A Developer Diary
{about:"code learn and share"}

October 14, 2015 By Abhisek Jana — 3 Comments (Edit)

How to Learn and Master Angular easily – Part3

Welcome to the “How to Learn and Master Angular easily – Part3” !

In chapter 3 we will learn on how to create Modules in Angular js. Then we will start developing our TaskTracker application and learn few Angular native directive such as ng-repeat, ng-class etc. We will slowly build the foundation that is required. It is very important to start slowly and understand each concept.

In Part2 we have learned about Angular Architecture, Service , Controller, View, Two Way Binding etc.

Module:

Modules are like plugins, where all of the related logics are placed together. Module helps if you are creating a common functionality which can be used across multiple application, then you can add the common module you created as a dependency to your app.You will create at least one Module for any application, in Part 2 the code snippets had the module created.

Question: You might have many modules created, but how to tell angular which module is the parent application? or using different words, how to bootstrap an

Angular JS Application ?

Answer: There is a directive named `ng-app` where you specify the module name of your application.

Here is the code snippet from Chapter 2. Notice the module name `TaskTracker` was given in `ng-app`.

```
    {{message}}
```

Now lets see how to add a custom module as a dependency to the application and use it.

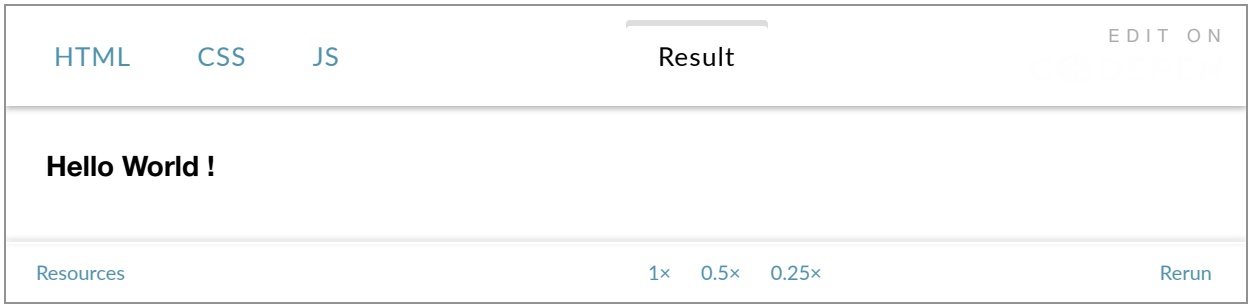
Module Dependency:

Here is an example of how we can create custom module and use them across the application using `Dependency Injection`. In case you had noticed the `[]` in the second argument when we were defining the module name, this is where we need to specify the dependent module name.

In this example we created a custom module name `MyModule` and declared that as a dependency in our application named `TaskTracker`. Then we just had to mention the name of the service in our controller in order to call the `callSomething()` method.

```
{{message}}
```

Here is the live example:



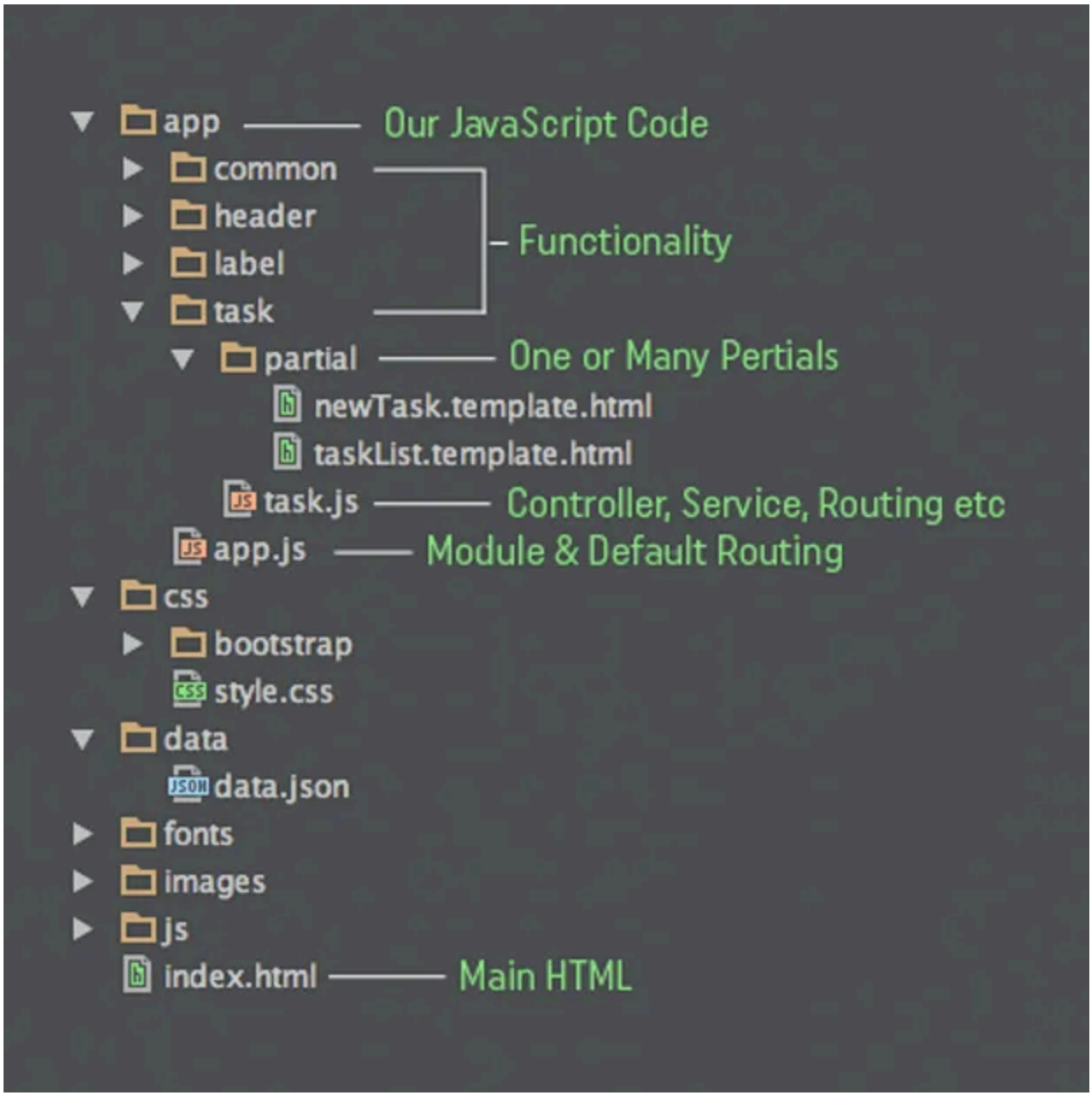
This is a very simple example, in real life mostly reusable directives are used as module, like UI Grid, UI Router , Bootstrap UI etc. You can think modules are just like .DLL or .JAR or .SO files which can be easily plugged into any application.

Note : The `ng-app` can be defined in `html` element also rather than in the `body` element.

Folder Structure :

Its very important to maintain a proper folder structure in Angular for any enterprise project since at runtime all of them belogs to the same context and it will be very difficult to maintain.

In our TaskTracker Application we will follow the following structure.



Inside the app folder we have the app.js and then different folders for each functionally.Then each of them will have a folder named partials for the partials(view/template) and one js file for all the necessary angular component such as Controller, Service, Routing, Filter etc.

In case you are working on a very complex project you can create one js file for each Angular component like task.js, task-controller.js, task-service.js, task-directive.js etc.

Lets make changes to the TaskTracker application. Download the chapter 1 html code from github from the following link and start from there.

`https://github.com/adeveloperdiary/angular-for-web-developers`

Add the `angular_1.4.js` before closing the head element after the other libraries. The angular 1.4 library is already included added in the chapter 1 js folder.

Add the `ng-app="taskTracker"` in the body element in `index.html`. This will bootstrap the application.

```
....
```

Now, add a folder name app and create a file named `app.js`. Create an Angular module name `TaskTracker`;

```
var module=angular.module('taskTracker',[]);
```

Add the `app/app.js` at the end of the body element.

Now run the index.html in Chrome. If you are using bracket.io or webstrom then it will automatically have the web server. Otherwise you need a webserver or appserver to be installed.

