Test Cases

Team: Yellow Submarine

Chenxu Jiang, Pengyu Chen
Dingcheng Hu, Jiaqi Wu
Wanhui Qiao, Yuandong Zhang
Ruitian Lin, Tianyi Wang
Shuyi Ni, Zijin Fu

Yellow Submarine, YS

Creation Date: March 3, 2017
Export/Print Date: March 3, 2017

Table of Contents

USER PROFILE

This category of use cases outlines the structure of profile, including login system, location, and personal information

Title: Account Sign Up [UP1]

Description: System provides functionality for users to sign up an account and use the account to log into our system.
User Goals: User wants to unique identity in our system to chat and make new friends.
Desired Outcome: User shall see a page for them to fill in the email address, passwords and username, then have a page to use that account to log in our system.
Actors:

1. User of application
2. System
3. Parse database

Requirements: System shall allow a user to signup by entering unique username, valid email address and password.
Details:
Priority Level: Priority 1
Status: Implemented & tested
Pre-conditions:

1. User has internet connection.
2. Account with username testapp and email address testapp@test.com, both of which do not exist in the database.
3. Account with username exist and email address exist@test.com, either one already exists in the database. [Alternative Workflow]

Trigger: User has been on the Sign Up screen.
Failed Conclusions:

1. Username/Email address already exists.
2. Email address is not valid.

Workflow:

1. User shall be at the application's Sign Up screen on app's startup by default.
2. System (Front-End) shall render the sign-up screen which contains input boxes for the User to type in their email address testapp@test.com, password testpassword and username testapp.
3. The Parse database shall save the User credentials into the database.
4. System Front-End shall redirect user to the Main Page screen.

Result: This test passes when the System displays the Main Page screen.

Alternative Workflow [Failed Conclusion 1]:

1. After testing the passing workflow, logout and try again
2. User shall be at the application's Sign Up screen on app's startup by default.
3. System (Front-End) shall render the Sign Up screen which contains input boxes for the User to type in their email address testapp@test.com, password testpassword and username testapp.
4. System shall warn user about invalid input such as repeated username, repeated email or unqualified password by sending user inputs to Back-End and verifying inputs during users are typing.
5. System shall use a cross(X) signal to show that username, email or password is invalid, then to prompt the User to try to input email address, password or username again.

Result: This test passes when the System shows a cross in the Sign up screen and then the User can enter email address, password or username.

Title: Forgotten Password Reset [UP2]

Description: The application provides a functionality of resetting forgotten password in the login display, in case of users forget their password. The system shall give send a message to the user's email with a reset password link, which redirects them to update their password. The system shall return back to the Login screen to allow the User to sign in with new password.
User Goals: reset their password using their email address if forget their password.
Desired Outcome: User shall have a new password and System shall replace old password with new one in the database.
Actors:

1. User of Application
2. System
3. Parse database

Dependency Use Cases:

Account Sign Up [UP1]

Requirements: System shall generate reset email token and send an email containing reset password token to allow user to change the password.
Details:

Priority level: 1
Status: Not Implemented

Pre-conditions:

1. User is at the Login screen.
2. User has internet connection.
3. There is an existing account with username testapp, email testemail@gmail.com.
4. User has logged into the account with a valid email address.

Trigger: User clicks Forgot Password? Button from the Login screen.
Failed Conclusions:

1. No account match is found for provided email address in database.
2. User does not enter anything in the email field.

Workflow:

1. User shall press the Forget Password? option from the login page.
2. System Front-End shall render to the user input box to enter their registered email.
3. User shall enter their email testemail@gmail.com and click Next button.
4. System shall verify with database that an account is attached to the email.
5. System Front-End shall render a new input box for user to enter their reset token.
6. System Back-End shall send an email containing reset token to user's registered email.
7. User shall go to their email account and copy the reset token to the input box.
8. System Front-End shall redirect user to enter the new password if reset token is verified.
9. User shall click Reset Password button when they are done.
10. System shall reset user's password, and display "Changed Successfully" and direct user to the Login screen.

Result: This test passes when the User has been redirected to the Login screen.

Alternative Workflow [Failed Conclusion 1]:

1. Execute Workflow step 1-2.
2. User shall enter their email testemailnone@gmail.com and click “Next” button.
3. System shall verify with database that an account is attached to the email.
4. System shall prompt the User with an error message that displays “Email Not Found”.

