**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定义、首字母缩写词和缩略语**

无。

**1.4参考资料**

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。

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。用户可以设置生成诗词的格式。

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软件结构**

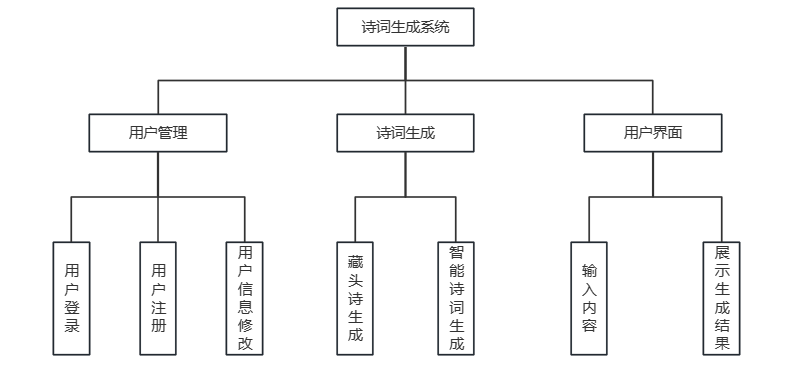


图2.2.1 系统功能结构图

**2.3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**

3.1.1 模块概述

用户管理模块负责处理用户的注册、登录、个人信息管理以及安全性维护。该模块是系统用户交互的基础，确保用户数据的安全性和准确性。

3.1.2 架构设计

用户注册：允许新用户创建账户，收集必要的用户信息。

用户登录：提供用户验证过程，发放认证令牌。

用户信息管理：允许用户查看和更新其个人信息。

安全性措施：实现密码加密、HTTPS通信、XSS和CSRF防护。

3.1.3 数据流

用户通过前端界面提交注册或登录信息。

前端将信息通过API发送到后端服务。

后端服务接收请求，进行参数校验和安全性检查。

注册请求将新用户信息存储到数据库，并生成用户令牌。

登录请求验证用户信息，成功后发放JWT令牌。

用户信息更新请求直接更新数据库中的用户记录。

3.1.4 接口定义

3.1.4.1用户注册接口

* 接口描述：用户注册时，接收注册信息，保存到用户数据库
* 请求方法：POST
* 请求URL：/api/v1.0/user/register
* 请求参数
  + username(string,required)：用户的用户名，必填
  + email(string,required)：用户的邮件，必填
  + password(string,required)：用户密码，必填
* 返回响应
  + 成功响应

{“data”:{userInfo and token},

“code”:1,

“msg”:”注册成功”}

* + 失败响应

{“data”:{},

“code”:2,

“msg”:” 邮箱格式有误，请重新输入”}

{“data”:{},

“code”:2,

“msg”:” 密码格式有误，请重新输入”}

{“data”:{},

“code”:3,

“msg”:”邮箱已注册”}

3.1.4.2用户登录接口

* 接口描述：用户登录时，接收登录信息，查询到用户数据库
* 请求方法：POST
* 请求URL：/api/v1.0/user/login
* 请求参数
  + email(string,required)：用户的邮件，必填
  + password(string,required)：用户密码，必填
* 返回响应
  + 成功响应

{“data”:{userInfo and token},

“code”:1,

“msg”:”登录成功”}

* + 失败响应

{“data”:{},

“code”:2,

“msg”:” 邮箱格式有误，请重新输入”}

{“data”:{},

“code”:5,

“msg”:” 未查询到用户信息，请检查账号或密码是否有误”}

3.1.4.3获取历史记录接口

* 接口描述：用户登录后，获取用户的查询历史信息
* 请求方法：GET
* 请求URL：/api/v1.0/user/history
* 请求参数
  + email(string,required)：用户的邮件，必填
* 返回响应
  + 成功响应

{“data”:{records list},

“code”:1,

“msg”:”成功”}

* + 失败响应

{“data”:{},

“code”:6,

“msg”:”用户未登录”}

3.1.4.4修改用户名

* 接口描述：用户登录后，修改用户名
* 请求方法：PUT
* 请求URL：/api/v1.0/user/modifyUsername
* 请求参数
  + email(string,required)：用户的邮件，必填
  + username(string,required)：修改后的用户名
* 返回响应
  + 成功响应

