系统设计说明书

魔幻现实处理加工

目录

1. 引言

1.1 编写目的

本文档描述了一个专注于释放情绪压力的社区的系统设计，旨在提高社区质量，为用户提供更好的使用体验。该社区将部署在网页端，并在之后的阶段考虑部署到移动端。目标群体范围是大学生和青年工作者。

1.2 背景

随着社会压力的增加和生活节奏的加快，越来越多的人们面临着情绪压力的问题。特别是大学生和青年工作者群体，由于学业、工作压力大，社交压力等问题，更容易产生情绪困扰。因此，我们计划打造一个专注于释放情绪压力的社区。

2. 系统概述

2.1 系统架构

系统采用典型的前后端分离架构，前端使用 React 框架实现，提供用户友好的交互界面；后端使用 Node.js 和 Express.js 框架实现，处理业务逻辑和数据交互；数据库采用 MySQL 存储用户信息、文章内容等数据。

2.2 主要功能

系统主要功能包括：

- 用户注册和登录：用户可以注册新账户并登录系统。
- 编辑个人信息：用户可以编辑个人资料，包括性别、生日、邮箱、头像和个人描述等。
- 发布文章和评论：用户可以发布文章，并对其他用户的文章进行评论。
- 点赞功能：用户可以对文章、评论和其他用户进行点赞。
- 获取文章详情和列表：用户可以查看文章的详细信息，并根据标签分类或关键字搜索获取文章列表。
- 获取好友列表：用户可以查看自己的好友列表。

2.3 总体原则

系统设计遵循以下总体原则：

- 用户友好性：界面设计简洁明了，操作流畅，提供良好的用户体验。
- 数据安全性：采用 JWT 进行用户认证和授权，保护用户数据的安全性。

- 性能优化：采用缓存技术、负载均衡等手段进行性能优化，提高系统响应速度和并发处理能力。
- 可扩展性：采用前后端分离架构，容易扩展和维护，支持移动端部署。

3. 技术架构

技术栈

- 前端：HTML、CSS、JavaScript、React
- 后端：Node.js、Express.js
- 数据库：MySQL

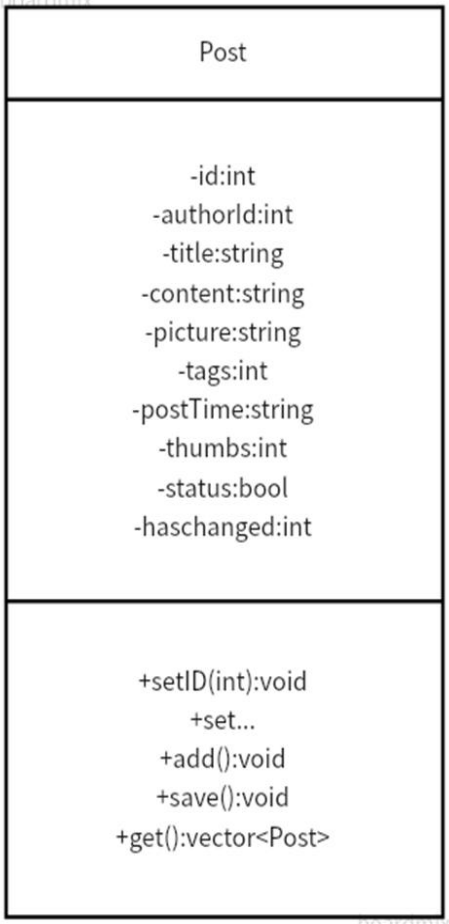
4. 数据库设计

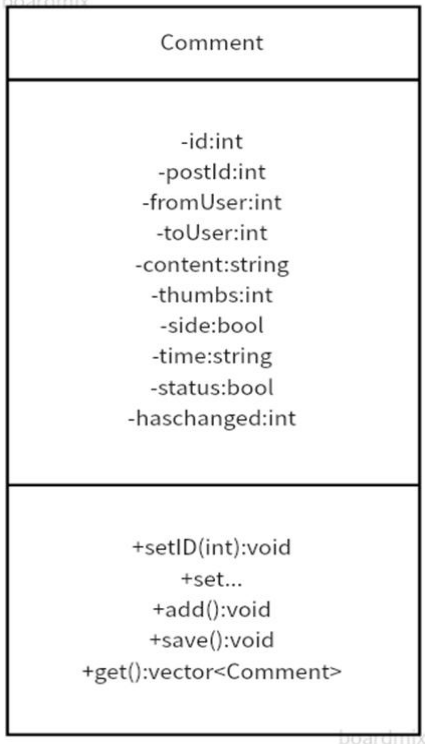
采用 MySQL 数据库，设计以下数据表：

- 用户表（User）：存储用户信息，如用户名、密码等。
- 文章表（Post）：存储用户发布的文章信息。
- 文章点赞表（PostLike）：存储用户对文章的点赞信息。
- 评论表（Comment）：存储用户对文章的评论信息。
- 评论点赞表（CommentLike）：存储用户对评论的点赞信息。
- 好友表（Friendship）：存储用户之间的好友关系信息。

