基于Django+React的疫情防控社区志愿者管理系统的设计与实现

【原文对照报告-大学生版】

报告编号：21ce1ed8985s9se7　　检测时间：2023-03-30 20:38:51　　检测字符数：37261

作者姓名：李佳音　　所属单位：陕西国际商贸学院

检测结论：

全文总相似比 **41.81%** = 复写率 41.81% + 他引率 0.0% + 自引率 0.0% + 专业术语 0.0%

其他指标：

自写率：58.19%

高频词：系统,用户,信息,进行,数据

典型相似文章：　无

疑似文字图片：0

指标说明：

复写率：相似或疑似重复内容占全文的比重　　他引率：引用他人的部分占全文的比重

自引率：引用自己已发表部分占全文的比重　　自写率：原创内容占全文的比重

典型相似性：相似或疑似重复内容占全文总相似比超过30%　　专业术语：公式定理、法律条文、行业用语等占全文的比重

相似片段：

总相似片段 267

期刊：23　博硕：124　综合：1

外文：0　自建库：45　互联网：74

检测范围：

中文科技期刊论文全文数据库　　中文主要报纸全文数据库　　中国专利特色数据库

博士/硕士学位论文全文数据库　　中国主要会议论文特色数据库　　港澳台文献资源

外文特色文献数据全库　　维普优先出版论文全文数据库　　互联网数据资源/互联网文档资源

高校自建资源库　　图书资源　　古籍文献资源

个人自建资源库　　年鉴资源　　IPUB原创作品

时间范围：1989-01-01至2023-03-30

原文对照

颜色标注说明：

- 自写片段
- 复写片段（相似或疑似重复）
- 引用片段（引用）
- 专业术语（公式定理、法律条文、行业用语等）

学号 19103020128
陕西国际商贸学院本科毕业论文
基于Django+React的疫情防控社区志愿者管理系统的设计与实现
二 级 学 院：信息工程学院
专 业 名 称：计算机科学与技术
学 生 姓 名：李佳音
指 导 教 师：马小菊
二〇二三年三月
郑 重 声 明

本人呈交的学位论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确的方式标明。本学位论文的知识产权归属于培养单位。
本人签名： 日期：

摘 要

计算机技术在现代信息管理的应用中，使计算机成为管理员应用现代技术的重要工具。能够高效便捷的解决管理员的管理工作，实现对志愿者信息管理的自动化，提高效率。

本文介绍了一种基于DRF框架和React框架的疫情防控社区志愿者管理系统，旨在帮助机构或组织更好地管理和运营志愿者队伍，提高组织的效率、减少管理成本。首先，本文对志愿者管理的相关背景和现状进行了分析和总结，指出了传统志愿者管理的弊端和不足。其次，本文提出了一种基于DRF框架和React框架的疫情防控社区志愿者管理系统的设计方案，并详细介绍了系统的功能模块及其实现方法。系统的主要功能包括用户管理、活动管理、报名信息管理和统计分析。最后，本文对系统进行了系统测试、功能测试和接口测试，根据测试结果分析可得出结论，本系统具有良好的用户体验和稳定性，可以满足志愿者管理的实际需求。本文所提出的基于DRF框架和React框架的疫情防控社区志愿者管理系统具有一定的创新性和实用性，可为志愿公益活动管理提供新的思路和方法。系统功能模块齐全，实现了对志愿者以及疫情防控公益活动管理的系统化、科学化，既可以提高服务质量，又大大的促进了管理系统的发展。

关键词：志愿者管理系统；DRF框架；React框架

Abstract

In the application of modern information management, computer technology has become an important tool for administrators to apply modern technology. It can solve the management of administrators

efficiently and conveniently, realize the automation of volunteer information management, and improve efficiency.

This paper introduces a community volunteer management system for epidemic prevention and control based on DRF framework and React framework, which aims to help organizations or organizations better manage and operate volunteer teams, improve organizational efficiency and reduce management costs. First of all, this paper analyzes and summarizes the background and current situation of volunteer management, and points out the drawbacks and shortcomings of traditional volunteer management. Secondly, this paper proposes a design scheme of community volunteer management system for epidemic prevention and control based on DRF framework and React framework, and introduces the functional modules and implementation methods of the system in detail. The main functions of the system include user management, activity management, registration information management and statistical analysis. Finally, this paper conducts system test, function test and interface test on the system. According to the analysis of test results, it can be concluded that the system has good user experience and stability, and can meet the actual needs of volunteer management. The community volunteer management system for epidemic prevention and control based on DRF framework and React framework proposed in this paper is innovative and practical, and can provide new ideas and methods for the management of voluntary public welfare activities. With complete functional modules, the system has realized the systematic and scientific management of volunteers and public welfare activities of epidemic prevention and control, which can not only improve the quality of service, but also greatly promote the development of the management system.

Key words: volunteer management system; Django restful framework; React framework

目录

1 绪论

1.1 课题背景与意义

计算机技术的快速发展在一定程度上改变了全球几乎所有组织团体自我管理的方法，自上世纪九十年代起，我国已经有企业开始提出通过互联网处理信息。二十一世纪来，我国的国家综合实力有了很大的提升，经济得以飞速发展，社会各行各业都开始加入信息化的趋势中。

以往的志愿者公益活动的信息，都是通过工作人员手工完成统计。这种方式无法满足巨量的数据信息处理，缺乏信息时效性，在查询和修改信息时有很大的不便之处。但现在计算机技术体系已将相对成熟能够满足人们生活中的很多需求。本系统的信息处理有着人工无可替代的优点。计算机有着巨大的算力，处理信息迅速高效，有着极大的储存量，同时信息追溯性强，人工成本低廉。能够极大地提高工作效率，有了成熟的社区志愿者管理系统，公益活动的举行在各个方面更加科学高效，更加简单系统。

通过社区志愿者管理系统的使用，可以从以下几个方面简述该系统对公益行业发展带来的便利和优化。首先通过该系统可以有效地提升公益行业的规模，同时处理大量的志愿者报名信息，使整个行业发展得更加规范，更加合理，以带动整体的工作效益。其次依附于该系统，社区志愿者可以用更好的管理结构，优化活动人员报名比例，节省大量的管理中损耗的成本。综上，该系统对公益志愿行业的发展有长足的便利。

1.2 国内外研究现状

目前，关于志愿者服务和志愿者组织管理的研究文献数量有限。随着志愿活动的发展，志愿者和公益活动的管理问题将越发凸显。目前，国内关于志愿者管理的相关研究主要集中在以下点，第一个是宏观性的“问题——对策”模式；第二点是对志愿者的激励；第三点是使用人力资源管理方法进行考察；第四点是特殊背景下的特殊志愿活动；第五点是关于对青少年志愿者的思想教育问题。

我国志愿者管理大概是在90年代初刚起步，因为经济稍微落后等因素，对志愿服务、没有足够的精力来进行管理。没有对志愿者的工作制定相关的法律法规来完善它，也没有对志愿者的管理提出适合的政策。相较于国外的研究状况，国内对志愿者管理的研究就比较晚，而且主要集中在最近十多年。在过去的这些年，国内对志愿者以及志愿服务方面的研究越来越深刻。国内针对高校志愿者服务管理系统的分析与设计依旧不多。

国外在志愿者服务行业的研究要早于国内很多年，并且也产生了巨大的国际反响，引起了世界的普遍关注。这和国外发达国家的相关法律法规、管理政策、经济发展状况、国外的公民素质普遍偏高，志愿意识强烈是密切相关的。但就当时的经济水平和计算机发展状况而言，没有互联网服务的志愿者管理要想有高效率是十分局限的。志愿

活动组织者很难将自己活动及时宣传让大众都知道，也难以第一时间通知志愿者参与活动。并且，当时计算机十分昂贵，普遍设计的小型管理系统功能匮乏，只能简单地记录活动信息，实现并普及志愿服务管理系统是比较困难的事情。随着国外志愿服务组织的发展和大型志愿活动等的开展，国外志愿者相关政策的发展也促使着志愿服务的条件日趋成熟，志愿服务管理系统设计的渐渐完善。

1.3 本课题工作

现在对本系统进行基本介绍并在这个基础上分析整个系统开发的过程，以便全面的完成该系统，在开发前对志愿者公益行业进行了大量的调研并进行综合的整理，然后对已有的信息进行细致的分析，来决定系统需要实现什么样的功能，然后根据已有的数据设计系统，根据实际情况以及自身所掌握的技术，我确定使用 Python技术来完成该系统后端的开发，在数据储存方面选择了MySQL数据库。后端使用DRF框架，实现RESTful API接口。前端页面设计与开发使用React技术，实现单页面应用。这些项技术都经历了很长时间的发展，都十分成熟，从网页端动态化处理和对信息的处理，都安全高效实用。

2 系统开发环境

2.1 Python 技术

Python语言继承了ABC语言的特性，是一种面向对象的动态类型语言，最初被设计用于编写自动化脚本(shell)，随着它的版本不断更新，语言新功能持续添加，越来越多被用于前后端分离、独立的、大型项目的开发。Python的语法简洁清晰，适用于数据分析、人工智能、机器学习等多种领域。

Python是一种解释型语言，意味着它可以在运行时解释代码，而不需要编译。Python支持多种编程范式，包括面向对象编程，函数式编程和过程式编程。同时，Python也是一种高级编程语言，具有功能强大的编程功能，可以用于开发Web应用程序，移动应用程序，网络服务和桌面应用程序。它还提供了一系列强大的编程库，可以用于实现各种功能，如数据处理，机器学习，图形处理等。Python还提供了一个交互式编程环境，可以让开发者快速开发和测试代码。

Python的优点有很多，主要包含以下几点：Python是一种易于学习的编程语言，它具有简洁的语法，使程序员能够快速上手。Python拥有丰富的第三方库，可以满足开发人员的各种需求。Python支持多种编程范式，可以满足不同的开发需求。Python拥有强大的可移植性，可以在多种操作系统上运行；Python拥有优秀的可读性，可以让程序员更容易理解代码。

2.2 MySQL数据库

MySQL是一种关系型数据库，由瑞典MySQL AB公司开发，属于 Oracle 旗下产品。MySQL 是最流行的关系型数据库之一，在 WEB 应用方面，MySQL是最好的 RDBMS (Relational Database Management System，关系数据库管理系统) 应用软件之一。

MySQL数据库在数据存储方面有很强大的功能，易于使用，易于安装和维护；支持多种存储引擎，可以根据不同的应用场景选择合适的存储引擎；支持多种编程语言，可以使用多种编程语言来操作 MySQL 数据库；支持多种安全机制，可以有效地保护数据库的安全；支持多种复制机制，可以提高数据库的可用性和性能；支持多种分布式架构，可以满足不同的业务需求。

2.3 B/S结构

B/S结构是指浏览器/服务器结构，是一种客户端/服务器结构，客户端通过浏览器发出请求，服务器接收请求并处理，然后将处理结果返回给客户端，客户端再将结果显示出来。

数据层、控制逻辑层和视图层组成了 B/S 结构的三个主要层级。用户的全流程访问是以视图层窗口，凭借控制层转移数据层的数据来完成的。此三个层级相对独立，有利于分别维护和稳定操作，又能做到数据共享以实现高效。

2.4 DRF（Django REST Framework）框架

DRF(Django REST framework)是一个基于Django的RESTful Web框架，它提供了一系列的工具，可以帮助开发者快速构建RESTful API。它提供了一个可扩展的架构，可以让开发者轻松地构建和维护RESTful API。它还提供了一系列的序列化器，可以帮助开发者将数据转换为JSON或XML格式，以便在Web浏览器和移动设备之间传输数据。

它提供了一系列的功能，包括路由，视图，序列化，认证，权限，调试、文档，测试。它还支持多种数据库，如PostgreSQL，MySQL，SQLite，MongoDB等。DRF框架的发展可以追溯到2010年，当时它是一个简单的Python库，用于构建RESTful API。随着时间的推移，DRF框架的生态变得越来越强大。现在，它已经成为一个完整的Web框架，可以帮助开发人员快速构建和部署RESTful API。

