

# 《数据库系统原理》 大作业

## 系统实现报告

题目名称：P.R.T.S.罗德岛管理系统

20373585 夏瑞斌

20373569 俞逸洋

20373586 赵雨晨

2022 年 12 月 23 日

# 一、系统结构设计

## (一) 环境与框架使用情况

本次项目开发采取了前后端分离的开发策略，使用了 Vue3 搭建前端，使用 Django 搭建后端。具体情况见下

### 1. 前端

前端开发框架	Vue3	vue 3.2.39
项目搭建脚手架	Vite	vite 3.1.3
Javascript 运行环境	Node.js	node.js 16.
包管理工具	npm	npm 8.15.0

所使用的组件库

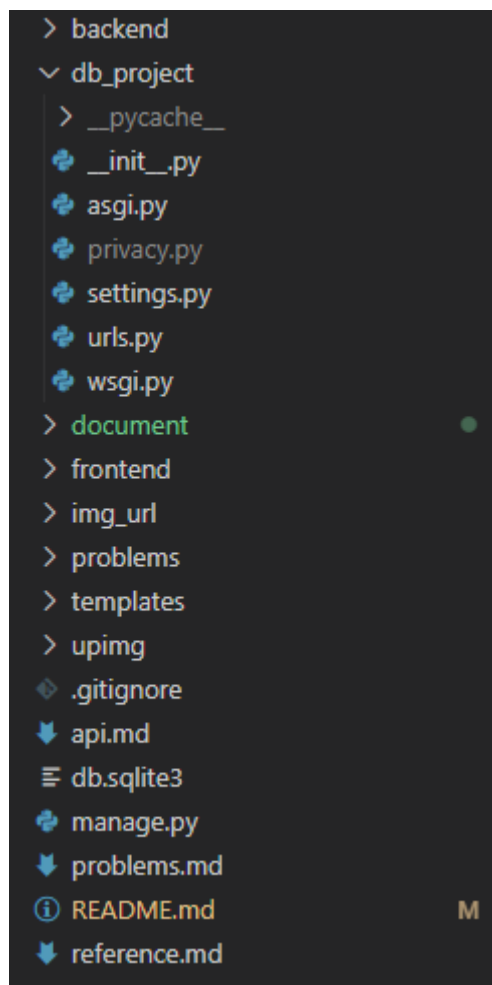
element-plus	基础 UI 的组建库	element-plus@2.2.17 @element-plus/icons-vue@2.0.9
router	前端路由	vue-router@4.1.5
vueuse	提供组合式 api 使用工具函数	@vueuse/components@9.3.0 @vueuse/core@9.3.0 @vueuse/integrations@9.3.0
axios	处理跨域 http 请求	axios@0.26.1
vuex	前端的状态管理模式和库	vuex@4.0.2
sass	CSS 的预编译语言	sass@1.56.1
v-md-editor	markdown 相关组件	@kangc/v-md-editor@2.3.15

## 2. 后端

后端开发框架	Django	Django 4.1.1
前后端交互	Django Rest Framework	djangorestframework 3.14.0
数据库管理系统	MySQL	MySQL8.0
包管理工具	pip	npm 8.15.0

## （二） 项目体系结构

### 1. 项目总体结构如下：



其中总体项目文件由 django 搭建（运行`django-admin startproject db-project`得来），通过 `python manage.py` 进行相应管理。

