# 1、系统概述

**1.1、系统简介**

简要概述系统的基本情况和背景。

本系统是一个小巧轻量、集发布和讨论于一体的论坛平台；基于北邮人论坛的使用和周围同学的反馈背景，让用户体验更加简约流畅。

**1.2、术语表**

定义系统或产品中涉及的重要术语，为读者在阅读文档时提供必要的参考信息。

|  |  |  |
| --- | --- | --- |
| **序号** | **术语或缩略语** | **说明性定义** |
| 1 | PM | Project Manager,项目经理 |
| 2 |  |  |
|  |  |  |

**1.3、系统运行环境**

包括对硬件平台、操作系统、数据库系统、编程平台、网络协议等的描述。

硬件平台：Legion Y7000P 2019、Dell G15 5511、MacBook Air(M1)

操作系统：Windows、Mac OS、Linux

前端框架：Vue.js + Vue Route + Naïve UI + Element Plus + Less + Axios

后端框架：FastAPI+SQLAlchemy

数据库：Sqlite

编程平台：VSCode

网络协议：TCP/IP

**1.4、开发环境**

列举进行系统分析、程序设计和程序开发时要使用的工程工具和开发语言。应描述每一工具软件的名称、版本等。

工程工具：

集成开发环境：VSCode、PyCharm

版本控制工具：Git

自动化构建工具：Vite

调试工具：Volar、Unicorn、Chrome浏览器

截止到项目发布时的最新版

开发语言：

Vue：3.2.47

Python：3.9.12

# 2、数据结构说明

本章说明本程序系统中使用的全局数据常量、变量和数据结构。

2.1、常量

包括数据文件名称及其所在目录，功能说明，具体常量说明等。

2.2、变量

本章说明本程序系统中使用的全局数据常量、变量和数据结构。

2.3、数据结构

包括数据结构名称，功能说明，具体数据结构说明（定义、注释、取值）等。

# 3、模块设计

**3.1、软件结构**

以图形方式给出软件系统的子系统（或软件包）划分，模块划分，子系统间、模块间关系等，并用接口来描述各模块之间的调用关系，给出各模块之间的松散耦合关系。

**3.2、功能设计说明**

结合上图阐述软件的基本设计思想和理念。

**3.3、模块1**

详细描述各功能模块的功能、数据结构、具体算法和流程等。

3.3.1、设计图

3.3.2、功能描述

简要描述模块1的业务功能。

3.3.3、输入数据

详细描述用户输入的数据(包括任何输入设备)以及这些数据的有效性检验规则。

详细描述从物理模型中的哪些表获取数据以及获取这些数据的条件。

3.3.4、输出数据

详细描述模块1所产生的数据以及这些数据的表现形式。

3.3.5、数据设计

给出本程序中的局部数据结构说明，包括数据结构名称，功能说明，具体数据结构说明（定义、注释设计、取值）等。相关数据库表，数据存储设计（具体说明需要以文件方式保存的数据文件名、数据存储格式、数据项及属性等。）

3.3.6、算法和流程

详细描述根据输入数据产生输出数据的算法和流程。

3.3.7、函数说明

具体说明模块中的各个函数，包括函数名称及其所在文件，功能，格式，参数，全局变量，局部变量，返回值，算法说明，使用约束等。

3.3.8 全局数据结构与该模块的关系

说明该模块访问了哪些全局数据结构。

**3.4、模块2**

……

# 4、 接口设计

**4.1、 用户接口**

说明将向用户提供的接口。

**4.2、 外部接口**

描述本软件同外界的所有接口，包括软件、硬件、本系统与各支持系统之间的接口关系、控制方式。

前端和后端之间的接口采用RESTful API设计风格，使用HTTP协议进行通信。前端通过axios库向后端发送请求，后端返回JSON格式的响应。

**4.3、 内部接口**

4.3.1、 接口说明

例如：xx子模块通过xx从xx子模块取得xx等，相关标准，调用示例，可根据需要增加章节描述接口。

4.3.2、 调用方式

例如：

/\*\*

\*通过用户服务号码取得该客户认证密码等信息，如果该客户存在返回为0，其他情况参考错误编码

\*/

public RUserInfo getUserInfo (String userNo);

