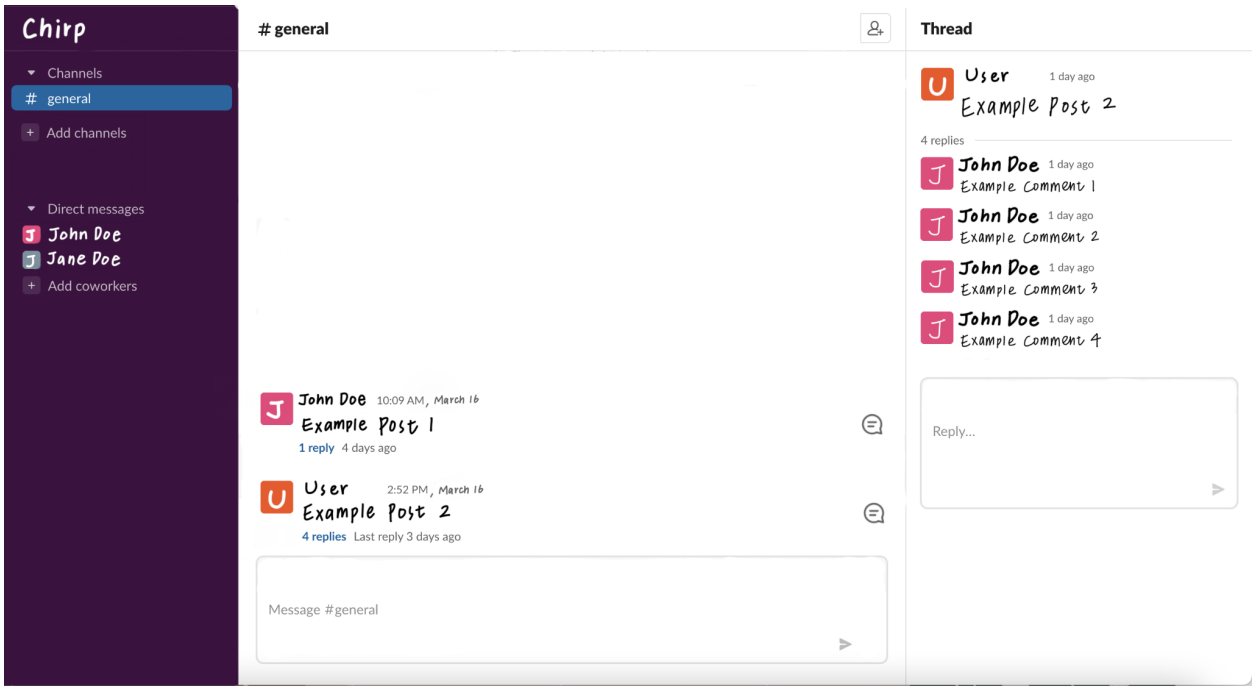
**Project Title: Chirp**

**Project Member:**
Henry Lu(ihsuanl), Yen-Ju Liao (yenjjulia), Amelia Lee (chichenl)

**Inspiration of Naming:**
It's a reference to the sound that birds make, which could tie in nicely with a messaging app that allows people to "chirp" back and forth with each other.

**Project Description:**
Chirp is an **instant messaging and collaboration platform** designed for teams. It allows team members to communicate and collaborate in real time via chat or uploading images. With Chirp, users can organize conversations into separate channels based on topics. It also offers features like messaging threads to help teams stay organized and focused.



# Product Backlogs

## ## Overview

- Init: Project setup (Sprint 1)
- Auth: User authentication (Sprint 1)
- Post: Create channel (Sprint 2)
- Post: Search channel (Sprint 2)
- Post: Posts (Sprint 2)
- Post: Comments (Sprint 2)
- Common: Upload images (Sprint 2)
- Chat: New conversation (Sprint 3)
- Chat: Messages (Sprint 3)
- Chat: Search messages (Sprint 3)
- Chat: Online status (Sprint 3)

