Group J

WhiteBoard

Software Design Report for a Active Teaming Application

Version <2.0>

Linda Li, Yifeng Jin, Masuda S. Farehia

Table of Contents

Software Design Report

1. Introduction

This section illustrates the overall picture of the system with the use of the Collaboration Class Diagram (CCD). The report includes collaboration class diagram, petri net and the ER diagram which are meant to provide the data structure and logic to carry out the functionalities. There is a collaboration for each user given by each edge which defines the condition for that specific condition to give a brief understanding

Collaboration Class Diagrams

Link for Collaboration Diagram:

https://github.com/Masuda50/csc322/blob/master/collaboration%20diagram1.png

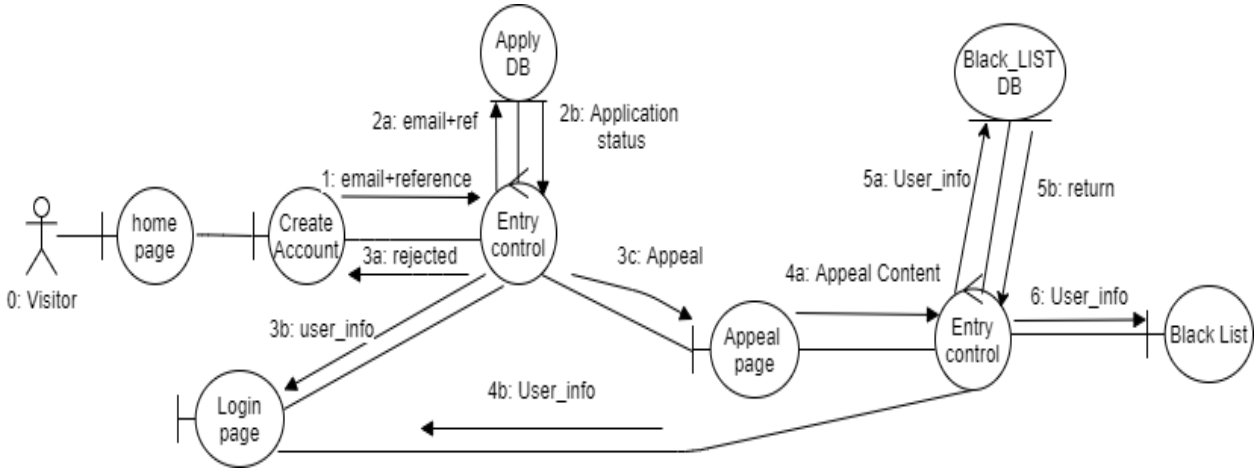
2. Use Cases Scenarios

This section includes normal and exceptional scenarios for each use-case. Each use case will include a collaboration class diagram as well as a petri net which will help visualize the functionalities of each use case.

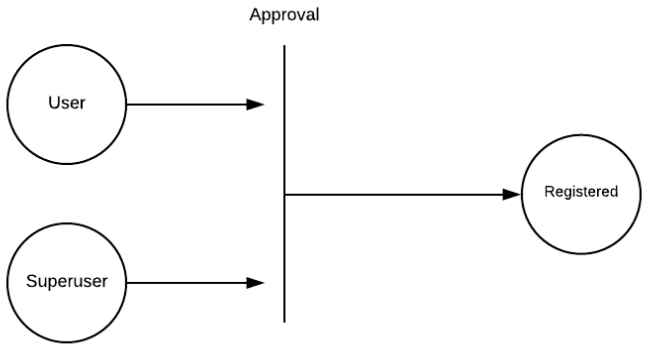
Use Case 1: User Registration

- If a visitor is not a registered user, register by inputting required information which will be stored in the Apply DB. An unregistered visitor will only be allowed to access guest privileges otherwise.
- In the case where they register for an account, the user must wait for a verification from the superuser to notify whether the application has been approved or denied.

