects.create_user(username='test_user', password='
            test_password')
    def test_authenticate(self):
        # 测试用户认证功能
        client = APIClient()
        response = client.post(reverse('login'), {'username': 'test_user', '
            password': 'test_password'}, format='json')
        self.assertEqual(response.status_code, 200)
```

在上述代码中，我们主要对 CustomBackend 类中的自定义用户验证的函数功能进行了测试，测试了用户输入的密码正确，返回用户信息和用户输入密码错误，返回 None 两种情况。由于采用前后端分离的模式进行项目构建，后端的 view 相关的函数主要采用获取 request 的方式来获取参数。在测试过程中，我们使用 Mock 方式来模拟 request 传参过程。

```
class MockRequest():
    def __init__(self, data):
        self.data=data
class RegisterViewSetTest(TestCase):
    @classmethod
    def setUpTestData(cls):
        #Create 13 authors for pagination tests
        number_of_users = 13
        for user_num in range(number_of_users):
            User.objects.create(username='user%s' % user_num, password = '
                654321%s' % user_num,)
        Clazz.objects.create(year='2020', major='计算机', clazz='1')
        Student.objects.create(name='syq', user=User.objects.get(id=1),
            gender='f', clazz=Clazz.objects.get(id=1))
```

```
    def test_create_user_fail(self):
        view = RegisterViewSet()
        data = {'username': 'user2', 'name': 'abc'}
        url = "localhost:8000/register"
        mockrequest = MockRequest(data)
        requests.post = mock.Mock(return_value=mockrequest)
        res = requests.post(url, data=data)
        result = view.create(res)
        self.assertEqual(result.status_code, 201)
    def test_create_user(self):
        view = RegisterViewSet()
        data = {'username': 'user14', 'name': 'abc', 'password': 123456}
        url = "localhost:8000/register"
        mockrequest = MockRequest(data)
        requests.post = mock.Mock(return_value=mockrequest)
        res = requests.post(url, data=data)
        result = view.create(res)
        #user_detail = UserDetailSerializer(data=request.data)
        self.assertEqual(result.status_code, 200)
```

如上述代码所示，我们对用户注册部分进行了测试。针对类中 create 函数需要的 request 参数,我们在测试中自定义了 MockRequest 类并使用了 Mock 方法对正常了 request 数据获取方法进行了模拟，通过返回信息的网页状态 status-code 对结果进行判断并测试了用户注册成功与用户注册失败两种分支结果。

2. model 数据库模型测试

model 相关的测试主要测试了数据库每个字段的设置是否与预想相同，例如字段是否存在、字段长度设置是否符合预期、数据储存顺序的设置是否有效等。以 user 部分 Company 类的数据库模型测试为例，model 的基础部分的测试过程如下所示:

```
class CompanyModelTest(TestCase):
"""公司模型测试"""
@classmethod
def setUpTestData(cls):
    # Set up non-modified objects used by all test methods
    Company.objects.create(name='Tech Co.', industry='Technology',
        location='Beijing')
def test_name_label(self):
    company = Company.objects.get(id=1)
    field_label = company._meta.get_field('name').verbose_name
    self.assertEquals(field_label, "公司名称")
def test_industry_label(self):
    company = Company.objects.get(id=1)
    field_label = company._meta.get_field('industry').verbose_name
    self.assertEquals(field_label, "行业")
def test_location_label(self):
    company = Company.objects.get(id=1)
    field_label = company._meta.get_field('location').verbose_name
```

```
        self.assertEquals(field_label, "位置")
    def test_name_max_length(self):
        company = Company.objects.get(id=1)
        max_length = company._meta.get_field('name').max_length
        self.assertEquals(max_length, 100)
    def test_industry_max_length(self):
        company = Company.objects.get(id=1)
        max_length = company._meta.get_field('industry').max_length
        self.assertEquals(max_length, 100)
    def test_location_max_length(self):
        company = Company.objects.get(id=1)
        max_length = company._meta.get_field('location').max_length
        self.assertEquals(max_length, 100)
    def test_object_name(self):
        company = Company.objects.get(id=1)
        expected_object_name = company.name
        self.assertEquals(expected_object_name, str(company))
```

从上述代码中可以看出, Company 类中包含公司名称, 行业, 位置三个字段并且均设置了字段长度, 基础测试就分别测试这三个字段的名称和长度。测试部分中 setUpTestData() 函数是 Django 测试框架中专门用于设置当前测试类的数据的函数, 我们在该函数中声明了一个 Company 类的对象, 在其他函数中对它的公司名称, 行业, 位置三个字段的名称和长度进行了测试。如果当前测试的 model 存在字段设置以外的功能, 例如 model 的名称、使用 str() 方法返回字符串等, 我们也会对这些功能进行测试。

3. xadmin 管理员相关文件测试

