**Design Report**

**For**

**SABRE**

**Written by:**

**Mohammad Saif**

**Lucas Lin**

**Shazid Rahman**

**Version 2.0**

**April 29, 2020**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description | Author |
| 03/21/20 | 1.0 | First phase:  Design of the system | Lucas Lin  Shazid Rahman Mohammad Saif |
| 04/29/20 | 2.0 | Second phase:  This report provided data structure and logic to carry out functionality of the app | Lucas Lin  Shazid Rahman Mohammad Saif |

Table of Contents

[1. Introduction](https://docs.google.com/document/d/1yb7mGQNksH6AjO55wuycVZ0ngPajNApGW_1j5NZvIOg/edit#heading=h.7agzol9guif6) 3

2. Use Case Analysis 11

[3.](https://docs.google.com/document/d/1yb7mGQNksH6AjO55wuycVZ0ngPajNApGW_1j5NZvIOg/edit#heading=h.riwhqm19xfbk) Entity Relationship Diagram 30

4. Detailed Design 36

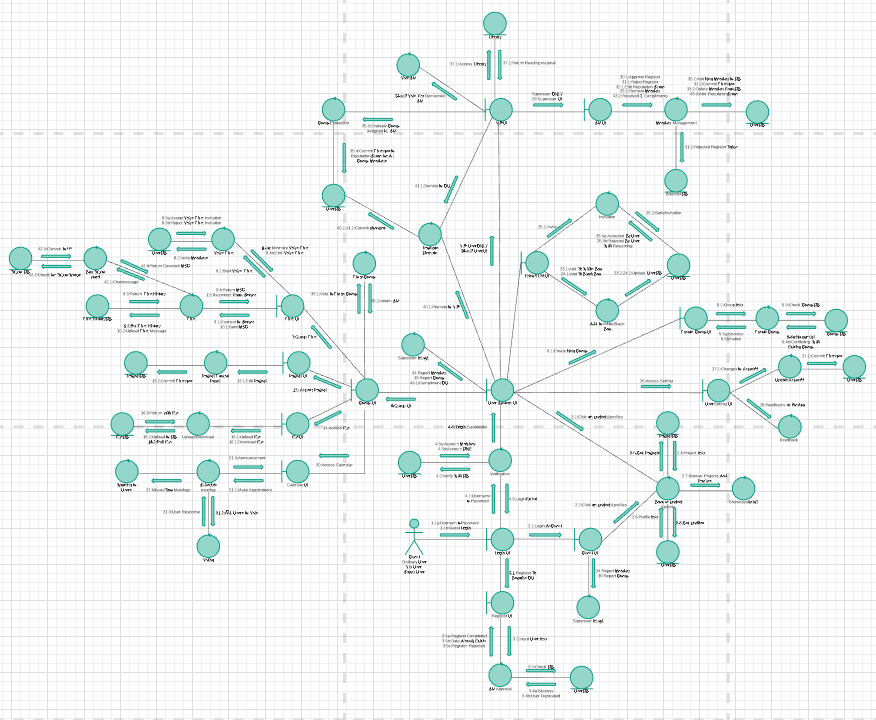
5. System Design 48

6. Meeting Minutes 54

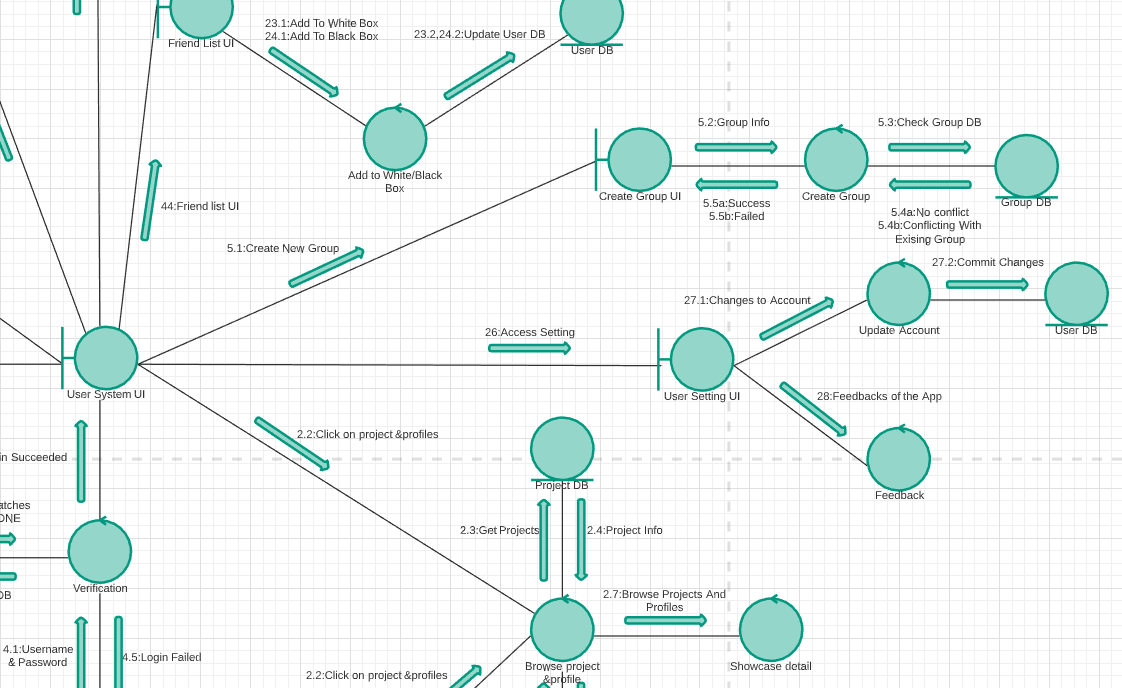
7. Github Repository 55

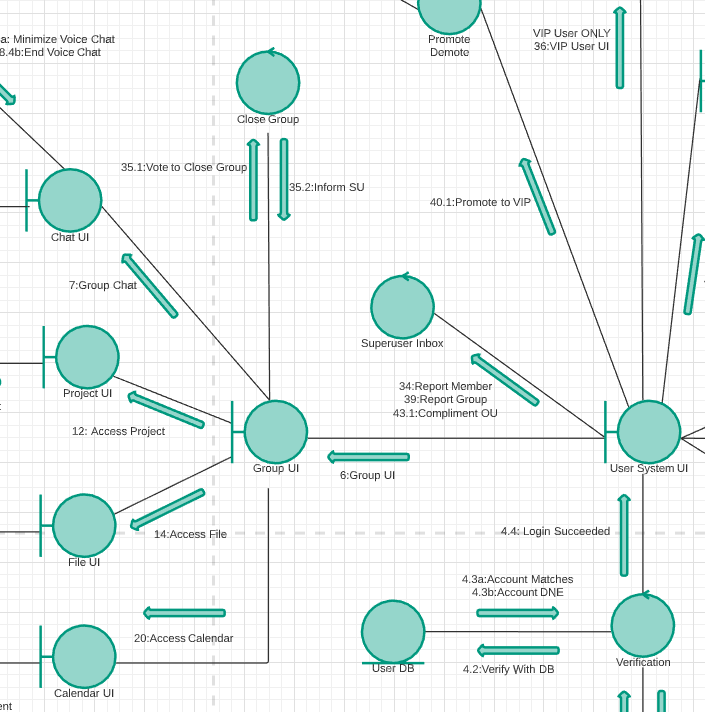
# 1. Introduction

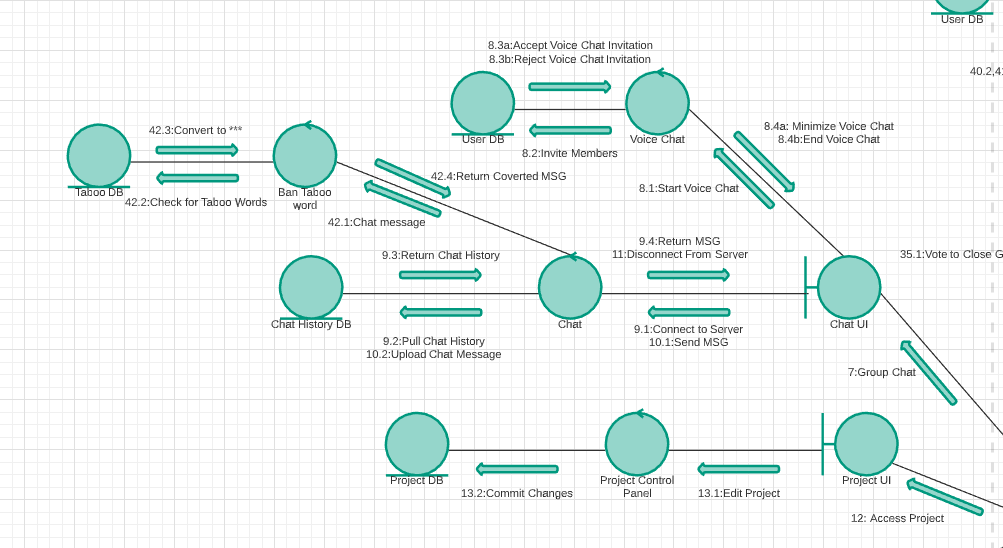
This report gives an overview of the design for the entire study app system which we call Studii. This document is meant to detail the functionalities of the system, and how they are carried out as specified in the previous report. This system will introduce the class diagram which will show how the overall system is supposed to function. Detailed functionality will follow later about the system.