Collaboration Class



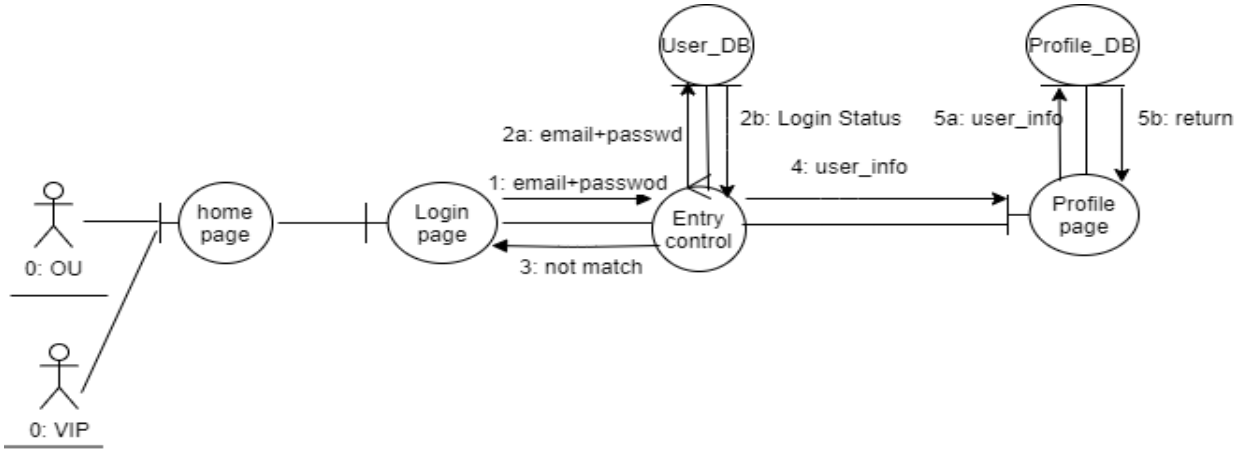
Petri-Net



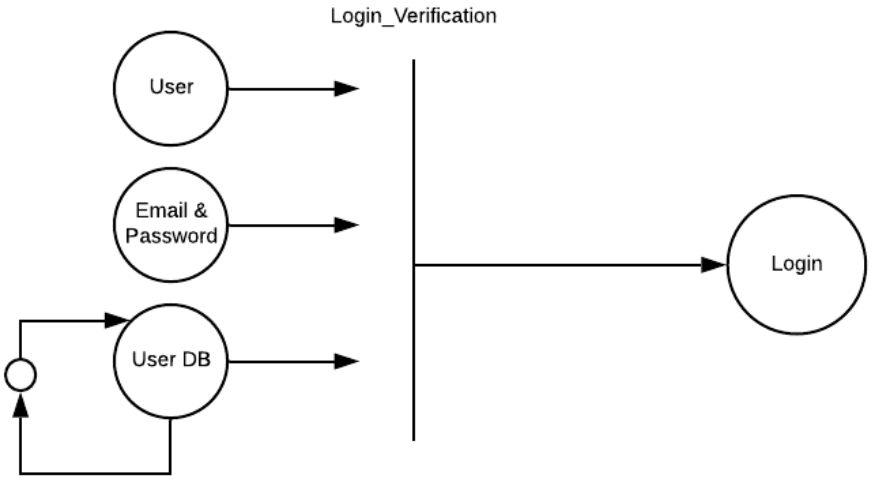
Use Case 2: Login

- If the user is not a registered user, they will have the option to create one
- If a user has an account, navigate to the login page and enter the validated email and password where information will be confirmed based on User_DB. If login status from User_DB confirms the user, the user is directed to their account profile page. If the login status does not confirm the user, it will return not matched.

Collaboration Class



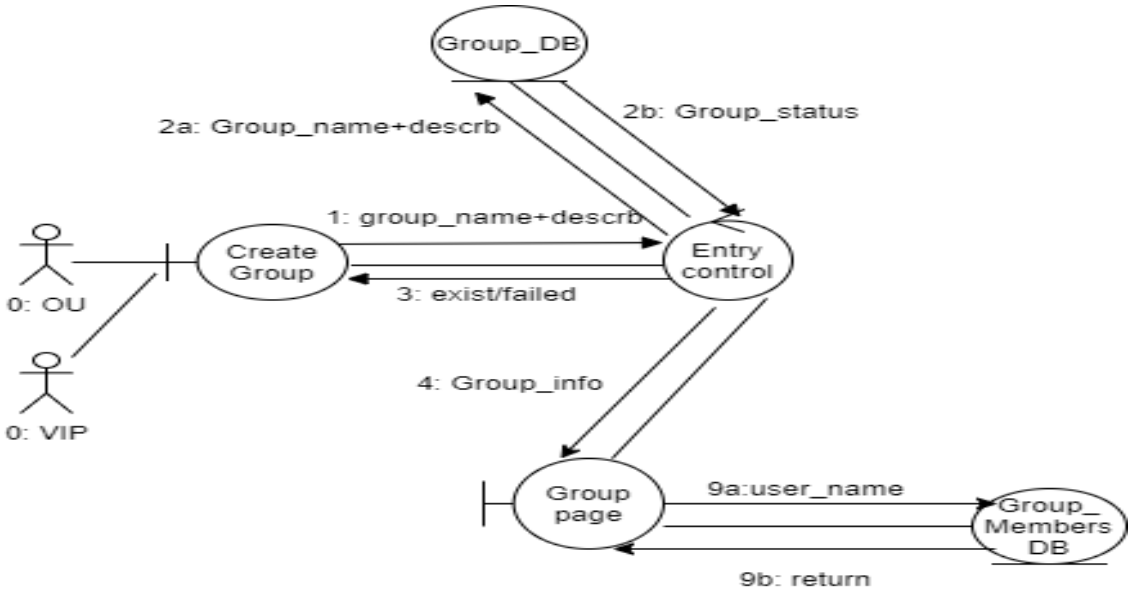
Petri-Net



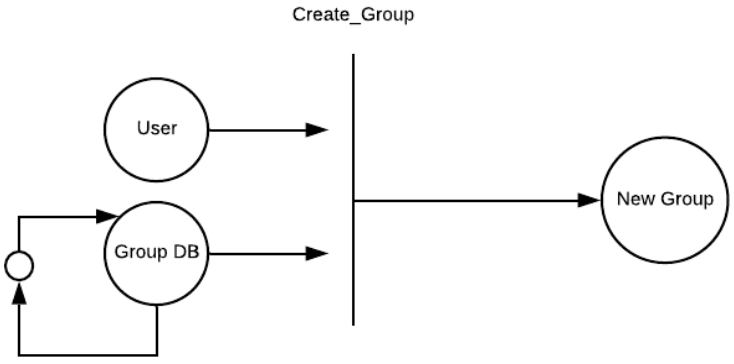
Use Case 3: Create Group

- If the group name does not yet exist, the user is allowed to create group after entering group name, group members, and description which will be stored in the Group_DB
- If the group name already exists in Group_DB the group status will return that the name exists and the user will be prompted to enter another group name.