（1） backend 目录

属于 django 的一个 app，用作整个项目的后端

（2） db\_project 目录

基本 django 项目的配置。其中 `privacy.py` 记录相关数据库账户信息，在 `urls.py` 中引用作为连接数据库的凭证

（3） frontend 目录

前端目录，为 vue3 项目（根目录运行`npm init vite@latest frontend -- --template vue`得来）

## 2. 前端 frontend 目录结构如下：

src 目录中为主要前端内容，

（1） api 目录：前后端交互使用的接口函数

（2） component 目录：前端所使用的相关小组件

（3） composable 目录：组合式 api 一些小函数包装

（4） router 目录：路由以及嵌套路由的配置

（5） store 目录：vuex 相应内容，负责前端暂存储状态和值

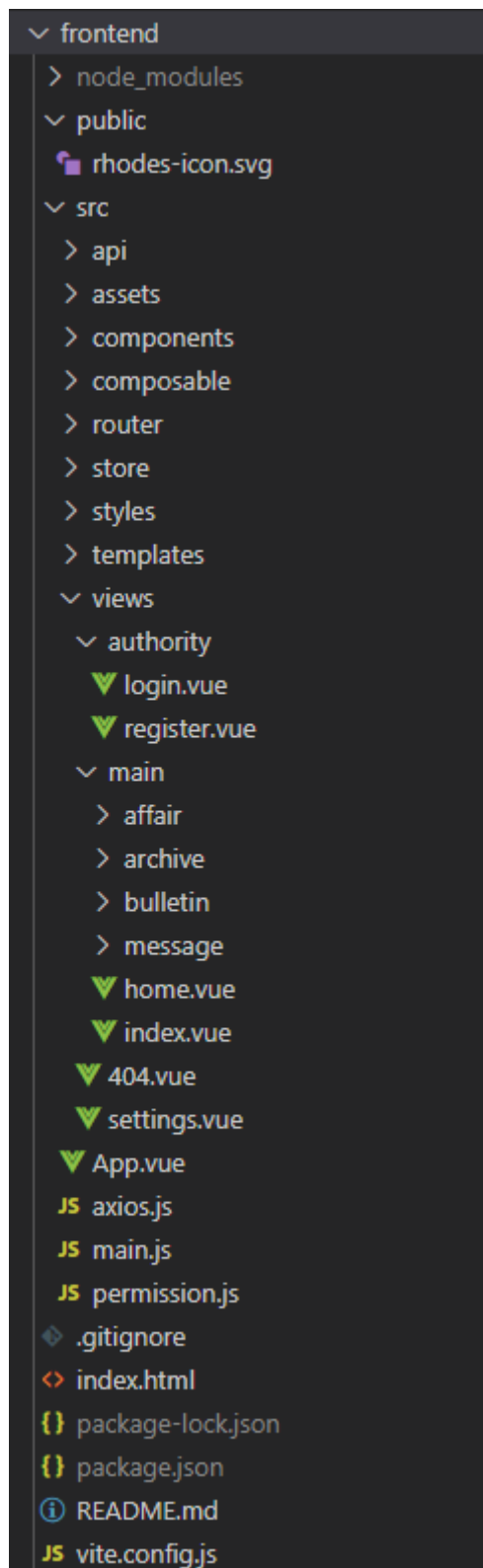
（6） styles 目录：相关 css 文件

（7） views 目录：主要的 vue 页面文件

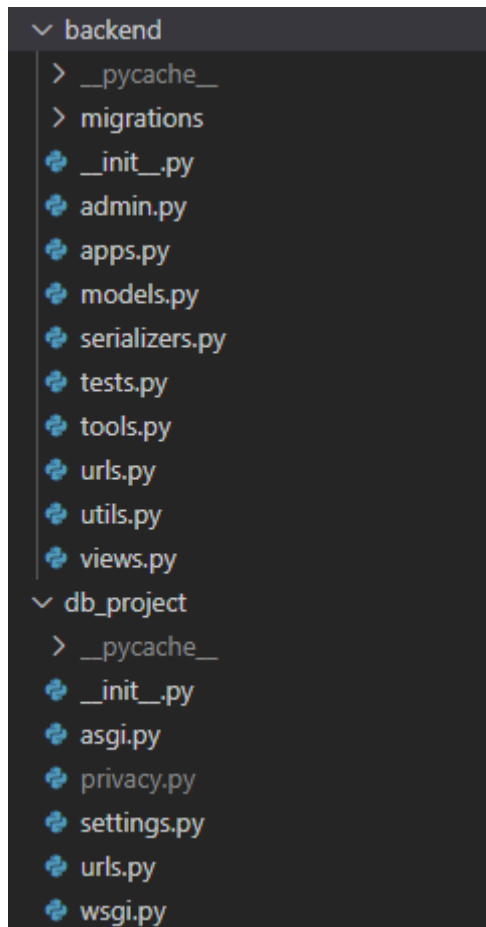
项目是单组件文件（SFC），所有页面其实来源于根组件 `App.vue`。其内采用嵌套路由的方式进行页面跳转，则目录组织依据逻辑也由功能和嵌套规则划分。

其中 `authority` 目录为账户管理相关页面，`login.vue` 为登录页面，`register` 为注册页面；`main` 目录为主题页面，其中 `home.vue` 为登陆后主界面，`index.vue` 为通用侧边栏界面（在嵌套的外层）

四个并列目录 `affair`、`archive`、`bulletin`、`message` 四个目录对应系统相应的四个子功能。每个中包含基本布局和相关组件的 `vue` 文件。



3. 后端 backend 目录结构如下：



- (1) models.py: 数据表的相关模型
- (2) serializers.py: 序列化器，DRF 提供
- (3) tools.py: 工具类函数
- (4) urls.py: 后端路由
- (5) views.py: django 视图，生成响应对象

### (三) 项目功能结构

地方下本项目系统分为账户管理子系统和剩下 4 个基本功能模块。由于时间限制，后两个模块并未实装仅由基本框架

具体如下：



## 二、数据库基本表的定义

用户信息表：基本账户信息，用于登录及以后的拓展操作

用户信息表 (user\_account)

数据项名字	数据类型	约束	备注
CodeName	char(30)	primary key	代号；相当于用户名
Password	char(20)	not null	
Permission	tinyint	not null, unsigned	值越小，权限越高；最小为0
Mail	char(20)		邮件地址

用户资料表：用户个人自资料信息，用于平台个人信息展示与交流

用户资料表 (user\_profile)

数据项名字	数据类型	约束	备注
CodeName	char(30)	primary key	代号；相当于用户名
Gender	tinyint		性别（二值）
Class	char(20)		职业分类
Region	char(30)		地区
Race	char(20)		种族
Avatar	blob		头像
Bio	tinytext		自述

**待审批用户表：** 由于该系统针对为公司干员，其注册入职过程需要经由高层/管理员审核。当用户注册后，相关信息会先转填入此表中，等待审核

**待审批用户表 (account\_approve\_queue)**

数据项名字	数据类型	约束	备注
CodeName	char(30)	primary key	代号；相当于用户名
Password	char(20)	not null	
Permission	tinyint	unsigned	待管理员分配
Gender	tinyint		
Class	char(20)		职业分类
Region	char(30)		地区
Race	char(20)		种族
Description	text		备注

**文章实体表：** 记录情报通知模块相应文章的实体信息

**文章实体表 (passage)**

数据项名字	数据类型	约束	备注
PId	int	primary key	文章编号
Title	char(45)	not null	标题（15个中文字）
Poster	char(30)	not null, foreign key	发布者（CodeName）
PostDate	timestamp	not null	发布时间
LastEditor	char(30)	not null, foreign key	最后编辑者（CodeName）
LastEditTime	timestamp	not null	最后编辑时间
Content	text	not null	内容，64KB（21845个中文字）
Type	tinyint	not null	文章类别：公告、晋升公示、地区情报

**回复实体表：** 对相应公告和文章的回复功能

**回复实体表 (reply)**

数据项名字	数据类型	约束	备注
RId	int	primary key	reply编号
Replier	char(30)	not null, foreign key	回复者（CodeName）
AttachedPId	int	not null, foreign key	依附的文章编号（PId）
Content	tinytext	not null	



## 消息实体表：记录通讯联络模块的消息实体

消息实体表 (message)

数据项名字	数据类型	约束	备注
MIId	int	primary key	message编号
Type	tinyint	not null	对个人-0; 对群组-1
SendFrom	char(30)	not null, foreign key	发送方
SendToPerson	char(30)	foreign key	接收个人
SendToGroup	int	foreign key	接收群组
ContentText	tinytext	not null	消息文本
Picture	blob		图像信息
Time	timestamp	not null	发布时间

## 群组实体表：记录群组实体

群组实体表 (group)

数据项名字	数据类型	约束	备注
GId	int	primary key	group编号
GName	char(30)	not null	group名
Owner	char(30)	not null, foreign key	群主，初建为创建该群的干员
BornTime	timestamp	not null	群组创建时间

## 用户-群组关系表：用户和群组之间的联系

用户-群组关系表 (operator\_group)

数据项名字	数据类型	约束	备注
GId	int	primary key, foreign key	group编号
CodeName	char(30)	primary key, foreign key	干员名称，得是已有用户
isOwner	tinyint		是否是群组

所有表的实现方式均由 django 的 model.py 代为构造

```
models.py x
backend > models.py > UserProfile
1 from django.db import models
2 import datetime
3
4
5 # Create your models here.
6
7
8 class UserAccount(models.Model):
9     # Oid = models.AutoField('Oid', primary_key=True)
10    CodeName = models.CharField('CodeName', max_length=30, primary_key=True)
11    Password = models.CharField('Password', max_length=20, null=False)
12    Permission = models.PositiveSmallIntegerField('Permission', null=False)
13    Mail = models.CharField('Mail', max_length=20, default="")
14
15    class Meta:
16        db_table = 'user_account'
17
18    def __str__(self):
19        return self.CodeName
20
21
22 class UserProfile(models.Model):
23     # Oid = models.AutoField('Oid', primary_key=True)
24
25    CodeName = models.CharField('CodeName', max_length=30, primary_key=True)
26    Gender = models.IntegerField('Gender', choices=((0, 'male'), (1, 'female')) # 0表示男性, 1表示女性
27    Class = models.CharField('Class', max_length=20)
28    Region = models.CharField('Region', max_length=30)
29    Race = models.CharField('Race', max_length=20)
30    Avatar = models.ImageField(verbose_name='Avatar', upload_to='img_url', null=True)
31    Bio = models.TextField('Bio', default="")
32
33    class Meta:
34        db_table = 'user_profile'
35
36    def __str__(self):
37        return self.CodeName
38
39
```