```
    from django.test import TestCase
from user.adminx import CompanyAdmin, JobseekerAdmin, HRAdmin
from user.models import Company, Jobseeker, HR
import xadmin
from user.resource import JobseekerResource
class CompanyAdminTest(TestCase):
    def test_company_admin_config(self):
        company_admin = CompanyAdmin(Company, xadmin.site)
        self.assertEqual(company_admin.list_display, ['id', 'name', 'industry
            ', 'location'])
        self.assertEqual(company_admin.list_filter, ['industry', 'location'])
        self.assertEqual(company_admin.search_fields, ['id', 'name', '
            industry', 'location'])
        self.assertEqual(company_admin.list_display_links, ['name'])
        self.assertEqual(company_admin.list_per_page, 10)
        self.assertEqual(company_admin.model_icon, 'fa fa-building')
class JobseekerAdminTest(TestCase):
    def test_jobseeker_admin_config(self):
        jobseeker_admin = JobseekerAdmin(Jobseeker, xadmin.site)
        self.assertEqual(jobseeker_admin.list_display, ['id', 'name', 'user',
             'gender', 'resume', 'skills'])
        self.assertEqual(jobseeker_admin.list_filter, ['gender', 'skills'])
```

```
20          self.assertEqual(jobseeker_admin.search_fields, ['id', 'name', '
                skills'])
21          self.assertEqual(jobseeker_admin.list_display_links, ['name'])
22          self.assertEqual(jobseeker_admin.list_per_page, 10)
23          self.assertEqual(jobseeker_admin.model_icon, 'fa fa-user')
24          self.assertEqual(jobseeker_admin.relfield_style, 'fk-ajax')
25          self.assertEqual(jobseeker_admin.import_export_args, {'
                import_resource_class': JobseekerResource})
26  class HRAdminTest(TestCase):
27      def test_hr_admin_config(self):
28          hr_admin = HRAdmin(HR, xadmin.site)
29          self.assertEqual(hr_admin.list_display, ['id', 'name', 'user', '
                department', 'company'])
30          self.assertEqual(hr_admin.list_filter, ['company', 'department'])
31          self.assertEqual(hr_admin.search_fields, ['id', 'name', 'department',
                 'company__name'])
32          self.assertEqual(hr_admin.list_display_links, ['name'])
33          self.assertEqual(hr_admin.list_per_page, 10)
34          self.assertEqual(hr_admin.model_icon, 'fa fa-users')
```

4. 测试结果

在后端测试部分，我们一共编写了 134 个测试样例，测试结果显示测试通过率为 100

八、 项目实现流程

本项目整体实现流程可以分为如下几个阶段：

1. 需求分析与系统设计：主要内容包括每周定时集体会议商量讨论方案，记录讨论内容
