**苏州大学**2022-2023-1学期

**WEB应用开发 课程期末考核报告**

院系 计算机科学与技术学院

年级 2020

专业 软件工程

学号 2030416018

姓名 高歌

成绩

提交日期 2023/1/8

**目 录**

[1 需求分析 3](#_Toc124114683)

[2 数据库设计 3](#_Toc124114684)

[3 模块设计与实现 5](#_Toc124114685)

[3.1 技术框架 5](#_Toc124114686)

[3.2 用户管理 6](#_Toc124114687)

[3.3 登录页 10](#_Toc124114688)

[3.4 用户档案页 10](#_Toc124114689)

[3.5 主页 11](#_Toc124114690)

[4 总结 13](#_Toc124114691)

# 需求分析

完成一个简单的招聘信息发布网站，包含两个模块：用户管理模块（鉴权）和招聘信息管理模块。

对于用户模块，应完成以下功能：

1. 基本的用户增删改查接口
2. 根据用户所在权限组的不同（匿名用户/普通用户/管理员），为API访问设置不同的权限。如匿名用户不得删改招聘信息，只能查询已有的招聘信息等。

对于招聘信息管理模块，应完成以下功能：

1. 基本的招聘信息增删改查接口
2. 对于已有招聘信息的简单搜索

# 数据库设计

由于后端用户管理的实现使用Shiro + Spring Data JPA，因此使用较经典的用户-角色-权限模型。将其分为三张表。三张表设计如下。

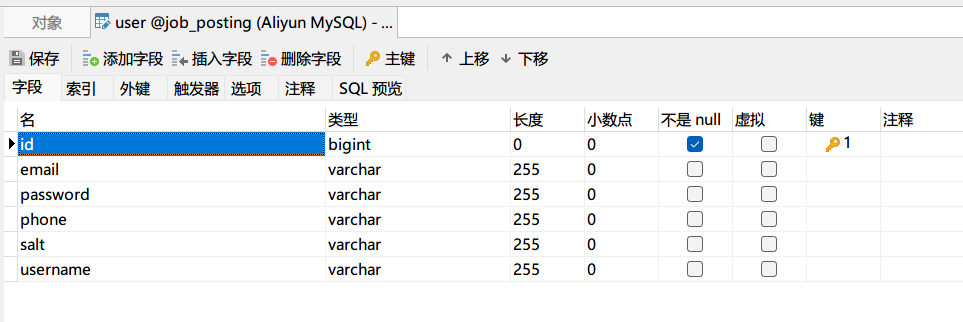


图 ‑ user表

user表除主键id、email、phone这些一般信息外，还存储用户名和密码，以及密码的加盐值。这是由于在数据库中明文存储密码非常不安全，因此在后端实现中使用双重MD5码校验进行加盐处理。具体存储信息如图所示。

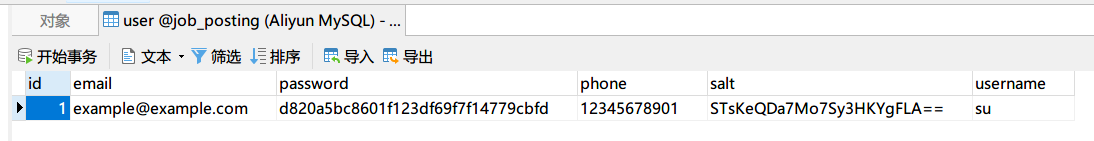


图 ‑ user表存储示例

可以看到，即使数据库信息泄露，攻击者也无法从数据库中存储的混淆后的密码和盐值中还原得到原密码。这一过程是单向的，只能校验密码而无法从这些信息中得到原密码。

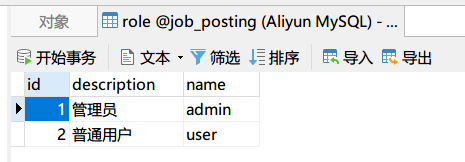


图 ‑ role表

role表初始时包含所有用户角色。在这里设置为两个角色，admin和user。admin角色拥有最高权限，而user角色只能对自己发布的招聘信息进行修改和删除操作。每个user对应多个role。