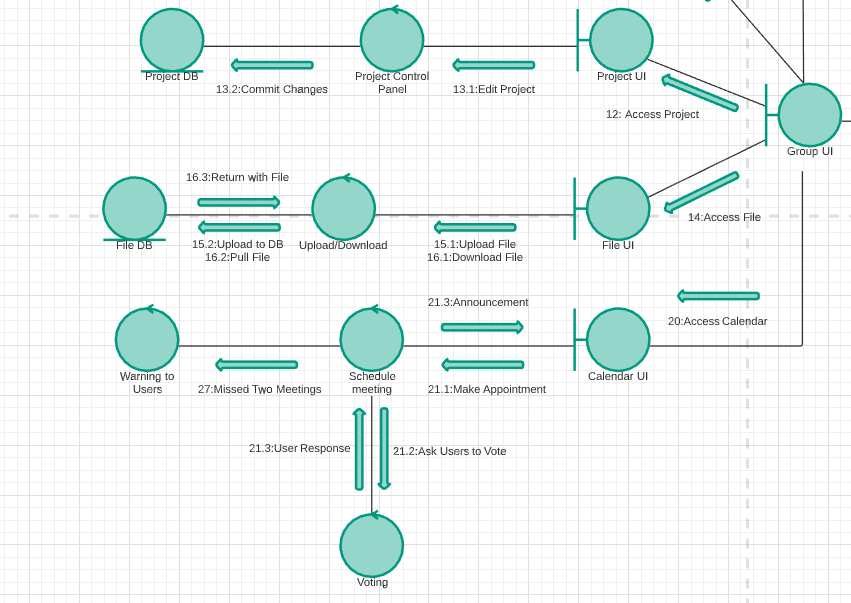


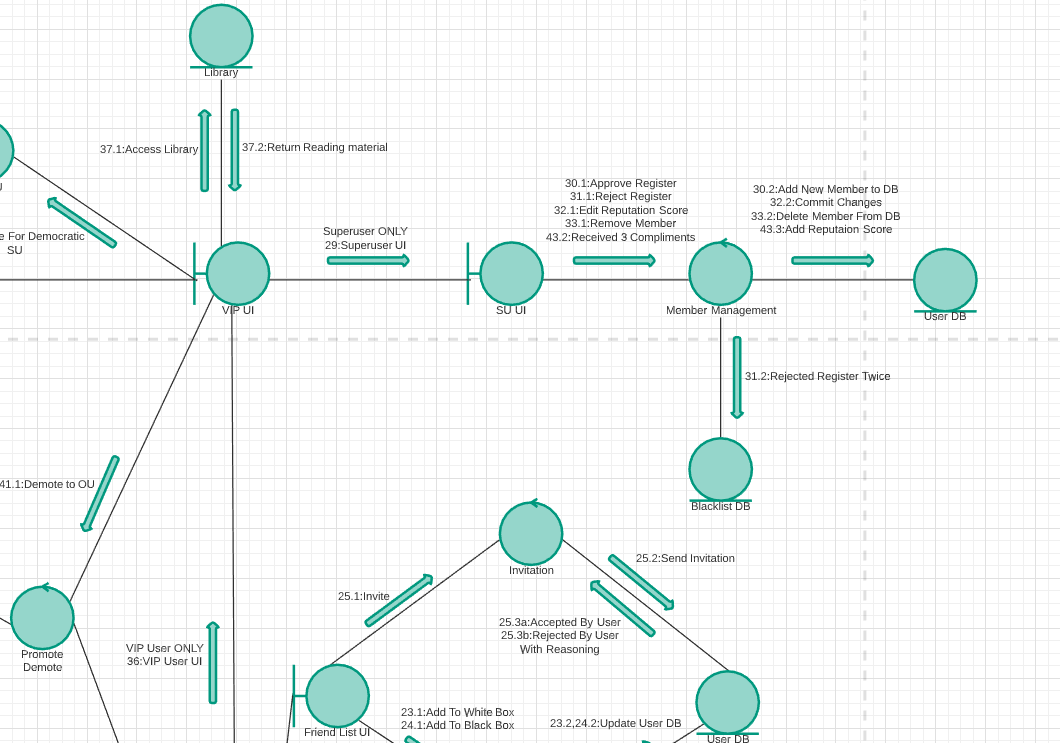
# 

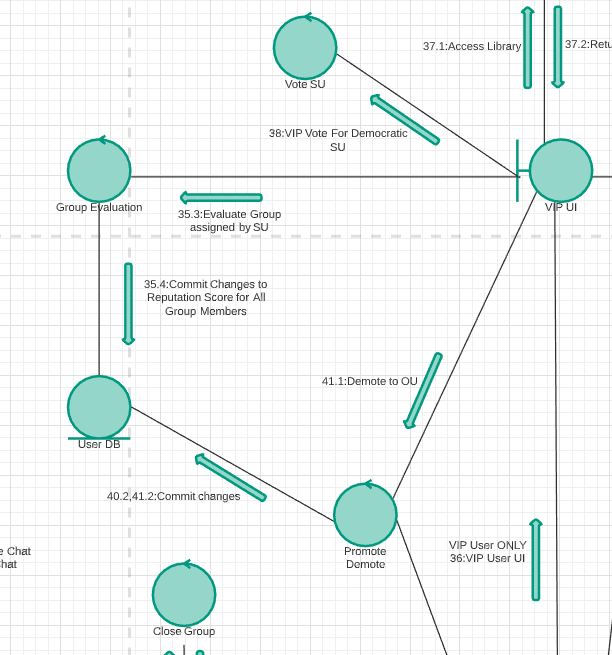












# 

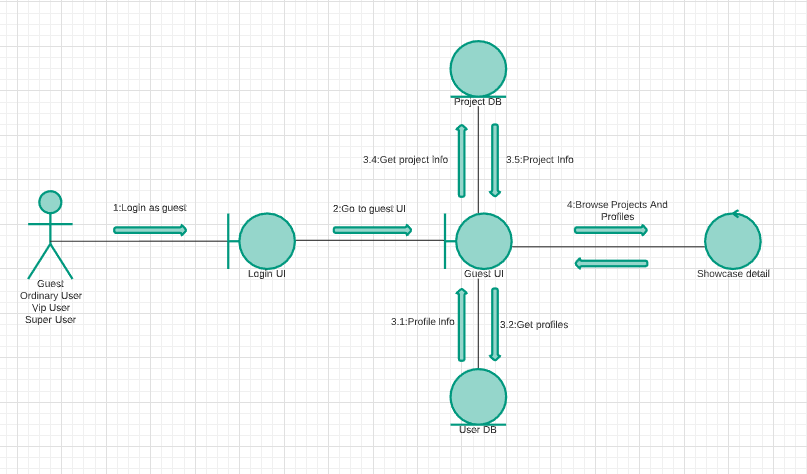
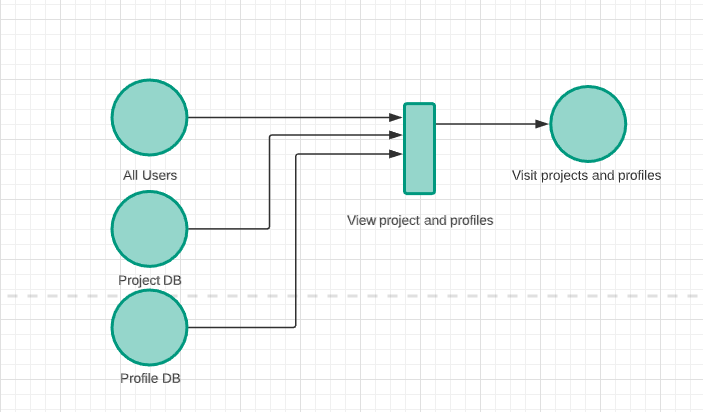
# 2. Use Case Analysis

**Visit top group projects and profiles**

All users can visit public projects and profiles. They are able to view project details and registered users’ profiles.

**Exception**

If all the projects are set to private, Visitors cannot browse any projects.



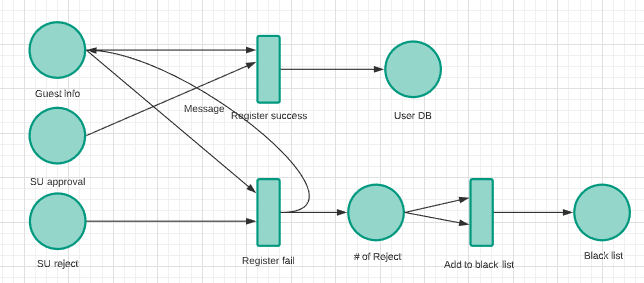
**Register to become an ordinary user**

Only guests/visitors can register to become ordinary users. They can access some of the project and user files but they can’t join any group in the system. After they register, the information will be sent to the superuser of the system. The superuser will decide whether they can join or not.

**Exception**

If the information entered is already in the database, the app will alert the user to reenter the information. If the superuser rejects the user, the user can try one last time to register. If it happens twice, the user will be on the blacklist.



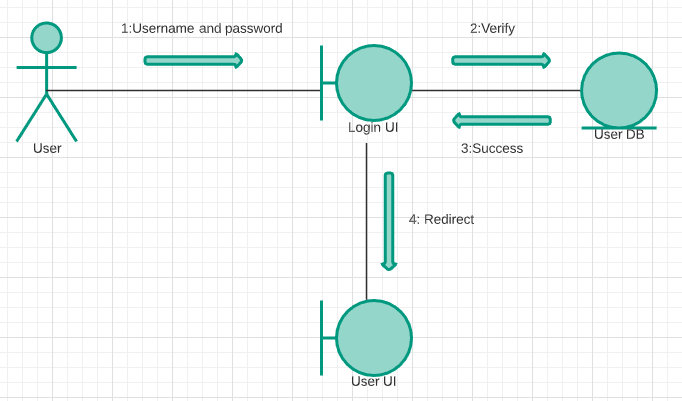


**Log in to the System**

After the user provides the username and password, the app will check if it matches the database. If it succeeds the app will jump to the user system UI.

**Exception**

If a user tries to login in with the wrong username and password, the app will not log in and warn the user.

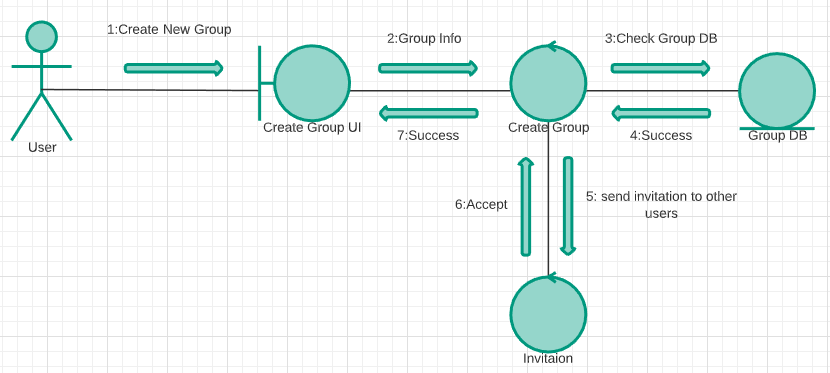


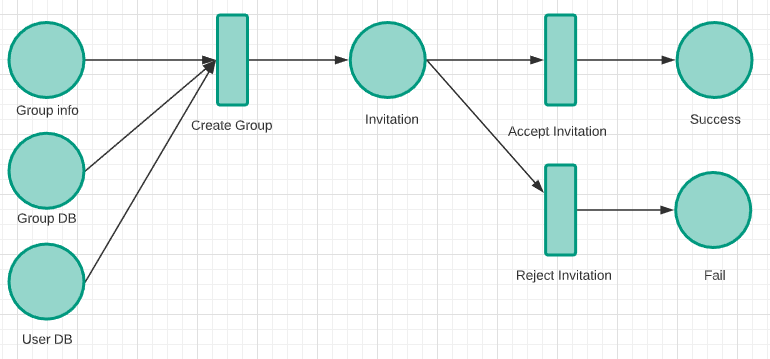
**Create a new group**

After the user is accepted to the system, they can form a group by inviting other people to join in. All the users who received an invitation are required to make a choice of yes or no.

**Exception**

If no one accepts the invitation, the group will not form.



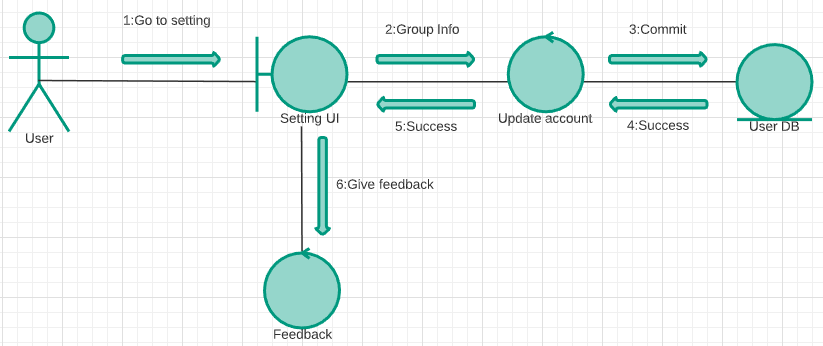


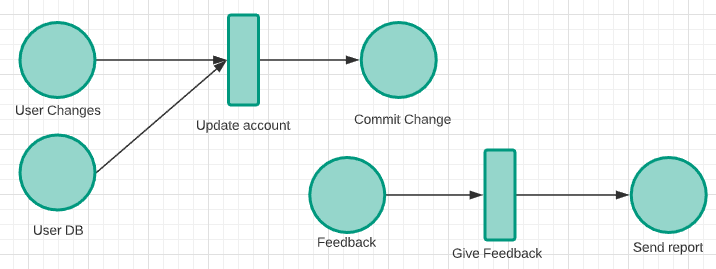
**Setting and feedback**

Users can go to settings and update their account including profile, and account password. They can also click on the feedback button to give feedback on the system.