数据库结构设计


User
-id:int
-username:string
-password:string
-nickname:string
-gender:bool
-isOnline:bool
-thumbs:int
-description:string
-email:string
-picture:string
-birthday:string
-privilege:bool
-haschanged:int
+setID(int):void
+set...
+add():void
+save():void
+get():vector<User>


Post
-id:int
-authorId:int
-title:string
-content:string
-picture:string
-tags:int
-postTime:string
-thumbs:int
-status:bool
-haschanged:int
+setID(int):void
+set...
+add():void
+save():void
+get():vector<Post>


Comment
-id:int
-postId:int
-fromUser:int
-toUser:int
-content:string
-thumbs:int
-side:bool
-time:string
-status:bool
-haschanged:int
+setID(int):void
+set...
+add():void
+save():void
+get():vector<Comment>

❤ 为用户、文章、评论分别创建了一个类，类中的属性与数据库中的字段一一对应。

❤ 通过调用 set 方法来设置对象中的属性

❤ add 方法往数据库中新增一条数据

❤ save 方法对数据库中已有的数据进行修改

❤ get 方法用于根据已经设置的值在数据库中查找并返回

5. API 设计

5.1 注册接口

- 请求方式：POST
- 路径：/api/users/register
- 请求参数：
  - username (string, required): 用户名
  - password (string, required): 密码
  - nickname (string, required): 昵称
  - gender (string, optional): 性别
  - birthday (string, optional): 生日
  - email (string, optional): 邮箱
  - picture (string, optional): 头像

```
{
    "command": "register",
    "params": {
        "username": "贺敏",
        "password": "idk",
        "nickname": "...",
        "gender": 1,
        "birthday": "1971-04-24",
        "email": "m.rgknyyws@qq.com",
        "picture": "http://dummyimage.com/400x400"
    }
}
```

说明：其中性别、生日、邮箱、头像为非必选项，能够满足某些用户想要保护隐私的需求。头像若不指定则使用默认头像。

5.2 登录接口

- 请求方式：POST
- 路径：/api/users/login
- 请求参数：
  - username (string, required): 用户名
  - password (string, required): 密码

```
{
    "command":"login",
    "params":{
        "username":"...",
        "password":"..."
    }
}
```

说明：在发送登陆请求后后端需要进行一系列验证，包括密码的加密与比对等，如登陆成功还需要将改用户的状态设置为在线状态。

5.3 编辑个人信息接口

- 请求方式：POST
- 路径：/api/users/edit-profile
- 请求参数：
  - username (string, required): 用户名
  - gender (string, optional): 性别
  - nickname (string, required): 昵称
  - birthday (string, optional): 生日
  - email (string, optional): 邮箱
  - picture (string, optional): 头像
  - description (string, optional): 个人描述

```
{
    "command": "modifyInfo",
    "params": {
        "username": "沈勇",
        "password": "non elit deserunt dolor sed",
        "nickname": "乔刚",
        "gender": "女",
        "birthday": "2006-10-16",
        "email": "z.ilvdzuiyn@qq.com",
        "picture": "http://dummyimage.com/400x400",
        "description": "hello"
    }
}
```

说明：与注册大致相同，多一个 description 字段

5.4 发布文章接口

- 请求方式：POST
- 路径：/api/posts/create
- 请求参数：
  - title (string, required): 文章标题
  - content (string, required): 文章内容
  - picture (string, required): 文章图片

```
{
    "command":"postPost",
    "params":{
        "title":"...",
        "content":"...",
        "picture":"..."
    }
}
```

5.5 发布评论接口

- 请求方式：POST
- 路径：/api/comments/create
- 请求参数：
  - content (string, required): 评论内容
  - userId (string, required): 用户 ID
  - postId (string, required): 文章 ID
  - replyToUserId (string, optional): 回复的用户 ID（如果是回复评论则为评论的用户 ID）

```
{
    "command":"postComment",
    "params":{
        "to":1,
        "content":"我也觉得***"
    }
}
```