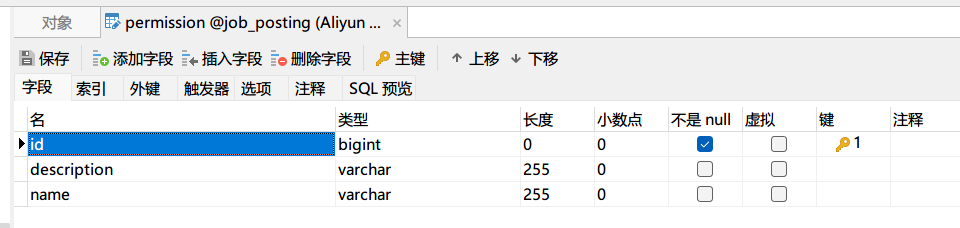


图 ‑ permission表

permission表是role表的细化。每个role对应多个权限。不过在此demo中该表实际上并未实际使用。这主要是由于需求比较简单，暂时直接判断role即可。

同时还有两个关联表，命名中的“mtm”表示“Many To Many”，多对多关系。

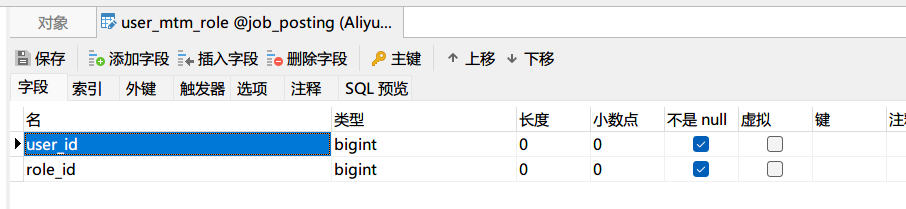


图 ‑ 关联表user\_mtm\_role

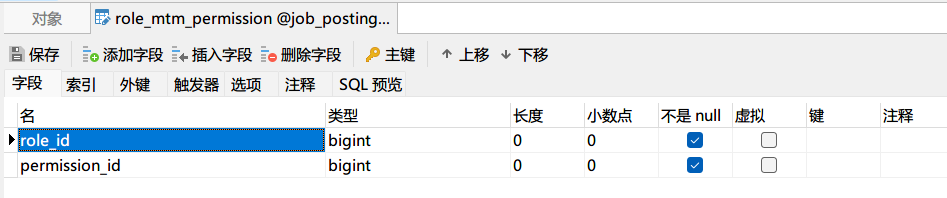


图 ‑ 关联表role\_mtm\_permission

除与用户权限相关的表外，还包含一个job\_posting表用于存储招聘信息。

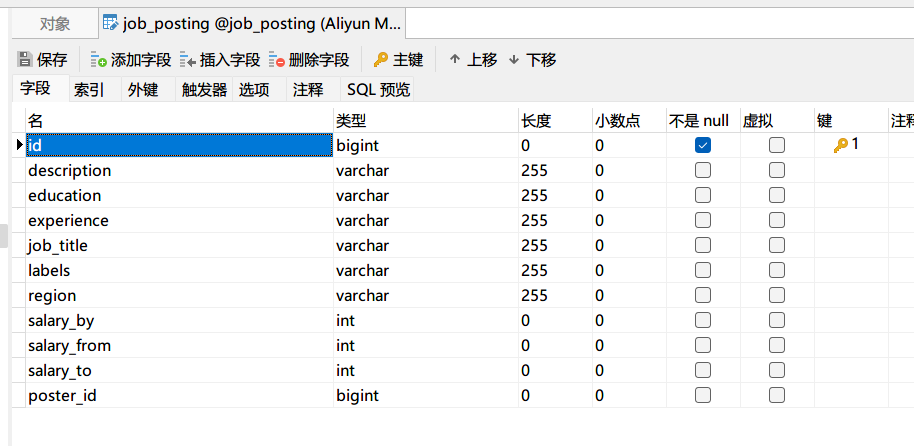


图 ‑ job\_posting表

该表除了需要说明一下poster\_id字段对应上传者的id外，其余字段应当无需太多说明。需要注意的是这里因为只是demo，所以description使用了varchar(255)，实际应当用text类型，以存放更多数据。

