**基于 Python 的社区养老管理系统设计与实现**

姓 名： 宣奔

学 号： 1230696180

指导教师： 刘宁

**2025年7月18日**

**摘 要**

我国老龄化加剧，社区养老成为社会化服务的关键环节，但多数机构仍依赖人工台账与纸质档案，导致信息滞后、检索耗时、统计困难、流程无章。为此，本文设计并实现一套基于 Python 全栈的数字化管理平台。系统采用浏览器/服务器模式：后端以 Django 为核心快速封装业务接口，前端借助 Vue.js 与 Ant-Design-Vue 打造响应式交互界面，数据库选用 MySQL 5.7 保障数据一致与安全。研究首先通过走访机构管理者、护理员与家属，提炼出“住户档案、护理人员、亲属联络、健康史、房态、活动安排、账户权限、操作日志、系统配置”九大高频需求，并补充性能、可靠性、可维护等非功能指标。设计阶段坚持分步实施、易用优先、流程标准化与可扩展原则，完成整体架构、功能拆分及数据模型。实现环节详述环境搭建、接口开发及页面交互逻辑。最终经功能、并发与安全测试验证，平台可显著提升运营效率与服务质量，为社区养老智慧化升级提供可行路径。

**关键词：**社区养老；Python；Django；Vue.js；管理系统

**Abstract**

Amid China’s deepening demographic shift, community-based eldercare has become pivotal to the social care network, yet most facilities still rely on manual ledgers and paper records. This legacy approach triggers delayed updates, sluggish retrieval, error-prone statistics, and non-standardized workflows. To overcome these bottlenecks, we present a full-Python digital management platform. The solution follows a Browser/Server pattern: Django powers the back-end for rapid API delivery, Vue.js coupled with Ant-Design-Vue renders a responsive front-end, and MySQL 5.7 secures transactional data integrity. Field interviews with administrators, caregivers and family members distilled nine priority domains—resident profiles, staff roster, family contacts, medical histories, room occupancy, activity scheduling, user rights, audit logs and system settings—supplemented by non-functional targets for throughput, reliability and maintainability. The design adheres to phased roll-out, usability-first, process standardization and future-proof extensibility, yielding a layered architecture, modular functions and normalized schema. Implementation details span environment setup, interface coding and UI workflows. Comprehensive functional, load and security tests confirm that the system markedly elevates operational efficiency and service quality, offering a pragmatic blueprint for the digital transformation of community eldercare.

**Keywords:** Community Elderly Care; Python; Django; Vue.js; Management System

目 录

[1 绪论 6](#_Toc165627467)

[1.1 研究背景 6](#_Toc165627468)

[1.2 研究现状 7](#_Toc165627469)

[1.3 研究目标 8](#_Toc165627470)

[2 相关技术介绍 9](#_Toc165627471)

[2.1 Python语言 9](#_Toc165627472)

[2.2 Django框架技术 10](#_Toc165627473)

[2.3 Vue前端技术 12](#_Toc165627474)

[2.3.1 Vue框架 12](#_Toc165627475)

[2.3.2 Ant-Design-Vue组件 13](#_Toc165627476)

[2.4 Node技术 13](#_Toc165627477)

[2.5 MySQL数据库 14](#_Toc165627479)

[3 需求分析 15](#_Toc165627480)

[3.1 系统功能需求分析 15](#_Toc165627481)

[3.1.1 业务需求分析 15](#_Toc165627482)

[3.1.1.1 老人管理业务 17](#_Toc165627482)

[3.1.1.2 护工管理业务 18](#_Toc165627482)

[3.1.1.3 亲属管理业务 18](#_Toc165627482)

[3.1.1.4 病史管理业务 19](#_Toc165627482)

[3.1.1.5 房间管理业务 19](#_Toc165627482)

[3.1.1.6 活动管理业务 20](#_Toc165627482)

[3.1.1.7 用户管理业务 20](#_Toc165627482)

[3.1.1.8 日志管理业务 21](#_Toc165627482)

[3.1.1.9 系统信息管理业务 21](#_Toc165627482)

[3.1.2 数据需求分析 18](#_Toc165627483)

[3.1.2.1 核心数据实体 22](#_Toc165627483)

[3.1.2.2 数据关系与约束 23](#_Toc165627483)

[3.2 系统非功能性需求分析 20](#_Toc165627484)

[3.2.1 系统处理能力需求 20](#_Toc165627485)

[3.2.2 可靠性需求 20](#_Toc165627486)

[3.2.3 可用性需求 21](#_Toc165627487)

[3.2.4 维修性需求 21](#_Toc165627488)

[3.2.5 环境适配需求 21](#_Toc165627489)

[4 系统设计 2](#_Toc165627490)2

[4.1 设计原则 2](#_Toc165627491)2

[4.1.1 阶段开发原则 2](#_Toc165627492)2

[4.1.2 易用性原则 2](#_Toc165627493)2

[4.1.3 业务完整性原则 2](#_Toc165627494)2

[4.1.4 业务规范化原则 2](#_Toc165627495)3

[4.1.5 可扩展性原则 2](#_Toc165627496)3

[4.2 总体架构设计 23](#_Toc165627497)

[4.2.1 B/S 架构设计 23](#_Toc165627497)

[4.2.2 前后端分离架构 23](#_Toc165627497)

[4.2.3 系统分层设计 2](#_Toc165627497)4

[4.3 功能设计 25](#_Toc165627498)

[4.3.1 老人管理模块 25](#_Toc165627498)

[4.3.2 护工管理模块 25](#_Toc165627498)

[4.3.3 亲属管理模块 25](#_Toc165627498)

[4.3.4 病史管理模块 26](#_Toc165627498)

[4.3.5 房间管理模块 26](#_Toc165627498)

[4.3.6 活动管理模块 26](#_Toc165627498)

[4.3.7 用户管理模块 26](#_Toc165627498)

[4.3.8 日志管理模块 27](#_Toc165627498)

[4.3.9 系统信息模块 2](#_Toc165627498)7

[4.4 数据库设计 2](#_Toc165627499)7

[4.4.1 设计规则 2](#_Toc165627500)7

[4.4.2 表结构设计 2](#_Toc165627501)8

[5 系统实现 3](#_Toc165627502)0

[5.1 开发环境搭建 3](#_Toc165627503)0

5.1.1 后端环境配置 30

5.1.2 前端环境配置 32

[5.2 核心功能模块实现 3](#_Toc165627509)2

5.2.1 老人管理模块 32

5.2.2 护工管理模块 36

5.2.3 亲属管理模块 36

5.2.4 病史管理模块 36

5.2.5 房间管理模块 37

5.2.6 活动管理模块 37

5.2.7 用户管理模块 37

5.2.8 日志管理模块 38

5.2.9 系统信息模块 39

[6 系统测试 4](#_Toc165627519)0

[6.1 测试概述 40](#_Toc165627520)

[6.1.1 测试目的与范围 40](#_Toc165627520)

[6.1.2 测试环境 4](#_Toc165627520)1

[6.2 测试用例设计与执行 42](#_Toc165627521)

[6.2.1 功能测试 42](#_Toc165627521)

[6.2.2 性能测试 42](#_Toc165627521)

[6.2.3 安全性测试 4](#_Toc165627521)3

[6.3 测试结果分析 4](#_Toc165627522)3

[7 总结与展望 4](#_Toc165627526)4

[参考文献 44](#_Toc165627527)

# 绪论

## 研究背景

在社会发展的进程中，人口结构的变化始终是一个备受关注的重要议题。随着我国经济社会的持续发展以及医疗卫生条件的显著改善，人口老龄化进程呈现出加速态势。社区养老作为社会化养老服务体系的关键构成部分，其重要性日益凸显，社会对社区养老服务的需求也在与日俱增。

根据国家统计局发布的权威数据，截至 2022 年底，我国 60 岁及以上的老年人口数量已超过 2.8 亿，在总人口中所占比例达到 19.8%。这一数据清晰地表明，我国已经步入老龄化社会，并且老龄化程度正在不断加深。相关预测显示，到 2035 年，我国 60 岁及以上老年人口占总人口的比例将超过 30%，届时我国将正式进入重度老龄化社会。这一趋势不仅给社会经济发展带来了诸多挑战，也对社区养老服务的质量和管理效率提出了更高的要求。

传统的社区养老机构在管理方面，大多依赖人工记录和纸质档案。这种管理模式在信息更新、查询以及数据统计等方面存在明显的弊端。例如，人工记录容易出现信息错误和遗漏，纸质档案的存储和检索也极为不便，导致信息更新滞后、查询困难、数据统计繁琐等问题频繁出现。在面对日益增长的养老服务需求时，传统管理模式难以满足高效化、精细化的管理要求，无法为老年人提供及时、准确、全面的服务。

在此背景下，借助信息技术构建社区养老管理系统成为提升社区养老服务质量和管理效率的关键举措。信息技术具有高效、便捷、准确等诸多优势，能够实现对养老信息的快速处理和共享，为社区养老管理的数字化转型提供有力支持。Python 作为一种简洁高效、易于学习和使用的编程语言，具有丰富的库和框架资源。搭配 Django、Vue.js 等主流框架，Python 能够快速开发出功能完善、交互友好的 Web 应用。这些技术优势为社区养老管理系统的开发提供了可靠的技术支撑，使得开发一款高效、实用的社区养老管理系统成为可能。因此，设计并实现一款基于 Python 的社区养老管理系统具有重要的现实意义，不仅有助于提高社区养老机构的管理水平，还能为老年人提供更加优质、便捷的养老服务。

