The Snickr:

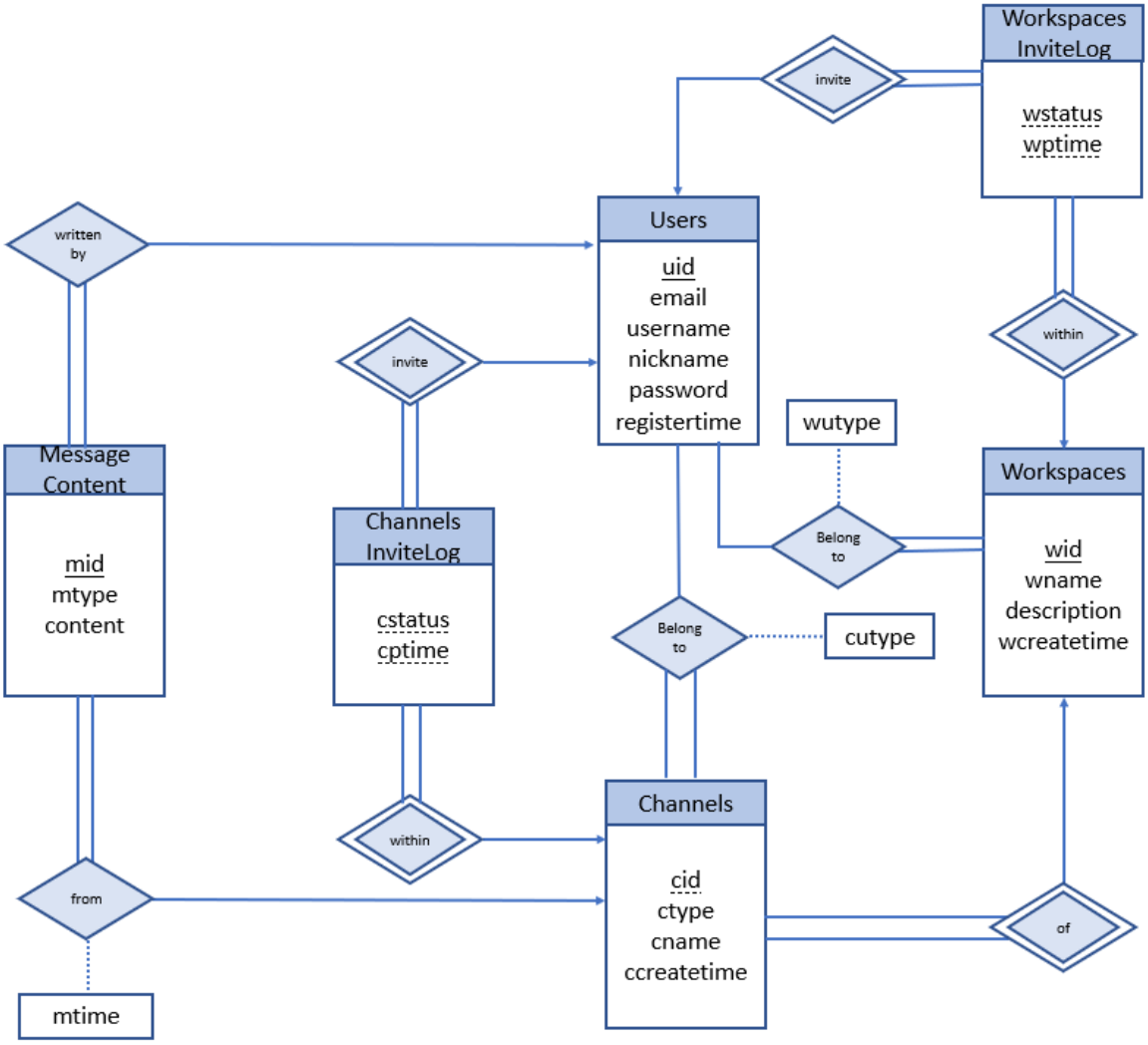
Web Application and Schema Design

Jiaqi Li, jl9555, N14088502

Zhenghan He, zh1158, N15127395

Introduction

In this project, we are trying to build a web-based collaboration system called “snickr”, similar to Slack. It consists of several main parts: Users, Workspaces, Channels, Messages and Invitations. The whole ER-diagram of this web-based collaboration system is shown below.



Explanation

Here is the explanation of the tables which turned from the E-R model and the schema. The primary keys of each table are denoted by underlines.

Users (uid, email, username, nickname, password, registertime)

- Table "Users" stores detailed information of all users who have signed up at our website. Every time when someone signs up at our website, we first check whether his e-mail address is unique in our database and then assign a globally unique "uid" to him. We also store his username, nickname, password and time stamp of this register.

Workspaces (wid, wname, description, wcreatetime)

- Table "Workspaces" stores detailed information of all workspaces. Every time when a user creates a workspace, we first check whether the "wname" is unique in our database and then assignment a globally unique "wid" to that workspace. We also store description and time stamp of creation.

WU (wid, uid, wutype)

- Table "WU" stores relationship between different workspaces and users. Every time when a user creates a workspace or someone accepts the invitation to a particular workspace, we add a record into this table.
- The attribute "wutype" has three types. "ORIGINAL_ADMIN" indicates this user is creator of this workspace. "SELECTED_ADMIN" means this user is an administrator selected by the creator. "MEMBER" shows that this user is a member in the workspace.
- WU(wid) references Workspace(wid)
- WU(uid) references User(uid)

WorkspacesInviteLog (createuid, inviteduid, wid, wstatus, wptime)

- Table "WorkspacesInviteLog" stores all the invitations to a particular workspace between different users. A user must be an administrator (either original administrator or selected administrator) and must log in his own workspace so that he can invite other people to the workspace. We make sure of this through our front-end judgment. Every time when he invites someone else to join, we add a record in this table.
- The attribute "createuid" and "inviteduid" means the user who sent invitation and the user who was invited respectively. The attribute "wptime" is the time stamp of this record. The attribute "wstatus" has three types. "SENT" indicates the invitation was created. "ACCEPT" means that the user who was invited accepted that invitation. "REFUSE" shows that the user who was invited refused that invitation.

- WorkspacesInviteLog(createuid) references User(uid)
- WorkspacesInviteLog(inviteduid) references User(uid)
- WorkspacesInviteLog(wid) references Workspace(wid)

Channels (wid, cid, cname, ctype, ccreatetime)

- Table "Channels" stores detailed information of all channels. A user must log in his own workspace and then create a channel. We make sure of the situation where the user creating the channel is a member of the workspace through our front-end judgment. Every time when a user creates a channel, we record its "wid" and assign "cid" to that channel.
- "ctype" has three different types, which are public, private, and direct.
- Channels(wid) references Workspace(wid)

CU (wid, cid, uid, cutype)

- Table "CU" stores relationship between different channels and users. Every time when a user creates a channel OR someone accepts the invitation to a particular private or direct channel OR a user joins a public channel by himself, we add a record into this table.
- The attribute "cutype" has two types. "CREATOR" indicates this user is the creator of this channel. "MEMBER" shows that this user is a member in this channel.
- CU(wid, cid) references Channels(wid, cid)
- CU(uid) references User(uid)

ChannelsInviteLog (createuid, inviteduid, wid, cid, cstatus, cptime)

- Table "ChannelsInviteLog" stores all the invitations towards a particular channel between different users. For private or direct channels, a user must be a creator and must log in his own workspace so that he can invite other people to the channel he created. We make sure of this through our front-end judgment. Every time when he invites someone else to join, we add a record in this table.
- The attribute "createuid" and "inviteduid" means the user who sent invitation and the user who was invited respectively. The attribute "cptime" is the time stamp of this record. The attribute "cstatus" has three types. "SENT" indicates the invitation was created. "ACCEPT" means that the user who was invited accepted that invitation. "REFUSE" shows that the user who was invited refused that invitation.
- ChannelsInviteLog(createuid) references User(uid)
- ChannelsInviteLog(inviteduid) references User(uid)
- ChannelsInviteLog(wid, cid) references Channels(wid, cid)