**### Init: Project setup**
*> Prerequisites for starting development*

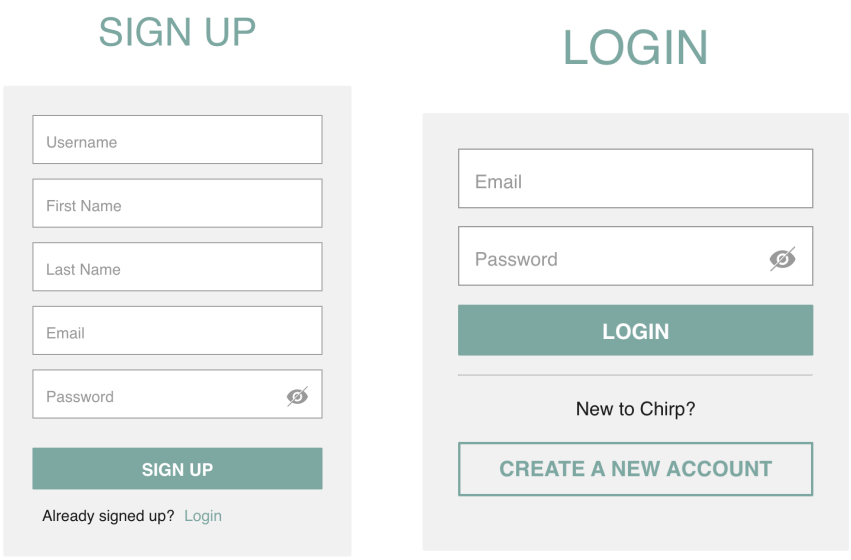
- Project setup
  - ~~setup Spring Boot web server~~
  - setup React JS web client
  - setup CI with GitLab runner
  - setup Django web server

- Database schema design
- Wireframes or HTML mockups

**### Auth: User authentication**
*> Allow users to create and log in to their accounts using email and password.*

- Log in
  - The user should be able to log in with their credentials if s/he had registered before
- Sign up
  - The user should be able to sign up for a new account with their email address

```
Unset
POST /api/login
POST /api/signup
POST /api/logout
```