DRF框架的优势包括：
易于使用：DRF框架具有易于使用的API，可以轻松地构建RESTful API。
可扩展性：DRF框架具有可扩展性，可以根据需要轻松扩展功能。
安全性：DRF框架具有强大的安全性，可以有效地防止CSRF攻击和XSS攻击。
可读性：DRF框架具有可读性，可以轻松地调试和维护代码。
可定制性：DRF框架具有可定制性，可以根据需要轻松定制功能。
2.5 React框架

React是前端三大主流框架之一，是一个专门用来构建前端界面UI的JavaScript库，它的性能较高，代码逻辑简单易懂。

React使用声明式编程，允许开发人员更轻松地创建动态的用户界面。React可以让开发人员更轻松地处理视图层，并且可以更快地渲染用户界面。它还可以帮助开发人员更轻松地管理状态，从而更轻松地创建可维护的应用程序。

React拥有强大的功能和优点：使用JSX语法，可以更容易地构建用户界面。可以使用组件化的方式来构建应用，更容易维护和扩展。拥有虚拟DOM，可以更快地渲染页面。拥有强大的社区支持，可以获得更多的帮助。拥有一些实用的工具，可以更快地开发应用。
2.6 Ant Design Pro

Ant Design Pro 是基于 Ant Design 和 umi 的封装的一整套企业级中后台前端/设计解决方案，致力于在设计规范和基础组件的基础上，继续向上构建，提炼出典型模板/业务组件/配套设计资源，进一步提升企业级中后台产品设计研发过程中的『用户』和『设计者』的体验。

Ant Design Pro 在力求提供开箱即用的开发体验，为我们提供完整的脚手架，涉及国际化，权限，mock，数据流，网络请求等各个方面。为这些中后台中常见的方案提供了最佳实践来减少学习和开发成本。

同时为了提供更加高效的开发体验，Ant Design Pro提供了一系列模板组件，ProLayout，ProTable，ProList都是开发中后台的好帮手，可以显著的减少样板代码。
2.7 系统开发环境
操作系统：Windows10
Python运行环境：Python3.6
Node环境：Node -v16.13.2
数据库：MySQL 5.7
React运行环境：React17.0.0
2.8 系统开发工具
编译器：PyCharm，VSCode
数据库可视化工具：Navicat Premium
测试工具：Postman
接口文档工具：Swagger
浏览器：Google 浏览器和Edge浏览器
3 系统分析
3.1 可行性分析

疫情防控社区志愿者管理系统的主要功能是对公益活动信息和每个活动参加志愿者的管理。可行性分析要从系统的经济层、技术层和可操作性对系统进行全面分析，以下根据这三点对疫情防控社区志愿者管理系统进行全面的分析。
3.1.1 技术可行性

疫情防控社区志愿者管理系统主要采用Python语言和tsx语言，基于B/S结构进行开发，DRF(Django REST Framework)框架完成接口的实现，React框架实现前端的设计与开发，MySQL数据库实现对数据的存储。以上技术为本系统的关键技术，可以支撑并完成对本系统的开发语实现，所以在技术可行性分析方面没有太大的问题。
3.1.2 操作可行性分析
现在已有的管理系统对使用者来说都是很简单方便，该系统也例外，根据使用者的角度，该系统在开发上致力于简化结构，使页面尽可能简洁，优化用户操作，使用者只需要在浏览器中打开系统并进行登录就可以进行使用操作系

统。综上所述，该系统的操作简单，方便，可以进行开发。

3.1.3 经济可行性

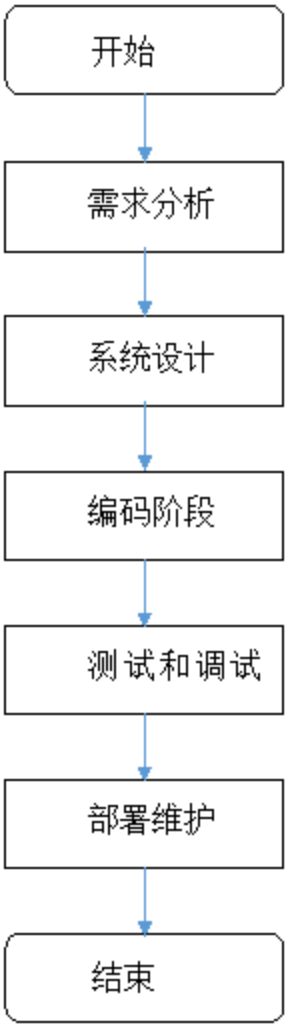
疫情防控志愿者管理主要用于管理公益活动的发布和志愿者的信息，该系统的运行所需配置不高，很容易购买，在开发和建设期间必要投入一定的资金，但该系统可以提高效率，减少相应的管理成本，对整个行业的发展都能起到促进作用，带来更多的利益，所以该系统在经济方面是可行的。

3.2 系统流程分析

3.2.1 系统开发流程

系统的开发流程包括以下几个阶段：

1. 需求分析：确定系统的功能和性能要求，以及用户的需求和期望。
2. 设计阶段：设计系统的架构、模块、接口和数据结构等。
3. 编码阶段：根据设计文档和开发计划进行编码实现。
4. 测试和调试阶段：对系统进行功能测试、性能测试、安全测试等，并进行调试，确保系统的可靠性和稳定性。
5. 部署和维护阶段：部署系统到实际环境中，并对系统进行维护和升级。



系统开发的流程图如图3.1所示：

图 3.1 系统开发流程图

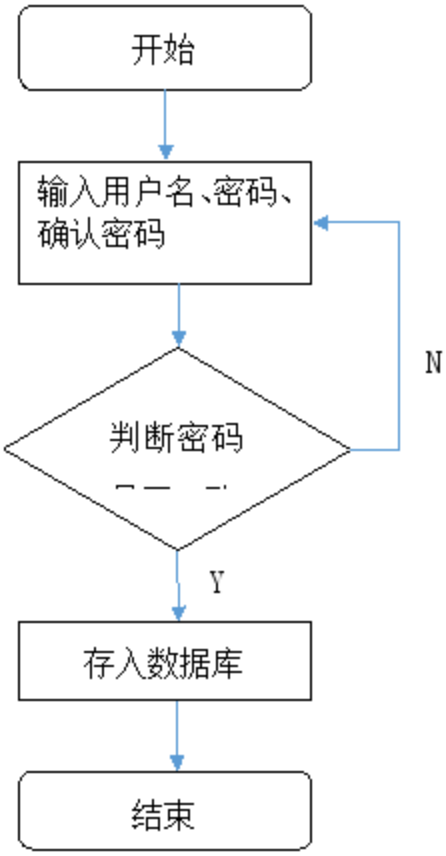
3.2.2 用户注册流程

普通用户和公益企业账号首先要进行注册才能获取系统的登录权限。

系统注册流程分析：

1. 打开注册页面：用户需要打开系统的注册页面，填写相应的注册信息。
2. 填写注册信息：用户需要填写相应的注册信息，包括用户名、密码、邮箱、手机号码等，确保信息的正确性和完

整性。
3. 校验信息：系统需要对用户填写的信息进行校验，确保信息的正确性和完整性。
4. 发送验证邮件/短信：系统需要向用户发送验证邮件或短信，以验证用户的身份和信息。
5. 确认验证信息：用户收到验证邮件或短信后，需要确认验证信息，并点击验证链接。
6. 验证成功：系统收到用户提交的验证信息后，进行相应的验证，验证成功后，用户可以登录系统。
7. 返回结果：系统会返回注册的结果，包括注册是否成功、注册的用户名等。
8. 操作完成：用户根据系统返回的结果，进行相应的处理，例如登录系统、修改信息等。
在设计系统注册流程时，需要考虑以下几个方面：
1. 界面设计：注册页面应该设计清晰、简单，便于用户填写相关信息。
2. 数据校验：系统需要对用户填写的信息进行校验，确保数据的正确性和完整性。
3. 异常处理：系统需要对异常情况进行处理，例如用户填写错误信息、系统故障等情况。
4. 用户体验：操作流程应该简单、易于理解，减少用户操作的复杂性和困难度。
5. 安全保障：系统应该采取相应的安全措施，确保用户的数据安全和隐私保护。
通过合理的系统注册流程分析，可以提高用户的使用效率和用户体验，同时也可以提高系统的可靠性和安全性。



注册的流程图如图3.2所示：
图 3.2 注册流程图
3.2.3 用户登录流程
用户的信息安全至关重要，关系到一个系统的数据库安全，安全不达标，会造成信息失窃损毁，对用户带来无法挽回损失，因此本系统在使用前，要输入使用者的用户名和密码登录，保证用户信息不会泄露。用户登录的流程图如图3.3所示。

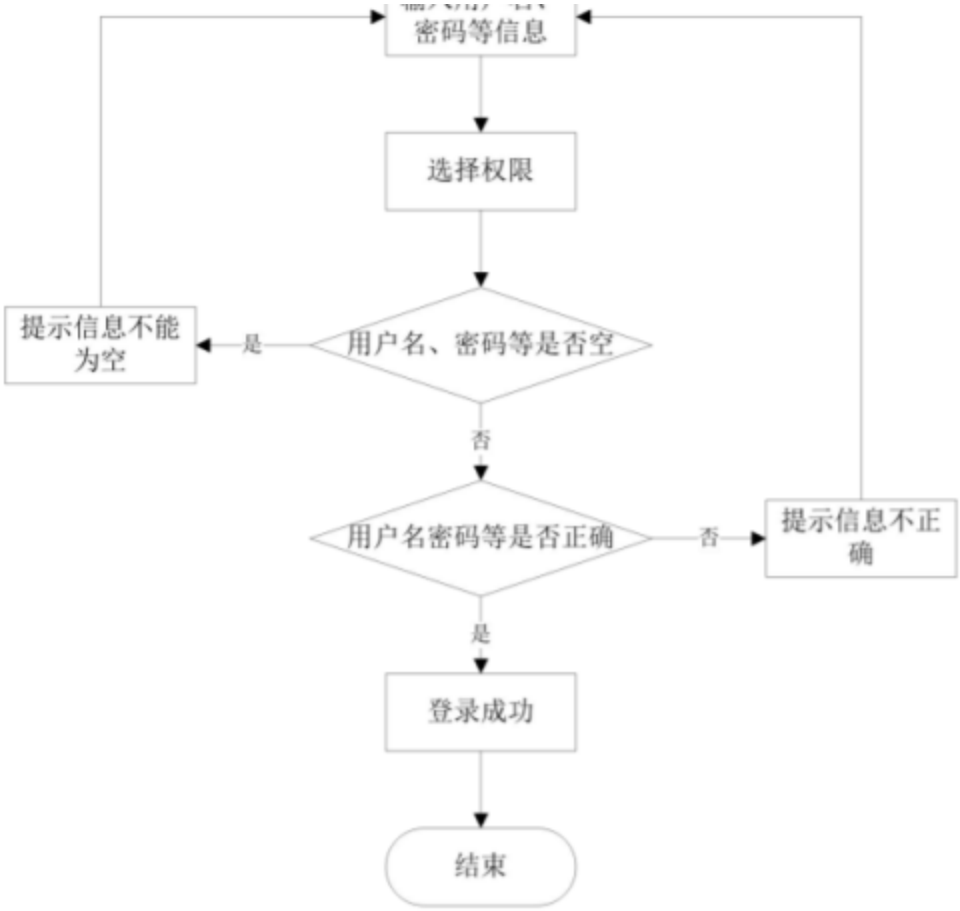


图 3.3 用户登录流程图

3.2.4 系统操作流程

系统操作流程，是指用户在体验或使用系统时所需要遵循的操作步骤和流程，其主要目的是为了帮助系统用户完成相应的操作任务。系统操作流程包括以下几个方面：

1. 登录操作：用户需要输入账号和密码等信息，登录系统后才能进行后续的操作。
2. 界面操作：用户需要了解系统的界面布局和操作方式，包括菜单栏、工具栏、状态栏等，以便于快速定位和操作所需要的功能。
3. 功能操作：用户需要根据自己的需求，选择对应的功能模块进行操作，例如查询、新增、编辑、删除等操作。
4. 数据输入和输出：用户需要输入所需要的数据信息，或者根据查询结果查看相应的数据信息，以满足自己的业务需求。
5. 操作流程控制：在操作过程中，用户需要根据系统的提示信息和状态信息，控制操作流程，确保操作的正确性和完整性。
6. 操作结果处理：用户需要根据操作结果，进行相应的处理，例如保存数据、打印报表、导出数据等。
7. 系统退出操作：用户在完成操作后，需要进行系统退出操作，以保护系统和用户的数据安全。