## 研究现状

目前，国内外针对养老管理系统的研究已经取得了一定的成果，但由于不同国家和地区的养老模式、文化背景以及技术发展水平存在差异，研究重点和成果也各有特点。

国外在养老信息化领域起步较早，经过多年的发展，部分养老管理系统已经实现了较为成熟的功能。例如，美国的 Senior Living CRM 系统在养老管理方面具有代表性。该系统侧重于老人健康数据的追踪与服务个性化推荐，通过与智能手环、血压计等智能设备进行无缝对接，能够实时、精准地监测老人的健康状况，如心率、血压、睡眠质量等。基于这些数据，系统运用先进的数据分析算法，为每位老人提供个性化的健康建议和生活指导，有效提升了养老服务的质量和针对性。此外，欧洲的一些养老管理系统则更注重社交功能的实现。考虑到老年人在精神层面的需求，这些系统通过组织线上活动、建立老人社交圈等方式，为老年人搭建了一个交流互动的平台，丰富了他们的精神文化生活，提升了老人的生活质量。

相比之下，国内相关养老管理系统虽然也有一定的发展，但整体水平仍有待提高。目前，国内多数养老管理系统主要集中于机构内部管理，功能涵盖老人信息登记、护理安排等基础模块。然而，这些系统存在一些明显的问题。在技术架构方面，多数系统采用传统的 JSP + SSH 架构，这种架构前后端耦合度高，导致系统的扩展性和维护性较差，随着业务的发展和需求的变更，系统升级和维护成本高昂。在功能方面，国内系统存在同质化严重的问题，缺乏创新性和差异化竞争。部分开源项目功能不完善，难以直接应用于实际场景。例如，某些系统仅实现了老人信息的基本录入与查询，对于病史管理、活动管理等核心业务的支持不足，无法满足社区养老机构多样化的管理需求。

基于 Python 技术栈的养老管理系统研究相对较少。现有的一些基于 Python 的系统在模块完整性与用户体验上仍有较大的提升空间。例如，部分系统未充分考虑护工的排班管理需求，导致护工工作量分配不均，影响了护工的工作积极性和服务质量。部分系统的用户界面设计不够友好，操作复杂，增加了养老机构工作人员的学习成本，降低了系统的使用效率。因此，结合社区养老的实际需求，开发一款基于 Python、Django 与 Vue.js 的管理系统，具有较好的创新性与应用价值，有望为社区养老管理带来新的突破。

## 研究目标

本研究旨在设计并实现一款功能完善、操作便捷、性能稳定的社区养老管理系统，以解决当前社区养老机构管理中存在的信息关联度与效率低下的问题，实现老人、护工、亲属等多方信息的一体化管理，为养老服务提供有力的数据支持。具体目标如下：

1. 老人信息全生命周期管理。实现老人从入住登记、健康监测到退住管理等各个环节的信息全面管理。在入住登记时，详细记录老人的基本信息、健康状况、生活习惯等；在健康监测方面，与智能医疗设备对接，实时获取老人的健康数据，并进行动态跟踪和分析；退住管理则包括退住原因记录、费用结算等功能，确保老人信息的完整性和准确性。
2. 护工排班管理与绩效统计。支持护工的排班管理，根据养老机构的业务需求和护工的技能水平、工作时间等因素，合理分配工作任务，避免护工工作量分配不均。同时，建立完善的绩效统计体系，对护工的工作质量、工作效率等进行量化评估，为护工的绩效考核和薪酬发放提供依据，提升护工工作效率。
3. 亲属与老人互动平台。提供亲属与老人的互动平台，通过该平台，亲属可以实时了解老人的生活状况、健康数据等信息，还可以与老人进行视频通话、留言交流等，增强亲属与老人之间的情感沟通，让亲属更加放心老人在养老机构的生活。
4. 病史管理功能集成。集成病史管理功能，详细记录老人的病史信息，包括疾病诊断、治疗过程、用药情况等。通过对病史数据的分析和挖掘，为老人的健康管理提供数据支持，帮助医护人员制定更加科学合理的治疗方案和护理计划。
5. 系统性能优化。优化系统性能，确保在高并发场景下的稳定运行。通过采用先进的数据库设计、缓存技术、负载均衡等手段，提高系统的响应速度和吞吐量，减少系统卡顿和故障的发生，为用户提供流畅的使用体验。

为实现上述目标，本研究将梳理社区养老管理的核心业务流程，明确系统的功能需求与非功能需求。通过访谈养老机构管理人员、护工及亲属，深入了解他们对系统的期望与需求，形成详细的需求规格说明书。基于 B/S 架构与前后端分离模式，设计系统的总体架构、功能模块及数据库结构。采用分层设计思想，将系统划分为表现层、业务逻辑层与数据访问层，提高系统的可维护性与扩展性。采用 Python + Django 开发后端接口，Vue.js + Ant - Design - Vue 构建前端页面，实现九大核心模块的功能。通过 RESTful API 实现前后端数据交互，确保系统的松耦合性。最后，通过功能测试、性能测试与安全性测试，验证系统的有效性与稳定性。功能测试覆盖所有核心模块，确保系统功能的正确性；性能测试模拟高并发场景，评估系统的响应时间与吞吐量；安全性测试检查系统的权限控制与数据加密机制，确保系统安全可靠。

# 相关技术介绍

## Python语言

Python 作为一种跨平台、解释型高级编程语言，由 Guido van Rossum 于 1991 年正式发布。其设计哲学将“代码可读性”与“简洁性”置于核心地位，采用了独特的缩进式语法结构。这种结构不仅极大地减少了冗余代码，使代码更加简洁明了，还能让开发人员将更多的精力聚焦于业务逻辑的实现，而非被复杂的语法规则所困扰。

Python 拥有极为丰富的标准库与第三方库资源。标准库涵盖了众多常用功能模块，如文件操作、网络编程、数据处理等，为开发人员提供了便捷的基础支持。第三方库则进一步拓展了 Python 的应用领域，无论是科学计算领域的 NumPy、Pandas，还是机器学习领域的 TensorFlow、PyTorch，都为相关开发提供了强大的工具。同时，Python 支持面向对象、函数式编程等多种编程范式，这使得它能够灵活适应不同场景下的开发需求，广泛应用于 Web 开发、数据分析、人工智能、自动化运维等多个领域。

在本社区养老管理系统中，Python 担当后端开发语言的重要角色。凭借其简洁的语法特性，开发人员能够以更少的代码量实现相同的功能，从而显著缩短了系统的开发周期。以 Python 3.8 版本为例，它提供的类型提示功能，允许开发人员在代码中明确指定变量的类型。这不仅有助于在开发阶段提前发现潜在的错误，减少后期调试的时间和成本，还能提高代码的可读性和可维护性。而异步语法特性则使得系统能够更好地处理高并发请求。在社区养老管理系统中，可能会同时有大量用户访问，如亲属查询老人信息、护工提交工作记录等。异步语法可以让系统在不阻塞其他请求的情况下，高效地处理这些并发请求，从而提升系统的响应速度，确保系统的稳定运行。

## Django框架技术

Django 是一个基于 Python 的开源 Web 应用框架，它遵循“电池包含”（Batteries Included）的设计理念。这意味着 Django 提供了完整的解决方案，涵盖了 Web 应用开发过程中所需的众多功能模块，极大地简化了开发流程，使开发人员能够更加专注于业务逻辑的实现，而无需从头构建各种基础功能。

2.2.1 **Django的MVT架构**

Django 采用 MVT（Model - View - Template）架构，它是 MVC（Model - View - Controller）模式的一种变体。在 MVT 架构中，各组件具有明确的职责分工：

1. 模型（Model）：模型是数据的核心抽象，负责与数据库进行交互。它定义了数据的结构与行为，通过 ORM（对象关系映射）机制，开发人员可以使用 Python 类来映射数据库表，实现数据的 CRUD（创建、读取、更新、删除）操作，而无需编写原生 SQL 语句。例如，在社区养老管理系统中，我们可以定义一个老人模型（Elderly），包含姓名、年龄、性别、健康状况等字段。Django 会根据这个模型自动生成对应的数据库表，开发人员只需通过操作模型类的方法，就可以方便地对老人信息进行增删改查操作，大大简化了数据操作的复杂性。
2. 视图（View）：视图主要负责处理用户请求。当用户发起一个请求时，视图函数会接收这个请求，调用相应的模型获取数据，并将处理后的结果传递给模板。视图在 MVT 架构中相当于 MVC 中的 Controller，起到了业务逻辑处理和数据传递的桥梁作用。例如，当用户请求查看老人列表时，视图函数会从数据库中查询所有老人信息，然后将这些信息传递给模板进行渲染展示。
3. 模板（Template）：模板定义了页面的展示格式。它接收视图传递的数据，并使用 Django 的模板语言（DTL）动态生成 HTML 页面。通过模板，开发人员可以将数据与页面展示分离，实现页面的灵活定制。例如，在模板中可以使用循环语句遍历老人列表，并生成对应的 HTML 表格，将老人的信息展示在页面上。MVT 架构实现了数据、业务逻辑与页面展示的分离，这种分离使得代码具有更好的复用性和可维护性。当需要修改页面样式时，只需修改模板文件，而无需改动视图与模型代码，大大提高了开发效率。

2.2.2 **Django核心功能**

（1）ORM系统：Django 的 ORM 系统是其一大亮点。它通过 Python 类映射数据库表，支持多种主流数据库，如 MySQL、PostgreSQL 等。开发人员可以使用面向对象的方式来操作数据库，无需关心底层 SQL 语句的编写。例如，通过定义 Elderly 模型类，我们可以方便地进行老人信息的增删改查操作。当需要插入一条新的老人记录时，只需创建一个 Elderly 对象并设置相应的属性，然后调用 save 方法即可将数据插入到数据库中。

（2）Admin后台：Django 自动生成的管理界面（Admin 后台）为数据管理提供了极大的便利。它支持数据的快速增删改查操作，开发人员只需通过简单配置，就可以自定义 Admin 后台的功能。例如，通过注册 Elderly 模型到 Admin 后台，我们可以快速实现老人信息的管理界面，包括添加、删除、修改和查询老人信息等功能，无需额外编写大量代码。

（3）中间件：中间件是介于请求与响应之间的钩子函数，它可以用于实现身份验证、日志记录、缓存控制等通用功能。通过编写自定义中间件，开发人员可以在请求处理的各个阶段插入自定义逻辑。例如，通过编写一个日志记录中间件，可以记录每个请求的 IP 地址、请求时间、请求方法等信息，便于后续的日志分析和系统监控。