Collaboration Class



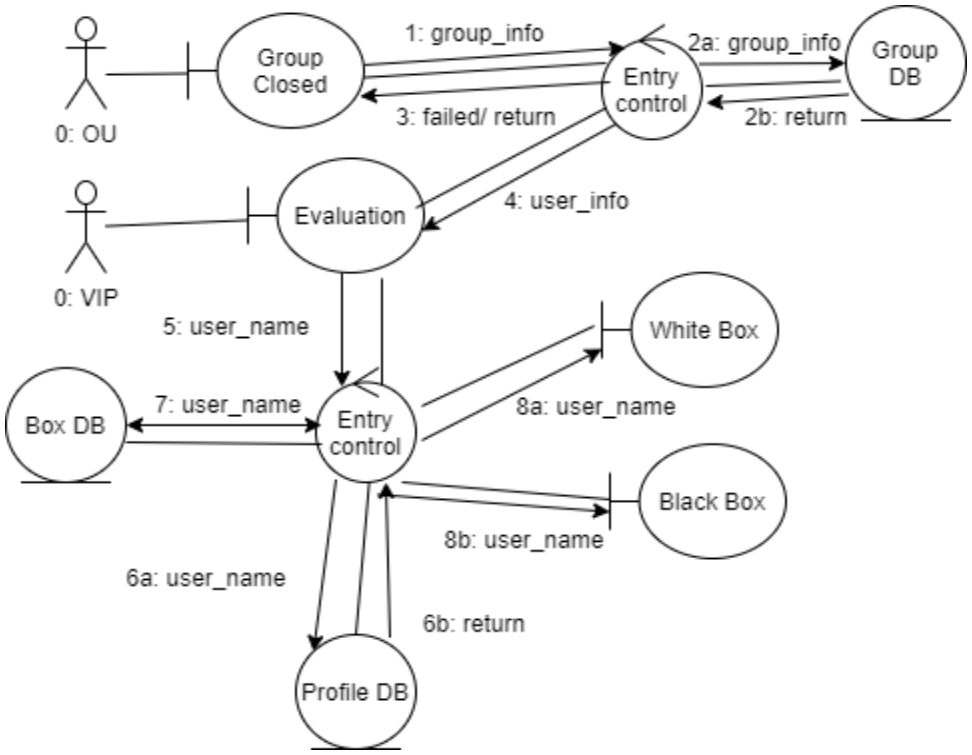
Petri-Net



Use Case 4: Group Evaluation

- When a group is closed, all members are allowed to evaluate other members and decide them into a white box or a black box. The members' info will be stored in Box DB. Also, a VIP will be assigned to evaluate the group and determine a reputation score to be added or deducted from each group member's individual reputation score. The members' info will be stored in Profile DB.

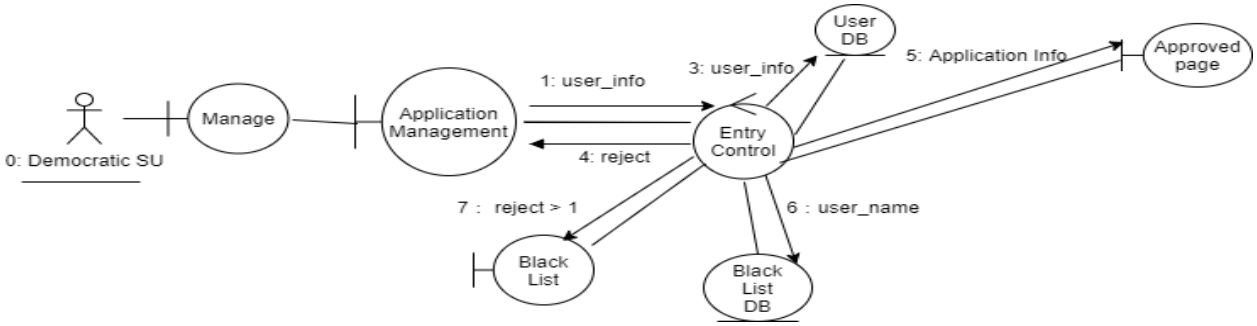
Collaboration Class



Use Case 5: Application Management

- If the Superuser (SU) accepts the application, the user information is stored in the User_DB then the user will receive an email of approval.
- If the SU rejected the application, the visitor can appeal to the SU. If the application is still rejected the applied user-info will be stored in Blacklist DB.