## 三、系统重要功能实现方法

### （一）系统功能概述

明日方舟——罗德岛管理系统。模拟泰拉大陆上一家医药公司（罗德岛）的日常经营模式模拟罗德岛日常运行情景。

### （二）前端功能实现

#### 1. 前端路由

在前后端分离的开发模式下，前端路由的意义在于页面跳转的 URL 规则匹配由前端来控制，通过客户端的算力来解决页面的构建，不向后端服务器发送请求，可以缓解服务器端的压力。

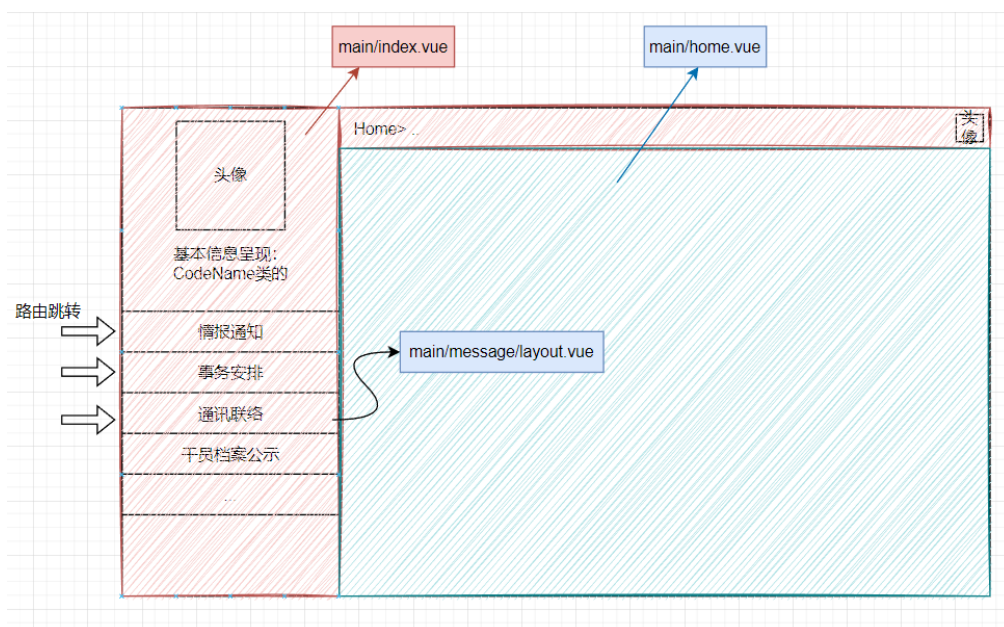
其工作方式如下：

1. 用户点击页面中路由链接，使浏览器发出请求
2. 前端路由监听此请求，解析 URL 路径，根据服务器的路由配置，返回相应信息（如 html 字符串、json 数据或图片等）

此项目中使用 Vue Router，即 Vue.js 的官方路由来进行路由配置与映射。

## 2. 嵌套路由

依托单文件组件的形式，进一步使用嵌套路由，重复利用每个页面相同的布局，使结构更加清晰



```
const routes = [
  {
    path: '/',
    redirect: '/main' // 重定向到home页面
  },
  {
    path: '/login',
    name: 'login',
    component: () => import('../views/authority/login.vue')
  },
  {
    path: '/register',
    name: 'register',
    component: () => import('../views/authority/register.vue')
  },
  {
    path: '/main',
    name: 'main',
    component: GlobalLayout,
    redirect: '/main/home',
    children: [
      {
        path: 'home', // -> '/main/home/'
        name: 'home',
        component: () => import('../views/main/home.vue')
      },
      {
        path: 'bulletin', // -> '/main/bulletin/'
        name: 'bulletin',
        component: () => import('../views/main/bulletin/announcements.vue')
      },
      {
        path: 'bulletin/singlePage/:id',
        name: 'bulletin/singlePage/:id',
        component: () => import('../views/main/bulletin/singlePage.vue')
      }
    ]
  }
]
```

### 3. 动态路由

为了实现“情报通知”模块的文章显示，这时文章内容不同但匹配的组件相同，于是就需要将这一给定的匹配模式的路由映射到同一个组件。在 Vue Router 中采用在路径中使用一个动态字段来实现，如：

```
path: 'bulletin/singlePage/:id',
name: 'bulletin/singlePage/:id',
component: () => import('../views/main/bulletin/singlePage.vue'),
```

获取其中的参数 id 方式为 `router.currentRoute.value.params.id`。

### 4. 页面刷新

前端路由也带来一个问题，即当用户对页内数据进行修改，如新建文章或修改个人信息时，使用 Vue Router 重新路由回当前页面，页面内容不会刷新。此时可以再需要刷新的标签中引入一个 key 属性，每次修改后更新 key 属性的值，vue 框架检测到 key 变化便会刷新这个标签中的内容。其实现如下：

```
1  <template>
2    <span :key="itemKey.value"></span>
3  </template>
4
5  <script lang='ts' setup>
6    import {ref} from "@vue/reactivity";
7
8    let itemKey = ref( value: 0)
9    const handleChange = () => {
10      // changes
11      itemKey.value = Math.random()
12    }
13  </script>
```

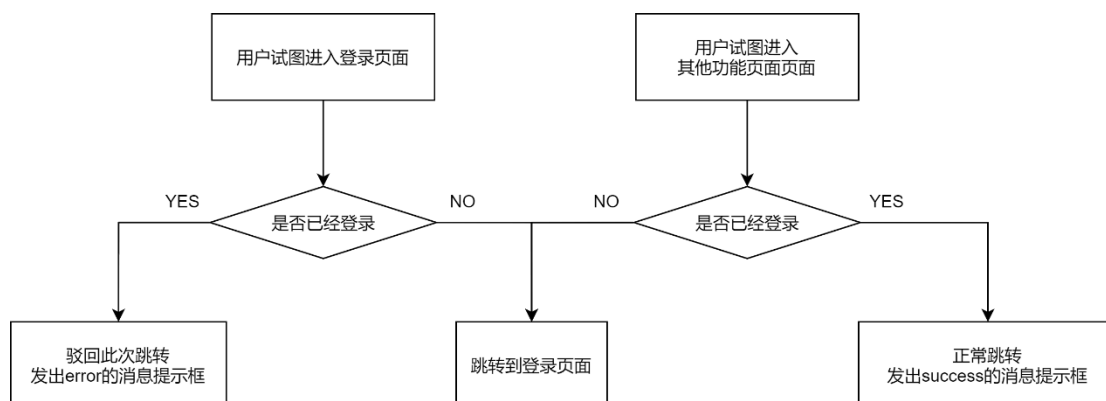
### 5. 路由守卫

路由守卫是路由在跳转前、后过程中的一些钩子函数，可以用来实现在诸如判断是否有权限进行路由跳转，如果没有则通过跳转到固定页面或取消此次操作的方式守卫路由。

此项目中，使用了全局前置守卫，实现 token 有效期内免登陆，且不能登

陆两次的功能。

其逻辑如下：



实现方式如下：

```
53 router.beforeEach( guard: async (to : RouteLocationNormalized , from : RouteLocationNormalized , next : NavigationGuardNext ) => {
54   const token = getToken()
55   if (!token && to.path !== '/login' && to.path !== '/register') {
56     // 报错，因为没有登录而且没有注册的意图
57     return next({path: '/login'})
58   }
59
60   if (token && to.path === '/login') {
61     // 已经登陆了则不能再访问登录页面防止重复登录
62     return next({path: from.path ? from.path : '/'})
63   }
64
65   if (token) {
66     // 载入当前用户的用户名、权限等信息
67     await store.dispatch( type: "get_info")
68   }
69
70   next()
71 })
```