Result: This test passes when the error message in step 4 is displayed.

Alternative Workflow [Failed Conclusion 2]:

1. Execute Workflow step 1-2.
2. User leave the email field empty and click “Next” button.
3. System shall verify with database that an account is attached to the email.
4. System shall prompt the User with an error message that displays “Please enter an email”.

Result: This test passes when the error message in step 4 is displayed.

Title: Change Profile[UP3]

Description: The User shall be able to change their profile information anytime they want, so that the user can keep their information up-to-date.
User Goals: User shall update their information(email, password, social media) inside the profile page.
Desired Outcome: The profile information shall be changed. System shall display “Update successfully”.
Actor: User of application
Dependency Use Cases:
Account Sign Up [UP1]
Requirements: System shall store all the information of users in database.
Details:
Priority Level: Priority 2
Status: Implemented & tested
Pre-conditions:
1. User has logged into the system with username testapp.
2. User is in the Profile screen.

Trigger: From the menu bar, User clicks Profile
Failed Conclusions:
1. Email is invalid.

Workflow:
1. User with username testapp is at application’s Main Page screen.
2. User shall click their Profile at navbar or their profile picture.
3. System Front-End shall redirect to their Profile screen containing username, password, email, social media field, etc.
4. User shall enter their new information(email, password, social media).
5. System shall check and notify if the new information is valid, during user input.
6. If the new information is valid, the system Back-End shall update the user’s profile information, and display “Changed Successfully”.

Result: This test case passes when the message is displayed in step 6.

Alternative Workflow:
1. User with username testapp is at application’s Main Page screen.
2. User shall click their Profile.
3. System Front-End shall redirect to their Profile screen containing username, password, email, social media field, etc.
4. User shall enter their email asfde, invalid facebook user id or invalid github user name
5. System shall check the email entered and displayed corresponding error.

Result: This test case passed when the error is displayed in step 5.

Title: Log In [UP4]

Description: The User shall be able to log in our system by providing an existing user's credentials. The system verifies the validity of the username/password combination and then logs the user in and pull their account's information.
User Goals: User shall be able to log in the system.
Desired Outcome: System shall securely sign the user into their right account and pull their personal information or display an error message if login failed.
Actors:

1. User of application
2. System
3. Parse database

Dependency Use Cases:

Account Sign Up [UP1]

Requirements: System shall keep user's username and password in database, and a login token at local, for verification of auto login.
Details:
Priority Level: Priority 1
Status: Implemented & tested
Pre-conditions:

1. User has logged off our system.
2. User is in the Login screen.
3. There is an existing account with username testapp and password testpassword.
4. User has internet connection

Trigger: User has logout from an existing account or launch the application for the first time. He wants to login our application
Failed Conclusion:

1. Username and password combination is wrong

Workflow:

1. User is at Login screen by default.
2. User shall input testapp as the username.
3. User shall input testpassword as the password.
4. User shall click the Login button.
5. System Front-End shall verify with Parse database that the credentials are valid.
6. System shall redirect the User to their Main Page screen.

Result: this test passes when the System displays the User's Main Page screen.

Alternate Path Workflow [Failed Conclusion 1]:

1. Execute Workflow Step 1-2.
2. User shall input testpass as the password.
3. User shall click the Login button.

4. System shall prompt the User with a message “Invalid Username/Password Combination”.

Result: This test passes when the system displays the error message in step 4

Alternate Path Workflow [Failed Conclusion 2]:

1. User is at Login screen by default.
2. User shall input testwrong as the username.
3. User shall input testpassword as the password.
4. User shall click the Login button.
5. System shall prompt the User with a message “Invalid Username/Password Combination”.

Result: This test passes when the system displays the error message in step 5

Title: Log Off [UP5]

Description: The User shall be able to log off our system.
User Goals: User wants to log off our system and switch accounts.
Desired Outcome: User shall see the login screen
Primary Actor: User of application
Dependency Use Cases:

Account Sign Up [UP1]

Requirements: System shall delete the local token for automatic login.
Details:
Priority Level: Priority 1
Status: Implemented & tested
Pre-conditions:

1. User has logged in our system with username testapp and password testpassword.
2. User is in the Profile screen.

Post-conditions:
User shall see the welcome screen after logged off
Trigger: User wants to log off our application, switch accounts.
Failed Conclution:

None

Workflow:

1. User is at application's logged-in main page.
2. User shall click their profile icon.
3. System Front-End shall redirect to their profile page.
4. User shall click the "Log off" button.
5. System Front-End shall redirect users to the login page and delete the logged in token stored in local storage.