在设计系统操作流程时，需要着重考虑用户体验，使得系统操作流程更加简单和易于使用。同时，需要将系统操作流程设计为可定制和可配置的，以便于满足不同用户的需求和使用习惯。

3.2.5 添加信息操作流程

系统添加信息操作流程包括以下几个步骤：

1. 打开添加信息对话框：用户需要打开系统中相应的功能模块，并选择添加信息的操作。
2. 填写信息：用户需要根据系统的提示，填写相应的信息，包括名称、描述、时间、地点等。
3. 检查信息：用户需要仔细检查填写的信息是否正确，确保添加的信息符合要求。
4. 提交信息：用户需要点击提交按钮，将填写的信息提交到系统中进行保存。
5. 系统处理信息：系统收到用户提交的信息后，会进行相应的处理，包括校验数据、保存数据等操作。

6. 返回结果：系统会返回添加信息的结果，包括添加是否成功、添加的信息ID等。
7. 操作完成：前端根据系统后端返回的结果，进行对应的处理，例如保存信息、继续添加信息、返回信息列表页面、更新信息列表等。
在设计添加信息操作流程时，需要考虑以下几个方面：
1. 界面设计：添加信息界面应该设计清晰、简单，便于用户填写信息和提交信息。
2. 数据校验：系统需要对用户输入的数据进行校验，确保数据的正确性和完整性。
3. 异常处理：系统需要对异常情况进行处理，例如用户输入错误信息、系统故障等情况。
4. 用户体验：操作流程应该简单、易于理解，减少用户操作的复杂性和困难度。
5. 安全保障：系统应该采取相应的安全措施，确保用户的数据安全和隐私保护。
通过合理设计添加信息操作流程，可以提高用户的使用效率和用户体验，同时也可以提高系统的可靠性和安全性。
3.2.6 修改信息操作流程
系统修改信息操作流程包括以下几个步骤：
1. 打开信息编辑对话框：用户需要打开系统中相应的功能模块，并选择编辑信息的操作。
2. 选择要编辑的信息：用户需要在系统中选择要编辑的信息，系统会显示该信息的详细信息。
3. 编辑信息：用户可以根据需要，对信息进行修改，例如修改名称、描述、时间、地点等。
4. 检查信息：用户需要仔细检查修改后的信息是否正确，确保修改的信息符合要求。
5. 提交信息：用户需要点击提交按钮，将修改后的信息提交到系统中进行保存。
6. 系统处理信息：系统收到用户提交的修改信息后，会进行相应的处理，包括校验数据、保存数据等操作。
7. 返回结果：系统会返回修改信息的结果，包括修改是否成功、修改后的信息内容等。
8. 操作完成：前端根据后端返回的结果，进行对应的处理，例如保存修改后的信息、继续编辑信息、跳转回列表页面等。
在设计修改信息操作流程时，需要考虑以下几个方面：
1. 界面设计：信息编辑界面应该设计清晰、简单，便于用户修改信息和提交信息。
2. 数据校验：系统的前后端需要有数据校验，对用户修改的数据进行字段校验，确保修改的数据符合数据库的数据字段设计规范。
3. 异常处理：系统需要对异常情况进行处理，例如用户输入错误信息、系统故障等情况。
4. 用户体验：操作流程应该简单、易于理解，减少用户操作的复杂性和困难度。
5. 安全保障：系统应该采取相应的安全措施，确保用户的数据安全和隐私保护。
通过合理设计修改信息操作流程，可以提高用户的使用效率和用户体验，同时也可以提高系统的可靠性和安全性。
3.2.7 删除信息操作流程
系统删除信息操作流程包括以下几个步骤：
1. 打开信息对应模块管理界面：用户需要打开系统中相应的功能模块，并选择管理信息的操作。
2. 选择要删除的信息：用户需要在系统中选择要删除的信息，系统会显示该信息的详细信息。
3. 确认删除：用户需要确认要删除该信息，系统会弹出确认删除的提示框。
4. 确认删除操作：用户需要再次确认要删除该信息，系统会再次弹出确认删除的提示框。
5. 系统处理信息：系统收到用户提交的删除信息请求后，会进行相应的处理，包括校验数据、删除数据等操作。
6. 返回结果：系统会返回删除信息的结果，包括删除是否成功、删除的信息内容等。
7. 操作完成：用户根据系统返回的结果，进行相应的处理，例如刷新信息列表、继续管理信息等。
在设计删除信息操作流程时，需要考虑以下几个方面：
1. 界面设计：信息管理界面应该设计简单明了，便于用户选择要删除的信息。
2. 数据校验：系统需要对用户勾选的数据与数据库的数据进行比对校验，确保数据存在且准确无误。
3. 异常处理：系统需要对异常情况进行处理，例如用户选择错误信息、系统故障等情况。
4. 用户体验：操作流程应该简单、易于理解，减少用户操作的复杂性和困难度。
5. 安全保障：系统应该采取相应的安全措施，确保用户的数据安全和隐私保护。
通过合理设计删除信息操作流程，可以提高用户的使用效率和用户体验，同时也可以提高系统的可靠性和安全性。
3.3 系统用例分析
3.3.1 管理员用例

本系统中权限最高的是管理员，通过管理员的账户可对整个后台的所有功能进行管理。
管理员用例分析：
1. 登录系统：
管理员需要输入用户名和密码才可以登录系统，管理员用户拥有系统的所有功能模块的权限。
2. 管理用户：
管理员可以对系统所有的用户信息进行管理，包括添加、删除、修改、注销用户等操作，以确保用户信息的准确性和完整性。
3. 管理信息：
管理员可以对系统中的信息进行管理，包括添加、修改、删除信息等操作，以确保信息的准确性和完整性。
在管理员用例分析中，需要考虑以下几个方面：
1. 用户交互体验：管理员可以通过简单、直观的界面操作系统，提高用户的使用效率和用户体验。
2. 安全保障：系统应该采取相应的安全措施，例如密码安全策略、数据备份策略等，确保系统的安全性和可靠性。
3. 异常处理：系统需要对异常情况进行处理，例如系统故障、数据错误等情况。
4. 数据管理：管理员需要对用户和信息进行管理，确保用户信息和系统信息的准确性和完整性。
5. 系统性能：管理员需要监测系统的性能和安全性，并及时发现和解决运行问题，确保系统的正常运行和稳定性。
通过合理的管理员用例分析，可以提高系统的管理效率和安全性，满足用户的需求和要求。
管理员用例如图 3.4 所示：

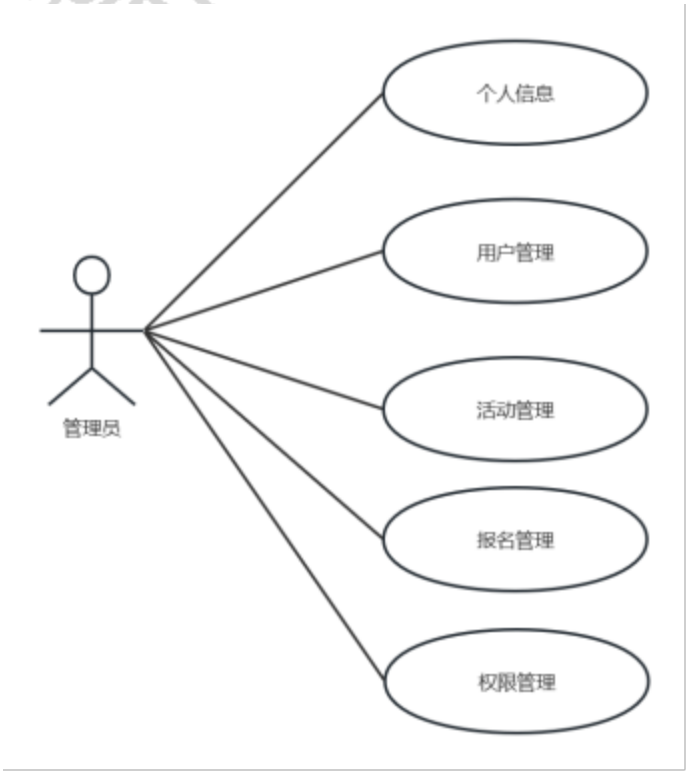


图 3.4 管理员用例图
3.3.2 公益企业用例
公益企业用户可以进行活动的发布、编辑、删除、查看，以及对普通用户的报名信息进行审核。
发布企业用例如图 3.5 所示：

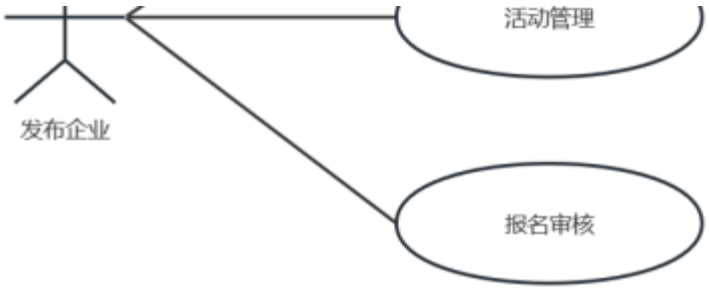


图 3.5 发布企业用例图

3.3.3 普通用户用例

普通用户登录系统，可以进入活动列表页面，查询活动，查看活动详情，报名自己感兴趣的公益活动，查看自己的报名状态。

普通用户用例如图3.6所示：

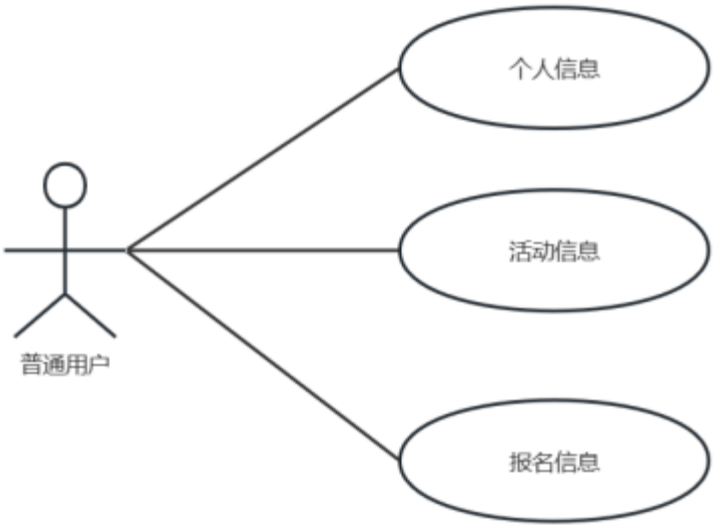


图 3.6 普通用户用例图

4 系统设计

4.1 系统结构设计

系统结构设计是指基于系统需求和系统功能，设计出系统的整体结构框架，包括系统的模块、组件、接口、数据流等。系统结构设计的目的是为了将系统划分为不同的部分，使得系统的开发、测试、维护和升级都更加容易和高效。

系统结构设计包括以下几个步骤：

1. 确定系统需求：明确系统的功能模块需求、可靠性需求、性能需求、数据安全需求等，以便于后续的设计工作。
2. 划分系统子模块：将系统划分为不同的子模块，每个子模块负责不同的功能，子模块之间通过接口进行通信和数据交换。
3. 设计系统接口：确定不同模块之间的接口协议和数据格式，以确保数据的正确传输和解析。
4. 确定系统数据流：确定系统中各个模块之间的数据流动方式，以及数据流程的控制和管理方式。
5. 设计系统组件：根据系统需求和模块划分结果，设计系统的组件，包括硬件组件、软件组件、网络组件等。
6. 确定系统架构：根据系统模块、组件和接口的设计，确定系统的总体架构，包括层次结构、模块之间的依赖关系、系统的部署方式等。
7. 进行系统测试：根据系统结构设计的结果，进行系统的测试和验证，确保系统的正确性、可靠性和性能。

