CSC 32200 <Team E>

<Active Teaming System>

Phase II Design Report
For Computer Application

Version <1.0>

Revision History

Date	**Version**	**Description**	**Author**
<04/22/2020>	<1.0>	The first version of the Active Teaming System.	Homin Lee, Bida Chen, Jose Vargas.

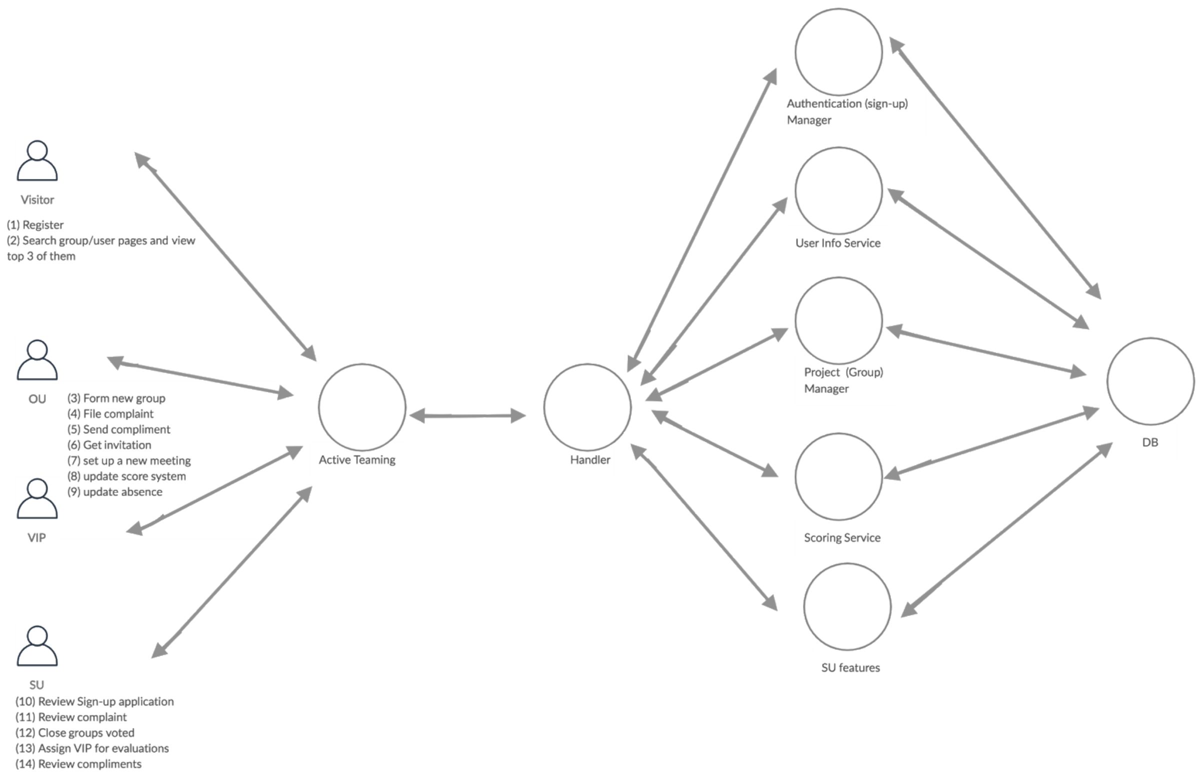
Table of Contents

1. Introduction

Active Teaming is a user-friendly teaming platform that helps people who are interested in similar projects with proper skill sets for that. This document will present the details of the design and rationale behind the development of the application along with the overview of our software and various other technical dependencies.

1.1. Collaboration Class Diagram of System.

Active Teaming application and its subsystems is given in the collaboration class diagram below. The users of Active Teaming are separated into four categories: Visitor, Ordinary Users (OU), VIP, and Super Users (SU). Each user type interacts with Active Teaming through the application's user facing interface. Any events requiring special action are communicated with the appropriate *Handler*. The Handler has access to a set of services/managers that function as interfaces to Active Teaming's primary database, using FireBase. Through the services provided by each module the Handler provides an appropriate response to the use-cases (1 - 14) of each user type.

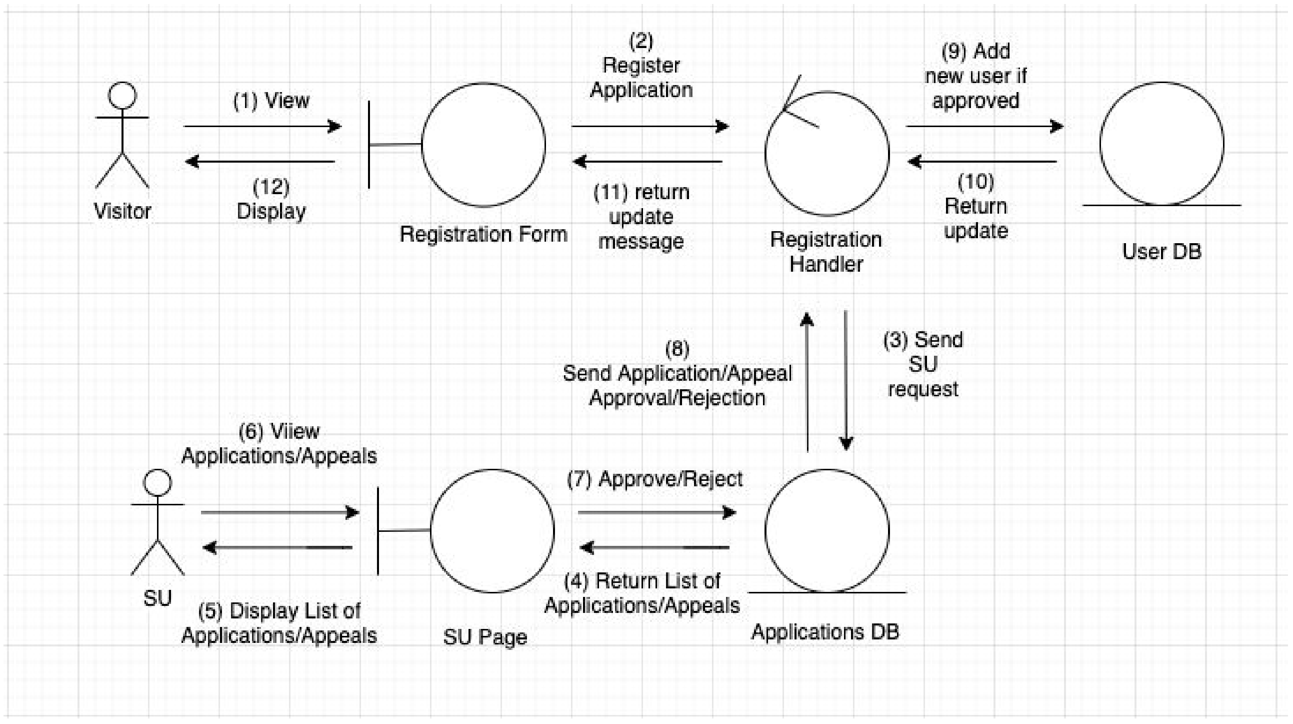


2. Use Case Analysis

2.1. Visitor/Guest Use Case Analysis

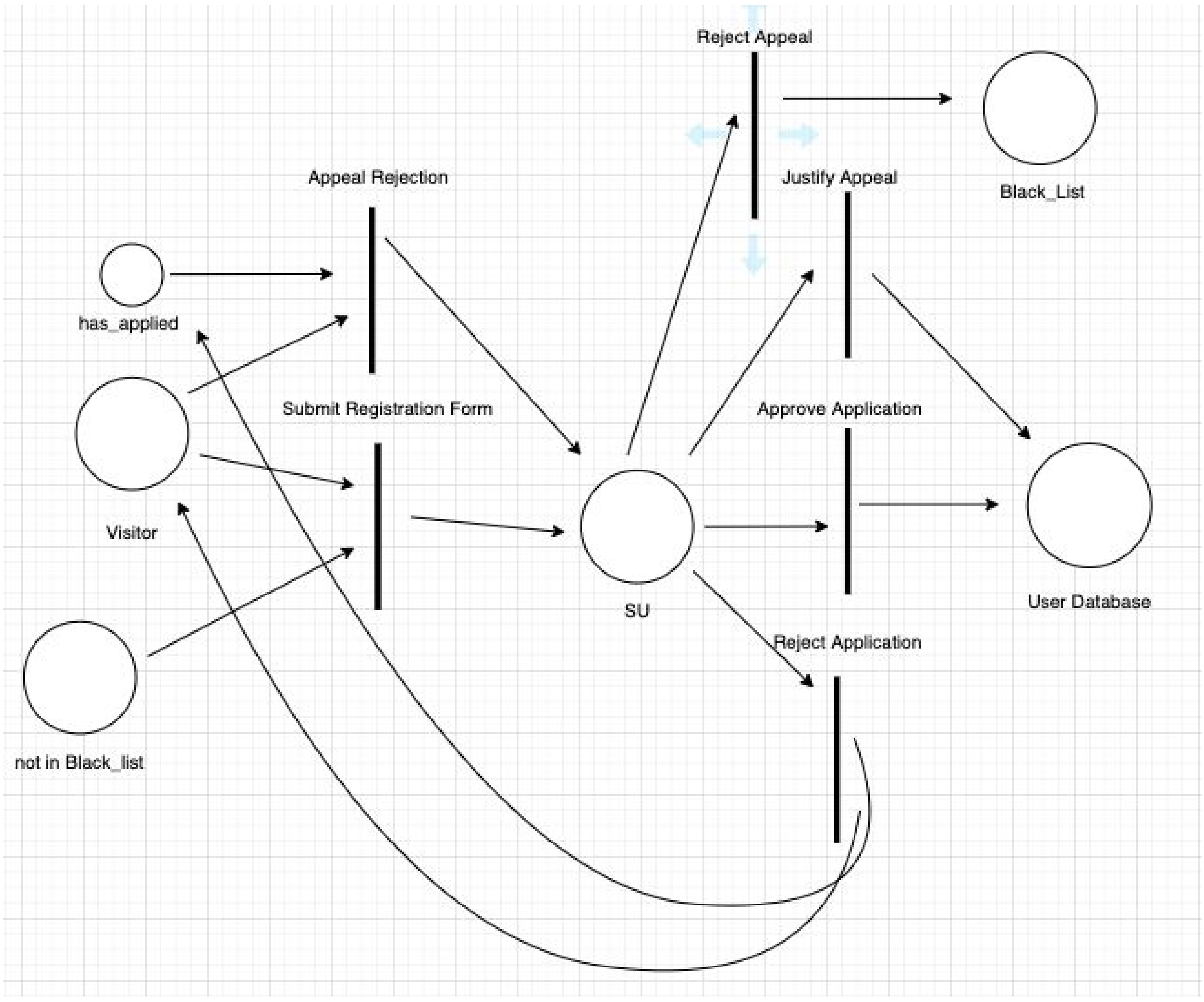
1. *Register*.

In the normal scenario, the visitor enters required signup information into the Registration Form which is then submitted into the system and handled by the Registration Handler (1) & (2). The application is then sent to SU (3) and the SU will review the application (6). After the application is reviewed by a SU, there are two possible scenarios: it is either approved or rejected (7). The response is received by the Registration Handler (8). If the application is approved, the visitor is added as a new user into the User Database (9) . Otherwise if the application is rejected, the visitor will receive a notification (10), (11) & (12). If the application is rejected there are two possible scenarios: The visitor may appeal or the visitor may not appeal. If the visitor does not appeal the decision, then the registration process ends at that point. If the visitor appeals the decision, there are two possible scenarios: the SU may reject or approve. If SU rejects, the registration process ends at that point. If the SU justifies the appeal, the visitor becomes a new user and is added to the User Database.

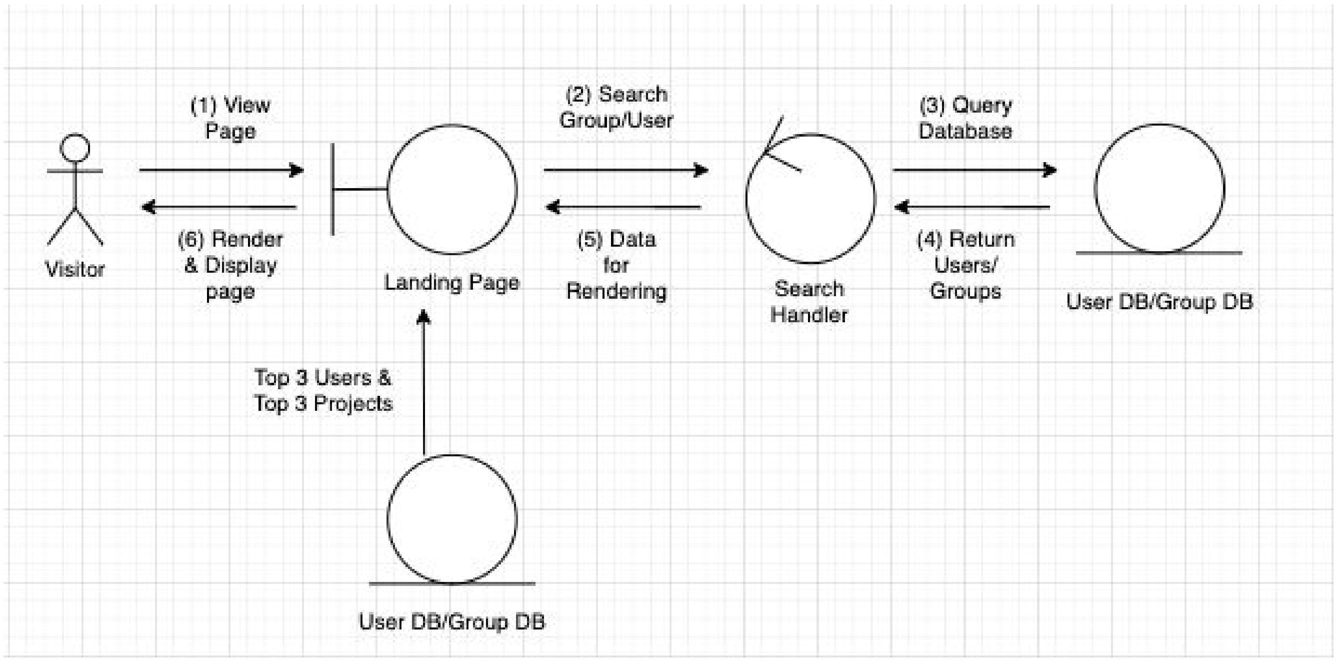


Visitor
(1) View
(12) Display
Registration Form
(2) Register Application
(11) return update message
Registration Handler
(9) Add new user if approved
(10) Return update
User DB
(8) Send Application/Appeal Approval/Rejection
(3) Send SU request
(6) Viiew Applications/Appeals
SU
(5) Display List of Applications/Appeals
SU Page
(7) Approve/Reject
(4) Return List of Applications/Appeals
Applications DB

