**东莞理工学院实训项目（2024年）之**

**RustChatGPT项目**

**系统设计文档**

编制单位： 东莞理工学院

日 期： 2022年10月

**修订历史记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **日期** | **AMD** | **修订者** | **说明** |
| V1.0 | 20240626 | A | 彭铭琨 | 新增系统设计文档 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

（A-添加，M-修改，D-删除）

目录

[系统设计文档 1](#_Toc17575)

[修订历史记录 2](#_Toc14534)

[（A-添加，M-修改，D-删除） 3](#_Toc27872)

[1. 项目概述 5](#_Toc22496)

[2. 技术选型综述对比 5](#_Toc5866)

[2.1 前端技术栈 6](#_Toc22495)

[2.2 后端技术栈 6](#_Toc24631)

[3. 系统架构设计 6](#_Toc21813)

[3.1 前端架构 6](#_Toc15637)

[3.2 后端架构 7](#_Toc7705)

[3.3 数据库设计 7](#_Toc10643)

[4. 技术架构 7](#_Toc21061)

[4.1 前端技术细节 7](#_Toc12996)

[4.2 后端技术细节 8](#_Toc3511)

[5. 数据库设计 8](#_Toc8956)

[5.1 数据库表结构 8](#_Toc24192)

[对话记录表（conversations） 8](#_Toc12528)

[历史记录表（history） 9](#_Toc1495)

[6. 技术选型的优缺点 9](#_Toc20712)

[6.1 前端技术栈 9](#_Toc18198)

[Vue.js 9](#_Toc92)

[Axios 9](#_Toc23368)

[6.2 后端技术栈 9](#_Toc26556)

[FastAPI 9](#_Toc5738)

[MySQL 9](#_Toc19248)

[Redis 10](#_Toc29595)

[7. API接口设计 10](#_Toc19088)

[8. 安全性设计 10](#_Toc10180)

[8.1 用户认证与授权 10](#_Toc12772)

[8.2 数据加密 10](#_Toc5757)

[9. 性能优化 11](#_Toc26418)

[9.1 前端性能优化 11](#_Toc9029)

[9.2 后端性能优化 11](#_Toc10080)

[10. 测试和部署 11](#_Toc26948)

[10.1 单元测试 11](#_Toc4974)

[10.2 集成测试 11](#_Toc30241)

[10.3 部署 11](#_Toc22626)

[11. 未来扩展 12](#_Toc18631)

[12. 总结 12](#_Toc20365)

# 

# 1. 项目概述

RustChatGPT项目旨在开发一个仿ChatGPT的网站，提供用户登录、注册、与ChatGPT交互以及管理用户历史对话记录等功能。该项目使用前后端分离的架构，前端技术栈主要包括HTML、CSS、JavaScript、Vue.js和Axios，后端主要使用Python、FastAPI、MySQL、Redis、SQLAlchemy以及OpenAI库等。

# 2. 技术选型综述对比

## 2.1 前端技术栈

* **HTML/CSS/JavaScript**：基础的Web开发技术，用于构建网站的结构和样式。
* **Vue.js**：用于构建用户界面的渐进式JavaScript框架，具有易用性、高性能和灵活性，适合单页应用程序（SPA）。
* **Axios**：基于Promise的HTTP客户端，用于与后端API进行通信，支持拦截请求和响应、取消请求等功能。

## 2.2 后端技术栈

* **Python**：后端开发的主要编程语言，具有丰富的库和框架，适合快速开发和迭代。
* **FastAPI**：基于Python的现代Web框架，具备高性能和易用性的特点，支持异步编程，适合构建RESTful API。
* **MySQL**：关系型数据库管理系统，具有高性能、高可靠性和高可用性，适合存储结构化数据。
* **Redis**：开源的内存数据结构存储，用作数据库、缓存和消息中间件，适合高频次读写操作的数据存储。
* **SQLAlchemy**：Python的ORM框架，提供数据库访问的高级抽象，支持多种数据库后端。
* **OpenAI库**：用于与OpenAI的GPT模型进行交互，实现ChatGPT的功能。
* **Request库**：简化HTTP请求处理的Python库。