# 5、数据库设计

描述所使用的数据库系统,及数据库和数据表设计。如果系统不以数据库方式存储数据则可省略。

本系统中使用的数据库包括以下数据表：

用户表（users）：

标识符：users

数据项：

id: 用户ID，整数类型，主键，自增长

username: 用户名，字符串类型，唯一

email: 电子邮件，字符串类型，唯一

password\_hash: 密码哈希值，字符串类型

avatar\_url: 头像URL，字符串类型

introduction: 个人介绍，字符串类型

create\_time: 注册时间，日期时间类型

update\_time: 最后一次更新时间，日期时间类型

is\_admin: 是否为管理员，布尔类型

is\_active: 是否已被激活，布尔类型

帖子表（posts）：

标识符：posts

数据项：

id: 帖子ID，整数类型，主键，自增长

title: 帖子标题，字符串类型

content: 帖子内容，字符串类型

create\_time: 帖子创建时间，日期时间类型

update\_time: 最后一次更新时间，日期时间类型

is\_deleted: 是否已被删除，布尔类型

user\_id: 发帖用户ID，整数类型，外键，关联用户表的id字段

评论表（comments）：

标识符：comments

数据项：

id: 评论ID，整数类型，主键，自增长

content: 评论内容，字符串类型

create\_time: 评论创建时间，日期时间类型

update\_time: 最后一次更新时间，日期时间类型

is\_deleted: 是否已被删除，布尔类型

user\_id: 评论用户ID，整数类型，外键，关联用户表的id字段

post\_id: 所属帖子ID，整数类型，外键，关联帖子表的id字段

parent\_id: 父评论ID，整数类型，外键，可以为空，关联评论表的id字段

关注表（follows）：

标识符：follows

数据项：

id: 关注ID，整数类型，主键，自增长

follower\_id: 关注者ID，整数类型，外键，关联用户表的id字段

followed\_id: 被关注者ID，整数类型，外键，关联用户表的id字段

点赞表（likes）：

标识符：likes

数据项：

id: 点赞ID，整数类型，主键，自增长

user\_id: 点赞用户ID，整数类型，外键，关联用户表的id字段

post\_id: 被点赞帖子ID，整数类型，外键，关联帖子表的id字段

comment\_id: 被点赞评论ID，整数类型，外键，可以为空，关联评论表的id字段

通知表（notifications）：

标识符：notifications

数据项：

id: 通知ID，整数类型，主键，自增长

content: 通知内容，字符串类型

create\_time: 通知创建时间，日期时间类型

is\_read: 是否已读，布尔类型

user\_id: 接收通知的用户ID，整数类型，外键，关联用户表的id字段

这些表之间的关系如下：

用户表（users）和帖子表（posts）是一对多的关系，即一个用户可以创建多个帖子，但一个帖子只能由一个用户创建。

帖子表（posts）和评论表（comments）也是一对多的关系，即一个帖子可以有多个评论，但一个评论只能属于一个帖子。

评论表（comments）和自身之间存在一种多级关系，即一个评论可以有多个子评论（即回复），也可以没有子评论。

用户表（users）和关注表（follows）之间是多对多的关系，即一个用户可以关注多个其他用户，也可以被多个用户关注。

用户表（users）和点赞表（likes）之间也是多对多的关系，即一个用户可以给多个帖子或评论点赞，一个帖子或评论也可以被多个用户点赞。

用户表（users）和通知表（notifications）是一对多的关系，即一个用户可以收到多条通知，但一条通知只能发送给一个用户。

# 6、系统出错处理

**6.1、 出错信息**

用一览表的方式说明每种可能的错误和故障，以及系统输出信息的形式、含义和处理方式。

**6.2、 补救措施**

说明故障出现后可能采取的补救措施，如恢复、再启动技术等。

# 7、其他设计

如系统安全设计、性能设计等。

**7.1 系统安全设计**

用户认证与授权：使用身份验证机制来验证用户身份，使用OAuth 2.0 和 Bearer JWT来生成和验证访问令牌，并确保只有经过身份验证和授权的用户才能访问受保护的资源。

密码安全：采用适当的密码存储和加密方式，使用哈希函数bcrypt对用户密码进行加密，确保用户密码不以明文形式存储在数据库中。

输入验证与过滤：在前后端都进行输入验证，过滤用户输入的数据，防止恶意输入、SQL注入等攻击。

安全漏洞防护：定期更新框架和依赖库，及时修复安全漏洞。限制对敏感接口和资源的访问。

**7.2 性能设计**

数据库性能优化：使用适当的索引和查询优化来提高数据库查询性能。合理设计数据库表结构，避免数据冗余和不必要的关联。

前端性能优化：使用前端技术和工具来优化页面加载速度。

后端性能优化：使用异步处理来提高请求处理能力。

代码优化：优化代码逻辑，避免不必要的计算和数据库查询操作。使用合适的数据结构和算法来提高代码效率。

异常处理与报警：及时捕获和处理异常，发送报警通知以便及时响应和解决问题。