一、 系统架构

1.1 总体架构

诗词生成系统采用前后端分离架构，以提高系统的灵活性、可维护性和可扩展性。这种架构允许前端和后端独立开发和部署，提高开发效率并简化部署和扩展操作。

前端架构：

技术框架：使用Vue.js，这是一个现代的JavaScript框架，用于构建单页面应用（SPA）。Vue.js的响应式和组件化特性使得开发复杂的用户界面变得简单和高效。

责任划分：

用户界面层：负责渲染页面视图，包括诗词展示、表单输入、导航等。

服务层：处理与后端API的通信，包括发送诗词生成请求和接收响应。

状态管理：使用Vuex进行应用级状态管理，统一管理所有组件的状态，便于状态的维护和追踪。

后端架构：

框架选择：

Flask，一个轻量级的Python web框架，适合快速开发简单但强大的Web应用。它提供了灵活的路由规则和对多种数据库的支持。

功能实现：

API服务：设计RESTful API，使前端能够通过HTTP请求与后端进行通信。

业务逻辑处理：实现诗词生成的核心算法，包括处理用户输入和调用机器学习模型。

数据库操作：使用SQLAlchemy ORM处理所有数据库交互，增加数据处理的安全性和便捷性。

1.2 技术栈

前端技术栈：

Vue.js：为用户界面提供反应性和组件化的构建能力。

HTML5：标准化的网页内容结构设计。

CSS3：用于样式化网页和界面布局，包括Flexbox和Grid系统以支持响应式设计。

JavaScript (ES6+)：实现交互逻辑，以及与后端API的通信。后端：Python 3.7+, Flask, SQLAlchemy（数据库操作）

后端技术栈：

Python 3.7+：后端编程语言，具有丰富的库支持和社区资源。

Flask：处理HTTP请求，提供REST API服务。Flask是选择的理由包括其简单性和扩展性。

SQLAlchemy：数据库ORM，允许以Pythonic方式进行数据库操作，支持多种数据库系统，简化数据库迁移和操作。

MySQL/SQLite：根据部署的规模和需求选择数据库。SQLite适用于轻量级或原型开发，MySQL适用于生产环境中需要更高性能和并发的情况。

机器学习框架：

PyTorch：以其动态计算图和易用性在学术界和研究中非常受欢迎。

二、 模块设计

2.1 用户管理模块

数据库模型：

Users: 用于存储用户信息，具体字段如下：

user\_id: 整型，主键，自增。

username: 字符串，用户名，唯一。

password\_hash: 字符串，存储经bcrypt加密后的密码。

email: 字符串，用户邮箱，唯一。

created\_at: 时间戳，记录用户的注册时间。

API 接口：

/api/users/register: POST请求，用于用户注册，接收用户名、邮箱和密码，返回注册状态。

/api/users/login: POST请求，用户登录，接收用户名和密码，返回JWT令牌。

/api/users/profile: GET请求用于获取用户信息，POST请求用于更新用户信息。

安全措施：

使用HTTPS协议保证数据传输安全。

密码通过bcrypt进行加密处理以确保安全存储。

2.2 诗词生成模块

基础诗词生成：

输入：用户提供启发诗词生成的关键字或句子。

处理：系统根据输入内容自动生成一首风格一致、主题相关的诗词。

模型选择：用户可以选择不同的深度学习模型，例如RNN、BERT等，以适应不同的生成需求。

藏头诗生成：

输入：用户输入若干个字，作为诗句的开头。

处理：系统生成一首每句诗词均以用户输入的字开头的藏头诗。

参数配置：用户可以设置诗句的长度、风格等参数。

自定义风格与形式：

风格选择：用户可以从预定义的风格（如唐诗、宋词等）中选择。

形式选择：用户可以指定诗词的具体形式，如五言绝句、七言律诗等。

API接口：

/api/poem/generate: POST请求，接受用户输入参数（包括模式选择、风格、形式等），返回生成的诗词。

性能优化：

使用GPU加速深度学习计算过程。

结果缓存，对于相同的请求提供快速响应。

2.3 用户界面模块

功能：

显示生成的诗词。

提供用户输入界面。

允许用户选择诗词的风格和形式。

技术实现：

使用Vue.js进行动态页面构建，确保前端反应迅速且用户体验流畅。

CSS3和JavaScript用于增强视觉效果和提升用户交互体验。

设计响应式用户界面，以适应不同设备和屏幕尺寸。

三、数据库设计

数据库系统：MySQL

字符集：使用UTF-8字符集，确保可以存储多语言，特别是中文字符。

3.1 用户表（Users）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 数据类型 | 键类型 | 描述 | 约束 |
| user\_id | INT | PRIMARY | 用户唯一标识 | 自增，非空 |
| username | VARCHAR(255) | UNIQUE | 用户名 | 非空 |
| password\_hash | CHAR(60) |  | bcrypt加密后的密码哈希 | 非空 |
| email | VARCHAR(255) | UNIQUE | 电子邮件地址 | 非空 |
| created\_at | TIMESTAMP |  | 注册时间 | 默认为当前时间 |