（4）路由系统：Django 的路由系统通过 URLconf 配置 URL 与视图的映射关系，支持正则表达式匹配。开发人员可以根据业务需求灵活地定义 URL 规则，将不同的 URL 映射到相应的视图函数上。例如，通过配置 URL 路由，我们可以将 /elderly/list/ 映射到展示老人列表的视图函数，将 /elderly/detail/<int:id>/ 映射到展示老人详细信息的视图函数，实现不同 URL 对应不同功能的页面展示。

（5）表单处理：Django 提供了强大的表单处理功能，它支持表单验证与数据处理，简化了用户输入的校验流程。开发人员可以定义表单类，指定表单字段及其验证规则。例如，通过定义 ElderlyForm 表单类，我们可以对用户提交的老人信息进行验证，确保姓名、年龄等字段的合法性。当用户提交表单时，Django 会自动调用表单类的验证方法，对输入数据进行校验，并将验证结果返回给用户，提高了数据的安全性和准确性。

## Vue前端技术

### Vue框架

Vue.js 是一款极具创新性的渐进式 JavaScript 框架，它专注于视图层的开发，具有高度的灵活性和可扩展性，能够与其他库或现有项目实现无缝集成。Vue 的核心特性为开发人员提供了便捷高效的开发体验：

1. 双向数据绑定：Vue 通过 v - model 指令实现了视图与数据的实时同步。当用户在视图中的输入框等元素中输入内容时，视图中的数据会自动更新；反之，当数据发生变化时，视图也会相应地进行更新。这种双向数据绑定机制大大减少了 DOM 操作，提高了开发效率，使开发人员无需手动编写大量的代码来同步视图和数据。
2. 组件化开发：Vue 将页面拆分为独立的组件，每个组件都包含自己的模板、逻辑和样式。这种组件化开发方式提高了代码的复用率和开发效率。例如，在社区养老管理系统中，我们可以将老人信息卡片拆分为一个独立的组件，这个组件可以在多个页面中复用，如老人列表页面、老人详情页面等。当需要修改老人信息卡片的样式或功能时，只需修改该组件的代码，而无需在多个页面中进行重复修改。
3. 虚拟 DOM：Vue 通过内存中的虚拟节点树模拟 DOM 操作。当数据发生变化时，Vue 会先在虚拟 DOM 中进行比较，找出需要更新的部分，然后一次性更新到真实 DOM 中。这种机制减少了浏览器的重绘与回流，提高了页面渲染性能。例如，在一个包含大量数据的表格中，当数据发生变化时，Vue 不会直接操作真实 DOM 中的每一个元素，而是先在虚拟 DOM 中进行计算和比较，然后只更新需要更新的部分，从而大大提高了页面的响应速度。
4. 响应式原理：Vue 利用 Object.defineProperty 监听数据变化。当数据发生变化时，Vue 会自动检测到变化，并通知视图进行更新。这种响应式原理使得视图能够实时反映数据的变化，为用户提供流畅的交互体验。

### Ant-Design-Vue组件库

Ant - Design - Vue 是基于 Ant Design 设计规范的 Vue UI 组件库，它为开发人员提供了丰富的企业级 UI 组件，如表格、表单、弹窗、菜单等，具有以下显著优势：

1. 设计规范统一：Ant - Design - Vue 遵循 Ant Design 的设计语言，确保界面风格一致。所有组件的样式、交互方式都遵循统一的设计规范，无论是按钮、输入框还是表格等组件，都具有相似的外观和操作方式，提升了用户体验，使用户在使用系统时能够感受到一致的交互逻辑。
2. 功能完备：Ant - Design - Vue 的组件覆盖了日常开发中的各种需求，并且支持自定义配置。以表格组件为例，它不仅支持基本的数据展示功能，还具备排序、分页、筛选、列宽调整等高级功能，能够满足复杂的数据展示需求。开发人员可以根据实际业务需求，对组件进行灵活配置，实现个性化的界面效果。
3. 易用性强：Ant - Design - Vue 提供了详细的文档与丰富的示例代码。文档中对每个组件的使用方法、属性、事件等都进行了详细的说明，并配有实际代码示例。开发人员可以通过阅读文档和参考示例，快速掌握组件的使用方法，降低开发难度，提高开发效率。
4. 兼容性好：Ant - Design - Vue 支持现代浏览器与 IE9 及以上版本，能够适配不同设备和浏览器环境。无论用户使用的是桌面电脑、笔记本电脑还是移动设备，无论使用的是 Chrome、Firefox 还是 IE 浏览器，都能够正常访问和使用系统，确保了系统的兼容性和稳定性。在本社区养老管理系统中，我们采用 Ant - Design - Vue 构建前端页面，通过使用其丰富的组件，如表格组件展示老人信息列表、表单组件实现用户输入等功能，大大提升了用户界面的美观度与交互体验。

## Node技术

Node.js 是基于 Chrome V8 引擎的 JavaScript 运行时环境，它打破了传统 JavaScript 只能在浏览器端运行的局限，允许在服务器端运行 JavaScript 代码。Node.js 具有以下核心特性：

1. 非阻塞 I/O：Node.js 采用事件驱动模型，在处理并发请求时无需阻塞等待。当一个请求发起 I/O 操作（如读取文件、访问数据库等）时，Node.js 不会等待操作完成，而是继续处理其他请求。当 I/O 操作完成后，通过事件回调机制通知程序进行处理。这种非阻塞 I/O 模型大大提高了系统的吞吐量，能够同时处理大量的并发请求，适用于高并发的 Web 应用场景。
2. 单线程：Node.js 使用单线程模型，避免了多线程切换带来的开销。在单线程模式下，程序的执行流程更加清晰，简化了编程模型。然而，单线程也意味着需要注意避免长时间同步操作，因为长时间的同步操作会阻塞整个线程，影响系统的性能。在实际开发中，可以通过将耗时操作改为异步操作或使用子进程等方式来解决这个问题。
3. 跨平台：Node.js 具有跨平台特性，能够在 Windows、Linux、Mac OS 等多种操作系统上运行。这使得开发人员可以在不同的平台上进行开发和部署，提高了系统的可移植性和灵活性。
4. 丰富的生态：Node.js 通过 npm（Node Package Manager）管理第三方模块，npm 拥有庞大的模块资源库，涵盖了各种功能的模块。开发人员可以通过 npm 方便地安装和使用这些模块，快速构建功能强大的应用。在本社区养老管理系统中，Node.js 主要用于前端项目的构建与依赖管理。例如，通过 npm 安装 Vue.js 及相关组件库，如 Ant - Design - Vue，为前端开发提供了必要的支持。

## MySQL数据库

MySQL 是一款开源的关系型数据库管理系统，在 Web 应用开发领域得到了广泛应用。它具有以下显著特点：

1. 性能高效：MySQL 采用优化的存储引擎，如 InnoDB。InnoDB 存储引擎支持事务处理，能够保证数据的一致性和完整性。同时，它还支持行级锁，在并发访问时能够减少锁冲突，提高系统的并发性能。在高并发场景下，MySQL 能够快速响应数据读写请求，满足系统的性能需求。
2. 稳定性好：MySQL 经过长期的实践检验，在各种应用场景中都表现出良好的稳定性。它具有完善的错误处理机制和故障恢复能力，能够有效地避免数据丢失和系统崩溃等问题，确保系统的稳定运行。
3. 易用性强：MySQL 提供简洁的 SQL 语法，开发人员可以方便地使用 SQL 语句进行数据的查询、插入、更新和删除等操作。同时，MySQL 还支持多种客户端工具，如 Navicat、MySQL Workbench 等。这些工具提供了图形化界面，使得数据库的管理和维护更加便捷，降低了开发人员的学习成本。
4. 可扩展性：MySQL 支持主从复制、分区表等功能。主从复制可以将一个 MySQL 服务器作为主服务器，将数据复制到一个或多个从服务器上，实现数据的备份和读写分离，提高系统的可用性和性能。分区表则是将一个大表按照一定的规则分成多个小表，提高数据的查询和管理效率。这些功能使得 MySQL 能够满足数据量不断增长的需求，具有良好的可扩展性。在本社区养老管理系统中，我们选用 MySQL 5.7 版本存储数据，通过 Django 的 ORM 机制实现数据的持久化操作，将系统中的各种数据，如老人信息、护工信息、亲属信息等，存储到 MySQL 数据库中，确保数据的安全和可靠。

# 需求分析

## 系统功能需求分析

### 业务需求分析

社区养老管理系统作为养老机构日常运营的关键支撑平台，其核心业务紧密围绕老人的日常照护需求以及机构的运营管理工作展开。该系统精心规划并设计了九大功能模块，以全面满足养老机构在管理、服务与运营等多方面的业务需求，具体业务需求阐述如下：

3.1.1.1 **老人管理业务**

系统需提供完善的信息录入界面，支持工作人员详细录入老人的基本信息，涵盖姓名、性别、年龄、联系方式等基础资料，同时记录入住日期、护理等级等关键信息，为后续的个性化照护服务提供准确依据。

为方便工作人员快速获取老人信息，系统应支持按姓名、身份证号等多种条件进行灵活查询。在信息修改方面，需建立严格的变更记录机制，每次信息修改都应详细记录修改人、修改时间以及修改内容，形成完整的变更日志，以便追溯信息变更过程。

