Final Documentation

1. Description

Many people today are afraid to explore different fashion options because they are not sure how their outfits look. We wanted to write an app that would allow users to upload photos of their outfits and get feedback from other people. This app roughly emulates Tinder's simple interface, which allows the raters to quickly approve or disapprove of an outfit and quickly give custom feedback.

2. Process

The Agile programming methodology was adapted by our team in various ways. For example, we did not use pair-programming because the majority of our team are experts in the domain of mobile applications so it would be counter-productive and time-consuming to do so. Our process was streamlined in order to parallelize development as much as possible. We did this by using a new versioning control system, GIT, rather than SVN. We made use of the branching and tagging system heavily in order to facilitate independent workflows and eventually smooth integration. Each programmer used the “GIT-flow” paradigm and created an independent feature branch to work on his/her assigned user story. The programmer continues to make frequent commits to their local feature branch until it is completed following which a pull request will be initiated and reviewed by another team member for integration into the main development branch. Testing-driven development, another aspect of the Agile methodology was also modified by our team.

Due to the fact that mobile applications are heavily GUI-based applications, it would be extremely difficult to implement Test-driven development. User interfaces are notoriously hard to test because of the number of possible state combinations. Instead, we used an end-to-end testing plan to test our UI after it was built. Additionally, we refactored the business logic and the presentation logic from the UI in order to implement whitebox testing for logical code. The bulk of the refactoring done by our project was necessary in order to write tests but resulted in a clean and modular code base. Our team also included technical debt for refactoring for several of our sprints. There. Besides Test driven development, our team modified the iteration cycles used in Agile development.

Our group opted for short iteration cycles or “sprints” in the vein of the SCRUM methodology. Last semester, we were only accountable for meeting each iteration meeting which hindered the momentum of the project. It was also extremely difficult to integrate our work in the version controlling system because of the extended periods of isolated development. This semester, we corrected this issue by integrating our work as frequently as possible at our iteration cycle meetings. We believe this method was more effective because

it held each member accountable for our assigned features. Overall, It addressed risk items as soon as they appeared and corrected our projected timeline. Our sprint cycles were one week long, but our group held meetings twice a week to correspond on recent progress (Once with a TA and once on our own).

3. Requirements & Specifications (User stores & Use Cases)

User Stories:

1. As a user, I want to be able to register and log in with my email
2. As a user, I want to be able to register and log in with Facebook
3. As a user, I want to be able to submit a photo to the app
4. As a user, I want to be able receive feedback on photos which I've submitted
5. As a user, I want to be able to specify audience from whom to receive feedback
6. As a user, I want to be able to view other submission photos
7. As a user, I want to be able to judge other submission photos with a "Like" or "Dislike" vote
8. As a user, I want to be able to comment on submissions

Use Cases:

1. As a user, I want to be able to register and log in with my email

- Primary Actor: User
- Goal in Context: Register new users for the application and login users that have the correct email and password.
- Scope: Authenticate users so that all actions performed on the application are tied to a specific user.
- Level: Summary
- Stakeholders and Interests
    - User: Wants to make sure that all their activity is saved under their username so that they can receive input from other users.
- Precondition: Backend Server must be up and running
- Minimal Guarantees: Always ensure that a user cannot register two accounts with the same email. It also ensures that a user will not be able to login without the correct email and password combination.

- Trigger: The user must not be logged in to the application. He then must enter his login credentials and click the login button.
- Success Scenario Trigger:
    - User enters his email into the application
    - User enters his password into the application
    - User clicks the “login” button
    - Backend system authenticates email and password combination
    - Backend system sends the “all okay” message to the application indicating a
    - successful login. The user is logged in.
- Extensions
    - 5a: Email and password combination incorrect so the backend system sends the “wrong password” message to the application. The user is not logged in.