2. 系统各功能具体实现：主要内容包括五名成员在各自本地实现分配的代码功能。
3. 项目汇总整理：主要内容包括功能代码汇总、前后端测试、以及项目文档撰写。

本小组共有五名成员，在需求分析与系统设计的准备阶段，多为全组一起讨论，在系统各功能具体实现和项目汇总整理部分的个人分工如下：

王浩	张麟浩	王照钦	王昕炜	李铭辉
系统前端与项目代码汇总	系统后端	系统测试	PPT 制作、视频录制与系统测试	界面设计、需求分析与文档撰写

任务分配以及预期完成时间：

1. 需求分析与系统设计：5 月 20 日-5 月 30 日
2. 系统后端与项目代码汇总：5 月 31 日-6 月 10 日
3. 系统前端：6 月 11 日-6 月 20 日
4. 系统测试：6 月 21 日-6 月 25 日
5. PPT 制作、视频录制与系统测试：6 月 26 日-6 月 30 日

九、 用户手册

(一) 引言

1. 编写目的

本章节旨在确立招聘网站的用户需求，明确这些需求在软件实现中的具体形态，以确保软件开发团队和用户对需求有一致且清晰的理解。本章向用户提供了一份详尽的招聘网站使用指南，包括功能、界面和操作的全面解释，旨在帮助用户深入理解并正确使用该软件。此外，本章还提供了逐步指导，以协助新用户迅速掌握软件功能和界面，降低学习成本。

2. 背景

1. 本用户手册所描述的软件系统的名称为：招聘网站
2. 本软件项目的提出者：
3. 本软件项目的开发者：
4. 本软件项目的目标用户：所有需要招聘和求职的互联网使用者

3. 定义

招聘网站是一个在线平台，它为求职者和雇主提供了一个便捷的互动空间。在这个平台上，求职者可以上传和更新他们的简历，利用关键词、行业类别、工作地点等条件来搜索和申请他们感兴趣的职位。同时，雇主则可以发布职位信息，展示公司文化和工作环境，吸引合适的候选人。招聘网站可以根据用户的求职偏好来推荐职位，提高求职效率。

4. 参考资料

1. 《软件工程—实践者的研究方法》 Roger S.Pressman, Bruce R.Maxim 编著
2. 《用户手册 *GB8567-88* 国家标准》
3. 人机交互：以用户为中心的设计和评估（第 *6* 版）

(二) 用途

1. 功能

此处仅介绍主要功能，更多详细界面功能演示见演示视频。

图 21: 前端界面

求职者端

1. 注册功能：求职者输入自己的个人信息，拖动滑块进行注册，注册成功会显示提示弹窗。
2. 登录功能：求职者输入自己的个人信息，拖动滑块进行登录，登录成功会显示提示弹窗。
3. 个人信息修改功能：求职者登录之后可在个人中心实现个人信息的修改，修改密码等操作。
4. 简历修改功能：求职者登录之后可在简历页面修改简历内容并保存。
5. 报名求职功能：求职者可以在职位市场页面选择理想的职位，报名求职并提交简历。
6. 求职结果查询功能：求职者求职后可在职位详情页查看求职结果，结果由招聘方反馈，求职结果会在聊天页面同步通知。
7. 聊天功能：求职者登录之后可在聊天页面与招聘方、其他求职者交流，支持文字聊天和消息记录。
8. 模拟面试功能：求职者登录之后可在面试页面选择编程语言与题目进行模拟面试，帮助提升面试答题技巧。

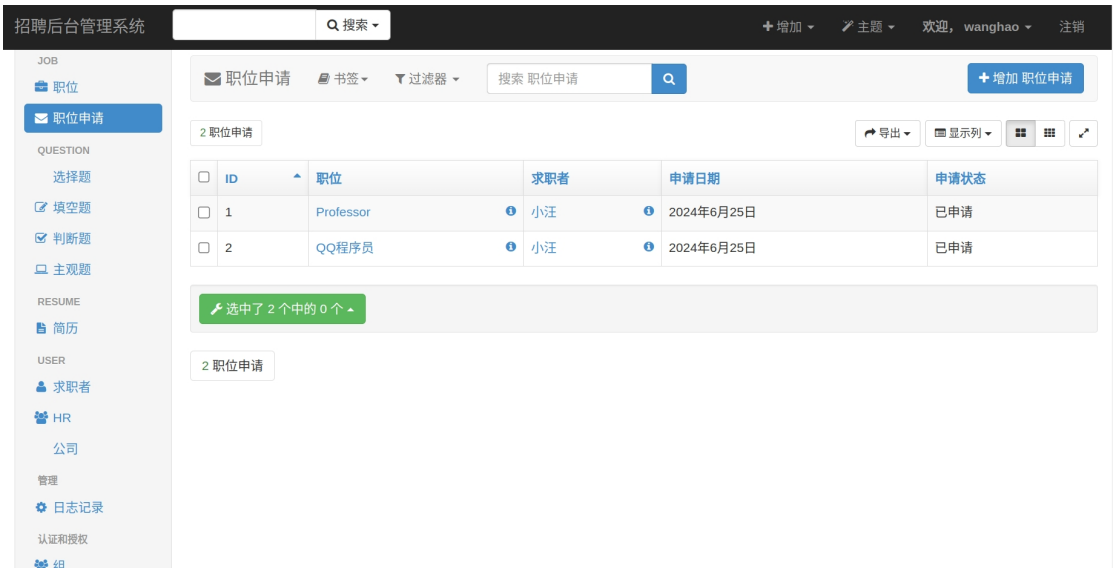

图 22: 后端界面

招聘方端

1. 登录功能：招聘方由管理员统一录入信息，输入个人信息，拖动滑块进行登录，登录成功会显示提示弹窗。
2. 个人信息修改功能：招聘方登录之后可在个人中心实现个人信息的增删改查，修改密码等操作。
3. 公司信息功能：招聘方登录之后可在公司信息页面实现修改公司的信息。
4. 职位管理功能：招聘方进入职位发布界面，可以填写职位信息并发布。已发布的职位可以进入详情页面，进一步修改信息或在招聘结束后删除职位。
5. 招聘功能：招聘方进入职位详情界面，可以查看投递的简历，可以筛选出符合要求的简历，选择合适简历后反馈面试信息，并对剩余简历批量回复婉拒消息。

管理员端

1. 登录功能：管理员输入个人信息，拖动滑块进行登录，登录成功会显示提示弹窗。若忘记密码可以点击界面底部提示，向邮箱发送邮件进行找回。
2. 个人信息增删改查功能：管理登录之后可在个人中心实现个人信息的增删改查，修改密码等操作。同时它还可以增删改查求职者、招聘方的个人信息和权限。

2. 性能

（1）精度

本项目的数据包括求职者数据、招聘方数据、职位数据、求职数据、聊天记录数据。其中求职者数据由个人基本数据、简历数据构成，招聘方数据由个人基本数据、公司信息数据构成。其精度说明如下：

- 求职者数据
  - 姓名：字符串类型，通常是以中文字符表示。

  - 性别：字符类型，通常是“男”或“女”。
  - 出生日期：日期类型，通常以“YYYY-MM-DD”的格式表示。

- 招聘方数据
  - 公司名称：字符串类型，通常是以中文字符表示。
  - 所属行业：字符串类型，通常是以中文字符表示。
  - 所在位置：字符串类型，通常是以中文字符表示。

- 公司数据
  - 公司名称：字符串类型，通常是以中文字符表示。
  - 性别：字符类型，通常是“男”或“女”。
  - 所属部门：字符串类型，通常是以中文字符表示。
  - 所属公司：字符串类型，通常是以中文字符表示。

- 简历数据
  - 求职者姓名：字符串类型，通常是以中文字符表示。
  - 教育经历：字符串类型，通常是以中文字符表示。
  - 工作经历：字符串类型，通常是以中文字符表示。
  - 掌握技能：字符串类型，通常是以中文字符表示。
  - 项目经历：字符串类型，通常是以中文字符表示。
  - 认证和证书：字符串类型，通常是以中文字符表示。

- 职位数据
