**软件详细设计说明书**

# 诗词生成系统

**Prepared by**

**Name:**

**杨鑫月**

**胡馨月**

**曾诗涵**

**陈文洁**

目录

[诗词生成系统 1](#_Toc16526)

[1 引言 1](#_Toc20910)

[1.1目的 1](#_Toc10368)

[1.2项目背景 1](#_Toc19816)

[1.3参考资料 1](#_Toc19145)

[2总体设计 1](#_Toc5573)

[2.1需求概述 1](#_Toc5414)

[2.2总体架构 1](#_Toc14583)

[2.3技术栈 2](#_Toc24279)

[3 模块设计 3](#_Toc5035)

[3.1用户管理模块 3](#_Toc12931)

[3.2诗词生成模块 3](#_Toc19441)

[3.3用户界面模块 4](#_Toc21488)

[4 数据库设计 4](#_Toc23690)

[4.1用户表（Users） 4](#_Toc18603)

[4.2 诗词表（Poems） 4](#_Toc31598)

[4.3 会话管理表（Sessions） 5](#_Toc8928)

[4.4 安全和维护 5](#_Toc15822)

[5 接口设计 5](#_Toc2023)

[5.1用户API 5](#_Toc11488)

[5.2诗词生成API 7](#_Toc30809)

[6 安全性设计 8](#_Toc23085)

[7 性能和可维护性 8](#_Toc19881)

[8 部署方案 8](#_Toc14143)

# 1 引言

**1.1目的**

编写详细设计说明书是软件开发过程必不可少的部分，其目的是为了使开发人员在完成概要设计说明书的基础上完成概要设计规定的各项模块的具体实现的设计工作。通过对所产生的功能模块进行过程描述，开发一个可以直接转化为程序语言的软件标识，这样可以使软件设计人员很好的了解软件的功能，并对软件的所有模块有更深的了解。

**1.2项目背景**

近年来，随着人工智能技术，尤其是深度学习领域的飞速发展，自然语言处理（NLP）技术取得了前所未有的进步。诗词，作为中华文化的瑰宝，一直以来都是文人墨客表达情感、抒发怀抱的重要方式。然而，传统的诗词创作需要深厚的文化底蕴和长时间的磨练，对于普通人来说，创作出一首合格的诗词并非易事。

在这样的背景下，利用人工智能技术来辅助或自动生成诗词成为了一个热门的研究方向。诗词生成系统应运而生，它不仅能够降低诗词创作的门槛，让更多人体验到创作的乐趣，还能在一定程度上传承和弘扬中华传统文化。诗词生成系统作为一个近年来发展迅速的研究领域，随着深度学习技术的成熟，已经取得了一些令人瞩目的成果。

**1.3参考资料**

无。

# 2总体设计

**2.1需求概述**

本系统是一个基于深度学习技术的诗词生成系统，能够生成具有格律规范性、语言流畅性、创造性和想象力的诗歌，集以下功能于一体：

用户管理：管理用户的登陆注册，记录并管理用户账户的个人信息，以便使用个性化服务和保存个人设置。

基础诗词生成： 根据用户输入生成基础的诗词内容，自动生成一首风格一致、主题相关的诗词。用户可以选择不同的深度学习模型来生成诗词。

藏头诗生成： 根据用户输入的若干个字，生成一首每句诗词均以用户输入的字开头的藏头诗。用户可以设置诗句的长度、风格等参数。

自定义风格与形式： 用户可以从预定义的风格（如唐诗、宋词等）中选择，指导诗词生成的风格。此外，用户还可以指定诗词的具体形式（如五言绝句、七言律诗等）。

界面显示：提供人机交互界面，便于用户操作。同时展示满足用户需要的诗词生成结果。

**2.2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。

2.2.1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。

2.2.2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。

**2.3技术栈**

2.3.1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）

2.3.2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。

2.3.3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。

# 3 模块设计

**3.1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。

**3.2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。

**3.3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。

# 4 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。

**4.1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 |

**4.2 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 |

**4.3 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 |

**4.4 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。

# 5 接口设计

**5.1用户API**

5.1.1用户注册

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" }

5.1.2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" }

5.1.3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" }

**5.2诗词生成API**

5.2.1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" }

5.2.2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" }

# 6 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。

# 7 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。

# 8 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。