Result: The test passes when user successfully log off after clicking the "Log off" button and brings user back to the page stated in step 5

Title: Social Media Account [UP6]

Description: User should be able to enter their social media account. This information is public to friends, so that friends could see the user's posts in the social media.
User Goals: User wants to make their social media account visible to their friends and get connected with friends in other social media.
Desired Outcome: User's social media account should be listed on the profile page, and it's visible to friends.
Primary Actor: User of application
Dependency Use Cases:

Account Sign Up [UP1]

Requirements: System shall store user's social media account in database.
Details:
Priority Level: Priority 2
Status: Implemented & Tested
Pre-conditions:

1. User has registered in our system with username testapp and password testpassword.
2. User has a social media account.

Post-conditions:

1. User's friends could see user's social media account.

Trigger: User wants their friends to find their social media accounts.
Failed Conclusion:

1. User decides to delete the content entered

Workflow:

1. Users are at application's log-in main page.
2. Users shall click their profile picture icon.
3. System Front-End shall redirect to their profile page containing username, password, email, social media field, etc.
4. Users shall enter testsocial in each social media box.
5. System Back-End shall update user's social media accounts in database and Front-End shall display "Updated Successfully" to users.

Result: The test passes when system displays the updated social network information as "testsocial" after user enters testsocial

CHATS

This category of use cases outlines the functionality of chatting, such as chatting history, report suspicious friends.

Title: Chatting in a chat room or with Friend[C1]

Description: User can chat with people who use the same Wifi and share the same location with me.
User Goals: Chat with others in a chat room.
Desired Outcome: All users in the same chat room can chat with each other.
Primary Actor:

1. User of Application
2. System
3. Parse database

Dependency Use Cases:

1. Account Sign Up [UP1]
2. Enter a Chat Room[CR2]

Requirements: System shall allow user to exchange information in the chatroom.
Details:

Priority Level: Priority 1.
Status: Implemented & tested

Pre-conditions:

1. User has logged in with the username testapp and password testpassword.
2. User has entered a chat room or on friends chat.
3. User is in the Chat Page.

Post-conditions:User posts and receives message in the chat room.
Trigger: User wants to chat with others in a chat room.
Failed Conclusions:

1. User is not able to send messages or User is not able to receive other users' messages in chronological order.

Receive Workflow:

1. User shall enter the Chat Page by either clicking on chatroom icons or friend icons.
2. The system Front-End shall redirect to tag profile/friend profile
3. The user shall click lower right button chat
4. The system Front-End shall redirect to chat screen
5. The system Front-End shall subscribe unread message documents of user's channel (Chatroom or Friends), subscribe means System Back-End would send any new updates of database to Front-End. Ex. When new message document is inserted to document

6. The system Front-End shall render messages chronologically from top to bottom with other users message on the left and users' sent message on the right.
7. When the users read the message sent by others, System Front-End shall store the message at local, to reduce server load.

Send Workflow:

1. Start from Step 2.
2. User shall enter some message in the input box and click "Send" to post the message.
3. System Front-End shall pass message to Back-End and store message in the database.
4. System Back-End shall send live update to Front-End.
5. If Front-End received the live update, it shall display the latest sent message on the list.
6. Else, it shall notify users the message is failed to be sent, due to network issue.

Result: this test passes when users could chat with each other with the System displaying the chat messages chronologically from top to bottom with other users message on the left and users' sent message on the right.

Title: Chat History[C2]

Description: The feature will allow the user to go back and browse past chat history with their friends.
User Goals: User needs to retrieve chat data with their friends.
Desired Outcome: Convenient access for user to their chat history stored in our database.
Actor:

1. User of Application
2. System
3. Parse database

Dependent Use Cases:

1. Account Sign Up [UP1]
2. Add Friends[F1]
3. Chat with Friends[F2]

Requirements: System shall allow user to retrieve chat histories.
Details:

Priority Level: Priority 2
Status: Implemented & tested

Pre-conditions:

1. User has logged in with the username testapp and password testpassword.
2. User has used our app to chat with their friends.
3. User has clicked friends icon.
4. User has in the Friend Chat Page.

