Design Use Cases

Making Friends Over Food App:

“Tables”

Team Never Eat Alone

Group Members:
Christopher Weaver, Sushanth Mukkamalla,
Vi Phung, Zihao Zhou, Khoa Bui, Xuan Zhang,
Zhening Huang, Yinlong Qian, Jinwei Ren,
Yuhong Sun, Sean Yeh

Table of Contents

Next iteration planned:

Priority	Description
1	Must Implement
2	Should Implement
3	Want to Implement
4	Won’t Implement

Status
Planned
In-Progress
Completed
Next Iteration

Design Use Case #1.1	Sign Up
Description:	The actor creates an account UCSD email in the application with an UCSD email and password.
Actor:	The user of the application.
User Goals:	Create an account to begin taking advantage of the applications functions.
Desired Outcome:	The user will successfully create an account that will be saved in the database and will be able to use for the duration of their time with the application.
Dependent Design Use Cases:	None.
Requirements:	SR 1.1
Pre-Conditions:	The user installs the application, opens it without an account and have a UCSD email.
Post-Conditions:	An account is registered in the online database and can be accessed so that the account can be logged into/have information stored with it.
Trigger:	The user would like to begin using the functionality of the application.
Workflow:	1. The user shall click the “Sign Up” button on the login page. 2. The frontend shall navigate the user to the sign up page using activity_create_account.xml. 3. The user shall enter a UCSD email and a password, and click the “Create Account” button. 4. The frontend shall check the input email is in the correct format and send the request to the account controller. 5. The account controller shall send a verification email to the user using the Firebase Create Account feature. 6. The user shall click the link in the verification email. 7. The backend shall create a new account in the database in AccountDAO.java. 8. The backend shall notify the frontend that the account is created successfully.

Alternate Workflow:	1. The user shall enter an invalid email or password and click the “Create Account” button. 2. The frontend shall detect the invalid inputs and toasts a failed message in CreateAccountView.java.
Status:	Completed
Priority:	1

Design Use Case #1.2	Login
Description:	The actor logs into the application using the registered account information through a login screen.
Actor:	The user of the application.
User Goals:	The user wants to be able to login and continue using the application.
Desired Outcome:	The user will successfully login and use the app.
Dependent Design Use Cases:	UC 1.1
Requirements:	SR 1.2
Pre-Conditions:	The user signs up for an account and has information ready to login.
Post-Conditions:	The application will direct the user to the main screen.
Trigger:	The user would like to login to the application.
Workflow:	1. The user shall enter the UCSD email and the password and click the “Sign In” button on the sign in page. 2. The frontend shall toast a processing message and send the request to the account controller. 3. The account controller shall log the user in using the Firebase Login feature in AccountDAO.java. 4. The frontend shall navigate the user to the homepage using search_fragment.xml.
Alternate Workflow:	1. The user shall enter invalid UCSD email or password. 2. The frontend shall toast a processing message and send the request to the account controller. 3. The account controller shall detect the invalid inputs in AccountDAO.java. 4. The backend shall send the information of the invalid inputs detected to the frontend. 5. The frontend shall toasts a message that explains the case in LogInView.java.
Status:	Completed
Priority:	1

Design Use Case #1.3	Log Out
Description:	The actor logs out the application using the “Log out” button.
Actor:	The user of the application.
User Goals:	The user wants to be able to log out and stop using the app.
Desired Outcome:	The user will successfully log out and be directed back to the login screen.
Dependent Design Use Cases:	UC 1.1, UC 1.2
Requirements:	SR 1.3
Pre-Conditions:	The user already has a registered account and logged onto the application.
Post-Conditions:	The user cannot see any information specific to the account that was previously logged on and gets directed to the login screen.
Trigger:	The user would like to log out of the application.
Workflow:	1. The user shall click the menu bar at the upper left corner and click the “Log Out” button. 2. The frontend shall send the request to the Account Controller. 3. The Account Controller shall log the user out using the Firebase logout feature in AccountDAO.java. 4. The frontend shall navigate the user to the login page using sign_in.xml.
Alternate Workflow:	None.
Status:	Completed
Priority:	1