2. As a user, I want to be able to register and log in with Facebook

- Name: Log in with Facebook
- Primary Actor: User
- Goal in Context: Allow new users to register or existing users to login with Facebook authentication.
- Scope: Authenticate users so that user-specific information can be tied to a user account.
- Level: Summary
- Stakeholders and Interests: User wants to load all of their data that was tied to their account.
- Preconditions:
    - The user should have the application installed
    - The application is running with the first screen showing upon entering the application
- Minimal Guarantees:
    - Ensures one account is associated with a specific Facebook account
    - Ensures user will not be able to login without their Facebook credentials
- Success Scenario Trigger::
    - User must be entering the application for the first time or be in logged out state upon entering the application’s initial screen
- Success Condition:
    - User chooses to login with Facebook
    - User specifies Facebook credentials
        - Extension Failure - User enters incorrect Facebook credentials
    - Facebook authenticates credentials

        - User authorizes permission for application to access Facebook information
            - Extension Failure - User does not authorize permission for application to access Facebook information
        - User logs in with Facebook
        - User is notified with a successful login

3. As a user, I want to be able to submit a photo to the app

- Primary Actor: User
- Goal in context: Submit a photo to the app
- Scope: Submitting - the primary submission criteria used for later judging and portfolio reviews
- Level: Submitting
- Stakeholders and interests:
    - Users as submitters: wants to have submission judged by others
    - Users as discoverers: wants to judge and discover other user photo submissions
- Precondition: user is logged in
- Minimal guarantees: only logged in users can submit
- Success Condition
    - Logged in User as Submitter: chooses to make submission
    - Logged in User as Submitter: captures photo
    - Logged in User: chooses to submit photo
    - Submission System: accept photo and display other submissions by submitter in question
- Extensions:
    - a1. Photo is not to liking: user would like to recapture photo
    - a2. Photo is not sufficient: user would like to add additional photos
    - a3. Default submission criteria is no accurate: user would like to specify new criteria

Use Case 4: As a user, I want to be able receive feedback on photos which I've submitted

- Primary Actor: User
- Goal in context: Receive feedback on photos which I've submitted as a user
- Scope: Feedback - the way users will receive input on their submissions
- Level: Submission Judgment
- Stakeholders and interests:

        - Users as submitters: wants to have submissions commented on and judged by others
        - Users as discoverers: wants to judge and leave feedback for other user submissions
    - Precondition: none
    - Success Scenario Trigger:
        - Logged in User: chooses to view submissions portfolio
        - Logged in User: chooses photo from portfolio to see further details
        - Logged in User: chooses to view comments for submission\
    - Extensions:
        - a1. Desired submission not originally displayed: user loads next page of photos

Use Case 5: As a user, I want to be able to specify audience from whom to receive feedback

- Primary Actor: User
- Goal in Context: Allow a user to specify the audience s/he wants feedback from.
- Scope: Feedback System - the way users will receive input on their submissions
- Level: Summary
- Stakeholders and Interests: User wants to be able to receive feedback from a specific audience.
- Preconditions:
    - The user should have the application installed
    - The user is logged in and authenticated.
    - The user has taken a photo and is selecting the options for that photo.
- Minimal Guarantees:
    - The audience that the use selects will be the audience that will see the submission.
- Trigger:
    - User takes a photo for submission
- Success Condition:
    - User is logged into the application
    - User chooses to submit a photo
    - User takes and selects a photo
    - User selects the desired audience for photo submission

