**Workshop 4 – AngularJS**

Please get the ‘FrontendCoding’ project from github. Run the necessary commands for node packages and bower dependencies.

**Part I. Angular Js Directives**

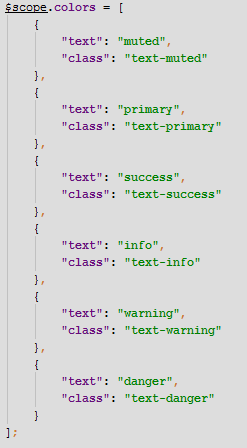
1. Add the following route to app.js file :

. when('/colors', {

templateUrl: views /demo/colors.html',

controller: 'Colors'

})

1. Open Colors.js and create the following array of objects (TODO #1) 
2. Open colors.html and create a select element in order to list all the “text” property of each element in the array above. Use ng-options with track by text(TODO #2 ).
3. In Colors.js create a changeColor() function. This function should save, in a scope variable, the selected color class. (TODO #3)
4. For the select element created use ng-change directive that will trigger the execution of changeColor() function.
5. In colors.html create a span element with some text “Some text” and use ng-class to change the css class of the span with the value of the “class” property of the selected object. Use the scope variable created at point 4 (TODO #4). Check that the font color changes for the text in the span.

**Part II. Forms**

1. Add the following route to app.js file :

.when('/forms', {

templateUrl: views /demo/form.html',

controller: 'Form'

})

1. Open Form.html and create an ng-form element. For the created form list all the available properties listed in the presentation (must be 6 in number). (TODO #5)
2. In Form.html create an input element and make it required.
3. For the created input element list the same properties as for the form (TODO#6)
4. In Form.html create a new span element with the text “Required!”. This element should be visible if the form input element has no value. (hint use $error TODO#7)

**Part III. Filters**

1. Open colors.html and create two input elements.
2. Using ng-repeat list the “text” property values of all the elements in colors array from Colors.js.
3. Use the value of the first input element to filter the list above and the value from the second input to order the filtered list (TODO#8)

**HR App:**

At the end of this workshop we will have the possibility to create and edit an employee.

1. Add routes for Employee add page:

.when('/employeeadd', {

templateUrl: views/employeeadd.html',

controller: 'EmployeeAddController'

})

2. Open 'EmployeeAddController.js'. In this controller store all the jobs, departments and managers. (hint see EmployeeListController) . The data must be stored in departments, managers and jobs scope variables.

Use the $commonResourcesFactory.findAllDepartmentsUrl for getting the departments, $commonResourcesFactory.findAllEmployeesUrl for getting all the managers and $commonResourcesFactory.findAllJobsUrl for getting all the jobs. (TODO#HR1).

Create a empty object named employee. This will be the object that we will build on to create the employee.

3. Now that we have all the elements we need let us create the view. Open employeeadd.html and create the following missing fields :

* firstName (input) -> use ng-model to bind to employee.firstName
* lastName (input) -> use ng-model to bind to employee. lastName
* email (input) -> use ng-model to bind to employee. email
* phoneNumber(input) -> use ng-model to bind to employee. phoneNumber
* jobTitle (select) -> use ng-model to bind to employee. jobTitle
* salary(input) (see manager filed for example) -> use ng-model to bind to employee. Salary
* department (select) (see manager filed for example) -> use ng-model to bind to employee. departmentId

Make all created field required and use “form-control” as css class for the inputs . For each created element create a label (see manager field for example).

(TODO#HR2)

4. After creating the elements , for the parent div of each element use ng-class to set ‘has-error’ css class if field is invalid(hint see manager field as an example). (TODO#HR3) .

5. Create Save and Reset buttons that use reset() and create(employee) functions from 'EmployeeAddController'. If the form is invalid the save button should not be enabled. Use ng-disabled to disable it (TODO#HR4).

6. Check if the add employee option works.

7. Add routes for Employee edit page:

.when('/employeeadd', {

templateUrl: views /employeeedit.html',

controller: EmployeeEditController

})

8. Open 'EmployeeEditController.js'. In this controller store all the jobs, departments, managers (hint see EmployeeListController) and search for employee with and specific id(see EmployeeViewController). The data must be stored in departments, managers, jobs and employee scope variables.

Use the $commonResourcesFactory.findAllDepartmentsUrl for getting the departments, $commonResourcesFactory.findAllEmployeesUrl for getting all the managers ,$commonResourcesFactory.findAllJobsUrl for getting all the jobs and $commonResourcesFactory.findOneEmployeeUrl+$routeParams.employeeid (TODO#HR5).

9. Open employeelist.html and add a button for edit. This button should use editEmployee(employeeId) function. (hint see view button DOTO#6)

10. Open employeeedit.html and use the same code from the form in employeeadd.html . (hint in order for the validations to work you need to use employeeEditForm TODO#HR7).

**Optional:**

As for Employee create List, View, Create and Edit options for Jobs.

The Jobs object has the following fields:

* jobId: text
* jobTitle: text
* maxSalary: number
* minSalary: number