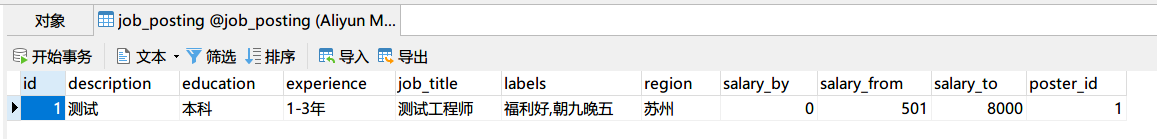


图 ‑ job\_posting表示例

# 模块设计与实现

## 技术框架

总体实现采用前后端分离方式。

后端使用Spring Boot + Data JPA + Shiro + Lombok + Springdoc（注意不是Springfox，由于Springfox在2020年停更，因此使用Springdoc替代，Springdoc使用OpenApi3规范，并且支持直接使用注解处理器将Controller中的注解作为Swagger-UI的Summary处理）。

前端使用Vue + TypeScript实现，使用的脚手架是Nuxt 3。由于Nuxt 3已做好了路由等配置，因此不需要额外配置。除此之外，前端还引入了Element Plus作为组件库，使用SASS（准确地说是SCSS）作为样式表，使用axios编写api。

具体技术实现请参考代码。

## 用户管理

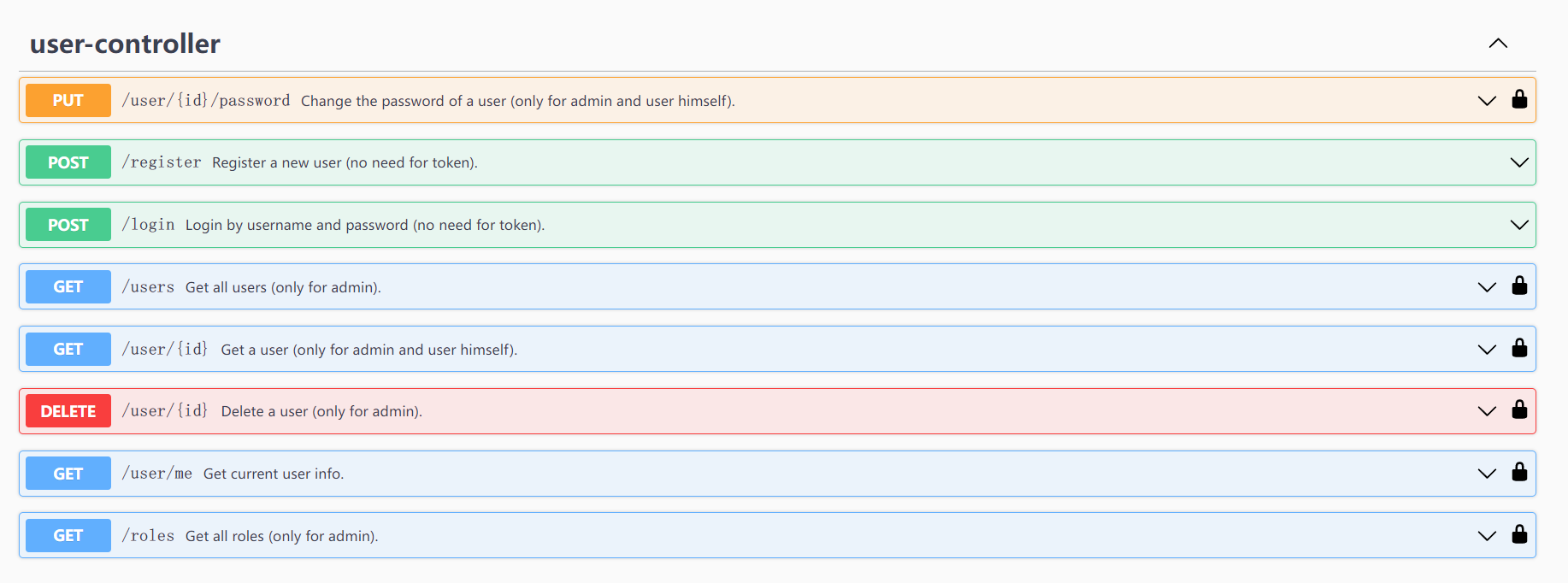
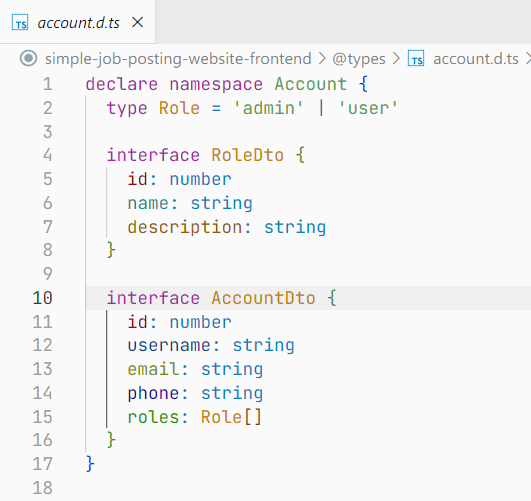