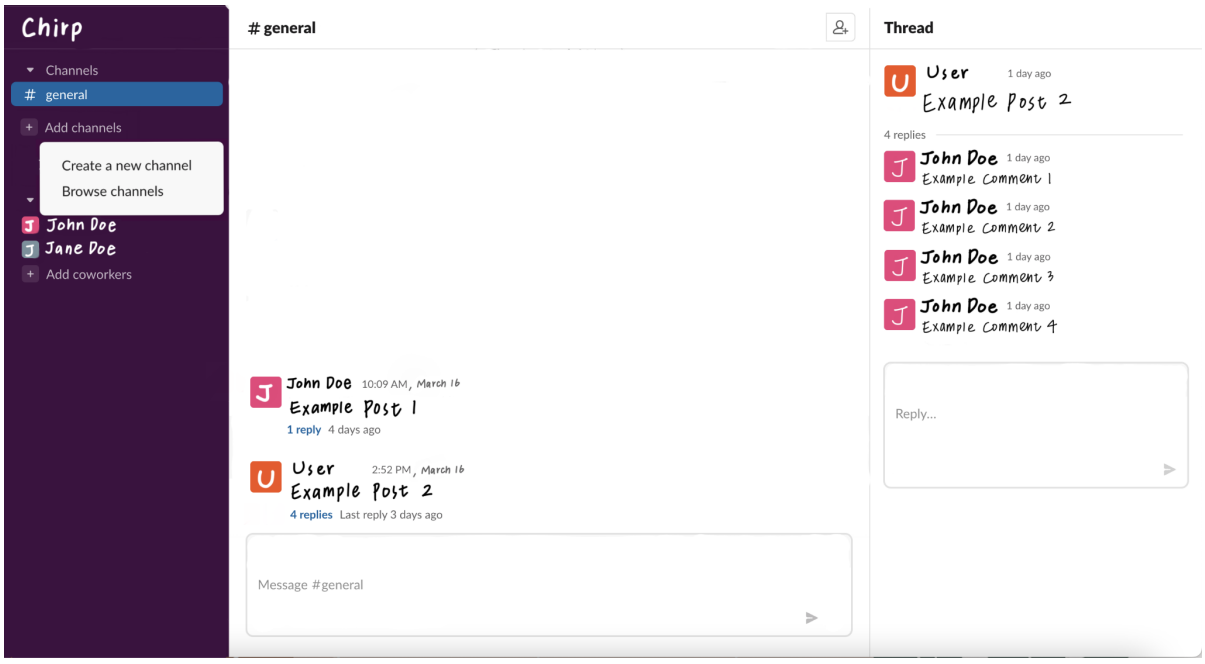

**### Post: Create channel**

*> Allow users to create a channel*

- Create channel
  - a popped-up modal for user to type in the channel's `name` and `descriptions`

Unset

`POST /api/channel`



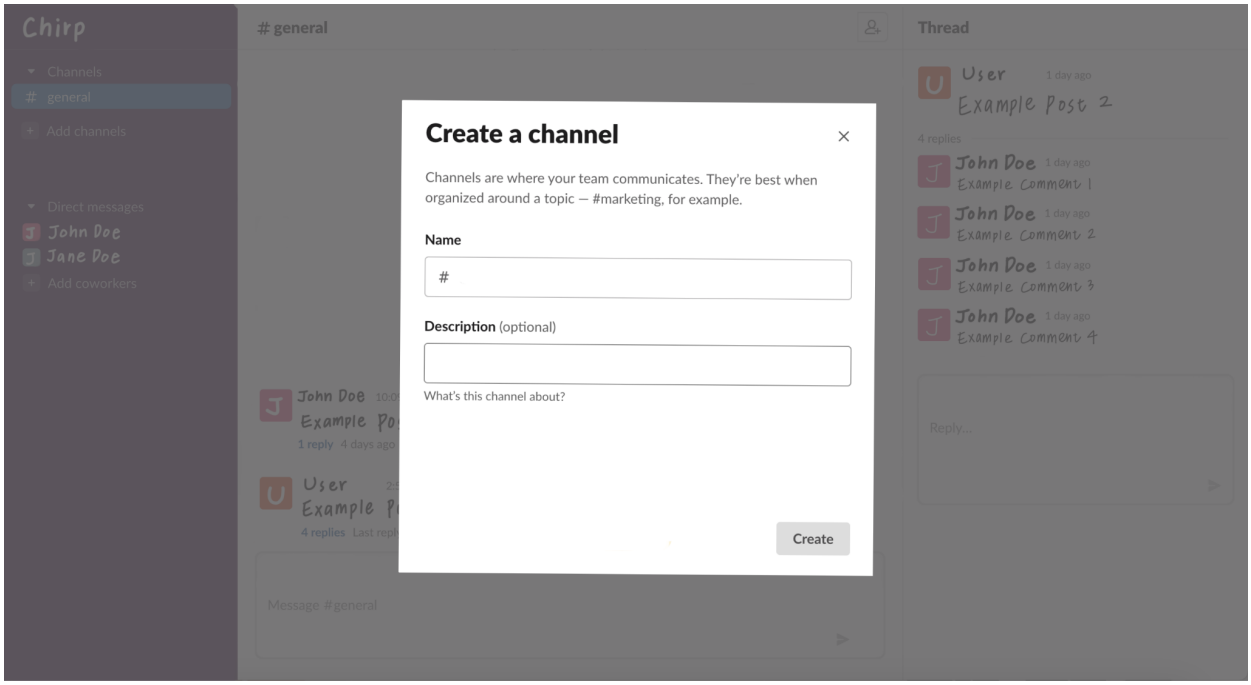
### Post: Search channel

*> Allow users to search and join the channel*

- Search channels
  - display a  list of channels for users to choose from
  - a search bar on top of the list for users to search and `filter` channels

- Join channel
  - allow users to join the selected channel

```
GET /api/channels  # query params for filtering
POST /api/channels/{id}/join
```




### Post: Posts

*> Allow users to create a post in channels*

- Create post
  - users can submit a post using the textarea at the bottom of the channel