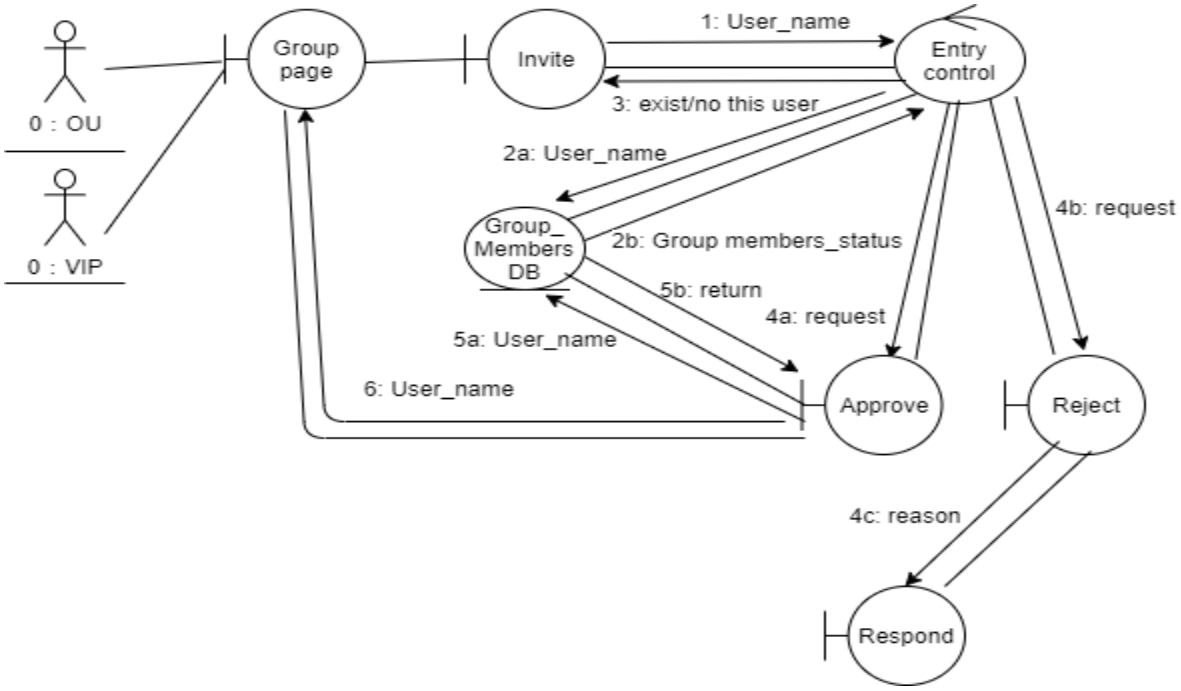
Collaboration Class



Use Case 6: Invite

- When a user creates a group, they are allowed to invite other users. If the invited user accepts the invitation to join the group the user is stored in Group_Members DB.
- If the invited user rejects the invitation request they must respond with a reason why.

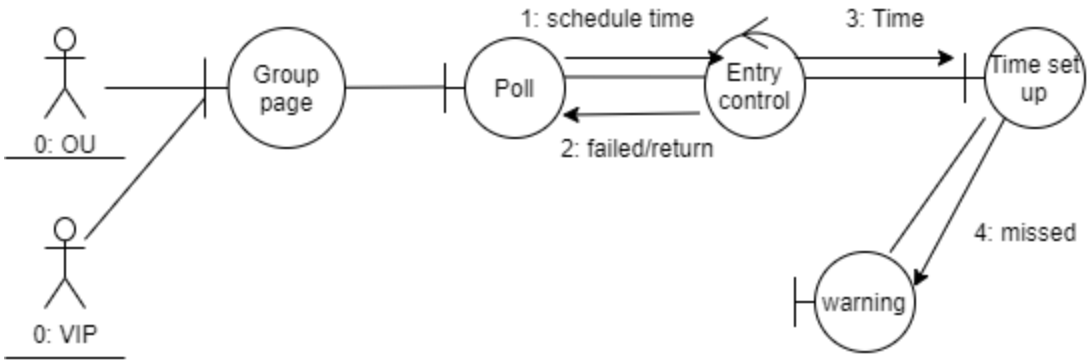
Collaboration Class



Use Case 7: Poll

- In the group page, the members are allowed to create polls to schedule a time for meetup.
- If the majority of the group members agree on a time, the meeting is set to that time. If not then the group needs to create a new poll.
- If a group member misses a schedule meeting two times the member will receive a warning.