**Exception**

If the user is under a bad WIFI condition, the changes made may not be saved in the system.



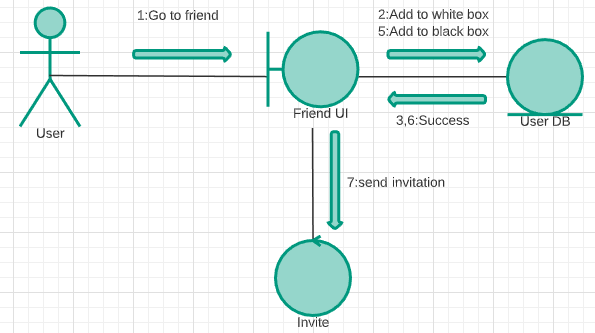


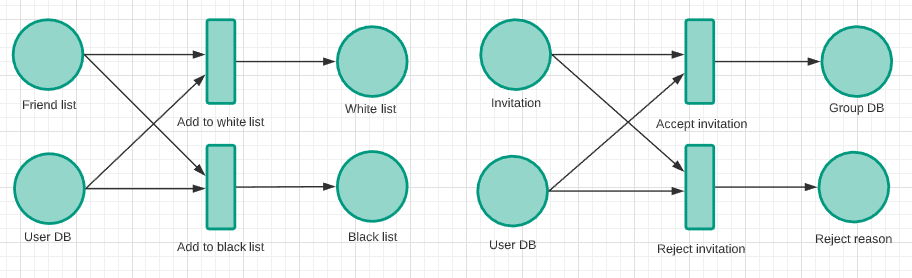
**Friend list**

Users can add other users as friends. They can add them to the white box and black box. If someone is in the black box, their invitation will be rejected automatically. Users that are in the white box can invite them to a group and they can either accept it or reject it for a reason.

**Exception**

If a member in the group is kicked out by the superuser, other users will not be able to add them to any box.



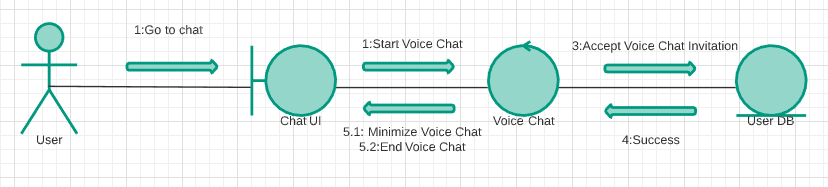


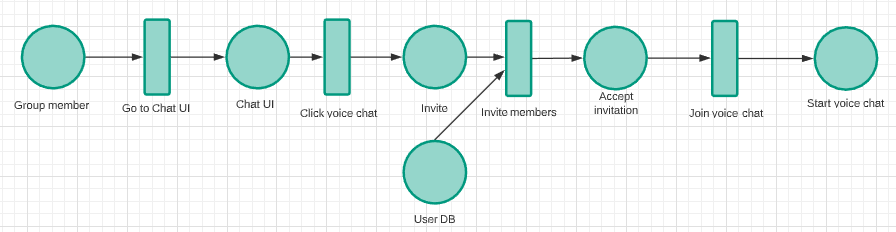
**Voice chat**

Users can invite members in the same group to voice chat. Invited members can accept or reject it.

**Exception**

The user will be disconnected if the WIFI is terrible or they are no longer in the group.



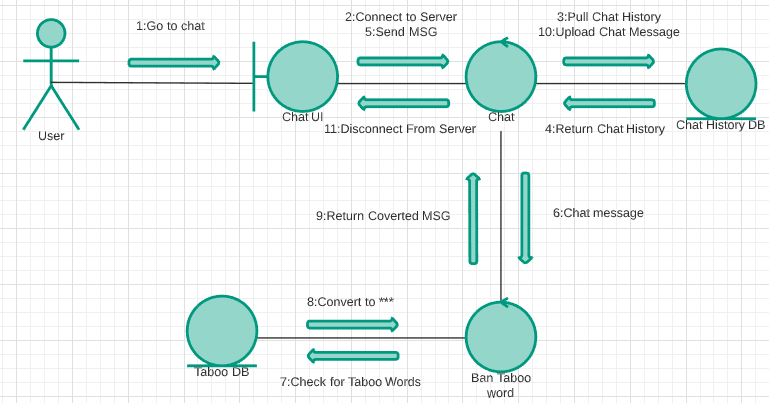
****

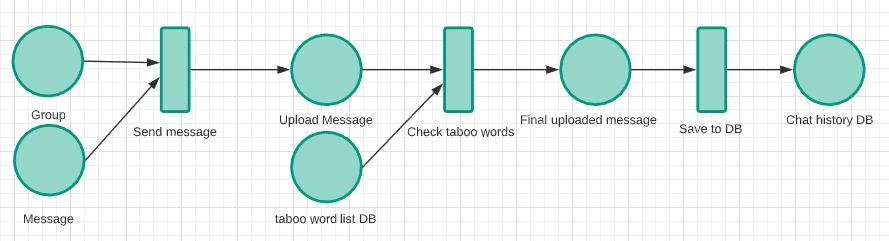
**Group Chat**

Users in the same group are able to send messages to each other. The message will be compared with the list of taboo words.

**Exception**

There may be cases when the user can’t get chat history or send a message due to bad WIFI conditions. If they’re removed from the group, the history in this group will be deleted for them automatically.



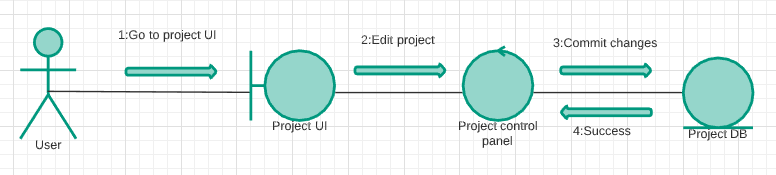


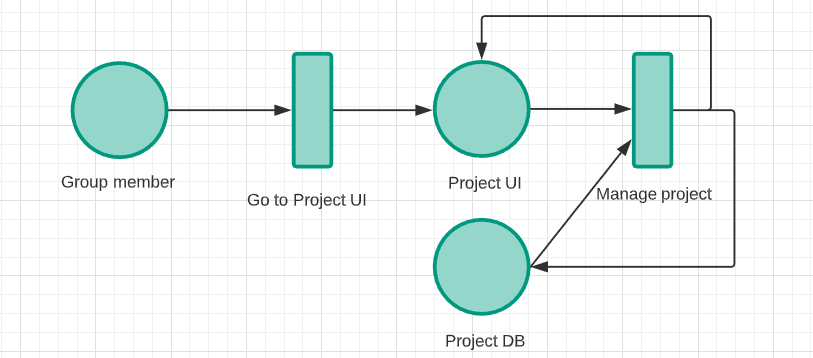
**Projects**

Users are able to create, delete, and edit projects in the group. Projects can be public to everyone or private to group members only. All the changes will be saved in the database afterward.

**Exception**

If the project is deleted by accident, The files will not be in the database anymore.



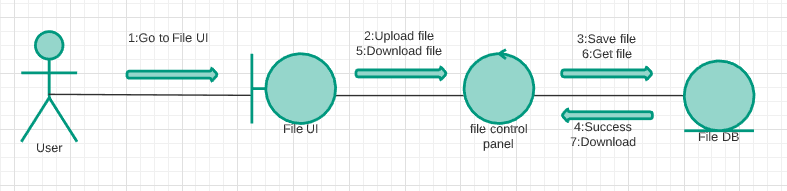


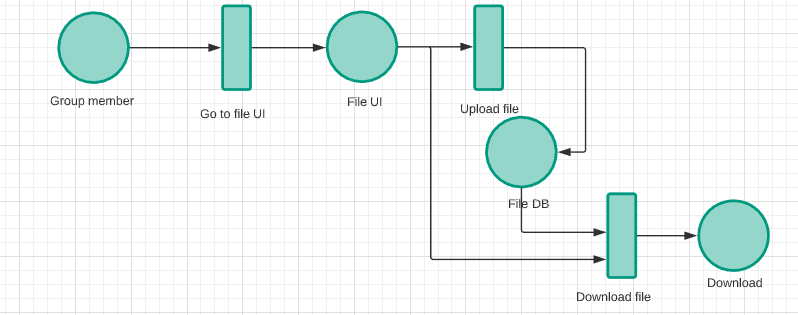
**Share File**

Members of the group can share any project relevant, reference, guide files with other members by uploading them to the database. All the group members are able to download these files for any purpose. When uploading files, tags are required for a more efficient sort.

**Exception**

When the files are uploaded, that means everyone can download it. If the wrong file is uploaded to the database, it is hard to trace and delete them.



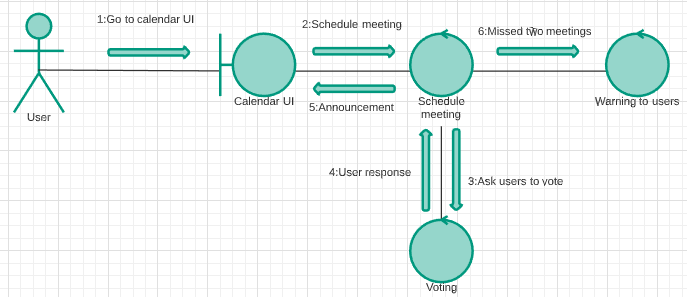


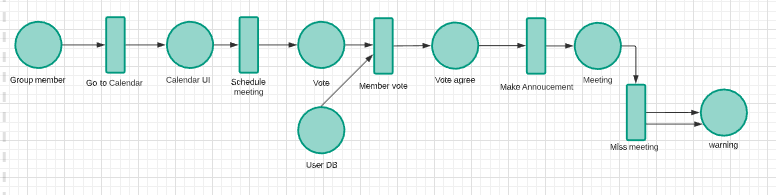
**Calendar**

For each group there will be a calendar for group meetings. Anyone in the group may schedule a meeting with other members. Once other members accept the schedule, the app will make an announcement to the whole group about the meeting. The announcement should include time, format, location if it is a physical meeting. If a member missed two meetings, they will get a warning.

**Exception**

If the time of the meeting is set to a time that is past, the meeting should no longer be valid and will not count as missed meetings.