图 ‑ 后端Swagger文档（http://localhost:8080/swagger-ui/index.html#/）

用户管理方面，后端提供的API参考上图。本项目的所有API均使用RESTFUL风格编写，直接使用response-status作为错误码，不额外封装BaseResponse。



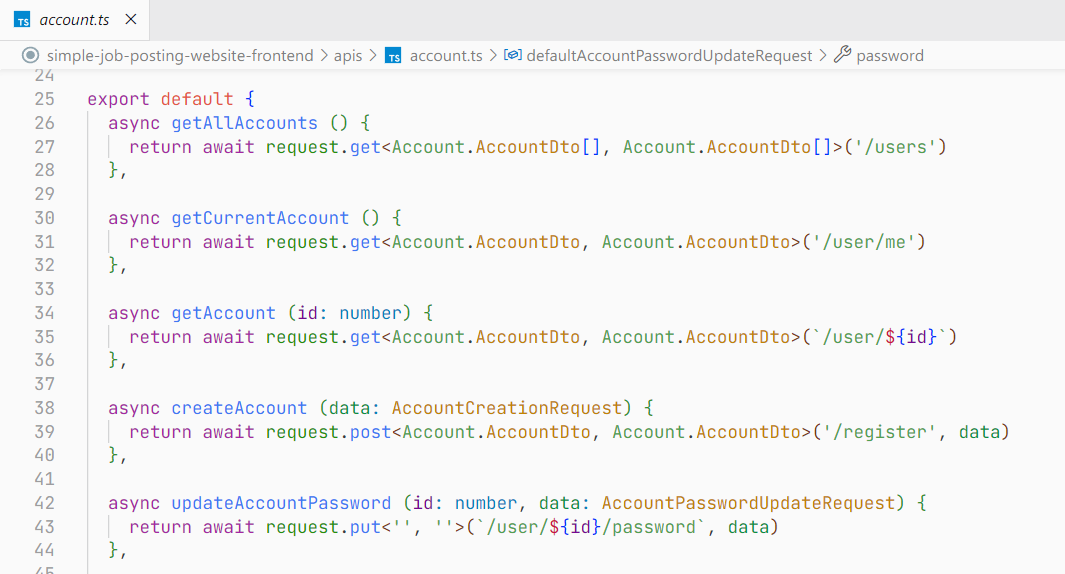


图 ‑ 前端API示例

上为前端代码中一部分与用户相关的API，已经编写了相关且完备的类型，并且使用axios编写了相应对接的API。由于时间仓促，并未使用Jest编写自动化测试，但已均手动测试通过，请见谅。