Collaboration Class

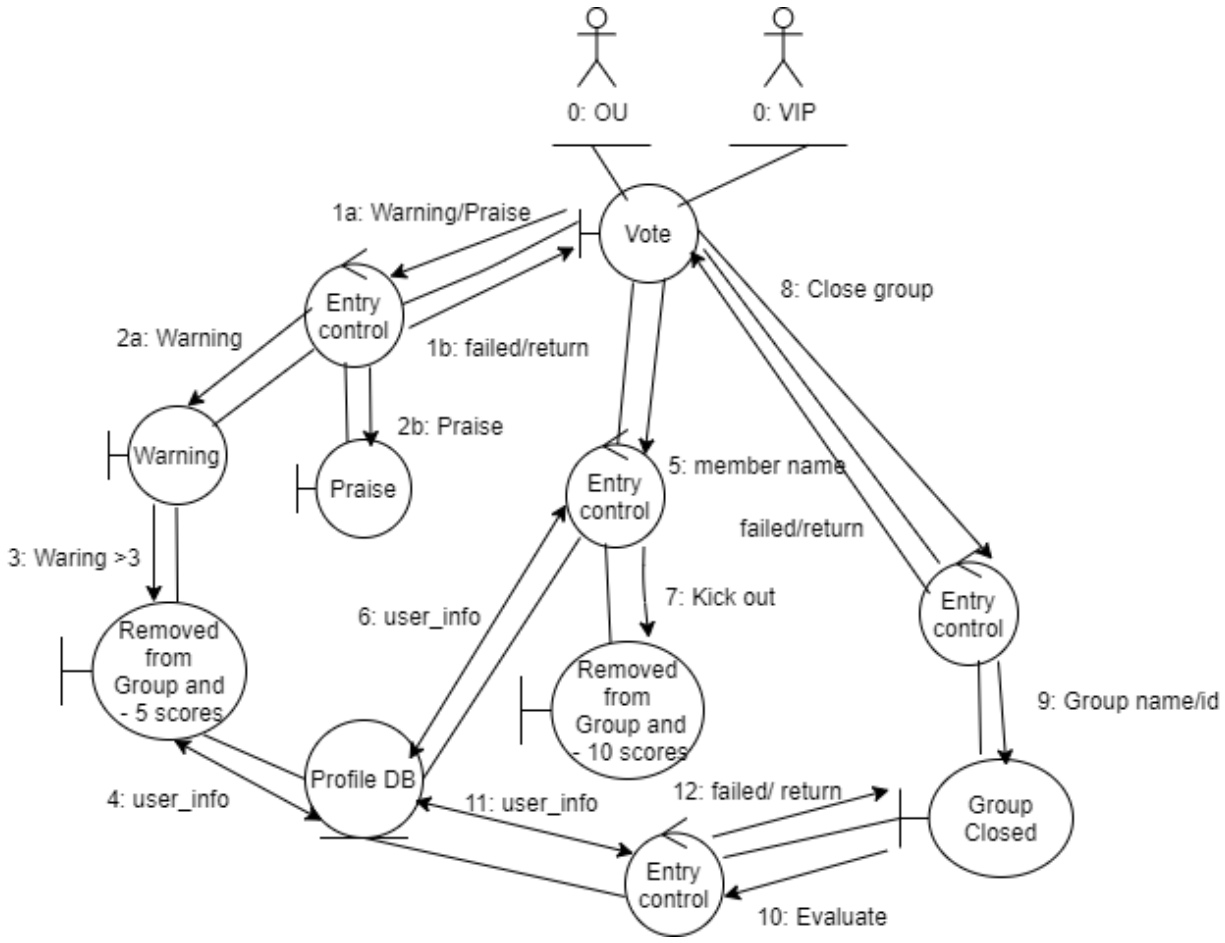


Use Case 8: Vote:

- In the group page, the members are allowed to vote to issue a warning or a praise to a group member, or vote to kick out a group member, or close the group. The vote must be unanimous.
- If a member receives 3 warnings will be automatically removed from the group and get a 5 point reputation score deduction and the member's info will be stored in Profile DB.
- If a member is voted out of the group by other members, the member will be removed from the group and receives a 10 point reputation score deduction and the member's info will be stored in Profile DB and will be removed in Group_Members DB.
- If all members vote to close the group, the group will close. Each member will receive the median reputation score given by all other members. And every member can decide if

s/he is willing to put the other member to her/his white-box or black-box afterwards and why. The group's info will be removed in Group DB and all members' info will be stored in Profile DB and Box DB.

Collaboration Class

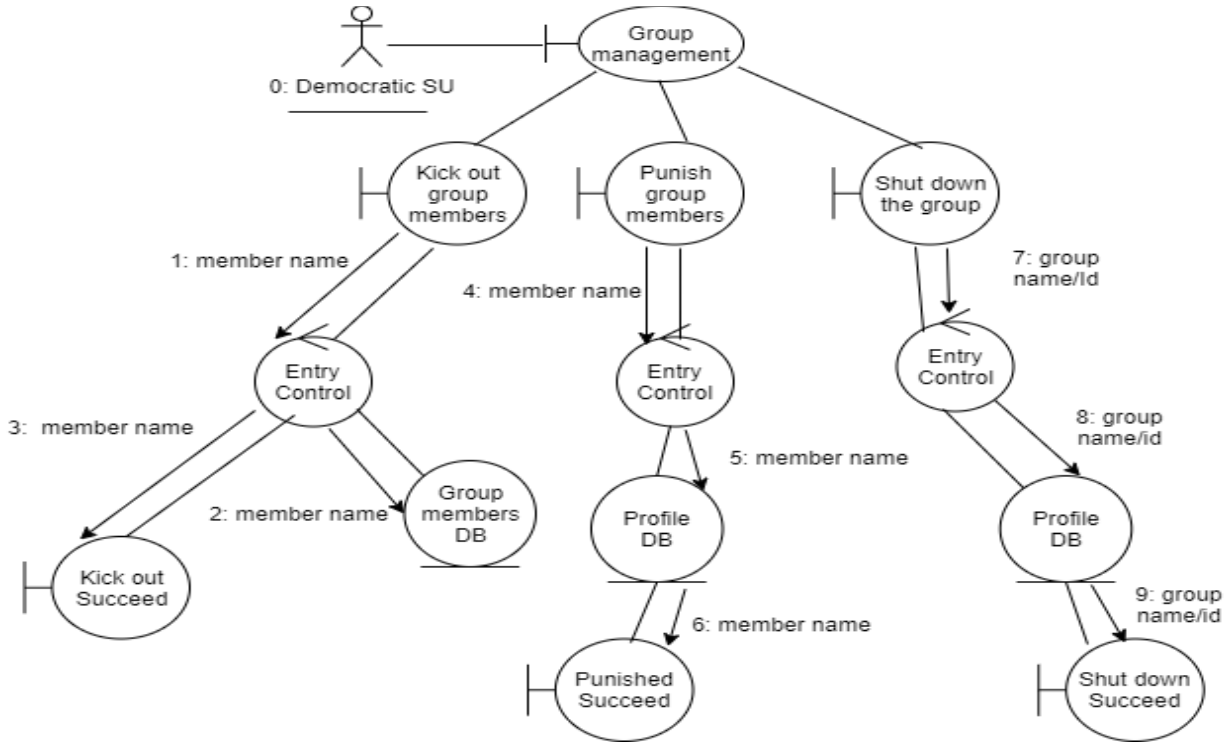


Use Case 9: Group Management

- In the group management page, the SU are allowed to kick out group members or punish group members or shut down the group if the SU received a complaint from other users.
- If the SU kicks out the member from a group, the member's info will be removed in Group_Member DB.