Use Case 6 - As a user, I want to be able to view other submission photos

- Name: View Submitted Photos

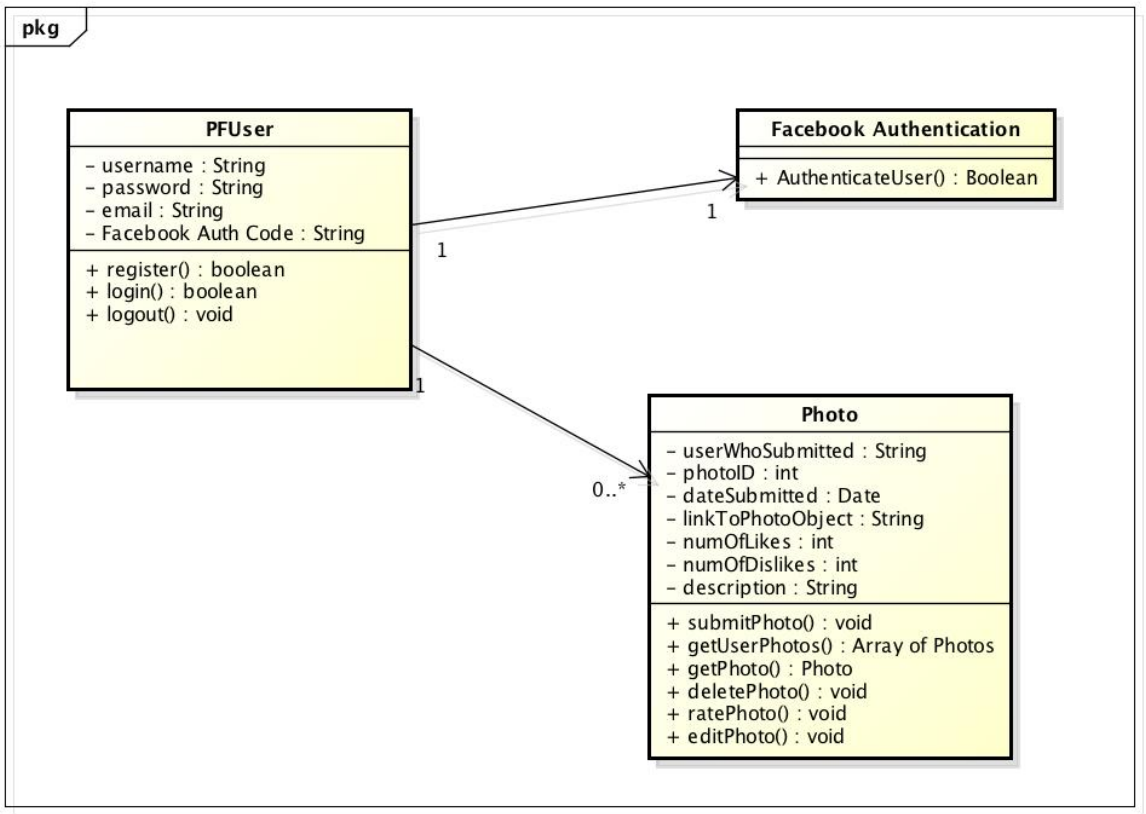
- Primary Actor: User
- Goal in Context: Allow a user to view photos submitted by others.
- Scope: Discovery mode, where you can view other submitted photos.
- Level: Summary
- Stakeholders and Interests: User wants to be able to see what photos others have submitted.
- Preconditions:
    - The user should have the application installed
    - The user is logged in and authenticated.
    - The photos displayed are following the submitters preferences.(ie Girl/Guy only)
- Minimal Guarantees:
    - Ensures a user can view a photo submitted by another user.
    - Ensures user will not be able to login without their Facebook credentials
- Trigger:
    - User must be in the application and choose to discover photos.
- Success Condition:
    - User chooses to discover photos
    - User specifies preferences
        - Extension Failure - User finds no matches to their preferences
    - Photos are displayed to user
    - User can choose a photo to display further information/photos by submitter
        - Extension Failure - No further information

Use Case 7 - As a user, I want to be able to judge other submission photos with a "Like" or "Dislike" vote

