

LHCbPR Modules Development Guide

Prepared by Ben hammou Amine
June 2015

Contents

1. Introduction	4
2. The Web Application Files Structure	4
3. Getting Started	5
3.1. Setting Up The Web Application Locally	5
3.2. Creating New Module	6
3.2.1. The Module Class	6
3.2.2. Adding a controller	9
3.2.3. Adding a View	10
4. Using Directives	12
4.1. Search Jobs	12
4.1.1. Usage Example	13
4.1.2. Syntax	13
4.2. ngTable	14
4.2.1. Usage Example	14
5. Using LHCbPR API	16
5.1. Usage Example	16

List of Figures

Figure 1 : LHCbPR web application files structure	4
Figure 2 : Module files structure	4
Figure 3 : Commands To Setup The Web Application Locally	5
Figure 4 : Module Minimal Files Structure	6
Figure 5 : Sample module initialization	9
Figure 6 : Sample Controller Code	10
Figure 7 : Sample View Code	11
Figure 8 : Test view Screenshot	11
Figure 9 : Search Jobs Directive Screenshot	12
Figure 10 : Search Jobs Directive Usage Example - View Code	13
Figure 11 : Search Jobs Directive Usage Example - Controller Code	13
Figure 12 : Search Jobs Directive Syntax	13
Figure 13 : ngTable Usage Example - View Code	14
Figure 14 : ngTable Usage Example - Controller Code	15
Figure 15 : lhcbprResources Service Usage Example	16

1. Introduction

The LHCbPR web application was designed to be scalable and allow adding/removing modules easily. This document presents how to create new module and add it to the application.

2. The Web Application Files Structure

The LHCbPR web application has the following files structure:

```
bash
app/                # the build folder
master/            # the sources folder
|-- jade/          # the core application views
|-- js/            # the core application js files
|-- less/          # the core application styles
|-- modules/       # modules folder
|-- gulpfile.js    # configuration file for Gulp
|-- ...
vendor/            # the libraries folder
index.html
```

Figure 1 : LHCbPR web application files structure

All modules are stored under the `master/modules` directory. Each module have the following files structure:

```
bash
module_name/
|-- js/            # the javascript files
|   |-- controllers/ # controllers
|   |-- directives/  # custom directives
|   |-- filters/     # custom filters
|   |-- services/    # custom services
|   |-- init.js      # initialization file
|-- less/
|   |-- style.less   # custom styles
|-- views/          # views
```

3. Getting Started

3.1. Setting Up The Web Application Locally

In order to be able to clone and run the web application locally, you will need to install the following tools first:

Git: If you don't already have it, You can install Git using your package manager such as `yum` or `apt-get` .

Nodejs and npm: To install Nodejs and npm follow instructions on their website <https://nodejs.org/>

Bower: Plase run this command to install bower

```
npm install -g bower
```

bash

LiveReload: You can optionaly install LiveReload for you browser to enable automatic page reload when you change source files. Please visit this website for further instructions: <http://livereload.com/>

Now you can clone the web application and run it locally using the following commands:

```
# clone the github repository
git clone https://github.com/LHCbDev/lhcbpr-www.git
# go inside the master folder
cd lhcbpr-www/backend-angular/master
# install gulp and npm dependencies.
[sudo] npm install
# install vendor dependencies
bower install
# compile and run the application
gulp
```