MessageContent (mid, uid, mtype, content)

- Table "MessageContent" stores detailed information of all messages appeared. Every time when someone edited and published a message, we assign a globally unique "mid" to this message. We also store his uid, type of the message and content of the message.
- The attribute "mtype" has six types. "TEXT" means it's a plain text message. "IMAGE" indicates a message of image, which stores its URL only. "HYPERLINK" shows that a message includes a hyperlink. "MARKUP" means a message contains rich text markup. "EMOJIS" indicates that it's a plain emojis message. "HYBRID" shows that a message is a combination of "TEXT" and "EMOJIS".
- MessageContent(uid) references User(uid)

MessageFrom (wid, cid, mid, mtime)

- Table "MessageFrom" stores relationship between different channels and messages. A user can only read or publish messages through his own channels or channels that he joined. We make sure of this through our front-end judgment. Each time when a user publishes a message to a particular public, private or direct channel, we add a record into this table. We also store the time stamp of this message when it's been post.
- MessageFrom(wid, cid) references Channels(wid, cid)
- MessageFrom(mid) references MessageContent(mid)

Create the schema

```
DROP SCHEMA IF EXISTS  snickr;
CREATE SCHEMA snickr;
USE snickr;

CREATE TABLE Users(
  uid INT(11)  NOT NULL AUTO_INCREMENT PRIMARY KEY,
  email VARCHAR(32) NOT NULL UNIQUE,
  username VARCHAR(32),
  nickname VARCHAR(32),
  password VARCHAR(32),
  registertime TIMESTAMP
);

CREATE TABLE Workspaces(
  wid INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  wname VARCHAR(32),
```

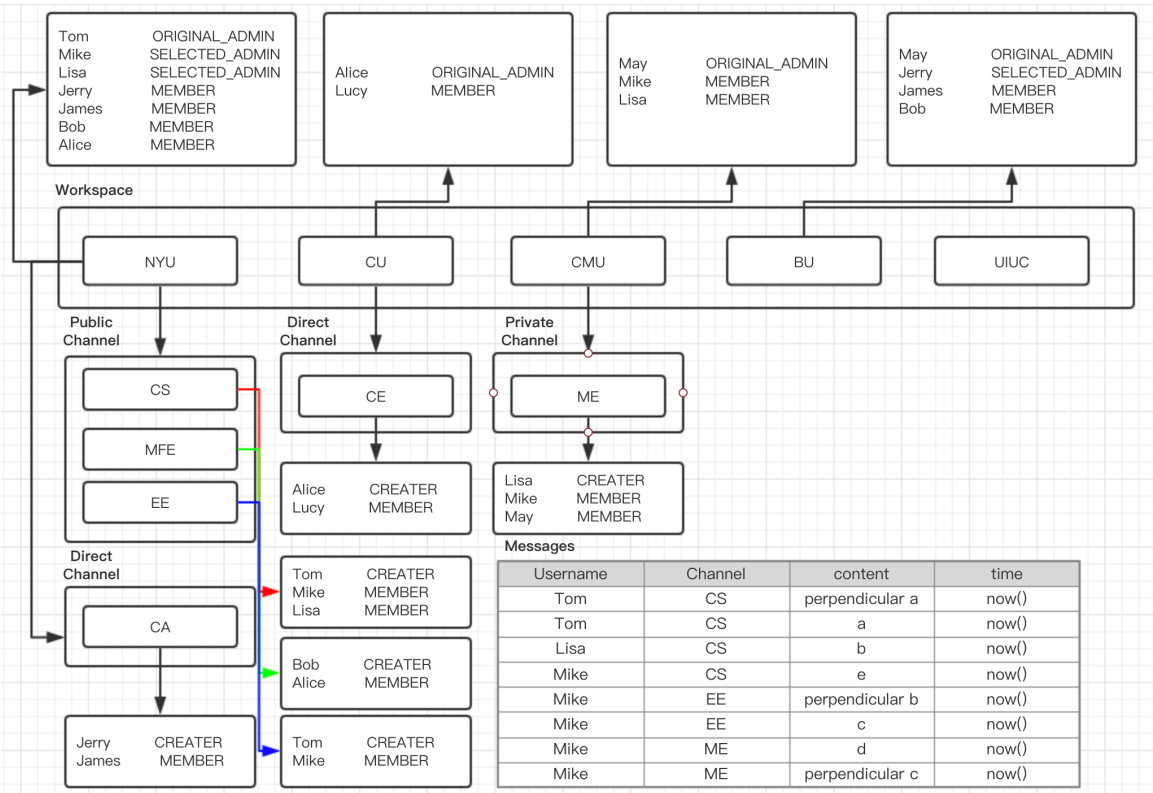
```
  description VARCHAR(32),
  wcreatetime TIMESTAMP
);

CREATE TABLE Channels(
  cid INT(11) NOT NULL AUTO_INCREMENT,
  wid INT(11),
  cname VARCHAR(32),
  ctype ENUM('PUBLIC', 'PRIVATE','DIRECT'),
  ccreatetime TIMESTAMP,
  PRIMARY KEY (cid,wid),
  FOREIGN KEY (wid) references Workspaces(wid) ON DELETE CASCADE
);

CREATE TABLE MessageContent(
  mid INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  uid INT(11),
  mtype ENUM('TEXT','IMAGE','HYPERLINK','MARKUP','EMOJIS','HYBRID'),
  content VARCHAR(128),
  FOREIGN KEY (uid) references Users(uid) ON DELETE CASCADE
);

CREATE TABLE MessageFrom(
  wid INT(11),
  cid INT(11),
  mid INT(11),
  mtime TIMESTAMP,
  PRIMARY KEY (wid,cid,mid),
  FOREIGN KEY (mid)
      REFERENCES MessageContent(mid)
      ON DELETE CASCADE,
  FOREIGN KEY (cid,wid) references Channels(cid,wid) ON DELETE CASCADE
);

CREATE TABLE WorkspacesInviteLog(
```

```
  createuid INT(11),
  inviteduid INT(11),
  wid INT(11),
  wstatus ENUM('SENT','ACCEPT','REFUSE'),
  wptime TIMESTAMP,
  PRIMARY KEY (createuid,inviteduid,wid,wstatus,wptime),
   FOREIGN KEY (createuid) references Users(uid) ON DELETE CASCADE,
  FOREIGN KEY (inviteduid) references Users(uid) ON DELETE CASCADE,
  FOREIGN KEY (wid) references Workspaces(wid) ON DELETE CASCADE
);

CREATE TABLE ChannelsInviteLog(
  createuid INT(11),
  inviteduid INT(11),
  wid INT(11),
  cid INT(11),
  cstatus ENUM('SENT','ACCEPT','REFUSE'),
  cptime TIMESTAMP,
  PRIMARY KEY (createuid,inviteduid,wid,cid,cstatus,cptime),
  FOREIGN KEY (createuid) references Users(uid) ON DELETE CASCADE,
  FOREIGN KEY (inviteduid) references Users(uid) ON DELETE CASCADE,
  FOREIGN KEY (cid,wid) references Channels(cid,wid) ON DELETE CASCADE
);

CREATE TABLE WU(
   wid INT(11),
   uid INT(11),
   wutype ENUM('ORIGINAL_ADMIN','SELECTED_ADMIN','MEMBER'),
   PRIMARY KEY (wid,uid),
   FOREIGN KEY (wid) references Workspaces(wid) ON DELETE CASCADE,
   FOREIGN KEY (uid) references Users(uid) ON DELETE CASCADE

);

CREATE TABLE CU(
```

```
  wid INT(11),
  cid INT(11) ,
  uid INT(11) ,
  cutype ENUM('CREATOR','MEMBER'),
  PRIMARY KEY (wid,cid,uid),
  FOREIGN KEY (cid, wid)
      REFERENCES Channels(cid, wid)
      ON DELETE CASCADE,
  FOREIGN KEY (uid)
    REFERENCES Users(uid)
    ON DELETE CASCADE
);
```

Test Data

In order to make a test on our schema, we design the test data very carefully. It includes some interesting test cases. Here's the overall view of our test data.