  - 职位名称：字符串类型，通常是以中文字符表示。
  - 职位描述：字符串类型，通常是以中文字符表示。
  - 职位要求：字符串类型，通常是以中文字符表示。
  - 所属公司：字符串类型，通常是以中文字符表示。
  - 工作地点：字符串类型，通常是以中文字符表示。
  - 发布时间：日期类型，通常以“YYYY-MM-DD”的格式表示。
  - 薪资：int 类型，通常是以数字字符表示。

- 求职数据
  - 职位名称：字符串类型，通常是以中文字符表示。
  - 申请人名称：字符串类型，通常是以中文字符表示。
  - 申请状态：字符串类型，通常是以英文字符表示。
  - 申请日期：日期类型，通常以“YYYY-MM-DD”的格式表示。

- 聊天记录数据
  - 发送方名称：字符串类型，通常是以中文字符表示。

  - 接收方名称：字符串类型，通常是以中文字符表示。
  - 消息内容：文本类型，通常是以中文字符表示。
  - 发送日期：日期类型，通常以“YYYY-MM-DD”的格式表示。
  - 是否接收：布尔类型。

（2）时间特性

招聘网站时间特性如下：

- **响应时间** 所有功能的响应时间应在 1 到 2 秒内。
- **更新处理时间** 基于系统规范的设计，软件更新、修复的复杂性较低，软件更新的打包、发布过程相对简单，因此更新处理时间短，较为迅速。
- **数据传输时间** 软件数据源存储在本地，故软件具有较快的数据传输时间。为保证软件具有良好的用户体验，所有功能的实现在保证安全性和正确性的情况下，应将响应时间降到最低。

（3）灵活性

当需求发生某些变化时，招聘网站对这些变化的适应能力的具体情况如下：

- 操作方式的变化本软件具有非常清晰的编程框架与结构，能够很好地应对操作方式需求的变化，例如从网页版调整为手机界面操作软件。
- 运行环境变化本软件能够在 64-bit Windows 10 及以上版本操作系统中运行。
- 对软件需求的变化本软件编程结构清晰，面对用户对软件功能需求的变化能够拥有足够的灵活性，适应需求变化所带来的不同场景和要求。
- 时间特性运行环境的改变不影响软件的响应时间以及数据传输速率。

3. 安全保密性

招聘网站的安全特性包括但不限于以下几点：

- 用户密码输入时界面均显示为加密状态，防止窥屏。
- 用户密码存储时，采用 **pbkdf2_sha256** 算法迭代 180000 次，防止用户密码泄露。
- 应用程序应对用户输入进行验证和过滤，以防止恶意输入和攻击，如注入攻击、跨站脚本攻击等。
- 三类用户职责分明，用户无法实现越权的任何操作，实现了数据的安全与保密。

4. 运行环境

- 硬设备
  1. 处理器 12th Gen Intel(R) Core(TM) i7-1260P 2.10 GHz
  2. 系统类型 64 位操作系统基于 x64 的处理器
- 支持软件
  1. 操作系统 Linux 20.04
  2. 编译系统 Python 3.6 + Django 3.0 + Node.js 14.0
  3. 浏览器推荐 Edge、Google、FireFox

5. 使用过程

1. 安装与初始化

   - 项目部署环境具体参数

     (a) Python 3.6.0

     (b) Django 3.3.0

     (c) Node.js 14.0

     (d) SQLite Studio 3.4.3

   - 项目部署流程

     (i) 对于项目前端而言：

     – 使用 npm run install 安装依赖

     – 使用 npm run serve 运行前端环境

     (ii) 对于项目后端而言：

     – 使用 pip install -r requirements.txt 安装依赖

     – 使用 python manage.py migrate 迁移数据库

     – 使用 python manage.py runserver 运行后端环境

2. 文卷查询本软件所有用户数据、职位数据、求职数据都是基于软件的所有查询操作，无需用户额外输入查询参数。用户参数在用户登录时获取完成，采用隐式传参的方式在服务器端进行执行数据的查询。由于软件数据以数据库基表的形式进行存储，所以数据查询本质上采用 SQL 语句完成。

3. 出错处理和恢复软件客户端在正常运行的下，一般不会出现问题。多数问题出现在服务端，当服务器端无法访问，或服务端运行崩溃时，将短暂关闭服务器端。在关闭服务端后，用户使用软件会获取到服务无法访问等相关提示。当软件出现卡顿，崩溃等情况，强制关闭即可。服务端会在修复后进行重启，用户重启软件即可正常使用服务。为保证系统的稳定性，我们会定期进行压力测试，稳定性测试，确保用户能够及时获取服务。

4. 终端操作本软件基于 C/S 模式，采用客户端服务器模式开发，软件在各终端独立运行，无需终端之间连接，各终端与服务器独立通信。终端在进行修改操作时，服务器采用事务处理原则，遵从事务处理原子性，保证用户数据的准确性。服务器具有一定的并行能力，支持多终端同时访问。

十、 项目部署环境具体参数

(一) 项目语言

以 Python 语言为主，涉及 JavaScript、CSS、Vue 13.3、HTML 等。

(二) 项目框架

使用 Django+Vue 框架，前后端分离。其中 Django 主要实现后端，Vue 主要实现前端。

(三) 引用包的具体版本

1. Python 3.6.0
2. Django 3.3.0
3. Node.js 14.0
4. SQLite Studio 3.4.3

其余依赖包的具体版本如下所示 (`requirements.txt`):

