Dev Guideline

# General Convention

` There is some agreement throughout this document.

* The base URL is the directory that contain the root folder of the application.
* The syntax of whole JavaScript code is based on **ES6**.
* */\*@ngInject\*/* or **'ngInject'**; are used to declare the injection for services, providers, and other injectable thing into target. It is used to replace the traditional declaration. Here is example of using them.

|  |
| --- |
| **Using ‘ngInject’** |
| **let** *controller* = **function** (SSOService,  $timeout,  toaster) {  **'ngInject'**;  **let** vm = **this**;  }; **export default** *controller*; |

|  |
| --- |
| **Using** */\*@ngInject\*/* |
| */\*@ngInject\*/* **export default** ($stateProvider) => {  $stateProvider  .**$state**(**'auth'**, {  **url**: **'/auth'**,  **abstract**: **true**,  **template**: **'<auth-master></auth-master>'** }); }; |

The difference of using two these syntax is that the first one is declared immediately after and inside the function that using these injections while the second one is declared before.

* **Angular UI Route** is used for configure routing in whole application. We had add some more feature in the original function **$stateProvider.state(…)** so that we can extend more routing optional (like permission for page…)

# How to create new module

To create a new module, you firstly create new folder named by the functional of that module in directory **src/javascripts/app/modules**. you then create new file named **index.js** that contain the declaration for module and used to link another files of current module.

You have to define something like this:

|  |
| --- |
| **index.js** |
| **let** AppAuthComponent = angular.module(**'app.core.masters.auth'**, [])  **export default** AppAuthComponent; |

The config declaration uses to declare routing. You have to declare it in this file and also some another providers (service, factory, value, constant..) that belong to current module. Using **import** syntax of ES6 to import another file content. You have to **export** the module so that we can use it in linked module file.

At this step, you created a new modules. But it can’t run itself and need to link to main module. This main module is bootstrapped then your module is also become part of the app.

# How to link a module to main module

To link an existing module create in step 2, go to the file that contain the declaration of main module located in **src/javascripts/app/modules/main.js**.

1: Import the module that you already created in Step 2 of this document. For example:

|  |
| --- |
| **src/javascripts/app/modules/main.js**. |
| **import** auth **from "./auth/index"**; // Or just simplify by import aside from “./auth”.  // […] |

Remember that, you have to export the module at the correct syntax that introduced in previous step.

2: You then declare the injection into the main app like this

|  |
| --- |
| **src/javascripts/app/modules/main.js**. |
| **import** auth **from "./auth/index"**;  // […]  **let** module = **angular**.module(**'app.modules'**, [ auth.**name**]);  **export default** module; |

# How to declare routing in a module

Rolling back to the context that you are creating new module. You then will need to create a routing configuration.

First of all, create a new file inside the current module.

|  |
| --- |
|  |
| *For example, we have module named* ***auth*** *and I created* ***index.js*** *in Step 2. I then create new file named* ***auth.route.js*** *that file name starts with the module name and end with the postfix* ***.route****.* |

In this file, just export an object that configure the routing like what you need.

Here is sample routing configuration for this module

|  |
| --- |
| **src/javascripts/app/modules/auth/auth.route.js** |
| */\*@ngInject\*/* **export default** ($stateProvider) => {  $stateProvider  .**$state**(**'auth'**, {  **url**: **'/auth'**,  **abstract**: **true**,  **template**: **'<auth-master></auth-master>'** }); }; |

What we need to notice here is the using of strange **$state** function rather than **state** as normally like **angular-ui-route.** As mention before, we had create new function that have run like the origin **state** but have some more feature.

After having creating a object that contains the configure of routing, you need to add it to current module. Come back to the **index.js** file and what we need to change is here.

|  |
| --- |
| **src/javascripts/app/modules/auth/index.js** |
| **import** *routes* **from './auth.route'**;  **let** AppAuthComponent = **angular**.module(**'app.core.masters.auth'**, [])  .**config**(*routes*) |

# How to create a component/ controller/ directives … in a module

To create a new component (or controller, directive, service …) of a module, we just re-use the old syntax like we did when we configure the routing.

