



常德职业技术学院
Changde Vocational Technical College

毕 业 设 计 成 果

基于 Django 和 vue3 的综合个人网站管理系统
-豆瓣电影分析与个人网站的设计与实现

系 名 称	机电与信息工程系
专 业 名 称	软件技术
学 生 姓 名	袁越
班 级	2306
学 号	202319180635
校内指导教师	
企业指导教师	
完 成 时 间	2026 年 5 月

目录

- 1 设计概述 1
 - 1.1 设计目的与意义 1
 - 1.1.1 开发平台 1
 - 1.1.2 运行环境 1
- 2 豆瓣电影分析与个人网站 2
 - 2.1 主要功能 2
 - 2.2 用例图 3
- 3 豆瓣电影分析与个人网站的概要设计 4
 - 3.1 功能模块设计 4
 - 3.2 数据库设计 5
- 4 豆瓣电影分析与个人网站的详细设计与实现 10
 - 4.1 电影分析模块 10
 - 4.1.1 电影展示 10
 - 4.1.2 数据可视化 12
 - 4.2 用户管理模块 14
 - 4.2.1 用户登录注册 14
 - 4.2.2 JWT 登录认证 18
 - 4.3 留言板模块 19
 - 4.3.1 留言管理 19
 - 4.3.2 留言 API 20
 - 4.4 系统管理模块 21
 - 4.4.1 站点配置 21
 - 4.4.2 后台管理 23
- 5 总结/结语 27
- 参考资料 28

1 设计概述

1.1 设计目的与意义

本项目旨在通过 Django 框架构建集数据展示与个人服务于一体的综合性 Web 系统。通过豆瓣 Top250 电影数据的爬取、反扒、分析与可视化，为用户提供直观的电影推荐与影史趋势洞察，还满足影视爱好者的数据查询的需求；同时，个人网站模块集成用户认证、留言互动等功能，让任何人都可以和我互动，增强用户参与感与交互体验。前后端分离架构提升开发效率与系统扩展性。

本项目的开发与实现，不仅能够将 Django 框架、网络爬虫、数据处理、前后端分离等技术进行综合实践与应用，形成一套完整的 Web 系统开发流程，提升技术落地与工程化实践能力；还能为影视爱好者提供一个集数据查询、观影参考、互动交流于一体的专属平台，满足用户在电影信息获取与社交互动方面的双重需求。

1.1.1 开发平台

运行平台：Windows 10/11、Linux 发行版、macOS 12 及以上

数据库管理软件：Navicat（MySQL 图形化工具）

开发工具：VS Code、PyCharm

版本控制系统：Git、GitHub

1.1.2 运行环境

关系型数据库：MySQL 8.0+

兼容浏览器：Chrome80+、Firefox75+、Edge80+

Python 环境：Python 3.8-3.12、pip 22.0+、venv（虚拟环境工具）

2 豆瓣电影分析与个人网站

2.1 主要功能

本系统面向不同类型用户提供电影信息展示、用户互动及后台管理等服务，旨在打造一个基于数据分析的个性化个人网站。系统主要面向游客、注册用户和系统管理员三类角色，核心功能如下：

（一）电影信息展示与分析

游客无需登录即可访问电影列表页面，支持按关键词搜索电影及查看数据可视化图表。系统基于豆瓣 Top250 数据，提供多维度统计图形，如评分分布、国家年代分布、标签词云等，辅助用户了解电影全貌。

用户注册与登录认证

（二）注册用户需通过注册功能提交个人信息，系统校验后创建账户。用户登录成功后，系统返回身份验证令牌（JWT），用于后续操作权限验证，并允许访问留言功能和个人中心。

（三）留言互动功能。

注册用户登录后可访问留言板模块，支持发布留言、浏览个人留言历史。系统为每条留言记录用户身份、留言时间等信息，确保互动行为的可追溯性和内容合规性。

（四）后台管理功能。

系统管理员登录后可进入后台管理界面，具备配置站点信息、维护导航项、导入/修改电影数据、审核或删除留言等管理权限。管理员通过专属接口访问系统，系统对其操作权限进行严格验证，确保数据安全。

（五）角色权限隔离与访问控制。

系统基于用户角色划分权限，游客仅可浏览公开页面；注册用户在登录状态下可进行交互操作；管理员拥有最高操作权限。系统通过中间件与权限机制保障不同角色的访问边界，提升系统安全性与交互体验。

2.2 用例图

根据需求分析，设计系统用例图如图 2-1 所示。

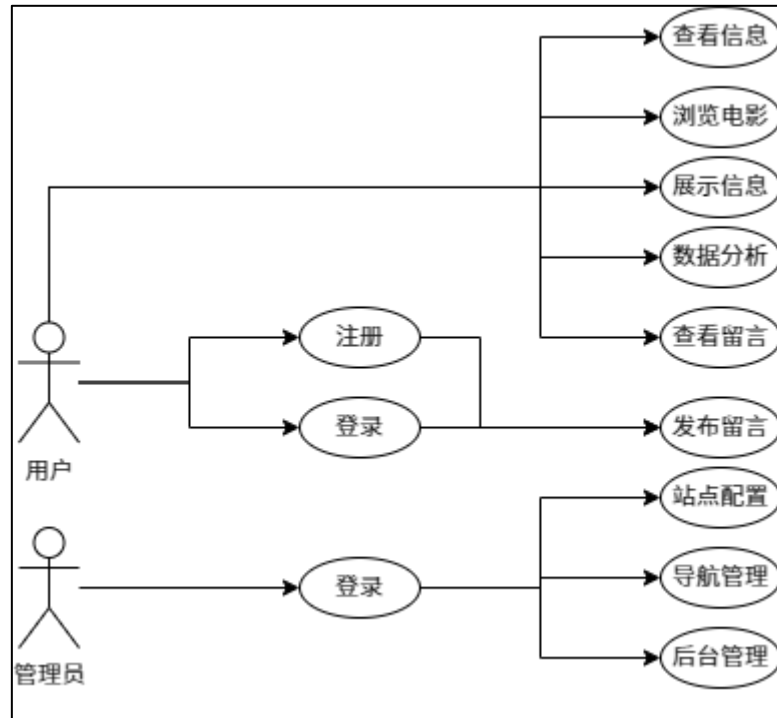


图 2-1 用例图

3 豆瓣电影分析与个人网站的概要设计

3.1 功能模块设计

系统采用 Django + Vue3 前后端分离架构，功能模块设计如下：电影分析模块负责从豆瓣 Top250 抓取电影数据，并提供关键词搜索、分页浏览及基于 pyecharts/ECharts 的多维度可视化图表展示；用户管理模块基于 JWT 实现注册、登录及权限划分，将访问者分为游客、注册用户和管理员三类，并对用户密码进行加密存储；留言板模块通过 RESTful API 为登录用户提供发布留言和查看历史留言功能，所有留言均由管理员在后台审核后公开；系统管理模块采用单例模式，管理员在专属后台界面中维护站点全局配置、导航项以及电影数据，一切操作均经过权限校验，确保系统配置一致性与数据安全。

