HR App

- Frontend Day 5 – Workshop -

There are several bugs spread throughout the source code. Pay attention to details; consult the **Developer Console** in browser.

At the end of this workshop, one will have the ability to create and edit an employee.

1. Add routes for Employee add page:

```
.when('/employeeAdd', {
     templateUrl: 'views/employeeAdd.html',
     controller: 'EmployeeAddController'
 })
```

2. Open the `EmployeeAddController.js` file. 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.

   Make use of `CommonResourcesFactoryBackup.findAllDepartmentsUrl` in order to get the departments, `CommonResourcesFactoryBackup.findAllEmployeesUrl` to get the managers and `CommonResourcesFactoryBackup.findAllJobsUrl` to obtain the jobs. TODO#HR1

   Nice-To-Know: A best practice would be to store these functionalities in a service (Hint: See `EmployeeService`). Ask what a `$promise` is.

   Create an empty object dubbed 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. jobId
- `salary` (input) (*Hint*: See manager field for example) -> use `ng-model` to bind to `employee.salary`
- `department` (select) (*Hint*: See manager field 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` (*Hint*: 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('/employeeEdit', {
     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 a specific id (*Hint*: See `EmployeeViewController`). The data must be stored in departments, managers, jobs and employee $scope variables.

   Employ `CommonResourcesFactoryBackup.findAllDepartmentsUrl` to get the departments, `CommonResourcesFactoryBackup.findAllEmployeesUrl` to retrieve all the managers, `CommonResourcesFactoryBackup.findAllJobsUrl` to obtain all available jobs and `CommonResourcesFactoryBackup.findOneEmployeeUrl + $routeParams.employeeId`. TODO#HR5
9. Open `employeeList.html` and add a button for editing. This button should use the `editEmployee(employeeId)` function (*Hint*: See 'View' button). TODO#HR6
10. Open the `employeeEdit.html` file and use the same code from the form present in `employeeAdd.html`. (*Hint*: In order for the validations to work, you need to use the `employeeEditForm`). TODO#HR7
11. Also, create deletion functionality analogous to add/edit.

- Additional Exercise -

As with employees, create 'List', 'View', 'Add' and 'Edit' actions, but this time for jobs. The `job` object has the fields enumerated below:

- `jobId`: text
- `jobTitle`: text
- `maxSalary`: number
- `minSalary`: number

Extra: `$watch`, `$timeout`, `$interval`