- Load posts
  - users should see the previous posts with the specific connection upon navigating into the chat room

- pagination should be implemented (offset-limit based)
    - scroll up to load more messages

```
Unset
POST /api/channels/{id}/post
GET /api/channels/{id}/posts
```

**### Post: Comments**
*> Allow users to start or leave comments on any messages in the chat room*

- Create comment
    - Users should be able to submit comments to any posts (in a separate view)

- Load comments
    - There will be a button showing the count of the comments right below the `post`
        - On clicking the button, a separate view listing the corresponding comments will pop out (no pagination is needed)

```
Unset
POST /api/channels/{id}/posts/{id}/comment
GET /api/channels/{id}/posts/{id}/comments
```

**### Common: Upload images**
*> Allow users to upload images in the chat room*

- Upload Image
    - Users should be able to upload images in the chat room
    - Users can select any image files that are less than 10MB from their disk

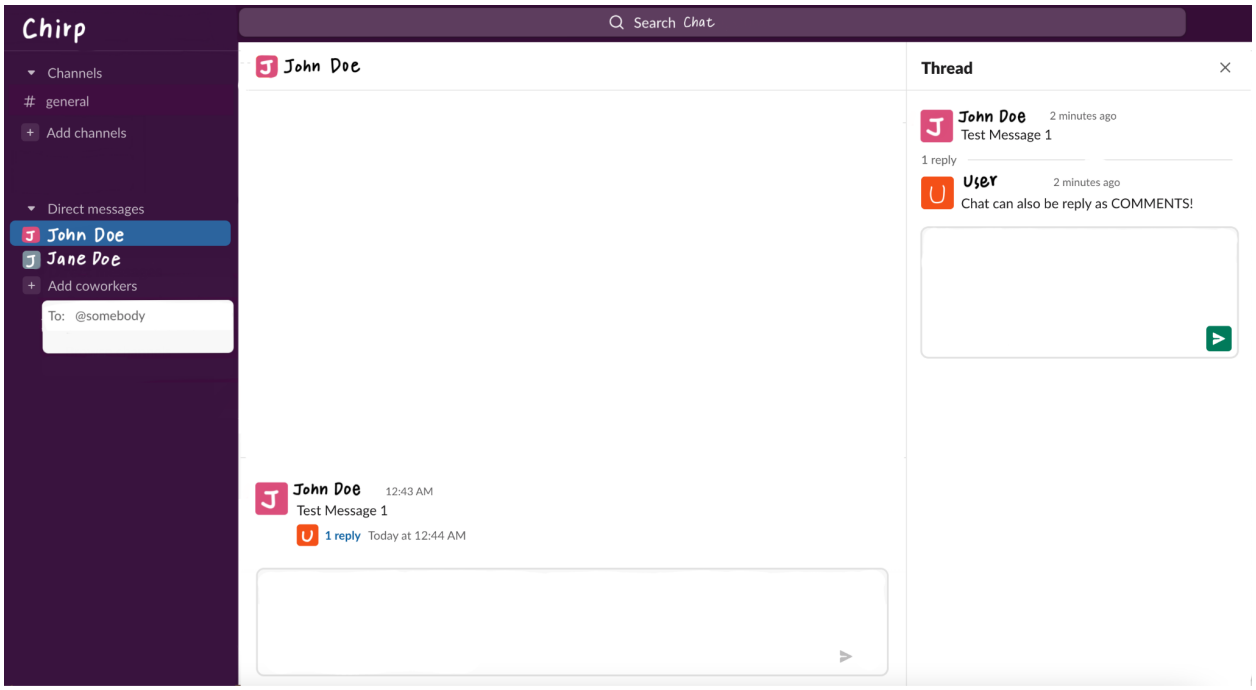
```
Unset
POST /api/image
GET /api/images/{signed_url}
```

### Chat: New conversation

*> Allow users to search for other users and initiate a conversation*

- Search for other users
  - Users can search for other users via their `email address` or `username` and then send messages followingly.
  - a popped-up modal for user to type in the user's `username` and `emails`

```
Unset
GET /api/users
POST /api/conversation
```




### Chat: Messages

*> Allow users to chat with others, and the same goes the other way around. (support plain text only in the product backlog)*

- Send message
  - Users should be able to send messages to other users, one at a time. i.e., it should be a one-to-one chat room, but multiple chat rooms are supported.