Design Use Case #1.4	Password Recovery
Description:	The actor resets their password with the “Forgot password” button on the login screen.
Actor:	The user of the application.
User Goals:	The user wants to be able to change their password.
Desired Outcome:	The user will successfully create a new password for their account.
Dependent Design Use Cases:	UC 1.1, 1.2
Requirements:	SR 1.4
Pre-Conditions:	The user already has a registered account.
Post-Conditions:	The user is directed to the login screen.
Trigger:	The user has forgotten their password and would like to be able to login the app.
Workflow:	1. The user shall click “Forgot password?” button on the login page. 2. The frontend shall navigate the user to the password recovery page using reset_password_layout.xml. 3. The user shall enter their UCSD email and click the “Reset Password” button. 4. The frontend shall check the input email is in the correct format and send the request to the backend. 5. The backend shall send a password recovery link to the user’s email using the Firebase Password Recovery feature in AccountDAO.java. 6. The backend shall toast a message that Password reset email has been sent in AccountDAO.java. 7. The frontend shall navigate the user to the login page using sign_in.xml. 8. The user shall open their email from the app. 9. The user shall click the link. 10. The user shall enter new password. 11. The user shall click “Save”. 12. The backend shall update the new password field

	of the user's object in the database using the Firebase Password Recovery feature in AccountDAO.java.
Alternate Workflow:	1. The user shall enter invalid email on the password recovery page. 2. The frontend shall check the input and toasts a message that the input is an invalid email in PasswordRecoveryiew.java.
Status:	Completed
Priority:	1

Design Use Case #2.1	Create Personal Profile
Description:	The actor can create a personal profile by inputting data (i.e. pictures, hobbies, classes, college, year, etc)
Actor:	The user of the application.
User Goals:	The user wants to show what they are interested in and anything else that they would like to share publicly.
Desired Outcome:	The user will have a profile with all the newly input details available to be seen by the system, themselves (on the "My Profile" screen), and other users.
Dependent Design Use Cases:	UC 1.1, UC 1.2
Requirements:	S.R 2.1
Pre-Conditions:	The user has a registered account and is logged in.
Post-Conditions:	The user's new profile is stored in the database.
Trigger:	The user would like to begin using the application and share their interests publicly.
Workflow:	1. The user shall click the "log in" button for the first time. 2. The frontend shall navigate the user to the profile creation page using create_personal_profile.xml and initialize the features of the elements in CreatePersonalProfileView.java. 3. The user shall enter Username, Interest Levels, Interest Tags and so on. 4. The user shall click "Continue" at the bottom of the profile creation page. 5. The frontend shall check the inputs and send the request to the profile controller. 6. The profile controller shall save all the inputted data in the database in ProfileDAO.java. 7. The frontend shall navigate the user to the homepage using search_fragment.xml.
Alternate Workflow:	1. The user shall close the application when they are on the profile creation page. 2. The system shall let the frontend navigate the user

	to the profile creation page again when the user logs in next time.
Status:	Completed
Priority:	1

Design Use Case #2.2	Edit Personal Profile
Description:	The actor can edit their personal profile by inputting new data or editing existing profile.
Actor:	The user of the application.
User Goals:	The user wants to update interests, pictures, etc. or delete any data that they would like to keep in private.
Desired Outcome:	The user will have a profile with all the newly input details available to be seen by the system, themselves (on the "My Profile" screen), and other users.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 2.1
Requirements:	S.R 2.2
Pre-Conditions:	The user already has a registered account, is logged in and has an existing profile.
Post-Conditions:	The database is updated to reflect the edited profile.
Trigger:	The user would like to change their profile information.
Workflow:	1. The user shall click the menu bar at the upper left corner and then the "Edit Profile" button. 2. The frontend shall navigate the user to the profile edit page using edit_personal_profile_form.xml. 3. The user shall enter their Username, rate Interest Levels again or add new Interest Tags and so on. 4. The user shall click "Continue" button at the bottom of the profile page. 5. The frontend shall check the inputs and send the request to the profile controller. 6. The profile controller shall save all the inputted data in the database in ProfileDAO.java. 7. The frontend shall toast a successful message and navigate the user to the homepage using search_fragment.xml.
Alternate Workflow:	1. The user shall click the menu bar at the upper left corner and then the "Edit Profile" button. 2. The frontend shall navigate the user to the profile