{“data”:{},

“code”:1,

“msg”:”修改成功”}

* + 失败响应

{“data”:{},

“code”:6,

“msg”:”用户未登录”}

3.1.4.5修改密码

* 接口描述：用户登录后，修改用户密码
* 请求方法：PUT
* 请求URL：/api/v1.0/user/modifyPassword
* 请求参数
  + email(string,required)：用户的邮件，必填
  + originPsw(string,required)：原始密码
  + currentPsw(string,required)：修改后密码
* 返回响应
  + 成功响应

{“data”:{},

“code”:1,

“msg”:”成功”}

* + 失败响应

{“data”:{},

“code”:6,

“msg”:”用户未登录”}

{“data”:{},

“code”:2,

“msg”:”用户密码错误”}

{“data”:{},

“code”:6,

“msg”:”新密码不合法”}

3.1.5 安全性考虑

密码安全：Werkzeug 提供的 security 模块来对用户密码进行哈希加密存储。

XSS防护：前端进行输入验证和输出编码。

CSRF防护：使用SameSite Cookie属性或CSRF Token进行登录验证。

3.1.6 性能优化

缓存：对于频繁访问的用户信息，使用缓存减少数据库查询。

索引：在用户表的用户名和邮箱字段上创建索引，提高查询效率。

**3.2诗词生成模块**

3.2.1 模块概述（模型体裁）

诗词生成模块是本系统的核心功能之一，它利用深度学习技术自动生成具有特定风格和形式的诗词。该模块支持用户输入关键字或句子，选择不同的风格和形式，生成相应的诗词内容。

3.2.2 架构设计

输入处理：接收用户输入的关键字、风格偏好、诗词形式等信息。

模型选择：根据用户选择的风格和形式，从预定义的深度学习模型库中选择最合适的模型进行诗词生成。

生成算法：使用选定的模型，结合输入信息，生成诗词内容。

结果输出：将生成的诗词内容返回给用户。

3.2.3 数据流

1. 用户通过前端界面提交生成请求，包括关键字、风格、形式等参数。

2. 请求通过API发送到后端服务。

3. 后端服务接收请求，进行参数校验。

4. 根据参数，从模型库中选择相应的深度学习模型。

5. 将用户输入传递给选定的模型，执行诗词生成算法。

6. 生成的诗词内容经过格式化后，通过API返回给前端。

7. 前端接收到诗词内容，并展示给用户。

3.2.4 接口定义

3.2.4.1藏头诗生成接口

* 接口描述：通过用户输入和选择的条件，生成藏头诗
* 请求方法：POST
* 请求URL：/api/v1.0/poetry/acrosticGenerate
* 请求参数
  + email(string,required)：用户的邮件，必填
  + keyword(string,required)：输入的关键词，必填
  + style(string,required)：选择的诗词的风格
* 返回响应
  + 成功响应

{“data”:{generate info},

“code”:1,

“msg”:”成功”}

* + 失败响应

{“data”:{},

“code”:6,

“msg”:”用户未登录”}

3.2.4.2智能诗词生成接口

* 接口描述：通过自定义生成的条件，生成诗词
* 请求方法：POST
* 请求URL：/api/v1.0/poetry/customizeGenerate
* 请求参数
  + email(string,required)：用户的邮件，必填
  + keyword(string,required)：输入的关键词，必填
  + style(string,required)：选择的诗词的风格
* 返回响应
  + 成功响应

