iOS Developer Exercise

Prepared for: Volt Line Company

Prepared by: Alaa Al-Zaibak

1 June 2018

Used Design Patterns

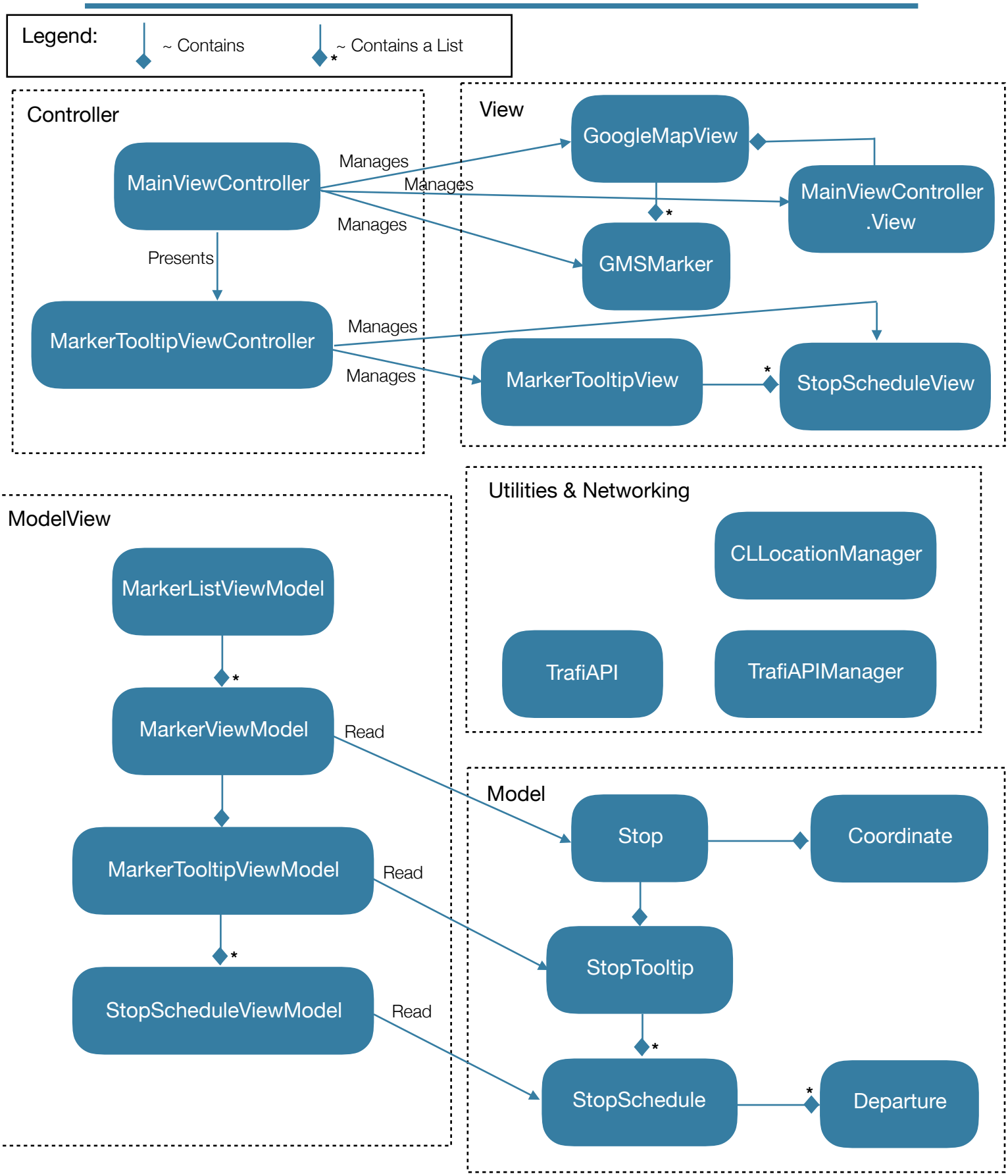
1. **MVVM**: The project designed with MVVM design pattern, *although there where some extra unneeded work to implement it instead of MVC for such a small project*, this extra work will be justified if the project was going to be larger in the future, and if the unit testing was going to be added to it.
2. **Singleton** design pattern - TrafiAPIManager & Utilities classes were created as singletons to restrict the instantiation of each class of them to one object, since we need a global access for it and only one instance of the object is required.
3. **Adapter** design pattern - TrafiAPIManager was used as an adapter for MoyaProvider to handle the API calls and return our defined models after creating them from the returned JSON from MoyaProvider.

Used Technologies & Libraries

1. xCode V9.3.1, Swift V4.1 (minimum supported platform is iOS 9.0)
2. Git (using Sourcetree app by Atlassian), to handle code versioning and pushing to the repository to Github.
3. Trafi API, used to get the data need to be presented on the map, like nearby stops and departures.
4. Cocoapods, used as dependency manager to handle project dependencies.
5. GoogleMaps, used to represent the map view and the markers (that represents the nearby stops) on top of it.
6. SnapKit V~4.0.0, a library used to handle and create auto layout for the created views in the app.
7. Moya V~11.0, a library used to handle trafi API requests, and getting nearby stops and a stop departures.
8. PullUpController, a library used to show the detailed info of a stop and its departures, a grip view was created and UIBlurEffect was added to the pull up controller view to enhance the look of the view represented using this controller.

