Developing Custom Scalable Vanilla JavaScript App

–Kiran A. N. April 2019.

Audience aimed is *intermediate*, requires understanding of JavaScript's module pattern, IIFE, 'prototype'. However blog also aims at non-JavaScript developers to get going at modular way of writing JavaScript code, so it could serve the immediate purpose of writing scalable app. Architecture can be decided and thought of according to their necessity.

Any app which requires good front-end data handling, UI handling requires a nice solid library and/or framework to work with. It is driven now a days by Angular, React, and Vue etc. How about we write our own nice reusable, scalable modular code that can be part of custom library or framework in future.

General Requirements of any front-end application.

Constants, UI Module, Events, Data handler, Logger, Ajax, Models etc

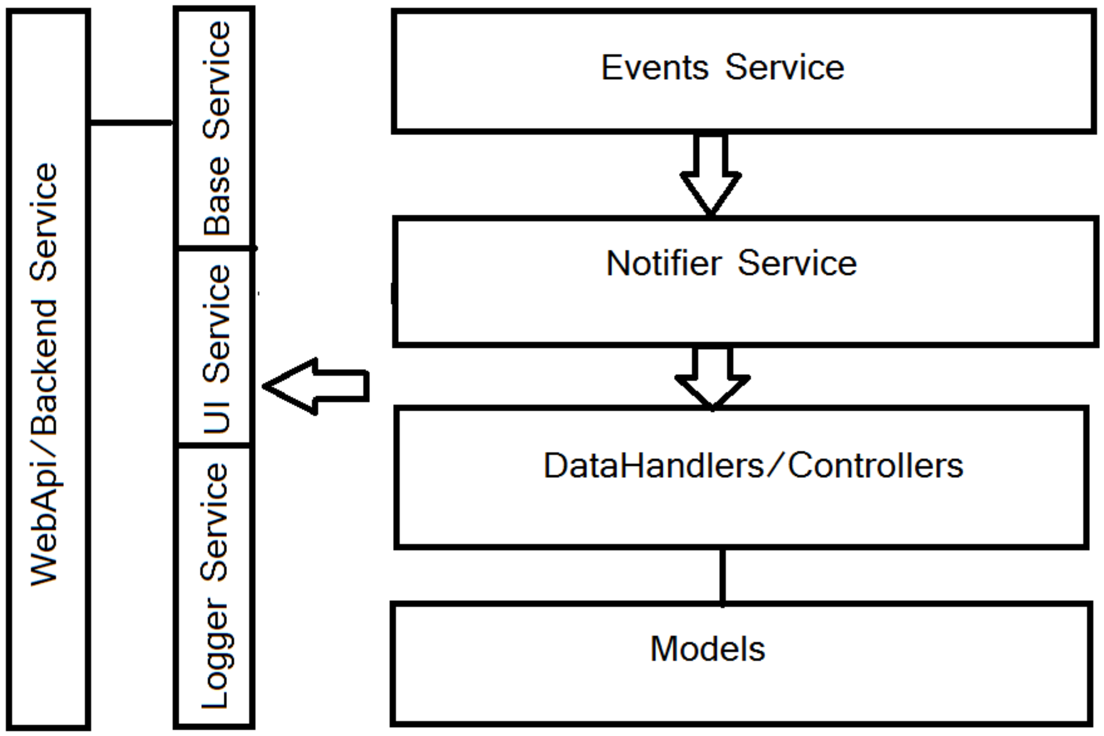

Fig: General/Template Design of data flow for our app

Exploring Models, Modules, and Services.

Event Handlers. Handling event of DOM elements for on click, on hover, onMouseMove, etc. Let the DOM load be faster and we register click events from Javascript after the domLoaded event occurs. (Not shown here assumed it is known).
Events Service takes notifier service as parameter. We are registering the notifiers to be called upon event bindings. The dependent notifier service will be explained shortly.

```
Register All Events of HTML content from Javascript

"use strict";

// bind it to appropriate click/hover/ any other methods
// Call Notifier methods
App.Modules.EventsService = (function EventsModule(notifier){

     var service = {};

     service.registerEvents = function(){
         document.getElementById('btn').addEventListener('click',notifier.getData());
     }
     return service;

})(App.Modules.NotifierService);
```

eventsHandler.js

Base Service is completely independent module which can be reused in other applications as well. Requires jQuery ajax, if not available then xml http can be used.

