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. Through our 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

The application consists of the following major system components：

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 team members are familiar with Java, which makes completing the project within the time constraint of two months more feasible.

Google Maps API

The Google Maps API provides the functionality to display a map and draw graphics onto it. Additionally, it provides the means to convert an address into geographical coordinates. We chose Google Maps API as it was essentially the only map API readily available. It is robust, extensive, free and easily integrated into the Android application. Using the Google Maps API offers up the ability to create a highly interactive map page.

Facebook Login API

Facebook supplies a login API, which provides 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 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, but rather global, 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 documentation that allows developers to accept PayPal and credit card payments in their application. Float handles both instant and future payments through PayPal; users can choose to directly donate to a campaign or pledge to make a payment in the future if a campaign succeeds. Instant payments make use of PayPal's mobile SDK and charges users straight from their credit card or PayPal account. Future payments are integrated in two parts, where the user first gives their permission to allow Float bill them through the mobile SDK. When a campaign succeeds, PayPal's REST API is used to create a payment which is billed to the user's PayPal account, executed using a backend server.

Location tracking

User location tracking is needed in the process of creating campaigns and spreading campaigns. Because of the way campaigns are displayed to the users in Float, we need to find an efficient yet easy to change methods for displaying campaign data visually on the map. The GPS system will request and return the user's current location information to the application.

Server

An Amazon EC2 cloud server is used to support future payments in the application. The reason we chose Amazon web services was that it is easy to set up and provide abundant free services. Additionally, advanced services can be added later with moderate cost. The server hosts two PHP scripts required to execute the future payment process. The first PHP script uses an OAuth2 authorization response sent from the mobile device and exchanges this for a refresh token, which is then passed back to Android and stored in the Firebase database. When a campaign succeeds and users who have pledged to the campaign need to be charged, their respective refresh tokens are retrieved from the database, and then used to create the payments through the second PHP script on the server.

Database server

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 discovered that MySQL was not a valid option as Android does not support MySQL. As an alternative, we came upon Parse. Upon further investigation, however, we 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.

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, charity and campaign information that can be queried, filtered, and updated.
Pros and cons about Firebase:
Pros:

1. Firebase is capable of handling the Real-Time data manipulation between devices
2. Firebase is a Cross Platform API which allows easier extension from Android to other platforms
3. JSON format allows easy modification on the data schema
4. Firebase provides a convenient library in Android, which reduces implementation time

Cons:

1. Firebase store data in JSON, which does not have a rigid schema. Messiness could be accumulated in the extension process if it is done carelessly.
2. Migration to SQL is difficult
3. Free version of Firebase is limited to 100 connections and 1Gb of Storage

Detailed Design

Data schema(old version)

Included for comparison purpose and show the decision making process

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 campaigns links a user to the list of campaigns the user initialized or participated in. In the campaign database, the important information is detailed here. For each campaign, the key people are initializers and participators. It also stores the description and float 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.