{“data”:{generate info},

“code”:1,

“msg”:”成功”}

* + 失败响应

{“data”:{},

“code”:6,

“msg”:”用户未登录”}

3.2.5 算法描述

模型初始化：加载用户选择的风格和形式对应的深度学习模型。

输入预处理：对用户输入的关键字或句子进行分词、清洗等预处理操作。

模型推理：将预处理后的输入数据输入模型，进行诗词生成。

结果后处理：对模型生成的诗词进行格式化，确保符合格律要求。

3.2.6 错误处理

参数校验：确保所有输入参数符合预期格式，如关键字不为空，风格和形式在预定义选项中。

模型选择异常：如果用户选择的风格或形式没有对应的模型，返回错误提示。

生成失败：如果模型生成过程中出现异常，记录错误日志，并通知用户生成失败。

3.2.7 安全性考虑

模型保护：确保深度学习模型文件安全存储，防止未授权访问。

输入过滤：对用户输入进行过滤，防止注入攻击。

错误日志：记录详细的错误日志，但不泄露敏感信息。

3.2.8 性能优化

缓存机制：对于高频请求的风格和形式，使用缓存机制减少重复计算。

异步处理：对于计算密集型的诗词生成任务，采用异步处理方式，提高系统响应速度。

3.2.9 可维护性

模块化设计：模块内部功能划分清晰，便于单独维护和更新。

文档注释：代码中包含充分的注释和文档，说明各部分功能和用途。

**3.3用户界面模块**

3.3.1 模块概述

用户界面模块是用户与系统交互的前端界面，提供直观、易用的界面设计，使用户能够轻松地进行注册、登录和个人信息管理。

3.3.2 架构设计

动态页面构建：使用Vue.js构建动态和响应式的用户界面。

输入验证：前端进行输入数据的初步验证。

状态管理：使用Vuex管理前端状态，确保用户操作的连贯性。

3.3.3 用户交互流程

用户访问系统，选择注册或登录。

用户填写相应的表单信息。

前端进行输入验证，确保信息格式正确。

验证通过后，前端将信息发送到后端。

后端处理请求，返回结果。

前端根据后端返回的结果更新用户界面。

3.3.4 技术实现

Vue.js：用于构建单页面应用，提供动态的用户界面。

HTML5/CSS3：设计标准化的网页内容结构和样式。

JavaScript (ES6+)：实现前端逻辑，包括API调用和用户交互。

3.3.5 设计原则

直观性：界面设计直观，用户易于理解和操作。

响应性：用户界面能够适应不同设备和屏幕尺寸。

用户友好：提供清晰的错误提示和帮助信息。

3.3.6 可访问性

无障碍设计：确保界面对所有用户，包括残障用户，都是可访问的。

3.3.7 安全性考虑

输入验证：前端进行XSS攻击预防的输入验证。

HTTPS：通过HTTPS协议确保用户数据的安全。

# 4 性能设计

在网络等运行环境良好的情况下，满足以下要求：

**4.1响应时间**

（1）响应时间：用户任意操作后5秒内系统给予反馈信息。

（2）更新处理时间：由系统运行状态来决定。

（3）数据的转换和传送时间：能够在20秒内完成。

# 5 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。

**5.1用户表（Users）**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 数据类型 | 键类型 | 描述 | 约束 |
| username | VARCHAR(17) |  | 用户名 | 非空 |
| password | CHAR(17) |  | 密码 | 非空 |
| email | VARCHAR(40) | PRIMARY | 电子邮件地址，唯一标识 | 非空 |
| created\_time | TIMESTAMP |  | 注册时间 | 默认为当前时间 |

**5.2 记录表（Records）提示词**

| 字段名称 | 数据类型 | 键类型 | 描述 | 约束 |
| --- | --- | --- | --- | --- |
| Id | INT | PRIMARY | 唯一标识 | 自增，非空 |
| email | VARCHAR(40) |  | 电子邮件地址，唯一标识 | 非空 |
| Style | VARCHAR(20) |  | 诗词风格 |  |
| functionName | VARCHAR(40) |  | 使用功能的名字 | 非空 |
| model\_type | VARCHAR(20) |  | 选择的模型的类别 | 非空 |
| description | VARCHAR(255) |  | 用户的描述 | 非空 |
| Result | VARCHAR(255) |  | 生成的结果 | 非空 |
| Length | INT |  | 选择的诗词生成长度 |  |
| created\_time | TIMESTAMP |  | 生成时间 | 默认为当前时间 |
| Style | VARCHAR(20) |  | 诗词风格 |  |

**5.4 安全和维护**

·索引：

在Users.email上创建唯一索引，加速查询并确保数据唯一性。

·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。

# 6 接口设计

**6.1用户API**

6.1.1用户注册

* 接口描述：用户注册时，接收注册信息，保存到用户数据库
* 请求方法：POST
* 请求URL：/api/v1.0/user/register
* 请求参数
  + username(string,required)：用户的用户名，必填
  + email(string,required)：用户的邮件，必填
  + password(string,required)：用户密码，必填
* 返回响应
  + 成功响应

{“data”:{userInfo and token},

“code”:1,

“msg”:”注册成功”}

* + 失败响应

{“data”:{},

“code”:2,

“msg”:” 邮箱格式有误，请重新输入”}

{“data”:{},

“code”:2,

“msg”:” 密码格式有误，请重新输入”}

{“data”:{},

“code”:3,

“msg”:”邮箱已注册”}

6.1.2用户登录

* 接口描述：用户登录时，接收登录信息，查询到用户数据库
* 请求方法：POST
* 请求URL：/api/v1.0/user/login
* 请求参数
  + email(string,required)：用户的邮件，必填
  + password(string,required)：用户密码，必填
* 返回响应
  + 成功响应