Alternatively, find a Petri-net that represents the use case of user registration for visitor below:



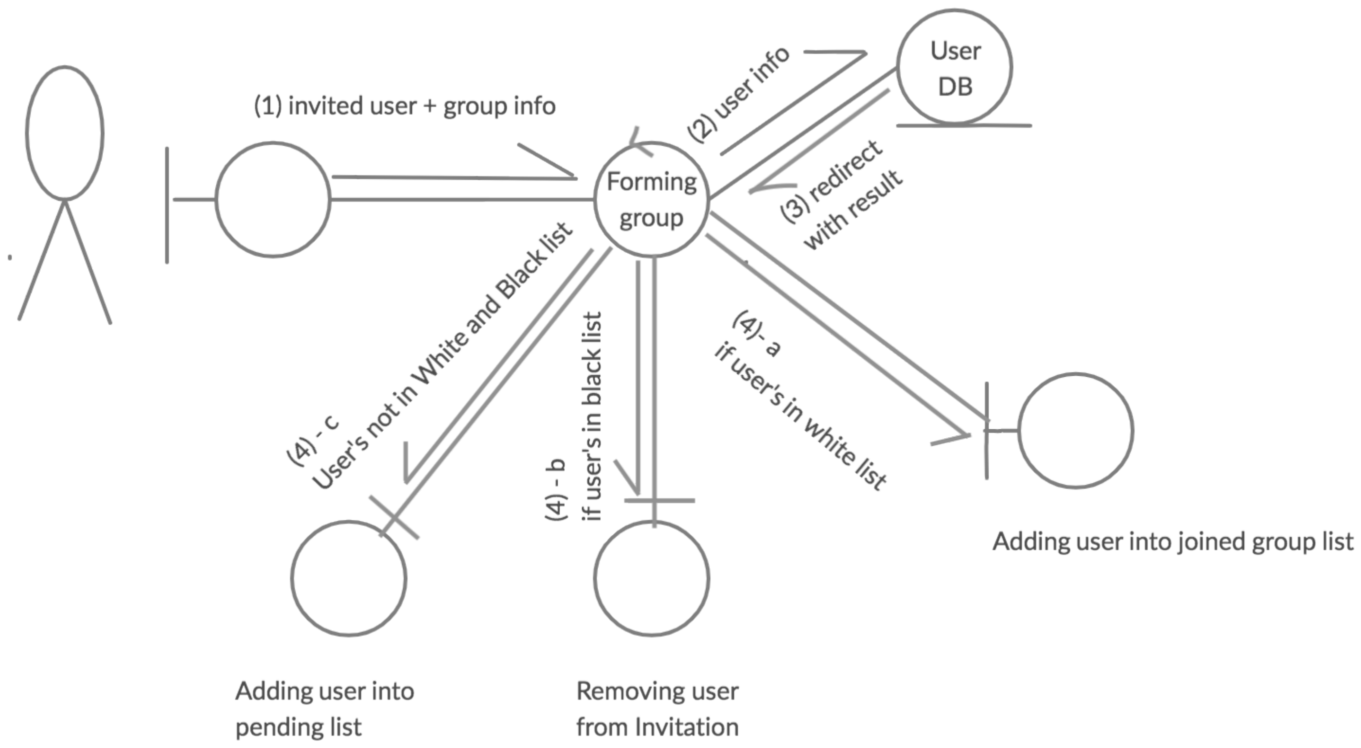
2. *Search Group and User Pages and View Top 3 Users and Groups*
   In the normal scenario, the visitor is able to view the landing page and see the top 3 OU/VIP/SU, and the top 3 Groups/Projects (1). The visitor is also able to search for specific users and groups or projects in the landing page search bar (2). The search handler will query the database (3), which will return the users or groups matching the input (4). The data to be rendered will then be specified and displayed to the user (5) & (6).



2.2. OU/VIP Use Case Analysis

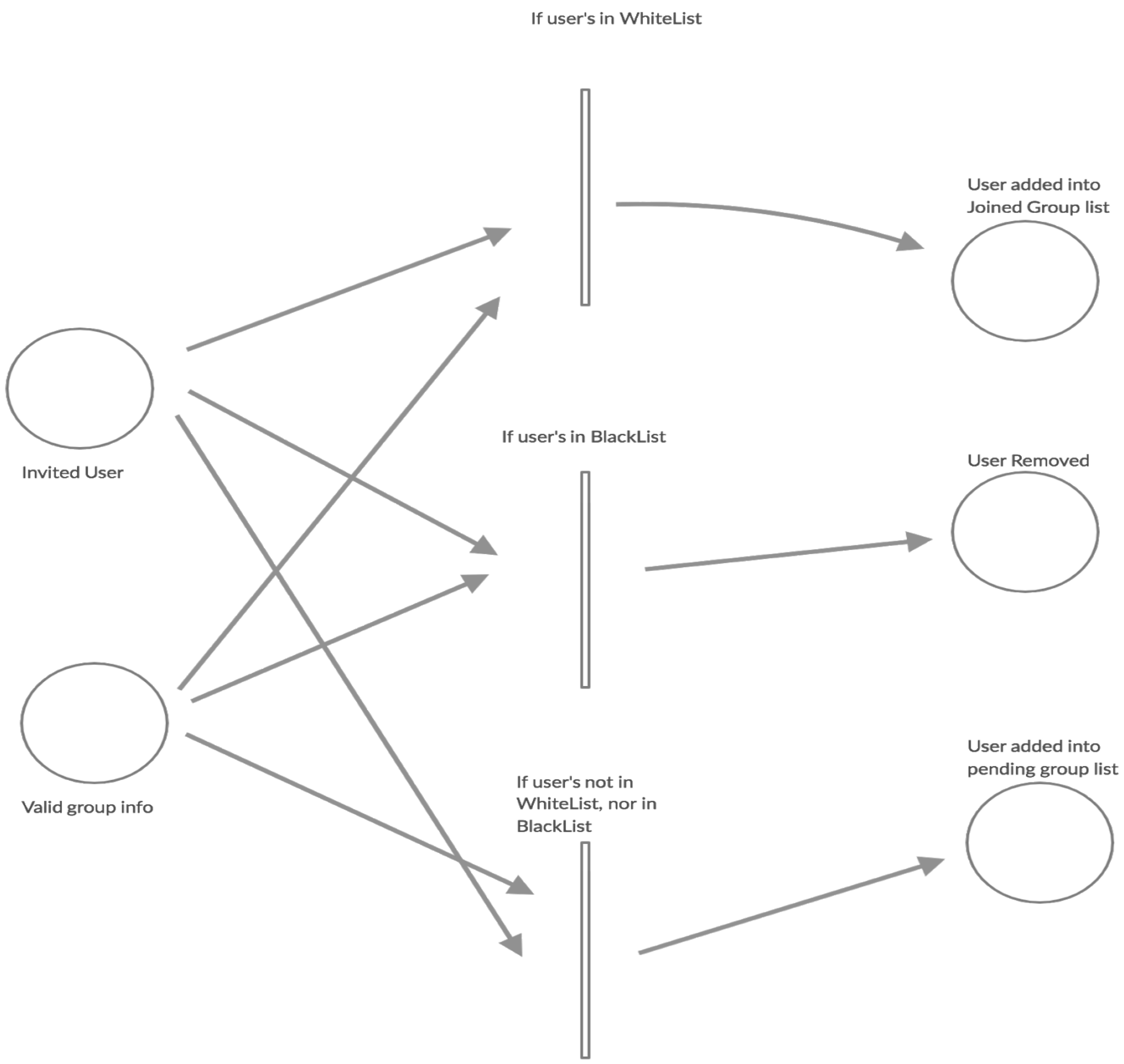
1. *Form new group*

When an invited user enters a forming group interface with group info that s/he is willing to join, (2) it checks the validation of the info in User DB and (3) return the result to the forming group process. (4)-a: If user's info is found in white list name, it adds the user into the joined group list. (4)-b: else if it is found in blacklist, then it removes the user from the invitation. (4)-c: if not both of the cases (4)-a, (4)-b, then it adds the user's info into the pending list.

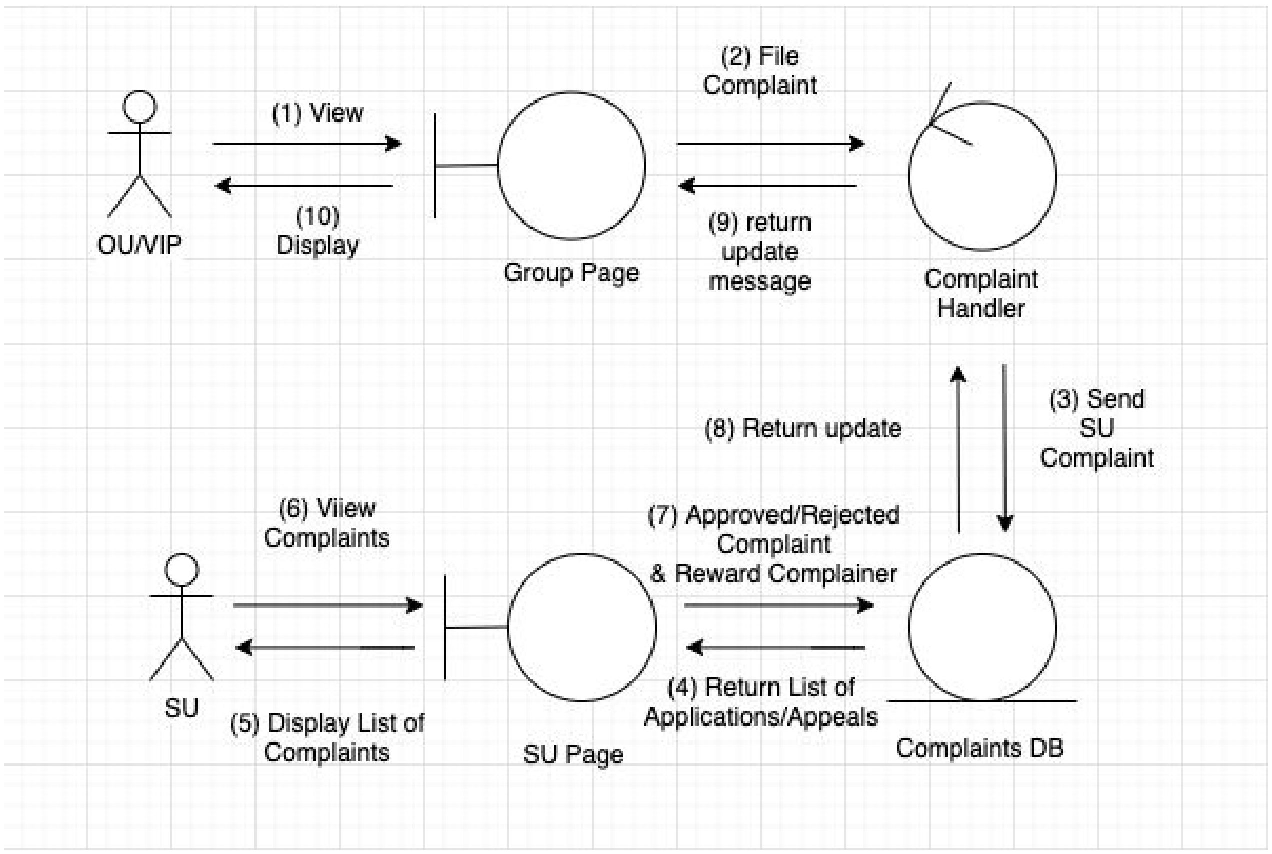


(1) invited user + group info
(2) user info
User DB
(3) redirect with result
Forming group
(4) - c User's not in White and Black list
(4) - b if user's in black list
(4)- a if user's in white list
Adding user into joined group list
Adding user into pending list
Removing user from Invitation

Alternatively, find below a Petri-net that represents the use case of forming a group:

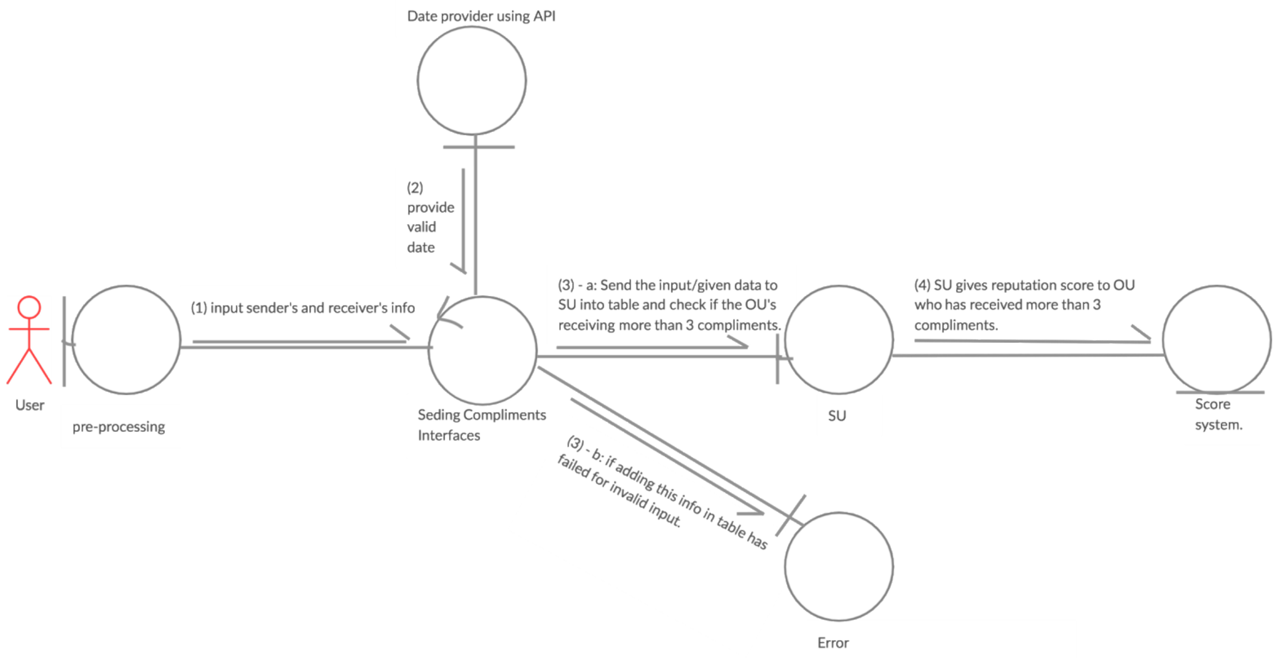

If user's in WhiteList
User added into Joined Group list
Invited User
If user's in BlackList
User Removed
Valid group info
User added into pending group list
If user's not in WhiteList, nor in BlackList

2. *File Complaint*
In the normal scenario, an OU/VIP will be able to file a complaint against another OU/VIP. The user views the form to file a complaint and submits to the system the complaint (1) & (2). The Complaint Handler will store the complaint in the Complaint DB and send it to the SU too (3), returning a message to the user after the complaint is successfully sent to SU. The SU will see the list of pending complaints and determine whether the complaint is invalid or justified (5), (6) & (7). Two natural scenarios arise: The SU rejects complaint or the SU approves complaint. If SU rejects the complaint, the complaint filing process ends at that point. If the SU approves the complaint, the filing user will be rewarded and the SU will punish the target of the complaint.