系统结构设计的目的是将系统复杂的功能分解成多个独立的子模块，每个子模块都可以独立的开发和测试，降

低系统的耦合度，最终整合成一个完整的系统。在设计过程中，需要考虑系统的可移植性、可复用性、可维护性、可靠性、可扩展性等方面的问题。最后，根据实际需求选择合适的设计模式和架构模式，以便于实现系统的功能和优化系统的性能。
系统功能模块如图4.1所示：

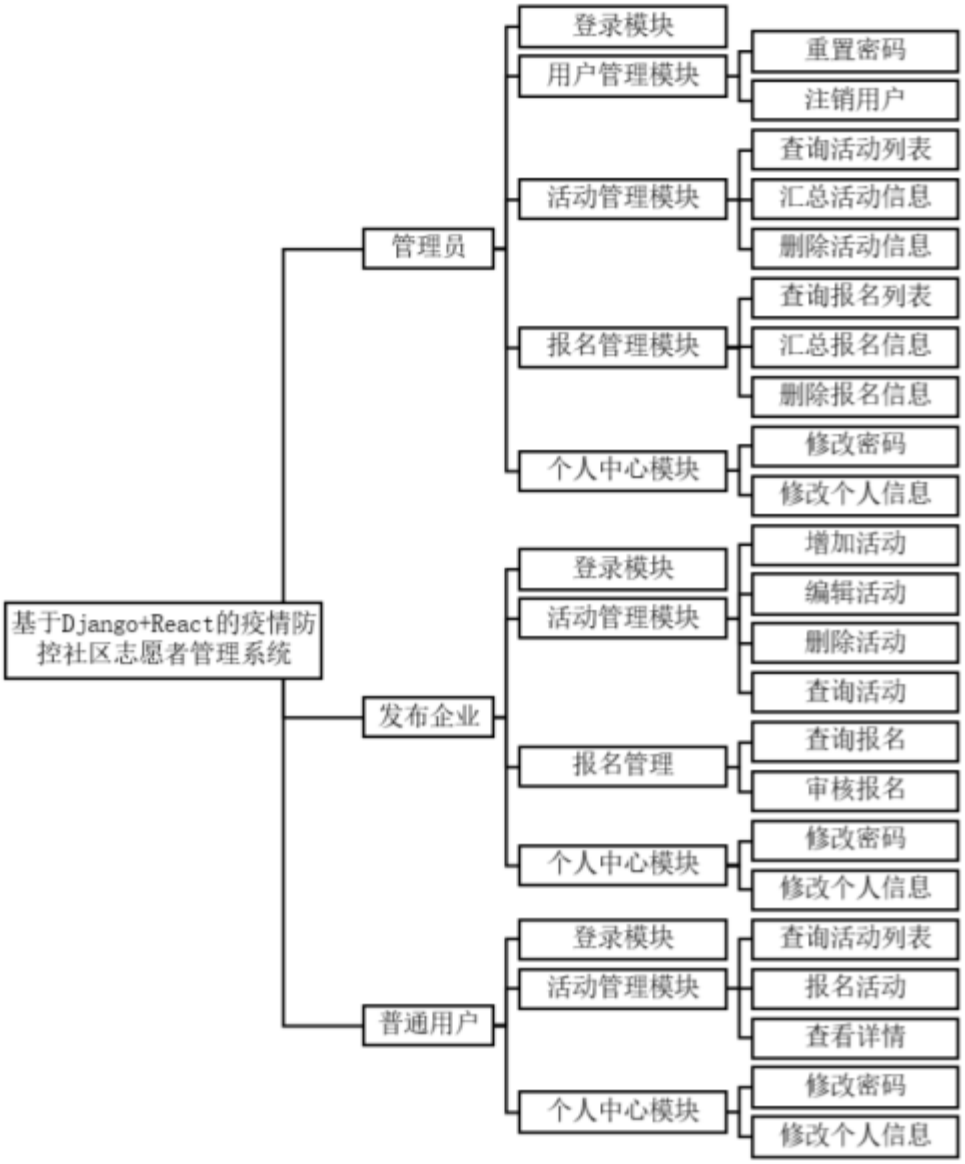


图 4.1 系统功能模块

4.2 数据库设计

数据库设计是指在遵循数据库理论的基础上，根据业务需求和数据特点，设计出合理的数据结构和数据关系，以便于数据的存储、管理和使用。数据库设计的流程，包含以下几个步骤：

1. 需求分析：确定数据库的数据特点，包括数据类型、数据量、数据关系等。
2. 概念设计：根据需求分析，设计出数据库的概念模型，包括实体、属性、关系等。
3. 逻辑设计：在概念模型的基础上，设计出逻辑模型，包括表结构、属性、关系、约束等。
4. 物理设计：在逻辑模型的基础上，设计出物理模型，包括表空间、索引、分区等。
5. 实现和测试：根据物理模型进行数据库的实现和测试，确保数据库的性能和可靠性。
6. 维护和优化：对数据库进行维护和优化，以保证数据库的稳定性和高效性。

在数据库设计的过程中，需要考虑数据的完整性、安全性、可扩展性以及性能等方面的问题，以确保数据库的高效和可靠。同时，需要遵循数据库设计的一些原则，如实体完整性、关系完整性、范式化等，以便于数据的管理

和使用。

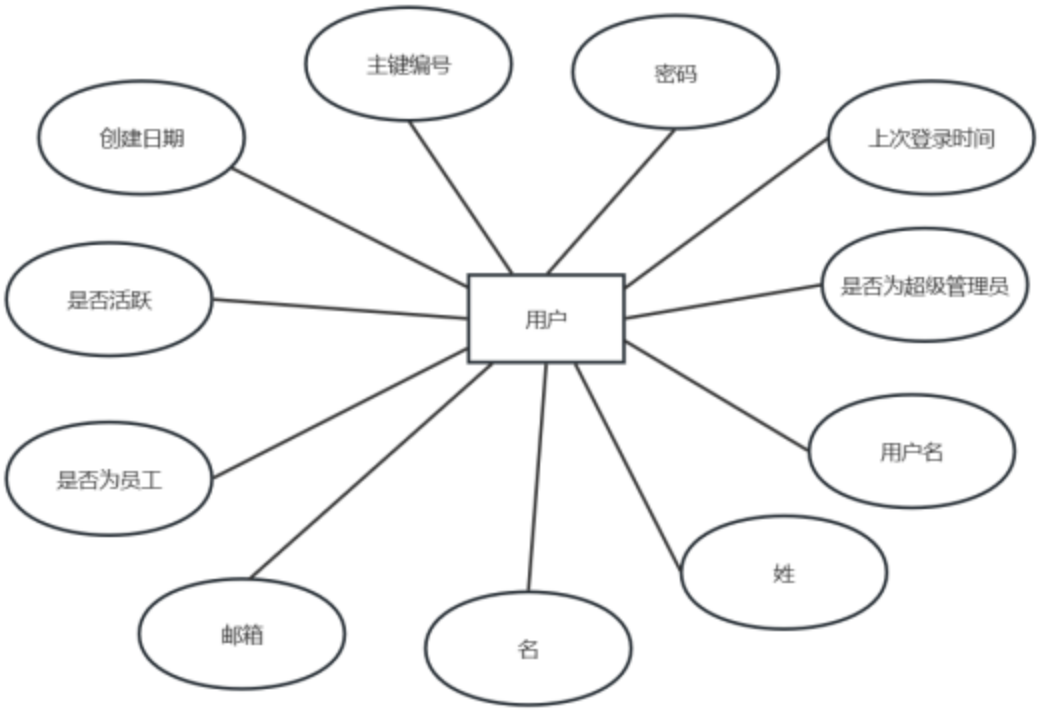
4.2.1 数据库设计原则

数据库设计原则是指在设计数据库时需要遵循的基本原则，这些原则可以使我们创建出的数据库结构合理、稳定、安全、易于维护和扩展。在设计该系统的数据库时要遵循原则包括以下几点：

1. 范式化原则：应该遵循范式化原则，即将重复的数据分解到不同的表中，以避免冗余数据的存在，从而提高数据的一致性和可靠性。
2. 实体完整性原则：确保每个表的每条记录都能被唯一地标识，还要保证每个表中的数据行都有对应的实体存在。
3. 关系完整性原则：在设计数据库时应该考虑数据之间的关系，确保数据之间的关系是正确的，并且能够保证数据的完整性。
4. 数据访问安全原则：在设计数据库时应该考虑数据库的安全性，包括数据的访问权限、数据的保密性、数据的完整性等。
5. 数据库性能原则：在设计数据库时应该考虑数据库的性能，包括数据库的查询性能、数据的存储效率、并发性能等。
6. 数据库扩展性原则：数据库的扩展性，包括数据库的容量、数据的增量、数据的迁移等。
7. 数据库备份与恢复原则：在设计数据库时，也应该考虑数据库的备份与恢复策略，通常情况下，对数据库进行备份，以保证数据的安全性和可靠性。

4.2.2 概念模型设计

根据系统需求分析本系统有用户、报名信息、活动信息。下面分别是各实体的实体图以及总的 E-R 图：



（1）用户实体图如图4.2所示：

图 4.2 用户ER图

（2）活动信息实体图如图4.3所示：

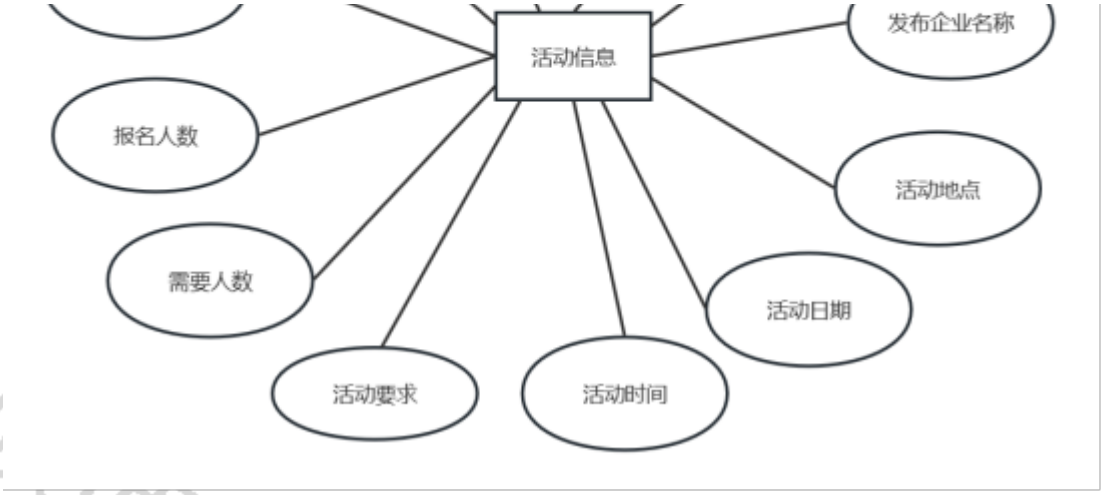
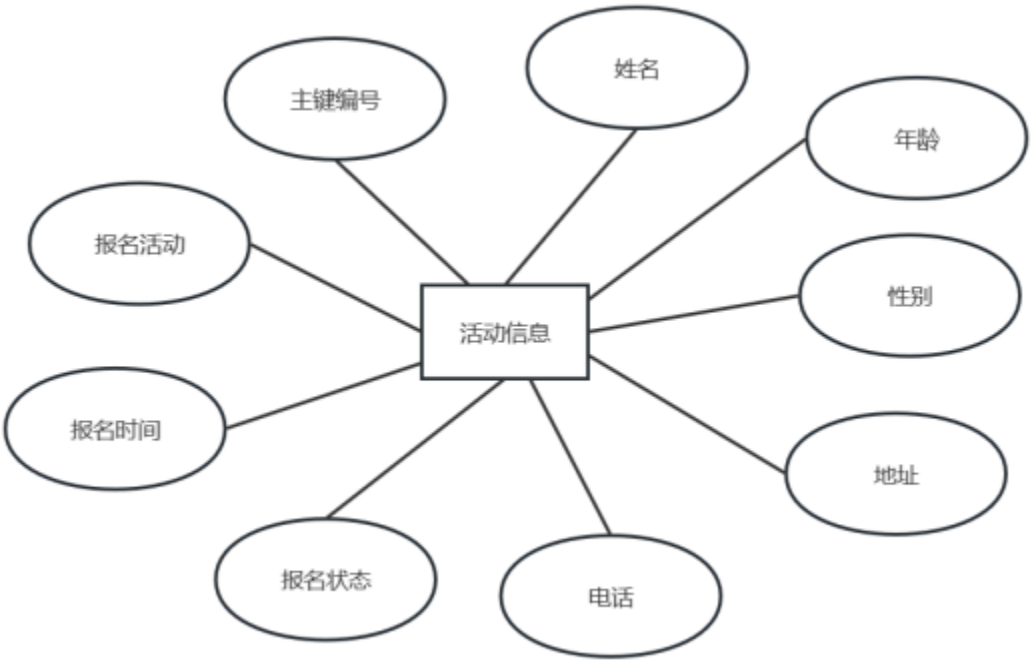