Here is sample code that declares a new component.

Firstly, we need to create new file named: **auth.component.js**

|  |
| --- |
|  |

We then export the object that defines the component like this. Note that we can import template for using like below.

32

|  |
| --- |
| **src/javascripts/app/modules/auth/auth.component.js** |
| **import** templateUrl **from './auth.template.html'**;  **let** Component = {  **restrict**: **'E'**,  **bindings**: {},  **templateUrl**: templateUrl };  **export default** Component; |

In the **index.js** file. We add the code like this to declare new component:

|  |
| --- |
| **src/javascripts/app/modules/auth/index.js** |
| **import** Component **from './auth.component'**; **import** *routes* **from './auth.route'**;  **let** AppAuthComponent = **angular**.module(**'app.core.masters.auth'**, [])  .**config**(*routes*)  .**component**(**'authMaster'**, Component);  **export default** AppAuthComponent; |

For another providers. Just replace the **.component** with suitable declaration.

# The structure of common used features (services, components …)

Sometimes, you need to place some services to use in many place of the application. So, here, we need to understand the structure of **common** folder.

**common** folder is used only for define common using components, services… if you want to define a new page, new feature,… you HAVE to put your code in folder **modules**.

Here is some folders that nested inside **common** folder.

|  |
| --- |
|  |

Each folder has some sub-folder that contain a module to define material (component, service). The **/src/javascripts/app/common/index.js** is where to link all components, directives into one module. Each folder has **index.js** is used to link all sub-modules on that folder into one module. Note that you can also split a module into some sub-modules and place them directly inside that containing folder.

Here is the definition of using of all these folders in **common** folder:

* **components**: used to add some common component using throughout the application. For example: golden layout.
* **context**: contain only a service that store the current user context (e.g: user token, name, role…)
* **directive:** declare some common directives (loading, dynamic table, alert…).
* **filters:** places some angular filters.
* **resources:** Where to add some hard-code json files.
* **services:** add service to call API or some utility functions.
* **templates:** Put some HTML files to define views of components or template.
* **validators:** Where to add some directives that allow to validate input.

# How to create new common item (directive, component, service…)

To create new common item (directive, component, service... ), first of all, you need to follow steps to create new item on (5) and move the folder to the correspond sub-folder (as described in (6)) in common folder.

After that, go to the **index.js** file of the sub-folder that you have added the new item. Declare the linking to your newly added item. Or if you want your component is dynamic loaded, just skip this step. We will learn on how to dynamic load a component later.

Here is an example that you have added component **collaboration**.

|  |
| --- |
|  |

If you want to add your component into application. Just add code in **/src/javascripts/app/common/components/index.js**

|  |
| --- |
| **Src/javascripts/app/common/components/index.js** |
| **import** collaboration **from './collaboration'**; **let** module = **angular**.*module*(**'common.components'**, [  collaboration.name ]); **export default** module; |

*Note that, in case of your module has custom style, we will add these style file directly inside current module and then we will link them into parent container style file*

Here is an example on how we link style file. For example in component **collaboration** you have created new style file named **collaboration.style.scss**

|  |
| --- |
|  |

Then, in **src/javascripts/app/common/components/\_index.scss**, you will need to add

|  |
| --- |
| **Src/javascripts/app/common/components/\_index.scss** |
| **@import "./collaboration/collaboration.style"**;  […] |

From here, you can easily add some common Angular items in common or adding new module. So from now, we will learn on how to load them when you need.

There is something that you need to use almost whole project. So it is better to static load them as mentioned above. ( e. g: by linking them to containing module in common). While some another is used in some specialized modules, you will have to dynamic load them just when that module is loaded.

# How to dynamic load an Angular item (components, services … )

To dynamic load an Angular item, we need to load the containing modules that have that item.

To load a module into another module, we will combine both Angular UI Route Resolve and ocLazyLoad (<https://oclazyload.readme.io/>).

When we go to a page, the controller on that page is triggered. Some dependency injection is need already loaded before controller ran. The main idea here is we will load dependency modules that have some dependency injection in that controller before the controller ran by using route.resolve.