Username	Channel	content	time
Tom	CS	perpendicular a	now()
Tom	CS	a	now()
Lisa	CS	b	now()
Mike	CS	e	now()
Mike	EE	perpendicular b	now()
Mike	EE	c	now()
Mike	ME	d	now()
Mike	ME	perpendicular c	now()

Totally, there are 10 users, 5 workspaces, 6 channels and 8 messages. In addition, our detailed test data will be shown in the following 7 tables and 2 views.

Table1.Users

uid	email	username	nickname	password	registertime
1	Tom@example.com	Tom	Tommy	tompasswprd	2019-01-01 12:00:00
2	Mike@example.com	Mike	Mickey	mikepwd	2019-01-01 12:00:00
3	Lisa@example.com	Lisa	nLisa	lisapwd	2019-01-01 12:00:00
4	Jerry@example.com	Jerry	nJerry	jerrypwd	2019-01-01 12:00:00
5	James@example.com	James	Jimmy	jamespwd	2019-01-01 12:00:00
6	Bob@example.com	Bob	BB	bbpwd	2019-01-01 12:00:00
7	Alice@example.com	Alice	nAlice	alicepwd	2019-01-01 12:00:00
8	Lucy@example.com	Lucy	nLucy	lucypwd	2019-01-01 12:00:00
9	May@example.com	May	MM	maypwd	2019-01-01 12:00:00
10	Ken@example.com	Ken	Keny	kenpwd	2019-01-01 12:00:00

There are 10 users in total. However, only nine of them belong to different channels in different workspaces. The last one doesn't belong to any workspace or channel. We allow such cases to happen, for example, this user just created an account and did not do anything yet.

Table2.Workspace

wid	wname	description	wcreatetime
1	NYU	NYUdescription	2019-01-02 12:00:00
2	CU	CUdescription	2019-01-02 12:00:00
3	CMU	CMUdescription	2019-01-02 12:00:00
4	BU	BUdescription	2019-01-02 12:00:00
5	UIUC	UIUCdescription	2019-01-02 12:00:00

There are 5 workspaces in total. However, only four of them have administrators and members. The last one doesn't hold people there. It simulates a situation where admins in that workspace called UIUC remove all members out of the workspace and then exit the workspace by themselves. Our database will store the information about such a vacant workspace for a period of time, say half a year.

Table3.WU

wid	uid	wutype
1	1	ORIGINAL_ADMIN
1	2	SELECTED_ADMIN
1	3	SELECTED_ADMIN
1	4	MEMBER
1	5	MEMBER
1	6	MEMBER
1	7	MEMBER
2	7	ORIGINAL_ADMIN
2	8	MEMBER
3	2	MEMBER
3	3	MEMBER
3	9	ORIGINAL_ADMIN
4	4	SELECTED_ADMIN
4	5	MEMBER
4	6	MEMBER
4	9	ORIGINAL_ADMIN

This table shows the membership for all workspaces. To explicitly show that a user can also join or even create multiple workspaces, we set the creator of workspaces No.3 and No.4 to be the same user.

Table4.Channels

cid	wid	cname	ctype	ccreatetime
1	1	CS	PUBLIC	2019-01-04 12:00:00
2	1	CA	DIRECT	2019-01-04 12:00:00
3	1	MFE	PUBLIC	2019-01-04 12:00:00
4	1	EE	PUBLIC	2019-01-04 12:00:00
5	2	CE	DIRECT	2019-01-04 12:00:00
6	3	ME	PRIVATE	2019-01-04 12:00:00

There are 6 channels in total. However, only three of workspaces have channel, because workspace No.4 doesn't have any channel. This will happen in reality. For example, this workspace was just created and did not have channels yet.

Table5.CU

wid	cid	uid	cutype
1	1	1	CREATOR
1	1	2	MEMBER
1	1	3	MEMBER
1	2	4	CREATOR
1	2	5	MEMBER
1	3	6	CREATOR
1	3	7	MEMBER
1	4	1	CREATOR
1	4	2	MEMBER
2	5	7	CREATOR
2	5	8	MEMBER
3	6	2	MEMBER
3	6	3	CREATOR
3	6	9	MEMBER

This table shows the membership for all channels. In order to show enough results in the SQL queries No.4, we make the first workspace have the most channels and each of the rest of workspaces at least has one channel.

Table6.MessageContent

mid	uid	mtype	content
1	1	TEXT	perpendicular a
2	1	TEXT	a
3	2	TEXT	perpendicular b
4	3	TEXT	b
5	2	TEXT	c
6	2	TEXT	d
7	2	TEXT	perpendicular c
8	2	TEXT	e

There are 8 messages in total. In order to generate some results in the SQL queries No.6 and No.7, we let user No.2 posted the most messages and add key word “perpendicular” to some of the messages.

Table7.MessageFrom

mid	wid	cid	mtime
1	1	1	2019-04-07 12:19:41
2	1	1	2019-04-07 12:19:41
4	1	1	2019-04-07 12:19:41
8	1	1	2019-04-07 12:19:42
3	1	4	2019-04-07 12:19:42
5	1	4	2019-04-07 12:19:42
6	3	6	2019-04-07 12:19:42
7	3	6	2019-04-07 12:19:42

In order to get some results in the SQL queries No.5, we let most messages posted to the workspace No.1 and the Channel No.1

View1.WorkspacesInviteLog

	create_user	invited_user	wname	wptime	wstatus
1	Tom	Mike	NYU	2019-01-03 12:00:00	SENT
2	Tom	Mike	NYU	2019-01-03 12:00:01	ACCEPT
3	Tom	Lisa	NYU	2019-01-03 12:00:00	SENT
4	Tom	Lisa	NYU	2019-01-03 12:00:01	ACCEPT
5	Tom	Jerry	NYU	2019-01-03 12:00:00	SENT
6	Tom	Jerry	NYU	2019-01-03 12:00:01	ACCEPT
7	Tom	Alice	NYU	2019-01-03 12:00:00	SENT
8	Tom	Alice	NYU	2019-01-03 12:00:01	ACCEPT
9	Mike	James	NYU	2019-01-03 12:00:00	SENT
10	Mike	James	NYU	2019-01-03 12:00:01	ACCEPT
11	Mike	Lucy	NYU	2019-01-03 12:00:00	SENT
12	Mike	Lucy	NYU	2019-01-03 12:00:01	REFUSE
13	Lisa	Bob	NYU	2019-01-03 12:00:00	SENT
14	Lisa	Bob	NYU	2019-01-03 12:00:01	ACCEPT
15	Jerry	James	BU	2019-01-03 12:00:00	SENT
16	Jerry	James	BU	2019-01-03 12:00:01	ACCEPT
17	Alice	Lucy	CU	2019-01-03 12:00:00	SENT
18	Alice	Lucy	CU	2019-01-03 12:00:01	ACCEPT
19	May	Mike	CMU	2019-01-03 12:00:00	SENT
20	May	Mike	CMU	2019-01-03 12:00:01	ACCEPT
21	May	Lisa	CMU	2019-01-03 12:00:00	SENT
22	May	Lisa	CMU	2019-01-03 12:00:01	ACCEPT
23	May	Jerry	BU	2019-01-03 12:00:00	SENT
24	May	Jerry	BU	2019-01-03 12:00:01	ACCEPT
25	May	Bob	BU	2019-01-03 12:00:00	SENT
26	May	Bob	BU	2019-01-03 12:00:01	ACCEPT

This table shows all the invitations towards a particular workspace between different users. Note that only administrators of a workspace can invite others to join .