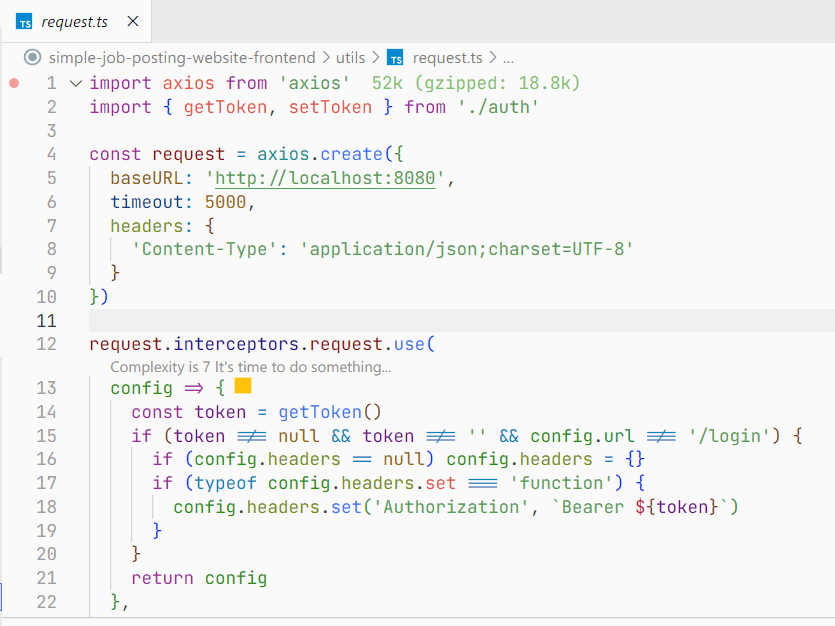


图 ‑ 拦截器

本项目使用JWT（Json Web Token）作为无状态的登录解决方案，后端编写了相关的拦截器与过滤器，这里不再过多演示。上图为前端axios的拦截器，大致原理是除login的response外，其他response均会在Authorization头中更新JWT，因此这里会拦截下来在全局的store中进行更新。同理，每次发送request时也进行拦截，在Authorization头中加入`Bearer ${token}`（此为JWT的一般使用形式）。