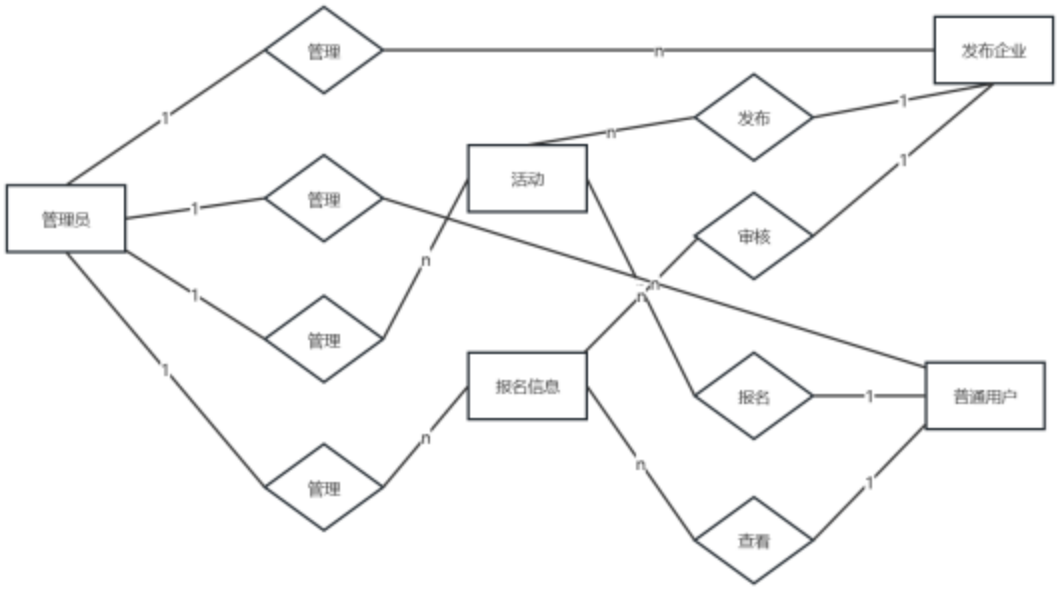
图 4.3 活动信息ER图



（3）报名信息的实体图如图4.4所示：

图 4.4 活动信息ER图

根据以上对系统的分析可知，在本系统包含了以下五个实体：管理员、普通用户、发布



企业、活动信息、报名信息。系统全局 E-R 图如图4.5所示：

图 4.5 全局E-R图

4.2.3 数据库逻辑结构设计

用户信息表包括编号、用户名、等个人信息，如表4.1所示：

表 4.1 用户信息表

字段名称	字段含义	类型	长度	键	是否为空
id	主键编号	int	11	主键	否
password	密码	varchar	128		否
last_login	上次登录时间	datetime	6		能
is_superuser	是否为超级管理员	tinyint	1		否
username	用户名	varchar	150		否
first_name	姓	varchar	30		能
last_name	名	varchar	150		能
email	邮箱	varchar	254		能
is_staff	是否为员工	tinyint	1		能
is_active	是否活跃	tinyint	1		能
date_joined	创建日期	datetime	6		否

活动信息表包含编号，名称等信息，如表4.2所示：

表 4.2 公益活动表

字段名称	字段含义	类型	长度	键	是否为空
id	主键编号	int	11	主键	否
name	活动名称	varchar	50		否
desc	活动描述	varchar	255		否
publish_company_name	发布企业名称	varchar	50		否
address	活动地点	varchar	50		否
start_date	活动日期	date	0		否

start_time	活动时间	time	6		否
demand	活动要求	varchar	255		否
need_person_num	需要人数	int	11		否
apply_person_num	报名人数	int	11		否
pass_person_num	通过人数	int	11		否
create_time	创建时间	datetime	6		否

报名信息表如表4.3所示：
表 4.3 报名信息表

字段名称	字段含义	类型	长度	键	是否为空
id	主键编号	int	11	主键	否
name	志愿者姓名	varchar	50		否
age	志愿者年龄	int	11		否
sex	志愿者性别	int	11		否
address	志愿者地址	varchar	50		否
tel	志愿者电话	varchar	255		否
apply_status	报名状态	int	11		否
apply_time	报名时间	datetime	6		否
belonging_activity_id	报名活动	int	11		否

分组信息表如表4.4所示：
表 4.4 分组信息表

字段名称	字段含义	类型	长度	键	是否为空
id	主键编号	int	11	主键	否
name	名称	varchar	150		否

权限信息表如表4.5所示：
表 4.5 权限信息表

字段名称	字段含义	类型	长度	键	是否为空
id	主键编号	int	11	主键	否
name	名称	varchar	255		否
content_type_id	类型id	int	111		否
codename	代码名称	varchar	00		否

用户权限关系对应表如表4.6所示：
表 4.6 用户权限关系关联表

字段名称	字段含义	类型	长度	键	是否为空
id	主键编号	int	11	主键	否
user_id	用户编号	int	11		否
perssion_id	权限编号	int	111		否

用户分组关系对应表如表4.7所示：
表 4.7 用户分组关系关联表

字段名称	字段含义	类型	长度	键	是否为空
id	主键编号	int	11	主键	否
user_id	用户编号	int	11		否
group_id	分组编号	int	111		否

分组权限关系对应表如表4.8所示：
表 4.8 分组权限关系关联表

字段名称	字段含义	类型	长度	键	是否为空
id	主键编号	int	11	主键	否
group_id	分组编号	int	11		否
perssion_id	权限编号	int	111		否

5 系统的设计与实现

5.1 公共模块

5.1.1 登录模块

登录模块是一个重要的功能模块，每个用户在打开系统界面时，都要进行登录操作。用户只有在登录状态下，才能进入系统内部。

登录页面包含登录表单、用户名输入框、密码输入框、登录按钮。当用户输入用户名和密码，点击登录按钮时，前端应该将用户名和密码通过AJAX发送到后台，以实现后端验证。后端应该接收到前端发送的用户名和密码，并根据数据库中的用户信息表进行验证。如果用户名密码与数据库匹配，则返回登录成功的状态码，页面跳转至系统首页。否则，返回登录失败的状态码。

如果登录成功，后端应该在服务器端记录用户的登录状态，并将用户的ID等信息存储在会话中，以便后续的操作。前端根据后台返回的登录状态，页面进行跳转。如果登录成功，可以跳转到主页等页面；如果登录失败，给出相应的提示信息并保留在登录页面。

登录页面如图5.1所示：
图 5.1 登录页面

5.1.2 个人中心模块

个人中心是一个重要的模块，该模块向用户展示个人基本信息，如用户名、姓、名、邮箱、手机、角色等。用

户可以在该模块中进行修改个人信息和修改密码的操作。在实现该模块时，需要对用户信息进行权限控制，保证个人信息的安全性、性能和用户体验。
个人中心页面如图5.2所示：

图 5.2 个人中心页面

5.1.3 文件导出模块

用户可以下载导出活动信息表、报名信息表、用户信息表等。后端收到用户的下载请求，将数据库中的表导出为Excel文件，返回给前端文件编号，前端再去通过编号请求下载文件接口，请求成功，浏览器执行下载操作。本系统的下载分为两种方式：默认下载和批量下载，默认下载，即下载用户当前在页面所看到的全部数据。批量下载下载用户在页面表格中勾选的数据。后端在处理文件的导出时，采用了python中的xlwt包，将数据转换为Excel文件。导出的活动表文件如图5.3所示：

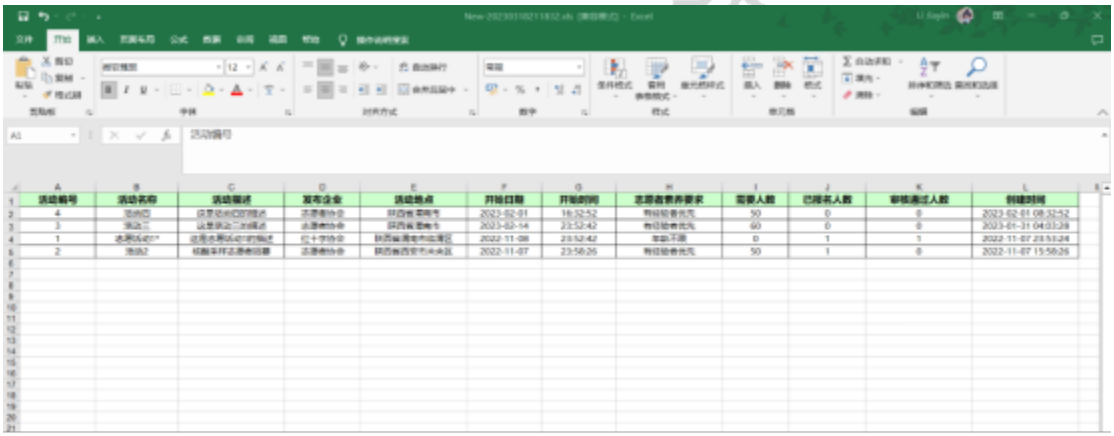


图 5.3 活动表Excel文件

文件导出部分实现代码如下：

```
import xlwt, datetime
from xlwt import *
    def write_to_excel(n, head_data, records, download_url):
     timestr = datetime.datetime.now().strftime("%Y%m%d%H%M%S")
wbk = xlwt.Workbook()
     sheet1 = wbk.add_sheet('sheet1', cell_overwrite_ok=True)
for filed in range(0, len(head_data)):
     sheet1.write(0, filed, head_data[filed], excel_head_style())
for row in range(1, n + 1):
for col in range(0, len(head_data)):
     sheet1.write(row, col, records[row - 1][col], excel_record_style())
sheet1.col(col).width = 300 * 20
```

```
wbk.save(download_url + 'New-' + timestr + '.xls')
return timestr
```

5.1.4 接口文档模块

本系统接入了接口文档，接口文档模块是一份描述系统或应用程序接口的文档，包括接口的设计、功能、参数、返回值、错误码等信息。本系统的接口接口文档模块的实现使用Django REST Swagger技术，Django REST Swagger是一个Django应用程序，它可以自动生成Web API的文档。它使用Swagger规范（也称为OpenAPI规范）来描述RESTful API，并提供交互式的文档页面，以便更容易地了解API的设计和使用方法。