系统功能模块如图 2-1 所示。

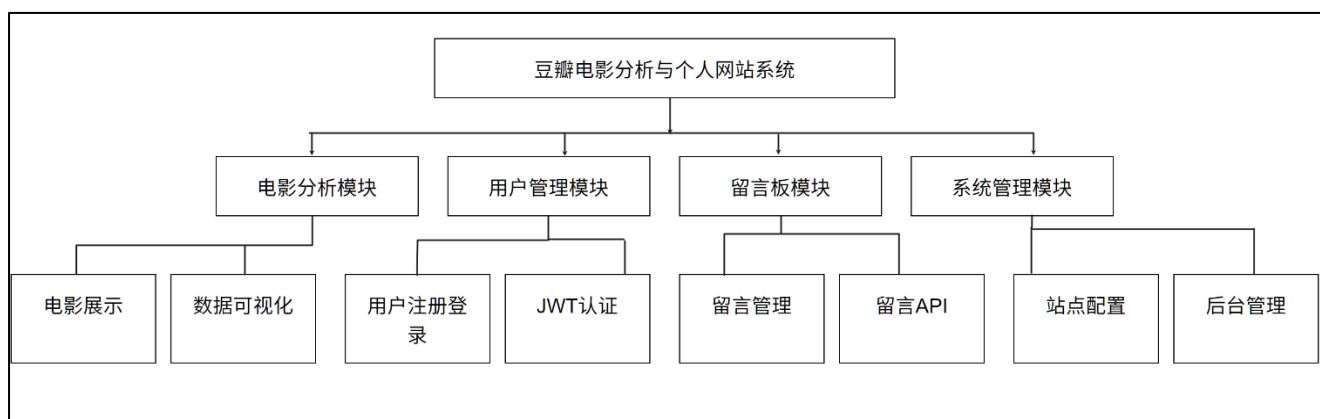


图 3-1 功能模块图

3.2 数据库设计

围绕用户认证、业务数据、系统管理及配置管理四大模块设计数据库：

(一) 用户认证：

用户表（auth_user）存储账号信息（用户名唯一、密码加密）。

(二) 业务数据：

电影表（movies_movie）以豆瓣 ID 为主键存储影片信息；留言表（lyb）关联用户 ID 记录留言。

(三) 系统管理：

导航配置表（lyb_navitem）存储导航项文本与链，站点配置表（site_config_siteprofile）。

E-R 图如图 3-2 所示。



图 3-2 数据库设计图

数据表包含模块如下

用户认证模块: auth_user;

业务数据模块: movies_movie、d_lyb;

配置管理模块: lyb_navitem、site_config_siteprofile。

用户认证表 authuser: 该表用于描述用户认证相关信息, 包含字段 id、password、lastlogin、issuperuser、username、firstname、lastname、email、isstaff、isactive、datejoined。其中 id 是主键, 所有字段均为必填。各字段说明如下:

表 1 用户表 auth_user

字段名	数据类型	是否必填	说明
id	int(11)	是	主键
password	varchar(128)	是	用户密码
lastlogin	datetime(6)	是	最后一次登录的时间
issuperuser	tinyint(1)	是	是否为超级用户
username	varchar(150)	是	用户名
firstname	varchar(150)	是	用户名字（名）
lastname	varchar(150)	是	用户姓氏（姓）
email	varchar(254)	是	用户电子邮箱
isstaff	tinyint(1)	是	是否为员工
isactive	tinyint(1)	是	用户是否激活
date_joined	datetime(6)	是	用户账号创建时间

电影业务数据表 `movies_movie`: 此表用于存储电影相关业务数据, 包含字段 `id`、`doubanid`、`title`、`img`、`href`、`quote`、`score`、`year`、`countries`、`genres`。其中 `id` 是主键, 所有字段均为必填。各字段说明如下:

表 2 电影业务数据表 `movies_movie`

字段名	数据类型	是否必填	说明
<code>id</code>	<code>bigint(20)</code>	是	主键
<code>douban_id</code>	<code>varchar(10)</code>	是	豆瓣电影 ID
<code>title</code>	<code>varchar(255)</code>	是	电影标题
<code>img</code>	<code>varchar(200)</code>	是	电影图片地址
<code>href</code>	<code>varchar(200)</code>	是	电影链接
<code>quote</code>	<code>longtext</code>	是	电影引言
<code>score</code>	<code>double</code>	是	电影评分
<code>year</code>	<code>varchar(10)</code>	是	电影年份
<code>countries</code>	<code>varchar(255)</code>	是	电影制作国家
<code>genres</code>	<code>varchar(255)</code>	是	电影类型

留言表 d_lyb: 该表用于记录留言信息, 包含字段 id、title、author、content、posttime。其中 id 是主键, 所有字段均为必填。各字段说明如下:

表 3 留言表 d_lyb

字段名	数据类型	是否必填	说明
id	bigint(20)	是	主键
title	varchar(200)	是	留言标题
author	varchar(200)	是	留言作者
content	longtext	是	留言内容
posttime	datetime(6)	是	留言发布时间

导航配置表 lyb_navitem: 此表用于配置导航相关信息, 包含字段 id、text、url。其中 id 是主键, 所有字段均为必填。各字段说明如下:

表 4 导航配置表 lyb_navitem

字段名	数据类型	是否必填	说明
id	bigint(20)	是	主键
text	varchar(100)	是	导航文本
url	varchar(200)	是	导航链接

网站配置表 site_config_siteprofile: 该表用于存储网站配置信息, 包含字段 id、profiledata、lastupdated。其中 id 是主键, 所有字段均为必填。各字段说明如下:

表 5 网站配置表 site_config_siteprofile

字段名	数据类型	是否必填	说明
id	bigint(20)	是	主键
profiledata	json	是	网站配置数据
lastupdated	datetime(6)	是	最后更新时间

4 豆瓣电影分析与个人网站的详细设计与实现

4.1 电影分析模块

电影分析模块是本系统的核心功能模块之一，主要围绕豆瓣 Top250 电影数据展开，集数据展示、筛选查询与多维度数据分析于一体。该模块通过 Django 后端对电影数据进行统一管理和处理，并在前端页面中以列表和图表的形式进行直观呈现。

4.1.1 电影展示

电影展示模块主要用于对豆瓣 Top250 电影数据进行统一展示。该模块通过后端接口获取电影数据，并在前端页面中以表格形式进行渲染，支持分页浏览、条件搜索以及电影详情跳转等功能，从而提升用户对电影信息的浏览体验。

电影展示的前端页面效果如图 4_1 所示。

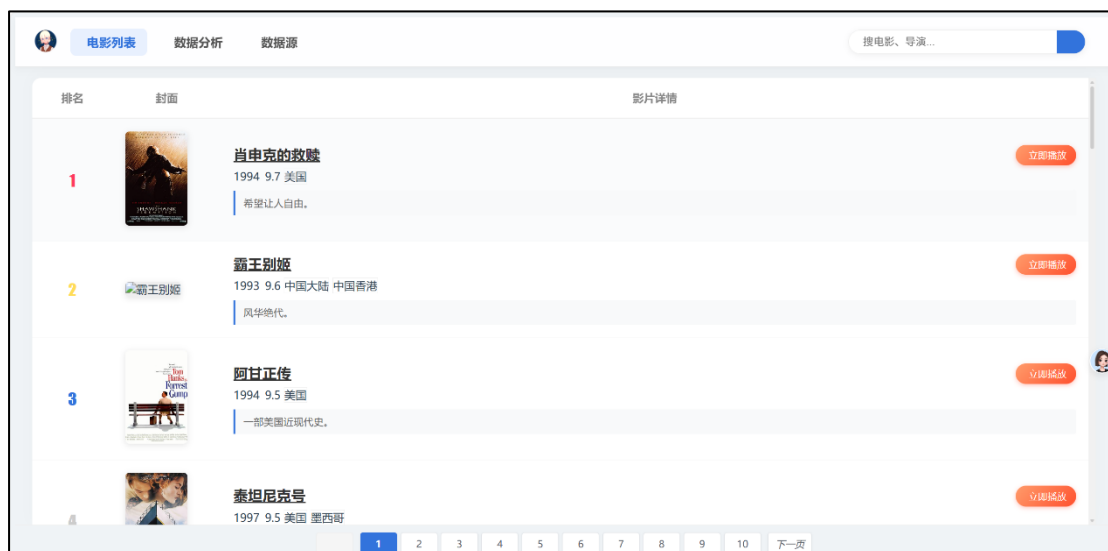


图 4_1 电影展示前端页面

电影展示页面的前端核心代码如图 4_2 所示。

```

<!-- 导航栏 -->
<div class="custom-navbar">

<!-- 内容区 -->
<div class="content-area">
  <div class="table-scroll-wrapper">
    <table class="table custom-table">
      <thead>
        <tr>
          <th style="width: 88px; text-align: center;">排名</th>
          <th style="width: 120px;">封面</th>
          <th>影片详情</th>
        </tr>
      </thead>
      <tbody>
        {% for movie in movies %}
          <tr class="movie-row">
            <!-- 排名 -->
            <td style="text-align: center;">
              <span style="font-family: 'Impact'; font-size: 1.5rem;
                color: {% if movie.id == '1' %}#ff3860{% elif movie.id == '2' %}#ffdd57{% elif movie.id == '3' %}#3273dc{% else %}#ccc{% endif %};">
                {{ movie.id }}
              </span>
            </td>

```

图 4_2 电影展示前端核心代码

电影展示功能对应的后端核心代码如图 4_3 所示。

```

def index(request):
    # 获取搜索查询参数
    search_query = request.GET.get('query','')

    # 直接从数据库查询
    if search_query:
        movies = Movie.objects.filter(title__icontains=search_query)
    else:
        movies = Movie.objects.all()

    # 将数据库对象转换为与原JSON 格式兼容的字典
    data_list = []
    for movie in movies:
        movie_dict = {
            'id': movie.douban_id,
            'title': movie.title,
            'img': movie.img,
            'href': movie.href,
            'quote': movie.quote,
            'score': movie.score,
            'tags': {
                '时间': movie.year,
                '国家': movie.countries.split(',') if movie.countries else [],
                '属性': movie.genres.split(',') if movie.genres else []
            }
        }
        data_list.append(movie_dict)

```

图 4_3 电影展示后端核心代码

4.1.2 数据可视化

为了更加直观地分析豆瓣电影数据，系统在电影分析模块中引入了数据可视化功能。该功能基于 pyecharts 与 ECharts 实现，通过多种图表形式对电影评分、国家分布、年代分布等信息进行统计分析。