3.2 诗词表（Poems）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 数据类型 | 键类型 | 描述 | 约束 |
| poem\_id | INT | PRIMARY | 诗词唯一标识 | 自增，非空 |
| user\_id | INT | FOREIGN | 创建者的用户ID | 非空，外键引用Users.user\_id |
| content | TEXT |  | 生成的诗词内容 | 非空 |
| style | VARCHAR(255) |  | 诗词风格 | 非空 |
| created\_at | TIMESTAMP |  | 诗词生成时间 | 默认为当前时间 |

3.3会话管理表（Sessions）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 数据类型 | 键类型 | 描述 | 约束 |
| session\_id | VARCHAR(255) | PRIMARY | 会话ID | 非空 |
| user\_id | INT | FOREIGN | 关联的用户ID | 非空，外键引用Users.user\_id |
| created\_at | TIMESTAMP |  | 会话创建时间 | 默认为当前时间 |
| expires\_at | TIMESTAMP |  | 会话过期时间 | 非空 |

3.4 安全和维护

索引：

在Users.username和Users.email上创建唯一索引，加速查询并确保数据唯一性。

在Poems.user\_id上创建索引，优化按用户检索诗词的性能。

外键约束：

Poems.user\_id外键约束确保用户删除时，相关诗词也能同时处理，或设置为不允许删除含有诗词的用户。

Sessions.user\_id外键约束用于维护用户会话的一致性。

安全措施：

使用参数化查询防止SQL注入。

对敏感数据如密码进行加密处理。

四、 接口设计

4.1用户API

1、注册用户：

Endpoint: /api/users/register

Method: POST

Input:

username: 用户名

email: 电子邮箱

password: 密码

Success Response:

Code: 201 CREATED

Content: { "message": "User registered successfully" }

Error Response:

Code: 400 BAD REQUEST

Content: { "error": "Username or email already exists" }

2、用户登录

Endpoint: /api/users/login

Method: POST

Input:

username: 用户名

password: 密码

Success Response:

Code: 200 OK

Content: { "token": "<JWT\_TOKEN>" }

Error Response:

Code: 401 UNAUTHORIZED

Content: { "error": "Invalid username or password" }

3、获取用户信息

Endpoint: /api/users/profile

Method: GET

Headers:

Authorization: Bearer <JWT\_TOKEN>

Success Response:

Code: 200 OK

Content: { "username": "JohnDoe", "email": "john@example.com", "created\_at": "2021-01-01T12:00:00Z" }

Error Response:

Code: 401 UNAUTHORIZED

Content: { "error": "Invalid or expired token" }

4.2诗词生成API

1、提交生成请求

Endpoint: /api/poem/generate

Method: POST

Headers:

Authorization: Bearer <JWT\_TOKEN>

Input:

keywords: 关键字或句子

style: 选择的风格，例如“唐诗”、“宋词”

form: 诗的形式，例如“五言绝句”

Success Response:

Code: 200 OK

Content: { "poem": "Generated poem text here..." }

Error Response:

Code: 400 BAD REQUEST

Content: { "error": "Missing or invalid parameters" }

2、获取生成结果

这一功能可以通过/api/poem/generate的实时响应实现，或者设计为异步操作，其中提交请求后返回一个操作ID，然后通过另一个端点查询操作状态和结果：

Endpoint: /api/poem/status/<operation\_id>

Method: GET

Success Response:

Code: 200 OK

Content: { "status": "completed", "poem": "Generated poem text here..." }

Error Response:

Code: 404 NOT FOUND

Content: { "error": "Operation not found" }

五、 安全性设计

数据加密：使用HTTPS加密数据传输。

防护措施：

SQL注入：使用ORM的参数化查询防止SQL注入。

XSS防护：在服务器端实施严格的输入验证和适当的输出编码。

CSRF防护：利用SameSite Cookie属性或CSRF Token。身份验证和授权：

JWT用于用户身份验证和会话管理，每次API请求都需携带有效的JWT。

角色基础的访问控制（RBAC），区分用户和管理员权限。

六、性能和可维护性

性能优化：使用缓存和异步处理提高响应速度和处理能力。

日志和监控：实施详细的系统日志记录和性能监控。

错误处理：建立统一的错误处理机制，提高系统稳定性。

七、 部署方案

环境需求：Windows部署

持续集成/持续部署（CI/CD）：使用GitLab CI/CD进行自动化测试和部署。