Please note that if your routing have some another resolve item and need dependency on some other service, we need to ensure the ocLazyLoad resolve be resolved firstly.

Here is example code that we recommend to use for dynamic loading some files.

|  |
| --- |
| **Src/javascripts/app/modules/manageData/index.js** |
| **[…]**  **resolve**: {  lazyLoad: ($q, $ocLazyLoad)=> {  **'ngInject'**;  **let** deferred = $q.*defer*();  ***require***.ensure([  **'app/common/directives/deleteDialog/deleteDialog'**, *// (1)* ], (require)=> {  *// Load file into app* **let** deleteDialogFactory = require(**'app/common/directives/deleteDialog/deleteDialog'**); *// (2)  // inject module* $ocLazyLoad.load([  {**name**: deleteDialogFactory.**default**.**name**},*// (3)* ]);   deferred.resolve();  });  **return** deferred.promise;  } },  **[…]** |

The above code is just a part of routing configure using Angular UI Route. In the example above, we need to load a service from file **deleteDialog**. So we need to ensure that that file is loaded into our system when this route reach. We just declare directory to the file and simple load that file like current without having to specify what factory or service we need.

# Using main.ocLazyLoad.js file to load libraries

As mentioned above, we have defined **$state** function that extend some more options for original **state** of **$stateProvider**.

Let see an example of newly option

|  |
| --- |
| **Src/javascripts/app/modules/manageData/index.js** |
| **[…]**  .**$state**(**'app.eventDetail'**, {  **url**: **'/event?cellName'**,  **views**: {  **'main'**: {  **templateUrl**: template,  **controller**: **'eventListDetailController'**,  **controllerAs**: **'$ctrl'**,   }  },  **authorization**: **true**,  **ocLazyLoad**: [**'ng-table'**, **'angular-moment'**, **'d3'**, **'nvd3'**],    […] });  […] |

As we can see, there is an option **ocLazyLoad**. It is just a shortcut way to load a library. Some key (**ng-table**, **d3**, **nvd3**..) we provided MUSTmatch keys on **window.OCLAZYLOAD** on **/src/main.ocLazyLoad.js**. The main idea is just look like what we did when we dynamic load on route. You can refer to **/src/javascripts/app/provider.js** file for detail implement. You can see on that file how we implement permission based on pages.

# Intercept API requests, responses

To intercept API request or response, we will refer to **/src/javascripts/app/provider.js** file. We can add headers for requests, or process results before it is resolved in promise.

# How to insert non-Angular library to current project

To insert a non-Angular library (for example: d3js, nvd3) to current project, first of all, either install that library via npm or copy the folder of library into **/src/js**. Then, simply import the library whether it is need.

# How to insert Angular library to current project

To insert an Angular library (for example: angular ui router…) to current project, first of all, either install that library via npm or copy the folder of library into **/src/js**. Then, we need to declare dependency to current module by go to file **/src/main.vendor.js** and add the name of module into field **modules**. From here, we can use all features of that module.

# How to create new golden layout window

The core idea of creating new golden layout is to wrap the whole content inside the window into Angular component. Then, we will use library golden layout to new a window.

We will consider the context in 2 situations: golden layout window is have no dependence to other window (e.g: for example the map window) and golden layout window that only open when another window is opened (e.g: new uti form).

We will also consider how to pass some data into component controller of that golden window.

We will also assume that each golden layout window contains ONLY ONE component. If we need more components, we need to wrap them into one component.

Here is steps to create new golden layout window that can run independence.