数据可视化页面整体展示效果如图 4_4 所示。

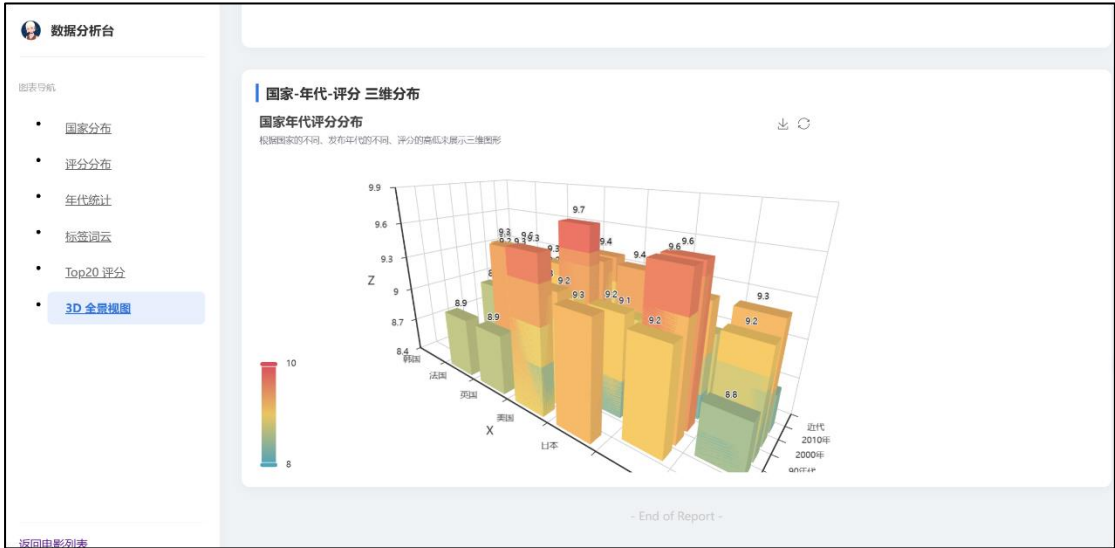


图 4_4 数据可视化页面展示

数据可视化页面的前端核心代码如图 4_5 所示。

```

3      <html lang="zh-CN">
121    <body>
123    <div class="dashboard-container">
157      <main class="main-content" id="main-scroll">
159      <h1 class="title is-4" style="color:#4a4a4a;">可视化报表</h1>
160      <p class="subtitle is-6" style="color:#888; margin-bottom: 30px;" ...>
163
164      <!-- 1. 国家分布 -->
165      <div id="pie_chart" class="chart-card">
166        <div class="chart-title">各国家/地区电影占比</div>
167        <div class="chart-box">
168          {{ pie_chart_html|safe }}
169        </div>
170      </div>
171
172      <!-- 2. 评分分布 -->
173      <div id="scatter_chart_html" class="chart-card">
174        <div class="chart-title">电影评分散点分布</div>
175        <div class="chart-box">
176          {{ scatter_chart_html|safe }}
177        </div>
178      </div>
179
180      <!-- 3. 年代统计 -->
181      <div id="bar_year_chart" class="chart-card">
182        <div class="chart-title">各年代上榜电影数量</div>
183        <div class="chart-box">
184          {{ bar_year_chart_html|safe }}
185          {{ bar_year_chart_html_1|safe }}
186        </div>
187      </div>

```

图 4_5 数据可视化前端核心代码

数据可视化功能对应的后端核心代码如图 4_6 所示。

```

126 def chart(request):
144     tags = movie["tags"]["属性"]
145     years = movie["tags"]["时间"]
146     countries = movie["tags"]["国家"]
147     scores = movie["score"]
148     for tag in tags:
149         tag_count[tag] += 1
150
151     for country in countries:
152         country_count[country] += 1
153
154     score_count[scores] += 1
155     if (count < 20):
156         top20_movie.append([movie["title"], movie["score"]])
157         count += 1
158     # year_score
159     years = re.findall(pattern: r'\d+', years)[0] if years and re.findall(pattern: r'\d+', years) else "0"
160     year_count[years] += 1
161     year_score.append([years, movie["score"]])
162
163     # 确保countries是有效的列表, 并且至少有一个元素
164     if countries and isinstance(countries, list) and len(countries) > 0:
165         country_year_count.append((countries, years, scores))
166     else:
167         # 如果countries为空或无效, 使用一个默认值
168         country_year_count.append(["未知", years, scores])
169
170     # 将结果转换为普通字典
171     tag_count = dict(tag_count)
172     country_count = dict(country_count)
173     year_count = dict(year_count)
174     score_count = dict(score_count)

```

图 4_6 数据可视化后端核心代码

4.2 用户管理模块

用户管理模块通过连接 Django 后端数据库, 实现用户信息 (如用户名、密码等) 的存储与管理。该模块支持用户注册与登录功能, 并通过前后端交互完成用户身份验证, 从而为系统的个性化功能提供基础支持。

4.2.1 用户登录注册

为了实现统一、美观的界面风格, 系统在用户登录与注册页面中采用了 MaterialDesign 3 的设计理念, 以纯色、圆角和阴影为核心设计元素, 使整体界面简洁大方、交互友好。

用户登录页面的展示效果如图 4_7 所示。



图 4_7 登录页面

登录页面后端核心代码如图 4_8 所示。

```
# 用户登录接口
@api_view(['POST'])
@permission_classes([permissions.AllowAny]) # 允许未登录用户访问
def login_user(request):
    """
    用户登录接口，验证用户名和密码，返回 JWT Token。
    """
    username = request.data.get('username')
    password = request.data.get('password')

    # 检查用户名和密码是否为空
    if not username or not password:
        return Response({'error': '用户名和密码不能为空'}, status=status.HTTP_400_BAD_REQUEST)

    # 验证用户身份
    user = authenticate(username=username, password=password)
    if user is None:
        return Response({'error': '用户名或密码错误'}, status=status.HTTP_401_UNAUTHORIZED)

    # 生成 JWT Token
    refresh = RefreshToken.for_user(user)
    return Response({
        'token': str(refresh.access_token), # 访问令牌
        'refresh': str(refresh),          # 刷新令牌
        'username': user.username
    })
```

图 4_8 登录页面后端核心代码

登录页面前端核心代码如图 4_9 所示。

```

<template>
  <div class="login-container">
    </div>

    <div class="auth-box">
      <h2>登录</h2>
      <form @submit.prevent="handleLogin">
        <div class="form-group">
          <label for="username">用户名</label>
          <input
            type="text"
            id="username"
            v-model="username"
            required
            placeholder="请输入用户名"
            :class="{ 'is-invalid': usernameError }"
            @input="usernameError = ''"
          >
          <div class="invalid-feedback" v-if="usernameError">
            {{ usernameError }}
          </div>
        </div>
        <div class="form-group">...
      </div>
      <div class="alert alert-danger" v-if="error" role="alert">
        {{ error }}
      </div>
      <button type="submit" class="auth-button">登录</button>
    </form>
  </div>
</template>

```

图 4_9 登录页面前端核心代码

用户注册页面的展示效果如图 4_10 所示。

The image shows a registration form titled "注册" (Register) in purple. It contains three input fields: "用户名" (Username) with placeholder "请输入用户名", "邮箱" (Email) with placeholder "请输入邮箱", and "密码" (Password) with placeholder "请输入密码". Below these is a purple "注册" (Register) button. At the bottom, it says "已有账号? [立即登录](#)" (Already have an account? [Login immediately](#)).