	page using edit_personal_profile_form.xml. 3. The user shall clicked the “<-” button at the upper left corner. 4. The frontend shall navigate the user back to the homepage using search_fragment.xml.
Status:	Completed
Priority:	1

Design Use Case #3.1	Search for Other Users
Description:	The user gets a list of other users. The search list should display each user's name and tags of interests. The users at the top of the search list should have similar interest levels as those of the user respectively.
Actor:	The user of the application.
User Goals:	The user will be able to find people who they may be interested to meet with.
Desired Outcome:	The user successfully find potential new friends that they can hang out with over a meal.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 2.1
Requirements:	SR 3.1
Pre-Conditions:	The user has logged in and has set up their profile.
Post-Conditions:	The app displays a list of users, those who are the best match to the user at the top.
Trigger:	The user wants to find people that they are interested to meet with.
Workflow:	1. The user shall click the "Search" button on the search fragment page. 2. The frontend shall send the request to the search controller. 3. The search controller shall query the database and generate a list of users in a particular order in SearchDAO.java. 4. The backend shall send the list of users to the frontend. 5. The frontend shall display the result as a list of users with their names and interest tags in SearchView.java.
Alternate Workflow:	None.
Status:	Completed
Priority:	1

Design Use Case #3.2	Find Random Users
Description:	The user finds random available users to chat with.
Actor:	The user of the application.
User Goals:	The user will be able to find random people who they may be interested to meet with.
Desired Outcome:	The user successfully finds friends that they can hang out with over a meal.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 2.1
Requirements:	SR 3.2
Pre-Conditions:	The user has logged in and has set up their profile.
Post-Conditions:	The app displays random users, one at a time.
Trigger:	The user wants to find people they are interested to meet with.
Workflow:	1. The user shall click the "Random Search" button on the search fragment page. 2. The frontend shall send the request to the search controller. 3. The search controller shall query the database and generate a list of users in a random order in SearchDAO.java. 4. The backend shall send the list of user to the frontend. 5. The frontend shall display the result as a list of users with their names and interest tags in SearchView.java.
Alternate Workflow:	None.
Status:	Completed
Priority:	1

Design Use Case #4.1	Select Matched Users
Description:	The user can select people who they may be interested to meet and eat with.
Actor:	The user of the application.
User Goals:	The user wants to choose who they may want to meet with to eat.
Desired Outcome:	The user successfully informs the application of the people they are considering meeting and eating with.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 3.1
Requirements:	SR 4.1
Pre-Conditions:	The user has started a search and received a list of users.
Post-Conditions:	The system receives the user's choices of their potential new friends and sends them an invite notification.
Trigger:	The user wants to tell the application people they are interested in meeting and eating with.
Workflow:	1. The user shall tap on a user's area on the search list page. 2. The frontend shall send the request to the profile controller. 3. The profile controller shall find all the information belong to the user in the database in ProfileDAO.java. 4. The frontend shall display the information of the user using activity_profile_viewer.xml. 5. The user shall click the "<-" button at the upper left corner and repeat steps 1-5. 6. The user shall press the toggle(s) on the right hand side of one or more users on the search list. 7. The frontend shall turn the toggle(s) from white to purple and record all the users selected in SelectMatchedUsersView.java.
Alternate Workflow:	1. The user shall click the "<-" button. 2. The frontend shall navigate the user back to the homepage using search_fragment.xml.