当老人退住时，系统应提供标准化的退住手续办理流程，记录退住原因、退住日期等重要信息，并将老人信息进行归档处理，确保数据的完整性和可追溯性。

3.1.1.2 **护工管理业务**

系统要能够全面登记护工的基本信息，包括姓名、性别、联系方式、入职日期等，同时记录护工的资质证书、擅长领域等关键信息，为合理分配护理任务提供参考。

根据老人的护理需求和护工的工作能力，系统应具备智能排班功能，为护工安排每日的护理任务，明确指定负责的老人以及工作时间段，确保老人得到及时、有效的照护。

系统需实时记录护工的出勤情况，包括出勤时间、请假情况等，同时收集老人及其家属对护工的护理评价，通过科学的算法生成绩效报表，为护工的绩效考核提供客观依据。

3.1.1.3 **亲属管理业务**

系统应支持详细记录老人亲属的姓名、关系、联系方式、住址等重要信息，建立老人与亲属之间的紧密联系，方便在需要时及时与亲属沟通。

为了规范亲属探访流程，系统需提供探访记录功能，详细登记亲属的探访日期、时长、陪同人员等信息，便于养老机构了解老人的社交情况和亲属的探访频率。

通过系统内置的消息推送功能，及时向亲属推送老人的健康状况、活动安排等重要信息，让亲属随时了解老人在养老机构的生活情况，增强亲属与养老机构之间的信任和沟通。

3.1.1.4 **病史管理业务**

系统要提供专门的病史录入界面，支持工作人员详细录入老人的既往病史、过敏史、手术史等重要健康信息，为老人的健康管理和医疗护理提供全面的参考。

建立老人日常健康数据记录机制，实时记录老人的血压、血糖、体温等关键健康指标，同时记录用药情况，包括药品名称、用药剂量、用药时间等，以便医护人员及时掌握老人的健康状况。

系统应具备与医疗机构信息系统对接的能力，关联老人的就诊记录、检查报告等信息，并支持附件上传功能，方便上传病历照片、检查报告等资料，为老人的医疗诊断和治疗提供完整的病历资料。

3.1.1.5 **房间管理业务**

系统要能够全面维护房间的基本信息，包括房间编号、类型、床位数量、设施配置等，为房间的合理分配和管理提供基础数据支持。

根据老人的入住需求和房间的实际情况，系统应提供智能入住分配功能，为老人分配合适的房间与床位，并实时记录床位占用情况，避免床位冲突和资源浪费。

建立房间维修记录和卫生检查记录机制，详细登记房间的维修时间、维修内容、维修人员等信息，同时记录卫生检查结果，确保老人的居住环境安全、整洁、舒适。

3.1.1.6 **活动管理业务**

系统应提供活动策划功能，支持工作人员创建详细的活动计划，包括活动名称、时间、地点、内容、参与人数限制等关键信息，为养老机构组织丰富多彩的老人活动提供便利。

为了方便老人参与活动，系统需提供活动报名功能，记录老人的活动报名情况，同时支持报名与取消操作，及时更新活动参与人数信息。

活动结束后，系统应收集老人对活动的评价与建议，通过数据分析功能，了解老人的需求和满意度，为优化后续活动安排提供参考依据。

3.1.1.7 **用户管理业务**

系统要具备管理员账户管理功能，支持创建、删除管理员账户，并根据不同管理员的职责和工作需求，分配相应的操作权限，如仅允许查看数据或具备修改权限等，确保系统的安全性和数据的保密性。

为方便管理员管理密码，系统应提供密码修改功能，同时支持管理员在忘记密码时通过预留邮箱进行密码重置，保障管理员账户的正常使用。

采用基于角色的访问控制（RBAC）技术，根据管理员的角色和职责，限制不同管理员的操作范围，确保每个管理员只能访问和操作其权限范围内的数据和功能，提高系统的安全性和管理效率。

3.1.1.8 **日志管理业务**

系统应自动记录管理员的关键操作，如添加老人信息、修改订单状态等，详细记录操作人、操作时间、操作内容等关键信息，为系统的安全审计和故障排查提供依据。

为了方便管理员查询日志信息，系统需提供灵活的日志查询功能，支持按操作人、时间、操作类型等条件进行查询，快速定位所需的日志记录。

系统应支持将日志数据导出为 Excel 格式，便于管理员进行存档和进一步的数据分析，为系统的优化和改进提供数据支持。

3.1.1.9 **系统信息管理业务**

系统要提供基础配置功能，支持管理员设置系统名称、Logo、联系方式等基础信息，使系统更符合养老机构的个性化需求。

为确保数据的安全性和可靠性，系统应定期自动备份数据库，同时支持管理员手动触发备份与恢复操作，在数据丢失或损坏时能够及时恢复数据，保障系统的正常运行。

系统需显示当前版本信息，并及时提示管理员版本更新信息，方便管理员及时了解系统的更新情况，进行系统升级和优化，提升系统的性能和功能。

### 数据需求分析

3.1.2.1 **核心数据实体**

系统的核心数据实体是构建整个社区养老管理系统数据模型的基础，它们相互关联、相互作用，共同支撑着系统的各项功能。具体包括：

1. 老人（Elderly）：作为系统的核心服务对象，该数据实体存储老人的基本信息与入住状态等关键数据，为老人的管理和服务提供基础信息支持。
2. 护工（Nurse）：记录护工的个人信息与工作状态，包括基本信息、资质证书、擅长领域以及排班情况等，便于对护工进行合理管理和任务分配。
3. 亲属（Relative）：关联老人与亲属的信息及关系，存储亲属的姓名、关系、联系方式、住址等信息，方便养老机构与亲属进行沟通和联系。
4. 病史（MedicalHistory）：存储老人的健康与医疗相关数据，包括既往病史、过敏史、手术史、日常健康数据、用药情况以及就诊记录等，为老人的健康管理和医疗护理提供全面的数据支持。
5. 房间（Room）：记录房间的基本信息与床位情况，包括房间编号、类型、床位数量、设施配置以及床位占用情况等，为老人的入住分配和房间管理提供基础数据。
6. 活动（Activity）：包含活动的策划与参与信息，如活动名称、时间、地点、内容、参与人数限制以及老人的报名情况等，便于养老机构组织和管理老人活动。
7. 用户（User）：系统管理员的账户与权限信息，包括用户名、密码、角色以及操作权限等，确保系统的安全性和数据的保密性。
8. 日志（Log）：管理员的操作记录，包括操作人、操作时间、操作内容等关键信息，为系统的安全审计和故障排查提供依据。
9. 系统配置（SystemConfig）：系统的基础设置数据，如系统名称、Logo、联系方式、数据备份策略等，使系统更符合养老机构的个性化需求。3.1.2.2 **数据关系与约束**

在社区养老管理系统中，各个核心数据实体之间存在着复杂而紧密的关系，同时为了确保数据的准确性和一致性，需要设置相应的数据约束。

数据关系

老人与亲属：一位老人可能有多位亲属，因此老人与亲属之间存在一对多的关系，通过 elderly\_id 关联，方便查询老人的亲属信息以及亲属与老人的对应关系。

老人与病史：每位老人对应一条病史记录，老人与病史之间是一对一的关系，通过 elderly\_id 关联，确保能够准确获取老人的健康和医疗信息。

老人与房间：多位老人可以入住同一房间，老人与房间之间是多对一的关系，通过 room\_id 关联，便于管理老人的入住情况和房间的床位占用情况。

护工与老人：一位护工可以负责多位老人，同时一位老人也可以由多位护工照护，护工与老人之间是多对多的关系，通过中间表（nurse\_elderly）关联，实现护工与老人之间的灵活分配和照护关系管理。

用户与日志：一位用户可以产生多条操作日志，用户与日志之间是一对多的关系，通过 user\_id 关联，方便追踪用户的操作行为和进行安全审计。

数据约束

主键约束：每个表都应设置主键，主键唯一标识一条记录，如 elderly\_id 为老人表的主键，确保老人信息的唯一性和可识别性。

外键约束：通过外键约束确保关联数据的一致性，例如删除老人信息时，需先删除其关联的亲属与病史记录，避免出现数据不一致的情况。

非空约束：对于必填字段，如老人的姓名、性别等，设置非空约束，不允许为空，确保数据的完整性和准确性。

唯一性约束：为了避免重复数据，对一些关键字段设置唯一性约束，如用户的用户名需唯一，保证系统的正常运行和数据的唯一性。

## 系统非功能性需求分析

### 系统处理能力需求

系统处理能力是衡量系统性能的重要指标之一，它直接影响到系统的使用体验和运营效率。社区养老管理系统需要具备以下处理能力：

（1）支持并发用户数≥50 人，确保在多个用户同时访问系统时，系统能够稳定运行，不会出现卡顿或崩溃的情况。

（2）单条数据查询响应时间≤1 秒，保证用户在查询老人信息、护工信息等数据时能够快速获取结果，提高工作效率。

（3）批量数据导入（如老人信息批量录入）时，处理 1000 条数据的时间≤30 秒，满足养老机构在大量数据录入时的需求，减少数据录入时间。

### 可靠性需求

可靠性是系统稳定运行的关键保障，社区养老管理系统需要具备高度的可靠性，以确保养老机构的日常运营不受影响。具体要求如下：

1. 系统可用性≥99.9%（每年故障时间≤8.76 小时），保证系统在绝大部分时间内能够正常运行，减少因系统故障导致的业务中断。
2. 数据存储可靠性≥99.99%，采用先进的数据存储技术和备份策略，避免数据丢失，确保老人信息、病史数据等重要数据的安全性和完整性。
3. 支持 7×24 小时不间断运行，节假日无间断服务，满足养老机构全年无休的运营需求，为老人提供持续的服务和支持。

### 可用性需求

系统的可用性直接影响到用户的使用体验和操作效率，社区养老管理系统应具备良好的可用性，方便养老机构工作人员使用。具体要求如下：

1. 界面设计简洁直观，操作流程符合养老机构工作人员的使用习惯，减少用户的学习成本和操作难度。
2. 新用户经过≤2 小时培训即可独立操作系统，通过提供详细的操作指南和培训课程，帮助新用户快速掌握系统的使用方法。
3. 提供操作指引与错误提示，在用户操作过程中，及时给予提示和引导，降低使用难度，避免用户因操作不当而出现错误。