## 6. 404 页面

为了防止用户无意中修改 url 进入了无定义的页面，在定义路由时，本项目加入了对非法页面的检测：

```
path: '/*:pathMatch(.*)*',
component: () => import('../views/404.vue'),
```

这个检测放置于所有有定义的路由之后，当解析出来的路径都无法匹配前面的规则时，跳转到写好的 404 页面，而 404 页面又会友好的把用户送首页：

```
<template #extra>
  <el-button type="primary" @click="$router.push('/')">BACK</el-button>
</template>
```

### （三） 后端功能实现

后端的搭建总体上使用了 DRF，通过对不同 view 层级的合理使用，来达到效果。下面选择对于一些功能的实现进行阐述。

#### 1. 总体框架

整体的搭建分为三个层级，分别是 APIView，GenericAPIView 搭配 Mixin 和 ModelViewSet。

对于一些最为常见的，如获取所有的用户列表、添加干员等，属于对数据库基本的增删改查等操作。对于这些操作，采用封装程度最高的 ModelViewSet 来提高代码的简洁程度。

```
98
99 class ApplicationModelView(ModelViewSet):
100     queryset = AccountApproveQueue.objects.all()
101     serializer_class = AccountApproveQueueSerializer
102     lookup_field = 'CodeName'
103     lookup_url_kwarg = 'CodeName'
104
```

对于一些功能上的变种，采取 GenericAPIView 搭配 Mixin，适当的解封，诸如登录过程，如同意用户的申请这一功能，需要将用户的从申请队列中提取信息并进行删除，经过适当的调整之后插入用户列表。故在这里采用了 GenericAPIView 和 CreateModelMixin 和 DestroyModelMxin 的方式。

```
class ApplicationOtherView(GenericAPIView, CreateModelMixin, DestroyModelMixin):
    flag = 0
    lookup_field = 'CodeName'

    def post(self, request, CodeName):
        self.flag = 0
        result = self.create(request)
        self.flag = 1
        self.destroy(request)
        return self.destroy(request)
```

而对于一些特殊的功能，则使用原生的 APIView。诸如登录步骤，通过用户名获取数据后，判断输入的密码与账户里的密码是否相同，并根据不同的结果返回不同的 status\_code。

```
def post(self, request, module):  
  
    if module == 'login':  
        data = request.data  
        try:  
            print(data)  
            user = UserAccount.objects.get(CodeName=data['CodeName'])  
            print(user)  
            if data['Password'] == user.Password:  
                token = get_jwt(user.CodeName)  
                result = {'CodeName': user.CodeName, 'token': token}  
                return Response(result)  
            else:  
                return Response(fail('密码错误'), status=403)  
        except UserAccount.DoesNotExist:  
            return Response(fail('用户不存在'), status=404)
```

后端的总架构为以上三种模型的有机结合，将前端的需求在后端进行转化，利用 DRF 框架转化为相应的对于数据库的增删改查等操作。

## 2. 图片处理

本项目对于图片的处理流程大致如下：上传一张图片（如用户头像），在数据库里记录下对应图片的名称。之后当前端需要访问该图片时，其先通过用户名访问 `user_profile` 数据项，之后通过当中记录的图片名来对存储在后端的图片进行访问。由于图片名可能存在重复，而在本项目的背景下用户名不会重复，故实际上记录的图片名称为图片名\_用户名的格式，来确保同名的记录不会访问错误。

## 3. JWT 验证

本项目采取了 JWT 验证的方式。当用户登录时，后端通过哈希算法，来为其生成一个字符串。之后前端对于该字符串予以保存。之后，当前端申请查看一些需要较高权限的内容时，其需要将 `jwt` 一同发送，通过解码的方式来获取其权限等级，与预设的权限等级进行比较，进而达到限制浏览的目的。



```
def get_jwt(username, role_data='default'):
    payload = {
        'exp': datetime.utcnow() + timedelta(seconds=3600), # 单位秒
        'iat': datetime.utcnow(),
        'data': {'username': username, 'role_data': role_data}
    }
    token = jwt.encode(payload, JWT_SECRET_KEY, algorithm='HS256')
    return str(token)
```

```
def has_permission(self, request, view):
    try:
        token = request.data['token']

        CodeName = token2name(token)
        if CodeName != request.data['CodeName']:
            return False
        test = UserAccount.objects.get(CodeName=CodeName)
        serializer = UserAccountSerializer(instance=test)
        if serializer.data['Permission'] < 2:
            return True
        else:
            return False
    except:
        return False
```

## 4. 接口设计

后端的接口设计基本上采用了了 restful 风格，即每一个 URL 代表 1 种资源,客户端使用 GET、POST、PUT、DELETE4 个表示操作方式的动词对服务端资源进行操作：GET 用来获取资源，POST 用来新建资源（也可以用于更新资源），PUT 用来更新资源，DELETE 用来删除资源。于此同时，再考虑到 restful 接口风格本身的局限性上适当加以修改，对于一些特殊的功能需求，允许在相关资源后加上动词，如@self 表示获取自己的信息，进而为前端提供多种可能接口，以满足功能要求。



```

path('login/<str:module>/', views.loginView.as_view()), # 登录界面的登录功能

path('test/<str:CodeName>/', views.testView.as_view()),

path('index/user/@self/', views.SelfView.as_view()),
path('index/user/<str:CodeName>/', views.UserDetailView.as_view()),
path('index/user/', views.UserListView.as_view()),

```

## （四） 前后端交互-跨域请求

### 1. 实现

出于浏览器的同源策略限制，两个 URL 的协议、域名、端口（如果有指定）都相同的话，浏览器才能让客户端接收服务器端传来的数据。当一个请求的 URL 中，这三者之间有一个与当前页面的 URL 不同即为跨域。所以在前后端交互的过程中，请求成为跨域请求，一行 “the server responded with a status of 500 (Internal Server Error)” 断绝了所有的来往。这时就需要去解决这个跨域问题。其解决方法有前端代理解决跨域和后端解决跨域两种。

