软件工程大作业项目说明文档

GitLab名称: Dream; 队伍名称: Dream; 成员: 万振南 李声强 曾宪伟

一、需求分析

（一）用户故事

1. 用户管理部分

- 作为一个访客，我可以在系统中注册账号，并填写必要的个人信息，以便我可以使用该系统的服务。
- 作为一个已登录用户，我可以注销我的账号，以确保我的个人信息和相关数据被安全地处理并从系统中移除。
- 作为一个注册用户，我可以使用我的认证信息登录系统，并在完成使用后登出，这样我可以访问并管理我的个人信息和消息。
- 作为一个已登录用户，我可以编辑我的个人信息，如用户名、密码、头像、邮箱和手机等，以保持我的信息的最新和安全。
- 作为一个已登录用户，我可以通过关键词搜索其他用户并浏览他们的基本信息，以便我可以找到并联系我感兴趣的人。
- 作为一个已登录用户，我可以通过搜索或群聊向其他用户发送好友请求，以便对方接受后与他们建立好友关系。
- 作为一个已登录用户，我可以删除我的某个好友，以便我可以管理我的社交圈。
- 作为一个已登录用户，我可以查看和管理我的好友列表，包括对好友进行分组，以便更好地组织我的社交联系。

2. 在线会话通用部分

- 作为一个用户，我可以在聊天界面看到参与人的头像和消息，我的消息以不同颜色突出显示，以便于我区分和跟踪会话。
- 作为一个用户，在聊天界面我可以发送文本消息，以便我与其他人进行交流。
- 作为一个用户，我可以看到每条消息的发送时间和已读状态，以便我了解对话的进展。
- 作为一个用户，我可以看到每个会话的未读消息数量，以便我快速捕捉需要回应的对话。
- 作为一个用户，我可以通过右键或鼠标悬浮在消息上访问一个拓展列表，以便我进行更多操作，如回复。
- 作为一个用户，我可以在聊天中回复特定的消息，并查看被多少消息回复，以便我可以更清晰地追踪对话的上下文。
- 作为一个用户，我可以查看与某人或群聊的完整聊天记录，以便我可以回顾过去的对话。
- 作为一个用户，我可以根据时间或成员筛选聊天记录，以便我找到特定的对话内容。
- 作为一个用户，我可以删除聊天记录，以便我管理我的对话空间和隐私。

3. 在线会话群聊部分

- 作为一个用户，我可以从好友列表中选择好友创建群聊，以便我可以与多人同时进行交流。
- 作为一个用户，我可以在特定界面查看群聊的基本信息，如群名、成员和历史公告，以便我了解群聊的基本情况。

- 作为群主，我可以指定群成员为管理员，以便管理群聊的日常运作。
- 作为群主，我可以将群主身份转让给其他成员，以便群聊的长期维护和管理。
- 作为群主或管理员，我可以从群聊中移除成员，以维护群聊的健康和秩序。
- 作为群成员，我可以邀请我的好友加入群聊，但需要通过群主或管理员的审核，以便保持群聊的质量和安全。
- 作为群成员，我可以选择退出群聊，以便我可以自主管理我的参与和社交空间。

（二）用例图

二、模块与API文档

（一）模块设计

1. 用户管理模块：处理用户注册、登录、注销以及用户信息的查看和修改。
2. 好友管理模块：实现用户间好友关系的添加、删除、查看及管理。
3. 会话管理模块：管理用户的私聊和群聊会话，包括消息的发送、接收、删除和查看。
4. 群聊管理模块：处理群聊的创建、管理群成员、群设置及群公告的发布与查看。

（二）API 文档摘要

1. 用户管理

- **用户注册**：`POST /user/register`
- **用户登录**：`POST /user/login`
- **用户注销**：`POST /user/logoff`
- **查看信息**：`GET /user/get_user_info`
- **修改普通信息**：`PUT /user/update_normal_info`
- **修改身份认证信息**：`PUT /user/update_auth_info`

2. 好友管理

- **用户查找**：`GET /user/search_friends`

- **用户详情**：`GET /user/profile/{user_name}`
- **发送好友申请**：`POST /user/send_friend_request`
- **查看好友申请**：`GET /user/friend_requests`
- **回应好友申请**：`POST /user/respond_friend_request`
- **删除好友**：`POST /user/delete_friend`
- **获取好友列表**：`GET /user/get_friends`
- **添加好友到分组**：`POST /user/add_friend_to_friend_group`