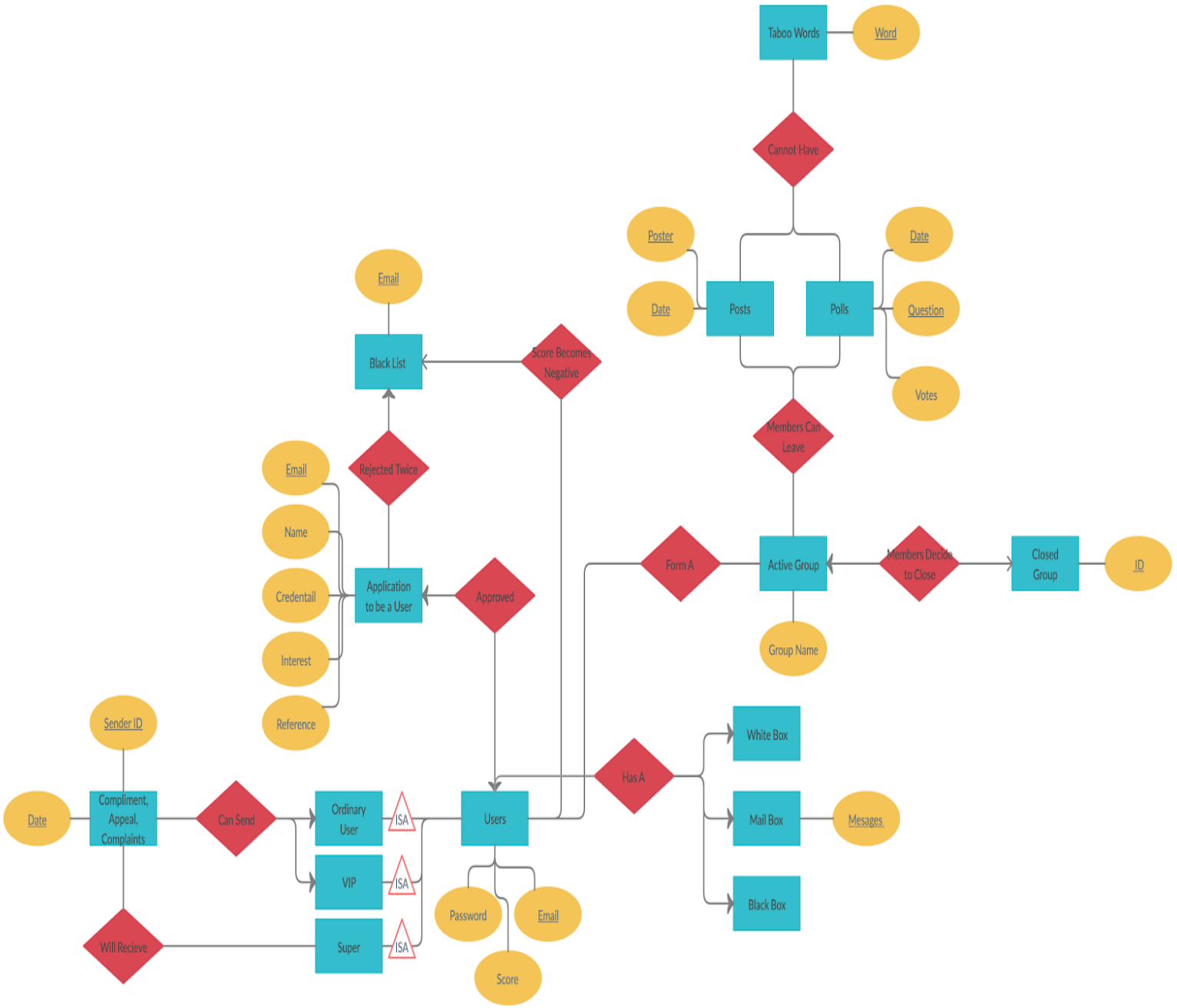
- If the SU punishes all involved by a certain score deduction, these members' info will be stored in Profile DB.
- If the SU shut down the group, the group will be closed and will be removed in Group DB.

Collaboration Class



3. E-R Diagram

The diagram below is an E-R Diagram that illustrates the overall database system and their interactions.



4. Detailed Design

Every method in pseudo-code to delineate the input/output and main functionalities.

#admin approving and rejecting accounts <admin is SUPER USER>

@app.route("/admin")

Input :: tb_applied information

```
Def admin():
        #approve a user
        If request.method = "POST":
                User info = Store the user's information from the post method
                If approve in request form:
                        If (user_info not in tb_user):
                                Cursor.Execute(insert user_info tb_user)
                                Send email
                        Else:
                        Cursor.Execute(delete user_info from tb_applied)
                        Mysql.commit():
        #reject a user
        If reject in request form:
                        Cursor.execute(delete user_info from tb_applied)
                        Mysql.commit()
                        Send email
        # load the admin page
        Applied = fetch all info from tb_applied
        if Applied:
                        return render_template('admin.html', applied=applied)
                return render_template('admin.html')
```

Output:: Users added to tb_user or deleted from system all together

---

```
# home page that shows public posts
@app.route("/")
```
Input :: tb_post information
```
def home():
        cursor.exectute(get all post and users associated with post)
         post = cursor.fetchall()
         if post:
        return render_template('index.html', post=post)
                return render_template('index.html')
```
Output :: post information is rendered on website

```
#  replies to public posts rendered
@app.route('/reply/<post_id>/')
@login_required
```
Input :: tb_post information
```
def into_reply(post_id):
        cursor.execute(all post from tb_post and their users)
        posted = cursor.fetchone()
        cursor.execute(join table reply and table user to get reply information)
         reply = cursor.fetchall()
         session['post_id'] = posted['post_id']
         return render_template('reply.html', posted=posted, reply=reply)
```
Output: all posts are linked to their replies

```
# reply feature
@app.route('/add_reply/')
```
Input :: a post
```
def add_reply():
```

```
        reply_content = request.form['reply_message]
        if not reply_content:
                flash('Please fill out the form!')
                return redirect(post being replied to)
   else:
                cursor.execute(insert data into table reply: user_id, reply_content, post_id)
                MYSQL.commit()
                return redirect(url_for(add reply)
```

Output: New reply added in connection to a post into the db

```
#login
@app.route('/login/')
```