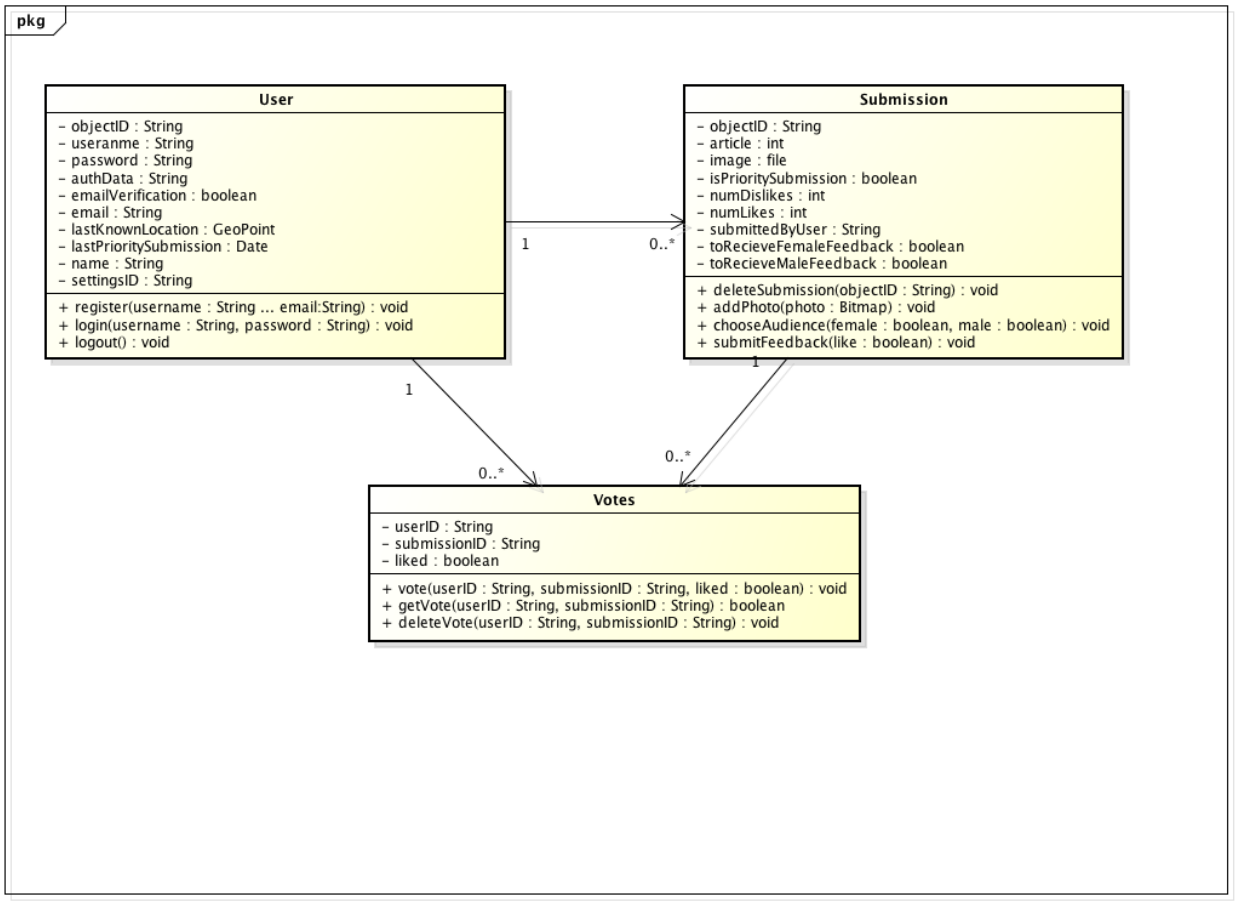
- Primary Actor - User
- Goal in Context - To like to dislike another person's outfit
- Scope - Outfit statistics. When someone likes or dislikes a person's photo, all that data is stored and displayed to the original outfit submitter to give them an overview of the general consensus of an outfit and if more people like or dislike an outfit.
- Level - Summary
- Stakeholders and Interests
    1. User who submits photo - wants to be told if in general, people like the outfit
    2. User who rates photo - wants to see new and interesting outfits and be able to rate outfits
- Precondition - User who is rating photo must be logged in