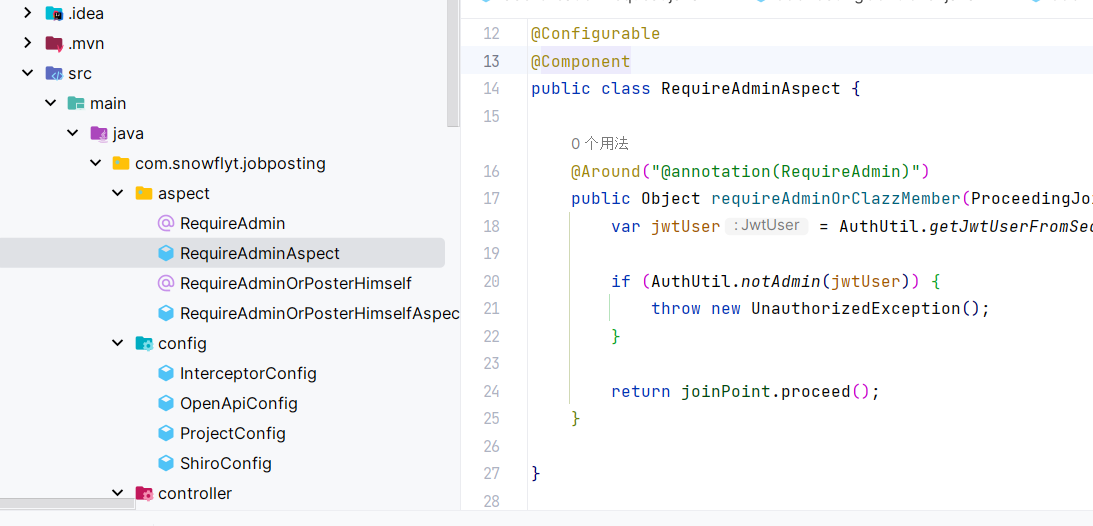


图 ‑ 后端鉴权相关注解

后端代码中使用注解配合Shiro实现鉴权处理，上为注解示例。



图 ‑ 鉴权注解使用示例

具体来说，在Controller中，需要像上图一样使用。

同时，在前端也做了middleware，访问白名单之外的页面会重定向到login。



图 ‑ 前端middleware

这样就实现了一套完备的鉴权验证逻辑。

## 登录页



图 ‑ 登录页

第一次进入页面会要求登录，存储在Cookies中。