View2.ChannelsInviteLog

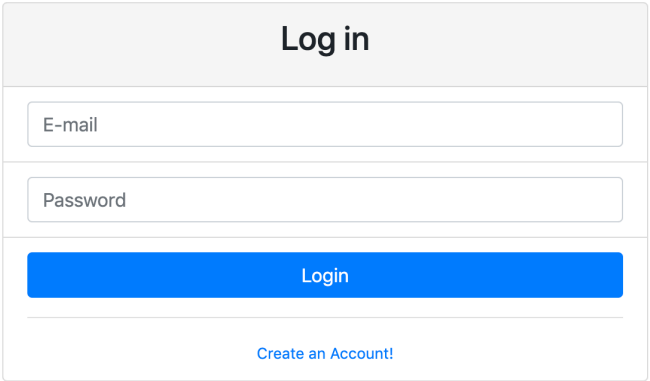
	create_user	invited_user	cname	cptime	cstatus
1	Tom	Mike	CS	2019-01-05 12:00:00	SENT
2	Tom	Mike	CS	2019-01-05 12:00:01	ACCEPT
3	Tom	Mike	EE	2019-01-05 12:00:00	SENT
4	Tom	Mike	EE	2019-01-05 12:00:01	ACCEPT
5	Tom	Lisa	CS	2019-01-05 12:00:00	SENT
6	Tom	Lisa	CS	2019-01-05 12:00:01	ACCEPT
7	Tom	Jerry	CS	2019-03-29 11:10:59	SENT
8	Tom	Jerry	CS	2019-03-31 13:01:00	SENT
9	Tom	Jerry	CS	2019-03-30 12:00:00	REFUSE
10	Tom	James	CS	2019-03-31 14:05:10	SENT
11	Lisa	Mike	ME	2019-01-05 12:00:00	SENT
12	Lisa	Mike	ME	2019-01-05 12:00:01	ACCEPT
13	Lisa	May	ME	2019-01-05 12:00:00	SENT
14	Lisa	May	ME	2019-01-05 12:00:01	ACCEPT
15	Jerry	James	CA	2019-01-05 12:00:00	SENT
16	Jerry	James	CA	2019-01-05 12:00:01	ACCEPT
17	Bob	James	MFE	2019-03-28 11:00:01	SENT
18	Bob	James	MFE	2019-03-30 08:00:01	SENT
19	Bob	James	MFE	2019-03-29 07:00:59	REFUSE
20	Bob	Alice	MFE	2019-01-05 12:00:00	SENT
21	Bob	Alice	MFE	2019-01-05 12:00:01	ACCEPT
22	Alice	Lucy	CE	2019-01-05 12:00:00	SENT
23	Alice	Lucy	CE	2019-01-05 12:00:01	ACCEPT

This table shows all the invitations towards a particular channel between different users. It has the most interesting test cases. We allow people to refuse and re-send the invitation. The invitations in the red boxes indicate the users that were invited to join the channel more than 5 days ago and that have not yet joined.

System Performance

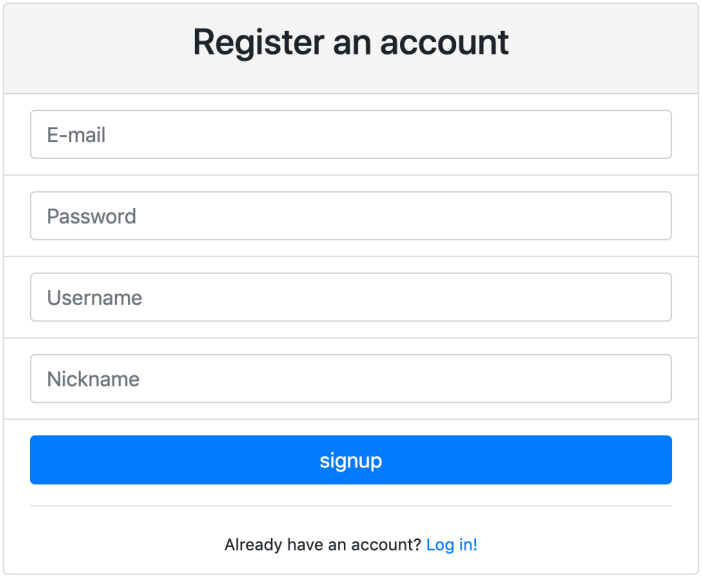
For project part two, we create a web-based application by using PHP, MySQL, and JavaScript and Bootstrap Library. The application accomplished most requirement of the system, and here is the explanation of each part of our application.

System Login:



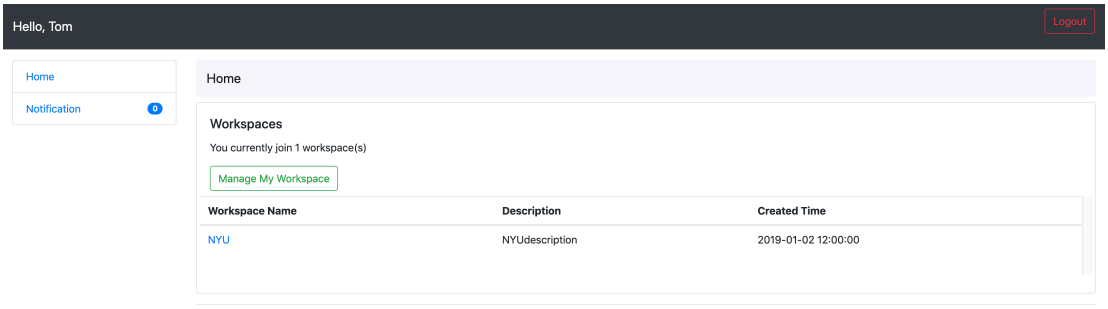
In this part, Users are required to enter their E-mail and password to login to the system. If Users do not have an account, Click the link below to the register page.

Register an account:



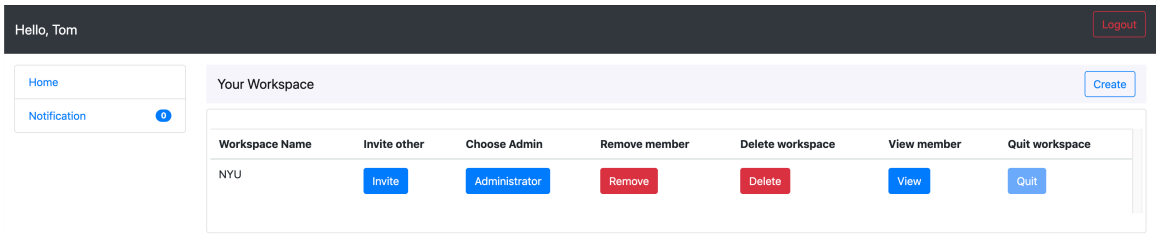
In this part, Users are required to set their E-mail address, password, username, nickname in the system to create an account. After all information are correctly set, users may login to the system by their email and password.