Status:	Completed
Priority:	1

Design Use Case #4.2	Send Invitation
Description:	The user can send invite notifications to all the people they have selected from the list.
Actor:	The user of the application.
User Goals:	The people selected on the list get to know they are invited by the user.
Desired Outcome:	The people selected by the user receive an invitation notification.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 3.1, UC 3.2, UC 4.1
Requirements:	SR 4.2
Pre-Conditions:	The user has run a search for other users. The user has selected at least one person on the list.
Post-Conditions:	The system sends invite notifications to the selected users.
Trigger:	The user wants people they have selected on the list to get the invitation notification.
Workflow:	1. The user shall click the “Send Invitation” at the bottom of the search list page. 2. The frontend shall send the request to the invitation controller. 3. The invitation controller shall send invitation(s) to the user(s) selected in InvitationDAO.java. 4. The frontend shall navigate the user back to the homepage using search_fragment.xml.
Alternate Workflow:	1. The user shall click the “<-” button at the upper left corner of the search list page. 2. The frontend shall navigate the user back to the homepage using search_fragment.xml.
Status:	Completed
Priority:	1

Design Use Case #4.3	Receive Invitation
Description:	The user should see the notice that someone want to dine with them. The user can then choose to accept or reject such invitation.
Actor:	The user of the application.
User Goals:	The user will be able to accept/decline this invitation.
Desired Outcome:	If the user accepts the invitation, the inviter will be able to chat with them. If the user declines the invitation, the inviter will see a notification.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 4.1, UC 4.2
Requirements:	SR 4.3
Pre-Conditions:	The user has received an invite notification.
Post-Conditions:	If the invite is accepted, a chat room will be created between the two users.
Trigger:	The user received an invitation.
Workflow:	1. The user shall click the "View Profile" button. 2. The frontend shall send the request to the profile controller. 3. The profile controller shall find all the information belong to the user in the database in ProfileDAO.java. 4. The frontend shall display the information of the user using activity_profile_viewer.xml. 5. The user shall click the "<-" button at the upper left corner and repeat steps 1-5. 6. The user shall click the "Accept" button. 7. The frontend shall send the request to the invitation controller. 8. The invitation controller shall find the inviter in the database and update the invitation status in the InvitationDAO.java. 9. The frontend shall display the accepted status to the inviter using chat_accept_decline_layout.java.
Alternate Workflow:	1. The user shall click the "Decline" button. 2. The frontend shall send the request to the

	invitation controller. 3. The invitation controller shall find the inviter in the database and update the invitation status in the InvitationDAO.java. 4. The frontend shall display the declined status to the inviter using chat_accept_decline_layout.java.
Status:	Completed
Priority:	1

Design Use Case #5.1	Create Chat Room
Description:	The user should be able to create a new chat room with people they have already matched with.
Actor:	The user of the application.
User Goals:	The user wants to create a chat room to communicate with a certain person.
Desired Outcome:	The user has a new chat room which they can invite other users to.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 4.1, UC 4.2, UC 4.3
Requirements:	SR 5.1
Pre-Conditions:	The user has accepted the invitation that has been sent to them.
Post-Conditions:	A chat room with two users is created.
Trigger:	The user accepted the invitation.
Workflow:	1. The frontend shall check the status of the invitation and send the request to the chat controller. 2. The chat controller shall create a chat room for the two users and store the chat room information in the database in ChatRoomsDAO.java.
Alternate Workflow:	None.
Status:	Completed
Priority:	1