Input: Information gathered from the login form

```
def login():
        if request.method == 'POST' and 'email' in request.form and 'password' in request.form:
        cursor.execute( Select email, password from tb_users)
                        account = cursor.fetchone()
                #admin login
                        if email and password == 'admin':
                        return redirect(url_for('admin'))
                elif account:
                # Create session data, we can access this data in other routes
                cursor.execute(get profile associated with this user)
                return redirect(profile_page)
         else:
                # Account doesn't exist or username/password incorrect
                msg = 'Incorrect email/password!'
                        # Show the login form with message (if any)
                        Return render_template('login.html', msg=msg)
```

Output: Profile on the user or an error message of a non-existent user

---

```
# Registration page
@app.route('/register/')
```

Input: Information gathered from the registration form

```
def register():
if request.method == 'POST' and has all the required inputs
        Form_info = request.form[form_info]  # get data from url
        cursor.execute(check if email already exist in tb_user or tb_applied)
                        account = cursor.fetchone()
                        if account:
                                msg = 'Invalid Email!'
                        elif not a proper email:
                                 msg = 'Invalid email address!'
                        elif not a proper username, interest, credential, reference:
                                msg = x must contain only characters!'
                        elif form empty:
                                msg = 'Please fill out the form!'
                       else:
                             cursor.execute("INSERT INTO tb_applied )
                MYSQL.commit()
                msg = 'You have successfully applied! Look for an email containing your
username and password'
        return render_template('login.html', msg=msg)
         elif request.method == 'POST':
                        msg = 'Please fill out the form!'
          return render_template('register.html', msg=msg)
```

Output: Column added to tb_applied if registration successful

---

```
# profile page
@app.route('/profile/myProfile')
```
Input: The user ID
```
def profile():
        if 'loggedin' in session:
        cursor.execute(all the account info for the user from tb_user and tb_profile)
        account = cursor.fetchone()
        cursor.execute(get user post information)
                        post_history = cursor.fetchall()
                        cursor.execute(get all group information related to the user)
                        group_info = cursor.fetchall()
                return render_template('profile.html', account=account,
                post_history=post_history, group_info=group_info)
          return redirect(url_for('login'))
```
Output: The users profile page with self information and group information

---

```
#logout page
@app.route('/logout/')
```
Input: Session data
```
def logout():
        # Remove session data
        session.pop('user_id', None)
        return redirect(url_for('login'))
```
Output: Login page with no session information

```
#post creation
@app.route('/post/')
@login_required
def post():
```

Input: post title and content by user

if request.method has username", "password" and "email" POST requests exist:

cursor.execute(Check if title exists using MySQL)

post = cursor.fetchone()

if post:

msg = 'Error: Title already exists!\n'

else:

cursor.execute(now insert new post into accounts table)

MYSQL.commit()

return redirect(('home'))

elif request.method == 'POST' and Form is empty:

msg = 'Please fill out the form!'

return render_template('post.html', msg=msg)

Output: New post inserted into database should it be valid

---

create a group

@app.route('/group/')

def create_group():

Input: User request to make group with name

if request.method == "POST":

group_info = request.form['group_info']

cursor.execute(check if the group name entered by the user already exist)

group_name_exist = cursor.fetchone()

if group_name_exist:

flash('Group Already Exist')

return redirect('profile')

cursor.execute(insert data into table group: group_name, user_id, group_describe)

MYSQL.commit()

cursor.execute(get the group id)

```
                group_id = cursor.fetchone()
                #add user to group member
                 cursor.execute(insert data into table group_members: group_id and user_name)
        mysql.connection.commit()
                return redirect(profile)
```

Output: New group in db with the creator in its members

```
# group page
@app.route("/into_group/<group_id>")
```

Input: Group Id

```
def into_group(group_id):
        cursor.execute(get the group's information: group_name, group_describe, group_id,
        group_created_time, group_creater if exist)
                group = cursor.fetchone()
                cursor.execute(get the group members' information: user_name, user_id)
        group_members = cursor.fetchall()
                return render_template('group.html', group=group,
group_members=group_members)
```

Output: Page with all group related post and members

```
# invite feature
@app.route("/invite/<group_id>")
```

Input: Name of user that would like to be added to the group

```
def invite(group_id):
        if request.method == 'POST':
                user_name = request.form['user_name']
                 cursor.execute(check the user name the user inputted if exist in user database)
                 user_name_exist = cursor.fetchone()
                if not user_name_exist:
```

```
                flash("User doesn't exist")
                return redirect(group_id)
            cursor.execute(otherwise, check the username if exist in this group)
            group_member = cursor.fetchall()
        if group_member:
                flash('This User Already in this Group')
                return redirect(group_id)
            cursor.execute(otherwise, insert data into table group_pending)
         MYSQL.commit()
        return redirect(group_id)
```

Output: User send invitation to be in the group

```
#poll creation
@app.route('/poll/')
@login_required
def poll():
```

Input: poll title and content by user

```
    if request.method has username", "password" and "email" POST requests exist:
                content = request.form['content']
                cursor.execute(Check if title exists using MySQL)
                poll = cursor.fetchone()
        if poll:
                msg = 'Error: Title already exists!\n'
         else:
                cursor.execute(now insert new poll into accounts table)
                MYSQL.commit()
        return redirect(('home'))
        elif request.method == 'POST' and Form is empty:
                 msg = 'Please fill out the form!'
```

return render_template('post.html', msg=msg)

Output: New poll inserted into database should it be valid

#delete group

@app.route('/<group-id>/')

@login_required

def delete_group():

Input: delete group request

if request.method has “delete group”:

group_evaluation()

cursor.execute(delete the group from group_tb and all its members)

MYSQL.commit()

return redirect(('home'))