本项目中采取的做法是前端代理解决跨域问题，即 vue 代理服务器 proxy 跨域，其方法是：通过请求本地的服务器，本地的服务器再去请求远程的服务器（后端部署接口的服务器），本地服务器再将请求回来的数据返回给浏览器。这里可见，本地服务器和浏览器之前不存在跨域，因为协议、域名、端口相同。

```

server: {
  proxy: {
    '/api': {
      target: 'http://127.0.0.1:8000/', // 服务器url
      changeOrigin: true,
      rewrite: (path :string ) => path.replace( searchValue: /\api/, replaceValue: ''),
    },
  }
}

```

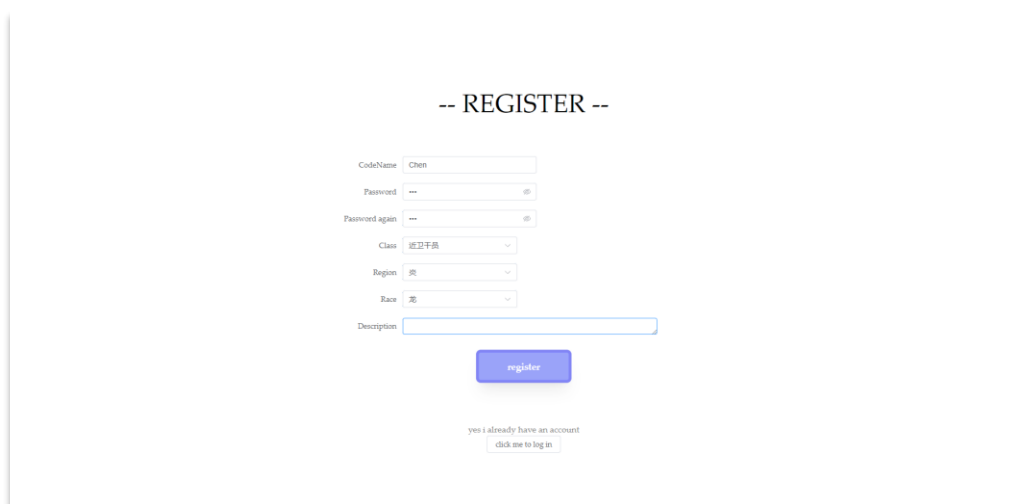
通过配置 proxy，便可以在前端通过接口与后端交互了。

### 2. 规范

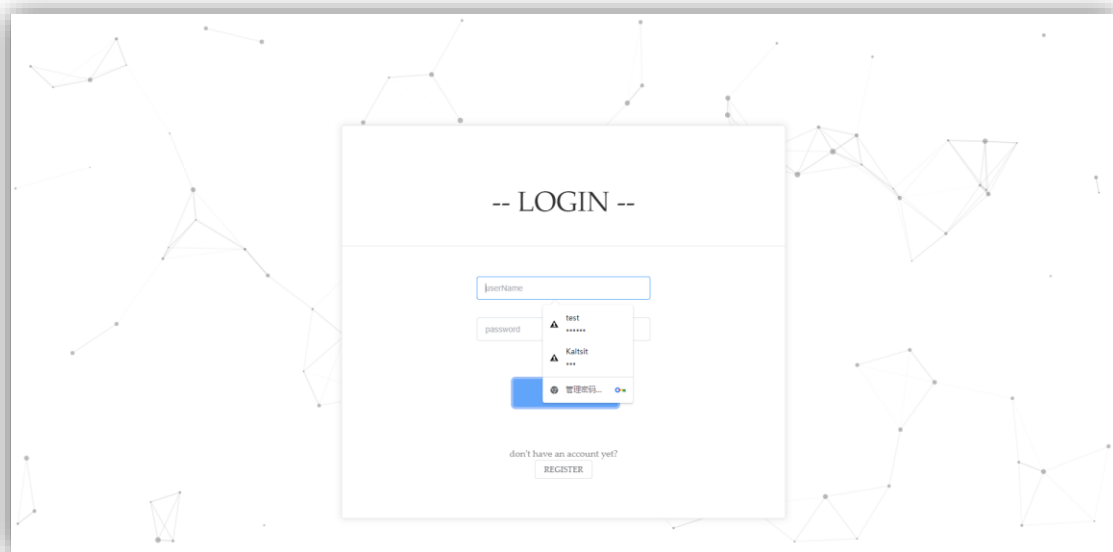
在本项目中，客户端和服务端之间的交互遵从 RESTful（Representational State Transfer，表现层状态转移）架构，在后端依托 Django 的 Rest Framework 工具进行开发。其要求为，客户端通过四个 HTTP 动词，对服务器端资源进行操作。但是 URI (Uniform Resource Identifier) 本身不包含动词，只是代表一种资源，GET 用来获取资源，POST 用来新建资源（也可以用于更新资源），PUT 用来更新资源，DELETE 用来删除资源。这样可以让客户端和服务端之间的交互在请求之间是无状态的情况下充分利用 HTTP 协议本身语义。