Dashboard:

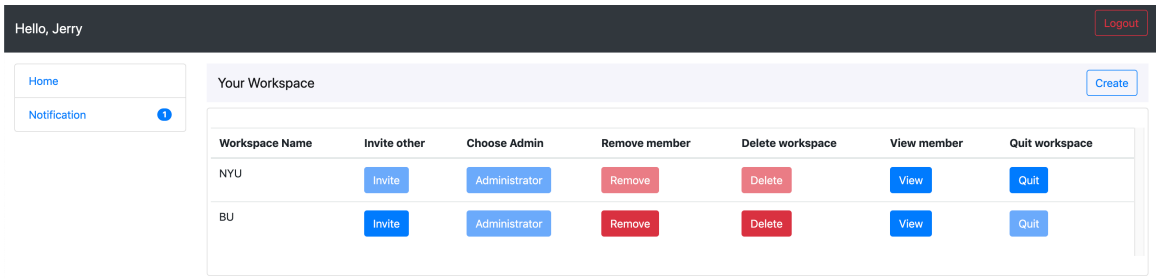


After login to the system successfully, Users get to their dashboard. In the dashboard, the workspaces that the user already in are showed. By click on the workspace's name, users can get channels' information in the workspace. By click on the Manage My Workspace button, Users can get the tool to manage their workspace. The dashboard also has link to Notification page and button to logout the system.

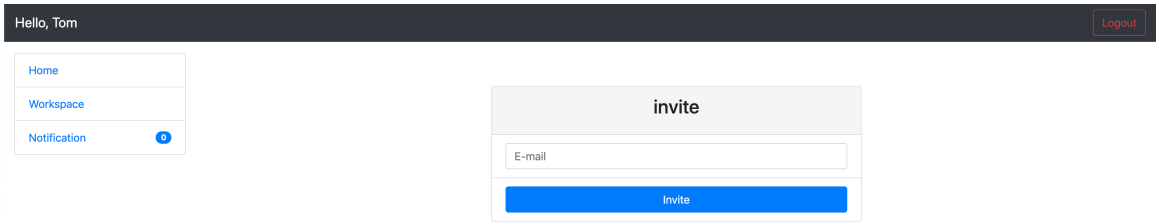
Workspace Management:



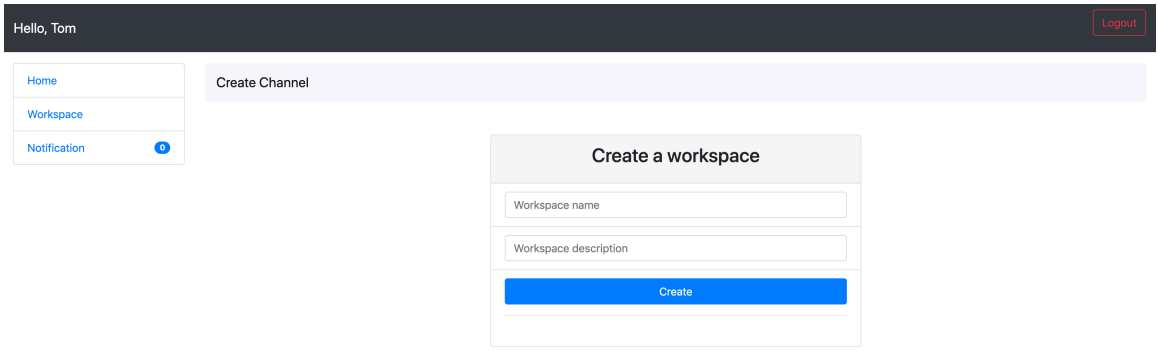
This page provides some management tools for users. If the user is administrator in the workspace, some high authority tools are available for him such as invite other people, set administrator, remove members, delete the workspace. However, if the user is just a member in the workspace, he can only view the member in the workspace and quit the workspace.



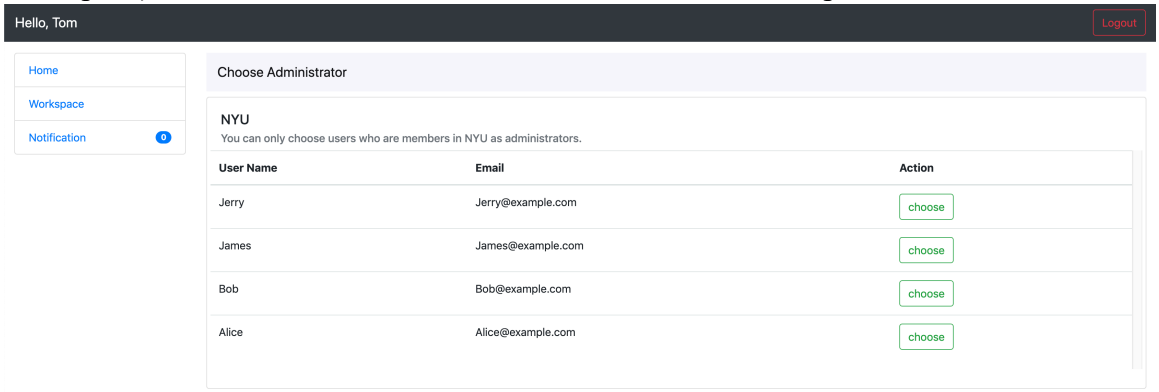
By clicking the invite button, administrator can invite other into the workspace by entering their email.



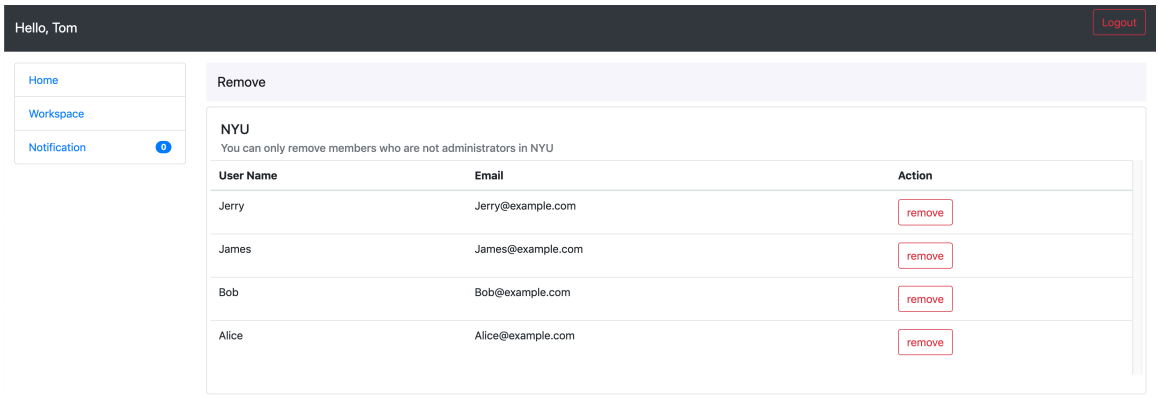
By clicking the create button, users can create workspace they want.



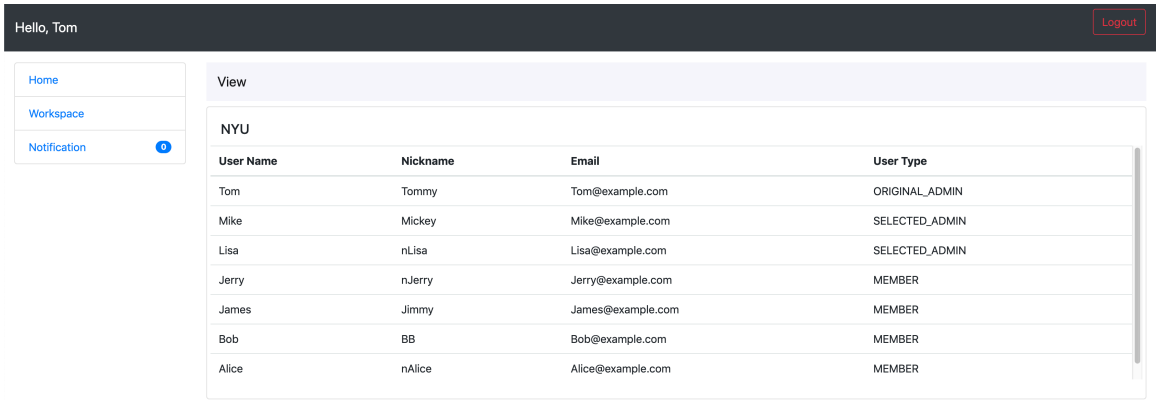
By clicking the Administrator button, the original administrator (the one creates the workspace) can set other members to administrator in the workspace.



