FLOAT

Design Document

Aaron So 17150137
Clarence(Junwei) Su 36387132
Ming Hin Matthew Lam 33056145
Rachel Yeo 30032122
Richard Xie 55786140
Samuel Farinas 17721144
Selina Suen 14022140

Introduction

During the course of this project, our goal is to construct the concept of Float as an Android application. With this application, users will be able to view, contribute to and start charity campaigns. As a consequence of how these charity campaigns are structured, the app itself is to rely heavily on the location of each user. To help support this, each user is to have an individual account, allowing the preservation of information and past movements.

All design decisions for the system were made carefully following team discussion. Choices made were heavily influenced by difficulty of implementation, cost and suitability, as we aimed to find an agreeable balance. The purpose of this design document is to explain the architectural and design choices that were made using the 4+1 architectural model.

Architecture and Rationale

Android

Android is a mobile operating system developed by Google, and is designed primarily for touchscreen mobile devices. We decided to choose the Android platform over the also prominent iOS platform because the majority of us are familiar with Java, which makes completing the project within the time constraint of two months more feasible.

Facebook Log-in

Facebook supplies a login API, which serves to provide a secure means to log a user into an application. We chose to use a login API as opposed to implementing our own login system in order to ensure that security standards are met while authenticating users. Using Facebook to aid in user login will also improve ease of use, as many people in today's society already own Facebook accounts and will not have to spend additional time to create a new account for the application.

PayPal

Float needs to deal with the exchange of payments between users and charities. Because Float is not aimed at a certain geographical region, choosing a payment method that is widely used is necessary. The storage of the payment information must also be handled securely. For these reasons, PayPal became the obvious choice for handling payments. PayPal offers an Android SDK with extensive documentation that allows developers to accept PayPal and credit card payments on their application.

Database

For the database, we were originally debating between two main options: NonSQL and MySQL. We initially decided to implement our database in MySQL for the following reasons.

1. The main advantage of NonSQL compared to MySQL is that NonSQL provides a larger degree of flexibility to alter the structure of databases in the future. This can result in reckless decision-making on the database structure design. Given that our project has a relatively simple data structure (three entities), volatility of the data structure is low. That being said, choosing MySQL over NonSQL will force us to put in more thought and consideration into the initial design
2. It is relatively simpler to find existing tools for MySQL, and the queries in MySQL are optimized to provide good performance.
3. If there ever comes a time where we desire to switch from one database to another, it would be easier to shift from MySQL to NonSQL.

However, as the project progressed, we realized that MySQL was no longer a valid option as Android does not support MySQL. As an alternative, we came upon Parse. Upon further investigation, however, it was also discovered that the Parse hosted service would be retiring on January 28, 2017. Following this, we then did further research and discovered Firebase, which is what we are currently using to implement the database.

Data

The data that will be stored in the database can be partitioned into three categories: users, campaigns, and actions. Since the users are the center of our application, the user database stores the most important information. The user database links to the campaign and the action databases, and IDs in different databases are the keys for communications between databases. The user database stores all the key information for a user including one's ID, name, account, lists of campaigns and actions and location.

The user's list of campaign links a user to the lists of campaign of either the user initialized or participated. In the campaign database, the important information is detailed here. For each campaign, the key people are initializers and participators. It also store the description and locations of the campaign since they are also the critical components.

In the action database, we decide it is necessary to record each action that a user makes, so we can keep track of the history. For each action, we need IDs for the user and the campaign that the user interacts with. For the recording purpose, date and location are important as well.

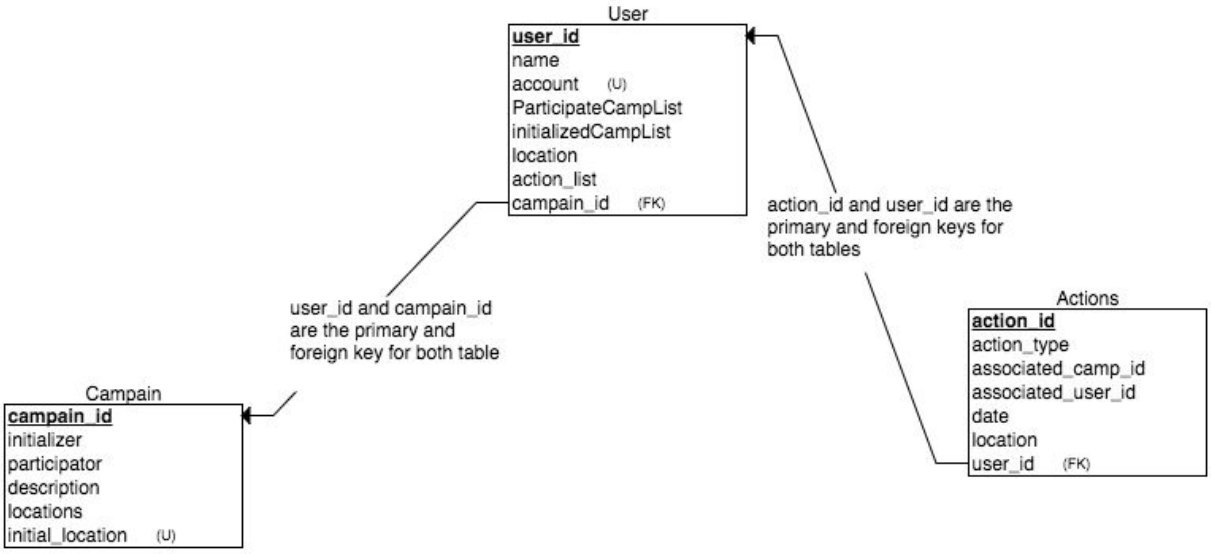

Figure 1. Diagram of the data that will be stored in the database, as well as possible interactions.