- Minimal Guarantees
    1. A user will always “like” or “dislike” a photo, there is no other option.
    2. Each photo will always be submitted by a user and tied to a user account.
    3. A right swipe will be a “Like” and a left swipe will be a “dislike”
- Triggers
    1. As soon as the user is logged in and authenticated, the “home page” of the app will start displaying photos for the user to “like” or “dislike”.
- Main success scenario
    1. User swipes left or right on a submitted outfit photo
    2. Backend will be notified of the action and rating
    3. The screen will display a green checkmark for “Like” or a red X for “Dislike”
    4. A new photo of an outfit is displayed
- Alternative scenario
    1. Application cannot authenticate/reach server, app will say “can not connect to server”

4. Architecture & Design *

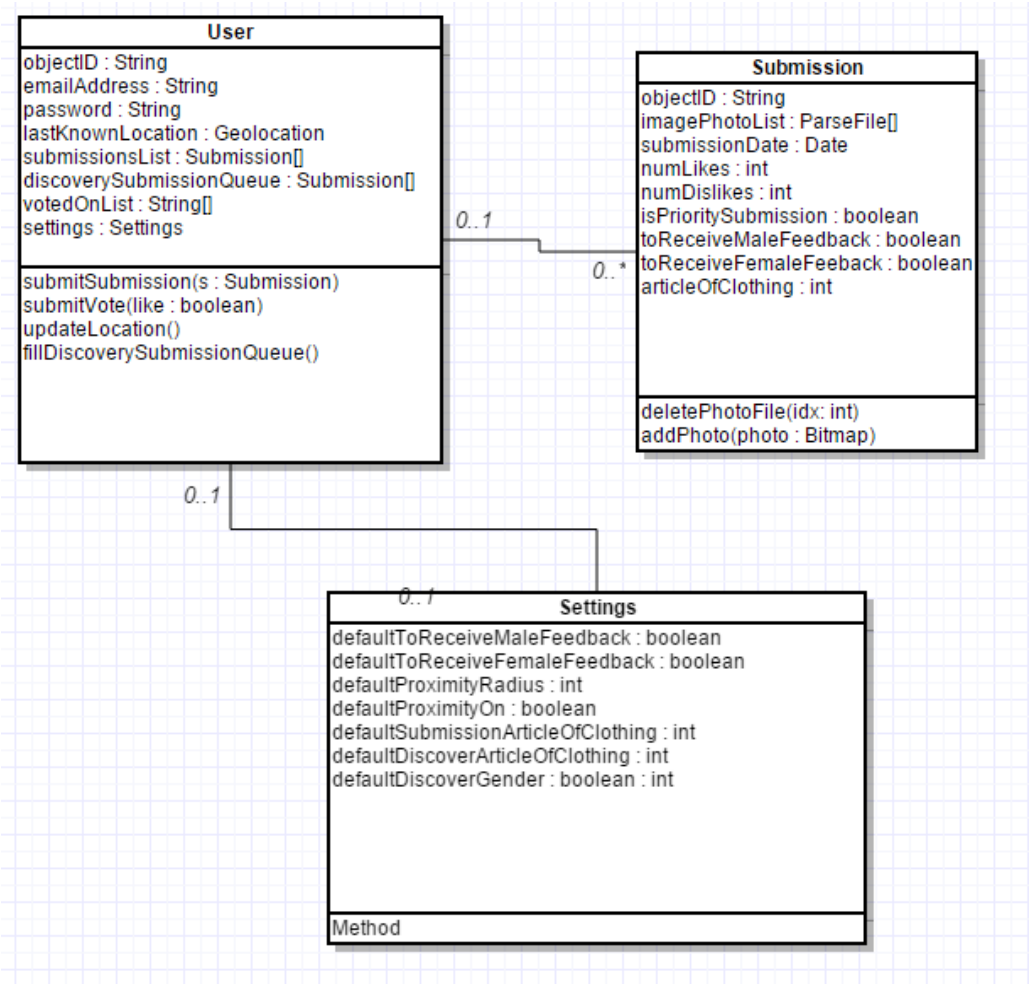


Class Diagram for Use Cases 1,2,3



Class Diagram for Use Cases 4 & 5

*: Classes for each use case may differ slightly by including only the necessary attributes in order to perform their use cases.