3. 会话管理

- **获取所有私聊**：`GET /user/get_private_conversations`
- **聊天界面**：`GET /user/conversation/{conversation_id}`
- **发送消息**：`POST /user/send_message`
- **删除消息**：`POST /user/delete_message`
- **查询聊天记录**：`GET /user/records/{conversation_id}`
- **删除聊天记录**：`POST /user/delete_records`
- **回复消息**：`POST /user/reply_message`

4. 群聊管理

- **发起群聊**：`POST /user/create_group_conversation`
- **获取所有群聊**：`GET /user/get_group_conversations`
- **添加管理员**：`POST /user/add_admin`
- **移除管理员**：`POST /user/remove_admin`
- **转让群主**：`POST /user/transfer_owner`
- **移除群成员**：`POST /user/remove_member`
- **邀请用户加入群聊**：`POST /user/invite_member`
- **查看邀请请求**：`GET /user/view_invitations/{group_id}`
- **处理进群邀请**：`POST /user/review_invitation`
- **用户退出群聊**：`POST /user/quit_group`
- **群主解散群聊**：`POST /user/delete_group`
- **创建群公告**：`POST /user/create_group_announcement`
- **查看群公告**：`GET /user/get_group_announcements/{group_id}`

（三）关键 API

1. 发送消息

- 接口描述：允许用户在指定的会话中发送消息。
- 请求URL：/user/send_message
- 请求方法：POST
- 请求体：

```
{
  "conversation_id": "会话ID",
  "content": "消息内容"
}
```

- 成功响应：

```
{
  "message_id": "消息ID"
}
```

- 失败响应：

```
{
  "code": "错误码",
  "info": "错误信息"
}
```

- 详细流程：
  - 用户在客户端填写消息并提交到服务器。
  - 服务器验证用户的JWT令牌，确保用户已登录且参与该会话。
  - 服务器处理消息内容，存入数据库，并生成一个唯一的消息ID。
  - 服务器返回消息ID给客户端，确认消息已被存储和发送。
  - 客户端接收到消息ID，更新用户界面，显示消息发送成功。
- 异常处理：
  - 如果用户未登录或令牌无效，返回401状态码。
  - 如果指定的会话不存在或用户不在会话中，返回404状态码。
  - 如果请求数据格式错误，返回400状态码。
  - 具体错误码和错误信息详见完整API文档。

2. 获取聊天界面

- 接口描述：获取指定会话的详细聊天记录，包括消息内容和发送者信息。
- 请求URL：/user/conversation/{conversation_id}
- 请求方法：GET
- 成功响应：

```
{
  "messages": [
    {
      "msg_id": "消息ID",
      "msg_body": "消息内容",
      "sender_id": "发送者ID",
      "sender_name": "发送者用户名",
      "sender_avatar": "发送者头像URL",
      "create_time": "发送时间",
      "is_read": "是否已读"
    },
```

```
    // 更多消息...
  ]
}
```

- 失败响应：

```
{
  "code": "错误码",
  "info": "错误信息"
}
```

- 详细流程：
- 客户端请求特定会话的聊天记录。
    - 服务器验证用户身份和访问权限，确保用户已登录并属于请求的会话。
    - 服务器从数据库中检索该会话的所有消息。
    - 消息数据被发送到客户端，客户端渲染聊天界面。
- 异常处理：
    - 如果用户未登录，返回401状态码。
    - 如果会话不存在或用户不属于该会话，返回404状态码。
    - 具体错误码和错误信息详见完整API文档。

3. 回复消息

- 接口描述：允许用户回复会话中的特定消息。
- 请求URL：/user/reply_message
- 请求方法：POST
- 请求体：

```
{
  "conversation_id": "会话ID",
  "reply_to_id": "被回复的消息ID",
  "content": "回复内容"
}
```

- 成功响应：

```
{
  "message_id": "新消息ID"
}
```

- 失败响应：

```
{
  "code": "错误码",
  "info": "错误信息"
}
```

- 详细流程：

    - 用户选择一条消息进行回复，并提交到服务器。
    - 服务器验证用户的JWT令牌，确保用户已登录且参与该会话。
    - 服务器验证被回复的消息是否存在于指定会话中。
    - 服务器处理回复内容，存入数据库，并生成一个新的消息ID。
    - 服务器返回新消息ID给客户端，确认回复已被存储和发送。
    - 客户端接收到新消息ID，更新用户界面，显示回复发送成功。
- 异常处理：
    - 如果用户未登录或令牌无效，返回401状态码。
    - 如果指定的会话或消息不存在，返回404状态码。
    - 如果请求数据格式错误或回复内容超出限制，返回400状态码。
    - 具体错误码和错误信息详见完整API文档。