3. *Send Compliment*

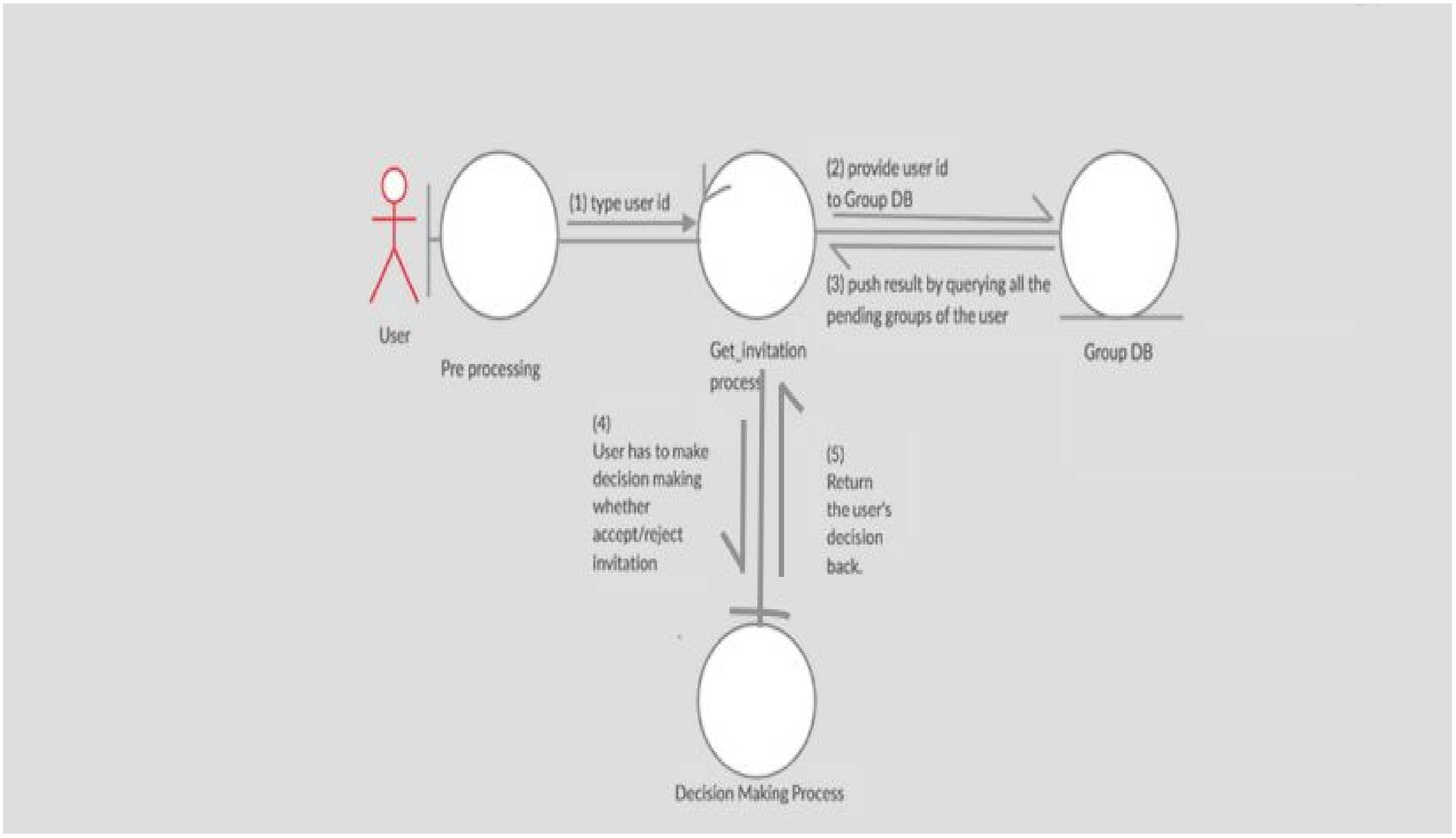
When a user is trying to send a compliment to another user, (1) the user is asked to type his/her info along with the receiver's info. (2) at the same time, Date will be provided into the sending compliments interface. (3) if all the typed, and given info are valid, they will be added into the table to send a compliment to SU. (3) - b: if it's not valid, it can't be added to the table and it returns an error. (4) When OU is receiving more than 3 compliments then SU gives the OU reputational score.



Date provider using API
(2) provide valid date
(1) input sender's and receiver's info
(3) - a: Send the input/given data to SU into table and check if the OU's receiving more than 3 compliments.
(4) SU gives reputation score to OU who has received more than 3 compliments.
User
pre-processing
Seding Compliments Interfaces
SU
Score system.
(3) - b: if adding this info in table has failed for invalid input.
Error

4. *Get invitation*

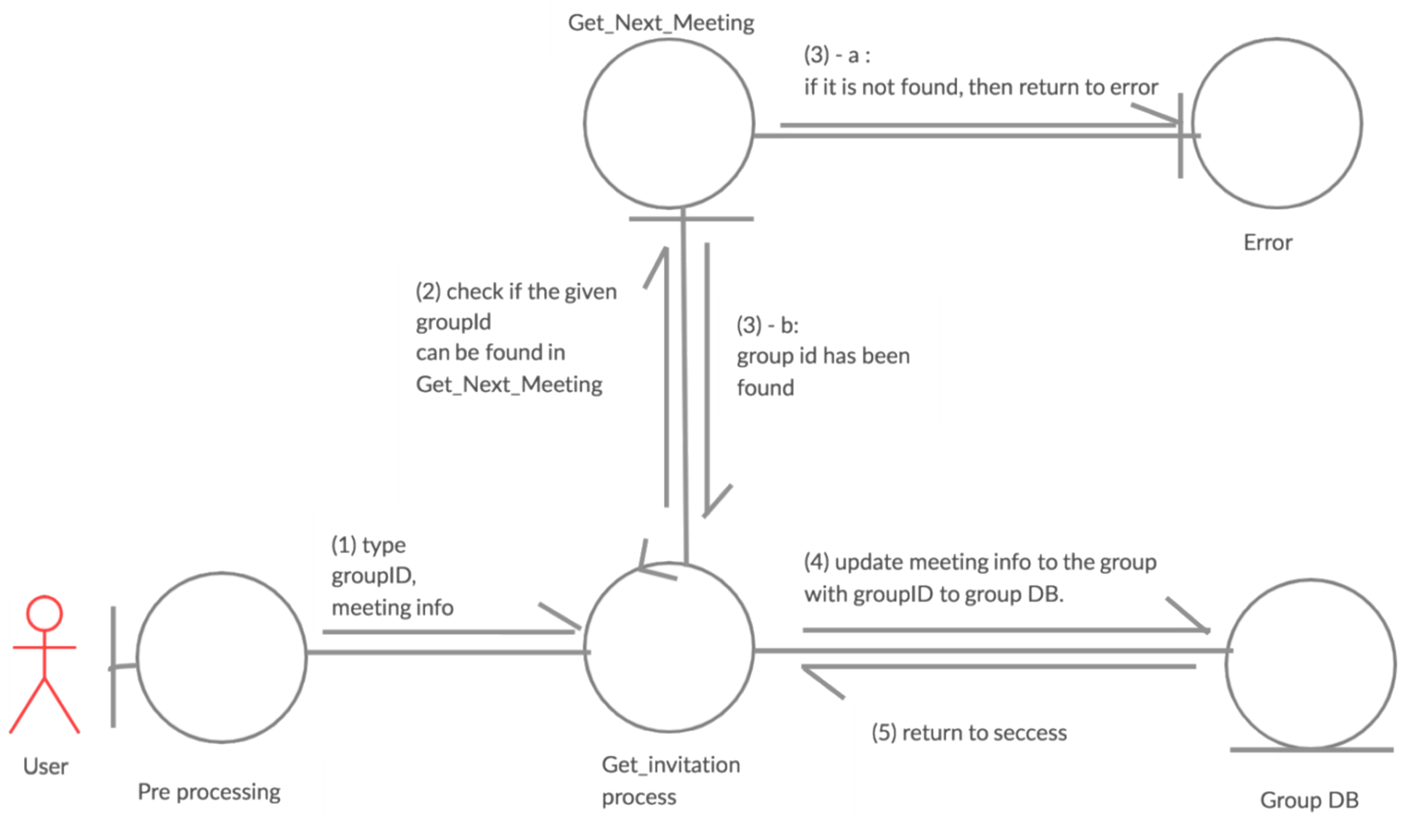
For a user to get an invitation, (1) user is asked to type its user id, and (2) group DB checks for validation of the given info. (3) Then it pushes the result to the get_invitation process by querying all the pending groups of the user. (4) After checking the validation of the input, User has to make a decision whether to accept/reject the invitation. (5) Then Decision Making process pushes back the result.



User
Pre processing
(1) type user id
Get_invitation process
(2) provide user id to Group DB
(3) push result by querying all the pending groups of the user
Group DB
(4) User has to make decision making whether accept/reject invitation
(5) Return the user's decision back.
Decision Making Process

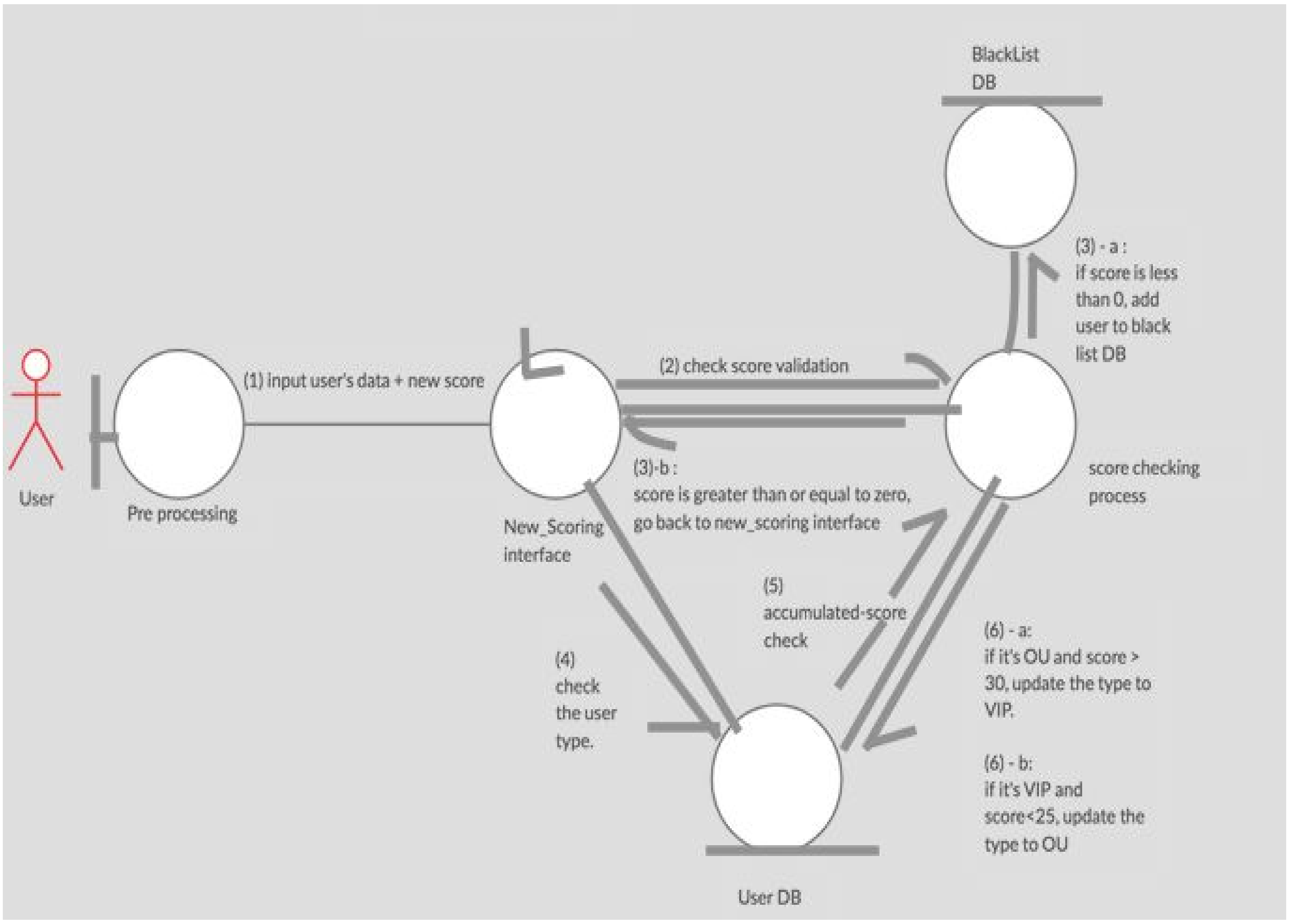
5. *New meeting*

When a user wants to have a new meeting, (1) the user is asked to type groupID and meeting info into the Get_invitation interface. (2) Then it checks if groupId can be found in Get_Next_Meeting. (3)-a: if it is not found, simply return to error. (3)-b: if Get_Next_meeting found the groupId and pushes back the result to the Get_Invitation process,(4) it will update meeting info to the group with groupID in group DB. (5) no error has occurred during the update, and it returns to success.