Class Diagram for Use Cases 6 & 7

iOS Application Framework:

Our application uses iOS framework which recommends model-view-controller paradigm. We followed this paradigm by creating our (view) interface in StoryBoards, by querying Parse for our model layer, and finally by creating view controller to interface between the two and perform business logic.

iOS Application Architecture:

App Delegate:

The App Delegate is the entry point for our application, it is where we setup all our API's and third party frameworks. The App Delegate hands off control to the storyboard to setup the first screen.

StoryBoard:

The StoryBoard is a file that displays the flow of the application as well as the interface for each screen. Each screen has an interface that is built in the storyboard and corresponds to a view controller where the application's logic resides.

View Controller:

The View Controller contains outlet actions to respond to the interface built in the storyboard. The View Controllers are also responsible for making calls to the Parse backend and refreshing the interface when a response is received. View controllers also contain logic to handle native frameworks in order to do things like access the camera or photos on a user's phone.

Cells:

Cells are custom classes that are created to handle displaying custom data in a View Controller. There are currently two cells in the project: One for displaying comments and one for displaying photo submissions.

Test Files:

There are two different types of testing files in our project. The first kind is a swift file that is named _____Tests.swift. These kind of files include white-box testing for all our major functionality in each corresponding view controller. The other type of file is named _____Test.js. These kind tests are used for end-to-end automated UI testing.

Android Application Architecture:

Gradle:

The Gradle scripts contain all the setup information for our application. This handles third party frameworks and APIs. It also handles building our app.

Resources:

The Resources contain all the xml files that define our application. This is how the UI is designed. The files contain drawables, layouts, menus and values. These files define the ID's that are stored in R for use in our application code.

Java:

The Java files contain 3 types of code files: Activities, Adapters, and Data. Activity code files define the behavior of the application and provide the functions that are interacted with by the interface. They are effectively the view. Adapters are used as bridge between views and data in our application. Data files, like User and Submission, are just objects that handle data we need to store in one place.

Manifests:

The Manifest file defines the main features of the application. This includes activities, and how the application acts on a device.

Test Files:

There are two different types of testing files in our project. The first kind of files are standard junit tests and are white box tests used to tests our data classes and our named ____Test.java. The second kind of files are automated UI tests. This files allow for black box testing of the UI and we use Espresso for this. These files follow the same naming convention of ____Test.java.

Backend:

We used Parse in order to facilitate the backend of our application. This influenced the design of our system because all of the calls to Parse were made to the background and were asynchronous. This allowed our application to run without blocking the UI.

5. Future Plans

- Include personal reflections on the project and process by each team member. Describe what you learned.
- Yusuf Sobh's Reflections:
    - Overall our process worked really well and we were able to work in parallel without much merge conflicts because we merged our work early and often. Something that I learned was that UI Automation Testing is the best way to ensure the UI is working correctly. The UI Automation Testing can also be used to perform end-to-end tests in an iOS Application. Another thing that I learned is that the building the UI with interface builder is not very intuitive and we could have saved a lot of time if we created all of the UI programmatically instead.
- Ian Eckles' Reflections:
    - Overall this class taught me that you must be organized and communicate with your teammates. This class builds off of CS 427. In CS 429 we needed to have even better organization and communication because our group had to provide all of the aspects of the project. With that said, there were three technical things that this project has taught me. First, I learned how to effectively use Git for a group project. I had always just used version control for turning in programs for grading, I never truly used it for version control. This project really made me learn Git and so now I am confident with my version control skills. The second thing I learned was the new programming language Swift. I had programmed a

little in Objective C but I haven't created an iOS project since swift was released. Sadly swift and xCode are still buggy and that was an annoying inconvenience. The last thing I learned was UI Automated Testing. I have used unit testing in the past. This type of testing is very useful for backend coding but we found it difficult to use for GUI testing. For this reason we learned UI Automated Testing which was very interesting at first. Now that I have used it in multiple iterations I am pretty comfortable with it.

