GroupFuel WebApp – Dev Manual

This manual assumes basic knowledge in JS and in angular.js.

Knowledge in backbone.js may help as well.

If you do not know what is NPM, Angular Service/Controller or $scope, we advise you to take a short break from this manual, open a basic JS tutorial and Angular's official documentation and come back when those terms will sound familiar to you.

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

The modules:

The application is built from several modules:

- Partials – static html files, the application's skeleton.
- Controllers – control and change the view (based on the partials).
- Services – encapsulate the backend request. Controllers will use services to get/post data to the backend.
- Filters, Factories, Routers – Angular minor modules for that handle side aspects of a web app.

The app is built as 'One Page App', means that there is one main page, and when user move between different sections only the content of the page is changed (Changing partials and controllers) using the Angular Router.

The main page is controlled by the NavigationController, which is responsible to invoke the login/logout actions. It's reachable from every part of the app, but only refer to the navigation bar.

The app modules are made of pairs of controllers and partials. Services, factories and filters are shared by different modules and will be described later.

- Login Module:
  The login module is based on LoginController and the login.html partial.
  It is launched as a modal by the navigation bar (by its controller).
  It's main function is doLogin() which sends the login info to the backend using the UserService.
- Signup Module:
  The signup module is based on SignupController and the signup.html partial.
  The url '/signup' leads to it.
  It has 2 main parts – userDetails object – most of the view bindings are on it, and the doSignup() function, which take this data, verify it and sends as a signup request to the backend.
- User Management Module:
  Contains 3 pairs of controllers and partials – edit profile, login, signup.
  The edit profile part provides one function – doUpdate() – it's arguments are taken from a scoped object, 'userDetailes' who's binded to the view. It uses the UserService to send an update request to the Parse backend. It also uses the service to retrieve user information.

The login part is run as a modal (invoked from the navigation bar usually) and provides to functions – doLogin() – sends a login request to the backend, and doCancel() which closes the modal.
The signup part provides one main function – doSignup(). Similar to the 'edit profile' part, it has a main object – userDetails which is bound to the view and holds the relevant data for a new user.

- Cars Managing Module
  Cars Managing Module is based on 3 controller and partials: MangeCarsController & managecars partial, AddCarController & addcar partial and eventually ManageCarsDriversController & managecarsdrivers partial.
  The first has two significant function which is called 'getOwnedCars' and 'updateDrivingCars' that updates the data to be presented in the tables.
  The second one is a module for adding a car, which is similar in the selection as can be seen In StatisticsController and in charge of adding a car for a user after filling every relevant field.
  Lastly, the 'manageCarDrivers' module is in charge of viewing, adding and removing drivers for a selected car and its functions are, as expected, creating a table for viewing and adding\ removing drivers.

- Personal Usage Module:
  The Personal Usage module is based on the UsageController and the Usage partial.
  This module is divided into 3 main parts: The fuel log that is created by getFuelEvets(), cars raw data which is created by getUsage() and the graphs(pie charts, line charts and Google Maps) which is created by updateUsage() and the Services it is using. It is worth mentioning that this module is using additional modules in order to create the graphs such as tc-charts and ngMap.

- Statistics Module:
  This module is based on the StatisticsController and the Statistics partial.
  It holds the selection data of the user in 'selected cars' object, while filtering the irrelevant object after each selection that the user makes. Then, the 'addSelection()' function is the main one of the controller, that is presenting the data in the Statistics table.

<u>The environment:</u>

GroupFuel web is an Angular.js application which use Parse.com as a backend and therefore has Backbone.js dependencies. In this section we'll describe how to handle this situation and the issues cause by it. We'll also give a brief review of our suggested environment settings.

- Requests to parse are done using Backbone promises – Angular does not track them and therefore does not react to them. To prevent the need to manually invoke the digest cycle each time (and handle different promises that resolve at the unknown times) we wrap each Backbone promise with an Angular promise, using the $q service as can be seen in any of the Angular Services.

- Requests to Parse retrieve Parse objects – Angular bindings are done using objects properties, while with Parse objects those properties are not reachable, only by getters. To solve this issue, we use factories to extend the Parse classes and add the functionality needed by Angular.
- To allow future change of the backend (and more correct design in general) we separate the backend calls from the business logic. Backend calls are done from Services (and this way enable different controllers to share them) while business logic is implemented in Controllers.
- Routing – Angular router is in charge of url parsing and changing the content of the page accordingly. This cause a serious problem – when deployed in Parse, website is only reachable through its main page (http://groupfuel.parseapp.com). Urls like http://groupfuel.parseapp.com/manage-cars are not accessible. The cause to this problem is the Parse will only accepts Backbone router. The 2 solutions available are to move to Backbone router (and to find a way to integrate it with Angular) or to dump Parse.

Adding a new Module:

As any application, development doesn't end when it's published. We'll describe the process of adding a new module (after designing it and preparing the backend for it), so it will be done in a way that does not create code duplications or break the Angular way of thinking.

- After deciding the functions that the new module will provide, separate each of the to 2 parts – the one that updates the view (or updated by the view) and the part that include the backend connection.
  The first part will be implemented in the controller, while the second will be part of a service.
- In case the service part already exist in an existing service, just inject it. Otherwise, other service functions might be generalized to support it as well. If it's a general action – it might belong in one of the general services (UserService or ParameterService), if not – add a new service for the module.
- Try not to add heavy calculations or big amounts of data to the controllers. Share constant data through services – they are singeltons that built once. Controllers are built everytime the routing changes.
- After implementing the backend requests (service) and data holders (controller), test the controller by mocking function calls and data binding.
- Only after testing, add the html partial and it's binding.
- Check if your new controller uses any private functions that are similar to private functions in different controllers – if it does, move them to a relevant service.