Post-conditions:

1. User interface will display the chat history.
2. Scrolling down will give some older chat history.

Failed Conclusions:

1. User cannot scroll down to see more chat history and the history messages are not in chronological order.

Workflow:

1. The user shall click chatroom icons or friend icons.
2. The system Front-End shall redirect user to friend/tag profile page
3. The user shall click “chat” button.
4. The system Front-End shall redirect user to chat page.
5. The system Front-End shall subscribe unread message documents of user’s channel (Chatroom or Friends), subscribe means System Back-End would send any new updates of database to Front-End. Ex. When new message document is inserted to document.
6. The system Front-End shall render at most 100 messages chronologically from top to bottom with other users message on the left and users’ sent message on the right.
7. The user shall scroll up to load previous messages.

8. The system Front-End shall search local message history and render another 100 messages to the chat page.

Result: this test passes when the System displays the chat history chronologically on the screen and user could scroll down to see older messages.

Title: Chatting with strangers[C3]

Description: The feature will allow the user to have instant conversation with strangers.
User Goals: User wants to talk and make friends with other people.
Desired Outcome: Users could have instant and private communication with strangers.
Primary Actor: Application users
Dependent Use Cases:

1. Account Sign Up [UP1]
2. Chatting in a chat room[C1]

Actors:

1. User of application
2. System
3. Parse database

Requirements: System shall allow the users to chat privately with a stranger
Details:

Priority Level: Priority 1
Status: Planned

Pre-conditions:

1. User has logged in with the username testapp and password testpassword.
2. User has entered in a chat room.
3. Someone else(a stranger) has sent a message in chat room.

Post-conditions:

1. User will be prompted into a temporary chat page.
2. User could type in and send message to stranger.
3. User could receive message from the stranger.
4. The user and the stranger can't see the profiles of each other.

Failed Conclusions:

1. None

Workflow:

1. The user is inside chat room.
2. The user shall click the stranger profile icon they want to chat with.
3. The system Front-End shall redirect the user to enter a stranger's profile page.
4. The user shall click the lower right button chat
5. The system Back-End would add the stranger to user stranger list in database.
6. The system Front-End shall redirect the user to enter the chat screen
7. Everything would be the same as [C2] from now on.
8. Refer to [C5], for Chat with Stranger's behavior in recent chats.

Result:

1. User shall be redirected to temporary chat page after clicks on the stranger's icon.

2. User can send message to the stranger.
3. User can receive message from the stranger.

Title: Report suspicious clients[C4]

Description: User should be able to report suspicious clients so that they can keep the chat room clean and tidy.
User Goals: A user wants to remove inappropriate message out of chatting room.
Desired Outcome: The inappropriate message should be erased from chatting board.
Primary Actor: Users of application
Dependency Use Cases:

1. Chatting in a chat room [C1]

Actors:

1. User of application
2. System
3. Parse database

Requirements: System shall allow user to report suspicious users.
Details:

Priority Level: Priority 2.
Status: Not Implemented

Pre-conditions:

1. User has entered a certain chat room.
2. Some other users have posted inappropriate messages in the chat room.

Post-conditions: The inappropriate message shall be erased from the chat room.
Trigger: User doesn't want to see inappropriate message in the chat room.
Failed Conclusions:

1. The message is not erased.

Workflow:

1. User, testapp, shall long-press the inappropriate message.
2. System Front-End shall pop up "like" and "report" button.
3. User should click the "report" button.
4. System Back-End shall change the message's rate property in database. If a message's rate is low enough, Back-End shall erase the message and send a warning to the message sender.
5. User shall no longer see the message in the chat room.

Result:

1. The inappropriate message shall be erased from the chat room.
2. The inappropriate message shall be deleted from the database.
3. The user who sent this message shall receive a warning.