接口文档页面如图5.4所示：

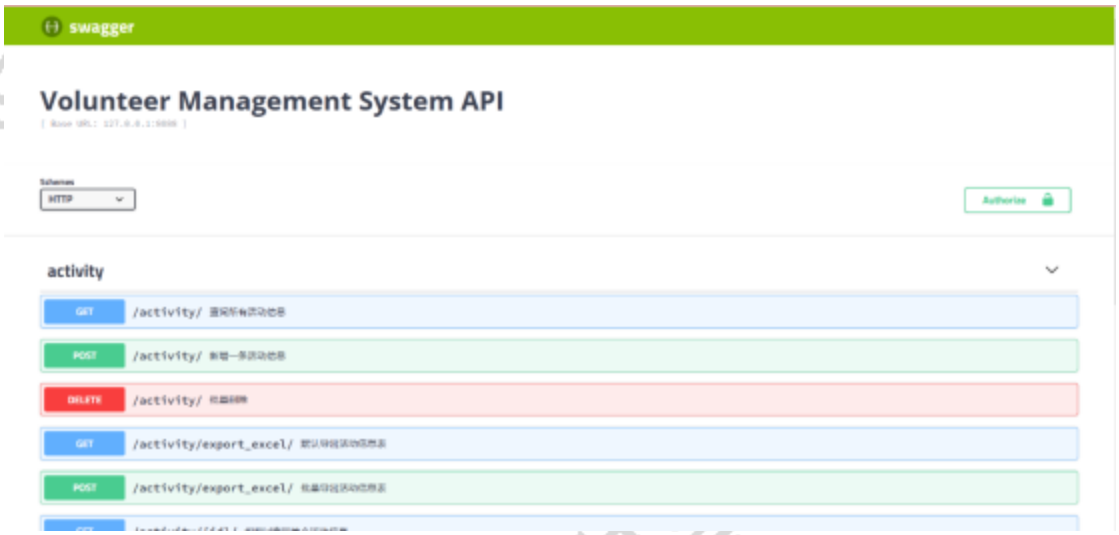


图 5.4 接口文档页面

5.2 普通用户模块

普通用户登录系统可以在活动列表页面，查询自己感兴趣的活动，报名活动。在我的报名页面，查看自己的报名信息，报名状态，修改报名信息。在个人中心页面，用户可以对个人信息和密码进行修改。

5.2.1 活动列表模块

进入活动列表模块，普通用户可以查看公益活动信息，查询自己感兴趣的公益活动，可以看到每个需要多少人数、已报名的人数、报名审核通过的人数，对自己想要参加的公益活动进行报名。

普通用户活动列表如图5.5 所示：

图 5.5 活动列表页面

5.2.2 报名列表模块

普通用户在我的报名页面，可以看到自己的报名信息，如果报名信息有误，可以自己的报名信息进行编辑操作。报名完成后，报名信息等待公益企业审核，审核完毕后，普通用户可以查看自己的报名状态，已通过审核或者

未通过审核。查看公益企业的信息通知。
我的报名页面如图5.6所示：

图 5.6 我的报名页面

5.3 公益企业用户模块

公益企业用户的主要功能模块有两个，活动管理和报名管理。根据公益企业的自身需求，发布公益活动，招募志愿者。公益活动发布后，普通用户可以进行报名，公益企业在报名管理界面对志愿者的报名信息进行审核。

5.3.1 活动管理模块

包括活动发布、活动编辑、活动查看、活动删除、活动导出、批量删除、批量导出等功能，可以帮助公益企业发布新的志愿者活动信息，吸引志愿者参与活动，同时也方便公益企业机构管理和调度活动。同时还包含活动数据统计与分析，对活动的需求人数，报名人数、审核通过人数等进行统计分析，为公益企业用户提供数据支撑，帮助公益企业更好地了解和优化活动。公益活动数据统计分析，是由后端进行根据报名信息进行计算统计，然后返回给前端。前端页面进行展示。

公益企业的活动管理模块前端页面如图5.7所示：

图 5.7 活动管理页面

活动管理部分实现代码如下：

```
from .serializers import ActivityModelSerializer
from utils.pagination import StandardPageNumberPagination
import sys, os
from utils.excel import *
class ActivityListAPIView(APIView):
    queryset = Activity.objects.all() # queryset 指明该视图集在查询数据时使用的查询集
    serializer_class = ActivityModelSerializer # serializer_class 执行该视图在进行序列化或者反序列化
```

```
时使用的序列化器
def get(self, request, *args, **kwargs):
"""
查询所有活动信息
"""
response = {'success': True}
activity_list = Activity.objects.all()
    paging_status = request.GET.get("pagingStatus")
for activity_item in activity_list:
    apply_activity = Activity.objects.get(id=activity_item.id)
    apply_person_num = apply_activity.apply_set.all().count()
    pass_person_num = apply_activity.apply_set.filter(apply_status=1).count()
    Activity.objects.filter(id=activity_item.id).update(
    apply_person_num=apply_person_num, pass_person_num=pass_person_num)
total = activity_list.count()
activity_serializers = ActivityModelSerializer(activity_list, many=True, context={'request':
request})
    pagination = StandardPageNumberPagination()
pg_data = pagination.paginate_queryset(queryset=activity_serializers.data, request=request,
view=self)
if paging_status == 'false':
response['data'] = activity_serializers.data
else:
response['data'] = pg_data
response['total'] = total
return Response(response)
@swagger_auto_schema(
operation_summary="创建活动", request_body=ActivityModelSerializer
)
def post(self, request):
"""
新增一条活动信息
"""
data = request.data
    serializer = ActivityModelSerializer(data=data)
serializer.is_valid(raise_exception=True)
serializer.save()
return Response({
'success': True,
'data': {
'message': '创建成功!'
}
})
def delete(self, request, *args, **kwargs):
"""
批量删除
"""
```

```
    delete_id = request.query_params.get('deleteId', None)
if not delete_id:
return Response({'message': '数据不存在！'})
for i in delete_id.split(','):
    get_object_or_404(Activity, pk=int(i)).delete()
response = {
'success': True,
'data': {
'message': '删除成功！'
},
}
return Response(response)
class ActivityDetailAPIView(APIView):
def get(self, request, pk):
"""
根据id查询单个活动信息
"""
# 查询pk指定的模型对象
try:
activity = Activity.objects.get(id=pk)
    apply_person_num = activity.apply_set.all().count()
pass_person_num = activity.apply_set.filter(apply_status=1).count() Activity.objects.filter
(id=activity.id).update(apply_person_num=apply_person_num, pass_person_num=pass_person_num)
except Activity.DoesNotExist:
    return Response({'success': True, 'data': {'message': '数据不存在！'}})
# 创建序列化器进行序列化
    serializer = ActivityModelSerializer(instance=activity)
# 响应
response = {
'success': True,
'data': serializer.data,
}
return Response(response)
def put(self, request, pk):
"""
根据id修改指定活动信息
"""
# 根据pk所指定的模型对象
try:
activity = Activity.objects.get(id=pk)
except Activity.DoesNotExist:
return Response({'message': '数据不存在！'})
# 获取前端传入的请求体数据
# 创建序列化器进行反序列化操作
serializer = ActivityModelSerializer(instance=activity, data=request.data, partial=True)
# 校验
serializer.is_valid(raise_exception=True)
```

```
serializer.save()
# 响应
response = {
'success': True,
'data': {
'message': '更新成功'
},
}
return Response(response)
def delete(self, request, pk):
"""
根据id删除指定活动信息
"""
# 查询pk所指定的模型对象
try:
activity = Activity.objects.get(id=pk)
except Activity.DoesNotExist:
return Response({'message': '数据不存在!'})
activity.delete()
# 响应
response = {
'success': True,
'data': {
'message': '删除成功!'
},
}
return Response(response)
```

5.3.2 报名管理模块

公益企业在报名管理模块，可以查看志愿者的报名新息，对志愿者的报名信息进行审核。审核状态分为三种状态，待审核、审核已通过、审核未通过。审核结束后，系统会自动给普通用户发送邮件通知，普通用户查看自己的报名状态，以及邮件通知。

报名管理页面如图5.8所示：

图 5.8 报名管理页面

5.4 管理员模块

管理员的功能包含了普通用户和公益企业用户的功能模块。同时，管理员还有额外的用户管理功能。

5.4.1 用户管理模块

管理员能够对系统所有用户进行管理，包括登录注册和用户信息管理，以便用户方便快捷地登录系统并管理自己的个人信息。管理员在用户管理模块，可以进行添加用户、查询用户、查看用户列表、修改用户、注销用户、导出用户、批量删除、批量导出等操作。对于忘记密码的用户，用户联系管理员后，管理员可以对该用户进行重置密码操作，用户密码初始化为123456。用户使用初始密码登录系统后，可自行在个人中心页面进行修改密码操作。对应违规用户或者长期未使用的用户，管理员可以对用户进行账号注销，账号注销后，用户不能再登录系统。

用户管理页面如图5.9所示：

图 5.9 用户管理页面

6 系统测试

6.1 测试目的

系统测试的测试目的主要包含以下几个方面：

验证系统是否满足和规格说明书的要求，是否符合预期的功能和性能要求；

评估系统的可靠性、稳定性和安全性，检查系统在异常情况下的表现和处理能力；

检查系统的易用性、可维护性和可扩展性，评估系统的可操作性和可维护性；

确保系统符合相关法律法规和行业标准，检查系统的合规性和安全性；

检查系统的兼容性，验证系统在不同的硬件、操作系统和浏览器等环境下的兼容性；

发现并修复系统中的缺陷和问题，提高系统的质量和可靠性。

6.2 接口测试

本系统所有API的测试使用Postman工具进行测试，Postman是一款常用的API测试工具，其主要特点是可以对API进行快速、方便的测试和调试。在Postman中，选择请求方法（如GET、PATCH、POST、PUT、DELETE等）和输入请求URL来创建接口的请求，在请求中添加请求参数，包括查询参数、请求体参数、请求头参数等。通过点击“发送”按钮来发送请求，Postman会自动处理请求并显示响应结果。请求结束后，通过Postman的界面来查看响应结果，响应结果包括状态码、响应头、响应体等信息。

测试查询报名列表接口过程如图6.1所示：

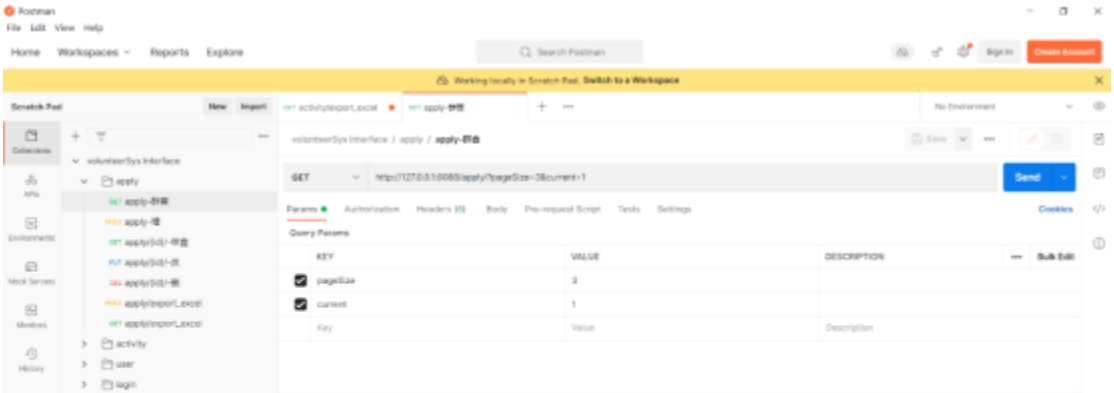
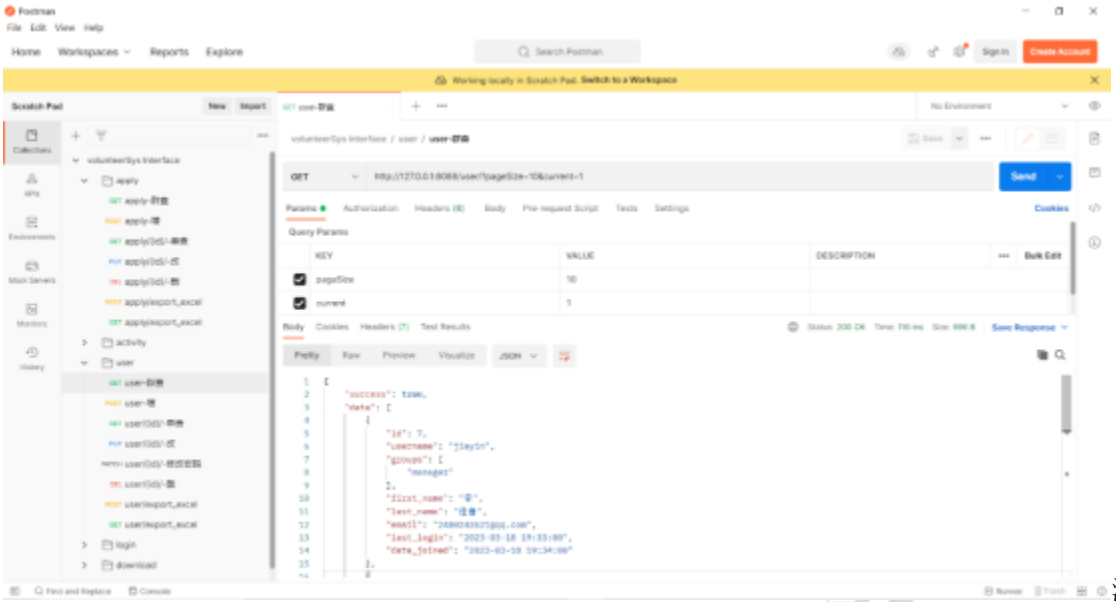