{“data”:{userInfo and token},

“code”:1,

“msg”:”登录成功”}

* + 失败响应

{“data”:{},

“code”:2,

“msg”:” 邮箱格式有误，请重新输入”}

{“data”:{},

“code”:5,

“msg”:” 未查询到用户信息，请检查账号或密码是否有误”}

6.1.3修改用户名

* 接口描述：用户登录后，修改用户名
* 请求方法：PUT
* 请求URL：/api/v1.0/user/modifyUsername
* 请求参数
  + email(string,required)：用户的邮件，必填
  + username(string,required)：修改后的用户名
* 返回响应
  + 成功响应

{“data”:{},

“code”:1,

“msg”:”修改成功”}

* + 失败响应

{“data”:{},

“code”:6,

“msg”:”用户未登录”}

6.1.4修改密码

* 接口描述：用户登录后，修改用户密码
* 请求方法：PUT
* 请求URL：/api/v1.0/user/modifyPassword
* 请求参数
  + email(string,required)：用户的邮件，必填
  + originPsw(string,required)：原始密码
  + currentPsw(string,required)：修改后密码
* 返回响应
  + 成功响应

{“data”:{},

“code”:1,

“msg”:”成功”}

* + 失败响应

{“data”:{},

“code”:6,

“msg”:”用户未登录”}

{“data”:{},

“code”:2,

“msg”:”用户密码错误”}

{“data”:{},

“code”:6,

“msg”:”新密码不合法”}

**6.2诗词生成API**

6.2.1基础诗词生成接口

* 接口描述：通过用户输入和选择的条件，生成诗词
* 请求方法：POST
* 请求URL：/api/v1.0/poetry/basicGenerate
* 请求参数
  + email(string,required)：用户的邮件，必填
  + keyword(string,required)：输入的关键词，必填
  + modeltype(string,required)：选择的模型类型，必填
* 返回响应
  + 成功响应

{“data”:{generate info},

“code”:1,

“msg”:”成功”}

* + 失败响应

{“data”:{},

“code”:6,

“msg”:”用户未登录”}

6.2.2藏头诗生成接口

* 接口描述：通过用户输入和选择的条件，生成藏头诗
* 请求方法：POST
* 请求URL：/api/v1.0/poetry/acrosticGenerate
* 请求参数
  + email(string,required)：用户的邮件，必填
  + keyword(string,required)：输入的关键词，必填
  + length(Integer,required)：输出诗词的长度，必填
  + style(string,required)：选择的诗词的风格
* 返回响应
  + 成功响应

{“data”:{generate info},

“code”:1,

“msg”:”成功”}

* + 失败响应

{“data”:{},

“code”:6,

“msg”:”用户未登录”}

6.2.3自定义诗词生成接口

* 接口描述：通过自定义生成的条件，生成诗词
* 请求方法：POST
* 请求URL：/api/v1.0/poetry/customizeGenerate
* 请求参数
  + email(string,required)：用户的邮件，必填
  + keyword(string,required)：输入的关键词，必填
  + length(Integer,required)：输出诗词的长度，必填
  + style(string,required)：选择的诗词的风格
  + modeltype(string,required)：选择的模型类型，必填
* 返回响应
  + 成功响应

{“data”:{generate info},

“code”:1,

“msg”:”成功”}

* + 失败响应

{“data”:{},

“code”:6,

“msg”:”用户未登录”}

# 7 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。

# 8 系统维护设计

**8.1 日志和监控**

为了确保系统的稳定性和可维护性，我们将实施以下措施：

（1）日志记录：

使用日志框架实现统一的日志管理。

记录关键操作的日志信息，包括用户行为、系统错误和性能指标。

**8.2 错误处理**

为了提高系统的稳定性，我们将建立以下错误处理机制：

（1）统一异常处理：

设计全局异常处理器，捕捉和记录未预料的异常。

（2）错误分类：

对错误进行分类，如客户端错误、服务端错误，并给予相应的处理。

（3）用户友好的错误信息：

向用户提供清晰、友好的错误信息，避免显示技术性错误详情。

（4）错误日志分析：

定期分析错误日志，找出常见问题和潜在的性能瓶颈。

# 9 部署方案

硬件配置：

CPU：配备4核处理器，支持多线程处理。

内存：8GB RAM，以保证缓存和临时数据存储的需求。

存储：使用SSD硬盘以提高I/O性能，至少256GB存储空间。

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。