Title: Recent Chats[C5]

Description: User should be able to access recent chats with one click.
User Goals: A user wants to have a convenient way to access recent chat.
Desired Outcome: User finds 2 lists at bottom of logged-in main page, for chats with friends or stranger, and chats in chatroom respectively.
Primary Actor: Users of application
Dependency Use Cases:

1. Chat With Friends [F2]
1. Chatting in a chat room [C1]

Requirements: System Back-End shall keep a user recent chats stack at database
Details:

Priority Level: Priority 2.
Status: Implemented Partially & Tested

Pre-conditions:

1. Users have logged in.
2. Users have chats with people already and have history in those chat pages.

Post-conditions: Chats would rank by latest message time and are displayed at 2 lists at bottom of logged-in main page, for chats with friends or stranger, and chats in chatroom respectively
Trigger: User wants to access to recent or unread chats fast.
Workflow:

1. The user shall be at logged-in main page.
2. The system Front-End shall subscribe to user's recents chat from System Back-End, and display recent chat in given order.
3. The system Front-End shall show the red dot on the upper left corner of tag icon and friend icon
4. The user shall see the recent chats list for friends and chatrooms. They shall click on friends or chatrooms icon to engage in chats.

For Friends and Chatroom: (Implemented)

1. Whenever there are new message documents added to database from other users or chatrooms, system back-end shall push the sender or chatrooms on top of users' recent chats stack. So that friends or chatrooms with newest messages are on top of the stack.

For Strangers: (Not Implemented)

1. Stranger would came in with a validThru property, system Back-End shall check the property, If validThru time is past, system Back-End shall remove stranger from user document.
2. Otherwise, whenever there are new message documents added to database from strangers, system back-end shall push the stranger on top of users' recent chats stack. So that stranger with newest messages are on top of the stack.

Result:

1. The newest messages shall be displayed in the main page.

Title: Chat Notification[C6]

Description: User should be get notification on chats, even they are not using the app.
User Goals: User wants to get notification on chats, even app is not opened.
Desired Outcome: User shall receive notification on phone, if app is not opened.
Primary Actor: Users of application
Dependency Use Cases:

1. Chat With Friends [F2]
2. Chatting in a chat room [C1]

Requirements: System Back-End shall keep user's online offline status at database
Details:

Priority Level: Priority 2.
Status: Not Implemented

Pre-conditions:

1. Users have logged in.
2. Users are not using the app.

Postconditions: Users see the notification and can click on the notification and enter the App main page.
Trigger: User wants to get notification on chats, even app is not opened.
Workflow:

1. Another user, testapp2, send message to the user, testapp.
2. System Back-End shall add message to message collection, and check if receiver is online.
3. Receiver is not online, System Back-End shall send request to Firebase Messaging Cloud to push notification to the user.
4. If testapp2 shall receive the notification, and shall click the notification to go to chat page to engage in chats.

Result:

1. The notification of new massage shall be seen by the user.
2. The user who receives the notification should be redirected to the chat page with the newest message.

CHATROOM

This category of use cases outlines the using of chat rooms, like how to create and enter a chat room, to send messages, or to change your anonymous identifiers.

Title: Create Chatroom [CR1]

Description: User shall be able to create a chat room tag, if their cannot find one that is suitable for their event or situation.
User Goals: Create chatroom tag for their own event, and set a filtration on people who are able to discover this event.
Desired Outcome: Only targeted people can discover and enter the chatroom.
Primary Actor: User of Application
Dependency Use Cases:

Chatting in a chat room[C1]

Requirement:

1. System shall keep both tag and wifis associated with the tag in database
2. System shall be able to discover nearby wifis.

Details:

Priority Level: Priority 1.
Status: Implemented & Tested

Pre-conditions:

1. User's wifi services are working and allow our application to access.
2. User is searching for chatroom tags nearby.

Post-condition:

1. User creates a chatroom tag for their own event.
2. User set filtration that only at specified place and time can the event be discovered.

Trigger: User cannot find a existed chatroom tag which they expect.
Workflow:

1. User shall drag down "Anchor" button at the home screen of the application.
2. System Front-End shall display the chat room interface and search nearby wifis.
3. System Back-End shall send back list of chat rooms under the wifi nearby.
4. User shall click "Create a New Chatroom" if they cannot find suitable chatrooms.
5. System Front-End shall popup a modal containing fields: "Chatroom name, Chatroom description Chatroom Duration, Chatroom Repetition (Based on weekdays)".
6. User shall fill in all of the required fields and click "OK" to create the tag (The chat room location is set based on user's location, creation time of the chat room must be in the range of chat room's discoverable time interval).
7. System Back-End shall insert associated wifis and created chat rooms into collections
8. User shall be directed to the chat room interface.

