COMP 3004A: Deliverable 3

System Architecture and Design

Project Name: Simply Active
Team Name: Dream Team

Team Members:

Morgan Lay	100981186
Raden Mathieu Almaden	101104851
Jiange Gu	100966298
Rodrigo Sandoval	100898585

Architecture

The intent of this part (1) is to identify, describe, and justify the architecture of your project.
The outcome of theArchitecture documentation is a system architecture that supports the functional goals and non-functional attributes of your project. Non-functional attributes should be specific enough to tell whether or not your app has them. Please make a diagram showing the client-server abstraction

One of our biggest concerns early was that a lot of our API's rely on other components being functional in order to work. We actively came up with solutions around this, namely that we would have an alternative position in case the user decides not to enable permissions for location services. We use Google Maps API, Google Places API, and the device geolocation as components integral to the functionality of our application. We needed to design a system that would enable us to work in parallel, so we divided the app into several components that are independent to each other, and make them accessible through the navigation menu.

The functionality of the project includes user login, user registration, search activities using Google Places, display locations of activities using Google Maps, recommender system using Google Places at home page, user profile contains the user activities summary and user personal information, and settings that users can customize the app. Non-functional attributes include security (the app encrypts users personal information and users need to give permission for the app to use GPS locations), response time (check users location in order to give an accurate list of activities nearby), local storage (saves the map and users information for better response time and for guest users), notifications, reward system, shared activity and sound.

The architecture of our program is set up similar to a main program and subroutine style. We have a main interface, and then have fragments of our pages (home, profile, search page, preferences, and maps page. See figure 1. Page 4.)

Our app Simply Active utilizes a significant amount of data from external components. Due to the nature of our app, we require information about a user's location and leverage that to display results based on the parameters specified in that portion of the application. We display real-world locations within a radius that corresponds with a user generated distance. We have several uses of external components within the app:

Login Page:

When the user initially interacts with the client, they will register a new user, login, or continue as a guest. If they choose to register, a new account will be initialized using their user-generated data. The new account information will be sent to Firebase, a database hosted on the cloud, and then the user will be prompted to login. When the user decides to login, their credentials will be submitted to Firebase to authenticate the login attempt. Upon success, the server will respond with the state of that user and authenticate the login session. They will then be redirected to a homepage customized according to their preferences.

Homepage:

The homepage uses the state from Firebase to determine recommended activities. It communicates those parameters to Google Places API and shows the user relevant activities for them.

Search Page:

The search page uses client generated data to search for activities using Google Places API. It then uses that data to generate a map with location pins using Google Maps API and display it to the client.

Maps Page:

The nearby activities page uses the users location data to search for activities using Google Places API. It then uses that data to generate a map with location pins using Google Maps API and display it to the client.

Profile and Preferences:

Updating settings and saving them sends data to Firebase to update the user's data.

All of our pages use the three-tiered architectural pattern and abstract the data such as in figure 2 (Page 5). When the client makes a request, it handles the logic of that request, and then sends a data request to the appropriate component.

Simply Active also uses an architectural pattern that resembles client-server. A significant amount of processing isn't done on the client, and the client is simply making

API calls to get data. This means that there is a layer of abstraction between the client and the server. Most functionalities in the app use data from the server.

Components:

- Geolocation
- Google Maps API
- Google Places API
- Firebase backend

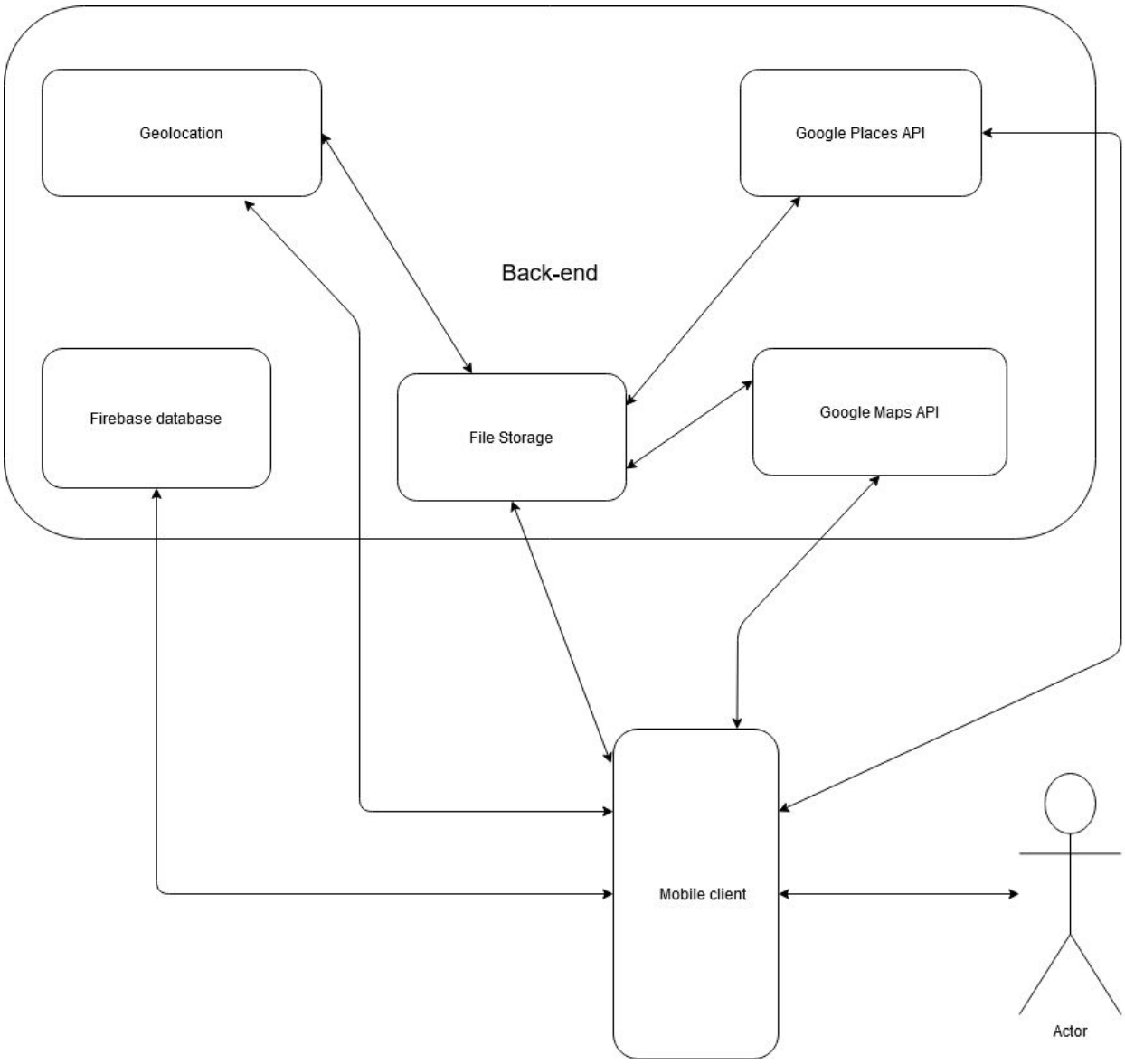


Figure 1

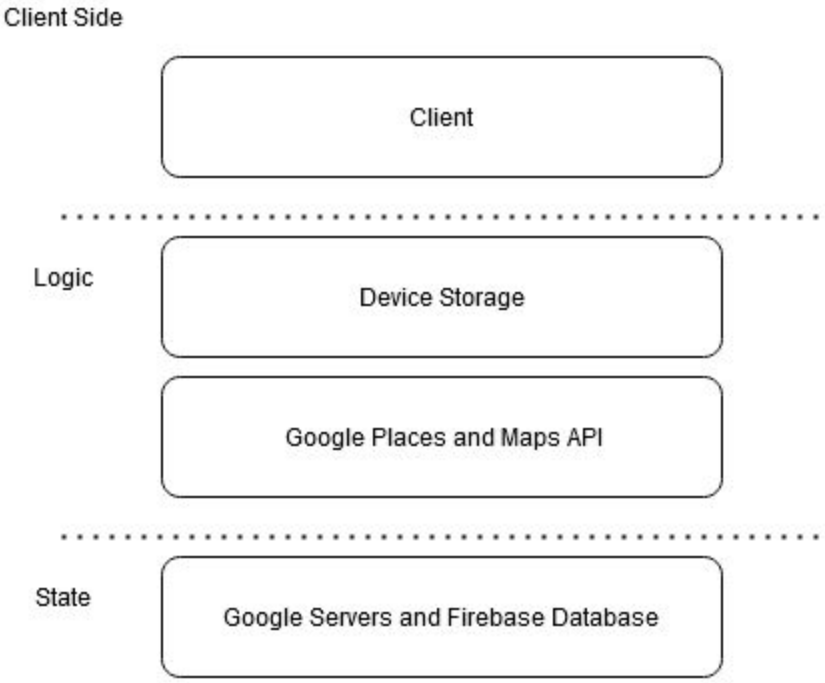

Client Side
Client
Logic
Device Storage
Google Places and Maps API
State
Google Servers and Firebase Database