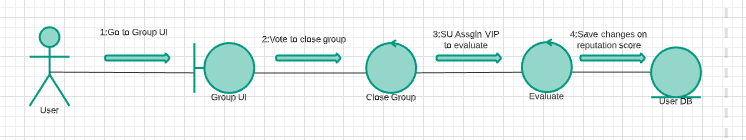


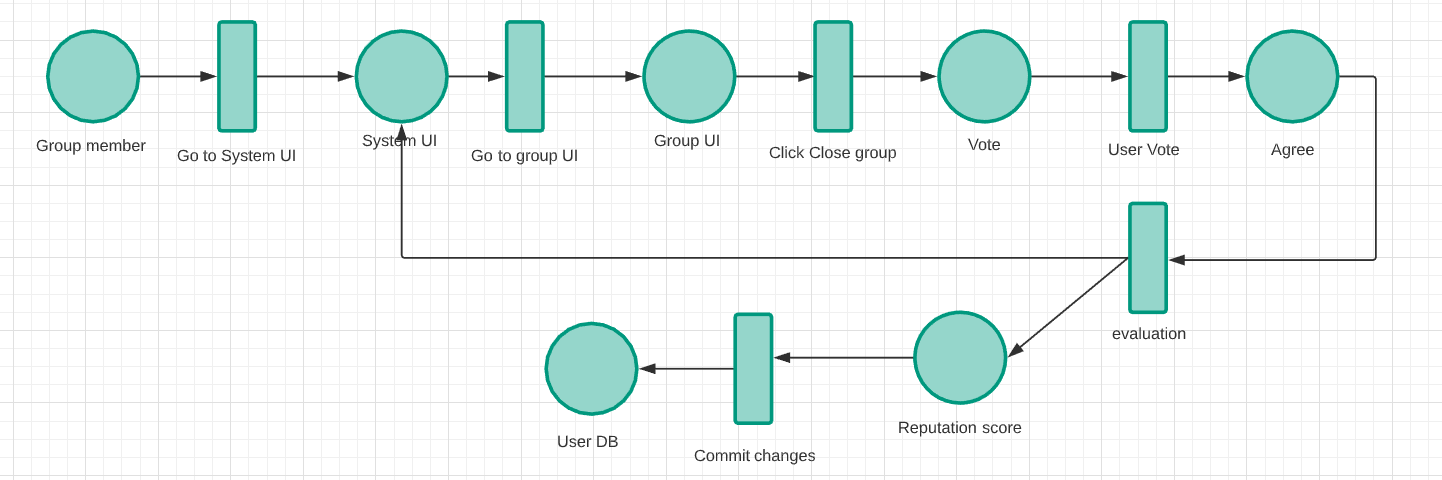
**Close group**

Members of the group can vote to close the group. When everybody agrees to close the group, one of the super users in the system will assign a VIP user to evaluate the group. The score that the VIP user gives will be the same for everyone in the group.

**Exception**

If the group is closed and the VIP user did not evaluate the group on time, the group members will need to wait for the VIP user to evaluate to get the reputation scores.



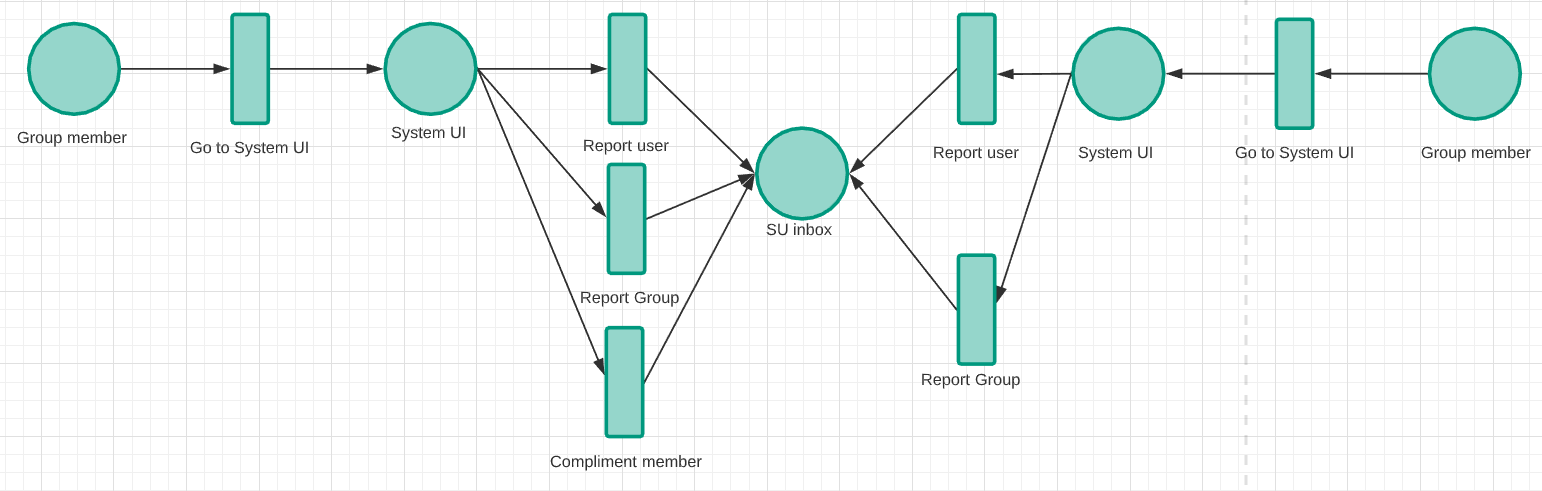


**Report/Compliment**

Users can report a group or a member for bad behaviors to the superusers. The reported message will be sent to super users’ inbox via the system which will not be a private conversation between the reporter and the superuser. Users can compliment other users via the superuser inbox. If any user got 3 compliments, their reputation score would increase automatically.

**Exception**

There is no limit on the number of reports and compliments from one user yet which can cause overload for the superusers.

****

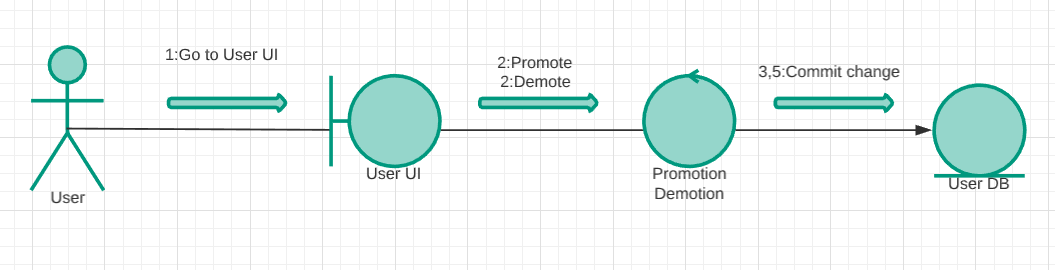
**USER: VIP**

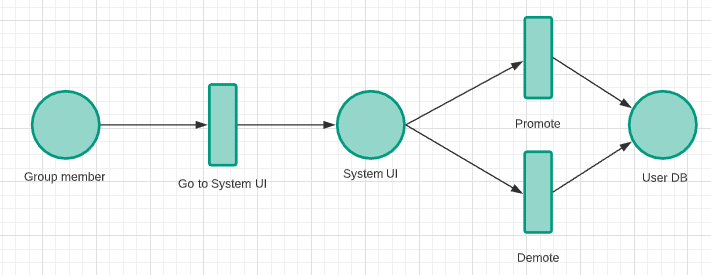
**Promotion and Demotion**

Ordinary users with a reputation score of 30 or more will be promoted to VIP users automatically. VIP users will be demoted if their reputation score is below 25.

**Exception**

Users with a reputation score under 30 and above 25 will remain in their hierarchy until their score break either upper limit or lower limit.



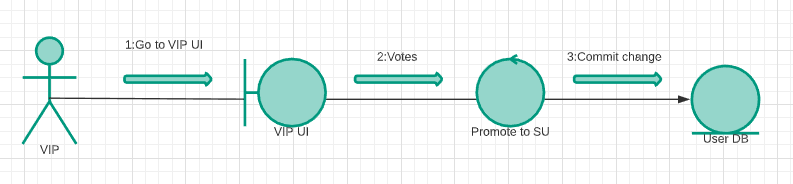


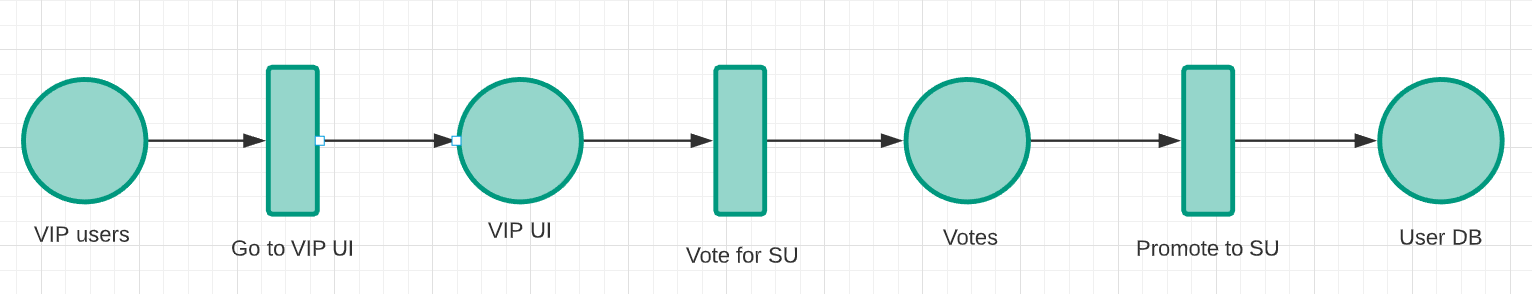
**Vote for democratic superuser**

All the VIP users in the system can vote for a democratic superuser among all the VIP users.

**Exception**

Two users may end up with the same amount of vote which will make the vote invalid and restart the vote.



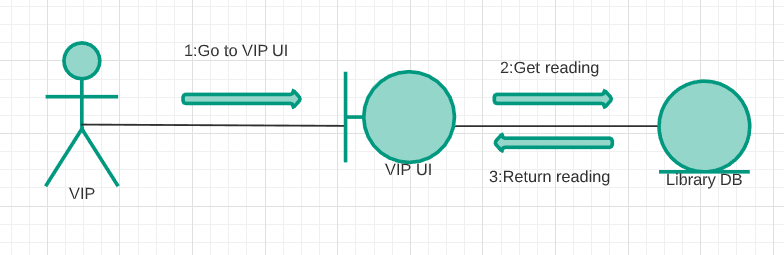


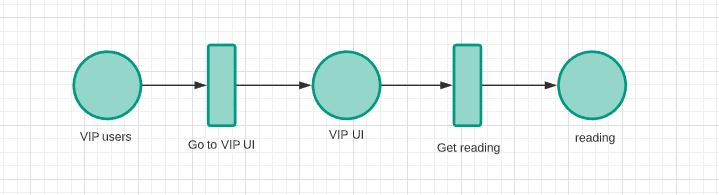
**Access library**

The library is used for storing any useful books, textbooks, articles, journals, and etc. The purpose of having the library is to allow users to acquire reference documents, textbooks, and other necessary reading easier.

**Exception**

Only VIP and superusers are allowed to access the library. The library may not have the reading which the user is looking for.





