Fuel Around

Executive Summary

Purpose:

Fuel Around is an Android Application that analyzes GPS data and a related Gas Pricing API to find which station would be most advantageous to fill up at by considering issues such as:

- Distance of the station from the user.
    - ex. Where's the closest station? Does it have the least expensive price?
- Gas mileage of the car.
    - ex. What is the cost to drive to an individual station when considering a single vehicle's MPG?
- Potential savings when comparing Distance vs. Price.
    - ex. Would it be less expensive to drive to a further location to save more money at the pump?

Fuel Around is an improvement on all related Android Applications because it considers pricing differences and distance to combine both lowest distance and lowest price. Other applications only consider one or the other and fails to do the calculations for the user. This shows the most potential for usage in extreme cases where large amounts of fuel are being purchased and the vehicle is significantly lower milage. The prime targets for this application would be Recreational Vehicle users and semi truck drivers.

Fuel Around also differentiates itself from other apps on the market because it takes into account that the user is going to get gas on their way to a specific location. We will incorporate Google Places API, so that Fuel Around has the most up-to-date in location information available.

Design:

Fuel Around follows the Model-View-Adapter architecture that clearly separates the applications logic from the user interface. MVA is a variation of the Model-View-Controller pattern. The difference from MVC is that the view and the model layers cannot directly communicate. The adapter is the only interface between the two other layers.

In this project, views are represented by XML files. Fragments and activities are part of the adapter layer. The adapters retrieve data from the model, and use it to update the view.

Adherence to the *Skinny Controller / Fat Model* and *DRY* best practices have helped to avoid anti-patterns such as *Copy & Paste Programming, Big Ball of Mud,* and *God Class*. By delegating business logic and data manipulation to the model, the adapter becomes streamlined in form and function. The fragment and activity classes, adapters, only exist to update the view with data from the model.

- Fuel Around follows the Model-View-Adapter (MVA) architecture.
    - Extensibility:
        - The object orientated design of MVA allows future expansion of Fuel Around to be implemented quickly and easily. It eases issues related to integration and reusability of fragments and helpers.
    - Maintainability
        - Compartmentalized programming allows for quick identification while bug tracking. Issues of changing API functionality, language version control, or advances in mobile hardware are handled more efficiently when core activities can have usability restored through minor changes to helper functions.
- The class diagram below shows clear organization and relationships between activity adapters, fragments, helpers, models and views.
    - All helpers have low level of complexity, this drastically reduces maintenance costs.
    - All helpers have a high potential for reusability which allows for quick expansion when implementing adapters with high cohesion.

Verification:

- Chidamber-Kemerer metrics, including LCOM (lack of cohesion methods), calculated by the MetricsReloaded plugin for Android Studio, are below the default maximum thresholds of the plugin for every class. This indicates that the class design well encapsulates and separates concerns.
- The average essential cyclomatic complexity of our methods is 1.31, calculated by MetricsReloaded. This indicates that methods aren't doing so much that they have hard to test, unpredictable behavior.
- Testing Fragment and Activity classes (Adapters, in our design) in Android can't be done in the same way as other classes, since they handle events from the Android system. However, by keeping business logic in the model, we can write tests for that code, and reduce the chance of something going wrong in our Adapters.

Conclusion:

The aim of utility applications has been to expose customers to ad revenue or entice them to purchase a service that exemplifies itself in value and practicality. Fuel Around excels

in providing a service that grows in value the more the user accesses it. These benefits grow exponentially for those fueling vehicles with larger gas tanks and lower gas mileage. In comparison to other fuel related utility applications, Fuel Around recognizes the average customer's on-the-go lifestyle and makes recommendations that maximizes their valuable time. Fuel Around maximizes savings based on an individual user's needs which sets it leaps and bounds ahead of rival applications in its field. Fuel Around exemplifies a low risk investment because of it's MVA application design. Applications with the MVA style significantly reduce the costs related to **maintainability** and **extensibility** which minimizes the TCO and vastly improves the cost of labor and the rate of return on further investments to this already valuable service.

Class Diagram:

Our class diagram, as shown below, has been simplified to illustrate the flow of data through the program. MyGasFeedAPI and CarQueryAPI, in the model layer, are using a small custom library to fetch JSON objects from the API's. The classes in the controller layer use information from the model to update the view. Helper classes remove logic from the controller that formats data for the view, keeping the controllers skinny and the code DRY.