Figure 2

Design

We prevent coupling by having state determined through the back-end, instead of relying on properly functioning calls between fragments. Our system works with architecture in mind with a subroutine style as the main focus. For our client, we use singular activities with the support of an interface to change over from fragment to fragment. On the first interaction with our system, we use a welcome page activity to lead into a our Login activity which is then supported with a back-end of Firebase to control our user authentication. In doing this, whether or not the user to enter as a guest, we lead into the main activity of the entire system.

To elaborate more on our design of activities, we can look at a simple class diagram in Figure 1 (Page 8). Here we can see that for each existing activity, there is a fragment, as well as a view model to showcase what our system is doing to the end-user. Each fragment will always have 1 activity that it's linked to, as well as a view model. However, the activities aren't limited to a singular fragment/view model which coincides with our idea of making our application into separate parts of a whole being.

In making our client a more fragmented focused design, we are able to keep most of our classes within the client, with the exception of APIs and requesting and receiving information. If you follow Figure 2 (Page 8), when the user successfully logs in or enters the client as a guest, they are then met with a user interface. This user interface is mainly a sidebar navigation menu and upon opening, we are met with a variety of options within the application. In doing, this we can clearly explain which page the user is on as well as move from page to page seamlessly. Each option being a fragment, and being able to switch from one to the other efficiently makes this possible as the activity is always the same, but is just running a different view and fragment.

Instead of relying on each fragment to run functions and always trying to communicate with other fragments, we tried to minimize communications with each other directly to further enhance our system. In doing so, each fragment gets loaded and unloaded when not on the specific page making our space more efficient. To circumvent the direct communication, we determine the states through the back-end, which in this case for us, is the firebase API.

Speaking of APIs, most of our classes are located within the client rather than the server, with the exceptions of classes involving user authentication and user information. More specifically, fragments such as login, search, and maps contain classes that are reliant on the API, which is server-based opposed to on-application. This is due to the relevant information within these fragments, being reliant on the external APIs, to function. We can't necessarily keep the authentication on the client as this would be a security risk, nor can we really showcase maps and locations without the Google API's unless we decided to create our own search engine, and map within the client. It is not feasible and there is no need to do it as the available resources already exist. As we can see in Figure 3 (Page 9), the API's are external to the Simply Active app but only affect certain fragments to make them work opposed to being connected to every single one.

When looking into the future and the potential of our system, we realize that the application is very personal. Meaning that a social aspect to the system doesn't fully exist, and since it's user oriented (to just the single user), there is a lack of interaction because of it, you end up feeling alone. Due to the nature of our app and its system, implementing this change shouldn't be too difficult. Creating a social aspect comes with relative ease due to the use of firebase, and fragmentation. We can easily create a fragment that involves friends, which will mainly run with the back-end of the client (firebase) as the information of friends needs to be authenticated and verified through firebase to prevent personal information being potentially leaked. With our system, it is not difficult to implement mainly because of the limitations we've placed on the communications of each fragment. Each fragment tries to be their own piece in a whole, and because of this, adding more pieces comes with relative ease.