Design Use Case #5.2	Send Messages in Chat Room
Description:	The user should be able to chat with another user or multiple users.
Actor:	The user of the application.
User Goals:	User created chat room to communicate with another user, potentially to set up a meet or exchange information.
Desired Outcome:	The message is sent to the receiving user with minimal delay.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 4.1, UC 4.2, UC 4.3, UC 5.1
Requirements:	SR 5.2
Pre-Conditions:	A chat room has been created.
Post-Conditions:	A message is sent from one user to another.
Trigger:	A user wants to communicate with other user who they want to meet.
Workflow:	1. The user shall click on the box to insert text. 2. The user shall enter in the message. 3. The frontend send the message to the chat controller. 4. The chat controller shall store the message in the database and get the ID of the other user in the chat room in ChatRoomsDAO.java. 5. The backend shall inform the frontend the ID of the other user in the chat room. 6. The frontend shall display the message to other user's chat room page in ChatRoomForm.java using chat_room_layout.xml.
Alternate Workflow:	1. The user shall click on the box to insert text. 2. The user shall click the "<-" button at the upper left corner of the chat room page. 3. The frontend shall discard the input message and navigate the user back the homepage using search_fragment.xml.
Status:	Completed
Priority:	1

Design Use Case #5.6	Set When and Where to Meet
Description:	The users of the chat room will be able to set when and where to meet.
Actor:	The user of the application.
User Goals:	The users of the chat room will be able to set the meeting location and time so they can access to the information at all time without looking back at previous messages.
Desired Outcome:	The users will successfully be able to set when and where to meet in the chat room.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 4.1, UC 4.2, UC 4.3, UC 5.1
Requirements:	SR 5.6
Pre-Conditions:	A chat room has been created. The user is in the chat room.
Post-Conditions:	The meeting time and location will be set, saved, and displayed to the user.
Trigger:	The users of a chat room want to initialize the location and time of the meeting.
Workflow:	1. The user shall click the button at the upper right corner of the chat room page and then the "Set Time/Location" button. 2. The frontend shall navigate the user to the time & location page using room_time_location.xml. 3. The user shall input the time and location they want everyone to meet and click the "Submit" button at the bottom of the page. 4. The frontend shall pass the information to the chat controller. 5. The chat controller shall store the information in the database in ChatRoomsDAO.java. 6. The frontend shall navigate the user back to the chat room page using chat_room_layout.xml and display the updated time and location in ChatRoomForm.java.
Alternate Workflow:	1. The user shall click the "<-" button at the upper left corner of the time & location page.

	2. The frontend shall navigate the user back to the chat room page using chat_room_layout.xml.
Status:	Completed
Priority:	1

Design Use Case #5.7	Edit When and Where to Meet
Description:	The users of the chat room will be able to edit when and where to meet.
Actor:	The user of the application.
User Goals:	The users of the chat room will be able to update the meeting location and time.
Desired Outcome:	The users will successfully be able to edit when and where to meet in the chat room.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 4.1, UC 4.2, UC 4.3, UC 5.1, UC 5.6
Requirements:	SR 5.7
Pre-Conditions:	A chat room has been created, meeting time and location already set. The user is in the chat room.
Post-Conditions:	The meeting time and location will be updated and saved.
Trigger:	The users of a chat room want to update the location and time of the meeting.
Workflow:	1. The user shall click the button at the upper right corner of the chat room page and then the "Set Time/Location" button. 2. The frontend shall navigate the user to the time & location page using room_time_location.xml. 3. The user shall input the location and time they want everyone to meet and click the "Submit" button at the bottom of the page. 4. The frontend shall pass the information to the chat controller. 5. The chat controller shall store the information in the database in ChatRoomsDAO.java. 6. The frontend shall navigate the user back to the chat room page using char_room_layout.xml and display the updated time and location in ChatRoomForm.java.
Alternate Workflow:	1. The user shall click the "<-" button at the upper left corner of the chat room page. 2. The frontend shall navigate the user back to the homepage using search_fragment.xml.