### 维修性需求

系统的维修性是指在系统出现故障时，能够快速修复并恢复正常运行的能力。社区养老管理系统需要具备良好的维修性，以减少系统故障对养老机构运营的影响。具体要求如下：

1. 系统故障时，平均修复时间（MTTR）≤2 小时，通过建立快速响应机制和专业的维修团队，确保在系统出现故障时能够及时进行修复。
2. 支持远程诊断与修复，减少现场维护成本，通过远程技术支持，快速定位和解决系统故障，提高维修效率。
3. 提供详细的日志记录，便于故障定位，系统应记录详细的操作日志和错误日志，为维修人员提供准确的故障信息，加快故障排除速度。。

### 环境适配需求

为了确保系统能够在不同的环境下稳定运行，社区养老管理系统需要具备良好的环境适配性。具体要求如下：

1. 后端支持在 Windows 10/11、Mac OS、Linux 等操作系统运行，满足不同养老机构的服务器环境需求。
2. 前端兼容主流浏览器（Chrome、Firefox、Edge、Safari），方便用户在不同的浏览器上访问系统，提高系统的使用便利性。
3. 支持在分辨率≥1366×768 的设备上正常显示，确保系统在各种常见的显示设备上都能够清晰、准确地显示内容，提供良好的用户体验。

4 **系统设计**

4.1 **设计原则**

4.1.1 **阶段开发原则**

本系统采用迭代开发模式，将整个开发过程划分为多个阶段，分阶段逐步实现系统的各项功能。具体而言，第一阶段着重完成系统的核心模块开发，涵盖老人管理、护工管理以及房间管理三大核心功能。通过集中精力构建核心模块，能够快速搭建起系统的基本框架，为后续开发奠定坚实基础。第二阶段在核心模块的基础上，进一步实现扩展模块，包括活动管理、日志管理等。这些扩展模块丰富了系统的功能，使其能够满足社区养老管理更为多样化的需求。第三阶段则聚焦于系统性能的优化与用户体验的提升，通过对系统代码的优化、数据库的调优以及界面交互的改进等措施，确保系统在高并发情况下的稳定运行，同时为用户提供更加流畅、便捷的操作体验。这种分阶段开发的模式有助于合理分配开发资源，控制开发进度，降低开发风险，使整个开发过程处于可控状态。

4.1.2 **易用性原则**

在界面设计方面，严格遵循直观、简洁的原则，力求为用户打造一个易于理解和操作的系统界面。减少不必要的操作步骤是提升易用性的关键举措之一，例如在老人信息查询功能中，用户仅需输入老人的姓名或身份证号，即可快速获取所需信息，避免了繁琐的多条件查询流程。同时，针对关键功能，系统提供快捷入口，如将老人入住登记、护工排班等常用功能放置在显眼位置，方便用户快速访问，从而有效降低用户的学习成本，提高工作效率。此外，界面布局合理，色彩搭配协调，字体大小适中，确保用户在不同设备上都能清晰、舒适地使用系统。4.1.3 **业务完整性原则**

本系统致力于覆盖社区养老管理的全流程，确保各个业务环节紧密衔接，不出现脱节现象。以老人入住流程为例，该流程并非孤立存在，而是与房间分配、护工安排、病史登记等多个环节紧密关联。当老人办理入住手续时，系统会自动根据房间的空闲状态为其分配合适的房间，并安排相应的护工负责老人的日常护理工作。同时，系统会引导用户填写老人的病史信息，确保老人的健康状况得到全面记录。通过这种一体化的设计，实现了业务流程的无缝对接，提高了社区养老管理的效率和质量。

4.1.4 **业务规范化原则**

系统严格遵循养老机构的管理规范与行业标准，确保各项业务操作符合实际需求。在护理等级划分方面，系统依据行业通用的标准，将护理等级细分为多个级别，并为每个级别制定相应的护理服务内容和标准。护工在为老人提供服务时，需严格按照系统设定的护理等级执行，确保服务的规范化和标准化。在健康数据记录格式方面，系统采用统一的格式记录老人的各项健康指标，如血压、血糖、心率等，方便医护人员进行分析和诊断。通过遵循业务规范和行业标准，系统输出的数据具有较高的准确性和可靠性，能够为社区养老管理提供有力的数据支持。

4.1.5 **可扩展性原则**

为适应社区养老管理业务的不断发展和变化，系统采用模块化设计理念，将各个功能模块进行独立封装，便于后续新增功能。例如，随着智能养老设备的发展，未来可能需要集成智能手环的健康数据采集功能。通过模块化设计，只需在系统中新增相应的模块，即可实现与智能手环的对接，无需对整个系统进行大规模修改。同时，在接口设计方面，预留了充足的扩展空间，支持与第三方系统进行对接。例如，与医院的电子病历系统对接，实现老人健康信息的共享，为老人提供更加全面、连续的医疗服务。这种可扩展性设计使得系统能够随着业务的发展不断升级和完善，延长系统的生命周期。

4.2 **总体架构设计**

4.2.1 **B/S 架构设计**

本系统采用 B/S（Browser/Server）架构，用户只需通过浏览器（如 Chrome、Firefox 等）即可访问系统，无需在本地设备上安装客户端软件。这种架构模式具有诸多优势，首先，降低了系统的部署与维护成本。系统管理员只需在服务器端进行软件的安装、配置和更新，用户无需关心本地设备的软件环境，减少了因软件版本不一致、兼容性问题等带来的维护工作量。其次，提高了系统的可访问性。用户可以在任何具备网络连接的设备上通过浏览器访问系统，不受时间和空间的限制，方便了社区养老管理工作人员随时随地处理业务。

B/S 架构分为三层，分别是表现层、业务逻辑层和数据层。表现层是用户与系统进行交互的界面，即浏览器中的 Web 页面。它负责将系统数据以直观、友好的方式展示给用户，并接收用户的操作指令，将用户输入的数据传递给业务逻辑层进行处理。业务逻辑层是系统的核心部分，位于后端服务器。它接收表现层传递过来的请求，根据预设的业务规则和逻辑对请求进行处理，如老人信息验证、权限判断等。在处理过程中，业务逻辑层可能会调用数据层提供的接口，获取或存储相关数据。数据层主要负责数据的存储与管理，采用 MySQL 数据库作为数据存储介质。MySQL 数据库具有稳定性高、性能优良、易于管理等特点，能够满足系统对数据存储和查询的需求。数据层通过定义数据表结构、建立索引等方式，优化数据的存储和查询效率，为业务逻辑层提供高效的数据支持。

4.2.2 **前后端分离架构**

本系统采用前后端分离架构，前端与后端通过 RESTful API 接口进行通信，实现独立开发与部署。前端（Web 层）基于 Vue.js 构建单页应用（SPA），Vue.js 是一款流行的前端框架，具有组件化开发、响应式数据绑定等优点，能够提高前端开发的效率和质量。通过 Axios 调用后端接口获取数据，Axios 是一个基于 Promise 的 HTTP 客户端，能够方便地在浏览器和 Node.js 中发送 HTTP 请求，处理响应数据。前端主要负责用户界面的展示和交互逻辑的实现，通过与后端接口的交互，实现数据的动态加载和更新。

后端（Server 层）基于 Django 开发 API 接口，Django 是一个功能强大的 Python Web 框架，提供了丰富的组件和工具，能够快速开发高质量的 Web 应用。后端负责处理前端发送的请求，根据请求的类型和参数，调用相应的业务逻辑进行处理，并将处理结果以 JSON 格式返回给前端。JSON 格式具有简洁、易解析的特点，便于前端进行处理和展示。

前后端分离架构具有显著的优势。一方面，前后端开发团队可以并行工作，前端开发人员专注于用户界面的设计和交互逻辑的实现，后端开发人员专注于业务逻辑的处理和数据的存储与管理，提高了开发效率，缩短了项目开发周期。另一方面，便于后期单独优化前端或后端性能。前端可以根据用户体验需求进行优化，如提高页面加载速度、优化交互效果等；后端可以通过优化数据库查询、引入缓存机制等方式提高系统的响应速度和处理能力。

4.2.3 **系统分层设计**

后端采用分层设计，将系统功能划分为视图层（Views）、业务逻辑层（Services）、数据访问层（Models）和公共层（Common），各层职责明确，相互协作，共同完成系统的业务功能。

视图层（Views）是系统的入口，负责接收前端发送的请求，对请求进行初步解析和验证，然后调用业务逻辑层提供的接口进行处理。在处理完成后，将业务逻辑层返回的结果进行封装，返回给前端。视图层主要关注请求的接收和响应的返回，不涉及具体的业务逻辑处理，起到了前后端交互的桥梁作用。

业务逻辑层（Services）是系统的核心业务处理层，实现了社区养老管理的各项核心业务逻辑，如老人入住流程、护工排班算法等。在处理业务逻辑时，业务逻辑层可能会协调多个模型的操作，调用数据访问层提供的接口获取或存储相关数据。业务逻辑层的设计需要考虑业务规则的复杂性和灵活性，确保业务逻辑的正确性和可扩展性。

数据访问层（Models）通过 Django ORM 与数据库进行交互，提供了数据的 CRUD（创建、读取、更新、删除）操作。Django ORM 是一种对象关系映射工具，它将数据库中的表映射为 Python 类，将表中的记录映射为类的实例，通过操作类实例来实现对数据库的操作。数据访问层屏蔽了数据库的具体操作细节，使得业务逻辑层无需关心数据库的类型和操作方式，提高了代码的可维护性和可移植性。

公共层（Common）包含工具类、常量定义、异常处理等通用组件。工具类提供了一些常用的功能函数，如日期格式转换、字符串处理等，方便其他层调用。常量定义将系统中使用的常量进行统一管理，便于修改和维护。异常处理机制对系统运行过程中可能出现的异常情况进行捕获和处理，保证系统的稳定性和可靠性。公共层的设计提高了代码的复用性，减少了代码冗余。