Figure 1: Simplified Class Diagram

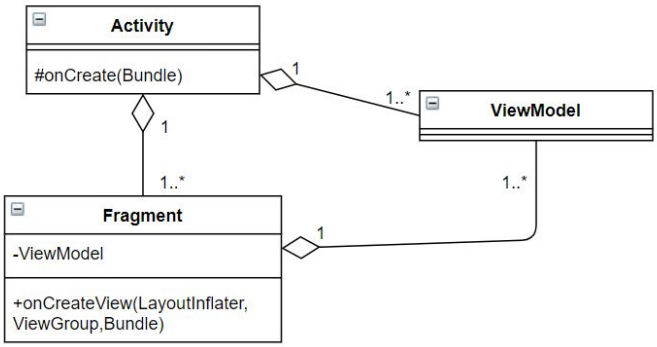


Figure 2: Internal app class diagram

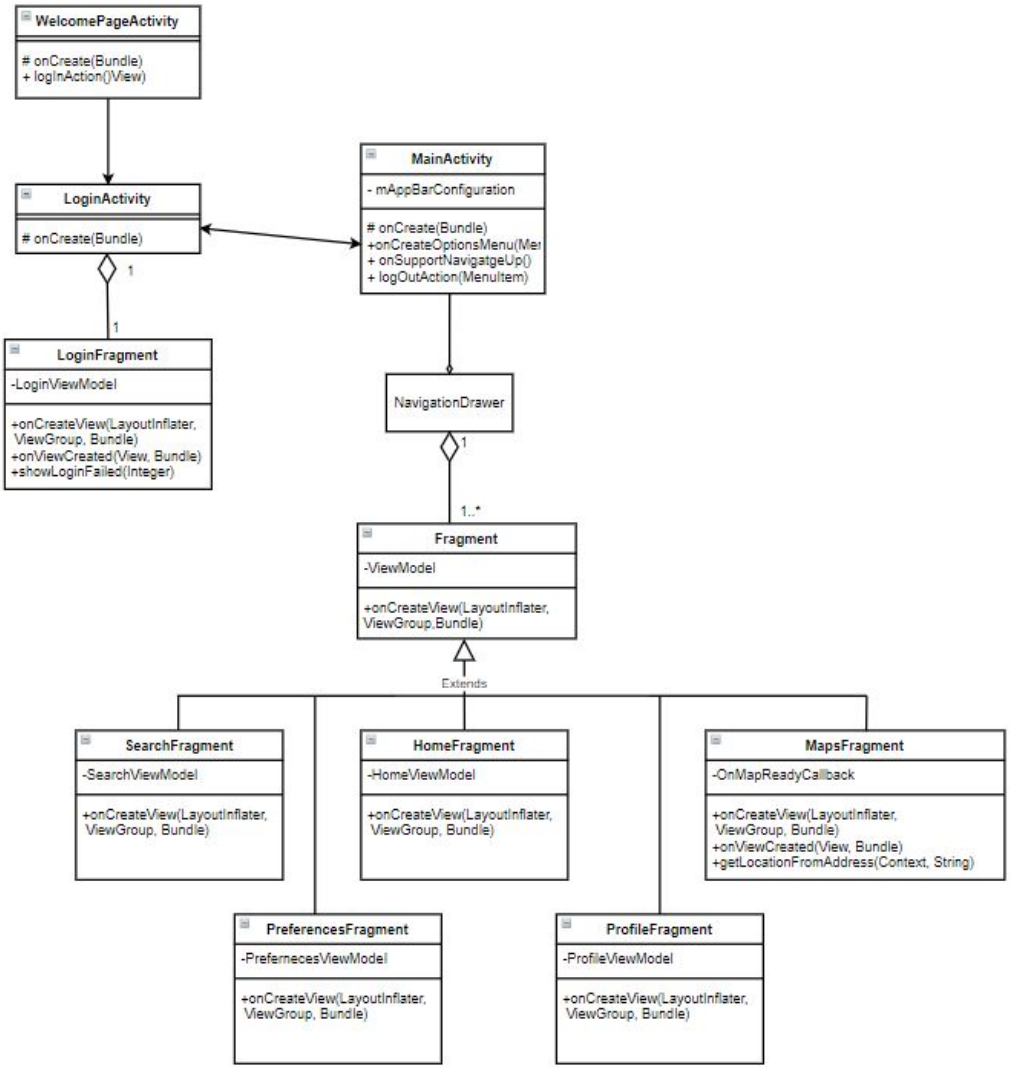
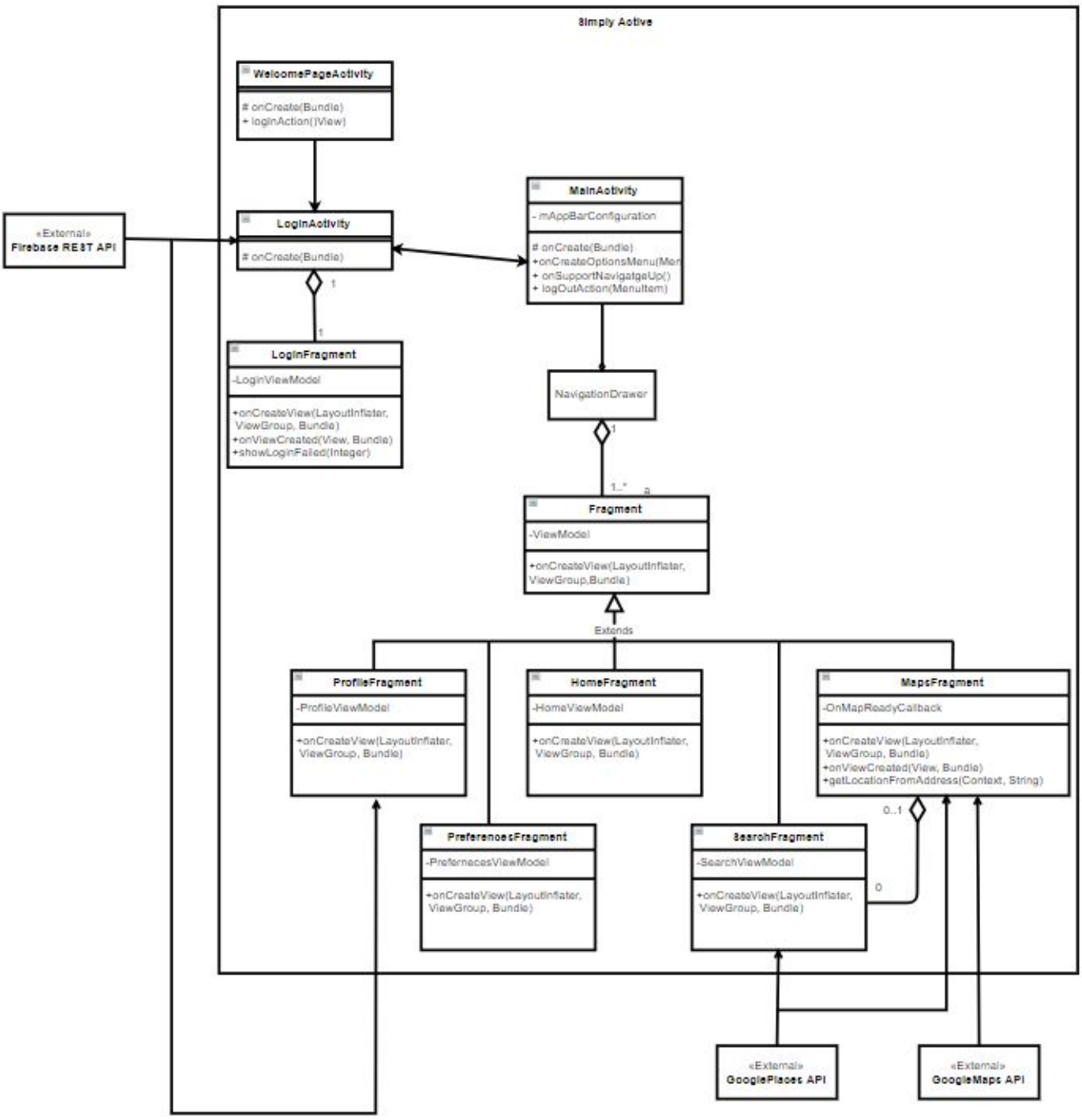