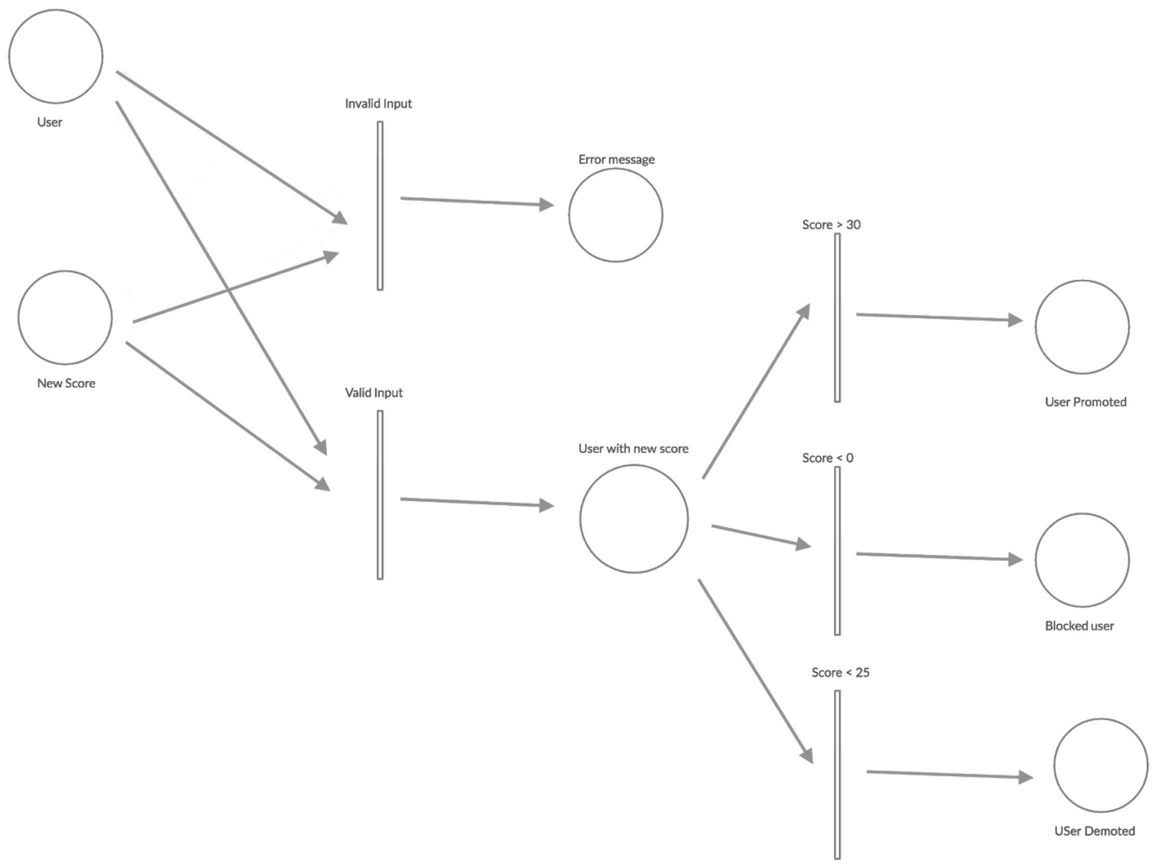

Get_Next_Meeting
(3) - a :
if it is not found, then return to error
Error
(2) check if the given groupId can be found in Get_Next_Meeting
(3) - b: group id has been found
(1) type groupID, meeting info
(4) update meeting info to the group with groupID to group DB.
(5) return to seccess
User
Pre processing
Get_invitation process
Group DB

6. *Score system*

In the Score system, (1) user is asked to input the user's data with new score into New_score process. (2) Then it sends the data to check the score to the score checking process. (3)-a: if the score is less than 0, user will be put in Blacklist DB. (3)-b: if not, push it back to new_scoring process and (4) check the user type in userDB. (5) Then grab the accumulated-score from score checking process. (6)-a: if user type is OU and the score is greater than 30, it updates the type to VIP. (6)-b: if user type is VIP and the score is less than 25, it updates the type to OU.

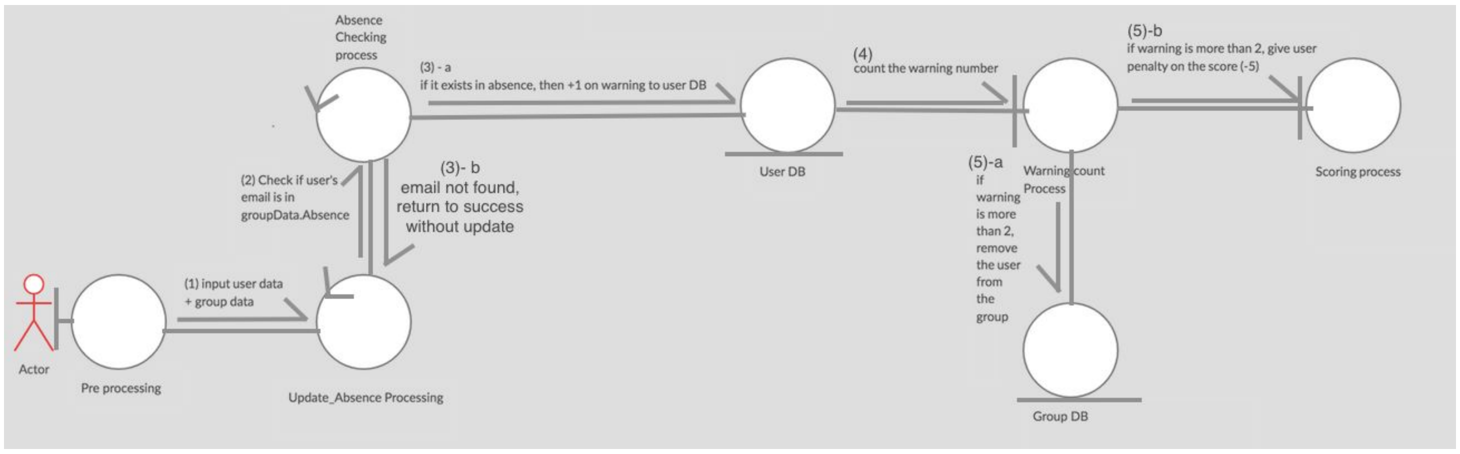

BlackList DB
(3) - a : if score is less than 0, add user to black list DB
(1) input user's data + new score
(2) check score validation
User
Pre processing
New_Scoring interface
(3)-b : score is greater than or equal to zero, go back to new_scoring interface
score checking process
(5) accumulated-score check
(4) check the user type.
(6) - a: if it's OU and score > 30, update the type to VIP.
(6) - b: if it's VIP and score<25, update the type to OU
User DB

Alternatively, find a Petri-net that represents the use case of the score system below:



7. *Update Absence*

For a user to be able to update absences, (1) the user is asked to input user data with group data into Update_Absence processing. (2) then check if the user's email is in groupData.Absence. (3)-a: if it exists in absence, then warning gets +1 in the user DB. (3)-b: if the user's email is not found, return to success without further updates. (4) Check how many warning numbers the user has. (5)-a : if warning is more than 2, it removes the user from the group in group DB. (5)-b: if warning is more than 2, it gives user penalty on the score (-5) in the scoring process.

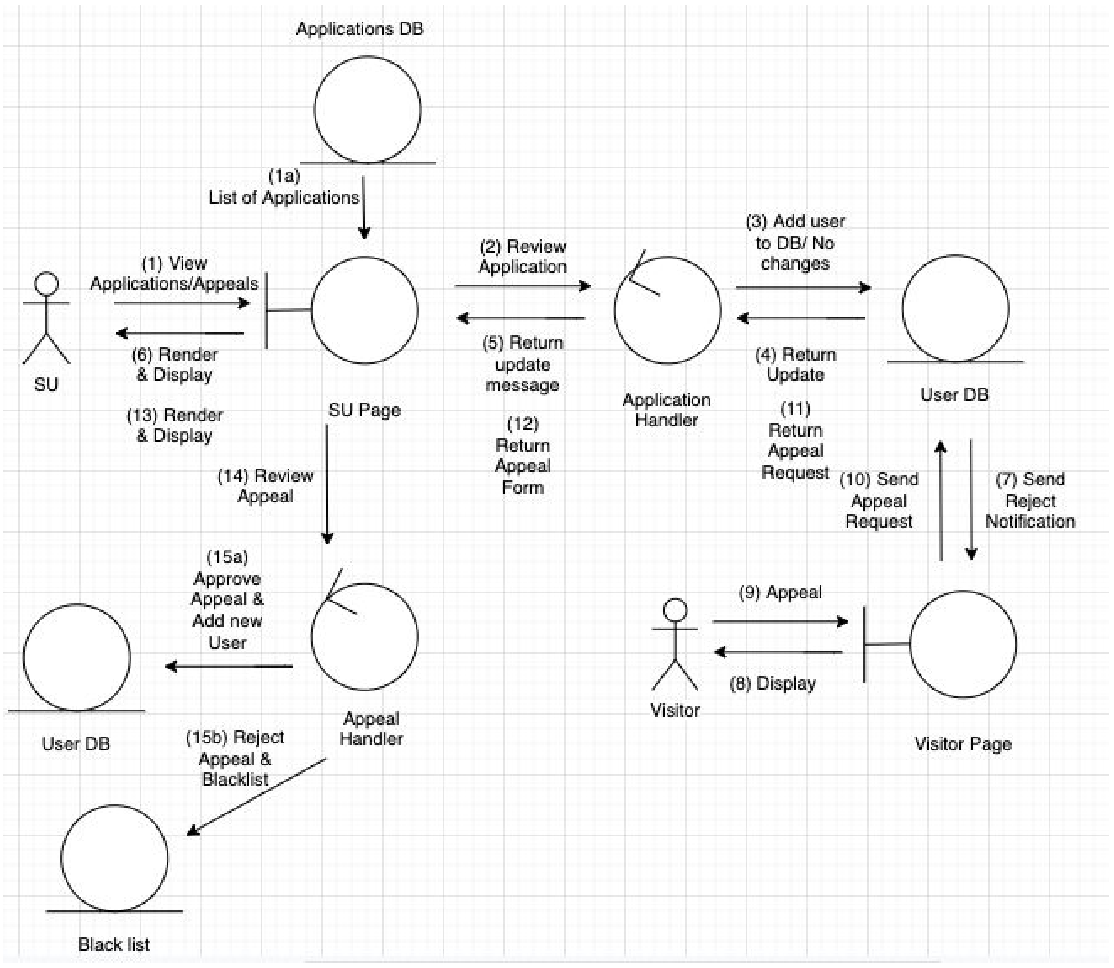


Absence Checking process
(3) - a
if it exists in absence, then +1 on warning to user DB
(4)
count the warning number
(5)-b
if warning is more than 2, give user penalty on the score (-5)
(2) Check if user's email is in groupData.Absence
(3)- b
email not found, return to success without update
User DB
(5)-a
if warning is more than 2, remove the user from the group
Warning count Process
Scoring process
(1) input user data + group data
Actor
Pre processing
Update_Absence Processing
Group DB

2.3. SU Use Case Analysis

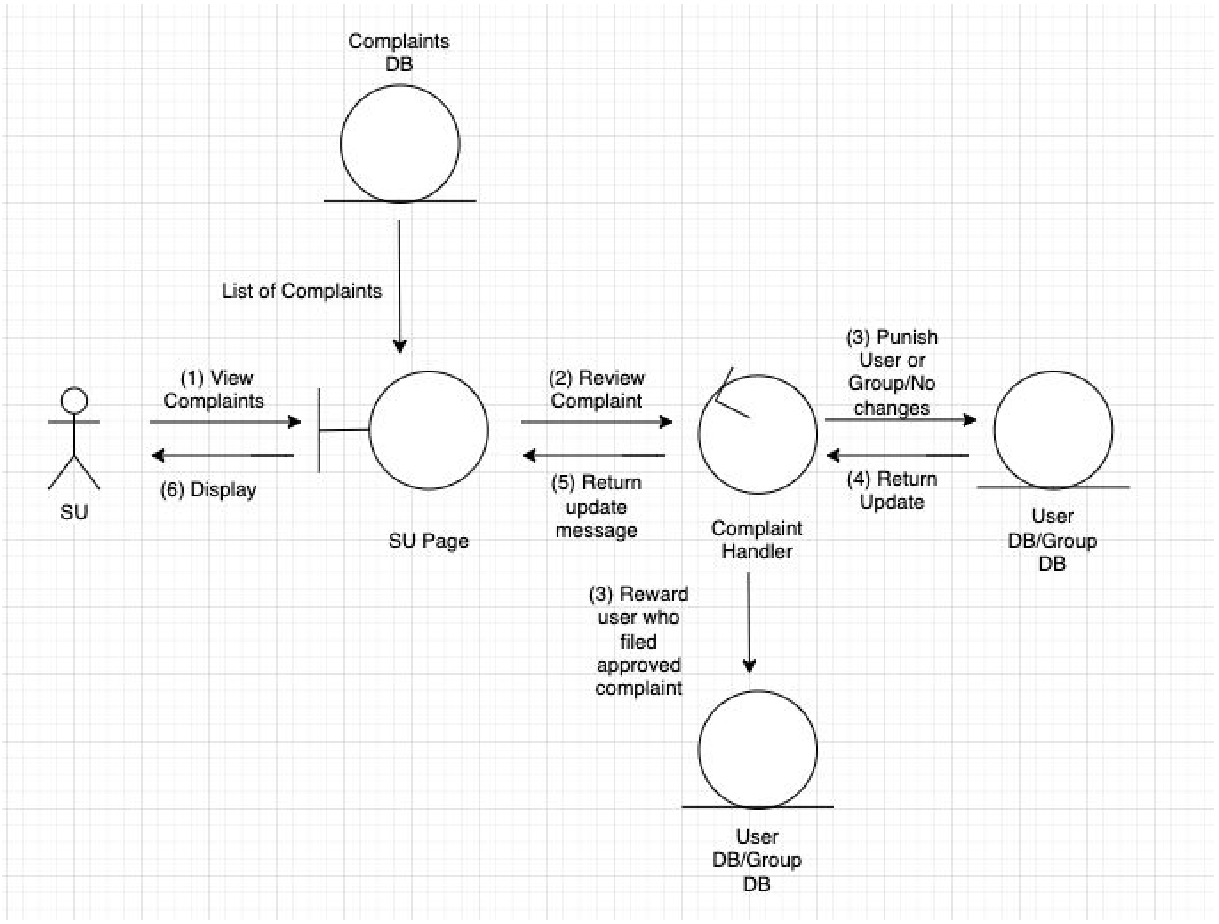
1. *Review Sign Up Applications*

In the normal scenario, the SU will be able to view a list of all the registration applications, review them and input the final decision into the system (1) & (2). Two natural scenarios arise: The SU approves the application or the SU rejects the application. If the SU approves the application, the applicant is added as a new user into the User Database by the Application Handler (3). If the SU rejects the application, the Application Handler sends a notification to applicant (7). The applicant has the option to appeal the rejection or to not appeal the rejection. If the applicant does not appeal the rejection, the system will add the applicant to a black list forever (15). If the applicant appeals the rejection, the Application Handler will send the appeal to the SU (10), (11), (12) & (13). If the SU justifies the appeal, Application Handler will add the applicant as a new user into the User Database (15). If the SU rejects the appeal, the system will add the applicant to a black list forever (15).