图 4_10 注册页面

注册页面的前端核心代码如图 4_11 所示。

```
<div class="auth-box">
  <h2>注册</h2>
  <form @submit.prevent="handleRegister">
    <div class="form-group" ...>
      <div class="form-group">
        <label for="email">邮箱</label>
        <input
          type="email"
          id="email"
          v-model="email"
          required
          placeholder="请输入邮箱"
        >
      </div>
      <div class="form-group" ...>
        <div class="alert alert-danger" v-if="error" role="alert">
          {{ error }}
        </div>
        <button type="submit" class="auth-button">注册</button>
      </div>
    </form>
    <p class="auth-link">
      已有账号? <a href="/login">立即登录</a>
    </p>
  </div>
```

图 4_11 注册页面前端核心代码

注册页面对应的后端核心代码如图 4_12 所示

```
# 用户注册接口
2 usages
@api_view(['POST'])
@permission_classes([permissions.AllowAny]) # 允许未登录用户访问
def register_user(request):
    """
    用户注册接口，接收用户名、密码和邮箱，创建新用户。
    """
    username = request.data.get('username')
    password = request.data.get('password')
    email = request.data.get('email')

    # 检查用户名和密码是否为空
    if not username or not password:
        return Response( data: {'error': '用户名和密码不能为空'}, status=status.HTTP_400_BAD_REQUEST)

    # 检查用户名是否存在
    if User.objects.filter(username=username).exists():
        return Response( data: {'error': '用户名已存在'}, status=status.HTTP_400_BAD_REQUEST)

    # 创建新用户
    user = User.objects.create_user(username=username, password=password, email=email)
    return Response( data: {'message': '注册成功'}, status=status.HTTP_201_CREATED)
```

图 4_12 注册页面后端核心代码

4.2.2 JWT 登录认证

为了提高系统的安全性与扩展性，用户认证模块采用了基于 JWT（JSON Web Token）的无状态认证方案。客户端在登录成功后获取令牌，并在后续请求中携带该令牌，服务端通过验证令牌完成用户身份校验。

JWT 认证的后端核心代码如图 4_13 所示。

```
def login_user(request):
    """
    用户登录接口，验证用户名和密码，返回 JWT Token。
    """
    username = request.data.get('username')
    password = request.data.get('password')

    # 检查用户名和密码是否为空
    if not username or not password:
        return Response(data={'error': '用户名和密码不能为空'}, status=status.HTTP_400_BAD_REQUEST)

    # 验证用户身份
    user = authenticate(username=username, password=password)
    if user is None:
        return Response(data={'error': '用户名或密码错误'}, status=status.HTTP_401_UNAUTHORIZED)

    # 生成 JWT Token
    refresh = RefreshToken.for_user(user)
    return Response({
        'token': str(refresh.access_token), # 访问令牌
        'refresh': str(refresh), # 刷新令牌
        'username': user.username
    })
```

图 4_13 JWT 认证后端核心代码

4.3 留言板模块

留言板模块为用户提供一个开放的互动平台。系统允许未登录用户匿名留言，同时已登录用户的留言可自动关联其用户名。用户可以对自己的留言进行修改，管理员账户则具备删除不良留言的权限，从而保证留言内容的规范性。

4.3.1 留言管理

我打算设计一个人人可以使用的留言板，他有精美的样式，不用强制输入用户名，从而实现人人可用。而已登录用户的留言，也可以自动获取到用户名。人人都可以留言和更改自己的留言，并且我还设置了管理员账户一个可以删除不良留言的功能。

留言板页面整体展示效果如图 4_14 所示。

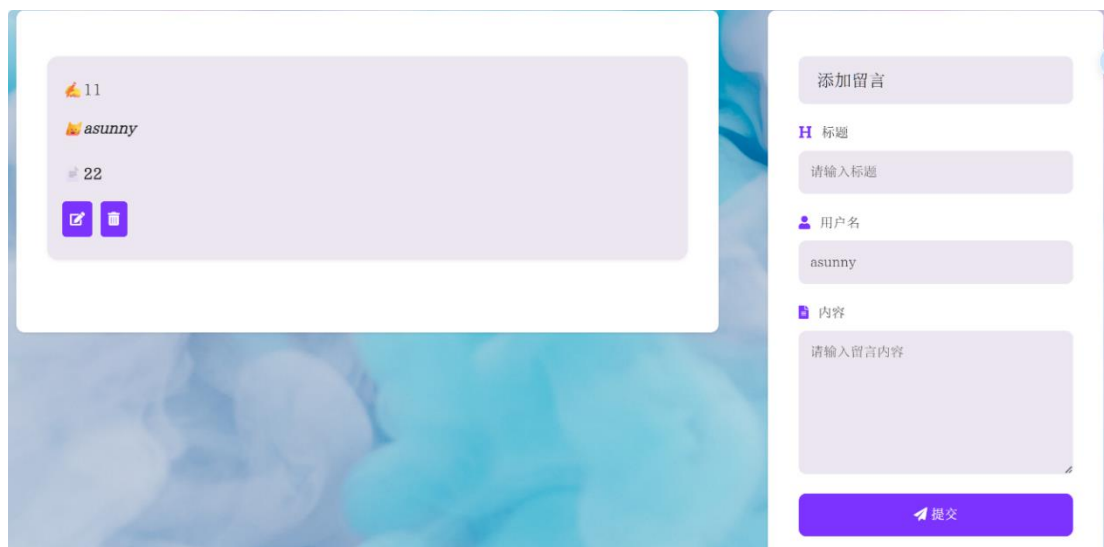


图 4_14 留言板块页面

留言板页面的前端核心代码如图 4_15 所示。

```
// 删除留言
const deletelyb = (item) => {    "deletelyb": Unknown word.
  let relativeDeleteUrl = item.url;
  if (typeof relativeDeleteUrl === 'string' && relativeDeleteUrl.startsWith('/api/is/')) {
    relativeDeleteUrl = relativeDeleteUrl.substring('/api/is'.length);
  }
  axios.delete(relativeDeleteUrl).then(res => {    "axios": Unknown word.
    getLyb()
  }).catch(err => {
    console.log(err)
  })
}

const savelyb = () => {    "savelyb": Unknown word.
  let newdata = {    "newdata": Unknown word.
    title: state.lyb.title,
    author: state.lyb.author,
    content: state.lyb.content
  }
  if(state.lyb.url==""){
    // 新增
    axios.post("lyb/",newdata).then(res => {    "axios": Unknown word.
      getLyb()
    }).catch(err => {
      console.log(err)
    })
  }
}
```

