Team-Up!

(Sprint 2 Retrospective)

https://github.com/srhee91/Team-Up

Travis Coria, Trevor Coria, Boheon Jeong, Yiyang Pan, Sang Rhee, Kartik Sawant

Sprint Retrospective

What went well?

We completed all of our sprint objectives and didn’t run into any major roadblocks.

- Implement *MyPreferences* class
    - The user can add/remove preferred categories. Change Preferences option is created in Settings tab. The preferences will be updated real time whenever user clicks and makes changes.

- implement *RecommendGroups* class
    - By adding preferred categories, the user can view a list of groups in these preferred categories in “Teams for You” option in Group tab.

- Implement Nearest Group functionality
    - The user can find five closest groups sorted by distance from his/her current location. The range is set to a limit of 10 miles to prevent unrelated groups far from the user. The list is sorted in the ascending order, meaning that the closest group will list on the top of the screen.

- Implement *Invite* class
    - The user can invite a member by inputting his or her username. If the username exists, then it will make an additional entry in the database. Every time a user goes to his or her profile page, the front pulls and searches the invite database to check if the user’s username has been invited. If he or she is invited, a popup box will appear asking if the user wants to join the group. Yes means the user will be added to the group and the entry will be deleted from the database. No means only the entry will be deleted from the database.

- Implement *Search* class
    - The user can search for groups and categories. The search bar provides auto completion using a simple algorithm, and will display all groups and categories that match the search query. Private groups are displayed, but they are not able to view the group's detail unless they are in the group.

- Implement *Meeting* class
    - The user can create meetings for any group that they are an admin of. Any user that is in that group can view any meetings that have currently been set. If the meeting time is in the past, the meeting will not be shown.

- Implement *Members* class
  - The admin of each group has authority to view list of members in the group through clicking member button in the group information view. If member button is clicked, admin of its group is directed to a tableView class where admin can transfer its admin status to one of group's members and kick a selected member out of the group. Whenever either admin is changed or a member is kicked, database is immediately updated with new data.

- Implement *Privacy* functionality
  - The privacy setting of group can be declared by the admin of the group at the creation or using edit button in group information view when group is already created. Each group has privacy setting either public or private. If a group is public, any user can join or view the group and its information. If a group is private, only invited users can join or view the group and its information. Whenever privacy setting is either created or changed, the group's privacy attribute in Group class is immediately updated in the database.

- Implement *Leave* functionality
  - The user can leave the group, which one has joined. If the user clicks the group, which one already join, leave button appears at the top right of group information view. By clicking a leave button, the user can leave the current group. The user is accordingly deleted in Member class of the database. If the user that left was the admin, the admin status is given to the oldest member.

- Implement *UploadGroupImage* class
  - The user can upload a picture to be used in their group by specifying an image URL. If the URL is a valid image, it will be set to the group's profile image. This image is displayed on the group's profile page.

- Implement *Chat* class
  - The chat can be accessed by tapping on the third tab on the bottom of any page. A user then picks a group that they belong to and he or she can chat with everyone in the group. The Chat page refreshes every 5 seconds and update/display the newly retrieved messages sent by other users.

What did not go well?

- In this sprint, we focused on features implementing. The UI design was kind of being ignored. We do realize a good UI design will greatly improve user experience. We will dedicate more time to improve it in the next sprint.

- Parse Database only handles 30 requests per second. When our application uploads or pull more than 30 objects at a time, Parse Database is not updating immediately or our application performs the process before all of data are fully pulled from Parse Database. Our application and Parse Database are not perfectly synchronized. We need to perform more testing to synchronize application with Parse.

- Incomplete user stories
    - None

How should you improve?

- Come up with a better UI design and implement it

- Create more testing cases for potential bug fix

- Robustness assurance