Ajax is usually used to fetch data from a web service. It stands for Asynchronous Javascript and XML. However nowadays Json is the most common data exchange format.

```
/// Base Service
"use strict";
App.Modules.BaseService = (function DataFetcher(){
     var service = {};
     var url = hostname + appConstants.BASE_URL ||"";

     service.setUrl = function(apiCall){
         url = url + apiCall;
     }


     var errorFn = function(error){
         console.log(error);
     }
     service.getData = function(method, contentType, success, error){

         // jQuery Ajax
         $.ajax({
             method:method,
             url:url,
             contentType:contentType,
             success:success,
             error:error || errorFn
         });
     }

     service.getBinary = function(method, url){
          // xml http
         var xmlHttp = new xmlHttp();

         xmlHttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
         }
     };

     xmlHttp.open(method, url, true);
         xmlHttp.send();
     }

     return service;

 // or any async calls
})();
```

baseService.js

Data Handler is another service which uses app driven custom models and can act accordingly. Uses UIModule to display data. Here in the example the bind model method is used to create mode and call ui-modules for display of data. Any other data formatting can be done in this module.

```
/* Data Handler */
"use strict";
App.Modules.DataHandlerService = (function DataHandler(models, uiModule){
   // bind to models
     var service = {};

     service.bindApiModel = function(data){
        // Do whatever
        var apiModel = models.getApiModel(data);
        apiModel.truncData();
        uiModule.show(apiModel);
     }
     return service;

})(App.Modules.Models, App.Modules.UIService);
```

dataHandler.js

Integrate any UI library

This module gives the flexibility to change the UI library at run time. It will help the application to integrate between UI frameworks without much hassles.

```
/* Change the UILib at run time, uses multi library with ease of handling */
"use strict";
App.Modules.UIService = (function UIModule(uiLibs){

    var service = function(){
    }

    service.show = function(){
        jquery.bind(); //uiLibs.bind();
    }

    service.open = function(){
        bootstrap.popup(); //uiLibs.popup();
    }

    service.animate = function(){
        jquery.animate();
    }

    var init = function(){
        if(uiLibs.length > 1){
           jquery = uiLibs[0];
           bootstrap = uiLibs[1];
        }
    }

    init();

   return service;
  //etc
})([jqueryUI,bootstrap] /* jquery*/);
```

uiModule.js

Custom **Models** of Application.
Control Object creation extension/modification. All App models can be part of this module. ApiModel example given here is private to `*Models*` Module. Hence overwriting the models is minimized.

```
"use strict";
App.Modules.Models = (function(){
   var service = {};

     // Now private to Models
      function ApiModel(data){
         var prop1= {}, prop2 = {};
         prop1 = data.prop1;
         prop2 = data.prop2;
      }
      ApiModel.prototype.truncData = function(){
         return this.prop1.trim();
      }

     /*
      ApiModel.prototype.method2 = function(){
          /// TODO:
      }
     etc
      */

      service.getApiModel = function(data){
          return new ApiModel(data);
      }

   return service;
})();
```

appModels.js

Application Constants

Generic application constants goes part of this function as one point lookup.

```
"use strict";
var appConstants = (function constants(){
      var constants = {
           GET:"GET",
           POST:"POST",
           JSON_TYPE:"application/json",
           BINARY_TYPE:"application/octet-stream",
           BASE_URL:'/api/'
      };
      return constants;
})();

Object.freeze(appConstants); // No modification can be applied to appConstants
```

appConstants.js

App Registry

Final app.js where we register our modules as part of App Namespace.

```
/// AppRegistry
var App = {
    Modules:{}
};
```

app.js

Thank you Have a Nice Day!