Figure 3: Overall Class Diagram



Assignment of Tasks

We maintain an active list of remaining tasks at:

https://docs.google.com/spreadsheets/d/1g_U95bu49oRkqYt4amel8qv3FPScJar7v2LJnOn9p40/edit?usp=sharing

Morgan: Thus far has implemented Google Maps API, organized API keys, and implemented the fragment for searching for locations. Morgan has worked on UI and UX layouts, formatted the home page, and created the maps fragment. He worked on the functionality of Google Places API returning results from a search. His remaining tasks include using user-generated information to maintain a set of parameters that dictate the likelihood of the user enjoying an activity. This is used for the recommendations on the home page and will need to be stored in firebase and updated through usage of the app. He intends on implementing reminders about events, and showing more information about places once selected on the map.

Raden: Thus far Raden has worked on Places API returning search results, and displaying the pins on the map. Also worked with GeoLocations API to further enhance the search results and include a way of showing the location of said search result. He has worked on getting location services functional, and implemented sliders for search page parameters. He has worked on the navigation, menu, and layout of the app. As well as implemented the general framework of fragments associated with the activities and their use cases. Designed component diagrams, and class diagrams.

Jiange: Designed and created the welcome page, login page and new user registration. He has worked on getting Firebase to create valid users, and is continuing to work on the Firebase backend of the app. He is working on having login sessions, and having the server send the state of the user to the client. He is working on the registration pages for initializing new client data. When a user logged in or registered, their information will be stored and displayed on the negaviation menu and settings page.

Rod: Rod intends on working on UI improvements.