# 3. 系统架构设计

## 3.1 前端架构

前端采用Vue.js框架，主要包括以下模块：

* **组件模块**：每个页面功能对应一个或多个Vue组件，如登录组件、注册组件、聊天组件等。
* **路由模块**：使用Vue Router进行路由管理，实现单页应用程序（SPA）功能。
* **状态管理模块**：使用Vuex进行全局状态管理，管理用户登录状态和会话信息。
* **接口请求模块**：使用Axios封装API请求，进行与后端的通信。

## 3.2 后端架构

后端采用FastAPI框架，主要包括以下模块：

* **用户管理模块**：实现用户的注册、登录、身份验证等功能。
* **聊天管理模块**：实现与OpenAI的GPT模型交互，处理用户的聊天请求。
* **历史记录管理模块**：实现用户历史对话记录的查询和管理功能。
* **数据库访问模块**：使用SQLAlchemy进行数据库操作，实现数据的持久化存储。
* **缓存模块**：使用Redis缓存高频次访问的数据，提升系统性能。

## 3.3 数据库设计

数据库使用MySQL，主要包括以下表：

* **用户表**：存储用户的基本信息，如用户名、密码等。
* **对话记录表**：存储用户与ChatGPT的对话记录，每条记录包括对话内容、时间戳等。
* **历史记录表**：存储用户的历史对话记录，包括对话ID、用户ID等。

# 4. 技术架构

## 4.1 前端技术细节

* **组件化开发**：Vue.js支持基于组件的开发，所有的页面元素和功能模块都被划分为独立的组件，使得代码结构清晰、易于维护和复用。
* **单页应用（SPA）**：通过Vue Router实现前端路由，用户可以在不同的页面之间切换而无需重新加载整个页面，提高用户体验。
* **状态管理**：使用Vuex进行全局状态管理，集中存储和管理应用的所有组件状态，保证了数据的一致性。
* **与后端通信**：使用Axios封装API请求，统一管理HTTP请求和响应，并通过拦截器进行全局错误处理。

## 4.2 后端技术细节

* **异步编程**：FastAPI支持异步编程，能够有效处理高并发请求，提高系统的响应速度和吞吐量。
* **路由和视图**：使用FastAPI定义路由和视图函数，实现各类接口的处理逻辑。每个接口都有明确的路径、请求方法和请求参数。
* **数据库操作**：使用SQLAlchemy进行数据库操作，定义数据模型，执行CRUD（创建、读取、更新、删除）操作。使用Alembic进行数据库迁移管理。
* **缓存机制**：使用Redis缓存高频次访问的数据，如用户会话信息和聊天记录，减少数据库的压力，提升系统性能。

# 5. 数据库设计

## 5.1 数据库表结构

**用户表（t\_users）**

id：用户ID，主键

username：用户名，唯一

password：加密后的密码

created\_time：创建时间

update\_time：更新时间

对话记录表(t\_history)