Result: This test case passes when the user is directed in the new chat room in step 8.

Title: Enter a Chatroom [CR2]

Description: User shall see a list of chatroom tags they have access to, and choose which chatroom to enter.
User Goals: Get a list of chatroom tags, decide which ones they are interested in, and decide to enter those chatrooms or not.
Desired Outcome: User will see the chat rooms list but only enter the chat rooms when they want to.
Primary Actor: Application User
Dependency Use Cases:

Account Sign Up [UP1]
Create Chatroom[CR1]

Requirements:

1. System shall keep both tag and wifis associated with the tag in database
2. System shall be able to discover nearby wifis.

Details:

Priority Level: Priority 1
Status: Implemented & tested

Pre-conditions:

1. User has logged in.
2. User has connected to a WiFi.
3. System has tags for the user to choose under the current network environment.

Post-conditions:

1. User shall get a list of chatroom tags that they have access to.
2. User shall be able to enter any of those chat rooms.

Trigger: User wants to enter a chatroom and chat with people inside.
Workflow:

1. The user shall drag down “Anchor” button at the home screen of the application.
2. The system Front-End shall display the chat room interface and search nearby wifis
3. The system Back-End shall send back list of chatrooms under the wifi nearby.
4. The user shall select one of the available chatroom tag that they want to enter.
5. The system Front-End shall redirect the user to chatroom profile screen.
6. The user shall click “Chat” button at the lower right corner
7. The system Front-End shall redirect the user to chatroom screen

Alternate Workflow:

1. User enters to the chatroom homepage.
2. If the chatroom list is empty, go to Create a New Chatroom.

Result: This test case passes when the user is redirected into the chatroom in step 5.

Title: Change Anonymous Identifying Avatar [CR3]

Description: The anonymous identifier avatar is user's only identifier in anonymous chatrooms. It is randomly generated by system, and is independent of user's profile avatar.
Desired Outcome: User can change their anonymous avatars when they are unsatisfied with current ones. Change cannot be made again within 24 hours.
User Goals: Be able to regenerate a random avatar for anonymous chatting if they have not changed in 24 hours.
Desired Outcome: User can randomly change their anonymous avatar for no more than once a day.
Primary Actor: Application User
Dependency Use Cases:
None
Requirements: System shall allow the user to change their avatar in the group
Details:
Priority Level: Priority 3
Status: Implemented & tested
Pre-conditions:
1. User, testapp, has logged in.

Trigger: User and wants to change an anonymous identifier avatar.
Failed Conclusions:
User has already changed the identifying avatar within the 24 hours.
Workflow:
1. The user is at application's welcome screen
2. The user shall click on "Profile" icon.
3. The system Front-End shall display the user Profile interface.
4. The user shall click on the "Random" button next to current avatar.
5. The system shall display another random avatar for user.

Result: This test passes when the front-end will consistently display the newly generated profile picture in the Profile Setting interface, in the Main page, and in the Chatroom interface.

Title: Save Chatroom [CR4]

Description: System provides functionality for users to save the chat group.
User Goals: User can still chat in the group even if they are not in the same location where the tag was created.
Desired Outcome: User can chat with the people in the previous chatroom.
Primary Actor: Application User
Dependency Use Cases:
Enter Chatroom [CR2]
Requirements: System shall keep user saved chat room in database.
Details:
Priority Level: Priority 2
Status: Implemented & tested
Pre-conditions:
1. User has logged into the system with username testapp.
2. User has entered a Chatroom.

Trigger: User wants to have an easy access to their favourite chatroom.
Failed Conclusion:
None
Workflow:
1. The user shall follow the workflow described in CR2 to enter a chatroom.
2. The user shall click “anchor” icon at the top of the chatroom interface
3. System front-end shall redirect user to chatroom profile page.
4. Click subscribe button at lower left corner.
5. The system Front-End shall display a pop-up message “Saved” at the bottom right of the chatroom interface
6. The system Back-End shall add the chatroom to saved list in collection.
7. The user shall see themselves as member of chatroom in the interface immediately.

