**BlackBox Framework**

This is an educational small JS framework. Its only purpouse is to make education of SoftUni students easier.

**Overview**

The BlackBox Framework is written in ECMAScript 2016 and basically wraps up several core functionalities, needed for the implementation of simple Single-Page Applications (SPAs). For the Framework to be fully functional, it needs to have access to several other JavaScript libraries. These libraries include - JQuery, JQuery noty, SammyJS. The logic that is normally used by the combination of this libraries in order to implement a simple SPA is what BlackBox makes easy and user-friendly.

The framework consists mainly of 4 elements:

* **Requester**:
  + A simple AJAX requester used to wrap the request-handling code into one class
  + Defines logic for **GET**, **POST**, **PUT**, **DELETE** requests
  + Builds it's own request headers, based on a, passed as an argument, Authorization Service
* **Authorization Service**:
  + This is a class containing functionality used mainly for authentication & authorization
  + Defines the logic needed for creating request headers and credentials
  + Builds the headers accoarding to given app credentials
* **Notification Service**
  + Operates in the form of several functions
  + Is not wrapped in a class, due ot its need of being used globally in the whole application
  + Defines logic for **success**, **error**, **info**, **warning** popup notifications
* **Event Service**
  + Operates in the form of several functions
  + Is not wrapped in a class, due ot its need of being used globally in the whole application
  + Defines logic for routing, triggering and handling events, and redirecting URLs

**Documentation**

* **Requester Class**:
  + *constructor* function
    - Initializes the class
    - Parameters: (authorizationService)
      * **authorizationService** - A parameter of type AuthorizationService, used to generate request headers
  + *get* function
    - Defines the logic used to initiate a **GET** request
    - Parameters: (url, successCallBack, errorCallBack)
      * **url** - A parameter of type string, defines the request url
      * **data** - A parameter of type object, holding the data in the request body
      * **successCallBack** - A parameter of type function, defines the function called on request success
      * **errorCallBack** - A parameter of type function, defines the function called on request failure
  + *post* function
    - Defines the logic used to initiate a **POST** request
    - Parameters: (url, data, successCallBack, errorCallBack)
      * **url** - A parameter of type string, defines the request url
      * **data** - A parameter of type object, holding the data in the request body
      * **successCallBack** - A parameter of type function, defines the function called on request success
      * **errorCallBack** - A parameter of type function, defines the function called on request failure
  + *put* function
    - Defines the logic used to initiate a **PUT** request
    - Parameters: (url, data, successCallBack, errorCallBack)
      * **url** - A parameter of type string, defines the request url
      * **data** - A parameter of type object, holding the data in the request body
      * **successCallBack** - A parameter of type function, defines the function called on request success
      * **errorCallBack** - A parameter of type function, defines the function called on request failure
  + *delete* function
    - Defines the logic used to initiate a **DELETE** request
    - Parameters: (url, data, successCallBack, errorCallBack)
      * **url** - A parameter of type string, defines the request url
      * **data** - A parameter of type object, holding the data in the request body
      * **successCallBack** - A parameter of type function, defines the function called on request success
      * **errorCallBack** - A parameter of type function, defines the function called on request failure
* **AuthorizationService Class**:
  + *constructor* function
    - Initializes the class
    - Parameters: (baseServiceUrl, appKey, appSecret, guestUserCredentials)
      * **baseServiceUrl** - A parameter of type string, defines the base service url of the API
      * **appKey** - A parameter of type string, defines the appKey of the current user in the API
      * **appSecret** - A parameter of type string, defines the appSecret of the current user in the API
      * **guestUserCredentials** - A parameter of type string, defines dummy user credentials, to make retrieval of data possible without login
  + *initAuthorizationType* function
    - Defines the Authorization type of the session authorization
    - Parameter: (authType)
      * **authType** - A parameter of type string, specifying the Session Authorization type for the current API
  + *getCurrentUser* function
    - Gets the currently logged in user and returns it
    - Parameters: ()
  + *isLoggedIn* function
    - Returns a boolean result, defining whether there is a logged in user, or not
    - Parameters: ()
  + *getAuthorizationHeaders* function
    - Generates and returns authorization headers for requests, based on several conditions
    - Parameters: (isGuest)
      * **isGuest** - A parameter of type boolean, specifying whether or not, to user the guest credentials if no user has been logged in
* **Notification Service**
  + *showPopup* function
    - Pops up a notification on the screen, of a given type, and with given text
    - Can be **success**, **error**, **info**, **warning**
    - Parameters: (type, text, position)
      * **type** - A parameter of type string, specifies the type of the popup notification
      * **text** - A parameter of type string, defines the content of the popup notification
      * **position** - A parameter of type string, defines the position of the popup notification if no such is given, the default is **top**
* **Event Service**
  + *initEventServices* function
    - Initializes everything needed for the service to work. This can be done only once
    - Parameters: ()
  + *redirectUrl* function
    - Defines logic for redirecting the client to a particular page of the application
    - Parameters: (url)
      * **url** - A parameter of type string, defines the url to which the client will be redirected
  + *bindEventHandler* function
    - Defines logic for binding event handlers to a particular event
    - Parameters: (event, eventHandler)
      * **event** - A parameter of type string, defines event name to which we want ot bind a handler
      * **eventHandler** - A parameter of type function, defines the handler function which will be called on event trigger
  + *onRoute* function
    - Defines logic for binding handlers to a particular route
    - Parameters: (route, routeHandler)
      * **route** - A parameter of type string, defines route to which we want ot bind a handler
      * **routeHandler** - A parameter of type function, defines the handler function which will be called when the user is on the specified route
  + *triggerEvent* function
    - Defines logic for triggering events
    - Parameters: (event, data)
      * **event** - A parameter of type string, defines the event that is being triggered
      * **data** - A parameter of type object, defines the data that is sent with the event trigger to the handler
  + *run* function
    - Defines logic for running the whole application on a particular root url (route)
    - Parameters: (rootUrl)
      * **rootUrl** - A parameter of type string, defines the root url on which the application will run

Modules and wrapped up library logic

* **Requester** - *JQuery*
* **AuthorizationService** - *JQuery*
* **NotificationService** - *JQuery Noty*
* **EventService** - *SammyJS*

**About**

This framework was created and documented by Ivaylo Jelev (Sanity) Github Alias - DCay. It is free to use under the terms and conditions of the MIT license. Criticizing and logging issues over the framework will be very helpful for the development of the framework.