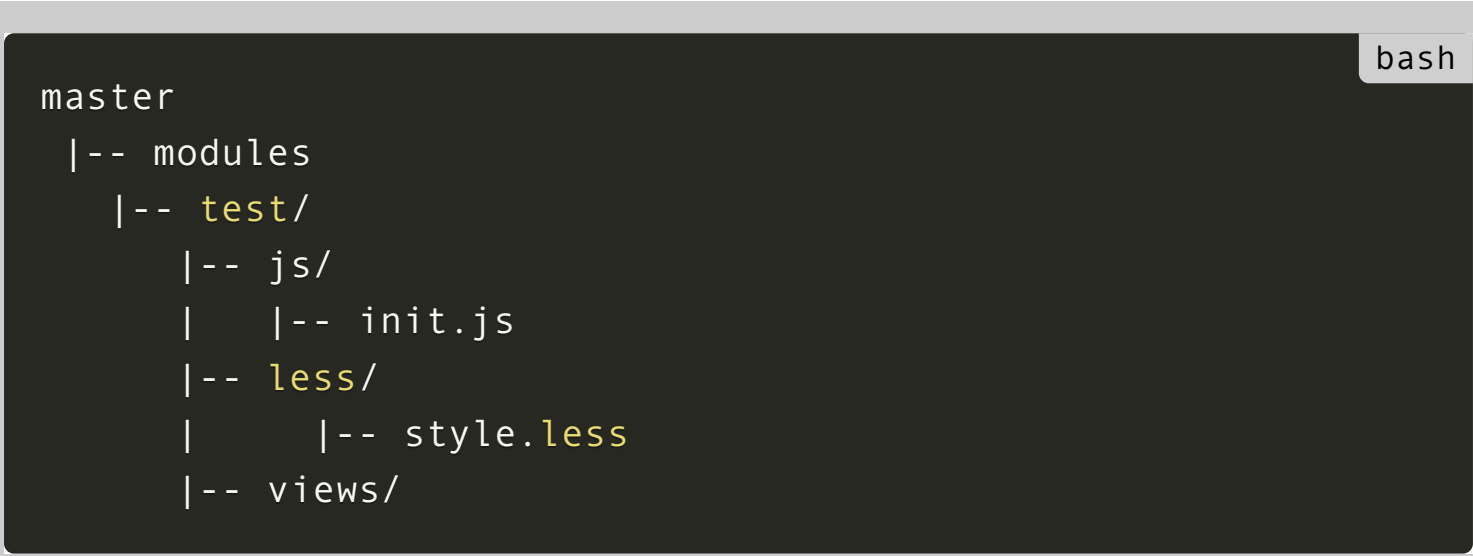
bash

Figure 3 : Commands To Setup The Web Application Locally

If everything goes fine, you should see the messages in the terminal telling you that most the task are done successfully. Going to <http://localhost:9000/> should show you the local website.

3.2. Creating New Module

Let's create a new module called `Test`. We start by creating a new directory under `master/modules` following the minimal module structure (we create only required files for the module to work; we can add more files later when needed):

A terminal window with a dark background and a 'bash' tab. It shows the command 'master' followed by a series of directory creation commands using 'mkdir -p'. The commands are: 'mkdir -p modules', 'mkdir -p test/', 'mkdir -p js/', 'mkdir -p js/init.js', 'mkdir -p less/', 'mkdir -p style.less', and 'mkdir -p views/'.

```
master
|-- modules
    |-- test/
        |-- js/
            |-- init.js
        |-- less/
            |-- style.less
        |-- views/
```

Figure 4 : Module Minimal Files Structure

Now we should write the initialization of our module on the file `test/js/init.js`. We will use the class `Module` for this.

3.2.1. The Module Class

This is a helper class simplifying the creation of modules. It offers easy methods and call the corresponding AngularJS methods behind the scene.

```
Module.create(name, title, position, settings)
```

name: The name of the module, it should be unique for each module.

title: The title of the module menu in the sidebar.

position: The position of the module menu in the sidebar. If two modules have the same position, there are shown one under the other.

settings: An optional plain object specifying the folder name of the module.

```
Module.create('name', 'Title', 1, {  
  folder: 'my_folder_name'  
});
```

javascript

If no settings is provided; the folder name is assumed to be the same as the module name.

This method returns an instance of the `Module` class, providing the following methods:

`addMenuItems(items)`

items: Menu object or array of menu objects. A menu object has the following format:

```
{  
  text: "Title", // Text to show on the sidebar  
  sref: "app.state.name", // Target state name  
  // prefixed by 'app.'  
  icon: "icon-grid", // Icon of the menu item  
  alert: "new" // Added a budge to the menu item  
}
```

javascript

This method returns the same calling `Module` instance so that we can chain calls to other methods.

addStates(states)

A state can be considered as a view or page of the module. One module can have multiple states and not all of them should have direct links on the sidebar.

states: State object or array of state objects. A state object has the following format:


```
javascript
{
  name: 'test.state_one', // Required
  url: '/my-url', // If not provided, it will be defined
  // based on the name ('/state-one' for this example)
  title: 'Title', // If not provided, it will be defined
  // based on the name ('State One' for this example)
  templateUrl: 'view.html', // name of the view file
  // with html extension instead of jade. If not provided
  // the view name will be 'state-one.html' in this case.
  controller: 'MyController', // name of the controller
  // handling the state. If not provided, it will be
  // 'TestStateOneController' for this case.
  resolve: ['test', 'chartjs'] // names of dependencies
  // on which this state depends. The name of its own
  // module should be part of the dependencies.
}
```

This method returns the same calling `Module` instance so that we can chain calls to other methods.

start()

This method should be called once all the desired menu items and states were added to the module. It executes the corresponding code on the AngularJS instance and integrate the module with the rest of the web application.