Result:
This test passed when the Anchor icon changes as step 3, and user could perform the following actions as described in step 4.

Friends

This group of use cases outlines the structure of managing friends, including adding and deleting friends, getting friends recommendation, and personal chat with added friends, sharing personal information like Facebook with added friends.

Title: Add Friends [F1]

Description: User can add friends through system recommendation.
User Goals: A user can be a friend with another user.
Desired Outcome: Two user become friends in app and can chat one-to-one.
Primary Actor: User of application
Dependency Use Cases:
Friends Recommendation [F4]
Requirements: System shall have a friends list of user in the database.
Details:
Priority Level: Priority 1
Status: Implemented & tested
Pre-conditions:
1. User has login with an account testapp.

Post-conditions:
1. User shall see new friends on friends list.

Trigger: User wants to add another user as friend.
User Workflow:
1. User shall select the user he or she wants to add as friends.
2. System Front-End redirect user to target's profile page.
3. User shall select "Add Friend" button.
4. System Front-End shall display "Request has been sent" message.
5. System Back-End shall add user to target's friend request list.
6. System shall then refresh the Friends Recommendation Page (do not show the user who has already been added).

Target Workflow:
1. System Front-End shall notify target about friend request on My Friends page.
2. User click on "Accept" button, to add the friend.
3. System Back-End shall add the user to target's friend list.
4. Target shall be able to see user in friend list.
5. System Back-End shall send message to user representing target "I have accept your request, let's chat" [C1]
6. User click on "ignore" button, to ignore the request.
7. System Back-End shall move user to target's turn down friends list, with a date. So that in a time interval target won't see request or recommendation of the user.

Result: this test passes when the newly added could be seen in the friend list

Title : See info of Friends [F2]

Description: System provides functionality for users to see info of their added friends.
User Goals: A user likes to get in touch with his friends in another app.
Desired Outcome: Two users can add each other on Facebook or other social media connected.
Primary Actor: User of application
Dependency Use Cases:
Add Friends[F1]
Requirements: System shall allow a user to check friends' information.
Details:
Priority Level: Priority 2
Status: Implemented & tested
Pre-conditions:
1. User has login with an account testapp.
2. User has added at least one friend.

Post-conditions:
1. The info of the friend will show up on screen.

Trigger: User wants to check friend's personal information.
Workflow:
1. The user shall click on friend's icon.
2. The system Front-End shall redirect the user to profile of that friend page.
3. The system Front-End shall retrieve the user data of the friend from database and render for the users.
4. The user shall see friend profile including social media accounts and etc

Result: this test passes when the user can see friend profile page in step 5

Title: Get Friend Recommendations [F3]

Description: System provides functionality for users to receive friend recommendations. Recommendations are based on common routines.
User Goals: A user wants to know people who have some common schedules with him or her.
Desired Outcome: User gets recommendations of some users who share similar routines.
Primary Actor: User of application
Dependency Use Cases:

Account sign up [UP1]
Save Chatroom [CR4]

Requirements: System keep subscribed user under tag document.
Details:
Priority Level: Priority 1
Status: Not Implemented
Pre-conditions:

1. User has logged in.
2. User has saved or joined tags.

Post-conditions:

1. Users get their friend recommendation automatically.

Trigger: User wants to make friends with people who share similar daily routines.
Failed Conclusions:

1. The user have either accept or ignore the recommendation for the day.
2. The user have either accept or ignore all possible recommendation for the week.

Workflow:

1. Create 2 test accounts, test_F3_1 and test_F3_2. (Account Sign Up [UP1])
2. Create 1 test chatroom, test_F3. (Create Chatroom [CR1])
3. Let test_F3_1 and test_F3_2 join test_F3. (Save Chatroom [CR4])
4. Using account test_F3_1, tap People Icon at bottom nav bar, system shall redirect user to friend page.
5. test_F3_1 shall see friend recommendation at top of page for test_F3_2.

Result: this test passes when step 5 is executed successfully

Failed Conclusion#1 Workflow:

1. Continue from last step of previous workflow, tap red cross icon on the recommendation to ignore this recommendation.
2. Recommendation shall disappear.