- Receive message
  - Users should see the incoming messages in a real-time manner.
  - For simplicity, we will do it with message polling (5-sec interval)

- Load message
    - Users should see the previous messages with the specific connection upon navigating into the chat room
        - Pagination should be implemented (offset-limit based)
            - Scroll up to load more messages
        - The latest message should appear at the very bottom

```
Unset
POST /api/conversation/{id}/message
GET /api/conversation/{id}/messages
```

**### Chat: Search messages**
*> Allow users to search the previous messages (within the chat room)*

- Search Messages
    - Users should be able to search for any previous messages within the chat room via a search bar
        - The search results will be texts similar to the input text (LIKE)

```
Unset
GET /api/conversation/{id}/messages
```

# Product Backlogs - Sprint 1 (Product Owner: yenjulia)

> complete list and assignees

**### Project setup**

- Project setup
  - ~~setup Spring Boot web server -> **ihsuanl**~~
  - setup React JS web client -> **ihsuanl**
  - setup CI with GitLab runner -> **ihsuanl**
  - **-** setup Django webserver -> **yenjulia**

- Create product backlogs -> **ihsuanl**
- Database schema design -> **yenjulia**
- Wireframes or HTML mockups -> **chichenl**

**### User authentication**

- Log in
  - frontend -> **ihsuanl**
  - backend -> **chichenl**
  - integration -> **chichenl**

- Sign up
  - frontend -> **ihsuanl**
  - backend -> **chichenl**
  - integration -> **chichenl**

# Product Backlogs - Sprint 2 (Product Owner: chichenl)

**### Post: Create channel** -> **chichenl**
- Create channel
  - a popped-up modal for user to type in the channel's `name` and `descriptions`

**### Post: Search channel** -> **chichenl**
- Search channels
  - display a  list of channels for users to choose from
  - a search bar on top of the list for users to search and `filter` channels
- Join channel
  - allow users to join the selected channel

**### Post: Posts** -> **yenjulia**
- Create post
  - users can submit a post using the textarea at the bottom of the channel
- Load posts
  - users should see the previous posts with the specific connection upon navigating into the chat room
    - pagination should be implemented (offset-limit based)
      - scroll up to load more messages

**### Post: Comments** -> **yenjulia**
- Create comment
  - Users should be able to submit comments to any posts (in a separate view)
- Load comments
  - There will be a button showing the count of the comments right below the `post`
    - On clicking the button, a separate view listing the corresponding comments will pop out (no pagination is needed)

**### Common: Upload images** -> **ihsuanl**
- Upload Image
  - Users should be able to upload images in the chat room
  - Users can select any image files that are less than 10MB from their disk

# Database Schema

User Table:

User_id, (primary key)
Email,
Username
First_Name,
Last_Name,
Hashed_Password,
Profile_image_id,

Conversation Table:
Conversation_id, (primary key)
Created_at,

UserConversation Table:
User_id  (foreign key to User.user_id)
Conversation_id (foreign key to Conversation.conversation_id)

Message Table:
Message_id (primary key)
Content,
Created_at
Conversation_id (foreign key to Conversation.conversation_id)
Sender_user_id (foreign key to User.user_id)

Post Table:
Post_id, (primary key)
User_id, (foreign key to User.user_id)
Channel_id (foreign key to Channel.channel_id)
Content,
Created_At,
Photo_id (*only one photo allowed per post)

Comment Table:
Comment_id, (primary key)
User_id, (foreign key to User.user_id)
Post_id, (foreign key to Post.post_id)
Content,
Created_at,

Photo_id,

Channel Table:

Channel_id, (primary key)
Channel_Name,
Owner_User_id,  (foreign key to User.user_id)

UserChannel Table:

User_id, (foreign key to User.user_id)
Channel_id, (foreign key to Chanel.channel_id)