## 四、系统实现结果

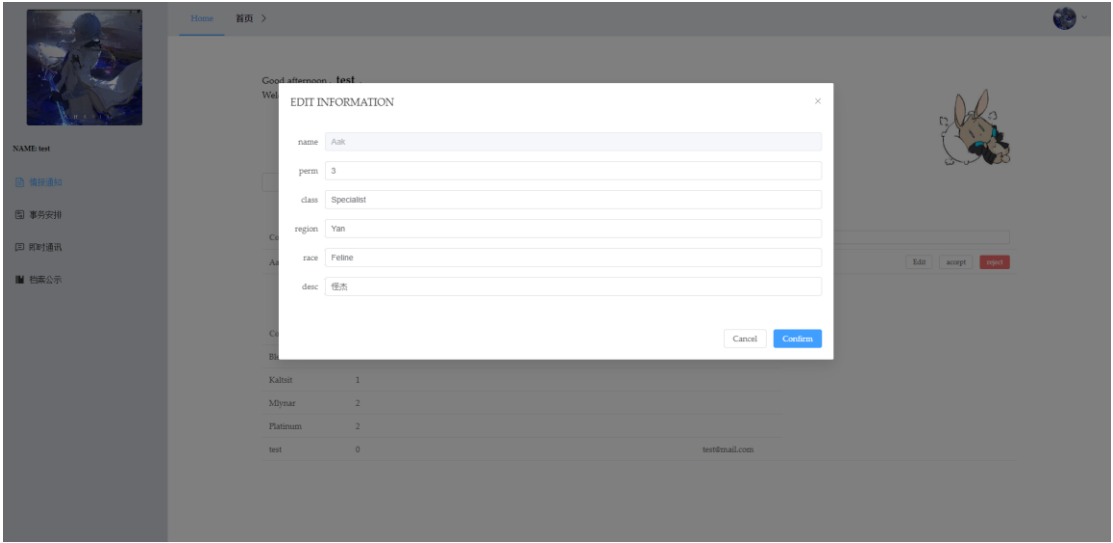
### （一）注册界面



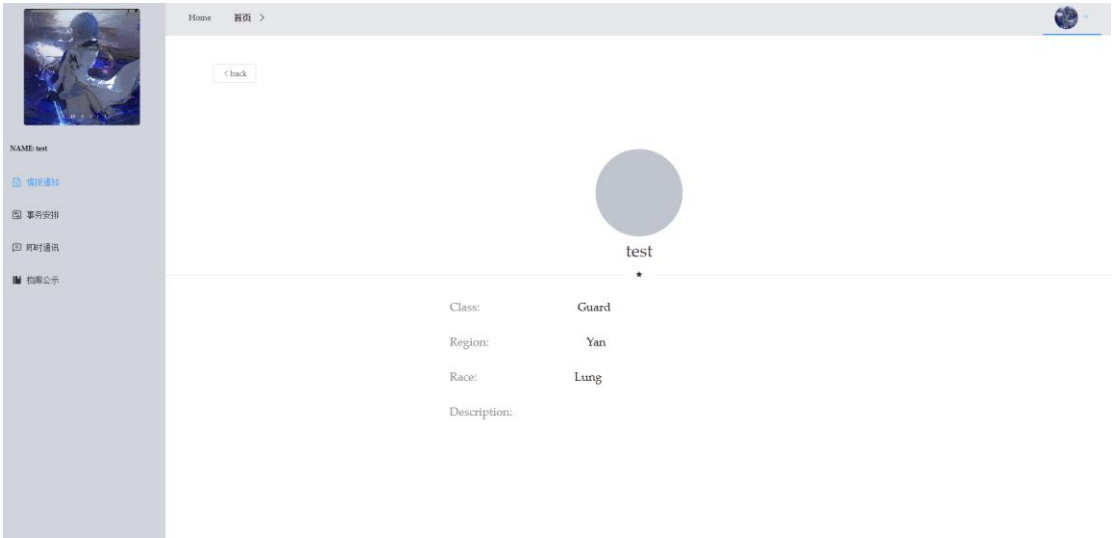
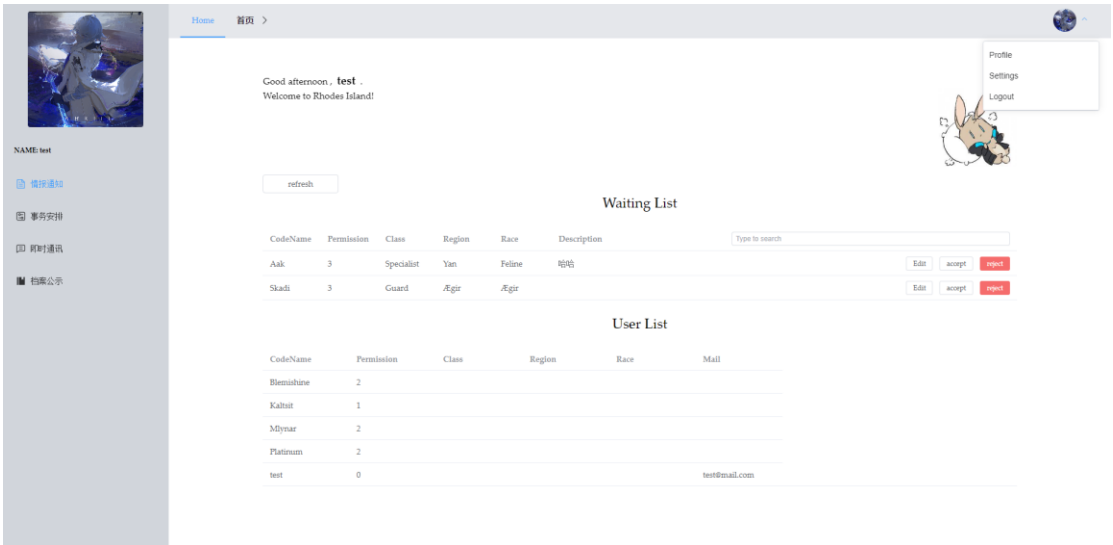