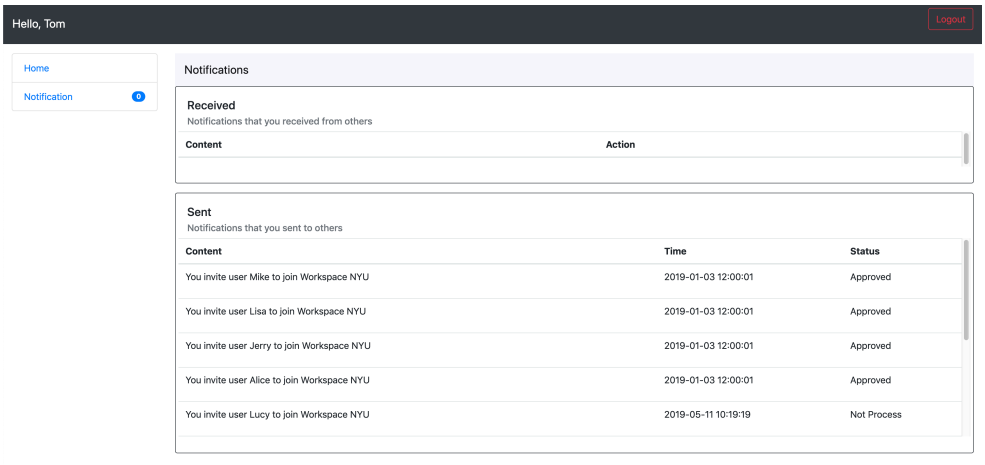
By clicking the remove button, the administrator can remove anybody in the workspace.



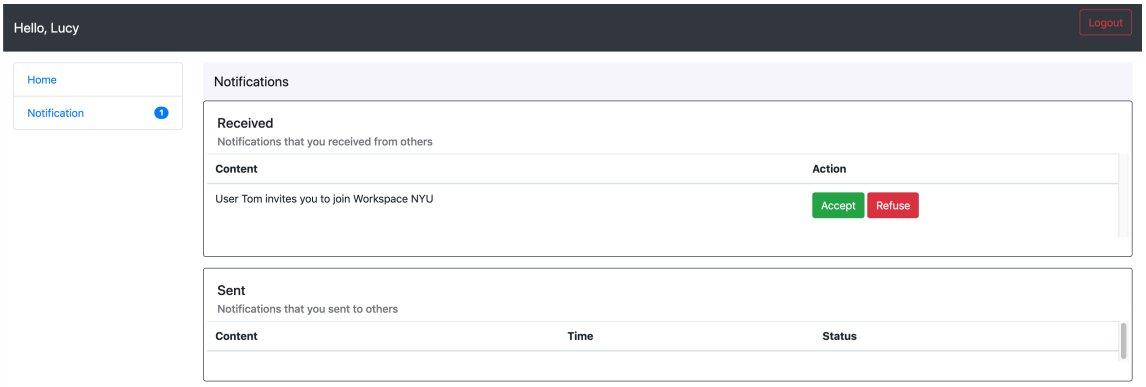
By clicking the view button, the member in the workspace can view all member in the workspace.



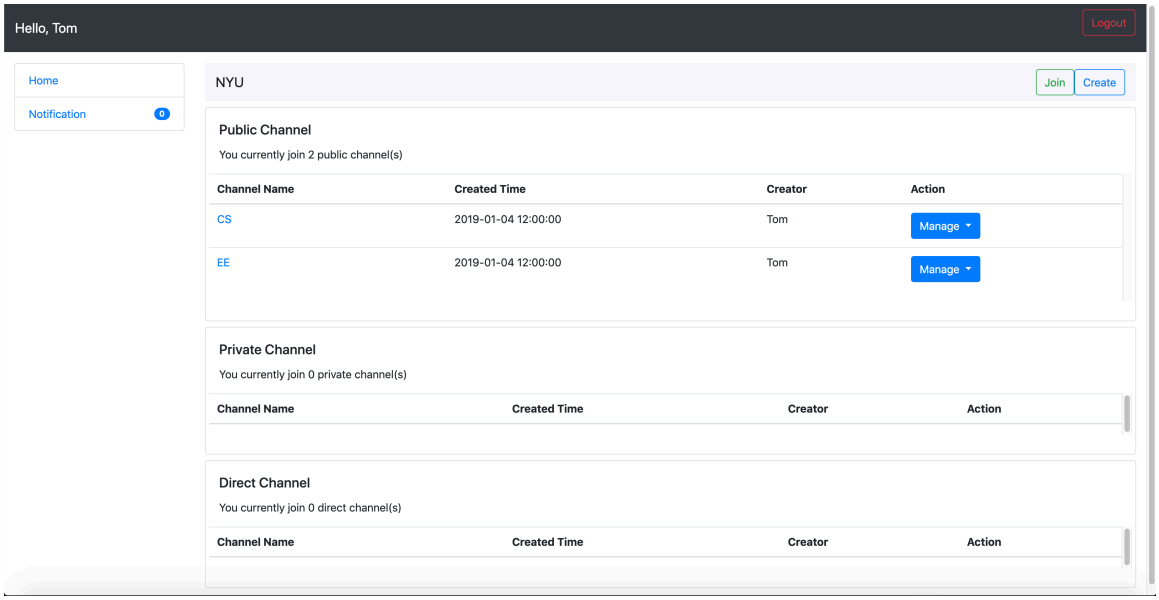
Notification:



In notification page, it will show the number of notifications waiting for process and the history of invitation record. For each invitation received, user can choose to accept or refuse the invitation.

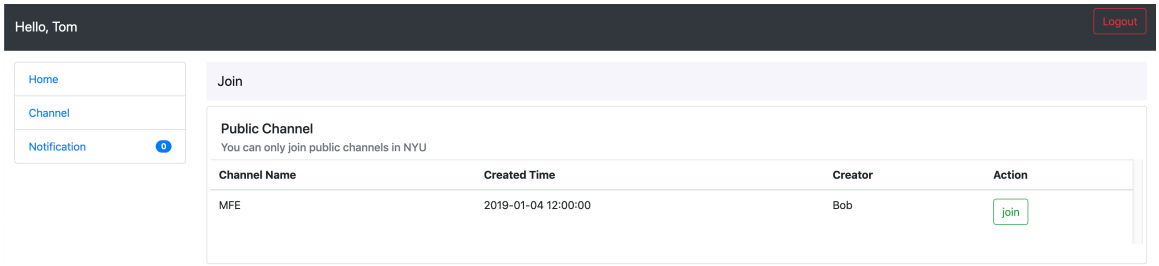


Channel Home:

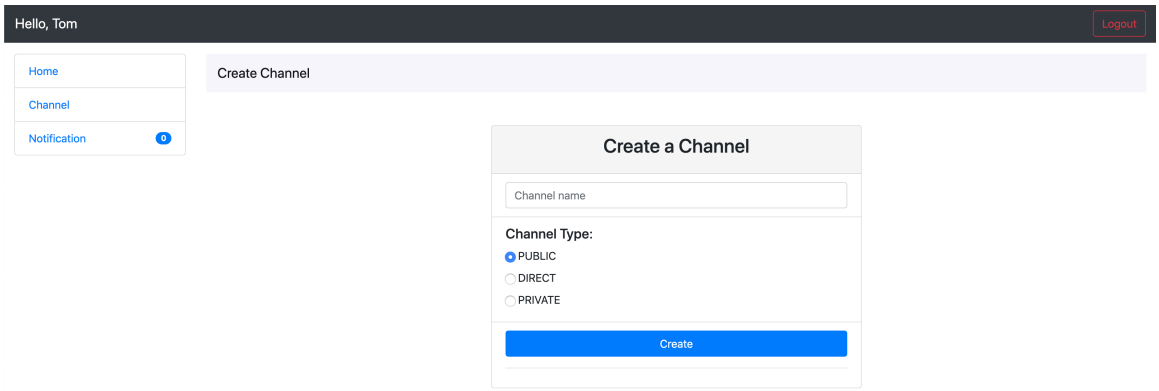


By clicking the workspace name in dashboard, Users get to the channel home page. In this page, users can view channels inside the workspace group by the type of channels and manage the channel when they are authorized to do so.