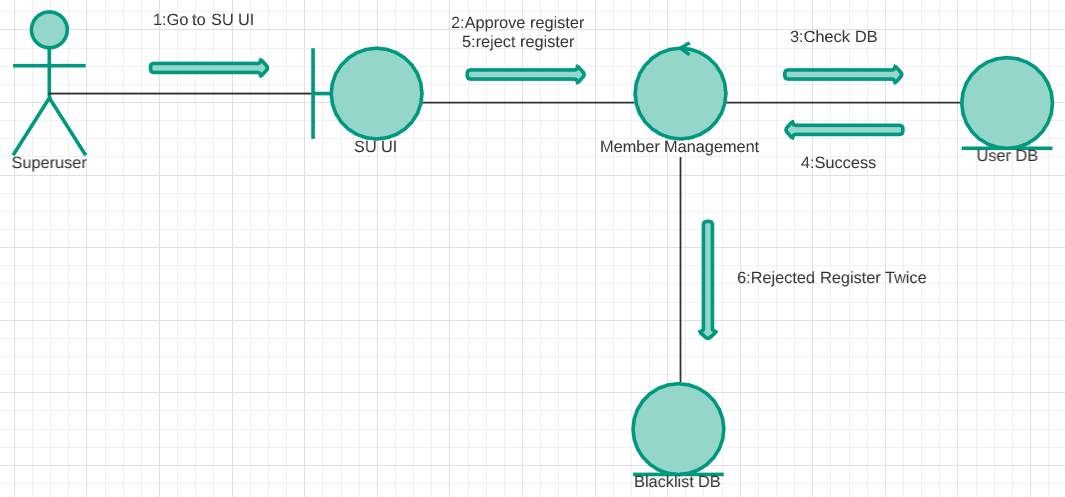
**USER: Superuser**

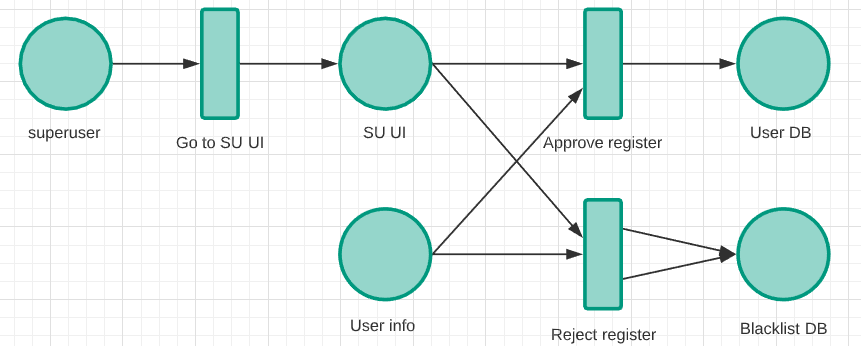
**Approve registers**

Superusers have the ability to approve visitors’ register but also reject their registers. Once the superuser approves the register, the visitor will become an ordinary user in the system. If the superuser rejects the register once, s/he still got one more chance to become an ordinary user. However, if s/he gets rejected again, s/he will have no chance of becoming an ordinary user.

**Exception**

If the user information is conflicting with the database, the registration will fail and the visitor has to go through the process again.



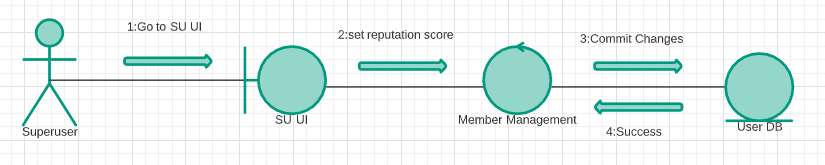


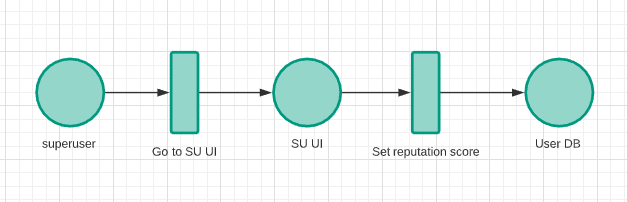
**Reputation score**

Superusers have the right to set and adjust anyone’ s reputation score.

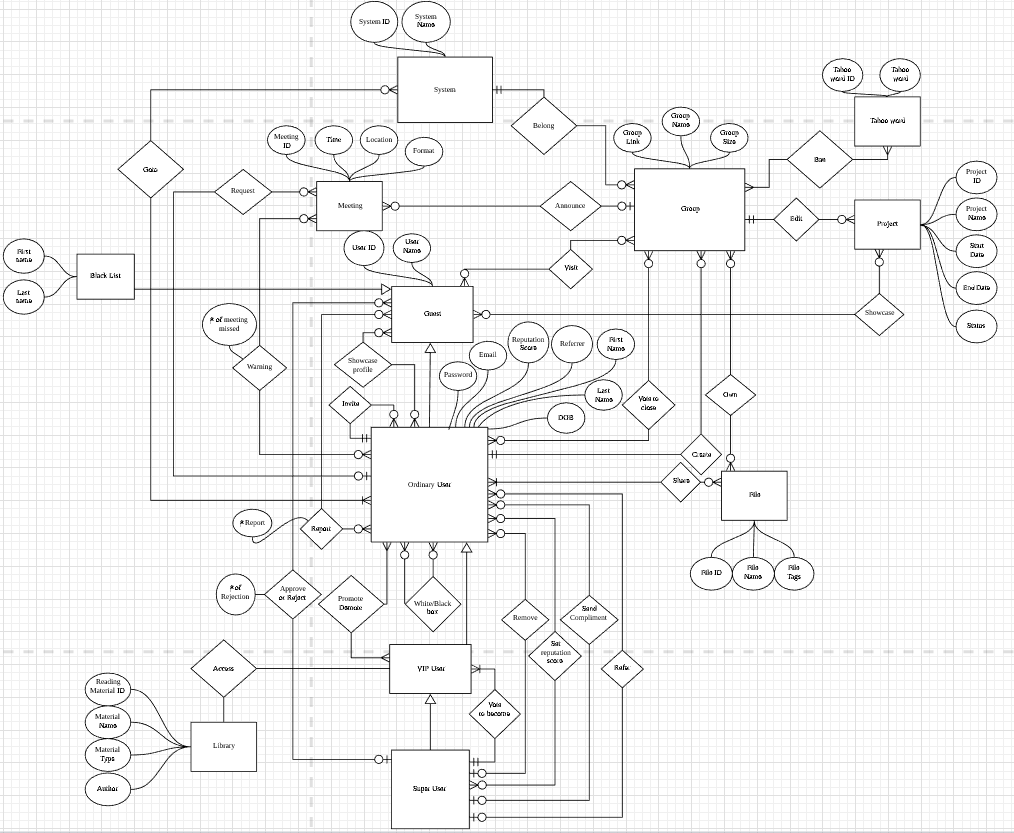
**Exception**

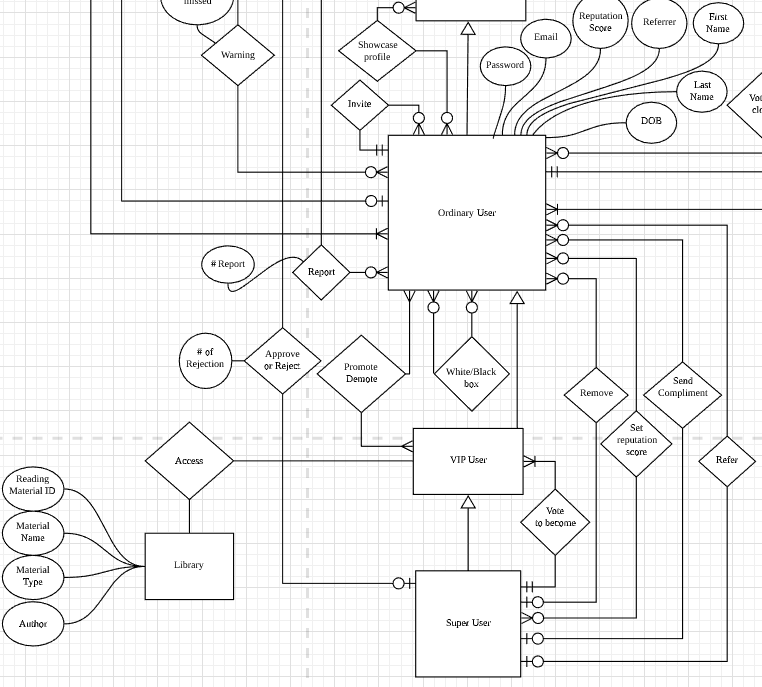
Superusers can only change the reputation score of a user within the system s/he is managing.

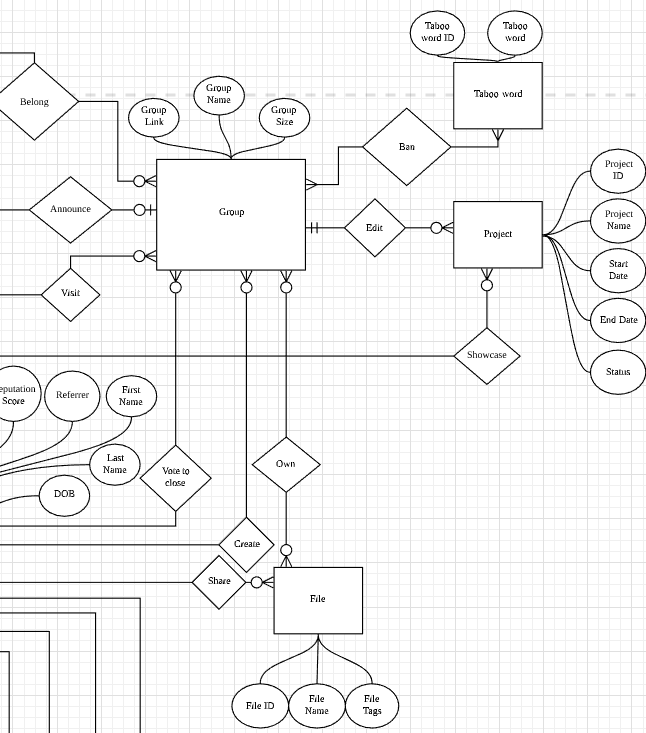


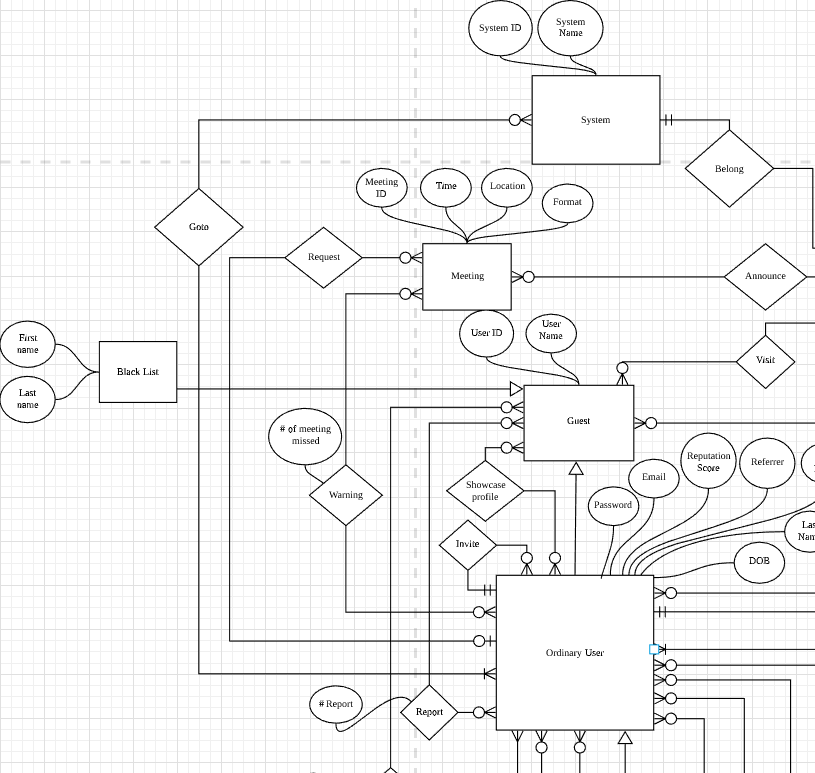


3. Entity Relational Diagram









Guest:

Attribute:

* User ID
* Username

Key: user ID

Blacklist:

Attribute:

* User ID
* Username
* First name
* Last name

Key: User ID

Ordinary user/VIP user/Superuser:

Attribute:

* User ID
* Username
* Password
* First name
* Last Name
* DOB
* Email
* Reputation score
* Referrer

Key:User ID

System:

Attribute:

* System ID
* System Name

Key:System ID

Group:

Attribute:

* Group name
* Group size
* Group link

Key: Group link

Project:

Attribute:

* Project ID
* Project name
* Start date
* End date
* Status

Key: Project ID

File:

Attribute:

* File ID
* File name
* File tag

Key: File ID

Meeting:

Attribute:

* Meeting ID
* Time
* Format
* Location

Key: Meeting ID

Library:

Attribute:

* Reading material ID
* Material name
* Author
* Type

Key: Reading material ID

Taboo word:

Attribute:

* Taboo word ID
* Taboo word

Key: Taboo word ID

# 

# 

# 

# 

# 4. Detailed Design

**1. signInExistingUser(String email, String password)**

Input:Users email and password

Output: Authentication success or fail

Description: Sign in preexisting user

public void signInExistingUser(String email, String password) {  
mAuth.signInWithEmailAndPassword(email, password)  
 .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {  
 @Override  
 public void onComplete(@NonNull Task<AuthResult> task) {  
 if (task.isSuccessful())   
 DocumentReference docRef = db.collection("Users").document(email);  
 docRef.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {  
 @Override  
 public void onComplete(@NonNull Task<DocumentSnapshot> task) {  
 if (task.isSuccessful()) {  
 DocumentSnapshot document = task.getResult();  
 if (document.exists()) {  
 //checking if its first time login  
 if (document.get("Login").equals("1")) {  
 Intent intent = new Intent(getApplicationContext(), firstTimelogin.class);  
 intent.putExtra("Email", email);  
 startActivity(intent);  
 }else{  
 Intent intent = new Intent(getApplicationContext(), AuthenticateduserActivity.class);  
 startActivity(intent);  
 }  
 //third task  
 } else {  
  
 }  
 //second task  
 } else {  
 Toast.*makeText*(getApplicationContext(), "No Internet Connection failed to retrieve data", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 });  
 //first task  
 } else {  
 Toast.*makeText*(getApplicationContext(), "Invalid Credentials", Toast.*LENGTH\_SHORT*).show();  
 }  
  
 }  
 });  
}

**2. resetPasswordLink()**

Input: Email

Output: Email containing link to reset password

Description: Reset password via email link

sendButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 String email=enterEmail.getText().toString();  
 String repeatemail=repeatEmail.getText().toString();  
 if(!email.isEmpty() && repeatemail.equals(email)){  
 FirebaseAuth auth = FirebaseAuth.*getInstance*();  
  
 auth.sendPasswordResetEmail(email)  
 .addOnCompleteListener(new OnCompleteListener<Void>() {  
 @Override  
 public void onComplete(@NonNull Task<Void> task) {  
 if (task.isSuccessful()) { Toast.*makeText*(authenticatedUserForgotPassword.this, "Sent a reset password links to " +  
 "your email", Toast.*LENGTH\_SHORT*).show();  
 finish();  
 }  
 }  
 }).addOnFailureListener(new OnFailureListener() {  
 @Override  
 public void onFailure(@NonNull Exception e) {  
 Toast.*makeText*(authenticatedUserForgotPassword.this, "Invalid email address", Toast.*LENGTH\_SHORT*).show();  
 }  
 });  
 }else{  
 Toast.*makeText*(authenticatedUserForgotPassword.this, "Please enter email in appropriate manner and " +  
 "make sure they match", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
});

**3. sendData()**

Input: Email, password, reference, interest, credential

Output: Database created in the firebase console

Description: Creates database for users registration request which can be viewed by the Super User.

public void sendData(String Name, String Email, String Interest, String Credential, String Reference) {  
 Map<String, Object> register = new HashMap<>();  
 register.put("Name", Name);  
 register.put("Email", Email);  
 register.put("Interest", Interest);  
 register.put("Credential", Credential);  
 register.put("Reference", Reference);  
  
  
 db.collection("Registration").document(Email)  
 .set(register)  
 .addOnSuccessListener(new OnSuccessListener<Void>() {  
 @Override  
 public void onSuccess(Void aVoid) {  
 Log.*d*("PASS", "DocumentSnapshot successfully written!");  
 }  
 })  
 .addOnFailureListener(new OnFailureListener() {  
 @Override  
 public void onFailure(@NonNull Exception e) {  
 Log.*w*("FAIL", "Error writing document", e);  
 }  
 });  
}

**4. signOut()**

Input: Menu option clicked tagged sign out

Output: User taken back to login page

Description: Users gets logged out from the app

public void signOut(){  
 FirebaseAuth.getInstance().signOut();  
}

**5. getCurrentUser()**

Input: None

Output: Current user email address

Description: Get currents users credential

public void getCurrentUser(){

if (user != null) {  
 currentUser = user.getEmail().toUpperCase().trim();

}

**6. registrationSubmit()**

Input: Button click submit

Output: A dialog message to confirm to proceed

Description: User submit their credential for the super user to review

public void registrationSubmit(View view) {  
  
 boolean check = true;  
 name = Name.getText().toString();  
 if (name.trim().isEmpty()) {  
 check = false;  
 }  
 email = Email.getText().toString().toUpperCase().trim();  
 if (email.trim().isEmpty()) {  
 check = false;  
 }  
 interest = Interest.getText().toString();  
 if (interest.trim().isEmpty()) {  
 check = false;  
 }  
 credential = Credential.getText().toString();  
 if (credential.trim().isEmpty()) {  
 check = false;  
 }  
 reference = Reference.getText().toString();  
 if (reference.trim().isEmpty()) {  
 check = false;  
 }  
  
 if (!check) {  
 Toast.*makeText*(getApplicationContext(), "Fill in all the fields", Toast.*LENGTH\_SHORT*).show();  
 } else {  
 new AlertDialog.Builder(this)  
 .setIcon(android.R.drawable.*ic\_dialog\_alert*)  
 .setTitle("Are you sure you want submit?")  
 .setMessage("Only proceed if all your information is correct.Super user will review your information and email you if he excepted your information or not ")  
 .setPositiveButton("Yes", new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which) {  
 sendData(name, email, interest, credential, reference);  
 finish();  
 }  
 })  
 .setNegativeButton("No", new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which) {  
 Toast.*makeText*(getApplicationContext(), "You can fix your information and submit", Toast.*LENGTH\_SHORT*).show();  
 }  
 }).show();  
  
 }  
  
}

**7. listView.setOnItemClickListener()**

Input: User clicks on the list

Output: Clicked line’s info will be displayed in a new screen

Description: User’s will get the info by clicking on certain line

listof.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
 @Override  
 public void onItemClick(AdapterView<?> parent, View view, int position, long id) {  
 String text = (listof.getItemAtPosition(position)).toString();  
 String[] parts = text.split("\n");  
 String part1 = parts[1];  
 Intent intent = new Intent(getActivity(), ReviewPage.class);  
 intent.putExtra("Email", part1);  
 startActivity(intent);  
  
 }  
});

**8. sendEmail(String to, String Subject, String text)**

Input: destination email, subject of email and the message

Output: Open the default email app on the phone

Description: This opens the email app on the phone with predefined text, user, and send to.

protected void sendEmail(String To, String Subject, String text) {  
 String[] TO = {To};  
 String[] CC = {""};  
 Intent emailIntent = new Intent(Intent.*ACTION\_SEND*);  
  
 emailIntent.setData(Uri.*parse*("mailto:"));  
 emailIntent.setType("text/plain");  
 emailIntent.putExtra(Intent.*EXTRA\_EMAIL*, TO);  
 emailIntent.putExtra(Intent.*EXTRA\_CC*, CC);  
 emailIntent.putExtra(Intent.*EXTRA\_SUBJECT*, Subject);  
 emailIntent.putExtra(Intent.*EXTRA\_TEXT*, text);  
  
 try {  
 startActivity(Intent.*createChooser*(emailIntent, "Send mail..."));  
  
  
 finish();  
  
 } catch (android.content.ActivityNotFoundException ex) {  
 Toast.*makeText*(ReviewPage.this, "There is no email client installed.", Toast.*LENGTH\_SHORT*).show();  
 }  
}

**9. sendDenyorApprove()**

Input: Button Click Deny or Approve

Output: Email app with predefined message whether denied or approved

Description: Super user can deny or approve a registration just by clicking deny or approve button. The users account gets created and

public void send(View view) {  
 int user\_point = Integer.*parseInt*( point.getText().toString());  
 switch (view.getId()) {  
 case R.id.*approveButton*:  
  
 String password = new Random().ints(7, 33, 122).collect(StringBuilder::new,  
 StringBuilder::appendCodePoint, StringBuilder::append)  
 .toString();  
  
 result = "Congratulations "+ Name +"! You are approved. Welcome to the SABRE community. You will start"  
 + " with initial point of " + user\_point + ". This is your initial password " + password +  
 ". You will have to change the password when you login for the first time. Have a productive time" +  
 "\n Sincerely, \n Samin Saif";  
  
 subject = "Registration approved";  
 // Initialize Firebase Auth  
 mAuth = FirebaseAuth.*getInstance*();  
 mAuth.createUserWithEmailAndPassword(mailto, password)  
 .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {  
 @Override  
 public void onComplete(@NonNull Task<AuthResult> task) {  
 if (task.isSuccessful()) {  
 db.collection("Registration").document(Email)  
 .delete()  
 .addOnSuccessListener(new OnSuccessListener<Void>() {  
 @Override  
 public void onSuccess(Void aVoid) {  
 Log.*d*("Success", "DocumentSnapshot successfully deleted!");  
 }  
 })  
 .addOnFailureListener(new OnFailureListener() {  
 @Override  
 public void onFailure(@NonNull Exception e) {  
 Log.*w*("Failed", "Error deleting document", e);  
 }  
 });  
 // Sign in success, update UI with the signed-in user's information  
 Log.*i*("SUCCESS!", "createUserWithEmail:success");  
 // FirebaseUser user = mAuth.getCurrentUser();  
 sendData(Name, mailto, Interest, Credential, Reference, user\_point);  
 sendEmail(mailto, subject, result);  
  
 } else {  
 // If sign in fails, display a message to the user.  
 Toast.*makeText*(getApplicationContext(), "SOMETHING WENT WRONG", Toast.*LENGTH\_LONG*).show();  
 }  
  
 // ...  
 }  
 });  
 break;  
  
 case R.id.*denyButton*:  
 result = "Sorry your registration wasn't approved. If this is your second try you are blocklisted and cannot"+ " and cannot resubmit your application. However, if it was your first attempt, you can try to " +" register again. This time under interest section also add why I should reconsider my decision."  
 + "\n \n Thank you for applying. \n Sincerely \n Samin Saif";  
 subject = "Registration fail";  
 sendEmail(mailto, subject, result);  
  