图 4_15 留言板页面前端核心代码

4.3.2 留言 API

为了实现留言功能的前后端分离，系统为留言板模块设计了相应的 RESTful

API 接口。

留言板模块的后端核心代码如图 4_16 所示。

```
# 留言板视图集，提供增删改查接口
2个用法
class LybrViewSet(viewsets.ModelViewSet):
    """
    提供留言板（lyb）模型的 API 接口，支持查看、创建、修改、删除留言。
    """
    queryset = lyb.objects.all().order_by('-posttime') # 获取所有留言，按发布时间倒序排列
    serializer_class = LybSerializer # 指定序列化器
    permission_classes = [permissions.AllowAny] # 允许任何人访问（不需要登录）
```

图 4_16 留言板块页面后端核心代码

留言板模块的数据库字段定义如图 4_17 所示。

```
# Create your models here.

class lyb(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=200)
    content = models.TextField()
    posttime = models.DateTimeField(auto_now_add=True)

    class Meta:
        db_table = "d_lyb" # 留言板
```

图 4_17 留言板块页面后端字段代码

4.4 系统管理模块

系统管理模块主要用于对网站的整体运行状态和核心配置进行统一管理，是保障系统稳定性与可维护性的重要组成部分。该模块面向管理员用户开放，通过后台管理界面对站点配置、电影数据展示规则以及系统账户等内容进行集中管理。

4.4.1 站点配置

站点配置模块用于管理系统的全局配置参数。该模块采用单例模式进行设计，

以确保系统在运行过程中始终使用唯一且一致的配置数据。

单例模式站点配置模块的后端代码如图 4_18 所示。

```
1 from django.contrib import admin
2 from .models import SiteProfile # 确保导入 SiteProfile
3
4 @admin.register(SiteProfile)
5 class SiteProfileAdmin(admin.ModelAdmin):
6     list_display = ('__str__', 'last_updated')
7
8     def get_queryset(self, request):
9         qs = super().get_queryset(request)
10        # 确保我们总是处理 pk=1 的单例
11        # 如果单例尚不存在, get_solo() 会创建它
12        return qs.filter(pk=SiteProfile.get_solo().pk)
13
14     def has_add_permission(self, request):
15        # 由于 get_solo() 会自动创建实例, 这里总是返回 False 避免通过 Admin 创建多个
16        # 或者, 如果 SiteProfile.objects.count() ≥ 1, 则返回 False
17        return not SiteProfile.objects.exists() # 只有在还没有任何记录时才允许添加第一个
18
19     def has_delete_permission(self, request, obj=None):
20        # 通常不允许删除这个唯一的配置项
21        return False
```

图 4_18 单例模式站点配置后端代码

站点配置模块

检索和更新站点配置文件的 API 端点。

GET: 全部打开以检索配置文件数据。

PUT: 仅限于管理员用户更新配置文件数据。

站点配置模块的后端代码如图 4_19 所示。

```

class SiteProfileAPIView(APIView):
    """ ... """

    permission_classes = [permissions.AllowAny] # 确保 GET 请求是公开的

    2 usages
    def get_object(self):
        # 使用模型的 get_solo 方法获取唯一的实例
        return SiteProfile.get_solo()

    def get(self, request, *args, **kwargs):
        profile = self.get_object()
        serializer = SiteProfileSerializer(profile)
        return Response(serializer.data)

    def put(self, request, *args, **kwargs):
        # 即使类别是 AllowAny, PUT 请求仍然需要显式的权限检查
        if not request.user or not request.user.is_authenticated:
            return Response(
                data={"detail": "Authentication credentials were not provided."},
                status=status.HTTP_401_UNAUTHORIZED
            )
        if not request.user.is_staff:
            return Response(
                data={"detail": "You do not have permission to perform this action."},
                status=status.HTTP_403_FORBIDDEN
            )

        profile = self.get_object()
        serializer = SiteProfileSerializer(profile, data=request.data, partial=True)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