4.3 **功能设计**

本系统共包含九大功能模块，每个模块都具有基础功能和扩展功能，以满足社区养老管理的不同需求。以下是对各功能模块的详细介绍：

4.3.1 **老人管理模块**

老人管理模块是系统的核心模块之一，主要负责老人信息的管理。基础功能包括老人信息的增删改查，用户可以通过系统界面方便地添加新老人的信息、修改已有老人的信息、删除不再需要管理的老人信息以及查询特定老人的信息。扩展功能包括入住登记、退住管理和信息导出（Excel 格式）。入住登记功能实现了老人入住流程的信息化管理，记录老人的入住时间、房间分配、护工安排等信息。退住管理功能记录老人的退住时间和原因，更新老人的状态信息。信息导出功能方便用户将老人信息以 Excel 格式导出，便于进行数据分析和备份。

4.3.2 **护工管理模块**

护工管理模块用于管理护工的相关信息。基础功能包括护工信息的增删改查，与老人管理模块类似，用户可以对护工信息进行全面的管理。扩展功能包括排班表生成和绩效统计图表展示。排班表生成功能根据护工的工作时间和任务要求，自动生成合理的排班表，提高排班效率。绩效统计图表展示功能通过图表的形式直观地展示护工的工作绩效，如工作时长、服务老人数量等，便于管理人员对护工进行考核和评价。

4.3.3 **亲属管理模块**

亲属管理模块主要负责管理老人亲属的信息。基础功能包括亲属信息的增删改查，用户可以添加老人的亲属信息，如姓名、联系方式、与老人的关系等，并对这些信息进行修改、删除和查询操作。扩展功能包括探访记录登记和消息发送记录查询。探访记录登记功能记录亲属探访老人的时间、探访事由等信息，方便管理人员了解老人的社交情况。消息发送记录查询功能记录系统向亲属发送的消息内容、发送时间等信息，便于管理人员跟踪消息的发送情况。

4.3.4 **病史管理模块**

病史管理模块用于管理老人的病史信息。基础功能包括病史信息的增删改查，用户可以添加老人的病史记录，如疾病诊断、治疗方案、用药情况等，并对这些信息进行修改、删除和查询操作。扩展功能包括健康数据趋势分析（如近 30 天血压变化曲线）和医疗附件上传/下载。健康数据趋势分析功能通过对老人健康数据的统计分析，生成直观的曲线图表，帮助医护人员了解老人的健康状况变化趋势。医疗附件上传/下载功能允许用户上传老人的医疗检查报告、病历等附件，方便医护人员查阅和参考。

4.3.5 **房间管理模块**

房间管理模块主要负责管理社区养老机构的房间信息。基础功能包括房间信息的增删改查，用户可以添加新房间的信息，如房间号、房间类型、床位数等，并对这些信息进行修改、删除和查询操作。扩展功能包括床位状态可视化（空闲/占用）和维护记录查询。床位状态可视化功能通过图形化的方式展示每个房间的床位使用情况，方便管理人员快速了解房间的空闲状态。维护记录查询功能记录房间的维护时间、维护内容等信息，便于管理人员对房间进行维护管理。

4.3.6 **活动管理模块**

活动管理模块用于管理社区养老机构组织的各类活动。基础功能包括活动信息的增删改查，用户可以添加新活动的信息，如活动名称、活动时间、活动地点等，并对这些信息进行修改、删除和查询操作。扩展功能包括报名统计和活动评价分析。报名统计功能统计参加活动的老人人数和名单，方便管理人员安排活动资源。活动评价分析功能收集老人对活动的评价意见，通过统计分析了解老人对活动的满意度，为后续活动的改进提供参考。

4.3.7 **用户管理模块**

用户管理模块主要负责管理系统的用户信息。基础功能包括用户信息的增删改查，用户可以添加新用户的信息，如用户名、密码、角色等，并对这些信息进行修改、删除和查询操作。扩展功能包括角色分配和权限设置。角色分配功能根据用户的工作职责和权限需求，为用户分配不同的角色，如管理员、操作员等。权限设置功能为每个角色设置相应的操作权限，确保用户只能访问和操作其权限范围内的功能和数据，保障系统的安全性。

4.3.8 **日志管理模块**

日志管理模块用于记录系统的操作日志。基础功能包括日志查询和筛选，用户可以根据时间、操作类型、操作人员等条件对日志进行查询和筛选，方便查找特定的操作记录。扩展功能包括日志导出和操作行为分析。日志导出功能将日志信息以文件形式导出，便于进行数据备份和长期存储。操作行为分析功能通过对日志数据的统计分析，了解用户的操作习惯和系统的使用情况，为系统的优化和改进提供依据。

4.3.9 **系统信息模块**

系统信息模块主要负责管理系统的配置信息和版本信息。基础功能包括系统配置修改，用户可以对系统的各项参数进行修改和调整，如数据库连接参数、系统通知设置等。扩展功能包括数据库备份/恢复和版本信息查询。数据库备份/恢复功能定期对系统数据库进行备份，并在需要时进行恢复操作，确保数据的安全性和完整性。版本信息查询功能显示系统的版本号、更新时间等信息，方便用户了解系统的版本情况。

4.4 **数据库设计**

4.4.1 **设计规则**

数据库设计遵循第三范式（3NF），第三范式要求每个非主键字段必须直接依赖于主键，而不能依赖于其他非主键字段，通过遵循第三范式可以确保每个字段只依赖于主键，减少数据冗余，提高数据的存储效率和一致性。例如，在老人信息表中，老人的姓名、年龄、身份证号等字段都直接依赖于老人的唯一标识 ID，而不依赖于其他非主键字段。

表名和字段名采用小写字母加下划线的命名方式，如 user\_info、create\_time 等。这种命名方式具有较高的可读性，便于开发人员理解和维护数据库结构。同时，统一的命名规范也有助于提高代码的一致性和可维护性。

对关键数据（如用户密码）进行加密存储，确保数据安全。在系统中，用户密码是敏感信息，如果以明文形式存储，一旦数据库被泄露，用户的账号安全将受到严重威胁。因此，采用加密算法对用户密码进行加密处理后再存储到数据库中，即使数据库被攻击，攻击者也无法直接获取用户的原始密码。

为常用查询字段（如老人的姓名、身份证号）建立索引，提升查询效率。索引是一种数据结构，它类似于书籍的目录，能够快速定位到数据库中的特定记录。当对常用查询字段建立索引后，数据库在执行查询操作时可以更快地找到所需数据，减少查询时间，提高系统的响应速度。

4.4.2 **表结构设计**

系统的核心数据表结构如下：

老人信息表（elderly）

| 字段名 | 数据类型 | 约束 | 说明 |

|--------|----------|------|------|

| id | int | 主键、自增 | 唯一标识 |

| name | varchar (50) | 非空 | 老人姓名 |

| gender | tinyint | 非空 | 性别（0 - 女，1 - 男） |

| age | int | 非空 | 年龄 |

| id\_card | varchar (18) | 唯一、非空 | 身份证号 |

| phone | varchar (20) | | 联系电话 |

| check\_in\_date | date | | 入住日期 |

| check\_out\_date | date | | 退住日期 |

| room\_id | int | 外键 | 关联房间表 |

| care\_level | varchar (20) | | 护理等级 |

| create\_time | datetime | 非空 | 创建时间 |

| update\_time | datetime | 非空 | 更新时间 |

护工信息表（nurse）

| 字段名 | 数据类型 | 约束 | 说明 |

|--------|----------|------|------|

| id | int | 主键、自增 | 唯一标识 |

| name | varchar (50) | 非空 | 护工姓名 |

| gender | tinyint | 非空 | 性别（0 - 女，1 - 男） |

| phone | varchar (20) | 非空 | 联系电话 |

| hire\_date | date | 非空 | 入职日期 |

| position | varchar (50) | | 职位 |

| status | tinyint | 非空 | 状态（0 - 离职，1 - 在职） |

| create\_time | datetime | 非空 | 创建时间 |

| update\_time | datetime | 非空 | 更新时间 |

房间信息表（room）

| 字段名 | 数据类型 | 约束 | 说明 |

|--------|----------|------|------|

| id | int | 主键、自增 | 唯一标识 |

| room\_no | varchar (20) | 唯一、非空 | 房间号 |

| type | varchar (30) | | 房间类型（如单人间、双人间） |

| bed\_count | int | 非空 | 总床位数 |

| available\_bed | int | 非空 | 剩余床位数 |

| status | tinyint | 非空 | 状态（0 - 停用，1 - 在用） |

| create\_time | datetime | 非空 | 创建时间 |

| update\_time | datetime | 非空 | 更新时间 |

用户表（user）

| 字段名 | 数据类型 | 约束 | 说明 |

|--------|----------|------|------|

| id | int | 主键、自增 | 唯一标识 |

| username | varchar (50) | 唯一、非空 | 用户名 |

| password | varchar (128) | 非空 | 加密后的密码 |

| role | varchar (20) | 非空 | 角色（如 admin、operator） |

| is\_active | tinyint | 非空 | 是否激活（0 - 否，1 - 是） |

| last\_login | datetime | | 最后登录时间 |

| create\_time | datetime | 非空 | 创建时间 |

其他表（如亲属表、病史表、活动表、日志表等）的结构设计参照上述格式，根据需求定义字段，确保数据的完整性和关联性。

5 **系统实现**

5.1 **开发环境搭建**

5.1.1 **后端环境配置**

Python环境安装

首先需安装Python 3.8运行环境。从Python官方网站下载对应操作系统的安装包，按照默认配置完成安装后，需将Python安装路径添加至系统环境变量中，以确保在命令行中可直接调用Python解释器。

Django框架及依赖库安装

进入项目server目录，通过requirements.txt文件批量安装后端依赖库：

bash

cd server

pip install -r requirements.txt

requirements.txt文件包含以下核心依赖项：

Django 3.2（Web框架核心）

djangorestframework（REST API支持）

pymysql（MySQL数据库驱动）

其他辅助库（如django-cors-headers等）

MySQL数据库配置

安装MySQL 5.7数据库服务器，建议采用默认配置安装

创建专用数据库：

sql

CREATE DATABASE IF NOT EXISTS python\_yanglao

DEFAULT CHARACTER SET utf8

COLLATE utf8\_general\_ci;