```
asgiref==3.2.3
certifi==2019.11.28
chardet==3.0.4
coreapi==2.3.3
coreschema==0.0.4
defusedxml==0.6.0
diff-match-patch==20181111
Django==3.0.3
django-crispy-forms==1.9.0
django-filter==2.2.0
django-formtools==2.2
django-guardian==2.2.0
django-import-export==2.0.2
django-reversion==3.0.7
djangorestframework==3.11.0
djangorestframework-jwt==1.11.0
et-xmlfile==1.0.1
future==0.18.2
httplib2==0.9.2
idna==2.9
itypes==1.1.0
jdcal==1.4.1
Jinja2==2.11.1
MarkupPy==1.14
MarkupSafe==1.1.1
odfpy==1.4.1
openpyxl==3.0.3
PyJWT==1.7.1
pytz==2019.3
PyYAML==5.3
requests==2.23.0
sqlparse==0.3.1
tablib==1.1.0
```

```
uritemplate==3.0.1
urllib3==1.25.8
xadmin2==2.0.3
xlrd==1.2.0
xlwt==1.3.0
```

(四) 项目安装部署流程

1. 对于项目前端而言:
   (a) 使用 `npm run install` 安装依赖
   (b) 使用 `npm run serve` 运行前端环境
2. 对于项目后端而言:
   (a) 使用 `pip install -r requirements.txt` 安装依赖
   (b) 使用 `python manage.py migrate` 迁移数据库
   (c) 使用 `python manage.py runserver` 运行后端环境

十一、 项目 GitHub 地址

https://github.com/want6to6good/softwore-work/tree/main

十二、 项目功能截图

项目具体功能实现的截图见讲解视频，此处不重复叙述。

十三、 代码功能介绍

本小节针对于用户管理 (user)、简历和消息管理 (resume)、oj 管理 (question)、工作管理 (job)，这四个核心的项目功能代码进行介绍，这些重要逻辑的代码实现主要位于 Django 后端中对应文件夹中的 Python 文件里。

在详细介绍这四个功能前首先对于每个文件夹中经常出现的 Python 文件进行简单介绍，具体功能如下:

1. admin.py: 该文件用于定义 Django 管理后台（Admin）中的模型（Model）的显示和配置。可以在这里注册模型，以便在管理后台中进行增删改查操作。通过在此文件中定义 ModelAdmin 类，可以自定义模型在管理后台中的展示样式、搜索和过滤选项、字段显示等。

2. adminx.py: 该文件通常是在使用第三方的 Django 插件 xadmin 时才会出现。xadmin 是一个替代 Django 默认管理后台的插件，功能更强大、样式更美观。adminx.py 文件用于配置

xadmin 插件中的模型显示和管理选项。

3. models.py: 该文件用于定义 Django 项目中的数据模型（Model），即数据库中的表结构。通过在这里定义类，可以创建数据库表、字段和约束等。Django 的 ORM（对象关系映射）系统将通过 Python 类与数据库表进行交互，从而实现数据的增删改查。

4. apps.py: 该文件用于配置 Django 应用（App）的一些属性，例如应用的名称、显示名称等。每个 Django 应用都有一个对应的 apps.py 文件。

5. resources.py: 该文件通常是在使用第三方的 Django 插件 django-import-export 时才会出现。django-import-export 是一个用于导入和导出数据的插件，可以方便地将数据从 Excel、CSV 等格式导入到数据库中，或将数据库中的数据导出到其他格式。

6. serializers.py: 该文件通常是在使用 Django REST framework（DRF）时才会出现。DRF 是一个用于构建 Web API 的强大框架，serializers.py 文件用于定义序列化器（Serializer），将复杂的数据对象转换为 JSON 等格式，以便于在 API 中传输和解析。

7. views.py: 该文件用于定义 Django 项目中的视图函数（View），即处理用户请求并返回响应的函数。在这里编写视图函数，可以实现网页的渲染、数据处理、逻辑控制等功能。视图函数可以返回 HTML 页面、JSON 数据等不同类型的响应。

每个对应功能文件夹下的这些 Python 文件分别用于定义后台管理界面的配置、数据库模型的定义、应用属性的配置、导入导出数据、序列化数据、处理用户请求的视图函数，在 Django 项目中各司其职，有助于组织和管理代码，使开发更加高效和模块化。下面主要对于每个功能文件夹下的 views.py 文件进行详细介绍：

1. 用户管理

该功能的 views.py 文件包括六个关键函数，分别对应着不同的 API 请求，实现了用户的注册、登录、修改密码，以及对学生和 hr 信息的增删改查等功能。在实现的过程中通过调用 Django REST framework 提供的功能来简化视图的编写和处理不同类型的请求，具体如下：

1. 自定义用户验证 CustomBackend: 继承自 ModelBackend，通过重写 authenticate 方法实现自定义的用户验证逻辑。在登录时，根据用户名来验证用户，并比对密码是否正确。

```
class CustomBackend(ModelBackend):
    """自定义用户验证"""
    def authenticate(self, username=None, password=None, **kwargs):
        try:
            user = User.objects.get(Q(username=username) | Q(email=username))
            if user.check_password(password):
                return user
        except Exception as e:
            return None
```

2. Jwt-response-payload-handler: 设置 JWT 登录成功后返回的数据。在登录成功后，返回 JWT Token、用户信息和求职者或 hr 信息的数据。