id：对话ID，主键

# 6. 技术选型的优缺点

## 6.1 前端技术栈

### Vue.js

**优点**：易用性高、性能优秀、文档丰富、社区活跃。

**缺点**：对于大型项目，可能需要引入更多的工具和库来完善开发环境。

### Axios

**优点**：API简单易用、支持Promise、可以拦截请求和响应。

**缺点**：对于非常复杂的HTTP请求场景，可能需要进行较多的配置。

## 6.2 后端技术栈

### Rust

**优点：**内存安全高、高性能、并发性强

**缺点：**学习曲线陡，社区规模相对较小、兼容性较差

### FastAPI

**优点**：高性能、易用性高、支持异步编程、文档生成自动化。

**缺点**：相对于Django等成熟框架，生态系统相对较小。

### MySQL

**优点**：性能高、可靠性好、支持复杂查询和事务处理。

**缺点**：对于海量数据，可能需要进行优化和扩展。

### Redis

**优点**：高性能、支持多种数据结构、应用场景广泛。

**缺点**：数据存储在内存中，对于数据量大的应用，需要考虑内存管理问题。

# 7. API接口设计

根据上传的接口设计说明书，定义了以下接口：

* **用户登录接口**：用于用户登录系统。
* **用户注册接口**：用于用户注册。
* **提示词接口**：用于接收用户的提示词并与ChatGPT进行交互。
* **查询用户所有历史对话记录接口**：用于查询用户的所有历史对话记录。
* **查询用户对应对话的历史记录接口**：用于查询用户某个特定对话的历史记录。
* **删除指定id的历史对话记录接口**：用于删除用户指定id的历史对话记录。

# 8. 安全性设计

## 8.1 用户认证与授权

* **JWT（JSON Web Token）**：使用JWT进行用户身份验证，前端在用户登录成功后获取JWT，并在每次请求时将JWT附带在HTTP头中。
* **权限控制**：在后端接口中，使用JWT对用户身份进行验证，并基于用户的角色和权限进行访问控制。

## 8.2 数据加密

* **密码加密**：使用加密算法对用户密码进行加密存储，确保密码的安全性。
* **数据传输加密**：使用HTTPS协议对数据传输进行加密，防止数据在传输过程中被窃取或篡改。

# 9. 性能优化

## 9.1 前端性能优化

* **代码分割**：使用Vue.js的代码分割功能，将代码按需加载，减少初始加载时间。
* **静态资源压缩**：对CSS、JavaScript等静态资源进行压缩，减少资源体积，提高加载速度。
* **图片优化**：使用适当的图片格式和压缩工具，减少图片文件大小。

## 9.2 后端性能优化

* **异步处理**：使用FastAPI的异步功能，处理高并发请求，提高响应速度。
* **缓存机制**：使用Redis缓存高频次访问的数据，减少数据库压力。
* **数据库优化**：通过索引、查询优化等手段，提升数据库的查询性能。

# 10. 测试和部署

## 10.1 单元测试

* **前端测试**：使用Jest和Vue Test Utils进行前端组件的单元测试。
* **后端测试**：使用pytest进行后端接口的单元测试，确保各个接口的正确性和稳定性。

## 10.2 集成测试

* **接口测试**：使用Postman进行接口测试，验证前后端通信的正确性。
* **E2E测试**：使用Cypress进行端到端测试，确保整个系统功能的完整性。

## 10.3 部署

* **前端部署**：使用Nginx进行前端静态资源的托管。
* **后端部署**：使用Gunicorn作为FastAPI的WSGI服务器，结合Nginx进行反向代理，实现高并发请求的处理。

# 11. 未来扩展

* **多语言支持**：增加多语言支持，提升用户体验。
* **移动端适配**：对移动端进行适配，提升在移动设备上的用户体验。
* **更多功能**：根据用户反馈和需求，增加更多功能，如聊天记录的分类管理、智能推荐等。

# 12. 总结

RustChatGPT项目通过前后端分离的架构设计，实现了一个高性能、易扩展的仿ChatGPT网站。前端采用Vue.js框架，构建用户界面和交互逻辑；后端采用FastAPI框架，提供高性能的RESTful API服务，并通过MySQL和Redis进行数据存储和缓存。整体技术选型兼顾了开发效率和系统性能，为项目的顺利实施奠定了基础。通过详尽的技术选型综述、架构设计、数据库设计、安全性设计、性能优化以及测试和部署方案，确保了项目的高质量和高可维护性。

RustChatGPT项目通过前后端分离的架构设计，实现了一个高性能、易扩展的仿ChatGPT网站。前端采用Vue.js框架，构建用户界面和交互逻辑；后端采用Rust、Python和FastAPI框架，提供高性能的RESTful API服务，并通过MySQL和Redis进行数据存储和缓存。整体技术选型兼顾了开发效率和系统性能，为项目的顺利实施奠定了基础。通过详尽的技术选型综述、架构设计、数据库设计、安全性设计、性能优化以及测试和部署方案，确保了项目的高质量和高可维护性。