说明：发布文章时需要指定你回复的用户的 id，如果没有则代表此条评论是一级评论

5.6 点赞接口

- 请求方式：POST
- 路径：/api/likes/create
- 请求参数：
  - targetType (string, required): 点赞对象类型（"post"表示文章，"comment"表示评论，"user"表示其他用户）
  - toUserID (string, required): 点赞对象 ID
  - toCommentID (string, required): 点赞对象 ID
  - toPostID (string, required): 点赞对象 ID

```
{
    "command": "doubleClick",
    "params": {
        "toUserID": 1,
        "toCommentID": 2,
        "toPostID": 3
    }
}
```

说明：点赞的对象有文章、评论和其他用户

5.7 获取文章详情接口

- 请求方式：GET
- 路径：/api/posts/:postId
- 请求参数：postID (string, required): 文章 ID

```
{
    "command":"postDetail",
    "params":{
        "postID":1
    }
}
```

5.8 获取好友列表接口

- 请求方式：GET
- 路径：/api/friends/list
- 请求参数：
  - keyword (string, optional): 关键字搜索

```
{
    "command":"getFriend",
    "params":{
        "keyWord":"110"
    }
}
```

说明：获取好友列表与搜索好友使用的是同一接口

5.9 获取文章列表接口

- 请求方式：GET
- 路径：/api/posts/list
- 请求参数：
  - tags (string, optional): 标签名称
  - keyword (string, optional): 关键字搜索

```
{
    "command": "getPost",
    "params": {
        "tags":["...","...","..."],
        "keyWord":"..."
    }
}
```

说明：获取文章使可以按照标签分类，或使用关键字搜索来获取文章，如果不使用标签或关键字搜索则系统会根据用户的好友列表来推荐文章。

5.10 查看个人信息接口

- 请求方式：GET
- 路径：/api/users/profile
- 请求参数：无

```
{
    "command":"info"
}
```

说明：后端会根据请求的 cookie 来分辨不同用户，因此无需任何参数。

6. 安全设计

系统的安全设计主要包括用户认证、数据传输和数据存储方面的安全措施：

6.1 用户认证

- 用户注册和登录：用户在注册和登录时需要提供用户名和密码，密码采用加密存储，并使用 HTTPS 协议传输，防止密码被窃取。
- JWT 认证：用户登录后，服务端颁发 JWT 令牌，客户端保存在本地，并在每次请求时通过 HTTP 头部发送 JWT 令牌进行认证，有效防止 CSRF 攻击。

6.2 数据传输

- HTTPS 协议：所有数据传输均采用 HTTPS 协议加密传输，保证数据在传输过程中的安全性。
- 输入验证：对用户输入进行有效性验证，防止 SQL 注入和 XSS 攻击。

6.3 数据存储

- 数据库安全：数据库采用 MySQL 进行数据存储，对敏感数据如用

户密码进行加密存储，避免明文存储。

- 数据备份：定期对数据库进行备份，以防止数据丢失或遭受攻击后能够及时恢复。

6.4 访问控制

- 身份验证：用户在访问需要权限的资源时，需要进行身份验证，未认证用户无法访问。
- 权限管理：根据用户角色和权限，控制用户对资源的访问权限，确保数据安全。

7. 性能优化

系统的性能优化主要包括以下方面：

7.1 缓存优化

- 数据缓存：使用缓存技术（如 Redis）对频繁访问的数据进行缓存，减少数据库访问次数，提高系统响应速度。
- 页面缓存：对静态页面进行缓存，减少页面渲染时间，提高页面加载速度。

7.2 负载均衡

- 使用负载均衡技术（如 Nginx、HAProxy）将请求分发到多台服务器上，避免单点故障，提高系统的并发处理能力和稳定性。

7.3 异步处理

- 使用异步处理技术（如消息队列）处理耗时操作，将部分业务逻辑异步执行，提高系统的并发处理能力。

7.4 数据库优化

- 数据库索引：对频繁查询的字段建立索引，加快查询速度。
- 数据库分表分库：当数据量较大时，可以考虑对数据进行分表分库，减少单表数据量，提高查询效率。

7.5 CDN 加速

- 使用 CDN（内容分发网络）加速静态资源的传输，减少用户访问时的网络延迟，提高页面加载速度。

7.6 定时任务

- 使用定时任务对系统进行数据清理、备份等操作，保持系统的稳定性和高效性。

8. 运维方案

采用日志管理工具对系统运行日志进行监控和管理，定期备份数据库和系统数据，保证系统稳定运行。

9. 结论

本文档描述了一个专注于释放情绪压力的社区的系统设计，旨在提高社区质量，为用户提供更好的使用体验。采用了现代化的技术栈和架构，以保证系统的稳定性和可扩展性。