**Step 1** Create the component that wrap the content of whole window.

**Step 2** Create another component to wrap the component created in step 1.

We will refer to **/src/javascripts/app/common/components/goldenLayout/goldenLayout.controller.js** to follow the implement on how to new golden layout window when component got initialized.

|  |
| --- |
| **/src/javascripts/app/common/components/goldenLayout/goldenLayout.controller.js** |
| **[…]**  **this**.**\_$window**.**GL**.layout = **this**.goldenLayout = **new** GoldenLayout({  **settings**: {  **showPopoutIcon**: **false**,  **showMaximiseIcon**: **false**,  **selectionEnabled**: **true** },  **content**: [{  **type**: **'row'**,  **content**: [{  **type**: **'stack'**,  **content**: [{  **id**: **'networkMap'**,  **type**: **'component'**,  **title**: **'Network Map'**,  **isClosable**: **false**,  **componentName**: **'angularModule'**,  **componentState**: {  **component**: **'network'** *// Change this component name to what you like* }  }]  }]  }],  **dimensions**: {  **headerHeight**: 60,  **minItemHeight**: 460,  **borderWidth**: 0  } }, window.**jQuery**(**this**.**container**)); |

The above code will create new golden layout and apply to the element of children of this component (e.g.: div element has id = golden-layout

|  |
| --- |
| **/src/javascripts/app/common/components/goldenLayout/goldenLayout.component.js** |
| **let** Component = {  **restrict**: **'E'**,  **template**: **`<div id="golden-layout"></div>`**,  controller: Controller }; |

You can attach event to the golden layout window by reading more code in that file.

From here, when we attach the component created in step 2 into view, it will create new golden layout window that have content is the component in step 1.

The next step, we will make the golden layout window displayed when we go to a route.

**Step 3** Refer to the map module in **/src/javascripts/app/modules/map/map.js**. We will create a route like this

|  |
| --- |
| **/src/javascripts/app/modules/map/map.js** |
| **[…]**  $stateProvider  .$state(**'app.map'**, {  page\_title: **'Map page'**,  ncyBreadcrumb: {  label: **'Map page'** },  url: **'/map'**,  views: {  **'main'**: {  templateUrl: require(**'app/modules/map/templates/map.html'**),  controller: **'MapController'**,  controllerAs: **'map'** }  },  resolve: {  lazyLoad: ($q, $ocLazyLoad)=> {  **'ngInject'**;  **let** deferred = $q.defer();  require.ensure([  **'app/common/components/goldenLayout'** ], (require)=> {  *// Load file into app* **let** goldenLayoutComp = require(**'app/common/components/goldenLayout'**);  *// inject module* $ocLazyLoad.load([  {name: goldenLayoutComp.default.name}  ]);   deferred.resolve();  });  **return** deferred.promise;  }  }  });  **[…]** |

Note that like any other module declaration, we need to ensure the component that we use loaded firstly (e.g. it is golden layout component in the above example).

Let’s take a look on the template of this routing to see the component hooked into this route

|  |
| --- |
| **src/javascripts/app/modules/map/templates/map.html** |
| **[…]**  <**div id="page-map" style="height**: 100%”>  <**golden-layout**></**golden-layout**> </**div**>  **[…]** |

At this step, when we go to …/map url, it will load the golden layout window in ui view has name “main”.

The 3-step guide above has given us a way to show up a golden layout window when we traverse to a page. We will the run to next step is to mention about how to create new nested window on already opened window (for example, we need to open new uti form from map window).

The main idea is that we will create new service which will trigger function to create new golden layout window. We will also have to wrap the content of golden layout window into component.

The difference compares to the way new independence golden layout window is that we need to know the container of the parent already opened window that we will know when we register component.

|  |
| --- |
| **src/javascripts/app/common/components/goldenLayout/goldenLayout.controller.js** |
| **[…]**  **this**.**goldenLayout**.registerComponent(**'angularModule'**, (container, state) => {  $this.**$log**.*debug*(**'show loader for lazy load'**, container, state);  $this.**mapPageOverlayLoader**.show();  load[state.**component**]($this.**\_$q**, $this.**\_$ocLazyLoad**, $this.**\_$timeout**)  .then(() => {  $this.**\_$timeout**(() => {  $this.**$log**.*debug*(**'hide loader for lazy load'**, container, state);  $this.**mapPageOverlayLoader**.**hide**();  *// loops in content list then append html into container* **let** element = container.getElement(),  template = element.html(**'<'** + state.**component** + **' style="display: block; width: 100%;" class="h-full"></'** + state.**component** + **'>'**).**children**()[0];  *// Compile the content* $this.**\_$compile**(template)(**angular**.*extend*($this.**\_$rootScope**.$new(), {**container**: container, **state**: state}));  });  }); });  **[…]** |

Here is steps to make a window opened nested inside another one.