Result: this test passes when step 2 is executed successfully

Failed Conclusion#2 Workflow:

1. Continue from last step of previous workflow, continue from Change system clock of test_F3_1 account to 3 days laters.
2. Repeat step 4, recommendation shall remain invisible.
3. Change system clock of test_F3_1 account to 7 days laters.
4. Repeat step 4, recommendation shall reappear.

Result: this test passes when step 4 is executed successfully

Title: Recommendation Visibility Setting [F4]

Description: System provides functionality for users to choose whether they will appear on other users' recommendation lists or not.
User Goals: A user doesn't want to be discovered by other users.
Desired Outcome: If users wants to be completely "invisible" to others, they won't appear on other users' friend recommendation lists.
Primary Actor: User of application
Dependency Use Cases:
Get Friend Recommendations [F4]
Requirements: System keep user's visibility setting in database.
Details:
Priority Level: Priority 3
Status: Not Implemented
Pre-conditions:

1. User has logged in.
2. User has entered Set Profile page.

Post-conditions:

1. User shall see their visibility mode changed.
2. If users set account to be "invisible", System Back-End will remove user from any recommendation lists of other users.

Trigger: Users want to change their visibility when using this app.
Workflow:

1. Create 2 test accounts, test_F4_1 and test_F4_2. (Account Sign Up [UP1])
2. Create 1 test chatroom, test_F4. (Create Chatroom [CR1])
3. Let test_F4_1 and test_F4_2 join test_F4. (Save Chatroom [CR4])
4. Using account test_F4_2, tap Cogs Icon at bottom nav bar, system shall redirect user to Profile page.
5. User shall toggle Visibility Button, to prevent being recommended.
6. Using account test_F4_1, tap People Icon at bottom nav bar, system shall redirect user to friend page.
7. test_F4_1 shall see no friend recommendation.

Result: this test passes when step 7 is executed successfully

Title: Delete Friends [F5]

Description: System provides functionality for users to delete friends at any time.
User Goals: A user needs to have the option to delete friends in this app.
Desired Outcome: Two users will be removed from each other's friend list if at least one of them choose to delete the other.
Primary Actor: User of application
Dependency Use Cases:
Add Friends [F1]
Requirements: System keep user's friend list in database.
Details:
Priority Level: Priority 2
Status: Implemented & Tested
Pre-conditions:
1. User has logged in with an account test.
2. User has added at least one friend.

Post-conditions:
1. The chosen user is removed from the friend list.
2. User who ordered this delete movement is removed from target user's friend list

Trigger: User feels uncomfortable interacting with their "friends".
Workflow:
1. The user shall enter Friend page.
2. The user shall click a certain friend's profile picture.
3. The system Front-End shall display the friend's profile information.
4. The user shall click the "Unsubscribe" button.
5. The system Back-End shall remove the selected friend from the user's friend list, and also remove the user from that friend's friend list.

Result: This test passes when the front-end will consistently display the friend list in My Friends page.

Title: Add Nicknames [F6]

Description: System provides functionality for users to add nicknames to friends.
User Goals: A user wants set or change a nickname to a friend, so that user can recognize that friend.
Desired Outcome: A user can set change the nickname of their friends in the friend list.
Primary Actor: User of application
Dependency Use Cases:

Add Friends [F1]

Requirements: System keep user's friend's nicknames in database.
Details:

Priority Level: Priority 3
Status: Implemented & tested

Pre-conditions:

1. User has logged in with an account testapp.
2. User has added at least one friend.

Post-conditions:

1. A certain friend's profile name will be replaced with this nickname.

Trigger: User wants to add nicknames to their friends.
Workflow:

1. The user shall enter My Friends page.
2. The user shall click a certain friend's profile picture.
3. The system Front-End shall display the friend's profile information.
4. The user shall click the "Add Custom Name" button.
5. The system Front-End shall display a input box for the user to enter custom name
6. The user shall enter a nickname and click the button "save".
7. The system Front-End shall replace the friend's profile name with custom name.
8. The system Back-End shall update the friend's custom name in user's document.

Result: This test passes when the nickname is showing up on friend's profile screen. And, if the friend has a nickname, then the friend's name on My Friends should be replaced with nickname.