```

图 4_19 站点配置后端代码

4.4.2 后台管理

系统后台管理模块为管理员提供了统一的管理入口，用于对站点内容、用户信息以及电影数据进行维护。

后台总管理页面如图 4_20 所示。

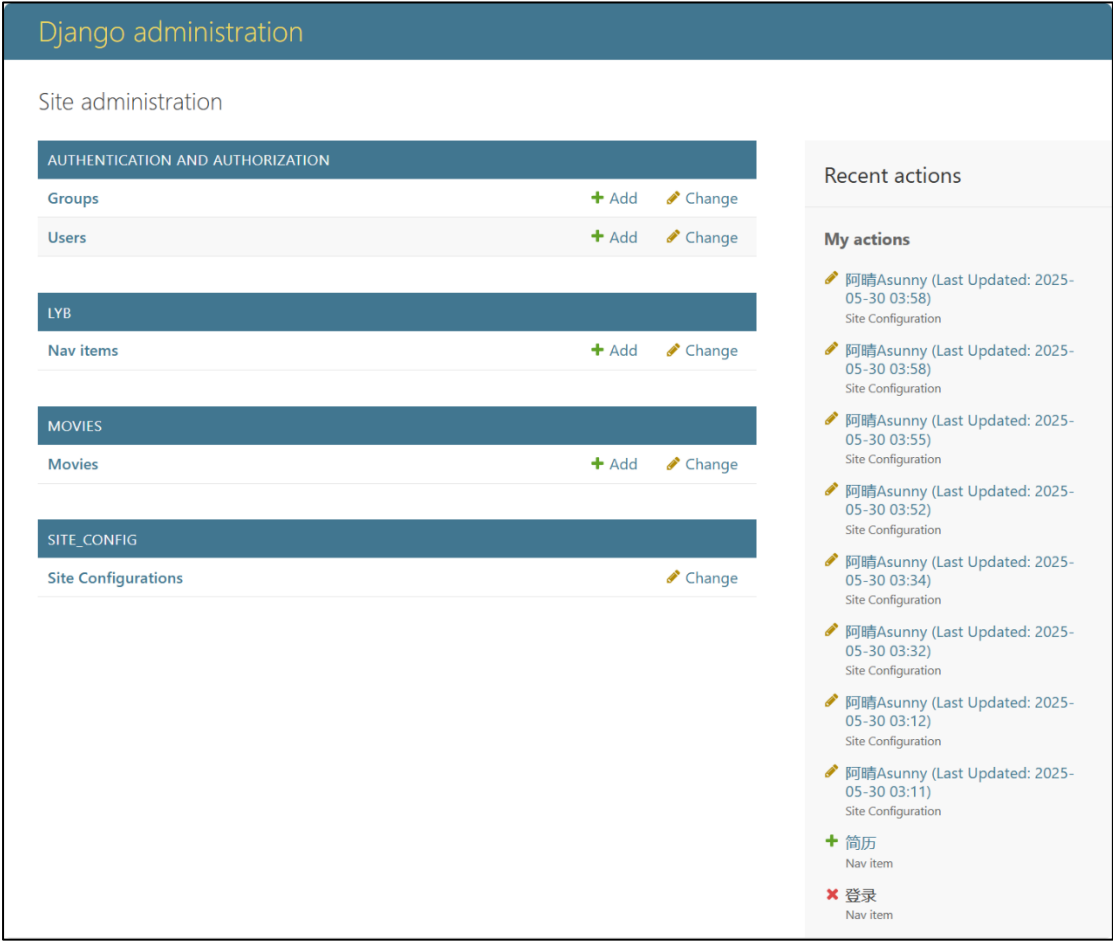


图 4_20 后台总管理页面

后台卡片配置页面如图 4_21 所示。

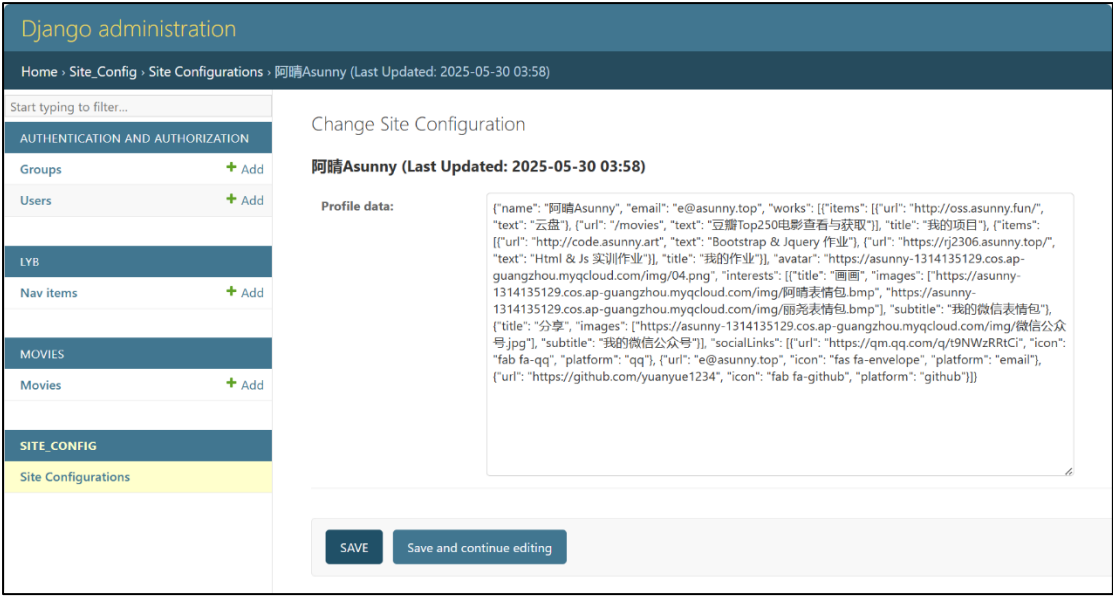


图 4_21 后台卡片配置页面

后台电影数据配置页面如图 4_22 所示。

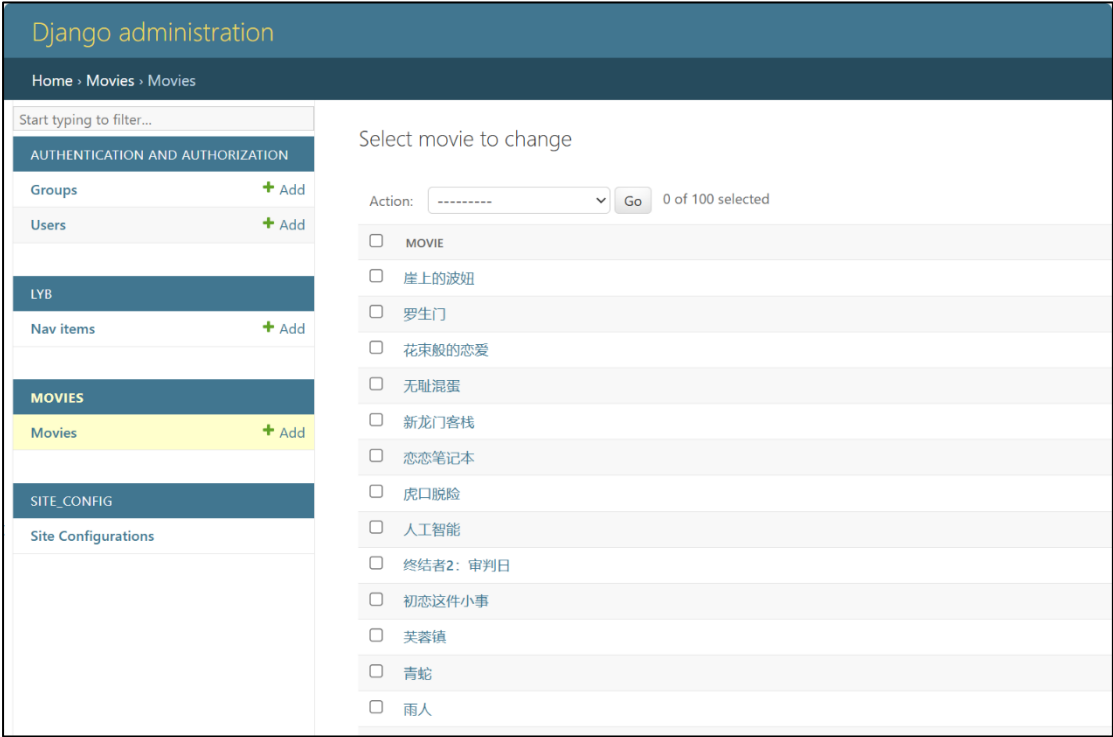


图 4_22 后台电影数据配置页面

后台分页配置页面如图 4_23 所示。

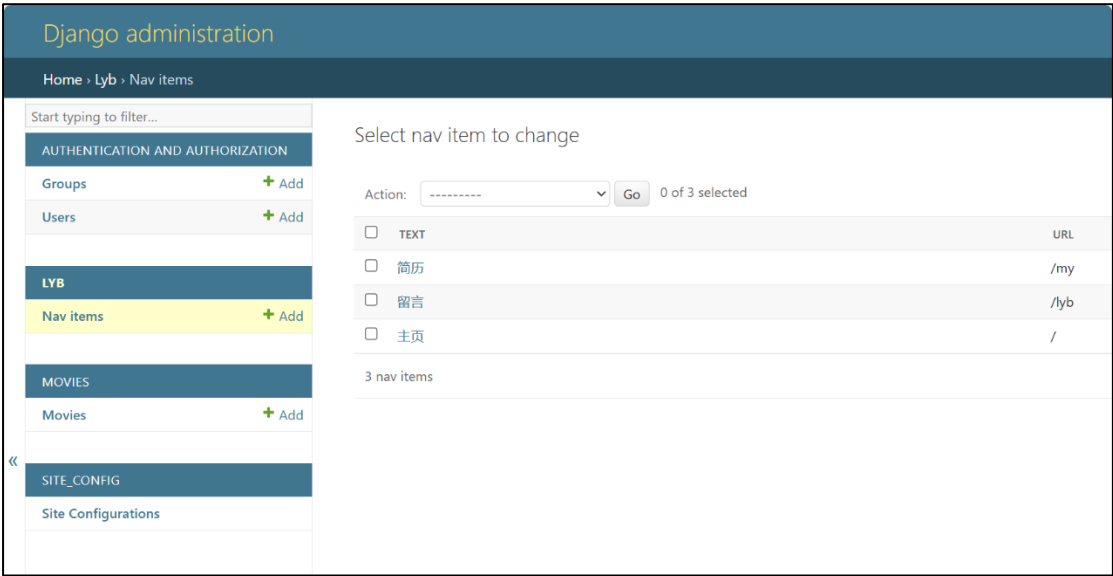


图 4_23 后台分页配置页面

后台账户管理页面如图 4_24 所示。

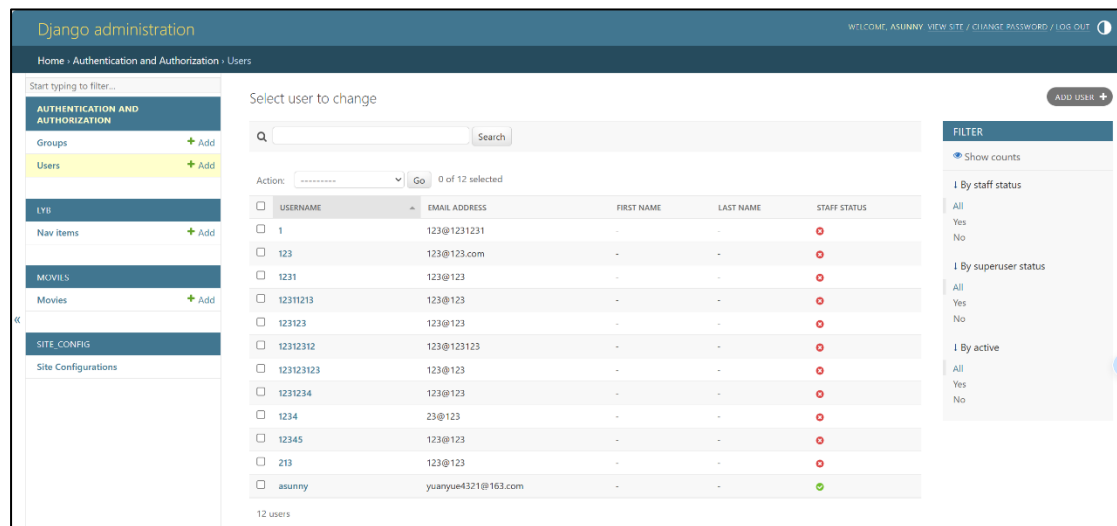


图 4_24 后台账户管理页面

5 总结/结语

本系统基于 Django + Vue3，实现了前后端分离的架构，集成了豆瓣电影数据分析与个人网站功能，采用谷歌的 Material Design 3 设计风格，其色彩、圆角元素、阴影和适应性组件和卡片元素为核心，具备实用性与展示性。主要特色包括电影数据多维度可视化（包含 3D 图表、词云图等）、JWT 认证及权限分级（游客 / 普通用户 / 管理员），并通过模块化设计实现了电影、留言板、站点配置等功能的解耦。在功能实现上，支持豆瓣 Top250 数据的搜索、分页、交互式展示，个人网站部分可留言互动与后台配置，基本达到了预期目标。