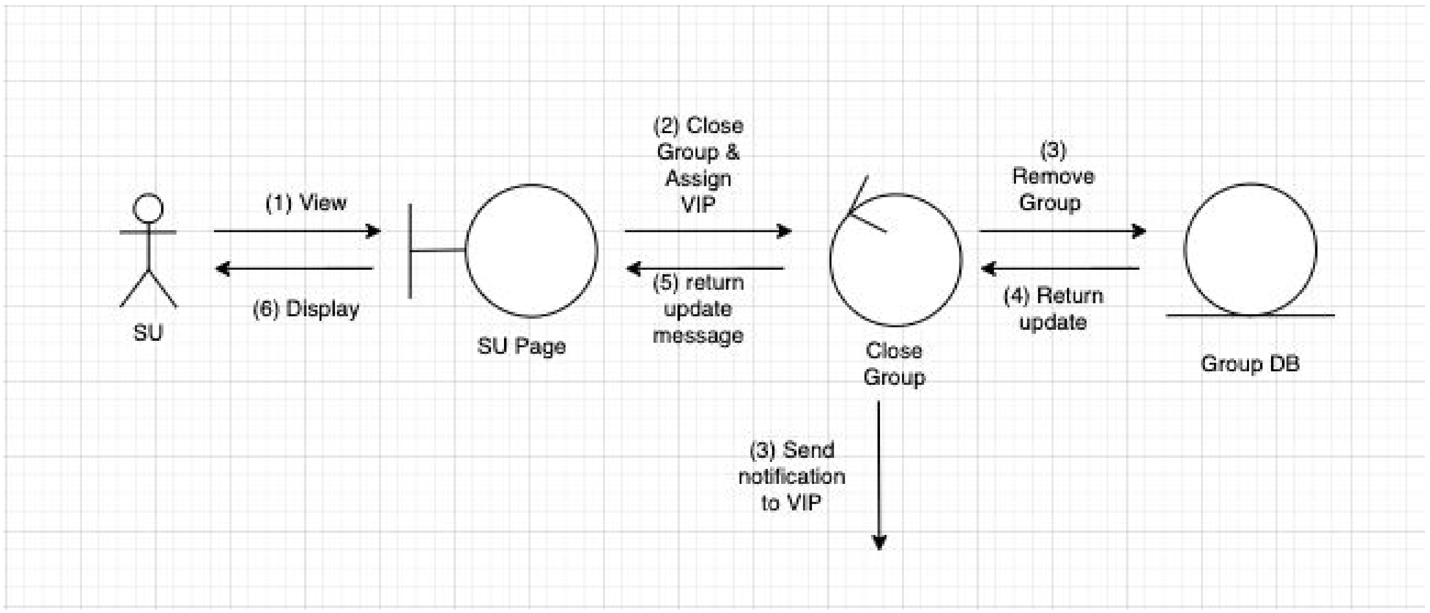


2. *Review Complaint*

In the normal scenario, the SU will be able to view and review user complaints (1). The decision is inputted into the system and the complaint handler receives it (2). Two natural scenarios arise: The SU justifies the complaint or the SU rejects the complaint. If the complaint is valid, the Complaint Handler will punish the user target or group target of the complaint (3). The Complaint Handler will also reward the filer of the complaint (3). An update message is sent back to the SU confirming the action was done successfully and the process ends at that point (4), (5) & (6). If the complaint is invalid, no changes are and an update message is sent back to the SU confirming the action was done successfully (4), (5) & (6), ending the process at that point.

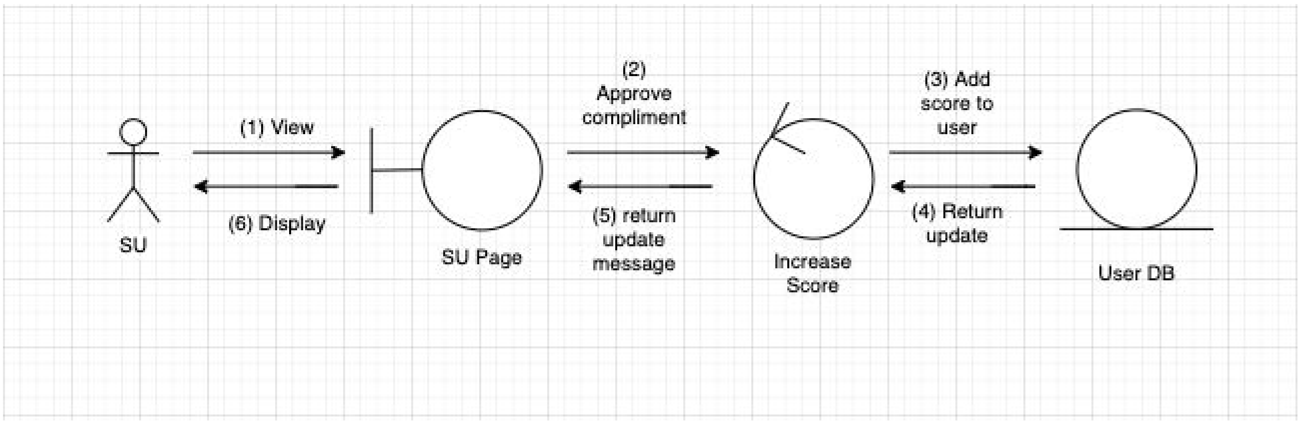


3. *Close Groups Voted to Close and Assign VIP for Evaluation*
   In the normal scenario, the SU will be able to view the groups that have been voted to be closed (1). The SU will close the group and assign a VIP user to complete the evaluation of the group (2). The SU will receive an update message from the system confirming action was done successfully (4), (5) & (6). Once the evaluation is completed by the VIP, the group is removed from the Groups Database (3).



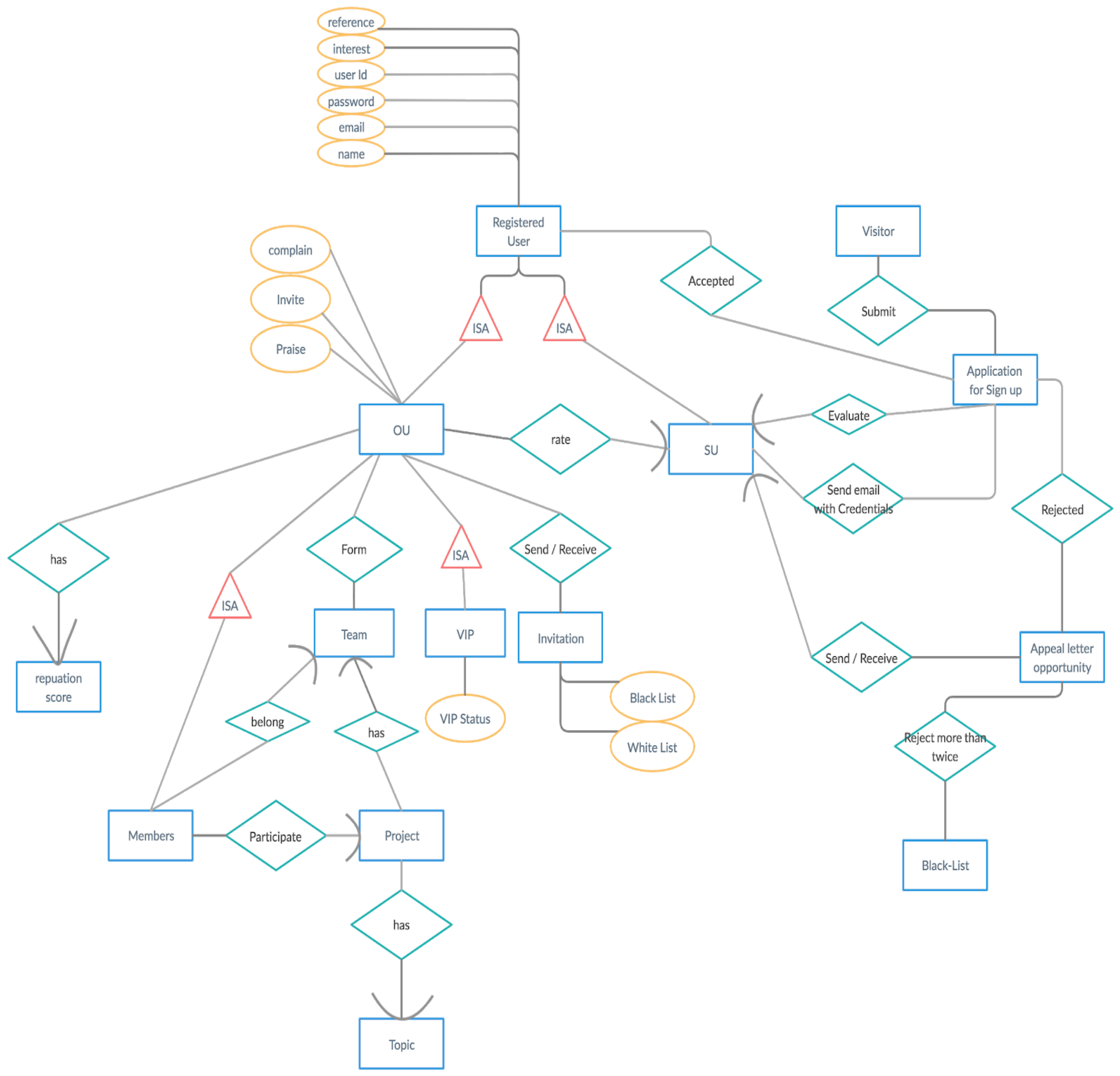
SU
(1) View
(6) Display
SU Page
(2) Close Group & Assign VIP
(5) return update message
Close Group
(3) Remove Group
(4) Return update
Group DB
(3) Send notification to VIP

4. *Review Compliments*
   In the natural scenario, the SU will be able to see and review compliments sent to other users (1). The SU may either approve the compliment or not respond. If the SU approves the compliment (2), the system will reward the target of the compliment (3) and return a successful update message to SU (4), (5) & (6). If SU does not handle compliment, there are no changes in the system.



3. Entity Relation Diagram of System

The following Entity Relation(E/R) diagram describes the relationship between different entities in Active Teaming System and it also includes the key and attributes of each entity.



4. Detailed Design

(a) Landing Page

```
@Input null
@Output a list of top rating OU profiles and SU profiles
function getUserList () {
  users = []
  query from user DB, push result to users
  return users
}

@Input null
@Output a list of top rating projects
function getGroupList () {
  groups = []
  query from group DB, push result to groups
  return groups
}
```

(b) Visitor Detail

```
@Input visitor's personal information (name, email, interest, credential, reference)
@Output success/error
function signUp (userInfo) {
  if userInfo in blacklist or in DB:
      return error
  else:
      add userInfo to registerUser DB
      return success
}

@Input email, password
@Output success/error
function signIn (email, password) {
  if email in blacklist or not in DB:
      return error
  else:
      sign in with email and password, and update current user data
      if first time sign in: redirect to reset password page
```

```
    return success
}

@Input user email
@Output registration status (pending, accepted, rejected, blocked, appealed)
//when user first sign up, status = pending
//when admin reject the user 1st time, status = rejected
//when admin accept the user, status = accepted
//when the user appeal, status = appealed
//when admin reject the user 2nd time, status = blocked
function registrationStatus (userEmail) {
  if userEmail not in registerUser DB:
    return error
  else:
    get userInfo from registerUser DB
    return userInfo.status
}

@Input user email
@Output success/error
function appeal (userEmail) {
  status = registrationStatus(userEmail)
  if status != rejected:
    return error
  else:
    update status to appealed in registerUser DB
    return success
}
```

(c) **OU Detail**

```
@Input user id
@Output list of references
function getReference (userID) {
  references = []
  query all the users of the user with userID referred from user DB, push result to
references
  return references
}

@Input current user data, referred user data, score
```

```
@Output success/error
function giveScore (referredUser, currentUser, score) {
  maxScore = 10 if currentUser is ‘OU’, 20 if currentUser is ‘VIP’
  if score > maxScore: score = maxScore
  if score < 0: score = 0
  update score to referredUser to User DB, if failed: return error
  return success
}

@Input user id
@Output list of whitelist users
function getWhitelist (userID) {
   whitelist = []
   query all the users of the user with userID put into whitelist from user DB, push result to whitelist
   return whitelist
}

@Input current user data, userToBeAdded data
@Output success/error
function addToWhitelist (userToBeAdded, currentUser) {
  if userToBeAdded not in user DB or userToBeAdded already in whitelist:
    return error
  add this user to current user’s whitelist, return success
}

@Input user id
@Output list of whitelist users
function getBlacklist (userID) {
   blacklist = []
   query all the users of the user with userID put into blacklist from user DB, push result to blacklist
   return blacklist
}

@Input current user data, userToBeAdded data
@Output success/error
function addToBlacklist (userToBeAdded, currentUser) {
  if userToBeAdded not in user DB or userToBeAdded already in blacklist:
    return error
  add this user to current user’s blacklist, return success
}

@Input current user data, new group data (name, description, private announcement,
invited users data)
@Output success/error
function formNewGroup (currentUser, newGroupData) {
  for invitedUser in newGroupData.invitations:
    if invitedUser not in user DB:
      return error

  for invitedUser in newGroupData.invitations:
    if current user in invitedUser whitelist:
      add new group to invitedUser joinedGroup list
      add invitedUser to this new group members

    else if current user in invitedUser blacklist:
      remove invitedUser from this new group invitations

    else:
          add new group to invitedUser pendingGroup list
  add this group to current user's group list and group table, return success
}

@Input current user data, complaint, reason
@Output success/error
function fileComplaint (currentUser, complaint group/ user, reason) {
  type = group if complaintGroup, else user
  sender = visitor if not currentUser, else currentUser data
  //use taboo system to process the reason, and update user's score
  newReason = tabooSystem(reason, currentUser)
  add this complaint and sender to complaint table, if failed: return error
  return success
}
```