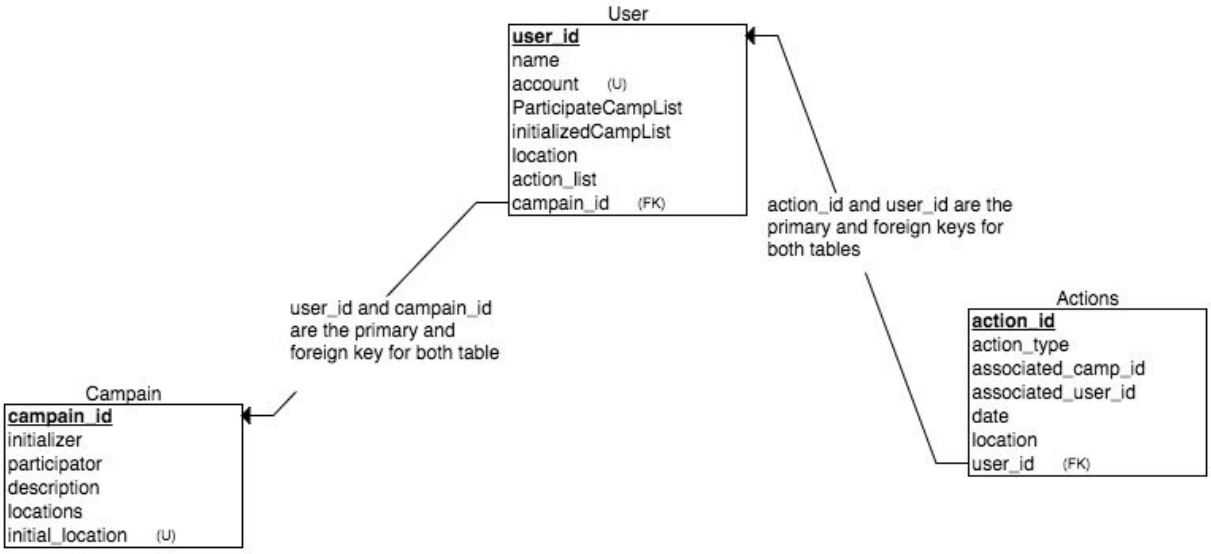


Figure 1. Mysql Data schema

Database design change:

The database schema changed radically from the original design. The main reason for this is that we switched from Mysql to Firebase. Firebase uses the JSON format to store data. Unlike Mysql, Firebase is NoSQL which allows a flexible and dynamic database schema. This mean that each object is stored in database as a file and we don't need to define definite relation between them. During the schema design process, we tried to name the fields of the objects as descriptive as possible for ease of maintenance and so that it will easier for future developers to understand the database.

Data schema(current version)

FloatDB

- Campaigns
  - Campaign_name
  - Accumulated_donation
  - Charity
  - Description
  - Destination
  - Goal_amount
  - Initial_location
  - Dest_location
  - List_locations
  - Owner_account
  - Time_left
  - Status
  - Campaign_pic
- Users
  - Name
  - Account_name
  - Date
  - Blurb
  - Is_charity
  - Amount_gain
  - Amount_raised
  - Amount_donated
  - City
  - Province
  - Country
  - Metadata_id
  - Refresh_token
  - List_of_campaign_followed
  - List_of_campaign_initialize