我的项目中也存在一些不足，例如部分图表交互刷新不够流畅，数据更新尚未实现自动化。

在开发过程中遇到跨域调试、图表渲染等问题，通过查阅文档、参考示例以及请教老师和同学逐步解决。整个过程帮助我熟悉了前后端分离开发流程，掌握了 JWT 认证机制和数据可视化方法，也提升了分析问题和独立解决问题的能力。

感谢指导老师在项目结构和技术方向上的指导，也感谢同学们在测试与反馈方面的帮助。通过本次毕业设计，我对 Web 项目的开发流程有了更全面更系统的理解，为未来的学习与工作打下了不错的基础。

参考资料

- [1] 兰琳琳. 基于 MySQL-Django-Vue 的在线考试系统[J]. 电脑知识与技术, 2024, 20(33):51-54.
- [2] 谢晓伟, 包琦. 基于 Django 与 Vue 的农产品质量追溯管理平台[J]. 电子技术与软件工程, 2022(012):000.
- [3] JeffForcier, PaulBissex, WesleyChun, 等. Django Web 开发指南[M]. 机械工业出版社, 2009.
- [4] 黑马程序员. Python 快速编程入门 (第 3 版). 北京:人民邮电出版社, 2023.
- [5] 黑马程序员. Vue.js 前端开发实战 (第 2 版). 北京:人民邮电出版社, 2023.
- [6] 朱二华. 基于 Vue.js 的 Web 前端应用研究[J]. 科技与创新, 2017(20):3. DOI:10.15913/j.cnki.kjycx.2017.20.119.
- [7] 张良峰. 基于 Django 和 Vue 的低代码平台构建[J]. 数字通信世界, 2024(1):45-48.