图 6.1 查询报名列表接口测试

图 6.2 查询用户列表接口测试



测试查询用户列表接口过程如图6.2所示：

6.3 测试结果

经过多组测试数据对系统进行测试，该系统功能相对齐全，基本必要满足信息处理的需求，用户登录、注册、活动管理、报名管理等功能都能实现，符合预期，系统的性能完好，可靠性良好。

结论

经过本论文的研究与分析，基于Django+React的疫情防控社区志愿者管理系统的设计实现取得了良好的效果，系统实现了疫情防控志愿者的招募、管理、统计、审核等功能，提高了志愿者的管理效率和任务完成率。在实际应用过程中，该系统得到了广泛的应用和认可，为疫情防控工作提供了有力的信息化支持。未来，可以在系统的基础上进一步完善功能和优化体验，为社区疫情防控工作提供更好的服务。

此外，本论文还对Django和React技术的特点和优势进行了分析和总结，对于开发类似系统的研究者具有一定的参考价值。其中，Django作为一种高效、易用、功能丰富的Web开发框架，能够快速构建高质量的Web应用；React作为一种优秀的前端开发框架，能够提供高效、可重用、易维护的UI组件和极佳的用户体验。通过使用Django和React相结合的方式，可以更加高效地开发出功能丰富、易于维护和后期扩展的数字化Web应用管理系统。

总之，本论文的研究成果为疫情防控社区志愿者管理系统的开发提供了有力的支持和借鉴，对于推动信息化建设和社区防控工作具有一定的实际意义和价值。

在未来的研究中，还可以进一步完善该系统，例如增加更加精细化的权限控制、优化用户体验、增加更加实用的功能等。同时，我们也可以探索其他前后端技术的应用，例如Vue.js、Angular等，以提高Web应用的开发效率和性能。另外，我们可以将该系统应用到其他领域，例如社区服务、志愿者管理等，为社区的数字化建设和发展做出更多的贡献。最后，我们也可以将研究重心转移到其他领域，例如智慧城市、医疗健康等，以应用技术手段推动信息化建设的进步和发展。

参考文献

[1]王志伟.新冠疫情防控集中隔离管理系统的应用与实践[J].通信管理与技术,2022(02):27-31.

[2]曾丽娟,邱毅,段涛,李建水,唐啸龙,邓大炜.基于B/S架构医院志愿者管理系统的设计与实现[J].医学信息,2021,34(07):27-30+34.

[3]赵春霞. 基于Web技术的志愿者服务管理系统设计与实现[D].南京:南京理工大学,2017.

[4]刘振东. 威海志愿者管理系统的设计与实现[D].哈尔滨:哈尔滨工业大学,2019.
[5]孔祥祺.志愿者积分管理系统的设计思路探讨[J].赤峰学院学报(自然科学版),2016,32(04):21-24+30.
[6]蔡栋,金波.基于Fabric的区块链慈善业务管理系统设计[J].中国高新科技,2020(03):62-65+68.
[7]马春晓,叶青,吕明.志愿活动管理系统的设计与实现[J].工业控制计算机,2022,35(01):135-136+139.
[8]黄智霖.高校校园活动管理系统设计——以厦门华厦学院为例[J].信息技术与信息化,2021(12):72-74+78.
[9]李珊. 基于微信小程序的学生活动管理系统的设计与实现[D].广东:广东工业大学,2019.
[10]穆荣,刘蒙蒙,王晓路.大学生社团活动管理系统的设计[J].电子世界,2017(15):159-162+166.
[11]刘新宇.疫情防控下高校志愿服务项目建设的路径探索[J].黑龙江教育(高教研究与评估),2022(10):90-92.
[12]金鑫,董耀众,张大伟,李伟良,肖磊,牟霄寒,孙建刚.基于移动应用的疫情防控管理系统的设计与构建——以电力企业为例[J].办公自动化,2022,27(18):6-9.
[13]金鑫,董耀众,张大伟,李伟良,肖磊,牟霄寒,孙建刚.基于移动应用的疫情防控管理系统的设计与构建——以电力企业为例[J].办公自动化,2022,27(18):6-9.
[14]Janhavi Desale,Kunal Gautama,Saish Khandare,Vedant Parikh,Dhanashree Toradmalle. NGO Support Software Solution: for effective reachability[J]. International Journal of Education and Management Engineering(IJEME),2020,10(6).
[15]Noor Afiza Mat Razali,Nurjannatul Jannah Aqilah Md Saad,Hasmeda Erna Che Hamid,Muhammad Ramzul Abu Bakar,Khairul Khalil Ishak,Nor Asiakin Hasbullah,Norulzahrah Mohd Zainudin,Suzaimah Ramli, Norshahriah Wahab. Volunteer Management System for Disaster Management[J]. International Journal of Recent Technology and Engineering (IJRTE),2019,7(5s4).
[16]. Deedia Inc.; Researchers Submit Patent Application, "Systems And Methods For Service Opportunity Management And Volunteer Management", for Approval (USPTO 20200142931)[J]. Politics & Government Week,2020.
[17]. InitLive; InitLive Donates Volunteer Management System To Aid In COVID-19 Relief[J]. Medical Letter on the CDC & FDA,2020.
[18]Kim Eunjung Cuskelly Graham. A Systematic Quantitative Review Of Volunteer Management in Events [J]. Event Management,2017,21(1).
[19]Jiang Qiwen,Zhu Xueyuan,Chen Lianghua,Zhao Ziyuan,Chen Yilong. Research on Time-Driven Activity-Based Management System of Public Hospitals [J]. Frontiers in Public Health,2022,9.
[20]Supun Wijekoon. University Activity Management System[J]. Journal of Information Technology & Software Engineering,2021,11(7).
致谢
在本论文的完成过程中，我获得了许多人的支持和帮助，在此向他们致以最诚挚的谢意。
首先，我要感谢我的论文指导教师，她在整个研究过程中给予了我耐心的指导和关心，在研究方法、技术选型、论文撰写等方面提供了宝贵的意见和建议。
其次，我要感谢我的家人和朋友，在繁忙的学业生活中，他们一直支持我、鼓励我，给我以精神上的支持和帮助。我还要感谢我大学期间，计科教研室的老师们，给我传授了很多专业知识，才有了我今天的编程开发水平。
同时，我也要感谢我在实习期间公司的同事，在我毕业设计系统开发的过程中，给我提供了很大的帮助与支持，在我每次遇到技术难点和无法解决的bug时，他都会耐心的帮我解决，给我提供了很大的技术支持。
最后，我要感谢所有在疫情防控一线奋斗的医护人员和志愿者，是他们的无私奉献的精神，为我们创造了安全、舒适、健康的生活环境，让我们深刻认识到科技在疫情防控中的重要作用。
感谢以上所有人的支持和帮助，让我的论文能够顺利完成。
附录
报名管理相关实现代码：
```
from django.http import HttpResponse
from rest_framework.generics import get_object_or_404
from rest_framework.views import APIView
from activity.models import Activity
```

```
from .models import Apply # 导入对应的模型类
from rest_framework.response import Response
from .serializers import ApplyModelSerializer # 导入对应的序列化器
from utils.pagination import StandardPageNumberPagination
import sys, os
from utils.excel import *
class ApplyListAPIView(APIView):
queryset = Apply.objects.all() # 指明查询集
    serializer_class = ApplyModelSerializer # 指明所使用的序列化器
def get(self, request, *args, **kwargs):
"""
查询所有报名信息
"""
response = {'success': True}
apply_list = Apply.objects.all()
total = apply_list.count()
apply_serializers = ApplyModelSerializer(apply_list, many=True, context={'request': request})
    pagination = StandardPageNumberPagination()
pg_data = pagination.paginate_queryset(queryset=apply_serializers.data, request=request, view=self)
response['data'] = pg_data
response['total'] = total
return Response(response)
def post(self, request):
"""
新增一条报名信息
"""
# 获取前端传入请求体数据
data = request.data
# 创建序列化器进行反序列化操作
    serializer = ApplyModelSerializer(data=data)
# 调用序列化器的is_valid方法进行校验
serializer.is_valid(raise_exception=True)
# 调用序列化器的save方法进行执行create方法
serializer.save()
# 响应
response = {
'success': True,
'data': {
'message': 'success!'
}
}
return Response(response)
def delete(self, request, *args, **kwargs):
"""
批量删除
"""
    delete_id = request.query_params.get('deleteId', None)
```

```
if not delete_id:
return Response({'message': '数据不存在!'})
for i in delete_id.split(','):
    get_object_or_404(Apply, pk=int(i)).delete()
response = {
'success': True,
'data': {
'message': '删除成功!'
},
}
return Response(response)
class ApplyDetailAPIView(APIView):
def get(self, request, pk):
"""
根据id查询指定报名信息
"""
# 查询pk指定的模型对象
try:
apply = Apply.objects.get(id=pk)
except Apply.DoesNotExist:
    return Response({'success': True, 'data': {'message': '数据不存在!'}})
# 创建序列化器进行序列化
    serializer = ApplyModelSerializer(instance=apply)
# 响应
response = {
'success': True,
'data': serializer.data,
}
return Response(response)
def put(self, request, pk):
"""
根据id修改指定报名信息
"""
# 根据pk所指定的模型对象
try:
apply = Apply.objects.get(id=pk)
except Apply.DoesNotExist:
return Response({'message': '数据不存在!'})
# 获取前端传入的请求体数据
# 创建序列化器进行反序列化操作
serializer = ApplyModelSerializer(instance=apply, data=request.data, partial=True)
# 校验
serializer.is_valid(raise_exception=True)
serializer.save()
# 响应
response = {
'success': True,
```

```
'data': {
'message': '更新成功'
},
}
return Response(response)
def delete(self, request, pk):
"""
根据id删除指定报名信息
"""
# 查询pk所指定的模型对象
try:
apply = Apply.objects.get(id=pk)
except Apply.DoesNotExist:
return Response({'message': '数据不存在!'})
apply.delete()
# 响应
response = {
'success': True,
'data': {
'message': '删除成功!'
},
}
return Response(response)
class ApplyExportExcelAPIView(APIView):
def post(self, request):
"""
批量导出报名信息表
"""
apply_codes = request.data.get("apply_code")
n = len(apply_codes)
# 表头字段
head_data = [u'报名编号', u'姓名', u'年龄', u'性别', u'地址', u'电话', u'报名活动', u'报名状态', u'申
请时间']
# 查询记录数据
records = []
for apply_code in apply_codes:
if apply_code != "":
    apply_obj = Apply.objects.get(id=apply_code)
id = apply_obj.id
name = apply_obj.name
age = apply_obj.age
    sex = '未知' if apply_obj.sex == 0 else '男' if apply_obj.sex == 1 else '女'
address = apply_obj.address
tel = apply_obj.tel
    apply_status = '待审核' if apply_obj.apply_status else '已审核' if apply_obj.apply_status == 1
else '未通过'
    apply_time = apply_obj.apply_time.strftime("%Y-%m-%d %H:%M:%S")
```

```
    belonging_activity_id = apply_obj.belonging_activity_id
     apply_activity = Activity.objects.get(id=belonging_activity_id).name
record = []
record.append(id)
record.append(name)
record.append(age)
record.append(sex)
record.append(address)
record.append(tel)
record.append(apply_activity)
record.append(apply_status)
record.append(str(apply_time))
records.append(record)
# 获取当前路径
cur_path = os.path.abspath('.')
# 设置生成文件所在路径
download_url = cur_path + '\\upload\\'
# 写入数据到excel中
     ret = write_to_excel(n, head_data, records, download_url)
return HttpResponse(ret)
def get(self, request):
"""
默认导出报名信息表
"""
applys = Apply.objects.all()
n = len(applys)
# 表头字段
head_data = [u'报名编号', u'姓名', u'年龄', u'性别', u'地址', u'电话', u'报名活动', u'报名状态', u'申
请时间']
# 查询记录数据
records = []
for apply_obj in applys:
id = apply_obj.id
name = apply_obj.name
age = apply_obj.age
     sex = '未知' if apply_obj.sex == 0 else '男' if apply_obj.sex == 1 else '女'
address = apply_obj.address
tel = apply_obj.tel
     apply_status = '待审核' if apply_obj.apply_status else '已审核' if apply_obj.apply_status == 1
else '未通过'
     apply_time = apply_obj.apply_time.strftime("%Y-%m-%d %H:%M:%S")
    belonging_activity_id = apply_obj.belonging_activity_id
     apply_activity = Activity.objects.get(id=belonging_activity_id).name
record = []
record.append(id)
record.append(name)
record.append(age)
```

```
record.append(sex)
record.append(address)
record.append(tel)
record.append(apply_activity)
record.append(apply_status)
record.append(str(apply_time))
records.append(record)
# 获取当前路径
cur_path = os.path.abspath('.')
# 设置生成文件所在路径
download_url = cur_path + '\\upload\\'
# 写入数据到excel中
    ret = write_to_excel(n, head_data, records, download_url)
return HttpResponse(ret)
```