Status:	Completed
Priority:	1

Design Use Case #5.8	Leave Chat Room
Description:	The user will be able to leave a chat room and remove it from the list of active chat rooms.
Actor:	The user of the application.
User Goals:	The user will be able to leave a chat room.
Desired Outcome:	The user will be able to leave a chat room and stop receiving notifications from it.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 4.1, UC 4.2, UC 4.3, UC 5.1
Requirements:	SR 5.8
Pre-Conditions:	A chat room has been created. The user is in a chat room.
Post-Conditions:	The chat room will be removed from the list of active chat rooms.
Trigger:	The user wants to leave a chat room.
Workflow:	1. The user shall click the button at the upper right corner of the chat room and then the “Leave Chat” button. 2. The frontend shall send the request to the chat controller. 3. The chat controller shall find the room and delete the room information in the database in ChatRoomsDAO.java. 4. The backend shall notify the frontend after the deletion of the room information. 5. The frontend shall toast the corresponding message in CharRoomXML.java navigate the user to the homepage using search_fragment.xml.
Alternate Workflow:	N/A.
Status:	Completed
Priority:	1

Design Use Case #7.1	Get History of Everyone Met
Description:	The user will have the ability to see other user's names they have interacted with in this app. This includes invites they have declined, current chat rooms, left chat rooms, and pending invitations.
Actor:	The user of the application.
User Goals:	The User can look through information of everyone they have met with so they will have the ability to contact them.
Desired Outcome:	The user will have the ability to see other user's names they have interacted with in this app.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 2.1, UC 4.2, UC 4.3, UC 5.7, UC 5.8
Requirements:	SR 7.1
Pre-Conditions:	The user has interacted with other users on the app.
Post-Conditions:	The users will be able to view interaction history.
Trigger:	The user wants to see all the users they have interacted with in the past, or currently interacting with.
Workflow:	1. The user shall click the "chat" button on the homepage. 2. The frontend shall send the request to the backend. 3. The backend shall get all the room information in the database and identify all the invitations related to the user and the statuses. 4. The backend shall pass the sorted information to the front end. 5. The frontend shall navigate the user to the invitation list page using chat_item_invite_pending.xml, and display the related users in ChatRoomXML.java.
Alternate Workflow:	1. The user shall click the "<-" button on the invitation list page. 2. The frontend shall navigate the user to the homepage using search_fragment.xml.

Status:	Completed
Priority:	1

Next iteration:

Design Use Case #2.3	Set Status
Description:	The actor will be able to set their status (i.e. setting it to available, hidden, away, or busy).
Actor:	The user of the application.
User Goals:	The user wishes to express their current status.
Desired Outcome:	The user will successfully update their status.
Dependent Design Use Cases:	UC 1.1, UC 1.2
Requirements:	S.R 2.3
Pre-Conditions:	The user is logged on and has navigated to the status box.
Post-Conditions:	The system saves the user's new status in database and displays it to the appropriate users.
Trigger:	The user would like to express their status.
Workflow:	1. The user shall click the “Menu” button at the upper left corner of the homepage. 2. The user shall click the “Availability” button. 3. The frontend shall display a combobox in StatusView.java. 4. The user shall click on one of the statuses provided. 5. The frontend shall send the request to the profile controller. 6. The profile controller shall store the updated status information in the database in ProfileDAO.java. 7. The backend shall notify the frontend that the status has been updated. 8. The frontend shall navigate the user back to the homepage using search_fragment.xml.
Alternate Workflow:	N/A
Status:	Planned for Next Iteration
Priority:	4

Design Use Case #5.3	Mute Notification in Chat Room
Description:	The user will be able to toggle receiving notification messages for chat rooms.
Actor:	The user of the application.
User Goals:	The user wants to create a chat room to communicate with a certain person/group of people.
Desired Outcome:	The user can mute a chat room if they are getting annoyed with all the notifications they are receiving.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 4.1, UC 4.2, UC 4.3, UC 5.1, UC 5.2
Requirements:	SR 5.3
Pre-Conditions:	A chat room has been created and messages are sent.
Post-Conditions:	The user has muted the chat room to stop receiving notifications.
Trigger:	The user may not want to be distracted at the moment and would like to turn off their notifications.
Workflow:	1. The user shall click on the button at the upper right corner of the chat room page and then the "Mute" button. 2. The frontend shall send the request to the chat controller. 3. The chat controller shall find the room in the database and update its information in ChatRoomsDAO.java. 4. The frontend shall receive the task has been done by the backend and highlights the "Mute" button in ChatRoomForm.java.
Alternate Workflow:	N/A.
Status:	Planned for Next Iteration
Priority:	4