- Charities
  - Logo
  - Name
  - Description
  - Website

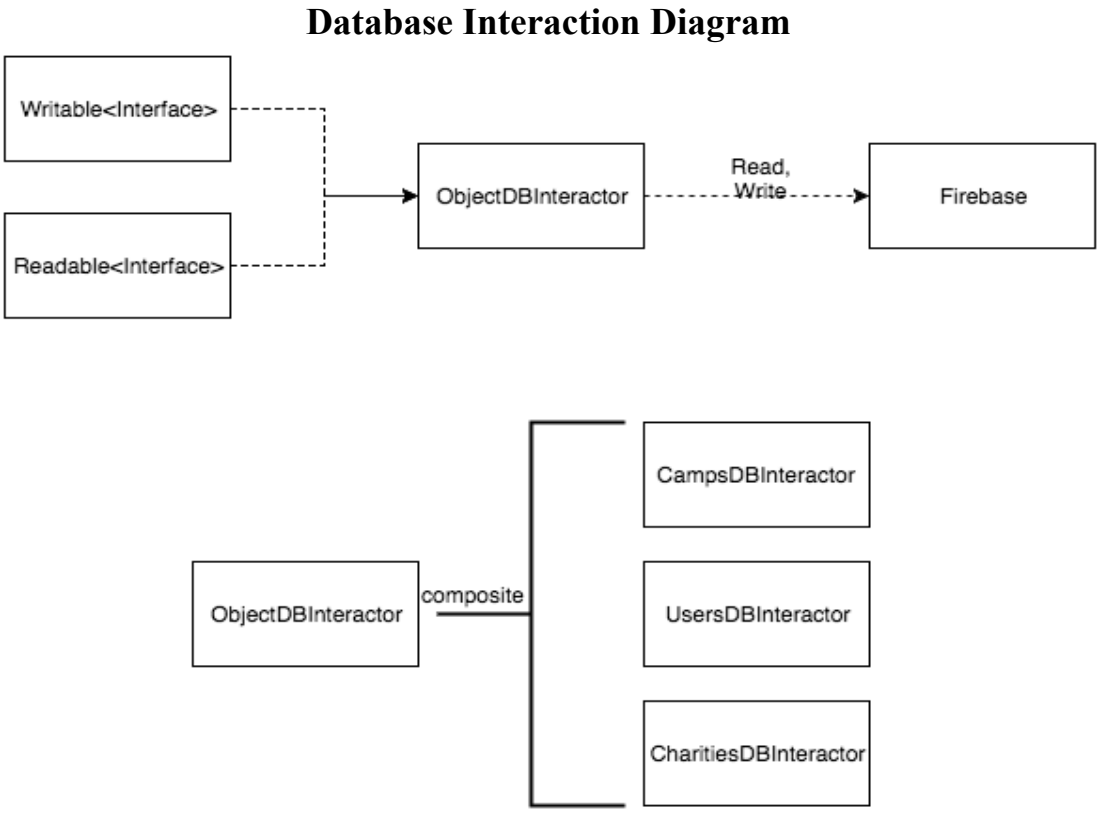


Figure 2. Interaction Layer between FLOAT and Firebase

To ensure loose coupling between subsystems and GRASP , we implemented a communication layer (ObjectDBInteractor) between the database and application. Each object interactor will be responsible for facilitating communication between corresponding object and database. For example, CampsDBInteractor contains methods that Campaigns object use to interact with the database. On top of this, we further abstract out a readable data interface and writable data interface. In the case that we want to extend the object class in database and the application, having abstract interface will make the extension as well as maintenance easier.

Class diagram

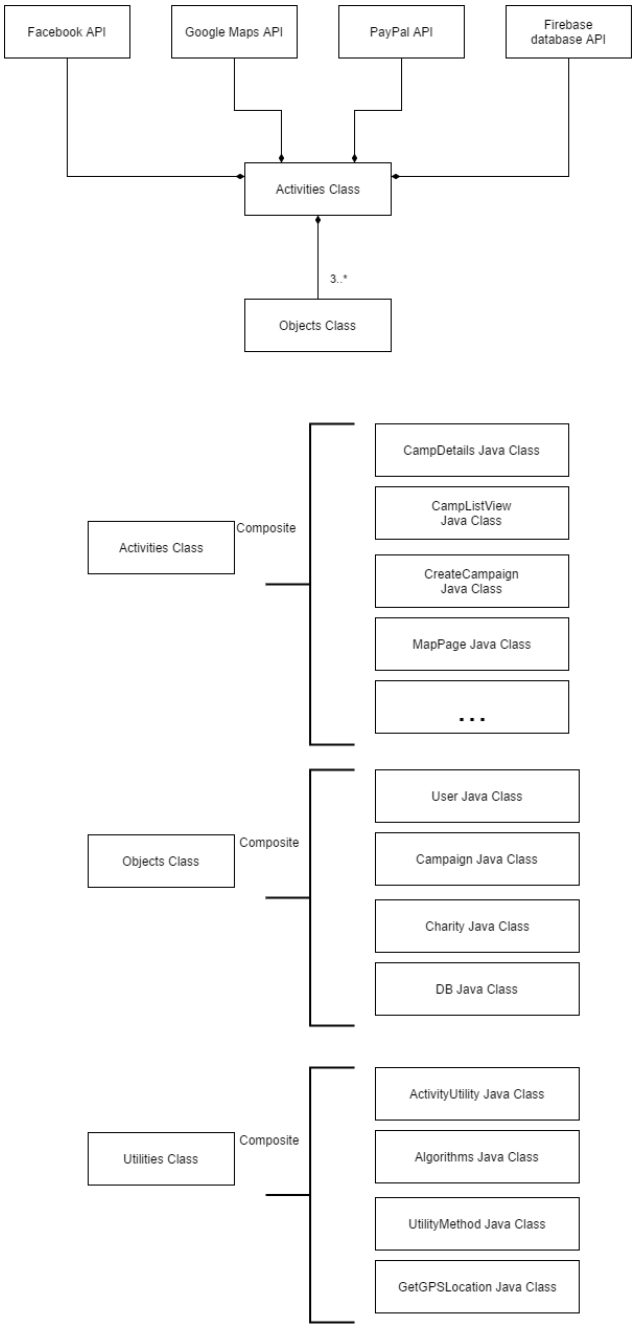


Figure 3. Class diagram for the different subsystems.
(top) The application is consists of the Activities Class, Objects Class, and the Facebook, Google Maps, PayPal, and Firebase APIs.