- Michael Luo's Reflection:
    - This project helped a lot to develop my personal skills as well as team skills, from learning how to make iOS applications to communicating with a team. One of our biggest challenges was with communicating about all our code and actually sharing and moving our code around. We used a version control program called Git and although we all had high level ideas of what our code should do, merging our code and functions together and helping each other use our code was a different story. Not only did we have to figure it out with the iOS team, but because we also shared a backend with the Android team, we had to communicate with the Android team about any backend and schema changes we made. Overall, it took a lot of planning and communication but we were able to smoothly work out everything and made sure each person knew what was going on. Another challenge we tackled head on was advanced GUI testing. Since we've always learned about unit testing, such as JUnit, we were very well versed in automated unit tests but one of the things we spent a lot of time on was automating GUI tests, from clicking buttons and asserting certain things were on the screen. We learned a lot and figured out just how hard making the GUI and testing the GUI, which I personally thing is something that is hugely overlooked in the modern CS curriculum. Overall, I learned a lot and enjoyed my experience working with the iOS team and building our "Outfitter" app.
- James Maguire's Reflection:
    - Overall this was a much more rewarding experience than 427. Though a necessary evil, CS 427 taught us why a usable, intuitive, well-organized code base is crucial. I think the team's approach to tackle the project was well implemented. We were very organized from the start with a clear idea/end product in mind and that was extremely helpful to meet deadlines and create an app at a steady pace. We managed to stay consistent across platforms throughout development and more or less on pace with each other. Personally, this was my first, full app experience developing for Android. It was a challenge, especially image manipulation but an overall worthwhile effort and experience. It was also great exposure to using a database tool--we were able to see the power of existing SDKs and services available to developers. We surprisingly

met all of our deadlines so the expectations we set for ourselves felt just right, not overestimating our capability but also not underestimating it. Challenges arised and slowed progress at times, but none that we were not able to work out in the end.

- Kris Kocinski's Reflection:
    - This semester has been a great learning experience in working as a team on a project. We were required to do everything from scratch, and this involved doing a good amount of research on what type of APIs and frameworks we could use. As the semester went on, we ran into challenges with workload and motivation. The android team had Git issues and we also had to migrate to a private repo. We conquered these challenges by reorganizing. First, we revamped our Wiki which helped in organizing all our notes. Then, we adjusted meetings to be more flexible, instead of locking ourselves to one particular time. I personally learned a about android testing, both basic and advanced tests. We used Espresso for advance GUI testing, which was cool to use because it automatically tests the UI for you. We used Parse for our backend, which alleviated a lot of challenges, and made it easy to share a backend with our iOS team. Overall, it was an enjoyable experience that has made good interview story material.
- Mihir Angal's Reflection:
    - This semester really taught me the value of good Software Engineering in team projects. I have worked on Android apps in the past, but usually I only worked on the app alone or in a small hackathon crew, and was never able to make very large or significant apps. This time I was able to build something really amazing by properly distributing work and using Git. Admittedly Git was an issue at the beginning because we had to move repositories and we kept committing .iml files which are meant to be machine specific. This caused a lot of issues getting code working on all our machines. We eventually got a hang of Git and were able to use it very well. I also gained experience understanding Android code written by others because I had to integrate my changes across all the code that was already written. I also learned more about Android testing, something that I had rarely touched before this project. All in all this project really prepared me for entering industry as an Android developer, and I believe I will be able to leverage what I learned here to become a much better industry developer in the future.