## 用户档案页

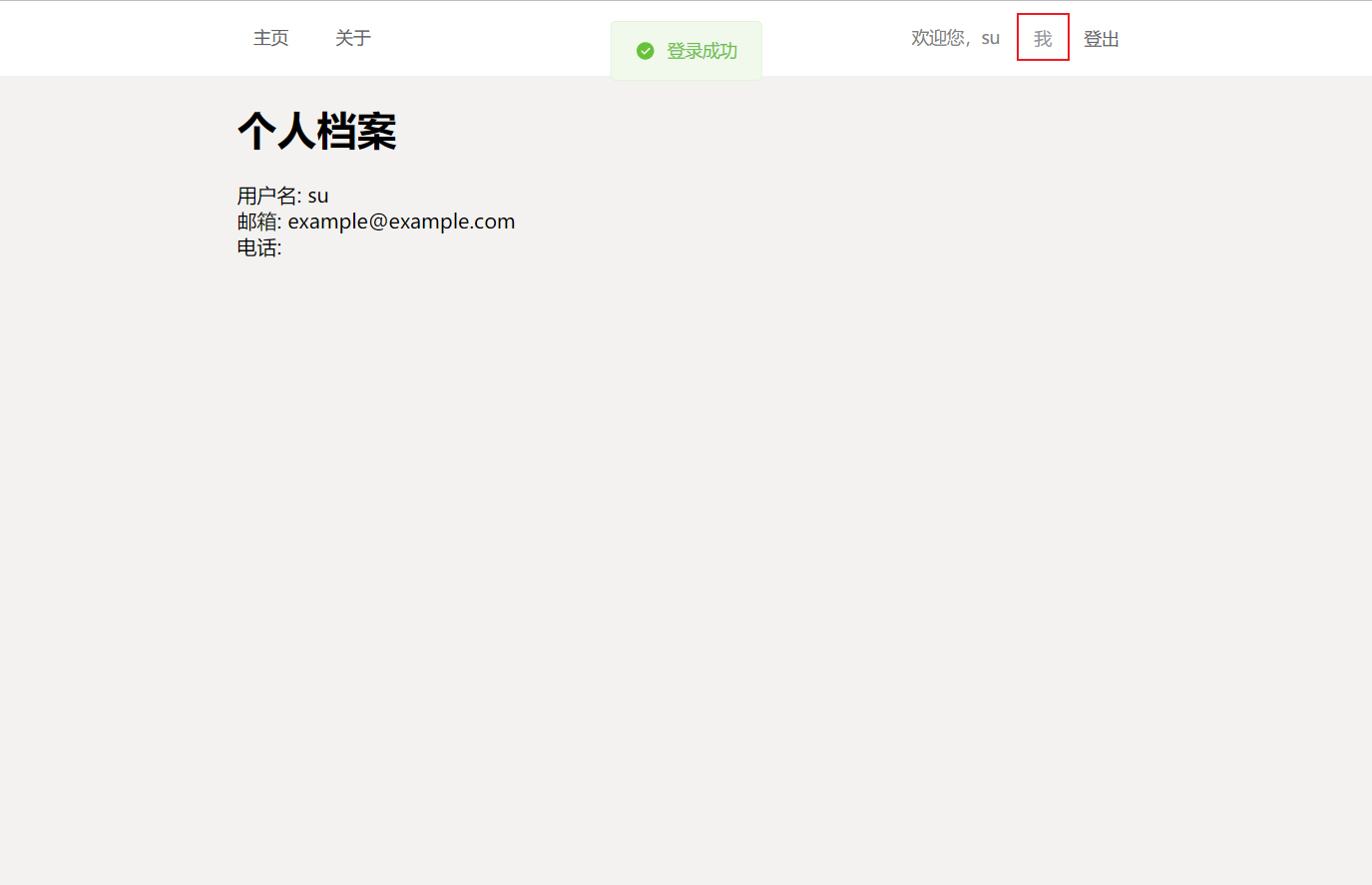


图 ‑ 档案页

显示用户档案。

## 主页





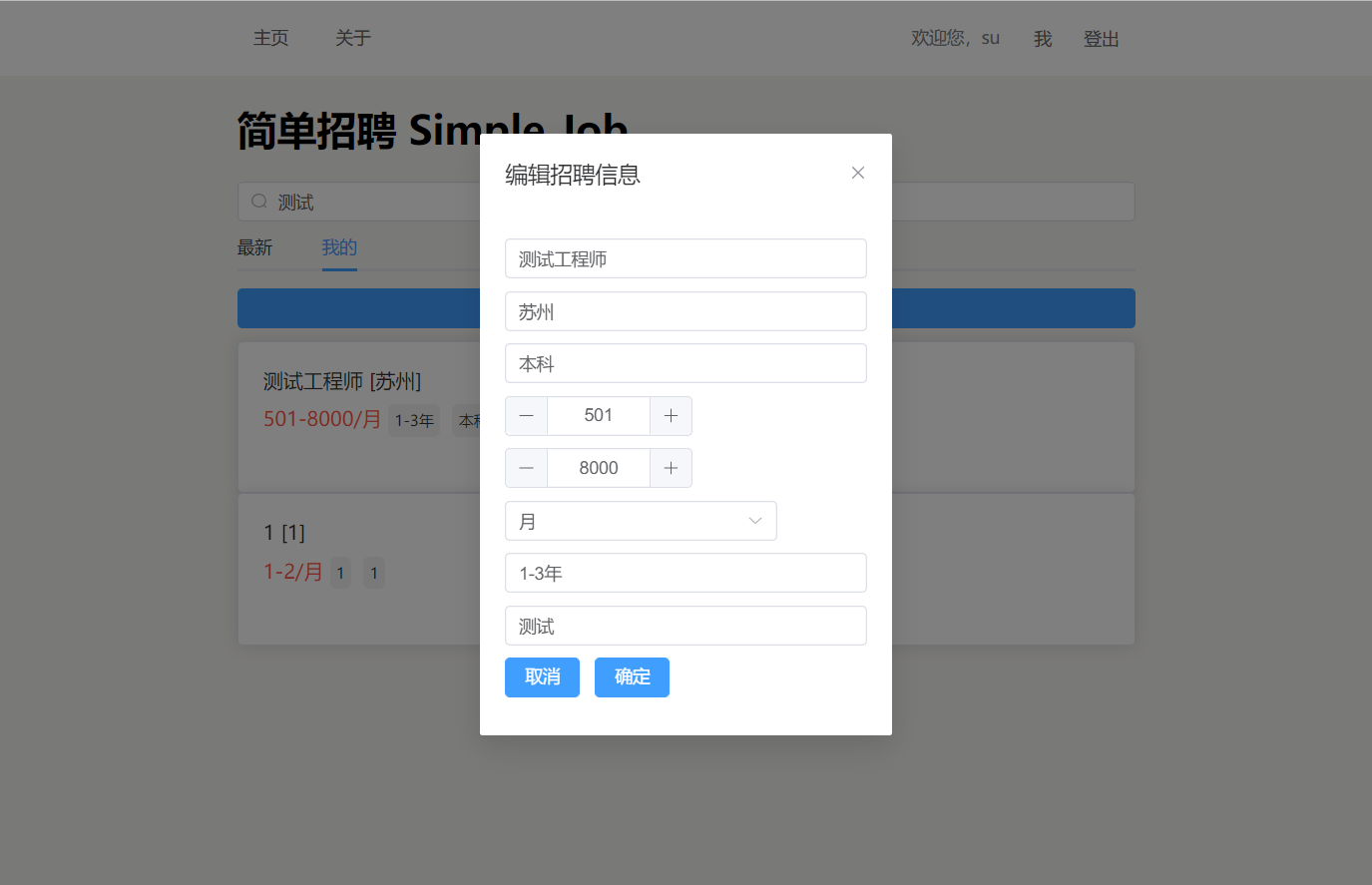
显示和搜索相关信息。（无需登录）



详情。



在这里可以看到自己上传的招聘信息，并进行增删改查。



# 总结

总的来说，本系统采用前后端分离架构实现了一个简单的demo。引入了较为完善的鉴权框架，比较安全。同时支持基本的增删改查。