**GroupFuel Android App – Dev Manual**

This manual assumes basic knowledge in Android development.

If you are not familiar with the Android framework, activity lifecycle, views and fragment – take a moment to brush up on those terms before reading further.

It's also advised to go through Parse Android docs, mainly the parts about Parse objects, queries, functions and the DB structure.

Overview:

The design philosophy of the app is to use one activity and many fragments. The activity (called the Main Activity) synchronizes the data the fragments are using and manipulating, and handles all fragment transactions. The fragments themselves each represent a UI component and are completely independent. Each fragment can be dropped in or out at will. The fragments themselves use the Main Activity, through the use of broadcasts on callbacks, to interact with each other. No fragment is aware of the other fragments.

Another important component is the Data Model. Each database table – such as cars, users, etc. is represented by a corresponding Java class. These classes enable the application to use Parse operations natively on regular java classes.

The Main Activity:

This is most important part of the application.

1) **Layout**. The main activity defines the layout of the entire app. At the top, it defines the toolbar with the application name and "three point menu", with settings and logout. It also defines the "fab", floating action button. This is a consistent button the open the "new fueling" fragment, which is accessible everywhere in the application. The last component it the content frame. This is where all the fragments are displayed.
2) **Interfaces**. The main activity implements a number of interfaces. CarListener, StationsListener, LocationListener and FuelingListener. These interfaces are the way the different fragments communicate with the main activity. Each fragment doesn't know the Main activity is calling them. They could be called by any activity. If a fragment requires something, it uses a private callback interface. Every activity that implements these interfaces can use the fragments.
3) **Synchronizers**. Most of the communication with the Parse databases is done by the Main Activity. It queries the Parse DB for cities, gas stations, cars, etc. there are saved in the Main activity, and called by the fragments, using the interfaces, when required.
4) **Broadcasts**. When the Main activity communicates with fragments, it uses the broadcast system. It doesn't check which fragments are currently active or need information. It broadcasts the information, and every fragment that is currently active intercepts the message and reacts accordingly. The Main activity broadcasts any updates to location, cars, etc.
5) **Fragment Transactions**. The main activity calls all fragments, and decides which fragment is called when. Except for one exception, no fragment calls other fragments. They all rely on the main activity for that.

ViewPager Container Fragment:

Most of the time, the fragment in the content frame will be the viewpager container. This fragment has two components. The top one is a tab ribbon. There are 3 tabs displayed here, each corresponding with a different activity. Users can either swipe between tabs or tap the tab titles. This is the one exception to the rules that fragments are independent – as a fragment container fragments, it requires the specific 3 fragments it displays and instantiate.

1) **Usage Fragments**. This fragment displays the different cars users either own or drive. Each car has basic information displayed – current mileage, fuel consumption, owner and drivers. This fragment also allows users to interact with their cars – preforming tasks such as adding a new car, removing an existing one and managing drivers.
2) **Fuel Log Fragments**. This fragment displays all the fueling events for the current user. Each fueling event includes the amount of fuel, the price, and gas station.
3) **Navigation Fragment.** This fragment allows users to navigate to different gas stations around them. Stations are displayed in order of distance from the users' current location (if available). Each station also includes up to date price information from other users, so a driver can always make an informed decision.

**Custom Views**. Each one of these fragments is, at the core, a list of Data Model objects. The Usage fragment is a list of cars, the Fuel Log fragment is a list of fueling events and the navigation fragment is a list of gas station. Each of those Data Model items has a specific view that displays all the relevant information. This custom view is completely re-usable, and only requires the corresponding data model item to be displayed.

New Fueling Fragment:

This is the fragment that is called when a user clicks the fab button. This represents the main use case of the application – the user is fueling his car, opens the app while he waits, enters the data, saves it, and goes on with his life. This fragment gets the car list, cities and stations form the main activity. If location data is available, then gas stations are selectable by location. Otherwise, a list of cities is provided, and the user can choose one of those. The fueling event is immediately available in the fueling log, and the navigation fragment is updated with price data.

Personal Fragment:

This fragment is used to update personal information. This fragment is opened in two different occasions – when a new user is registered, he is prompted to enter his personal information. The other is when a user specifically chooses to update his personal information from the settings menu.

Register and Login:

This part of the application is the only one that deviates from the one activity, many fragments design. There are 3 activities in this part. One is the dispatch activity, its only purpose is to be a landing point for the application. It checks if there is currently a logged in user. If there is one, the Main activity is launched. Otherwise, the user is given the choice between logging in or registering a new account. No matter what he chooses, he will be directed to the Main activity when done.

Data Model:

Each one of these classes behaves like a normal Java class. The interface is the typical getter and setter set of functions. The major difference is that every objects corresponds to a parse object, and can be retrieved or saved to the cloud using native Parse commands such as save. These are also the objects that are returned by cloud functions and sent to them. For more information about each class, see the corresponding entry in the Cloud Developer Guide.