在Django的settings.py文件中配置数据库连接参数：

python

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'python\_yanglao',

'USER': 'root',

'PASSWORD': '123456',

'HOST': 'localhost',

'PORT': '3306',

}

}

数据库初始化

执行以下命令完成数据库表结构创建及初始化：

bash

# 生成迁移文件

python manage.py makemigrations

# 执行数据表创建

python manage.py migrate

# 创建系统管理员账户

python manage.py createsuperuser

服务启动

启动Django开发服务器，监听所有网络接口的8000端口：

bash

python manage.py runserver 0.0.0.0:8000

服务启动后，可通过http://localhost:8000访问API接口，通过http://localhost:8000/admin访问管理后台。

5.1.2 **前端环境配置**

Node.js环境安装

从Node.js官方网站下载16.14 LTS版本进行安装，该版本已包含npm包管理器。安装完成后，可通过node -v和npm -v命令验证安装是否成功。

前端依赖安装

进入web目录后执行依赖安装：

bash

cd web

npm install

如遇网络问题，可使用淘宝镜像源加速下载：

bash

npm install --registry=https://registry.npm.taobao.org

接口地址配置

修改前端全局配置文件web/src/constants.js，设置后端API基础地址：

javascript

export const baseUrl = 'http://localhost:8000/api';

前端服务启动

执行开发模式启动命令：

bash

npm run dev

服务默认启动在8080端口，可通过http://localhost:8080访问前端应用界面。

5.2 **核心功能模块实现**

5.2.1 **老人管理模块**

后端实现

采用Django REST framework的ModelViewSet实现标准CRUD接口：

python

# server/elderly/views.py

from rest\_framework import viewsets

from .models import Elderly

from .serializers import ElderlySerializer

class ElderlyViewSet(viewsets.ModelViewSet):

"""老人信息管理视图集"""

queryset = Elderly.objects.all().order\_by('-create\_time')

serializer\_class = ElderlySerializer

def perform\_create(self, serializer):

"""重写创建方法，自动填充时间戳"""

serializer.save(

create\_time=timezone.now(),

update\_time=timezone.now()

)

前端实现

基于Vue.js和Ant Design Vue组件库构建管理界面：

vue

<!-- web/src/views/elderly/List.vue -->

<template>

<div class="elderly-container">

<a-card title="老人信息管理">

<a-button type="primary" @click="showAddModal">

<PlusOutlined /> 新增老人

</a-button>

<a-table

:columns="columns"

:data-source="dataSource"

:loading="loading"

rowKey="id"

class="mt-4"

>

<template #operation="{ record }">

<a-button type="link" @click="handleEdit(record)">编辑</a-button>

<a-popconfirm

title="确认删除该记录？"

@confirm="handleDelete(record.id)"

>

<a-button type="link" danger>删除</a-button>

</a-popconfirm>

</template>

</a-table>

</a-card>

<!-- 新增/编辑模态框 -->

<ElderlyModal

:visible="modalVisible"

:formData="currentRecord"

@cancel="modalVisible = false"

@submit="handleSubmit"

/>

</div>

</template>

<script>

import { getElderlyList, addElderly, updateElderly, deleteElderly } from '@/api/elderly';

import ElderlyModal from './components/ElderlyModal';

export default {

components: { ElderlyModal },

data() {

return {

dataSource: [],

loading: false,

modalVisible: false,

currentRecord: {},

columns: [

{ title: '姓名', dataIndex: 'name' },

{ title: '性别', dataIndex: 'gender' },

{ title: '年龄', dataIndex: 'age' },

{ title: '操作', slots: { customRender: 'operation' } }

]

};

},

created() {

this.fetchData();

},

methods: {

async fetchData() {

this.loading = true;

try {

const res = await getElderlyList();

this.dataSource = res.data;

} finally {

this.loading = false;

}

},

showAddModal() {

this.currentRecord = {};

this.modalVisible = true;

},

handleEdit(record) {

this.currentRecord = { ...record };

this.modalVisible = true;

},

async handleSubmit(form) {

if (form.id) {

await updateElderly(form.id, form);

} else {

await addElderly(form);

}

this.modalVisible = false;

this.fetchData();

},

async handleDelete(id) {

await deleteElderly(id);

this.$message.success('删除成功');

this.fetchData();

}

}

};

</script>

5.2.2 **护工管理模块**

护工管理模块在基础CRUD功能基础上，重点实现了排班管理功能。后端通过自定义API接口返回护工排班数据，前端采用日历组件进行可视化展示，支持按周/月视图切换及排班信息编辑。

5.2.3 **亲属管理模块**

亲属管理实现了与老人信息的关联关系。在亲属信息录入界面，通过级联选择器绑定老人信息，并设置亲属关系类型字段（如父子、配偶等）。后端通过ForeignKey字段建立数据关联，前端通过异步请求获取老人列表数据。

5.2.4 **病史管理模块**

病史管理支持医疗附件的上传与存储。后端采用Django的FileField字段存储文件路径，前端使用Ant Design Vue的Upload组件实现文件选择：

vue

<a-upload

name="file"

:action="`${baseUrl}/medical/upload/`"

:headers="{ 'Authorization': `Bearer ${token}` }"

@change="handleUploadChange"

>

<a-button>

<UploadOutlined /> 点击上传医疗附件

</a-button>

</a-upload>

5.2.5 **房间管理模块**

房间管理采用卡片式布局展示房间状态，通过不同颜色标识在用/停用状态。点击房间卡片可展开查看该房间的床位使用情况，包括总床位数、已入住人数及空闲床位信息。

5.2.6 **活动管理模块**

活动管理实现了活动报名功能，前端通过进度条组件直观展示活动报名情况：

vue

<a-progress

:percent="Math.round((record.joined\_count / record.total\_quota) \* 100)"

:status="record.joined\_count >= record.total\_quota ? 'exception' : 'active'"

/>

5.2.7 **用户管理模块**

基于Django内置的权限系统实现多角色控制。通过Group模型定义不同角色（如管理员、普通操作员），在前端路由配置中添加权限验证中间件，实现菜单级权限控制。

5.2.8 **日志管理模块**

采用Django中间件实现操作日志自动记录：

python

# server/common/middleware.py

import logging

from django.utils import timezone

from system.models import OperationLog

logger = logging.getLogger(\_\_name\_\_)

class OperationLogMiddleware:

def \_\_init\_\_(self, get\_response):

self.get\_response = get\_response

def \_\_call\_\_(self, request):

# 记录请求开始时间

start\_time = timezone.now()

# 处理请求

response = self.get\_response(request)

# 记录操作日志

if hasattr(request, 'user') and request.user.is\_authenticated:

try:

OperationLog.objects.create(

user=request.user,

method=request.method,

path=request.path,

status\_code=response.status\_code,

ip\_address=self.get\_client\_ip(request),

execution\_time=(timezone.now() - start\_time).total\_seconds(),

user\_agent=request.META.get('HTTP\_USER\_AGENT', '')[:200]

)

except Exception as e:

logger.error(f"日志记录失败: {str(e)}")

return response

def get\_client\_ip(self, request):

x\_forwarded\_for = request.META.get('HTTP\_X\_FORWARDED\_FOR')

if x\_forwarded\_for:

ip = x\_forwarded\_for.split(',')[0]

else:

ip = request.META.get('REMOTE\_ADDR')

return ip

5.2.9 **系统信息模块**

数据库备份功能通过调用系统命令实现：

python

# server/system/utils.py

import subprocess

from datetime import datetime

from django.conf import settings

def backup\_database():

"""执行数据库备份"""

timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S')

backup\_file = f"{settings.BACKUP\_DIR}/yanglao\_backup\_{timestamp}.sql"

try:

command = [

'mysqldump',

'-u', settings.DATABASES['default']['USER'],

f'-p{settings.DATABASES["default"]["PASSWORD"]}',

settings.DATABASES['default']['NAME'],

'>', backup\_file

]

# 注意：实际执行时需要处理shell重定向问题

# 更安全的实现方式：

with open(backup\_file, 'w') as f:

subprocess.run(

['mysqldump',

'-u', settings.DATABASES['default']['USER'],

f'-p{settings.DATABASES["default"]["PASSWORD"]}',

settings.DATABASES['default']['NAME']],

stdout=f,

check=True

)

return {

'success': True,

'message': f'数据库备份成功: {backup\_file}',

'file\_path': backup\_file

}

except subprocess.CalledProcessError as e:

return {

'success': False,

'message': f'数据库备份失败: {str(e)}'

}

6 **系统测试**

6.1 **测试概述**

6.1.1 **测试目的与范围**

在软件开发过程中，系统测试是确保软件质量的关键环节。本次系统测试的主要目的在于全面验证系统是否切实满足需求分析阶段所明确界定的功能与非功能需求。通过严谨细致的测试流程，发现并精准修复系统中潜藏的问题，从而为系统上线后的稳定运行提供坚实保障。

测试范围广泛覆盖了系统的九大功能模块，不仅着重考察各功能模块的正确性，确保其能够按照设计要求准确无误地执行相应操作，还对系统的性能指标进行严格评估，以验证系统在高负载情况下的响应速度和处理能力。同时，安全性测试旨在排查系统可能存在的安全漏洞，防止数据泄露、非法访问等安全问题的发生。易用性测试则关注用户与系统交互的便捷性和舒适性，力求为用户提供流畅、友好的操作体验。

6.1.2 **测试环境**

为确保测试结果的准确性和可靠性，搭建了与实际运行环境高度相似的测试环境，具体配置如下：

硬件环境：采用配备 Intel Core i5 处理器的计算机，该处理器具备较高的运算能力和多任务处理性能，能够满足系统运行的基本需求。配备 8GB 内存，为系统的流畅运行提供了充足的内存空间，有效避免了因内存不足导致的系统卡顿问题。500GB 硬盘则为系统数据的存储提供了足够的容量，确保测试过程中数据的完整性和安全性。