(d) **VIP Detail**

```
@Input user id
@Output list of closed groups
function getGroupsNeedToEval (userID) {
   groups = []
   query all the closed groups assigned to the user with userID for evaluating, push
result to groups
```

```
   return groups
}

@Input group data
@Output success/error
function evalGroup (groupData, score) {
  score = max(-20, score) = min(20, score)
  for member in groupData.members:
       member.score += score
       update new score to user DB
  remove assigned VIP user from this closed group
  set the status of this closed group to evaluated
  if any of above process failed: return error
  return success
}

@Input sender data, receiver data
@Output success/error
function sendCompliment (sender, receiver) {
  newCompliment = {sender, receiver, date}
  add this newCompliment compliment table, if failed: return error
  return success
}

@Input user id
@Output list of invitations
function getInvitations (userID) {
  invitations = []
   query all the pending groups of the user with userID from group DB, push result to
invitations
  return invitations
}

@Input invitation data
@Output success/error
function invitationAction (accept/reject, invitation, reason, currentUser) {
  if accept:
       add currentUser into invitation group members list
       add invitation group in current user's group list
  if reject:
    newReason = tabooSystem(reason, currentUser)
```

```
        update this invitation and new reason in group DB
        remove invitation group from pending groups list
  if failed:
    return error
  return success
}

@Input user data
@Output success/error
function newVoteSU (userData) {
  create a new vote = {userData.id, voteYes:0, voteNo:0, totalVoted:0, votedVIP:[ ]}
  add this vote to vote table if failed: return error
  return success
}

@Input null
@Output list of democratic SU votes
function getSUvote () {
  votes = []
  query all the SU votes from vote DB, push result to votes
  return votes
}

@Input choice, vip data, candidate data
@Output success/error
function voteForSomeone (choice, VIPdata, candidateData) {
  if choice = yes:
     //this vip have not voted yet
     if VIPdata.id not in votedVIP:
         update candidateData.yes+1, totalVoted+1, and VIPdata.id to vote DB
     else: return error
  else:
     if VIPdata.id not in votedVIP:
         update candidateData.no+1, totalVoted+1, and VIPdata.id to vote DB
     else: return error
  return success
}
```

(e) **Taboo System**

```
@Input null
@Output list of taboo words
```

```
function getTabooWords () {
  tabooWords = []
  query all the taboo words from taboo DB, push result to tabooWords
  return tabooWords
}

@Input user data, message said
@Output new message after filter all the taboo words
function getTabooWords (message, user) {
  newScore = user.score
  tabooWordsSaid = user.tabooWordsSaid
  tabooWords = getTabooWords()
  newMessage = convert all words in message that is also in tabooWords to ***

  for word in tabooWordsInMessage:
      if word in tabooWordsSaid: newScore-=5
      else: newScore-=1

  //update new score and user's role based on score using score system
  If scoreSystem(user, newScore) = error: return error

  return newMessage
}
```

(f) **Score System**

```
@Input user data, new score
@Output success/error
function scoreSystem (user, newScore) {
  if score < 0:
      add user to blacklist DB
      return success
  if user type is OU and newScore > 30: update type to VIP
  if user type is VIP and newScore < 25: update type to OU

  update new user data to user DB, if failed: return error

  return success
}
```

(g) **Group System**

```
@Input group id
```

```
@Output list of posts
function getPosts (groupID) {
  posts = []
  query all the posts of the group from group DB, push result to posts
  return posts
}

@Input current user data, group id
@Output success/error
function newPost (currentUser, groupID, post) {
  newPost = tabooSystem(post, currentUser)
  add new post to the group which the groupID point to, if failed, return error
  return success
}

@Input group id, user id
@Output list of tasks
function getAssgignedTasks (groupID, userID) {
   tasks = []
   query all the assigned tasks related with userID and groupID from group DB, push
result to tasks
   return tasks
}

@Input group id, user id
@Output list of tasks
function getFinishedTasks (groupID, userID) {
   tasks = []
   query all the finished tasks related with userID and groupID from group DB, push
result to tasks
   return tasks
}

@Input group id
@Output list of votes
function getUnsovledVotes (groupID) {
  votes = []
  query all the unsolved votes of the group with groupID from group DB, push result to
votes
  return votes
}
```

```
@Input group id
@Output list of warnings
function getWarnings (groupID) {
   warnings = []
   query all the warnings of the group with groupID from group DB, push result to warnings
   return warnings
}

@Input group id
@Output evaluation
function getWarnings (groupID) {
  if the group with groupID is closed:
       return query the evaluation of this group from group DB
  return null
}

@Input group id
@Output meeting info
function getNextMeeting (groupID) {
  meeting = query incoming meeting of the group with groupID from group DB
  return meeting
}

@Input group id, meeting info
@Output success/error
function newMeeting (groupID, meetingInfo) {
  incomingMeeting = getNextMeeting(groupID)
  if incomingMeeting != null:
       return error
  update meetingInfo to the group with groupID to group DB
  return success
}

@Input group data, user data
@Output success/error
function updateAbsense (groupData, userData) {
  if the userData.email in groupData.absense:
       update userData.warning + 1 to user DB
       if warning > 2:
          remove user from group
          newScore = userData.score – 5
          scoreSystem(userData, newScore)
  update groupData.absense.user + 1 to group DB
  if any update failed: return error
  return success
}

@Input user id, vote id, choice, groupData
@Output success/error
function updateVote (userID, voteData, choice, groupData) {
  totalVoted = voteData.total + 1
  yes = voteData.yes
  no = voteData.no
  if choice = yes: yes += 1
  else: no += 1
  if groupData.member.length = totalVoted:
      if voteData.type = meeting:
        if yes > no:
          status = approved
          newMeeting(groupID, voteData)
        else: status = rejected

      else if voteData.type = warning:
        if yes = totalVoted:
          status = approved
          update user.warning + 1 to user DB
             if warning > 2:
               remove user from group
               newScore = userData.score – 5
               scoreSystem(userData, newScore)

        else: status = rejected

      else if voteData.type = kickOut:
        if yes = totalVoted:
          status = approved
          newScore = user.score – 10
          scoreSystem(user, newScore)
          remove user from group
        else: status = rejected
      else if voteData.type = closeGroup:
        if yes = totalVoted:
          status = approved
          update this group.status = closed to group DB
        else: status = rejected

  update totalVoted, yes, no, status to vote DB
  if any of above DB operation failed: return error
  return success
}
```

(h) **SU Detail**

```
@Input null
@Output list of registrations
function getRegistration () {
  registrations = []
  query all the users of from pending user DB, push result to registrations
  return registrations
}

@Input user data
@Output success/error
function acceptRegistration (userData) {
  add userData to user DB, and send an email with login id and default password to
userData.email, if failed: return error
  return success
}

@Input user data
@Output success/error
function rejectRegistration (userData) {
  if userData.status = init: update it to reject to pending user DB
  else if userData.status = appealed: update it to blocked to pending user DB
  if failed: return error
  return success
}

@Input null
@Output list of complaints
function getComplaints () {
  complaints= []
  query all the complaints from complaints DB, push result to complaints
  return complaints
}

@Input target data, action
@Output success/error
function acceptComplaint (targetData, action) {
  update complaint to solved
  if target is user:
       if action = punish: update targetData.score-5 to user DB
  if target is group:
       if action = shut down and deduct scores:
         update targetData.status to closed
         for member in targetData.members:
         update member.score-5 to user DB

       else if action = shut down and kick out members:
         update targetData.status to closed
         for member in targetData.members:
           update member.blocked = true to user DB
  if any of above DB operation failed: return error
  return success
}

@Input target data
@Output success/error
function rejectComplaint (targetData) {
  update complaint to solved, if failed: return error
  return success
}

@Input null
@Output list of compliments
function getCompliments () {
  compliments= []
  query all the compliments from compliments DB, push result to compliments
  return compliments
}

@Input target data, action
```

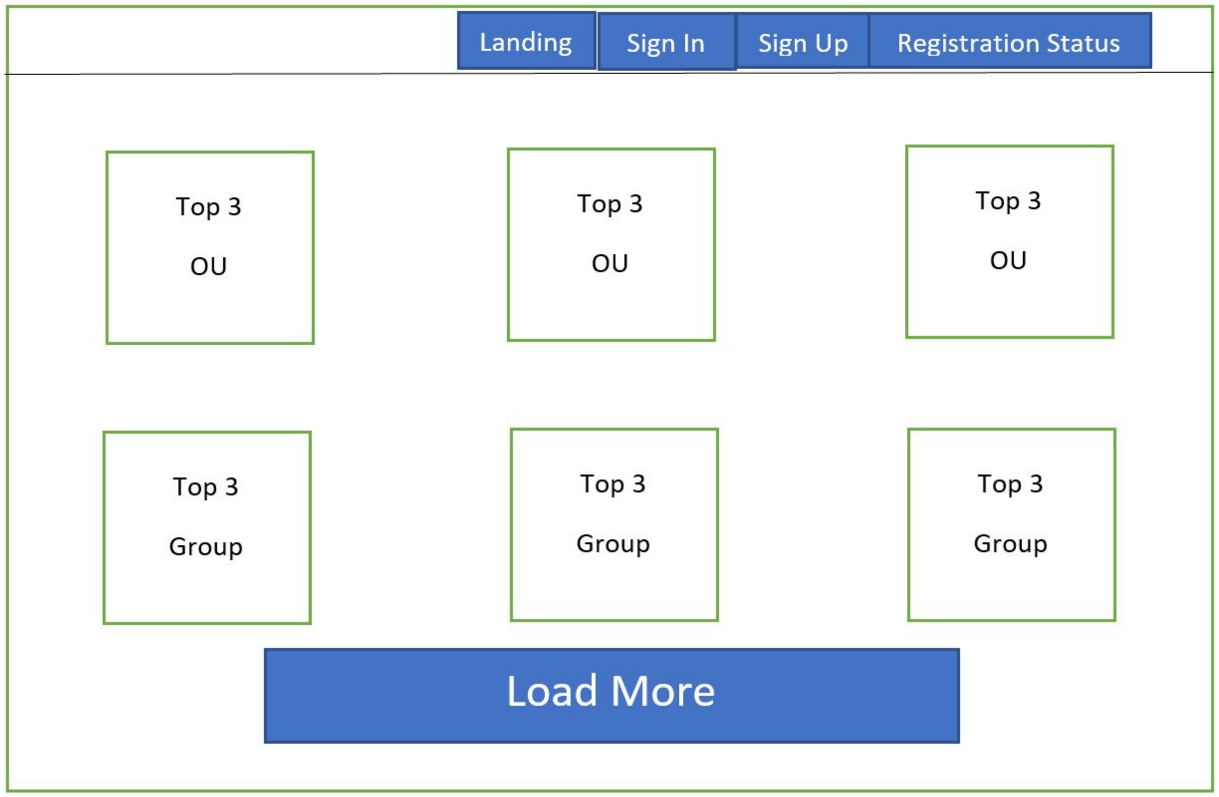
```
@Output success/error
function acceptCompliment (targetData) {
  if targetData.complimentCount > 2: return error
  else update targetData.score + 3, and targetData.complimentCount + 1 to user DB
  return success
}

@Input target data
@Output success/error
function rejectCompliment (targetData) {
  update compliment to solved, if failed: return error
  return success
}

@Input null
@Output list of VIPs
function getVIPSs () {
  VIPs= []
  query all the VIPs from user DB, push result to VIPs
  return VIPs
}

@Input null
@Output list of closed groups
function getClosedGroups () {
  closedGroups= []
  query all the closedGroups from group DB, push result to closedGroups
  return closedGroups
}

@Input target data, VIP data
@Output success/error
function assignEvaluation (targetData, VIPdata) {
  update target group to the VIP evalGroups list, if failed: return error
  return success
}
```

5. System Screens

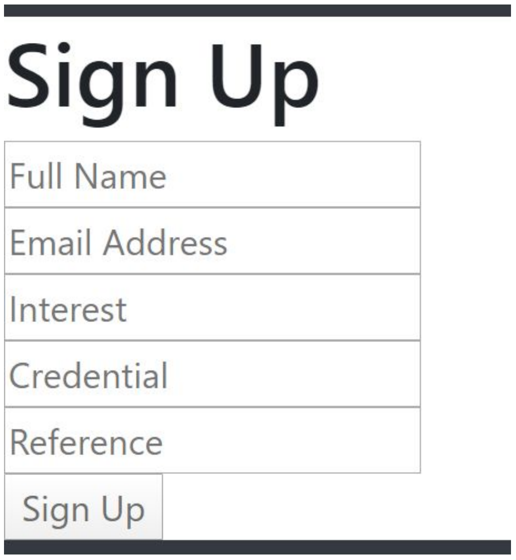
Major GUI screens of our system.

(i) Visitor screens

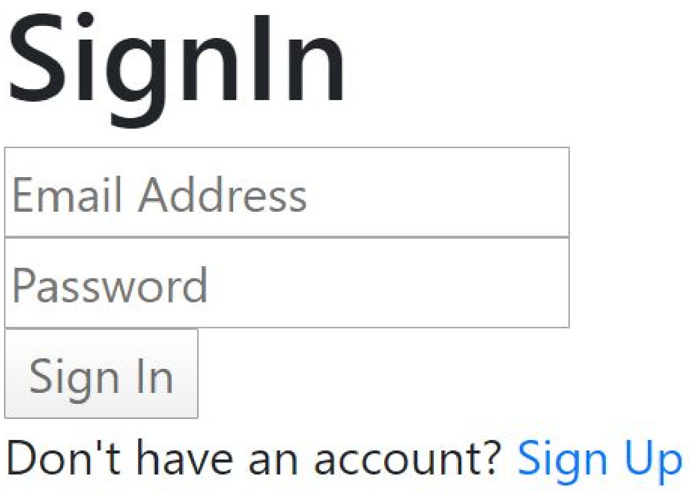
1. When the visitors visit our system, this is the first screen they will encounter. They can browse top 3 groups and OUs of our system. They can click the load more button to find more groups and OUs.



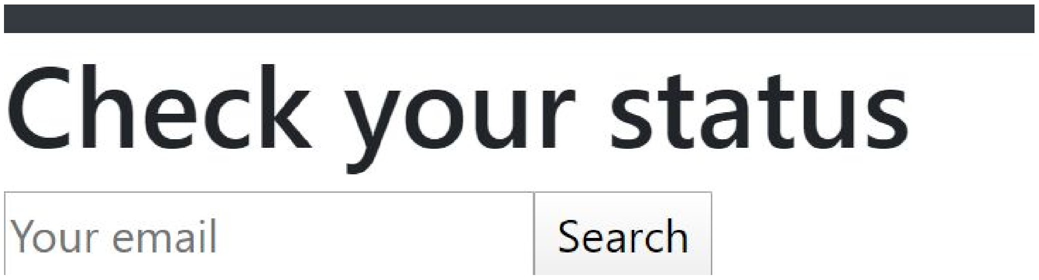
2. Sign up page
   Sign up page offers a form for visitors to sign up. Name and email address are required in order to register to our system. The sign up button is unclickable unless the user fills in required fields.



3. Sign in page
   Sign in page offers a form for the users to sign in. Email and password are required, and the sign in button would be clickable when the user fills in all required fields. If a user doesn't have an account yet, he/she can also click the Sign Up to sign up a new account.