4. 邀请用户加入群聊

- 接口描述：允许群成员邀请其他用户加入群聊。
- 请求URL：/user/invite_member
- 请求方法：POST
- 请求体：

```
{
  "group_id": "群聊ID",
  "invitee_ids": ["被邀请者ID列表"]
}
```

成功响应：

```
{
  "added_members": ["成功添加的用户列表"],
  "already_in_group": ["已在群中的用户"],
  "invitations_sent": ["邀请已发送的用户列表"]
}
```

- 失败响应：

```
{
  "code": "错误码",
  "info": "错误信息"
}
```

- 详细流程：
    - 群主或管理员提交邀请请求到服务器。
    - 服务器验证用户的JWT令牌和身份，确保用户已登录并具有邀请权限。
    - 服务器处理邀请请求，添加用户到群聊中，并更新群聊状态。
    - 如果是群主或管理员直接添加，成功添加后返回新成员列表。
    - 如果需要审批（非管理员或群主），则创建邀请审批记录。
    - 服务器返回操作结果，客户端根据反馈更新界面。

- 异常处理：
  - 如果用户未登录或令牌无效，返回401状态码。
  - 如果群聊不存在或用户没有邀请权限，返回404状态码。
  - 如果请求数据格式错误或邀请的用户列表格式不正确，返回400状态码。
  - 具体错误码和错误信息详见完整API文档。

（四）Websocket 逻辑

1. 连接建立

- 当客户端尝试建立WebSocket连接时（用户登录），服务器端的connect方法被调用。
- 在这个方法中，首先从客户端的查询字符串中提取用户名。
- 然后，使用这个用户名将当前的WebSocket连接添加到一个全局的用户组中，这样任何发送到这个组的消息都可以转发给所有连接到这个组的客户端。
- 最后，服务器接受这个WebSocket连接。

2. 消息处理方法

- WebSocket服务器端定义了几种不同类型的消息处理方法，例如notify, read, new, quit等。
- 每种类型的方法都会接收一个包含消息相关数据的事件对象，例如消息内容、发送者信息、时间戳等。
- 服务器通过调用self.send方法将这些消息数据封装成JSON格式后发送给客户端。
- **消息通知（notify）**：这是一个处理来自其他部分（例如，通过 HTTP 请求触发的消息发送）的内部通知的方法。它接收到一个事件对象，该对象包含了会话 ID、发送者信息、消息内容等多个字段，然后将这些信息编码为 JSON 格式并通过 WebSocket 发送给客户端。

```
{
  "type": "notify",
  "conversation_id": "会话ID",
  "sender_id": "发送者ID",
  "sender_name": "发送者名称",
  "sender_avatar": "发送者头像URL",
  "message_id": "消息ID",
  "content": "消息内容",
  "timestamp": "发送时间戳",
  "unread_count": "未读消息数",
  "last_read_map": "各成员的最后阅读状态",
  "reply_to": "回复目标消息ID"
}
```

- **阅读确认（read）**：当一个用户阅读消息时，此方法将通过 WebSocket 向客户端发送一个包含未读消息数和最后阅读状态的更新。

```
{
  "type": "read",
  "conversation_id": "会话ID",
  "unread_count": "未读消息数",
  "last_read_map": "各成员的最后阅读状态"
}
```

- **新成员加入（new）**：当新成员加入到某个会话时，此方法负责通过 WebSocket 向客户端发送新成员的相关信息，如成员 ID、名字、头像等。

```
{
  "type": "new",
  "conversation_id": "会话ID",
  "member_id": "成员ID",
  "member_name": "成员名字",
  "member_avatar": "成员头像URL",
  "last_read_map": "更新后的最后阅读状态"
}
```

- **成员退出（quit）**：当成员退出会话时，此方法通过 WebSocket 向客户端发送退出通知，包括成员 ID 和更新的最后阅读映射。

```
{
  "type": "quit",
  "member_id": "退出的成员ID",
  "conversation_id": "会话ID",
  "last_read_map": "更新后的最后阅读状态"
}
```