### （二）登录界面



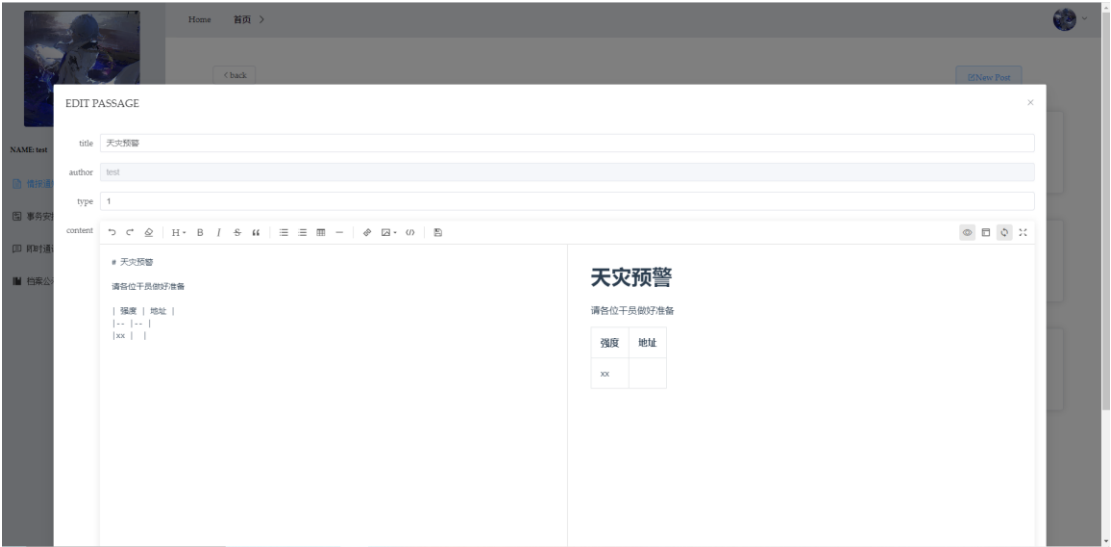
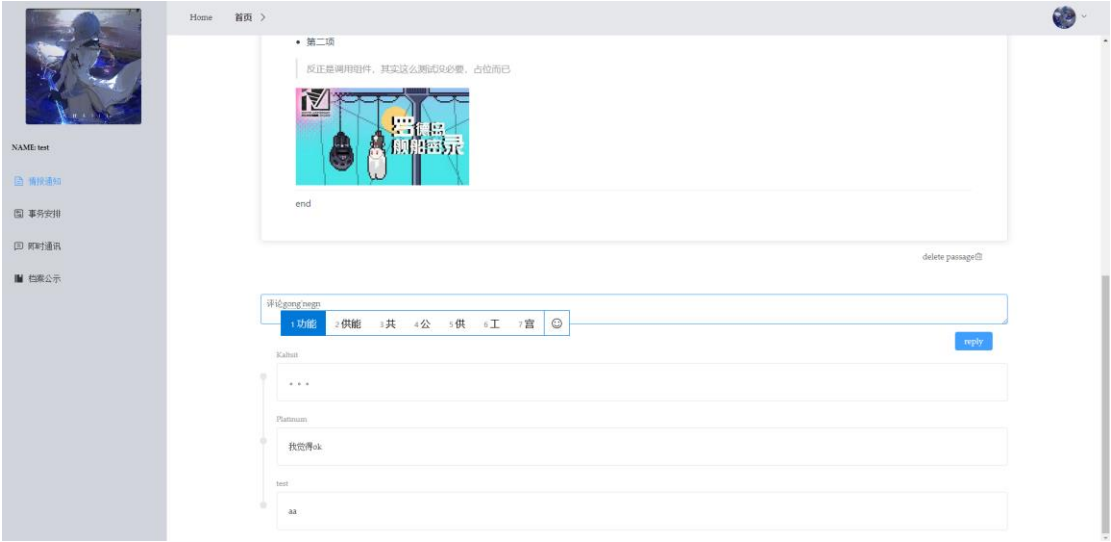
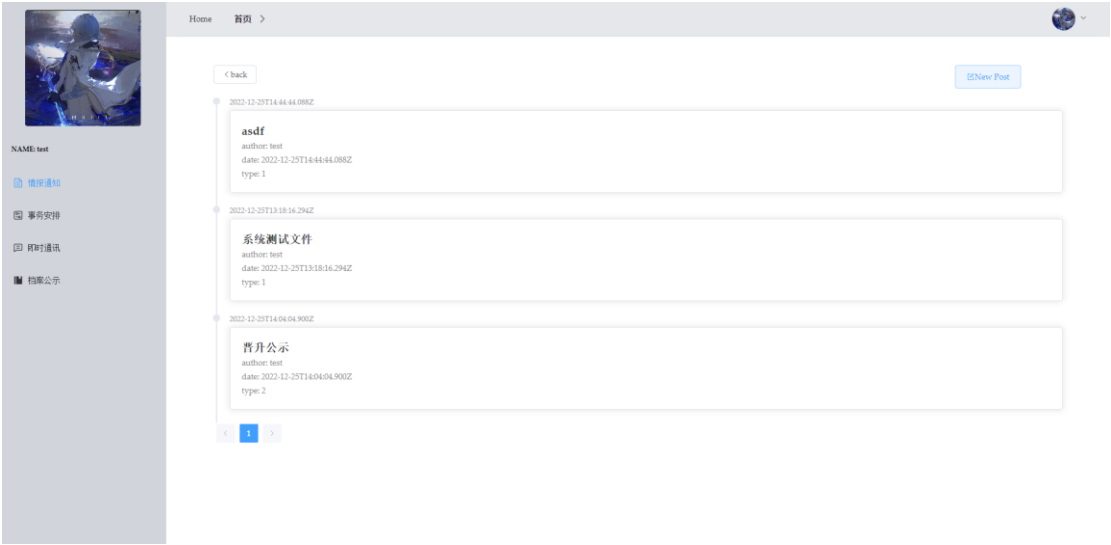
### (三) 账户审批