**Step 1** Create the component that wrap the content of whole window.

**Step 2** Write a service to call to create new golden layout window as we did with component in step 2 of above guide. Notice that we need to pass the container of parent window as the parameter of service function

**Step 3** Call that service, we have to pass the container of current window.

Here is example on the service declaration and how to trigger

|  |
| --- |
| **src/javascripts/app/common/services/goldenLayout/goldenLayout.service.js** |
| **[…]**  newUtiTab(container, uti, map, layer, latLng) {  **let** self = **this**;  **let** windowId = **'UTI\_'** + (uti ? uti.**utiId** || uti.id : 0);  **if** (self.setFocus(windowId)) {  **return**;  }  *// Tab config* **let** config = {  **id**: windowId,  **type**: **'component'**,  **title**: uti ? **'Edit UTI '** + uti.**utiName** : **'Add UTI'**,  **componentName**: **'angularModule'**,  **componentState**: {  **component**: **'create-or-update-uti'** },  **closeCondition**: **true**,  **popupable**: **false** };   *// Tab state* **angular**.*extend*(config.**componentState**, {  **map**: map,  **container**: container,  **data**: {  **model**: uti,  **layer**: layer,  **latLng**: latLng  }  });   container.parent.parent.addChild(config);  *// Save for popup/popin feature* **this**.**\_$window**.**GL**.**componentConfigs**[config.id] = config; }  **[…]** |

The syntax is just look like what we already did to new a golden layout window. The difference is just we need to append that window to container and it will open new window next to the parent window.

In the current working example project, we will call this service function as following:

|  |
| --- |
| **src/javascripts/app/common/components/radialSearch/radialSearch.controller.js** |
| **[…]**  self.**\_goldenLayoutService**.newUtiTab(self.**\_container**, resp.data, self.**\_map**, **null**, latLng); self.**\_utiEditedListener** = self.**\_$rootScope**.**$on**(**'CREATEORUPDATE.UTI.EDITED'**, (e, args)=> {  **if** (uti.**id** === args.uti.**id**) {  uti = \_.*extend*(uti, args.uti);  uti.name = args.uti.utiName;  } });  **[…]** |

The container in the first parameter is the parent container of current working window.

# How to pass data into component of golden layout window

Let consider the following configuration for a golden layout window.

|  |
| --- |
| **src/javascripts/app/common/services/goldenLayout/goldenLayout.service.js** |
| **[…]**  **let** config = {  **id**: windowId,  **type**: **'component'**,  **title**: uti ? **'Edit UTI '** + uti.**utiName** : **'Add UTI'**,  **componentName**: **'angularModule'**,  **componentState**: {  **component**: **'create-or-update-uti'** },  **closeCondition**: **true**,  **popupable**: **false** };  *// Tab state* **angular**.*extend*(config.**componentState**, {  **map**: map,  **container**: container,  **data**: {  **model**: uti,  **layer**: layer,  **latLng**: latLng  } });  **[…]** |

As we can see, we have **componentState** property so that we can pass data into component. We just fill data into this property and then use it in controller of component.

|  |
| --- |
| **src/javascripts/app/common/components/createOrUpdateUti/createOrUpdateUti.controller.js** |
| **[…]**  self.**\_state** = self.**\_$scope**.**$parent**.state; self.**\_layer** = self.**\_state**.**data**.layer; *// For case add uti in cell (should get lat lon from bts), else get latlng of layer* self.**\_latlng** = self.**\_state**.**data**.**latLng** || self.**\_layer**.latlng; self.**\_model** = self.**\_state**.**data**.model; self.**\_map** = self.**\_state**.**map**;  **[…]** |

We will have **state** property of parent scope from golden layout controller. To pass data into component of golden layout window that have no dependency from another window. We also just pass data into configure like this.