  
 }  
  
}

**10. getUserInfo()**

Input: None

Output: None

Description: Gets users info from the firebase Database and puts them in an array to show it in a list.

public void getUserInfo(final FirebaseCallback firebaseCallback) {  
 db.collection("Users").orderBy("Point", Query.Direction.*DESCENDING*)  
 .get()  
 .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {  
 @Override  
 public void onComplete(@NonNull Task<QuerySnapshot> task) {  
 int i = 0;  
 if (task.isSuccessful()) {  
 for (QueryDocumentSnapshot document : task.getResult()) {  
 emails.add(document.get("Email").toString());  
 name.add(document.get("Name").toString());  
 point.add((Long) document.get("Point"));  
 imageLink.add(document.get("Links").toString());  
  
 firebaseCallback.onCallback(emails, name, point, imageLink);  
 }  
 } else {  
 Log.*d*("INFO", "Error getting documents: ", task.getException());  
 }  
 }  
  
 });  
}

**11. createGroup()**

Input: Group name, and description

Output: Group created in the database

Description: Create group database in firebase

createGroup.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 if (!groupName.getText().toString().trim().isEmpty() && !descriptionGroup.getText().toString().trim().isEmpty()) {  
 Email.setAlpha(1);  
 inviteButton.setAlpha(1);  
 createGroup.setEnabled(false);  
 createGroup.setAlpha(0);  
 groupName.setEnabled(false);  
 descriptionGroup.setEnabled(false);  
  
 createGroup(groupName.getText().toString().toUpperCase(), descriptionGroup.getText().toString());  
  
 } else {  
 Toast.*makeText*(getContext(), "Please Enter Group name and Description", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
});

**12. checkIfGroupNameIsAvailable:**

Input: Group Name

Output: Message whether group name is available

Description: This method checks if the group user trying to create has a unique name or not

public void checkIfNameIsAvailable(String name){  
  
 db.collection("Groups")  
 .get()  
 .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {  
 @Override  
 public void onComplete(@NonNull Task<QuerySnapshot> task) {  
 if (task.isSuccessful()) {  
 for (QueryDocumentSnapshot document : task.getResult()) {  
 Log.*i*("NAMEEE", document.getId()+" and"+ name);  
 if(name.equals(document.get("Group Name"))){  
 checkAvail.setTextColor(ContextCompat.*getColor*(getContext(), R.color.*colorRed*));  
 checkAvail.setText("This group name is already been used");  
 createGroup.setEnabled(false);  
 }else{   
 checkAvail.setTextColor(ContextCompat.*getColor*(getContext(),R.color.*colorGreen*));  
 checkAvail.setText("Group name is available for use");  
  
 createGroup.setEnabled(true);  
 }  
 }  
 } else {  
 Log.*d*(*TAG*, "Error getting documents: ", task.getException());  
 }  
}

**13. inviteUsers()**

Input: Users email

Output: A group join request is sent

Description: Inviting multiple user to join the group

inviteButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 addData(groupName.getText().toString().toUpperCase(), descriptionGroup.getText().toString(), Email.getText().toString().trim().toUpperCase());  
 Email.setText("");  
 doneButton.setAlpha(1);  
 }  
});

//adding data to database

public void addData(String groupName, String Description, String email) {  
 Map<String, Object> groupData = new HashMap<>();  
 groupData.put("Invited by", currentUserName);  
 groupData.put("Contact", currentUser);  
 groupData.put("Group Name", groupName);  
 groupData.put("Group Description", Description);  
 groupData.put("Time", FieldValue.*serverTimestamp*());  
  
 DocumentReference docRef = db.collection("Users").document(email);  
 docRef.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {  
 @Override  
 public void onComplete(@NonNull Task<DocumentSnapshot> task) {  
 if (task.isSuccessful()) {  
 DocumentSnapshot document = task.getResult();  
 if (document.exists()) {  
 db.collection("Users").document(email).collection("Group Invite").document(groupName)  
 .set(groupData)  
 .addOnSuccessListener(new OnSuccessListener<Void>() {  
 @Override  
 public void onSuccess(Void aVoid) {  
 Log.*d*(*TAG*, "DocumentSnapshot successfully written!");  
 Toast.*makeText*(getContext(), "Successfully Invited " + email, Toast.*LENGTH\_SHORT*).show();  
 }});  
 } else {  
 Toast.*makeText*(getContext(), "No Such user Exists", Toast.*LENGTH\_SHORT*).show();  
 }  
 }

**14. displayListOfRequest()**

Input: None

Output: List of groups join request

Description: Users gets to see if the have any group join request

*displayListOfRequest(){*

*listOfRequest*=root.findViewById(R.id.*groupRequest*);  
ArrayAdapter<String> adapterlist=new ArrayAdapter<String>(getContext(), android.R.layout.*simple\_list\_item\_1*, groupRequest);  
*listOfRequest*.setAdapter(adapterlist);  
db.collection("Users").document(currentUser).collection("Group Invite").orderBy("Time", Query.Direction.*DESCENDING*)  
 .get()  
 .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {  
 @Override  
 public void onComplete(@NonNull Task<QuerySnapshot> task) {  
 if (task.isSuccessful()) {  
 for (QueryDocumentSnapshot document : task.getResult()) {  
 groupRequest.add(document.getId()+"\n"+"Invited by "+document.get("Invited by"));  
 ((BaseAdapter) *listOfRequest*.getAdapter()).notifyDataSetChanged();  
 }  
 } else {  
 Toast.*makeText*(getContext(), "Something went wrong", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 });

}

**15. updateList()**

Input: None

Output: Updated database

Description: Updates list of group request after changes in database

public void updatelist{

((BaseAdapter) *listOfRequest*.getAdapter()).notifyDataSetChanged();

}

**16: listOfRequest.onItemClickListner()**

Input: Clicked on item from list

Output: Info on the item clicked

Description: Users clicks on the group request and gets full details on the group

*listOfRequest*.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
 @Override  
 public void onItemClick(AdapterView<?> parent, View view, int position, long id) {  
 String text = (*listOfRequest*.getItemAtPosition(position)).toString();  
 String[] parts = text.split("\n");  
 String part1 = parts[0].trim();  
 Intent intent= new Intent(getContext(), GroupReviewActivity.class);  
 intent.putExtra("Group Name",part1);  
 startActivity(intent);  
 }

**17: acceptInvite()**

Input: Accept button clicked

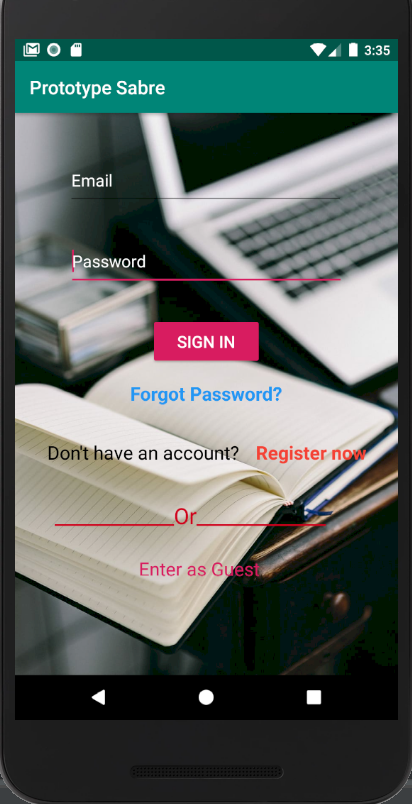
Output: Goes back to the list of group request

Description: User accepts the group request and added to the member database of the group

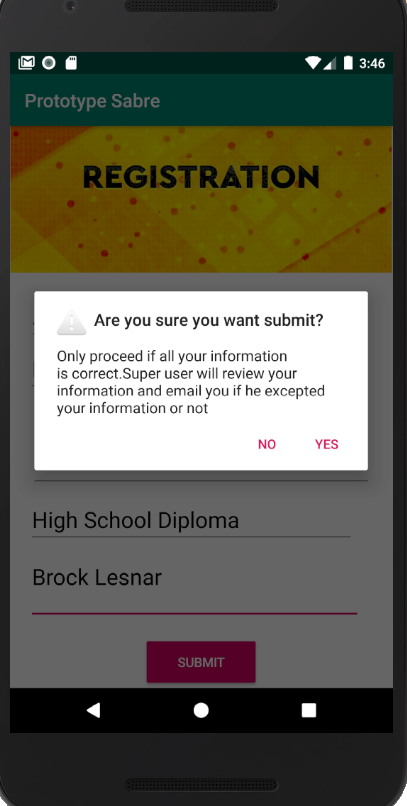
public void acceptInvite(View view){  
  
 Map<String, Object> data = new HashMap<>();  
 Map<String, Object> info = new HashMap<>();  
  
 info.put("Group Description", groupDescription);  
 info.put("Group Name", groupName);  
 info.put("Contact", contact);  
 info.put("Created By",invitedbBy);  
 data.put("Point", Point);  
 data.put("Name", Name);  
  
db.collection("Groups").document(groupName).collection("Members").document(currentUser) .set(data).addOnSuccessListener(new OnSuccessListener<Void>() {  
 @Override  
 public void onSuccess(Void aVoid) {  
 db.collection("Users").document(currentUser).collection("Groups").document(groupName)  
 .set(info).addOnSuccessListener(new OnSuccessListener<Void>() {  
 @Override  
 public void onSuccess(Void aVoid) {  
 db.collection("Users").document(currentUser).collection("Group Invite")  
 .document(groupName).delete().addOnSuccessListener(new OnSuccessListener<Void>() {  
 @Override  
 public void onSuccess(Void aVoid) {  
 oast.*makeText*(GroupReviewActivity.this, "Sucessfully added the group", Toast.*LENGTH\_SHORT*).show();updateDataset();}

**5. Gui Screen**

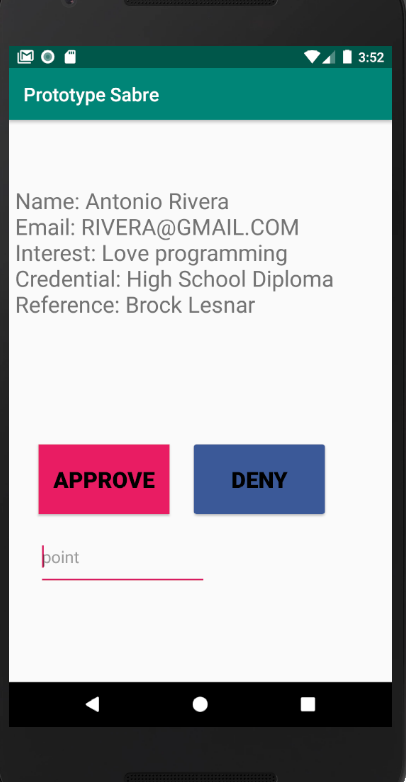
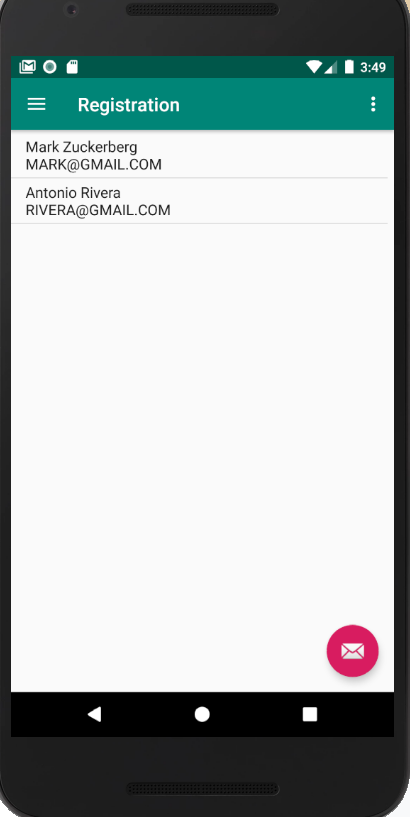
**Login Page:**