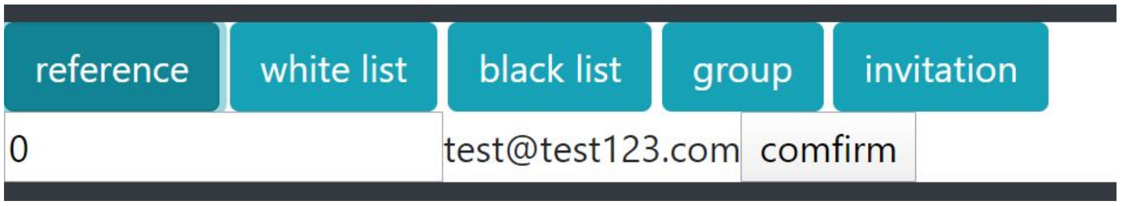


4. Registration status page
   This page offers a GUI for the visitors to check their registration status. Visitors can fill in there email address to search for the status.

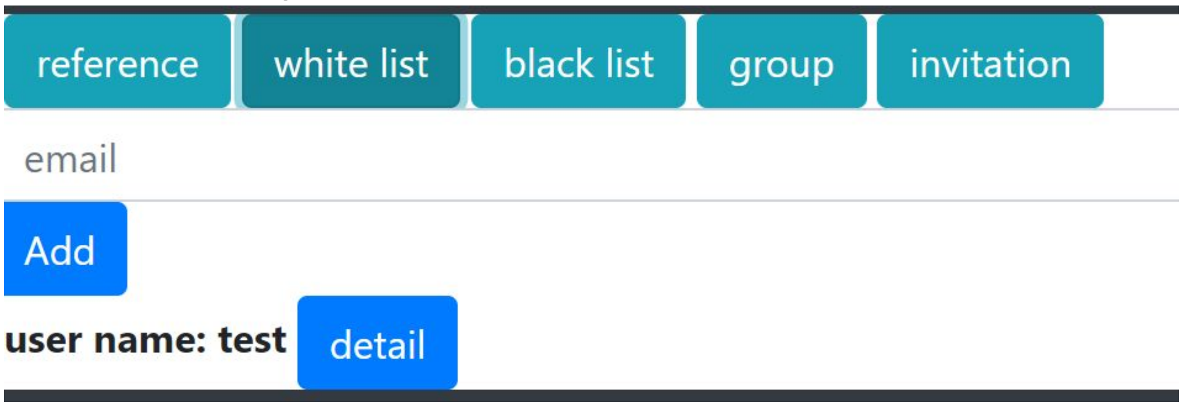


(j) **OU screens**

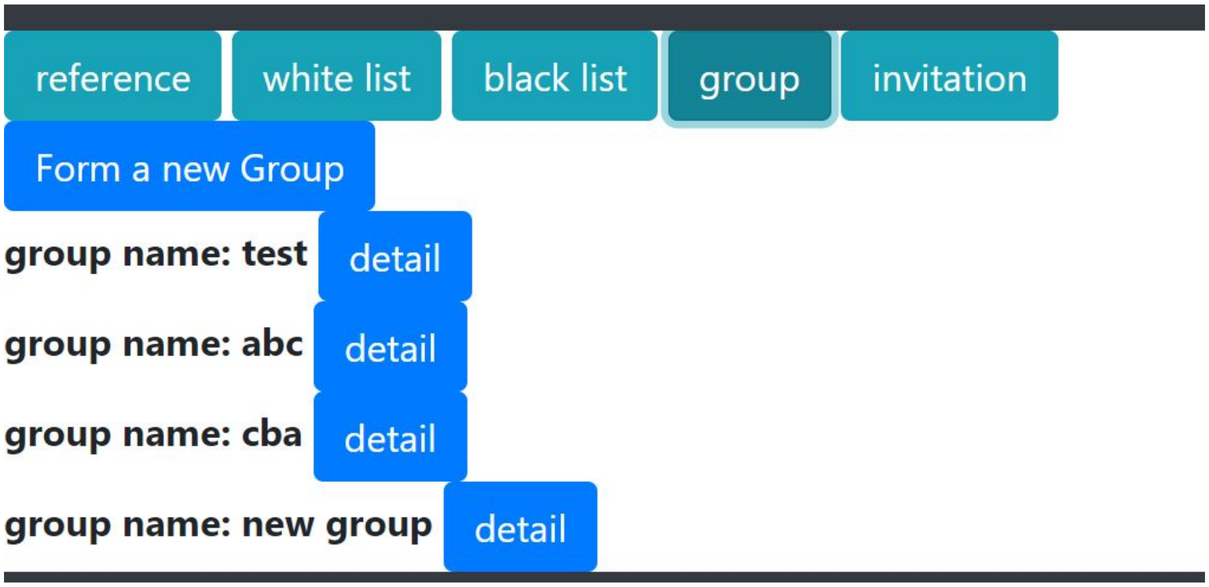
1. Once a OU login to our system, he/she can see a list of navigation buttons. First reference view, he/she can see a list of users that they referred, and gives them an initial score.



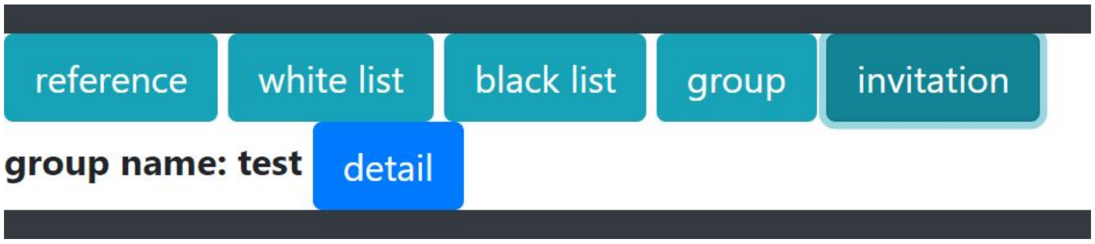
2. For white list view, they can see a list of users that are added to his/her whitelist, and he/she can add more users to his/her whitelist. He/She can also see details of each user and choose further actions including filing a complaint or sending a compliment. The black list has a similar view.



3. For the group view, the user can see a list of groups he/she has joined and the details of each group. He/She can also choose to form a new group and enter the basic information of the new group, and invite friends to join his/her new group.

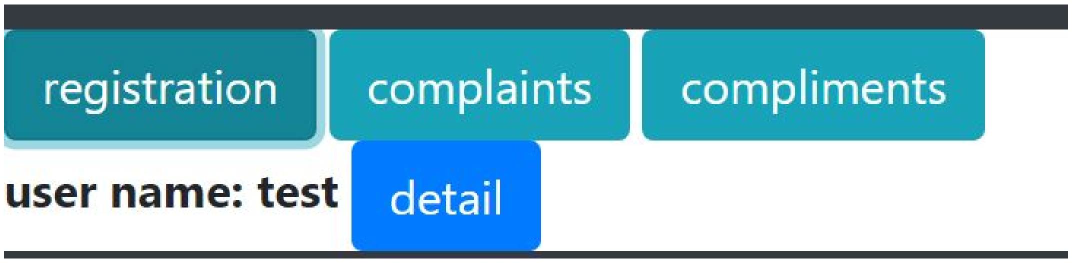


4. For the invitation view, the user can see a list of invitations, and choose accept/reject it.

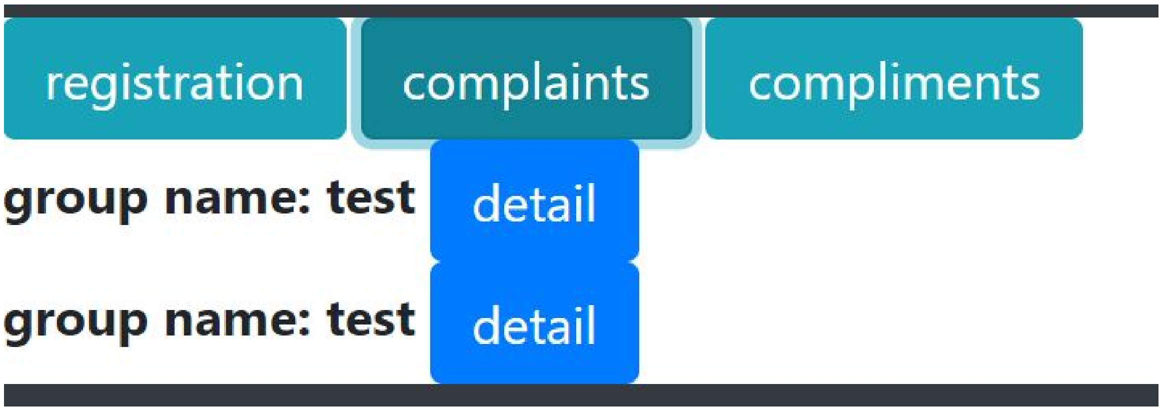


(k) **SU screens**

1. Registration view, the SU can see a list of users who tried to sign up to our system. The SU can check their information and decide to accept the registration or reject it.



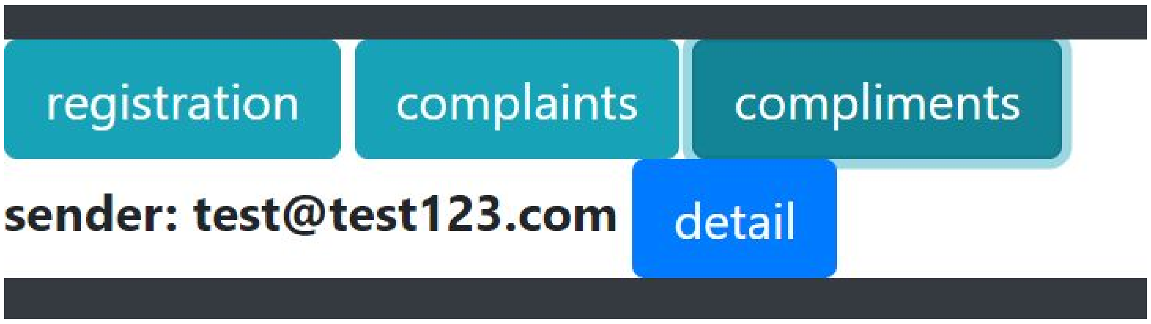
2. Complaints view, the SU can see a list of complaints that are filed by OU/visitors. If the complaint is related to an user, the SU can choose to deducted his/her score or kick him/her out of our system. If the complaint is related to a group, the SU can choose to close the group and punish all the members in this group.



group name: test

group name: test

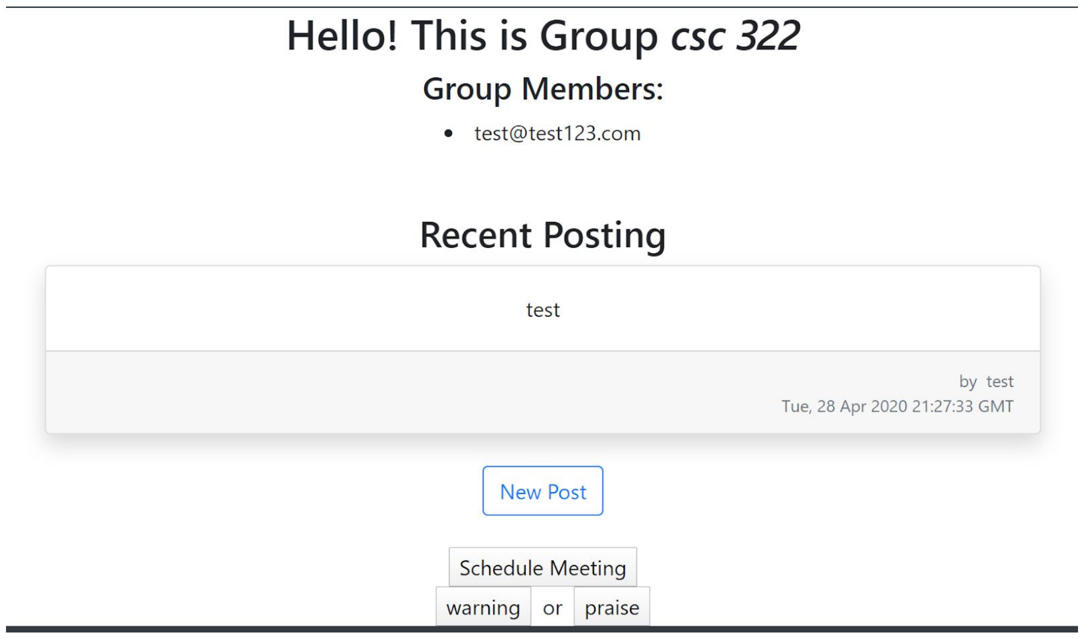
3. Compliments view, the SU can check the sender and receiver of each compliment and choose to reward the receiver or deny it.



sender: test@test123.com

(I) Group screens

For every group created by users, a group page is auto-generated by our system. There are some basic information will be shown at the main page of a group, and all group members can post to it. And group members can also schedule a meeting, vote to issue a warning or a praise to a group member.