```
    def jwt_response_payload_handler(token, user=None, request=None):
"""设置jwt登录之后返回token和user信息"""
try:
    jobseeker = Jobseeker.objects.get(user=user)
    return {
        'token': token,
        'user': UserDetailSerializer(user, context={'request': request}).
            data,
        'jobseeker': JobseekerSerializer(jobseeker, context={'request':
            request}).data
    }
except Jobseeker.DoesNotExist:
    try:
        hr = HR.objects.get(user=user)
        return {
            'token': token,
            'user': UserDetailSerializer(user, context={'request':
                request}).data,
            'hr': HRSerializer(hr, context={'request': request}).data
        }
    except HR.DoesNotExist:
        return None
```

3. 用户注册功能 RegisterViewSet: 继承自 CreateModelMixin 和 GenericViewSet，实现了用户的注册功能。在注册时，首先检查用户名是否已存在，然后根据前端传回的 role 是求职者还是 hr ‘，创建新的用户和对应的求职者/hr 信息，并将密码加密后存储到数据库。

```
    class RegisterViewSet(mixins.CreateModelMixin, viewsets.
        GenericViewSet):
"""用户注册"""
queryset = User.objects.all()
serializer_class = UserDetailSerializer
def create(self, request, *args, **kwargs):
    user = User.objects.filter(username=request.data['username'])
    if user:
        return Response({'msg': '用户名已存在'}, status=status.
            HTTP_201_CREATED)
    user_detail = UserDetailSerializer(data=request.data)
    if user_detail.is_valid():
        user_detail.save()
    user = User.objects.get(username=request.data['username'])
    # 密码转成密文存储
    user.password = make_password(user.password)
    user.save()
    # 看是hr还是求职者
    if request.data.get('role') == 'jobseeker':
```

```
            jobseeker = Jobseeker(user=user, name=request.data['name'])
            if jobseeker:
                jobseeker.save()
        elif request.data.get('role') == 'hr':
            company = Company.objects.get(id=request.data['company_id'])
            hr = HR(user=user, name=request.data['name'], company=company,
                    department=request.data.get('department', ''))
            if hr:
                hr.save()
        return Response(user_detail.errors)
```

4. 修改用户密码功能 UpdatePwdApi: 继承自 APIView，实现了修改用户密码的功能。在修改密码时，首先检查原始密码是否正确，然后将新密码加密后更新到数据库。

```
    class UpdatePwdApi(APIView):
    """修改用户密码"""
    def patch(self, request):
        # 获取参数
        old_pwd = request.data['oldpwd']
        new_pwd = request.data['newpwd']
        user_id = request.data['userid']
        # 获得请求用户
        user = User.objects.get(id=user_id)
        # 检查原始密码是否正确
        if user.check_password(old_pwd):
            user.set_password(new_pwd)
            user.save()
        else:
            return Response(data={'msg': 'fail'}, status=status.HTTP_200_OK)
        # 返回数据
        return Response(data={'msg': 'success'}, status=status.HTTP_200_OK)
```

5. 全体求职者信息视图 JobseekerViewSet: 继承自 ModelViewSet，实现了对全体求职者信息的增删改查功能。

```
    class JobseekerViewSet(viewsets.ModelViewSet):
    """求职者信息"""
    queryset = Jobseeker.objects.all().order_by('id')
    serializer_class = JobseekerSerializer
```

6. 查询单个求职者 get-personal-info，通过前端传回的 username，找到对应的求职者信息。

```
    @api_view(['GET'])
def get_personal_info(request):
    username = request.query_params.get('username', None)
    if username:
        try:
            user = get_object_or_404(User, username=username)
            jobseeker = Jobseeker.objects.get(user=user)
```

```
            serializer = JobseekerSerializer(jobseeker)
            return Response(serializer.data)
        except Jobseeker.DoesNotExist:
            return Response({"detail": "Not found."}, status=status.
                HTTP_404_NOT_FOUND)
    else:
        return Response({"detail": "Username parameter is required."}, status
            =status.HTTP_400_BAD_REQUEST)
```

7. 全体 hr 信息视图 HRViewSet: 继承自 ModelViewSet，实现了对全体 hr 信息的增删改查功能。

```
    class HRViewSet(viewsets.ModelViewSet):
    """HR信息"""
    # 查询集
    queryset = HR.objects.all().order_by('id')
    # 序列化
    serializer_class = HRSerializer
```

2. 工作管理

该功能的 views.py 文件涉及到工作岗位的创建、投递状态的修改，投递的创建，岗位列表和投递列表的展示的功能:

1. 查询工作岗位 JobListViewSet，根据前端传回来的字段，返回满足条件的工作岗位。

```
    class JobListViewSet(mixins.ListModelMixin, viewsets.GenericViewSet):
    """职位列表页"""
    # 这里要定义一个默认的排序，否则会报错
    queryset = Job.objects.all().order_by('id')[:0]
    # 序列化
    serializer_class = JobSerializer
    # 重写queryset
    def get_queryset(self):
        # 获取查询参数
        company_name = self.request.query_params.get("company_name")
        location = self.request.query_params.get("location")
        min_salary = self.request.query_params.get("min_salary")
        max_salary = self.request.query_params.get("max_salary")
        # 初始查询集
        queryset = Job.objects.all()
        # 过滤公司
        if company_name:
            queryset = queryset.filter(company_name=company_name)
        # 过滤工作地点
        if location:
            queryset = queryset.filter(location=location)
        # 过滤薪资范围
        if min_salary:
```