Now let's use this class to initialize our module. We will write the following code in the file `test/js/init.js` :



```
Module.create('test', 'Test Title', 1)
  .addMenuItems({
    text: "Test Module",
    sref: "app.test",
    icon: "icon-grid"
  })
  .addStates({
    name: 'test',
    resolve: ['test']
    // This will assume that:
    // title is 'Test'
    // templateUrl is 'test.html'
    // controller is 'TestController'
  })
  .start();
```

Figure 5 : Sample module initialization

The code above assume that we have a view and a controller. So we have to create them so that our module works properly.

3.2.2. Adding a controller

In Angular, a controller is a javascript function that handles a view or a part of the page. Take a look at the official documentation <https://docs.angularjs.org/guide/controller>

The controller javascript file can be stored anywhere inside the `test/js` directory. Let's store it inside a controllers directory like this `test/js/controllers/test.js` and write the following code:

```
/**
 * TestController
 */
App.controller('TestController', ['$scope', function($scope){
  // the $scope variable holds variables and methods
  // which can be used directly from the view
  // we define a variable message
  // with the value "Hello World !" for example:
  $scope.message = 'Hello World !';
  // and a function to show and alert
  $scope.showAlert = function(){
    alert('Here is the alert !');
  };
}]);
```

Figure 6 : Sample Controller Code

Please note that we have given the name "TestController" to our controller as assumed by the module declaration. The next step is to create a view and use the variable `message` and the function `showAlert()`.

3.2.3. Adding a View

A view is an HTML file to be rendered as part of the web page. But instead of writing views in basic HTML language which is very verbose. We are using the `Jade` template engine that makes writing HTML files more easier. Check the Jade official website for more details: <http://jade-lang.com>. One of the Gulp tasks is to compile every jade file and produce the corresponding HTML file. So you will not need to run Jade from the command-line manually.

Now Let's create our test view. its name should be `test.jade` because the templateUrl we are assuming in the module declaration is `test.html`. It should be stored inside the `test/views` directory.

```
h3 Just for test
div.alert.alert-info.
  the message is : {{message}}
button.btn.btn-primary(ng-click="showAlert()")
  | Click to show the alert
```

jade

Figure 7 : Sample View Code

Now after re-running the command `gulp` from the terminal. You should see the new test module added to the sidebar and once clicked it shows the view like this:

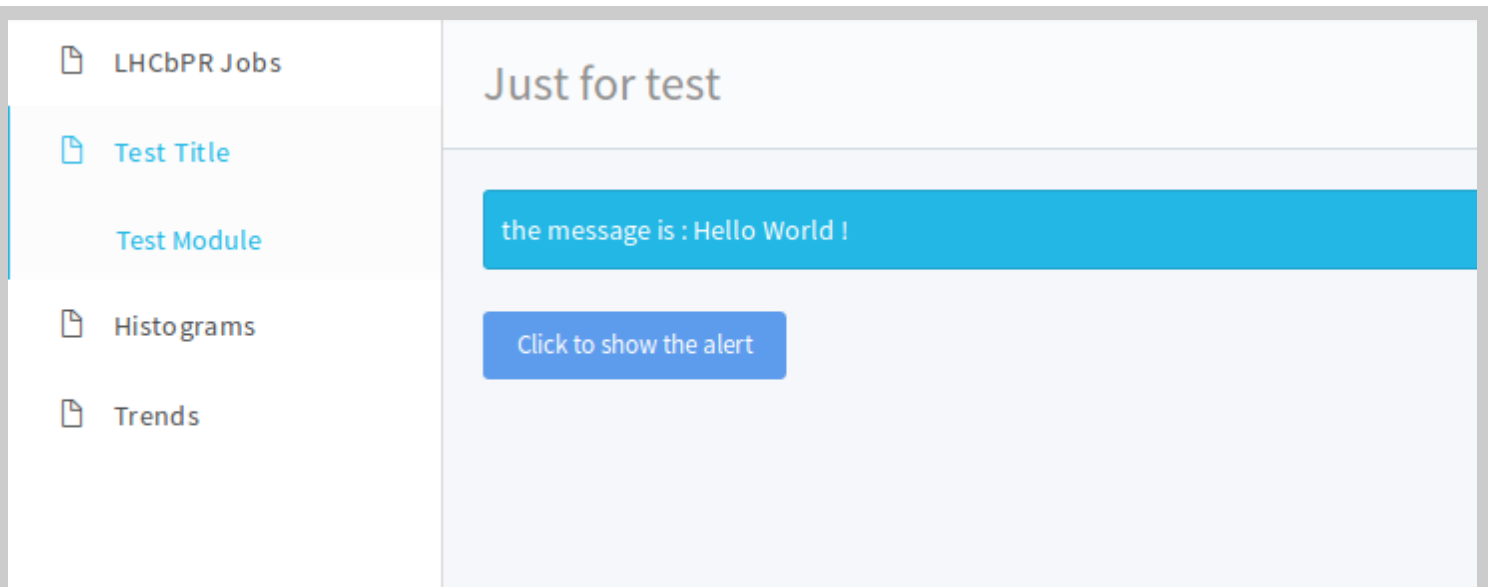


Figure 8 : Test view Screenshot

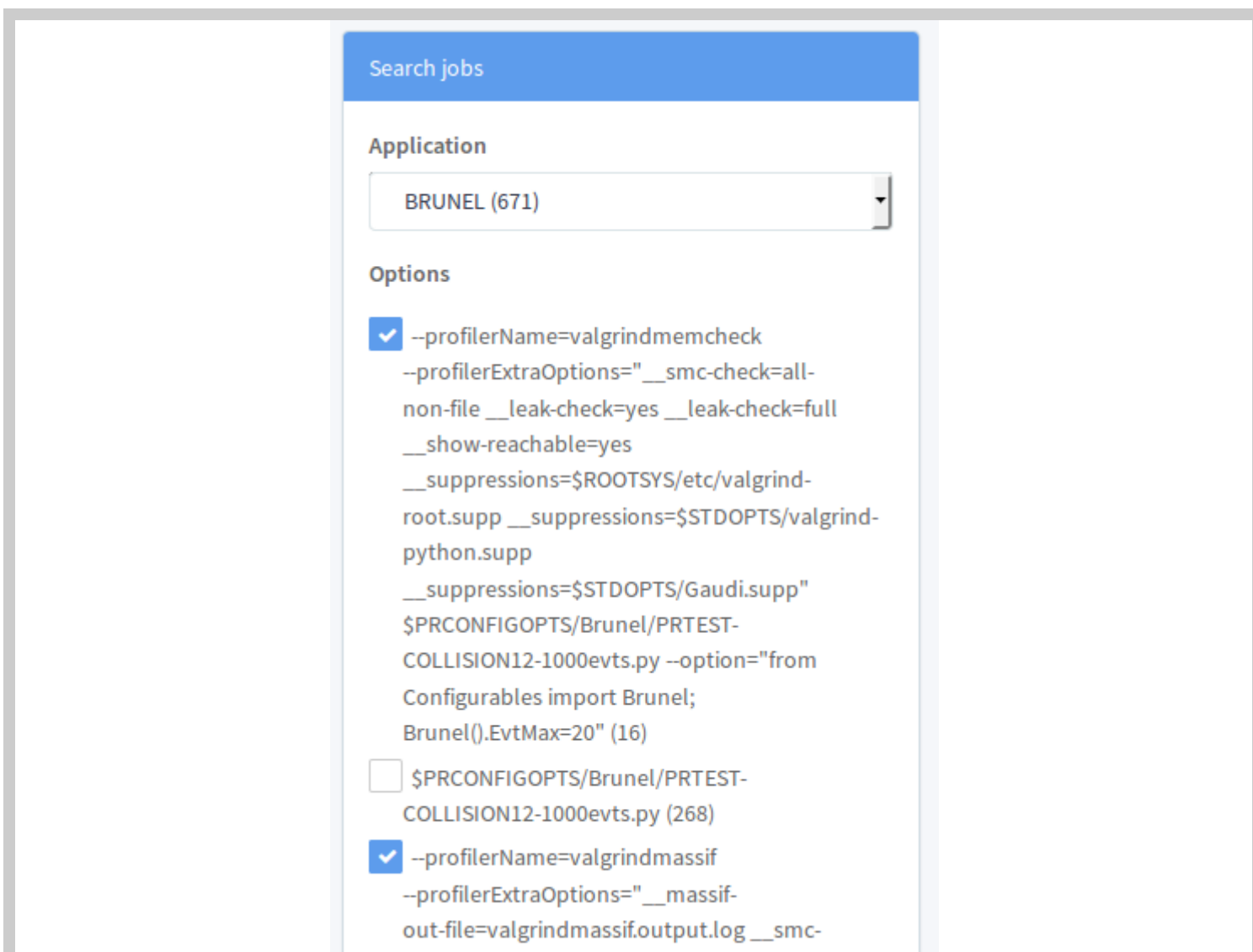
4. Using Directives

The LHCbPR web application contains many predefined directives that you can use in your modules. Check this link to know more about directives: docs.angularjs.org/guide/directive

The most used directives are explained below.