三、数据库设计文档

CustomUser 表（自定义用户表）

- 用途：存储用户信息，包括用户名、密码、头像等。
- 字段说明：
  - id：用户ID，主键。
  - avatar_base64：用户头像的Base64编码。
  - phone：用户电话号码。
  - 其他由 Django 内置的 AbstractUser 提供的字段如用户名、邮箱等。

FriendGroup 表（好友分组表）

- 用途：定义用户的好友分组。
- 字段说明：
  - id：分组ID，主键。
  - user：分组所属的用户，外键关联到 CustomUser 表。
  - name：分组名称。
- 约束条件：一个用户的分组名称必须唯一。

Friendship 表（好友关系表）

- 用途：存储用户之间的好友关系。
- 字段说明：
  - id：关系ID，主键。
  - user：发起好友请求的用户，外键关联到 CustomUser 表。

    - friend：被添加为好友的用户，外键关联到 CustomUser 表。
    - friend_group：好友所属的分组，外键关联到 FriendGroup 表，可以为空。
- 约束条件：用户与好友之间的关系是唯一的。

FriendshipRequest 表（好友请求表）

- 用途：存储用户之间的好友请求。
- 字段说明：
    - id：请求ID，主键。
    - user：发起好友请求的用户，外键关联到 CustomUser 表。
    - friend：接受好友请求的用户，外键关联到 CustomUser 表。
    - status：请求状态（待处理、接受、拒绝）。
    - create_time：请求创建时间。
- 约束条件：每个用户对另一用户的好友请求是唯一的。

Conversation 表（对话表）

- 用途：存储用户之间的对话信息，包括群聊和私聊。
- 字段说明：
    - id：对话ID，主键。
    - members：对话参与的用户，多对多关系关联到 CustomUser 表。
    - history_members：曾经参与过的用户，多对多关系关联到 CustomUser 表。
    - name：对话名称，可以为空。
    - avatar_base64：对话的头像，可以为空。
    - create_time：创建时间。
    - update_time：更新时间。
    - is_group：是否为群聊。
    - type：对话类型（私聊、群聊）。
    - owner：群主，外键关联到 CustomUser 表，可以为空。
    - admin：群管理员，多对多关系关联到 CustomUser 表。

Message 表（消息表）

- 用途：存储对话中的消息。
- 字段说明：
    - id：消息ID，主键。
    - conversation：所属对话，外键关联到 Conversation 表。
    - sender：消息发送者，外键关联到 CustomUser 表。
    - receivers：消息接收者，多对多关系关联到 CustomUser 表。
    - content：消息内容。
    - timestamp：发送时间。
    - read_by：已读用户，多对多关系关联到 CustomUser 表。

- deleted_by：删除消息的用户，多对多关系关联到 CustomUser 表。
- reply_to：回复的消息，外键关联到 Message 表自身。
- reply_count：回复数量。

UserConversationStatus 表（用户对话状态表）

- 用途：存储用户在特定对话中的状态信息。
- 字段说明：
  - user：用户，外键关联到 CustomUser 表。
  - conversation：对话，外键关联到 Conversation 表。
  - unread_count：未读消息数。
  - last_read_at：最后阅读时间。
  - last_read_message_id：最后阅读的消息ID。
- 约束条件：用户与对话之间的关系是唯一的。

Invitation 表（邀请表）

- 用途：存储群聊中的邀请信息。
- 字段说明：
  - id：邀请ID，主键。
  - group：所属群聊，外键关联到 Conversation 表。
  - invitee：被邀请的用户，外键关联到 CustomUser 表。
  - inviter：发起邀请的用户，外键关联到 CustomUser 表。
  - status：邀请状态（待处理、已接受、已拒绝）。
  - created_at：创建时间。
- 约束条件：在一个群聊中，对一个用户的邀请是唯一的。

GroupAnnouncement 表（群公告表）

- 用途：存储群聊中的公告信息。
- 字段说明：
  - id：公告ID，主键。
  - group：所属群聊，外键关联到 Conversation 表。
  - creator：公告创建者，外键关联到 CustomUser 表。
  - content：公告内容。
  - create_time：创建时间。
- 排序规则：按创建时间降序排序。

附录

完整API文档：https://zhennan1.github.io/software-engineering/api/