```
            queryset = queryset.filter(salary__gte=min_salary)
        if max_salary:
            queryset = queryset.filter(salary__lte=max_salary)
        # 随机排序
        queryset = queryset.order_by('?')
        return queryset
```

2. 查询职位申请 ApplicationListViewSet，通过工作名称和求职者，找到对应的投递信息。

```
        class ApplicationListViewSet(mixins.ListModelMixin, viewsets.
            GenericViewSet):
    """职位申请列表页"""
    queryset = Application.objects.all().order_by('id')[:0]
    serializer_class = ApplicationSerializer
    def get_queryset(self):
        # 获取查询参数
        job_name = self.request.query_params.get("job_name")
        jobseeker_name = self.request.query_params.get("jobseeker_name")
        # 初始查询集
        queryset = Application.objects.all()
        # 过滤职位
        if job_name:
            queryset = queryset.filter(job__title=job_name)
        # 过滤求职者
        if jobseeker_name:
            queryset = queryset.filter(jobseeker__name=jobseeker_name)
        return queryset
```

3. 更新职位申请状态 UpdateApplicationStatusView, 对对应的投递状态进行修改。

```
        class UpdateApplicationStatusView(APIView):
    """更新职位申请状态"""
    def put(self, request, *args, **kwargs):
        application_id = request.data.get('application_id')
        new_status = request.data.get('status')
        # 校验新的状态是否有效
        valid_statuses = ['applied', 'interview', 'hired', 'rejected']
        if new_status not in valid_statuses:
            return Response({"detail": "无效的申请状态"}, status=status.
                HTTP_400_BAD_REQUEST)
        application = get_object_or_404(Application, id=application_id)
        # 更新申请状态
        application.status = new_status
        application.save()
        return Response({"detail": "申请状态更新成功"}, status=status.
            HTTP_200_OK)
```

4. 创建新的工作岗位 CreateJobVie。通过对应的参数创建对应的工作需求。

```
        class CreateJobView(APIView):
    """增添新的工作岗位"""
    def post(self, request, *args, **kwargs):
        title = request.data.get('title')
        description = request.data.get('description')
        requirements = request.data.get('requirements')
        company_name= request.data.get('company_name')
        location = request.data.get('location')
        salary = request.data.get('salary')
        # 校验必填字段
        if not all([title, description, requirements, company_name, location,
             salary]):
            return Response({"detail": "所有字段都是必填的"}, status=status.
                HTTP_400_BAD_REQUEST)
        company = get_object_or_404(Company, name=company_name)
        job = Job.objects.create(
            title=title,
            description=description,
            requirements=requirements,
            company=company,
            location=location,
            salary=salary
        )
        return Response({"detail": "工作岗位创建成功", "job_id": job.id},
            status=status.HTTP_201_CREATED)
```

5. 创建简历申请 CreateApplicationView。

```
        class CreateApplicationView(mixins.ListModelMixin, viewsets.
            GenericViewSet):
    queryset = Application.objects.all()
    serializer_class = ApplicationSerializer

    @action(detail=False, methods=['post'])
    def create_application(self, request, *args, **kwargs):
        name = request.data.get('username')
        job_title = request.data.get('jobname')
        # 校验必填字段gongtou
        if not all([name,job_title]):
            return Response({"detail": "所有字段都是必填的"}, status=status.
                HTTP_400_BAD_REQUEST)
        user = get_object_or_404(User, username=name)
        seeker = Jobseeker.objects.get(user=user)
        job_instance = Job.objects.get(title=job_title)  # 使用正确的模型类和
            变量名进行查询
        application = Application.objects.create(
            job=job_instance,
            jobseeker=seeker
```

```
        )
        return Response({"detail": "简历投递成功"}, status=status.
            HTTP_201_CREATED)
```

3. 简历和消息管理

该功能的 views.py 文件主要实现了用户管理和简历管理的功能，包括创建和更新简历、获取简历列表、发送和接收消息等。以下是对每个函数和类的具体功能介绍：

1. change-resume 函数功能：处理用户提交的简历更新请求，更新或创建简历。输入：用户名、姓名、性别、教育背景、工作经验、技能、项目、认证等信息。输出：返回简历创建或更新成功的信息。

```
        @api_view(['POST'])
def change_resume(request):
    username = request.data.get('username')
    name = request.data.get('name')
    gender = request.data.get('sex')
    education = request.data.get('education', '')
    experience = request.data.get('experience', '')
    skills = request.data.get('skills', '')
    projects = request.data.get('projects', '')
    certifications = request.data.get('certifications', '')
    user = get_object_or_404(User, username=username)
    jobseeker = Jobseeker.objects.get(user=user)
    gender_map = {
        '男': 'm',
        '女': 'f'
    }
    if gender in ['男', '女']:  # 检查性别值是否有效
        jobseeker.gender = gender_map[gender]
        jobseeker.name =name
        jobseeker.save()
    # 检查简历是否存在
    resume, created = Resume.objects.update_or_create(
        jobseeker=jobseeker,
        defaults={
            'education': education,
            'experience': experience,
            'skills': skills,
            'projects': projects,
            'certifications': certifications,
        }
    )
    if created:
        return Response({"detail": "简历创建成功"}, status=status.
            HTTP_201_CREATED)
    else:
```