软件环境：选用广泛应用的 Windows 10 操作系统，其具有良好的兼容性和稳定性，能够为系统的运行提供稳定的底层支持。使用 Chrome 90.0 浏览器，该浏览器以其快速的页面加载速度、强大的兼容性和丰富的扩展功能而受到广泛青睐，能够准确模拟用户在实际使用过程中的浏览器环境。采用 MySQL 5.7 数据库管理系统，它是一款开源的关系型数据库，具有高性能、高可靠性和易于管理的特点，能够满足系统对数据存储和管理的需求。基于 Django 3.2 框架构建后端服务，Django 是一个高级 Python Web 框架，具有快速开发和简洁设计的特点，能够高效地处理业务逻辑和数据交互。该后端服务运行在 Python 3.8 环境下，Python 3.8 提供了丰富的标准库和第三方库支持，为后端服务的开发提供了便利。采用 Vue 2.6 框架搭建前端界面，Vue 是一款渐进式 JavaScript 框架，具有响应式数据绑定和组件化开发的特点，能够构建出高性能、易维护的用户界面。前端服务运行在 Node.js 16.14 环境下，Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行时，能够为前端开发提供强大的服务器端支持。

6.2 **测试用例设计与执行**

6.2.1 **功能测试**

功能测试是系统测试的核心内容，旨在验证系统各个功能模块是否能够按照需求规格说明书的要求正常工作。以下以老人管理模块为例，详细介绍测试用例的设计与执行情况。

用例 ID 测试场景 输入数据 预期结果 实际结果 测试状态

TC-E01 新增老人 姓名：张三，性别：男，年龄：75 老人信息成功录入系统，并在数据库中新增一条与之对应的记录 符合预期 通过

TC-E02 编辑老人 将张三的年龄修改为 76 数据库中张三记录的年龄字段更新为 76 符合预期 通过

TC-E03 删除老人 张三的 ID 数据库中张三的记录被成功删除 符合预期 通过

TC-E04 查询老人 姓名：张三 系统列表中准确显示张三的相关信息 符合预期 通过

在老人管理模块的功能测试中，通过手动操作的方式，按照预设的测试用例逐步执行各项操作，并仔细核对实际结果与预期结果是否一致。其他模块的测试用例设计思路与老人管理模块类似，均围绕各模块的核心功能展开，通过手动操作验证功能的正确性。经过全面细致的测试，结果显示所有核心功能均能够正常工作，为系统的稳定运行奠定了坚实基础。

6.2.2 **性能测试**

性能测试旨在评估系统在高并发情况下的响应速度和处理能力，以确保系统能够满足实际业务场景的需求。本次性能测试使用 Apache JMeter 这一专业的性能测试工具，模拟 50 个并发用户同时访问系统的老人列表查询接口，对系统的性能指标进行全面监测和分析。测试结果如下：

系统的平均响应时间为 0.5 秒，远低于需求规格说明书中规定的 1 秒上限，表明系统在处理用户请求时具有较高的响应速度，能够为用户提供流畅的操作体验。

系统的吞吐量达到 20 请求/秒，这意味着系统在单位时间内能够处理较多的用户请求，具备较高的业务处理能力，能够满足一定规模的用户并发访问需求。

错误率：在测试过程中，系统的错误率为 0%，说明系统在高并发情况下能够稳定运行，未出现因性能问题导致的请求失败或错误情况，具有较高的可靠性和稳定性。

6.2.3 **安全性测试**

安全性是系统运行的重要保障，本次安全性测试从多个方面对系统进行了全面评估，以确保系统能够抵御各种安全威胁。

尝试使用弱密码（如 123456）进行用户注册，系统能够及时检测到密码强度不足，并给出相应的提示信息，阻止用户使用弱密码注册，有效提高了用户账户的安全性。

使用普通操作员账号登录系统，尝试访问需要管理员权限的 “用户管理” 模块，系统能够准确识别用户权限，拒绝普通操作员的访问请求，确保了系统资源的安全性和数据的保密性。

在老人姓名查询框中输入 “' or '1'='1” 这一典型的 SQL 注入攻击语句，系统能够正确处理该输入，未执行恶意 SQL 语句，返回正常结果，表明系统具备有效的 SQL 注入防护机制，能够有效防止黑客通过 SQL 注入攻击获取系统数据或破坏系统数据库。

6.3 **测试结果分析**

经过全面细致的功能测试，共有 95% 的测试用例顺利通过。未通过的测试用例主要集中在 “活动报名人数超限” 场景，经过深入分析，发现是由于系统在处理活动报名人数统计时存在逻辑漏洞，导致在报名人数达到上限后仍允许用户继续报名。针对这一问题，开发团队及时进行了修复，并对修复后的系统进行了重新测试，确保问题得到彻底解决。总体而言，系统的核心功能均能够正常工作，功能完整性得到了有效保障。

性能测试结果表明，系统在模拟 50 并发用户的情况下，各项性能指标均表现出色。平均响应时间、吞吐量和错误率等关键指标均满足需求规格说明书的要求，说明系统具备较高的性能和稳定性，能够在实际业务场景中为用户提供高效、稳定的服务。

安全性测试未发现系统存在重大安全漏洞，系统在密码安全、权限控制和 SQL 注入防护等方面均表现出良好的安全性，基本满足了安全性需求。然而，随着网络安全威胁的不断变化和升级，系统仍需持续关注安全动态，定期进行安全评估和更新，以应对可能出现的新安全挑战。

7 **总结与展望**

在老龄化社会日益加剧的大背景下，社区养老管理的重要性愈发突出。为提升社区养老管理的效率与质量，本文设计并实现了一款基于 Python 的社区养老管理系统，在项目推进中完成了一系列系统且具逻辑性的工作。

项目初期，通过深入调研社区养老管理的实际业务场景，与养老机构工作人员、老人及其家属充分沟通，明确了涵盖老人信息管理、健康档案管理、活动管理等九大核心模块，同时细致梳理了系统的性能、安全性、易用性等非功能需求，为后续系统设计与实现指明方向。系统设计上，采用 B/S 架构与前后端分离模式，B/S 架构让用户通过浏览器即可访问，提高可访问性与便捷性；前后端分离使开发人员可并行工作，提升开发效率，便于维护扩展。功能模块设计依据九大核心模块详细划分功能、定义接口，数据库设计遵循规范化原则，保证数据一致性与完整性，为系统稳定运行筑牢数据基础。

系统实现阶段，充分利用 Python 生态系统资源，用 Django 框架开发后端 API，其强大功能和完善安全机制可快速搭建稳定后端服务，通过定义清晰路由和视图函数实现业务逻辑处理并与数据库高效交互。前端页面采用 Vue.js 构建，其组件化开发模式使开发更模块化、可维护，响应式数据绑定特性为用户带来流畅操作体验。开发过程中严格遵循编码规范，进行代码审查和单元测试，确保代码质量与稳定性，最终成功实现所有核心功能并通过初步功能验证。为保证系统质量与可靠性，开展全面系统测试，包括功能测试、性能测试和安全性测试。功能测试设计详细用例逐一验证模块功能；性能测试模拟不同规模用户并发访问，评估系统高负载性能；安全性测试从多方面评估系统安全性，检测漏洞。通过测试验证系统有效性，修复潜在问题，为正式上线提供有力保障。本系统功能模块完整，覆盖社区养老管理全流程，实现信息化智能化管理；操作界面简洁直观，降低使用门槛，提高工作效率；技术架构先进，便于后期维护扩展，能适应业务变化发展。

尽管本系统充分考虑了实际需求，但仍存在不足。目前系统仅支持简单统计图表展示，缺乏智能分析功能，无法基于老人历史健康数据构建预测模型进行健康风险预警，难以满足养老机构对老人健康管理的精细化需求。在大数据时代，数据价值巨大，如何挖掘利用数据为养老服务提供决策支持是改进方向。此外，当前版本主要针对 PC 端设计开发，未开发移动端应用。随着移动互联网普及，护工和家属倾向用手机管理工作生活，缺乏移动端应用影响养老服务便捷性和及时性。针对这些问题，未来将深入研究数据挖掘算法和模型，结合业务需求构建预测模型，实时监测分析老人健康指标，预测健康问题并及时预警，同时分析活动数据提供个性化活动安排。还将开发适用于 iOS 和 Android 平台的移动端应用，具备与 PC 端相似功能，利用手机硬件设备实现便捷数据采集交互，如护工扫描药品条码记录用药信息。此外，积极对接智能健康设备，实现老人健康数据自动采集传输，为护工和医生提供准确数据支持，减少人工录入工作量，提高数据准确性和及时性，进一步提升社区养老管理智能化水平。

通过持续优化改进，本社区养老管理系统将更好地满足智能化需求，为提升养老服务质量提供更强有力支持，助力构建和谐美好的老年生活环境。

参考文献

[1]刘华清.智慧社区养老服务管理系统的设计与实现[D].山东大学,2018.

[2]马旭.人工智能赋能“软件开发技术与应用”课程的革新与拓展：以实践驱动探索和创新[J].黑龙江教育(理论与实践),2025,(07):62-65.

[3]古凌岚,张婵,罗佳. 计算机系统化项目开发教程[M].人民邮电出版社:, 201802.313.

[4]肖睿,喻晓路,朱微,张超,戴立坤. Web应用设计及实战[M].人民邮电出版社:大数据开发实战系列, 201801.238.

[5]林聪,王龙洋,颜晨阳.基于Vue.js与Django的虚拟化管理平台设计[J].科技创新与应用,2025,15(15):43-46.

[6]陆惠恩,褚秋砚. 软件工程[M].人民邮电出版社:, 201712.284.

[7]陈晓东,付延杰.社区需求理论视角下农村社区养老模式探索研究——以S省D社区为例[J].中国市场,2025,(21):9-12.

[8]张冰心.Vue.js前端框架课程中的教师能动性探析[J].现代商贸工业,2024,(24):248-250.

[9]陈陆扬. Vue.js前端开发快速入门与专业应用[M].人民邮电出版社, 201702.207.

[10]耿海舰.大数据时代事业单位档案管理信息化建设的思考与策略[J].兰台内外,2025,(22):22-25.