Hello! This is Group *csc 322*

Group Members:

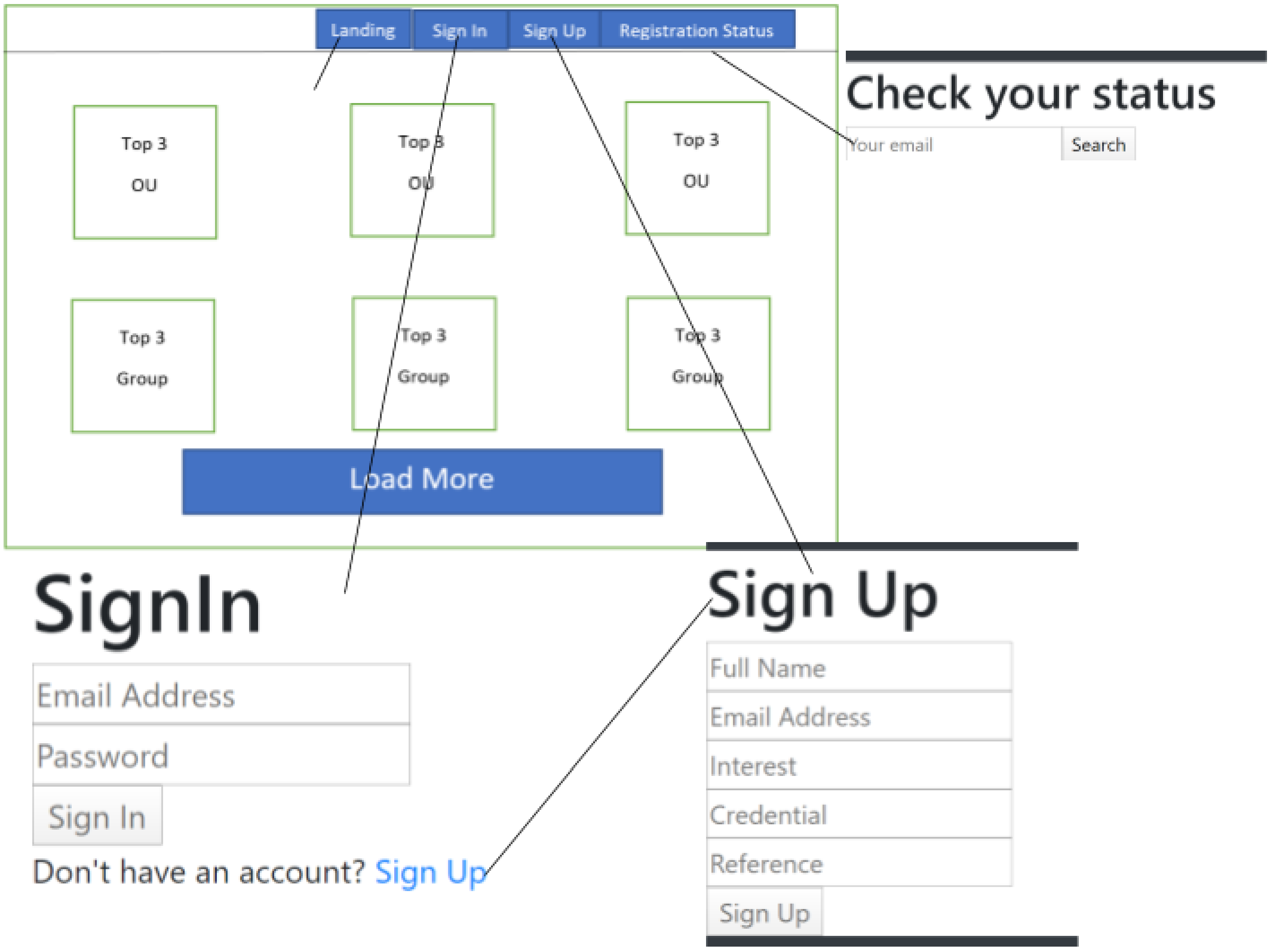
- test@test123.com

Recent Posting

test

by test
Tue, 28 Apr 2020 21:27:33 GMT

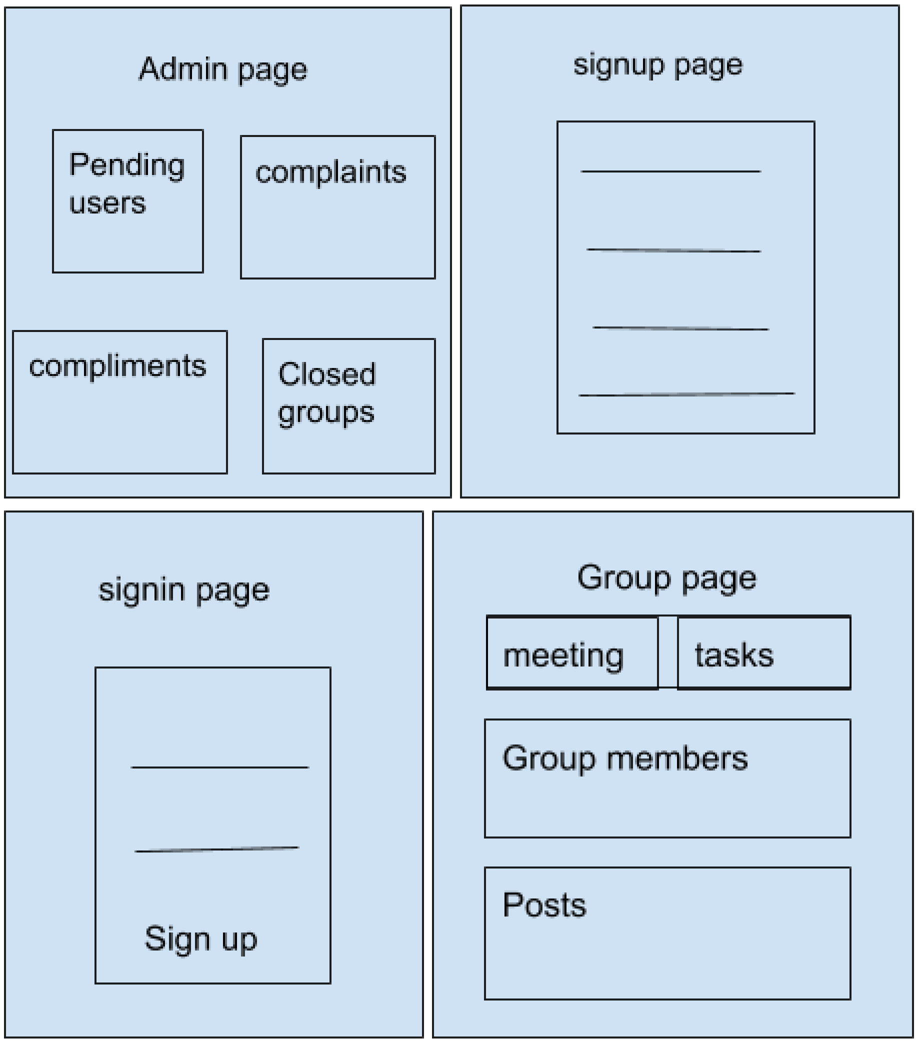
(m) **Prototype of home page**



6. Minutes of Group Meetings

Date: 3/28/20
Agenda:
1. Discuss web framework and the database will be used
2. Come up with some web page sketches by majority.

Actions:
1. Sketched main pages
2. Decided to use React and Bootstrap for front end



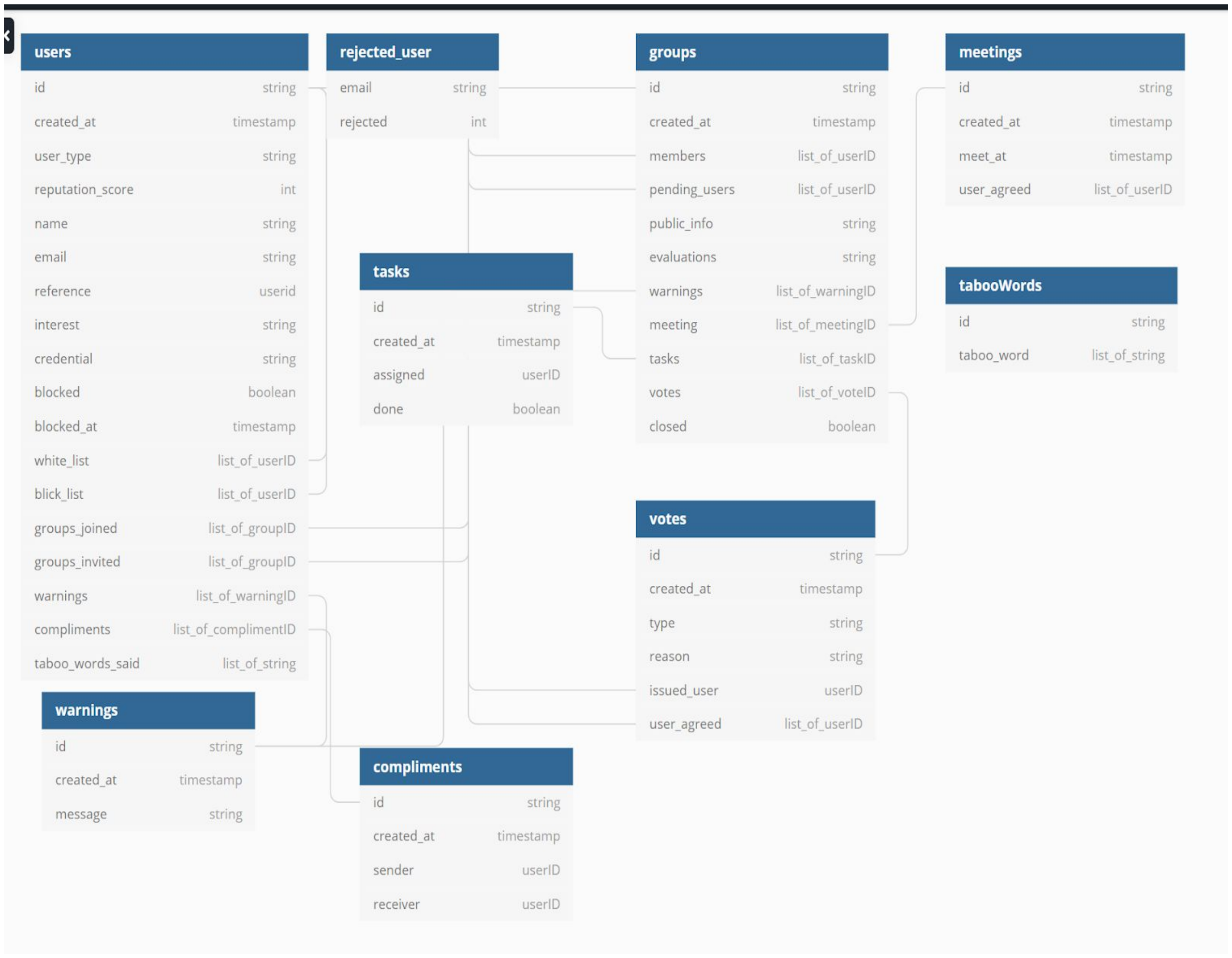
Date: 4/12/20

Agenda:

1. Look at Github repository created by Jose Vargas
2. Determine which data will be stored in Firebase

Actions:

1. Looked at the files in Teaming-System/src/components/Firebase/ written by Bida Chen
2. Organized Firebase by above entities
3. Designed database structure



users	
id	string
created_at	timestamp
user_type	string
reputation_score	int
name	string
email	string
reference	userid
interest	string
credential	string
blocked	boolean
blocked_at	timestamp
white_list	list_of_userID
blick_list	list_of_userID
groups_joined	list_of_groupID
groups_invited	list_of_groupID
warnings	list_of_warningID
compliments	list_of_complimentID
taboo_words_said	list_of_string

rejected_user	
email	string
rejected	int

groups	
id	string
created_at	timestamp
members	list_of_userID
pending_users	list_of_userID
public_info	string
evaluations	string
warnings	list_of_warningID
meeting	list_of_meetingID
tasks	list_of_taskID
votes	list_of_voteID
closed	boolean

meetings	
id	string
created_at	timestamp
meet_at	timestamp
user_agreed	list_of_userID

tasks	
id	string
created_at	timestamp
assigned	userID
done	boolean

tabooWords	
id	string
taboo_word	list_of_string

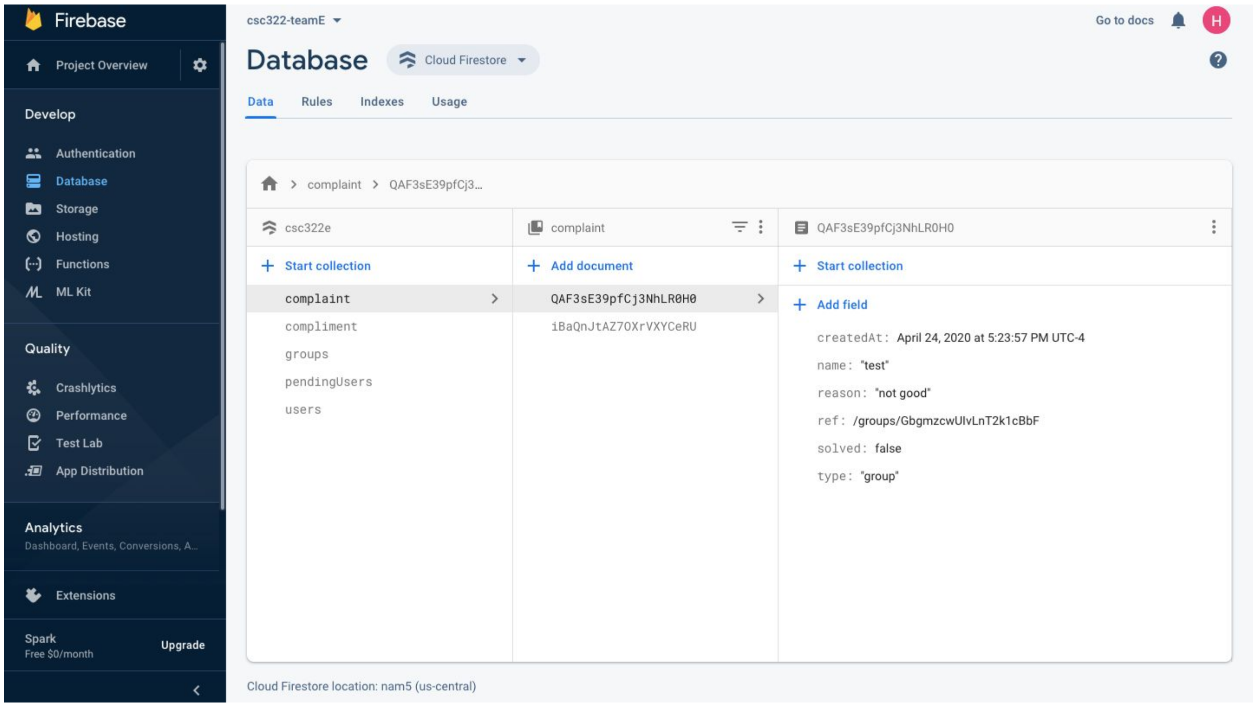
votes	
id	string
created_at	timestamp
type	string
reason	string
issued_user	userID
user_agreed	list_of_userID

warnings	
id	string
created_at	timestamp
message	string

compliments	
id	string
created_at	timestamp
sender	userID
receiver	userID

Decision:

1. Bida Chen is responsible to merge branches
2. Following data will be stored in Firebase



complaint > QAF3sE39pfCj3...

csc322e	complaint	QAF3sE39pfCj3NhLR0H0
complaint	QAF3sE39pfCj3NhLR0H0	createdAt: April 24, 2020 at 5:23:57 PM UTC-4
compliment	iBaQnJtAZ7OXrVXYCeRU	name: "test"
groups		reason: "not good"
pendingUsers		ref: /groups/GbgmzcwUlvLnT2k1cBbF
users		solved: false
		type: "group"

Cloud Firestore location: nam5 (us-central)

Date: 4/26/20

Agenda:

1. Discuss Phase II Report
2. Discuss what diagram to use in Phase II Report

Actions:

1. Learn how to embed data using React/FireBase from various online resources.
2. Split the work for Phase II Report based on different types of diagrams.

Decision:

1. We will be drawing Collaboration Class Diagrams, State Diagrams, and Petri-net Diagrams for each use-cases in the Phase II Report.

7. GitHub Repository

https://github.com/doublejvargas/Teaming-System