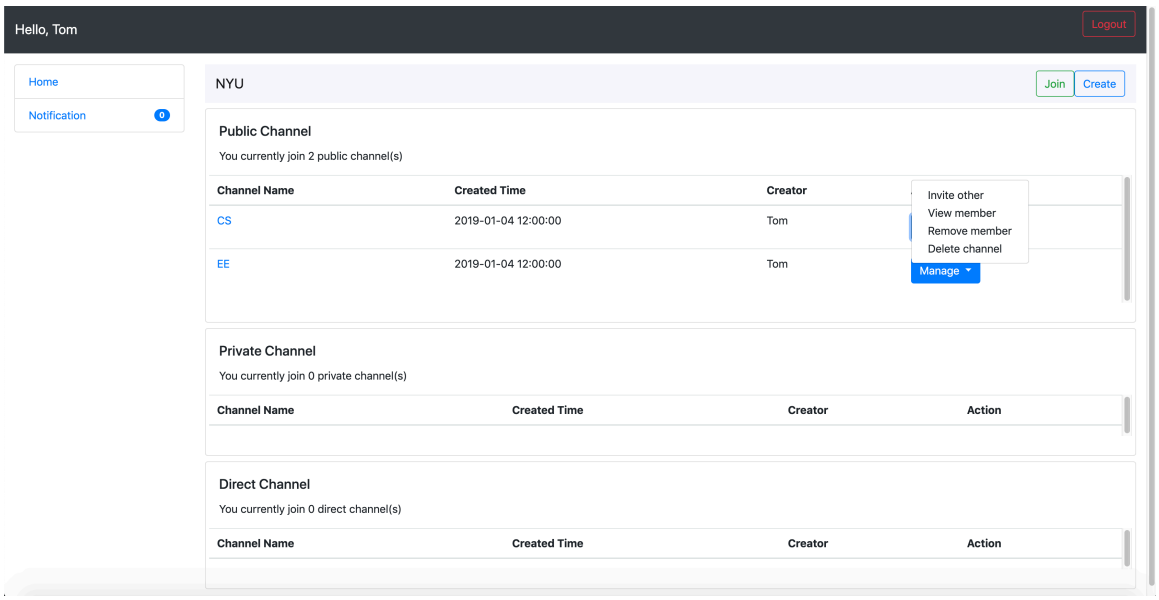
Channel Management:



By clicking the Join button, users can join any public channel in the workspace.

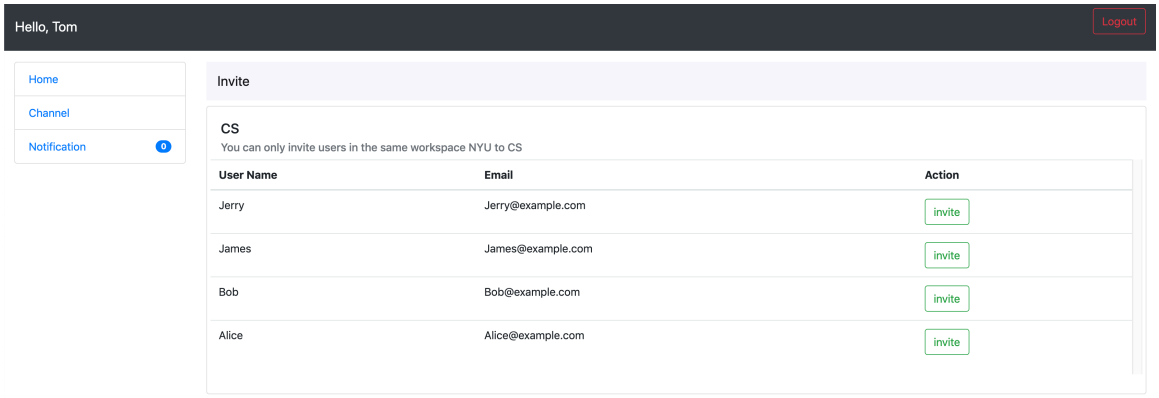


By clicking the create button, user can create channel inside the workspace.

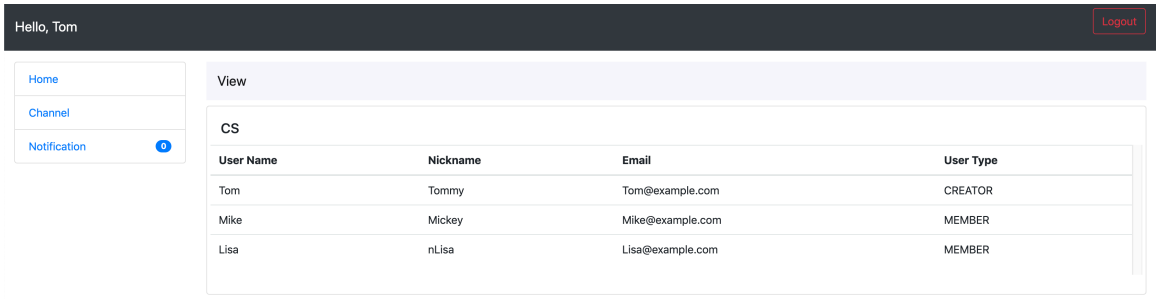


The page will give any available management option according to the role of users in the channel. Such as invite other people, view member in the channel, remove member from channel or delete the channel. What below are the screen shot of some main management option.

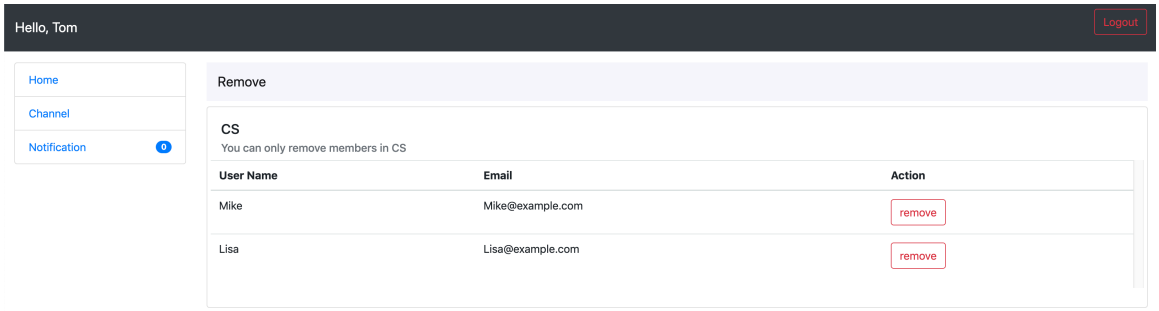
Invite other:



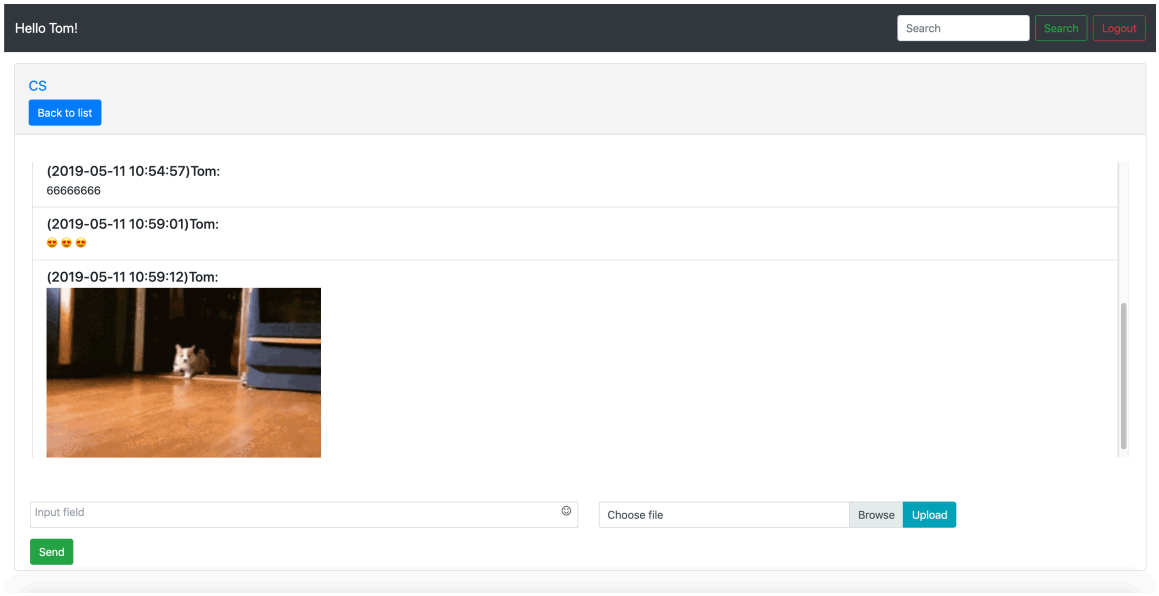
View channel member:



Remove member from channel:

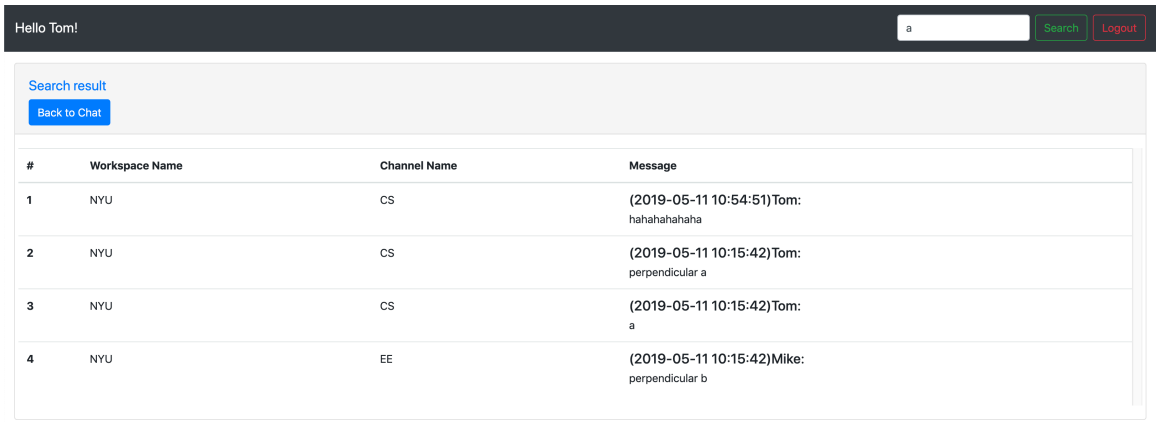


Chat:



By clicking the channel name, we enter the chatting page of the channel. The chatting page support text, emoji and image content. Each time someone in the channel send a message, the page will automatically refresh.

Search:



By entering search text and clicking the search button, users will get any available chatting record that contain the search text they just entered.

Some features in our design

Cookie

To avoid users typing their email and password every time when they used the system, we applied cookies and sessions to store the user's identification. In our implementation, we set the expired time to 1 hour. Thus within 1 hour, users only not need to type email and password once when they enter the system.

```php
if (sizeof($result) == 1)://登陆成功
    $userid = $result[0]['uid'];
    session_start();
    $lifeTime = 1 * 3600;
    setcookie(session_name(), session_id(), time() + $lifeTime, "/");
    $_SESSION['username'] = $userid;
    header("location: dashboard.php?admin=$userid");
    exit;
```

Against SQL injection

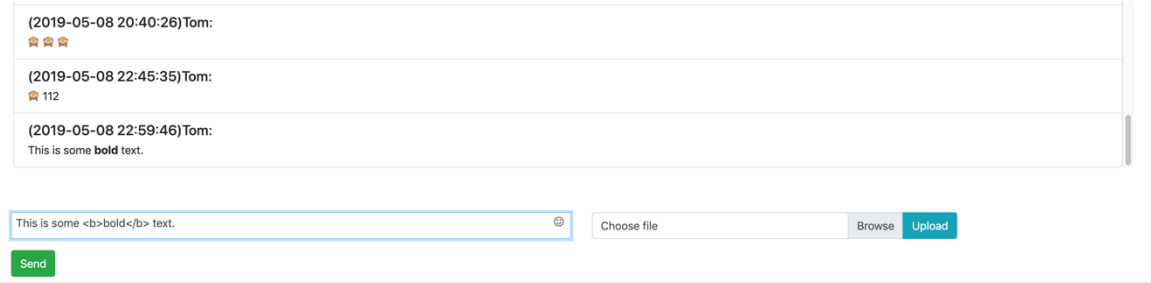
To avoid SQL injection, we do check each time when users input something. To implementing this, we used PDO connection to connect front end and MySQL and used bindParam to bind variables to SQL queries instead of directly connecting variable and SQL queries.

A simple example in our project:

```php
    $stmt = $conn->prepare("select uid from Users where email = ? and password = ?");
    $stmt->bindParam(1, $email, PDO::PARAM_STR, 32);
    $stmt->bindParam(2, $password, PDO::PARAM_STR, 32);
```

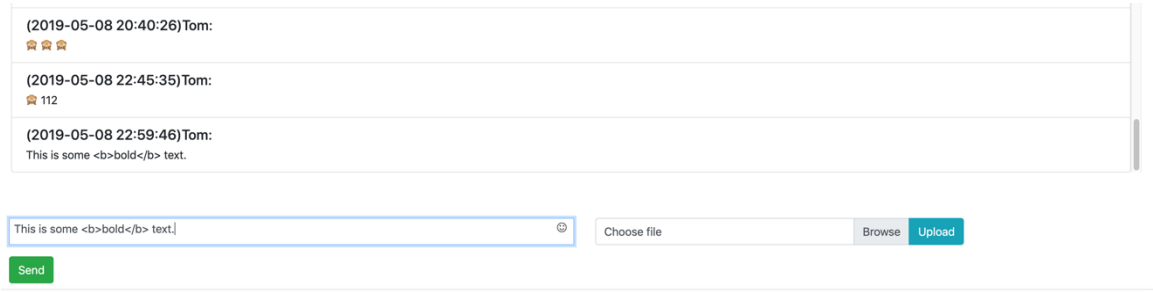
Against cross-site scripting

To against cross-site scripting, we sanitized the outputs to be returned to user's browsers by using htmlspecialchars. To illustrate this, we do some test like below:
Before protection：



We can see that input that contain script is execute directly.

After protection：



By doing transformation, the script cannot be executed.