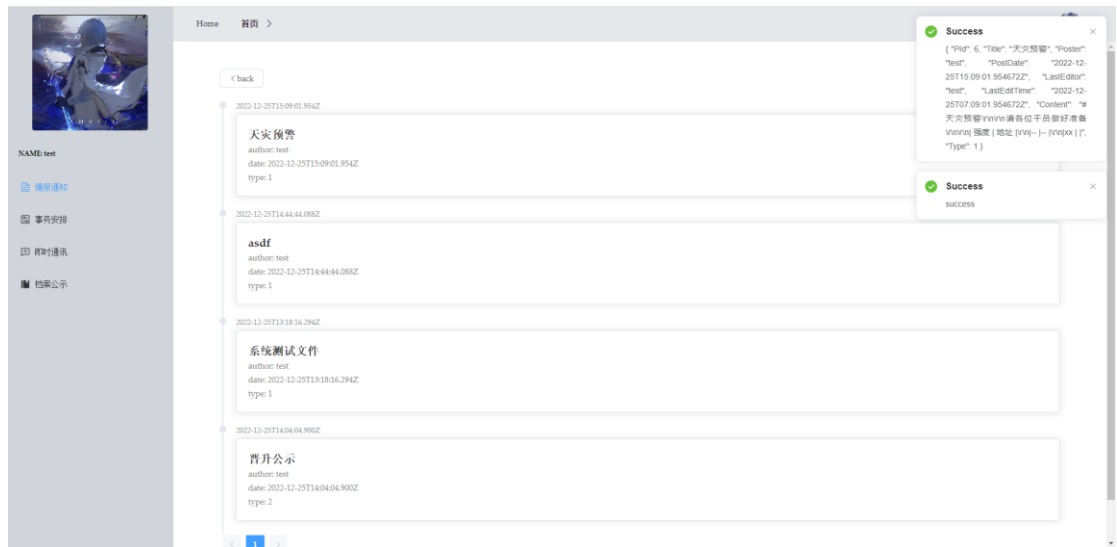


### (四) 用户主界面

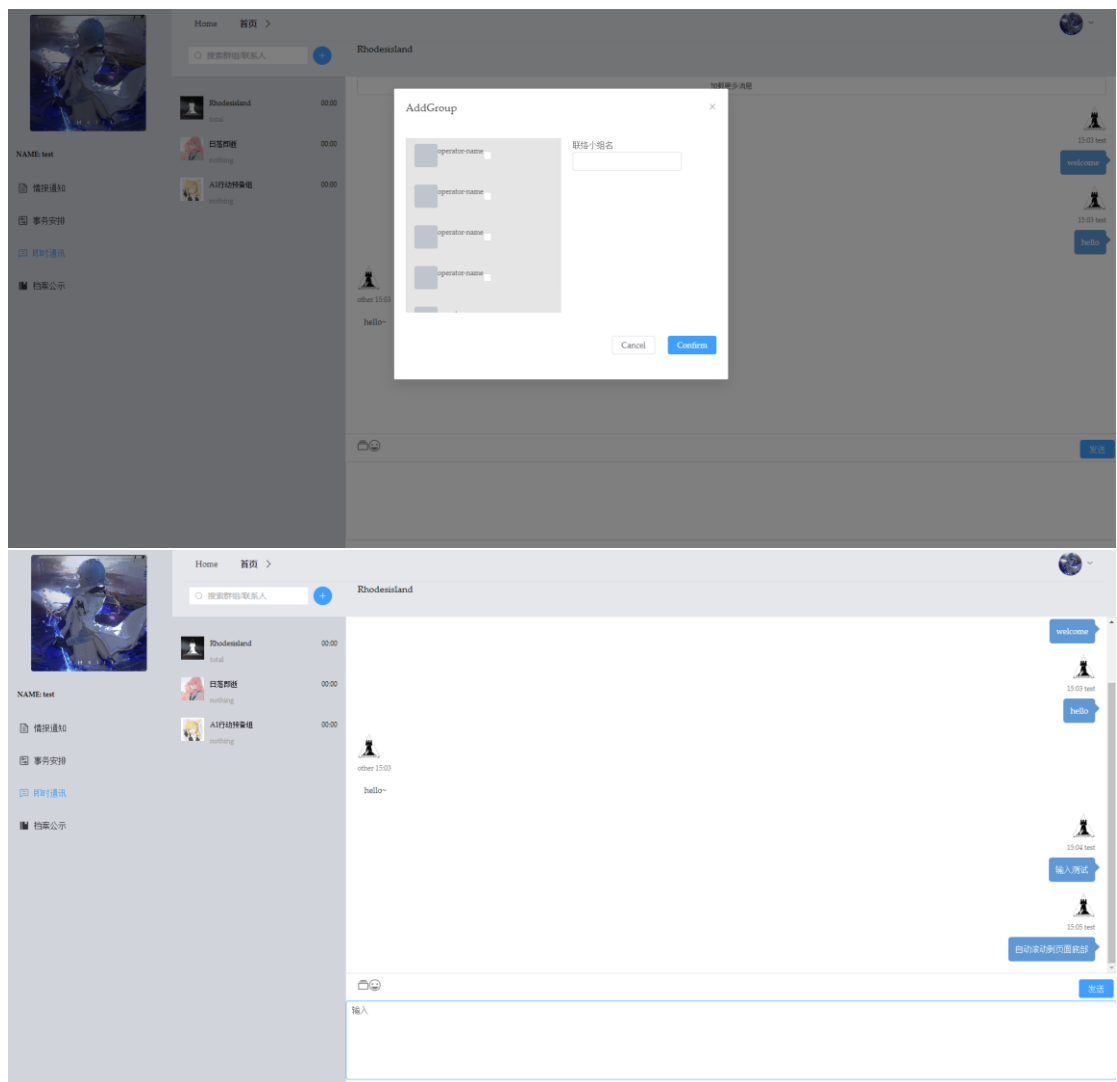


(五) 情报公示模块

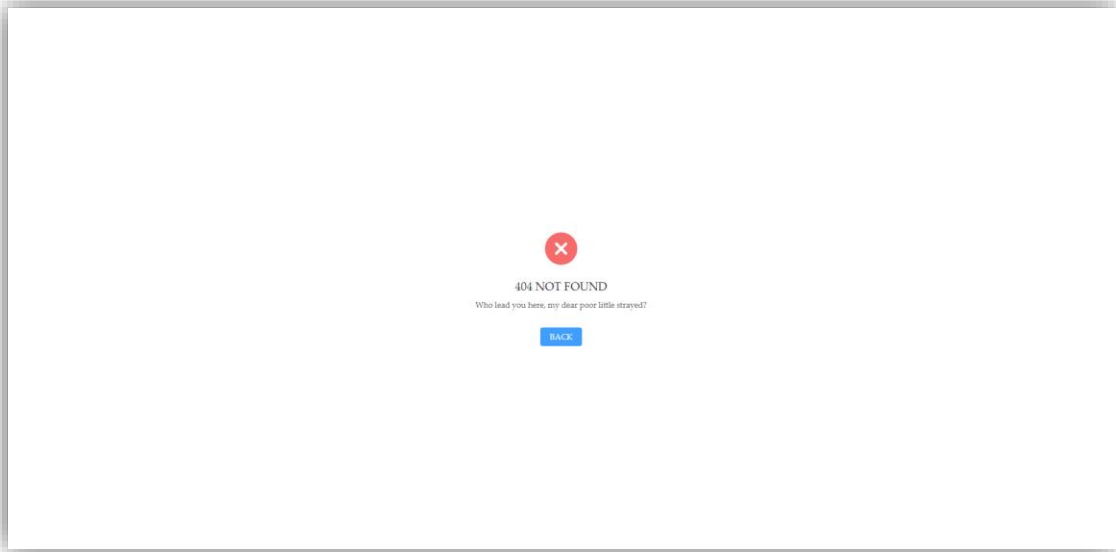




## (六) 通讯联络模块



## (七) 路由拦截



## 五、 总结

虽然开发时间紧张且项目时间管理欠妥，导致此次作业未能达到预期效果和功能，但这么一次系统级综合性强的 Web 应用课设仍然能也让我们学到了很多东  
西。为此，我们组每人分别做了如下总结。

### 夏瑞斌：

这次开发我自告奋勇地尝试做了组长，想以此提升和督促自己。项目伊始，从选择开发框架到项目基本内容搭建，从设计主题到具体内容实施，一切似乎欣欣向荣；然而到了项目中后期，随着各种课业的堆积和个人的懒惰，对项目的进程逐渐失去把控，项目进度落后；再加上本身技术能力不精（第一次接触前后端相关内容，也没勤加学习），最后也没能兑现“我尝试兜底”的豪言，十分地对不起努力的队友们。

所以除了对整个 web 项目开发有了从 0 到 1 的认识外，这次课设更给我以警示和经验。至于其他开发方面的总结，就交给两位亲爱的队友了~

### 赵雨晨：

我在这次开发中主要负责后端。在整个流程中，经历了从最开始的直接使用 mysql 语句，在到后来各种框架的使用，以及不断地向工业使用的代码作为逼近。虽说在整个过程中，出现了很多在过后看起来完全不应该出现的错误，但

是仔细思考，却也能找到原因。可以说，在数据库的小组作业让我无论是对于数据库本身的理解，亦或是对于项目开发的种种细节都有了进一步的理解。

### 俞逸洋：

此次数据库大作业让我系统的了解了前后端各自的职责以及前后端是如何进行的交互。我们采用的是前后端分离的模式进行开发，而我的主要工作集中在前端，使用 `vue` 实现了注册、登入登出、几张表的展示和公告系统。从混混沌沌的阅读 `vue` 应用的创建方式到“耐心地”和 `css` 打交道，再到对着怎么也不刷新的 `vue` 干着急，这次大作业让我学到了许多知识：前后端跨域问题的处理，如何利用控制台在浏览器里对前端进行调试（加输出、加断点），如何利用 `postman` 进行前后端联调，以及如何使用 `vue` 框架辅助编写三件套、`element plus` 等新奇的工具和插件的使用，也同时锻炼了自己调试的能力。由于前端用到的小工具很多，所以需要经常查找安装、配置方法，在此期间我也终于学会了如何阅读官方文档，在不跳步操作的情况下，确实提升了不少效率。

同时，此次大作业也让我体验到了一个快乐的小组合作。在队长的带领下，我们很快便确定了主题，并每周定期举行进度汇报和下一阶段工作内容的讨论，有时聚在一起边讨论边编码。在此期间，我们使用 `git` 进行代码版本的管理，这对于本人来说也是全新的体验，从一开始为了避免操作失误而建立过多的备份分支，到不断“继承”前面留下来的错误操作，再到现在的“熟练”掌握，横三顺四的 `git` 分支图谱目睹了这一切。非常感谢队友们的付出！也很感谢他们对我（因为迫切的想测试前端功能正确性而胡乱）篡改后端函数等逾矩行为的包容。