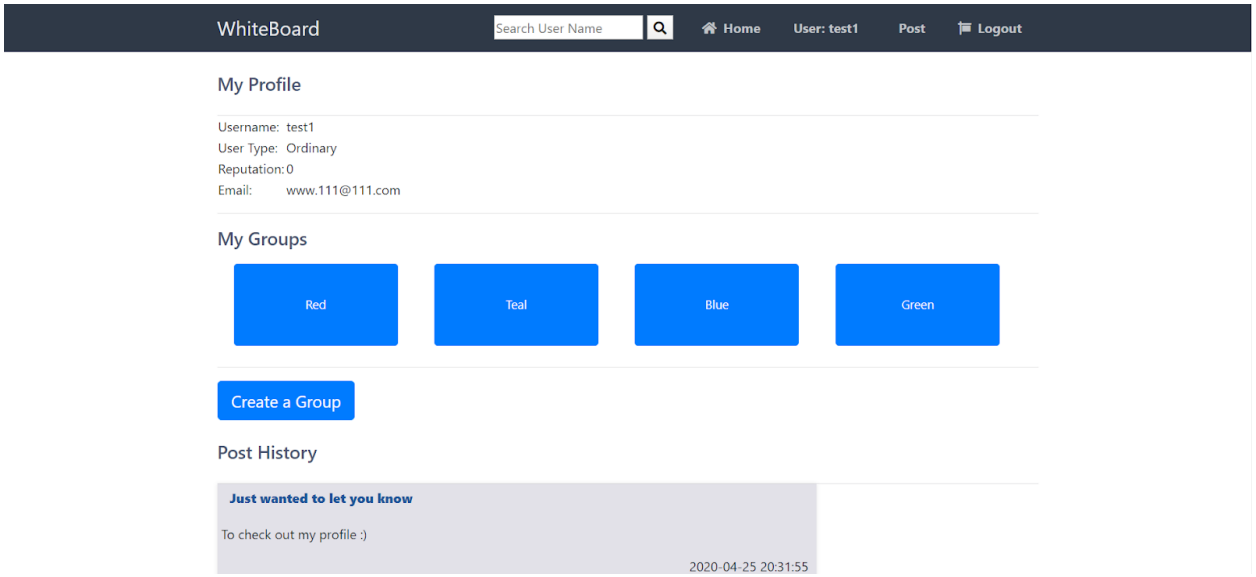
Output: group is taken out of database

---

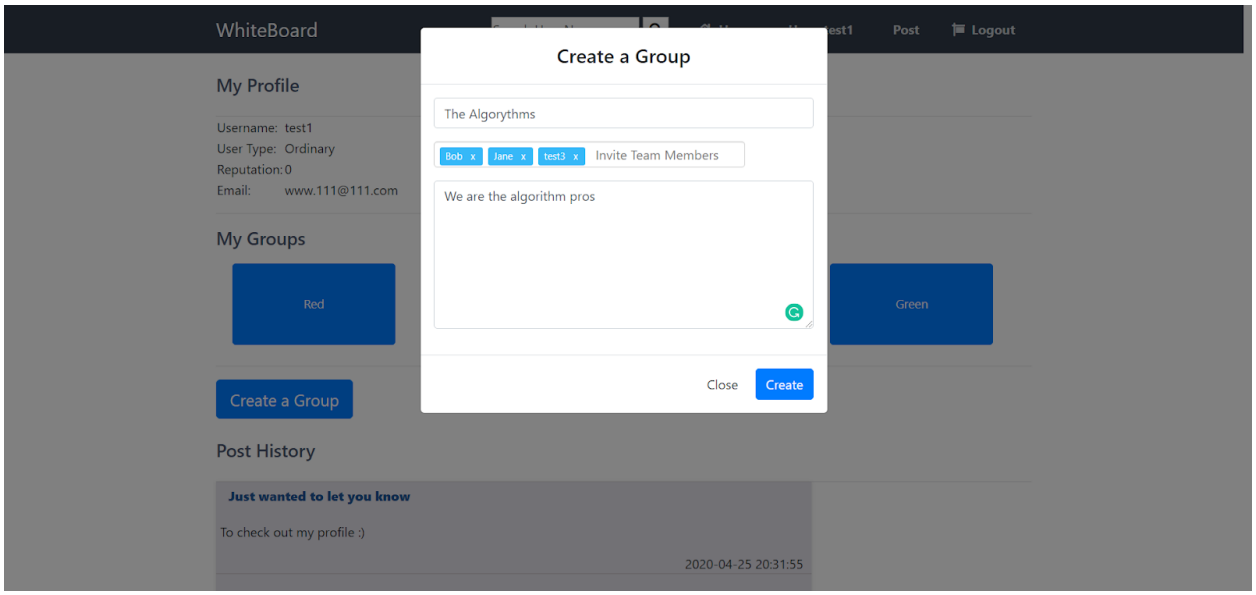
5. System Screens

Prototype for the system's User Profile Page and Create Group Functionality

Profile



Create Group



6. Group Meetings

Group Meeting 1:

Data & Time: March 12 @ 3:00P.M

Type: In-Person

Attendees: All members

Description: Group Introduction, Assigned each team members part for Phase 1: Specification report requirements

Group Meeting 2:

Data & Time: March 28 @ 3:00P.M

Type: Zoom

Attendees: All members

Description: Discussed project specifications, distributed task for each team member.

Group Meeting 3:

Data & Time: April 11 @ 5:00P.M

Type: Zoom

Attendees: All members

Description: Team showcased the task they worked on and discussed other feature specification that needs to be included.

Group Meeting 4:

Data & Time: April 18 @ 6:30P.M

Type: Zoom

Attendees: All members

Description: Distributed task for design report, showcased features worked on up to date.

7. Git Repository

Link: https://github.com/Masuda50/csc322