Design Use Case #5.4	Create Poll in Chat Room
Description:	The user will be able to vote along with others in the chat if they like the suggested time and location.
Actor:	The user of the application.
User Goals:	The participants in the chat room will be able to see if a majority of the participants approve of the suggested time and location.
Desired Outcome:	A graph detailing how many users like the suggested location or time will be displayed.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 4.1, UC 4.2, UC 4.3, UC 5.1
Requirements:	SR 5.4
Pre-Conditions:	A chat room has been created.
Post-Conditions:	The results of the poll will be shown in the chat.
Trigger:	A user wants to suggest time or location options to other members of the chat and figure out the best option.
Workflow:	1. The user shall click the button at the upper right corner of the chat room page and then the “Poll” button. 2. The frontend shall navigate the user to the poll page using poll.xml. 3. The user shall input time and location to click “Submit” button. 4. The frontend shall pass the information to the chat controller. 5. The chat controller shall store the poll information in the room found in the database in ProfileDAO.java. 6. The frontend shall navigate the user back to the chat room page using chat_room_layout.xml and display the poll information in ChatRoomForm.java.
Alternate Workflow:	N/A.
Status:	Planned for Next Iteration
Priority:	4

Design Use Case #5.5	Add Additional Users to Chat Room
Description:	The users in a group chat can add more users to it by sending invitations.
Actor:	The user of the application.
User Goals:	The participants of a group chat will be able to add more users to it.
Desired Outcome:	The users will successfully add an additional user to the chat room.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 4.1, UC 4.2, UC 4.3, UC 5.1
Requirements:	SR 5.5
Pre-Conditions:	A chat room has been created.
Post-Conditions:	A new user will be successfully added to the existing group chat.
Trigger:	The user wants to add an additional group member to the chat room.
Workflow:	1. The user shall click the button at the upper right corner of the chat room page and then the "Add Users" button. 2. The frontend shall navigate the user to the invitation list page using chat_accept_decline_layout.xml. 3. The user shall select one or more users on the invitation list page and click the "Send Invitation" button. 4. The frontend shall send the request to the invitation controller. 5. The invitation controller shall send invitation(s) to the selected user(s) in InvitationDAO.java. 6. The frontend shall navigate the user back to the chat room page using chat_room_layout.xml.
Alternate Workflow:	1. The user shall click the "<-" button on the invitation list page. 2. The frontend shall navigate the user back to the chat room page using chat_room_layout.xml.
Status:	Planned for Next Iteration

Priority:	4

Design Use Case #6.1	Meeting Reminder Notification
Description:	The user will get a notification when it is almost time to meet the group.
Actor:	The user of the application.
User Goals:	The user will get a reminder notification.
Desired Outcome:	The user will get notified from the app about the upcoming meeting and its time and location.
Dependent Design Use Cases:	UC 1.1, UC 1.2, UC 4.1, UC 4.2, UC 4.3, UC 5.1, UC 5.6. UC 5.7
Requirements:	SR 6.1
Pre-Conditions:	The meeting time and location has been set in the chat room.
Post-Conditions:	The user gets notified about the time and location of the meeting.
Trigger:	The user does not want to forget about the scheduled meetings.
Workflow:	1. The user shall log in the application. 2. The frontend shall send the request of checking all the user's meeting information to the chat controller. 3. The chat controller shall find all the chat rooms belong to the user and check the rooms' meeting time in ChatRoomsDAO.java. 4. The backend shall notify the frontend if there is any approaching meetings. 5. The frontend shall display a message to the user about the meeting location and time on the homepage using search_fragment.xml.
Alternate Workflow:	N/A.
Status:	Planned for Next Iteration
Priority:	4