(middle top) The Activities Class is composed of the Login, Map, JoinCampaign, and CreateCampaign Activity Java Classes.
(middle bottom) The Objects Class is composed of the User, Campaign and Charity Java Classes.
(bottom) The Utilities Class is composed of the ActivityUtility, Algorithms, UtilityMethod and GetGPSLocation Java Classes.

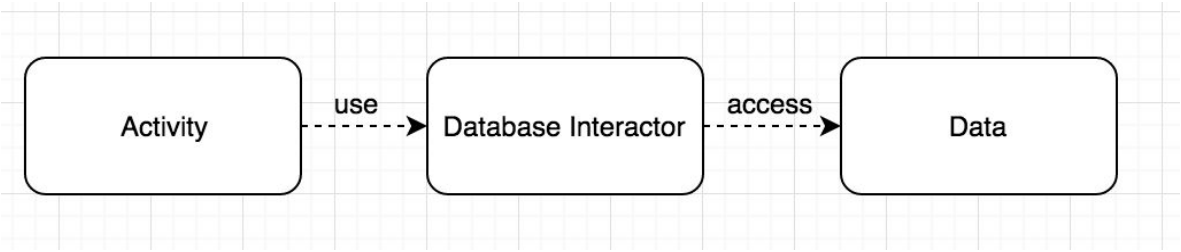
To ensure high cohesion, we created separate User, Campaign and Charity Java classes. We also have the different Activity classes (Login, map, Join/Create Campaign) as Controllers for each application page.

Design pattern

Some design patterns we used in this project are the following:

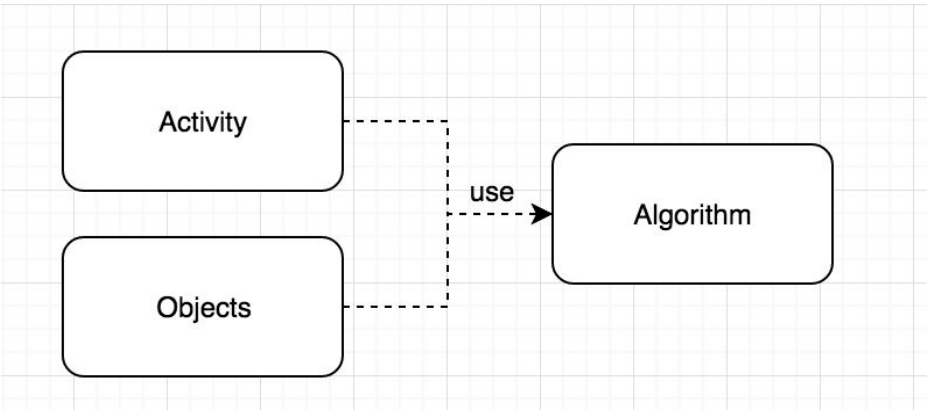
Singleton:

To facilitate interaction between android activities and database, we have classes called database interactor for each object type. The main purpose of database interactor is to enforce the communication protocol between Android and Firebase. Since Firebase is a nonSQL database, enforcing the communication protocol can prevent the data from becoming chaos.



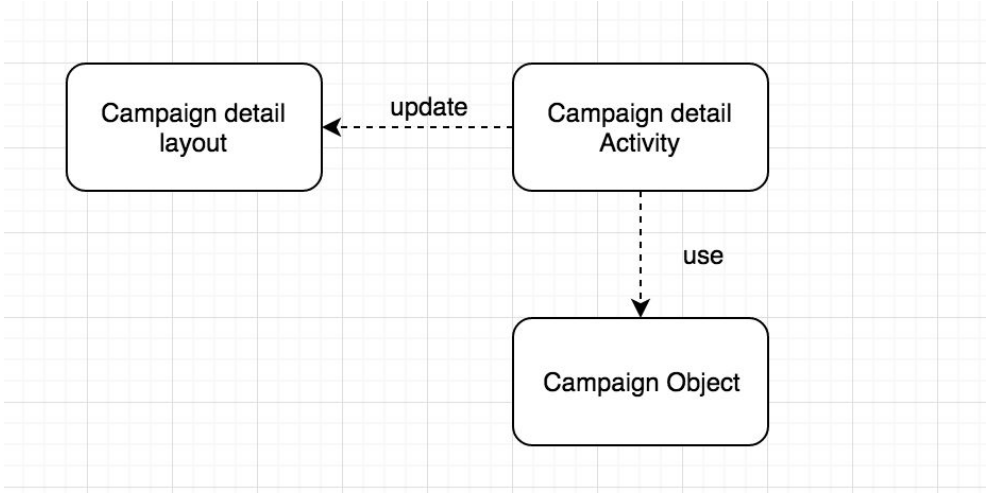
Strategy pattern:

To facilitate computation need for the project, we have an algorithm class to encapsulate the common methods shared among activities and objects.



Model-view-controller:

To facilitate updating on the GUI, we utilize model-view-controller pattern.



Overall System Diagram

Figure 4. Overall system diagram

GUI

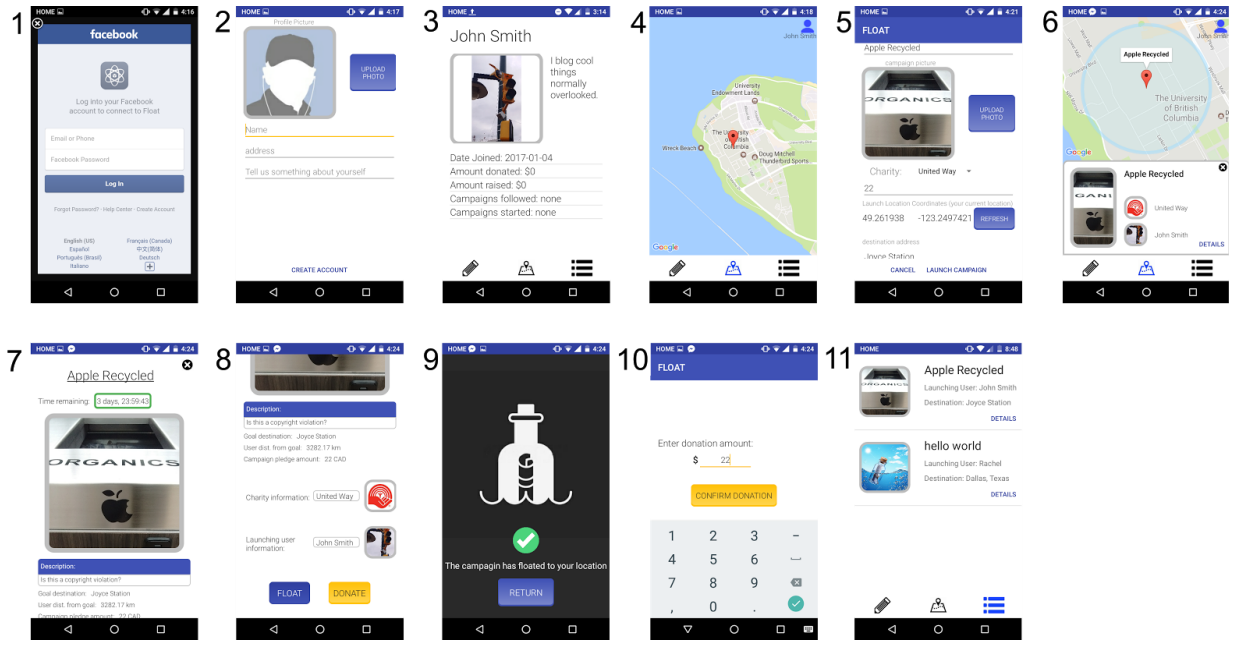


Figure 5. Overall GUI

(1) Facebook Login

(2) Create a user account with profile picture, username, address and a blurb

(3) User profile page

(4) Main map page that shows all the campaigns in a user's area, given by the map pointers

(5) Clicking on the pencil button on the map page's bottom left will direct users to a new page where they will fill out a campaign form

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

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

(8) Bottom half of campaign details

(9) Pop-up acknowledging the campaign was floated

(10) Selecting a donation amount

(11) The user can switch to a list view of the campaigns via the list button on the map page's bottom right

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.