****

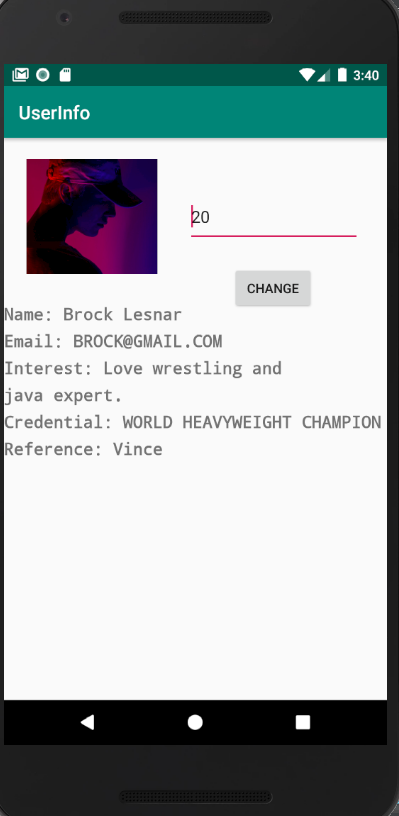
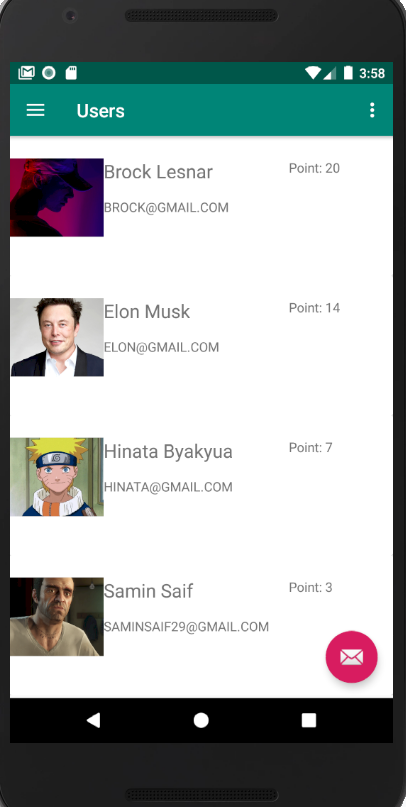
**Registration Page:**

** **

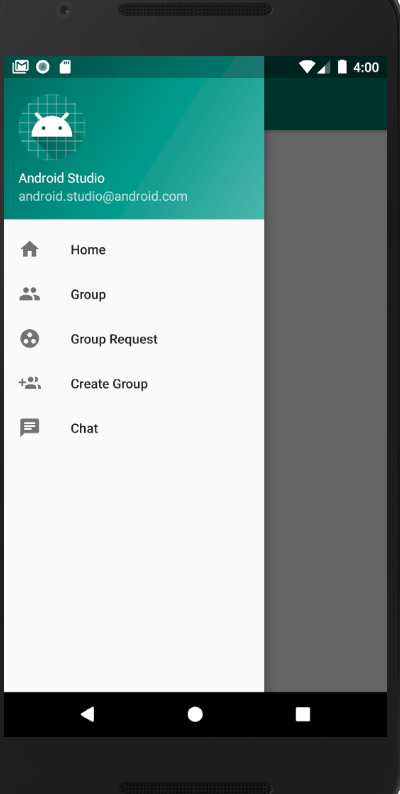
**Registration request viewed by super user (Prototype):**

****

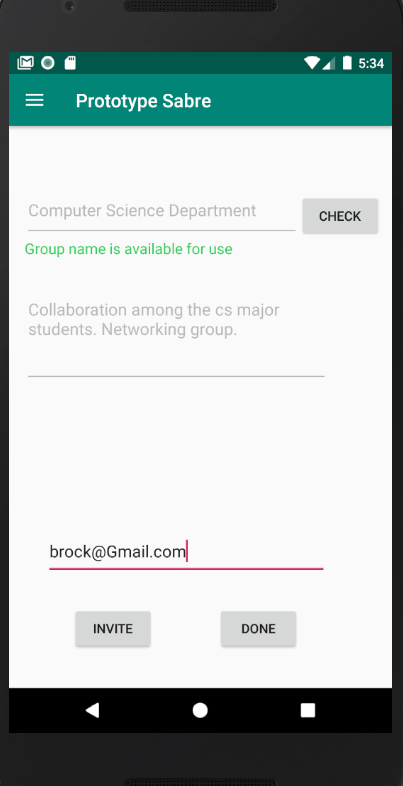
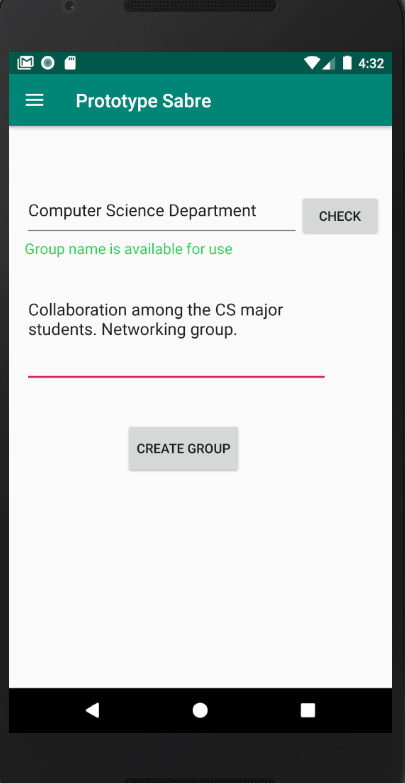
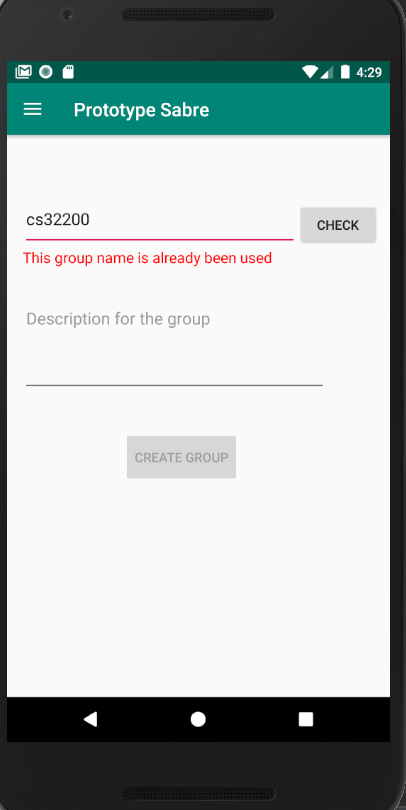
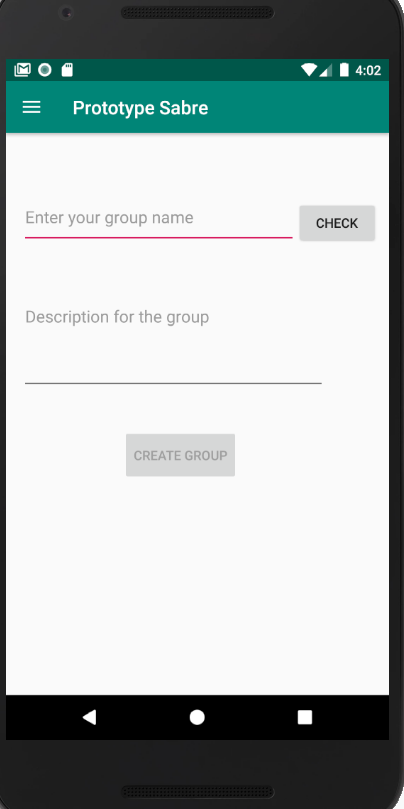
**Prototype userinfo viewed by SuperUser(Prototype):**

****

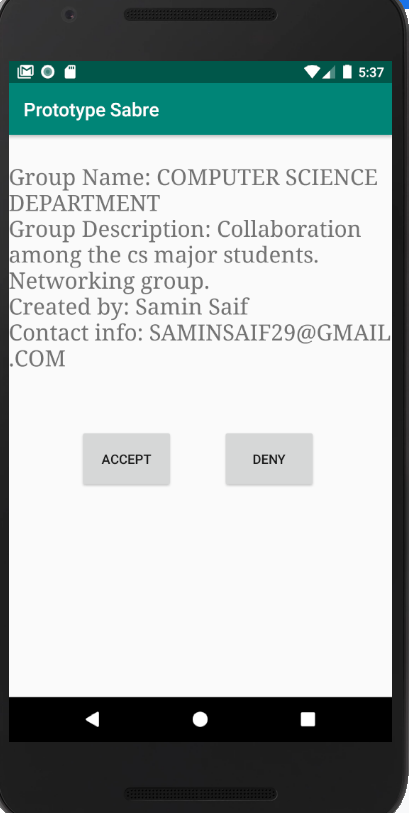
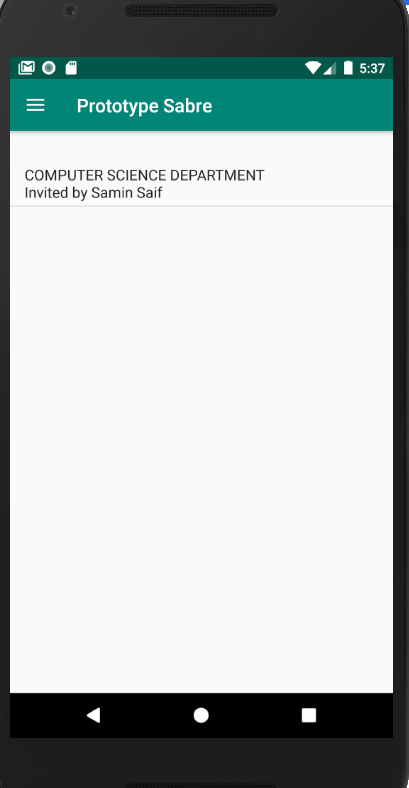
**Regular Users View (Prototype):**

****

**Creating Groups (prototype):**

****

**User reviewing group request page (Prototype):**

****

**6. Team Meeting:**

Meeting # 1 (3 hours) : We got familiar with our team members. We discussed language and frameworks we would use to develop the system. Then we divided the workload among each other.

Meeting # 2 (3 hours): We discussed and worked on the Phase 1 report, and equally distributed the work. We discussed how to enhance the report and made sure to set a deadline to finish the report, thus everyone can revise it.

Meeting # 3 (3 hours): We discussed and worked on the Phase 2 report, and equally distributed the work. We talked about the diagram aspect. Then we planned how to proceed on the project and the trade offs. We also set a deadline for the final project so we can go over the design, check functionalities, and perform testing for bugs.

**7. Github Link:** [**https://github.com/msaif2920/SABRE-PROTOTYPE/tree/Prototype**](https://github.com/msaif2920/SABRE-PROTOTYPE/tree/Prototype)