用户管理相关实现代码：

```
from django.contrib.auth.models import User, Group
from django.http import HttpResponse
from rest_framework.generics import get_object_or_404
from rest_framework.response import Response
from rest_framework import viewsets
from rest_framework.views import APIView
from users.serializers import UserSerializer, GroupSerializer
   from django.contrib.auth.hashers import make_password, check_password
from utils.pagination import StandardPageNumberPagination
import sys, os
from utils.excel import *
class UserListAPIView(APIView):
    queryset = User.objects.all().order_by('-date_joined')
serializer_class = UserSerializer
def get(self, request, *args, **kwargs):
"""
查询所有用户信息
"""
response = {'success': True}
user_list = User.objects.all()
paging_status = request.GET.get('pagingStatus')
total = User.objects.all().count()
user_serializers = UserSerializer(user_list, many=True, context={'request': request})
pagination = StandardPageNumberPagination()
pg_data = pagination.paginate_queryset(queryset=user_serializers.data, request=request, view=self)
if paging_status == 'false':
response['data'] = user_serializers.data
else:
response['data'] = pg_data
response['data'] = pg_data
response['total'] = total
return Response(response)
```

```
def post(self, request):
"""
新增一条用户信息
"""
# 获取前端传入请求体数据
data = request.data.copy()
password_ = data["password"]
data["password"] = make_password(password_)
# 创建序列化器进行反序列化操作
serializer = UserSerializer(data=data)
# 调用序列化器的is_valid方法进行校验
serializer.is_valid(raise_exception=True)
# 调用序列化器的save方法进行执行create方法
serializer.save()
# 响应
response = {
'success': True,
'data': {
'message': 'success!'
}
}
return Response(response)
def delete(self, request, *args, **kwargs):
"""
批量删除
"""
     delete_id = request.query_params.get('deleteId', None)
if not delete_id:
return Response({'message': '数据不存在!'})
for i in delete_id.split(','):
get_object_or_404(User, pk=int(i)).delete()
response = {
'success': True,
'data': {
'message': '删除成功!'
},
}
return Response(response)
class UserDetailAPIView(APIView):
def get(self, request, pk):
"""
根据id查询单个用户信息
"""
# 查询pk指定的模型对象
try:
user = User.objects.get(id=pk)
except User.DoesNotExist:
```

```
    return Response({'success': True, 'data': {'message': '数据不存在!'}})
# 创建序列化器进行序列化
serializer = UserSerializer(instance=user)
# 响应
response = {
'success': True,
'data': serializer.data,
}
return Response(response)
def put(self, request, pk):
"""
根据id修改指定用户信息
"""
# 根据pk所指定的模型对象
try:
user = User.objects.get(id=pk)
except User.DoesNotExist:
return Response({'message': '数据不存在!'})
# 获取前端传入的请求体数据
data = request.data.copy()
if 'password' in data.keys():
password_ = data['password']
data['password'] = make_password(password_)
# 创建序列化器进行反序列化操作
serializer = UserSerializer(instance=user, data=data, partial=True)
# 校验
serializer.is_valid(raise_exception=True)
serializer.save()
# 响应
response = {
'success': True,
'data': {
'message': '更新成功'
},
}
return Response(response)
def patch(self, request, pk):
"""
根据id修改用户密码
"""
# 根据pk所指定的模型对象
try:
user = User.objects.get(id=pk)
except User.DoesNotExist:
return Response({'message': '数据不存在!'})
# 获取前端传入的请求体数据
data = request.data.copy()
```

```
new_password = data['new_password']
old_password = data['old_password']
if check_password(old_password, user.password):
    new_data = {'password': make_password(new_password)}
# 创建序列化器进行反序列化操作
serializer = UserSerializer(instance=user, data=new_data, partial=True)
else:
return Response({'message': '旧密码输入错误！'})
# 校验
serializer.is_valid(raise_exception=True)
serializer.save()
# 响应
response = {
'success': True,
'data': {
'message': '更新成功'
},
}
return Response(response)
def delete(self, request, pk):
"""
根据id删除指定用户信息
"""
# 查询pk所指定的模型对象
try:
user = User.objects.get(id=pk)
except User.DoesNotExist:
return Response({'message': '数据不存在！'})
user.delete()
# 响应
response = {
'success': True,
'data': {
'message': '删除成功！'
},
}
return Response(response)
class GroupViewSet(viewsets.ModelViewSet):
"""
允许组查看或编辑的API路径
"""
queryset = Group.objects.all()
serializer_class = GroupSerializer
class LoginView(APIView):
def post(self, request):
"""
登录接口
```

```
"""
username = request.data['username']
password = request.data['password']
print(password, username)
    user = User.objects.filter(username=username).first()
if user and check_password(password, user.password):
    return Response({'status': 'ok', 'currentAuthority': username, 'user_id': user.id, 'type':
'account'})
else:
    return Response({'status': 'failed', 'code': 400})
class UserExportExcelAPIView(APIView):
def post(self, request):
"""
批量导出用户信息表
"""
user_codes = request.data.get("user_code")
n = len(user_codes)
# 表头字段
head_data = [u'用户编号', u'用户名', u'电子邮箱', u'姓', u'名', u'上次登录时间', u'注册时间', u'角
色']
# 查询记录数据
records = []
for user_code in user_codes:
if user_code != "":
user_obj = User.objects.get(id=user_code)
id = user_obj.id
username = user_obj.username
email = user_obj.email
first_name = user_obj.first_name
last_name = user_obj.last_name
    last_login = user_obj.last_login.strftime("%Y-%m-%d %H:%M:%S") if user_obj.last_login != None
else ''
    date_joined = user_obj.date_joined.strftime("%Y-%m-%d %H:%M:%S")
group = '管理员' if str(Group.objects.get(user=user_obj)) == 'manager' else '发布企业' if str(Group.
objects.get(user=user_obj)) == 'company' else '普通用户'
record = []
record.append(id)
record.append(username)
record.append(email)
record.append(first_name)
record.append(last_name)
record.append(str(last_login))
record.append(str(date_joined))
record.append(group)
records.append(record)
# 获取当前路径
cur_path = os.path.abspath('.')
```

```
# 设置生成文件所在路径
download_url = cur_path + '\\upload\\'
# 写入数据到excel中
    ret = write_to_excel(n, head_data, records, download_url)
return HttpResponse(ret)
def get(self, request):
"""
默认导出用户信息表
"""
users = User.objects.all()
n = len(users)
# 表头字段
head_data = [u'用户编号', u'用户名', u'电子邮箱', u'姓', u'名', u'上次登录时间', u'注册时间', u'角
色']
# 查询记录数据
records = []
for user_obj in users:
# if user_code != "":
# user_obj = User.objects.get(id=user_code)
id = user_obj.id
username = user_obj.username
email = user_obj.email
first_name = user_obj.first_name
last_name = user_obj.last_name
    last_login = user_obj.last_login.strftime("%Y-%m-%d %H:%M:%S") if user_obj.last_login != None
else ''
    date_joined = user_obj.date_joined.strftime("%Y-%m-%d %H:%M:%S")
group = '管理员' if str(Group.objects.get(user=user_obj)) == 'manager' else '发布企业' if str(Group.
objects.get(user=user_obj)) == 'company' else '普通用户'
record = []
record.append(id)
record.append(username)
record.append(email)
record.append(first_name)
record.append(last_name)
record.append(str(last_login))
record.append(str(date_joined))
record.append(group)
records.append(record)
# 获取当前路径
cur_path = os.path.abspath('.')
# 设置生成文件所在路径
download_url = cur_path + '\\upload\\'
# 写入数据到excel中
    ret = write_to_excel(n, head_data, records, download_url)
return HttpResponse(ret)
class DownloadAPIView(APIView):
```

```
def post(self, request, offset):
"""
公共下载excel文件方法
"""
from django.http import StreamingHttpResponse
def file_iterator(file_name, chunk_size=512):
with open(file_name, 'rb') as f:
while True:
c = f.read(chunk_size)
if c:
yield c
else:
break
# 显示在弹出对话框中的默认的下载文件名
the_file_name = 'New-' + offset + '.xls'
# 获取当前路径
cur_path = os.path.abspath('.')
# 设置生成文件所在路径
download_url = cur_path + '\\upload\\'
response = StreamingHttpResponse(file_iterator(download_url + 'New-' + offset + '.xls'))
    response['Content-Type'] = 'application/octet-stream'
response['Content-Disposition'] = 'attachment;filename="{0}"'.format(the_file_name)
return response
```

报名管理序列化器相关代码如下：

```
from rest_framework import serializers
from .models import Apply
from activity.models import Activity
class ApplyModelSerializer(serializers.Serializer):
'''报名信息序列化器'''
SEX_CHOICE = [
(0, '未知'),
(1, '男'),
(2, '女')
]
STATUS_CHOICE = [
(0, '待审核'),
(1, '已审核'),
(2, '未通过')
]
    id = serializers.IntegerField(label='Id', read_only=True)
    name = serializers.CharField(label='姓名', max_length=50, required=True)
    age = serializers.IntegerField(label='年龄', required=True)
    sex = serializers.ChoiceField(choices=SEX_CHOICE)
    address = serializers.CharField(label='家庭住址', required=True)
    tel = serializers.CharField(label='联系方式', required=True)
    apply_status = serializers.ChoiceField(label='报名状态', choices=STATUS_CHOICE, default=0)
    apply_time = serializers.DateTimeField(label='报名时间', read_only=True, required=False,
```

```
format='%Y-%m-%d %H:%M:%S')
belonging_activity = serializers.PrimaryKeyRelatedField(queryset=Activity.objects.all())
# 自定义额外字段
belonging_activity_name = serializers.CharField(source='belonging_activity.name', read_only=True)
# 局部全局钩子同Serializer类, 自定义校验手机号码方法
def validate_tel(self, value):
if (len(value) != 11):
raise serializers.ValidationError('电话号码长度不对!')
return value
# 重写create方法
def create(self, validated_data):
    return Apply.objects.create(**validated_data)
    # instance要被修改的对象, validated_data代表校验后用来改instance的数据
    def update(self, instance: Apply, validated_data):
# 用户名不能被修改
    Apply.objects.filter(pk=instance.id).update(**validated_data)
return instance
```