Class Diagram

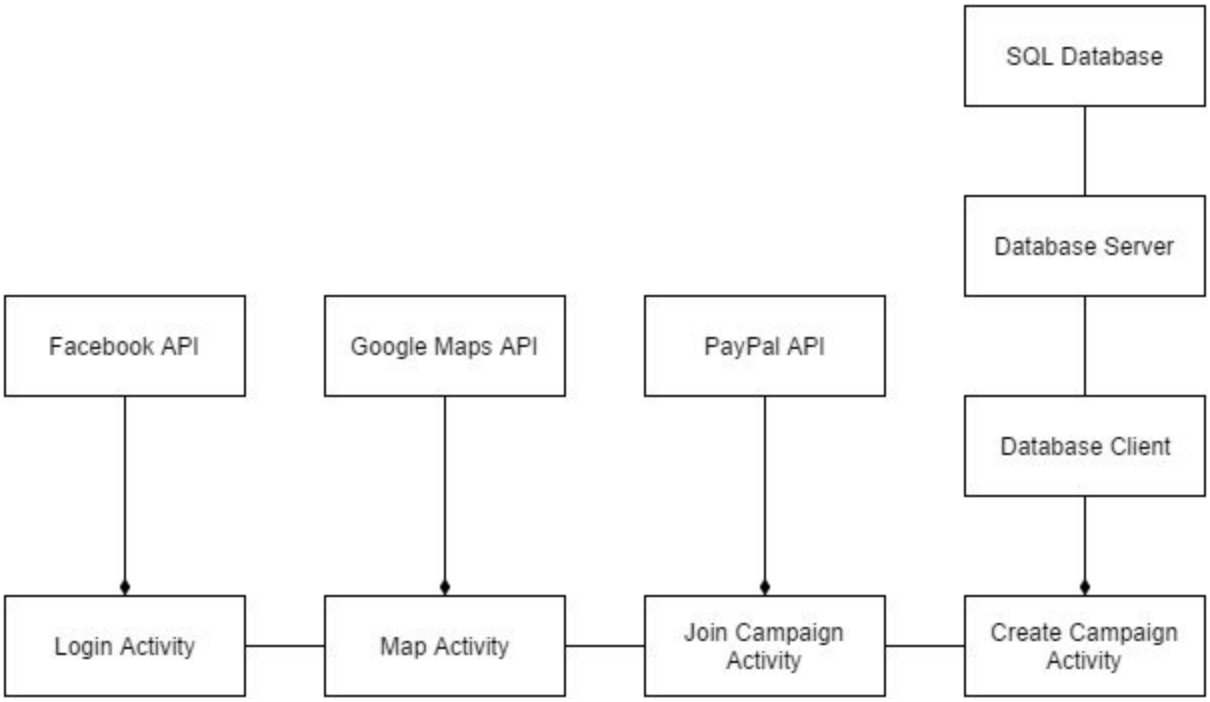

Figure 2. Class diagram for the different subsystems.

GUI

(1) Facebook Login

(2) Main map page that shows all the campaigns in a user's area, given by the map pointers. The user can switch to a list view of the campaigns by toggling the switch on the top left

(3) Clicking on the "create campaign" button on the map page will direct users to a new page where they will fill out a campaign form

(4) Clicking on the campaign pointer on the map will let a user preview the campaign it belongs to

(5) If the user chooses to view the campaign in detail from the preview, it will bring them to the campaign page. Users will have the option of spreading the campaign, or spreading and donating to the campaign

Firebase

Firebase is a backend mobile platform that stores data on the cloud. It is mainly used for multi-user applications, which is exactly the intended purpose of Float (bringing users together for the shared purpose of spreading charity awareness). Firebase will be used as the database server, holding user and campaign information that can be queried, filtered, and exchanged.

Validation

We will be validating all the subsystems with GUI prototypes in order to ensure that they satisfy the user's needs and stated use cases. To do this, we will find all previously described use cases relevant to a specific subsystem. Running through the GUI prototypes and observing the functioning of the system, we will then determine which use cases are implemented properly and which still need work. The system will then be modified to service requirements that have not yet been met, while also preserving those that were also covered previously. If time permits and we are able to retrieve necessary resources, we will also aim to test subsystems with volunteer customers. To do this, we will provide the GUI prototypes to volunteer testers, who will provide feedback and reviews on a given subsystem. Using the given information we will then be able to improve and polish the application, ensuring it meets all requirements/use cases and is able to provide adequate customer satisfaction.