```
35          return Response({"detail": "简历更新成功"}, status=status.HTTP_200_OK
                )
```

2. ResumeListViewSet 提供简历的列表视图，支持获取简历列表的操作。输入：无特定输入，默认获取所有简历。输出：返回简历列表。

```
1           class ResumeListViewSet(mixins.ListModelMixin, viewsets.
                GenericViewSet):
2       """职位申请列表页"""
3       queryset = Resume.objects.all().order_by('id')
4       serializer_class = ResumeSerializer
```

3. get-resume 根据用户名获取对应求职者的简历信息。输入：用户名. 输出：返回求职者的简历信息，若用户不存在则返回 404 错误。

```
1           @api_view(['GET'])
2   def get_resume(request):
3       username = request.query_params.get('username', None)
4       print(username)
5       if username is not None:
6           try:
7               user = get_object_or_404(User, username=username)
8               jobseeker = Jobseeker.objects.get(user=user)
9               resumes = Resume.objects.filter(jobseeker=jobseeker)
10              serializer = ResumeSerializer(resumes,many=True)
11              return Response(serializer.data)
12          except Jobseeker.DoesNotExist:
13              return Response({"detail": "Not␣found."}, status=status.
                    HTTP_404_NOT_FOUND)
14      else:
15          return Response({"detail": "Username␣parameter␣is␣required."}, status
                =status.HTTP_400_BAD_REQUEST)
```

4. create-message 处理消息创建请求，创建一条新的消息记录。输入：发送者用户名、接收者用户名、消息内容。输出：返回消息创建成功的信息和消息 ID。

```
1           def create_message(request):
2       if request.method == 'POST':
3           sender_username = request.POST.get('sender_name')
4           receiver_username = request.POST.get('receiver_name')
5           content = request.POST.get('content')
6           sender = get_object_or_404(User, username=sender_username)
7           receiver = get_object_or_404(User, username=receiver_username)
8           message = Message.objects.create(sender=sender, receiver=receiver,
                content=content)
9           return JsonResponse({'status': 'success', 'message_id': message.id})
10      return JsonResponse({'status': 'fail'}, status=400)
```

5. get-user-messages 获取指定用户发送和接收的所有消息。输入：用户名。输出：返回该用户的所有消息。

```
        def get_user_messages(request):
    if request.method == 'GET':
        username=request.POST.get('username')
        user = get_object_or_404(User, username=username)
        messages = Message.objects.filter(sender=user).union(Message.objects.
            filter(receiver=user)).order_by('-timestamp')
        messages_data = [{
            'sender': message.sender.username,
            'receiver': message.receiver.username,
            'content': message.content,
            'timestamp': message.timestamp,
            'is_read': message.is_read,
            'id':message.id,
        } for message in messages]
    return JsonResponse({'messages': messages_data})
```

6. mark-message-as-read 标记指定的消息为已读。输入：消息 ID。输出：返回消息标记成功的信息。

```
        def mark_message_as_read(request):
    if request.method == 'POST':
        message_id = request.POST.get('message_id')
        message = get_object_or_404(Message, id=message_id)
        message.is_read = True
        message.save()
        return JsonResponse({'status': 'success'})
    return JsonResponse({'status': 'fail'}, status=400)
```

4. oj 管理

该部分代码主要实现了模拟面试中的在线 oj 功能，前端在接收了求职者输入的代码后会发送到后端，后端接收代码后会在当前目录下生成一个临时的 python 文件并用命令行执行，如果执行完毕没有发生错误，则返回 ac，否则返回错误消息，代码如下：

```
    @csrf_exempt
# @api_view(['POST'])
def execute_code(request):
    if request.method == 'POST':
        try:
            code = request.POST.get('code')
            print(code)
            if not code:
                return JsonResponse({'error': 'No code provided'}, status
                    =400)
            # 创建一个唯一的文件名
            timestamp = timezone.now().strftime('%Y%m%d%H%M%S%f')
```

```
            filename = f'code_{timestamp}.py'
            filepath = os.path.abspath(__file__)
            # 将代码保存为文件
            with open(filepath, 'w') as code_file:
                code_file.write(code)
            # 执行代码
            result = subprocess.run(['python', filepath], capture_output=True
                , text=True, timeout=10)
            # 删除文件
            os.remove(filepath)
            return JsonResponse({
                'stdout': result.stdout,
                'stderr': result.stderr,
                'returncode': result.returncode
            })
        except Exception as e:
            return JsonResponse({'error': str(e)}, status=500)
    else:
        return JsonResponse({'error': 'Invalid request method'}, status=405)
```

十四、 总结

在本学期的软件工程大作业中,我和团队共同合作完成了一个招聘平台网站,采用了 Django+Vue 框架，实现了前后端分离的架构。该平台不仅满足了求职者和招聘方要求的基本功能，如职位发布、简历投递求职、招聘结果反馈等，还针对求职者需求额外设计了大模型模拟面试功能，以便求职者提升面试能力。

我们在项目开发过程中，进行了前期调研分析，编写了前后端单元测试，并完成了功能和性能测试。为了保证用户信息的安全性，我们采用了 pbkdf2_sha256 算法进行密码存储。

整个项目开发过程使我们深刻体验到了软件工程的全过程，从需求分析、功能设计、实现、代码合并，到测试和文档撰写等多个环节。这不仅锻炼了我们的软件工程能力，还增强了团队合作意识，收获了知识和友谊。