4.1. Search Jobs

This is the most used directive in analysis modules. It shows a form in which the user can filter jobs by application, options and versions. And notifies the controller each time the selection changes.



Search jobs

Application

BRUNEL (671)

Options

☒ --profilerName=valgrindmemcheck
--profilerExtraOptions="__smc-check=all-non-file __leak-check=yes __leak-check=full __show-reachable=yes __suppressions=\$ROOTSYS/etc/valgrind-root.sup __suppressions=\$STDOPTS/valgrind-python.sup __suppressions=\$STDOPTS/Gaudi.sup" \$PRCONFIGOPTS/Brunel/PRTEST-COLLISION12-1000evts.py --option="from Configurables import Brunel; Brunel().EvtMax=20" (16)

☐ \$PRCONFIGOPTS/Brunel/PRTEST-COLLISION12-1000evts.py (268)

☒ --profilerName=valgrindmassif
--profilerExtraOptions="__massif-out-file=valgrindmassif.output.log __smc-

Figure 9 : Search Jobs Directive Screenshot

4.1.1. Usage Example

Code to add on the view:

```
search-jobs(on-found="updateJobs(searchParams)")
```

jade

Figure 10 : Search Jobs Directive Usage Example - View Code

Code to add on the controller:

```
$scope.updateJobs = function(params){  
  // You can use the selected parameters:  
  // params.apps[0]: the selected application id  
  // params.options: array of selected options ids  
  // params.versions: array of selected versions ids  
};
```

javascript

Figure 11 : Search Jobs Directive Usage Example - Controller Code

4.1.2. Syntax

```
search-jobs(  
  on-found="updateJobs(searchParams)"  
  //- Required: the function to call with selected params  
  filter-options="true"  
  //- show options filtering? default is "true"  
  filter-versions="false"  
  //- show versions filtering? default is "true"  
)
```

jade

Figure 12 : Search Jobs Directive Syntax

4.2. ngTable

This directive can be used to add table with sorting, filtering and pagination features.

4.2.1. Usage Example

Code to add on the view:

```
jade
table.table.table-striped.table-bordered.table-hover(
  ng-table="tableParams")
thead
  th ID
  th Name
tbody
  tr(ng-repeat="attr in $data")
    td {{attr.id}}
    td {{attr.name}}
```

Figure 13 : ngTable Usage Example - View Code

Code to add on the controller:

javascript

```
App.controller('TestController', [
  '$scope', 'ngTableParams', function($scope, ngTableParams)
  {
    // We define some hard coded data
    $scope.attrs = [
      { id: 56, name: "EVENT_LOOP" },
      { id: 57, name: "EVENT_LOOP_count" },
      { id: 58, name: "EVENT_LOOP_rank" },
      { id: 60, name: "EVENT_LOOP_id" },
      { id: 81, name: "BrunelInit" },
      { id: 82, name: "BrunelInit_count" },
      { id: 83, name: "BrunelInit_rank" },
      { id: 85, name: "BrunelInit_id" },
      { id: 91, name: "L0DUFfromRaw" },
      { id: 92, name: "L0DUFfromRaw_count" }
    ];

    // Table parameters
    $scope.tableParams = new ngTableParams(
      { page: 1, count: 10 },
      { total: 0, getData: function($defer, params) {
        // We just show the hard coded values
        $defer.resolve($scope.attrs);
      }
    });

    // Add this line to fix a bug in the ng-table directive
    $scope.tableParams.settings().$scope = $scope;

  }]);
```

Figure 14 : ngTable Usage Example - Controller Code

For full ngTable documentation, please visit the official site: ng-table.com

5. Using LHCbPR API

The `lhcbprResources` service is based on `Restangular` and can be used to interact with the API.

5.1. Usage Example

here is an example retrieving the list of active applications

```
javascript
App.controller('TestController', [
  '$scope', 'lhcbprResources', function($scope, lhcbprResources)
  {
    // Get the list of active applications for example
    lhcbprResources.all('active/applications')
      .getList()
      .then(function(response) {
        // Do what ever you want with the response
        // This will show it on the console
        console.log(response);
      });

    // Some other code here
    // Please note that the AJAX calls are asynchronous
    // which means that the code here maybe executed while
    // waiting for the response

  }]);
```

Figure 15 : lhcbprResources Service Usage Example

For full Restangular documentation please visit <https://github.com/mgonto/restangular>