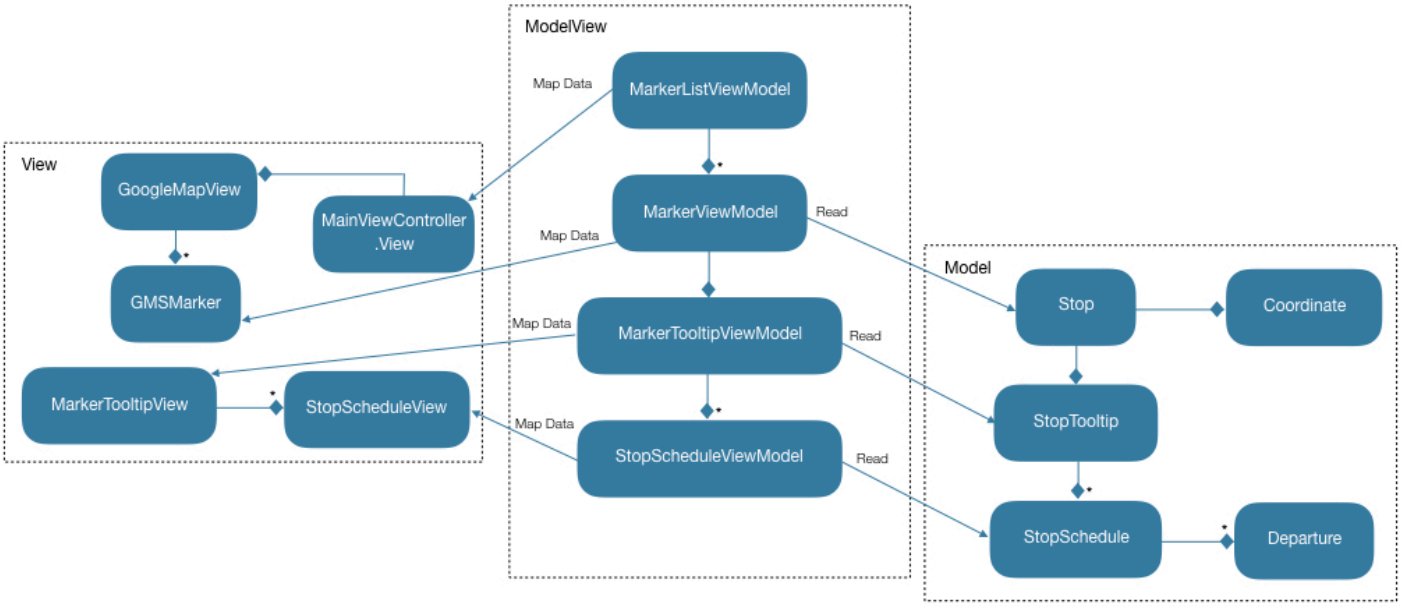
Architecture & Design

The next diagram shows the main architecture and global relations between the classes of the project:


Legend:
~ Contains
~ Contains a List
*
Controller
MainViewController
Manages
Manages
Manages
Presents
MarkerTooltipViewController
Manages
Manages
View
GoogleMapView
MainViewController
.View
GMSMarker
MarkerTooltipView
StopScheduleView
Utilities & Networking
CLLocationManager
TrafiAPI
TrafiAPIManager
ModelView
MarkerListViewModel
MarkerViewModel
Read
MarkerTooltipViewModel
Read
StopScheduleViewModel
Read
Model
Stop
Coordinate
StopTooltip
StopSchedule
Departure

MVVM:

The project is designed to follow MVVM design pattern, the next diagram shows the “view-view model-model” relations:



The previous diagram shows two deferent relations:

1. **Read**, Triggered in two situations:
   - When the user location retrieved (or changed), the MainViewController requests the stops nearby from the TrafiAPIManager and gets the stops, then it creates the MarkerListViewModel and attach the retrieved stop list with it.
     The created MarkerListViewModel is responsible of creating its list of MarkerViewModel and all the underlying hierarchy (with some missing values like the departure times and destinations).
   - When the user taps on marker at the map, the MainViewController creates and shows a MarkerTooltipViewController that creates & shows MarkerTooltipView, and requests the departures of taped stop from TrafiAPIManager to get the departure schedules, then it updates the MarkerTooltipViewModel with the retrieved data.
     The MarkerTooltipViewModel is responsible after that for modifying its list of StopScheduleViewModel (the departure times and destinations).

2. **Map Data**, This relationship is handled by view controllers as follows:
    - MainViewController creates number of markers equals to the MarkerModelView count and set their properties.
    - MarkerTooltipViewController creates the MarkerTooltipView and set its model view after creation, and it asks the MarkerTooltipView to update schedules when the data is retrieved from TrafiAPIManager and the model is updated.

NETWORKING:

The class “TrafiAPIMAnager” created a singleton that handles api requests and returns or update the models in need.

Moya library was used to handle networking abstraction, and defining the used API routes (that are defined using the enum TrafiAPI), while all the requests was handled by TrafiAPIMAnager as an adapter for the MoyaProvider class.