Open the developer tools are make sure you are not seeing any error in the console.

Let me know in case you face issue with the setup and I will help you.

Now create a controller in the `app.js` named `labelController` and add the label data in the `$scope`.

```
var module=angular.module('taskTracker',[]);

module.controller('labelController',function($scope){
    $scope.data={
        "labels": [
            {
                "name": "Work",
                "color": "color_red"
            },
            {
                "name": "Home",
                "color": "color_green"
            },
            {
                "name": "Personal",
                "color": "color_blue"
            }
        ]};
});
```

Open the index.html, now we will update the `sidebar-nav` `ul`. The first `li` is the application name and next three of them are the fixed label named Inbox, All Tasks and All Pending.

In our TaskTracker application user can add any custom label, so lets add them dynamically from the JSON object we created in the controller.

- Task Tracker

- Inbox
  14

- All Tasks
  14

- All Pending
  14

-

-

{{label.name}}

14

Here the ng-controller was added to attach with the View as learned in Part/Chapter 2. There are few new directives were added in the code, before we go though them, lets reload the index.html and you should be able to see the custom Labels added.

ng-repeat :

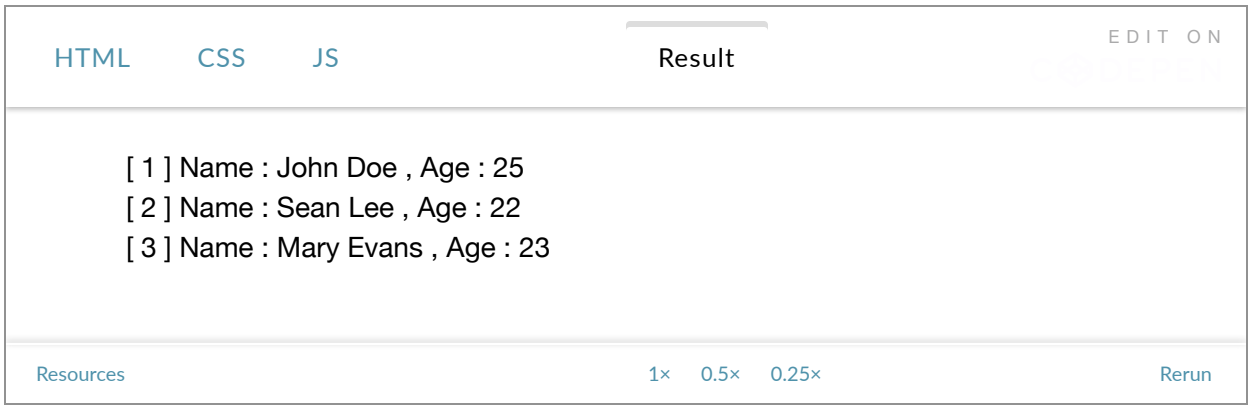
`ng-repeat` is the for loop in angular which takes an array and loops through it. Here the `data.labels` is the array as we have defined in the labelController's $scope object. The label is the local variable for each of the object.

Lets see few more example of this to understand fully.

- ```
  [ {{$index+1}} ] Name :  {{person.name}}
  , Age : {{person.age}}
  ```
  ```

list is an array of persons with name & age. It has been assigned to the `$scope` in the controller.
Now we are iterating through the array using `person in list` where each person represents one person object.
Then we are accessing the name and age using `person.name` and `person.age`. We are doing one more thing extra here, the `$index` represent the index of the element in the given array. We are accessing it and showing it in the list. We are not using `$index` in our TaskTracker application, however it can be useful in many scenarios.

Here is the live demo.



ng-class:

This is another directive to specify a css class name for any element. In our TaskTracker application we are dynamically setting the color of the custom Labels using `ng-class`. The `label.color` has the css class name for the different color.

Notice, we didn't have to use the `{{}}` in the `ng-class` or `ng-repeat` directive. Any angular directive (Native or Custom) can take an expression and the expression does not need any the curly braces since they are already Angular component.We will see more example of expression later.

Now lets load the list of tasks. Here is the JSON for tasks.

```
{"tasks":[
    {
        "id":1,
        "name":"Send Status Report",
        "dueDate":"12/30/1981",
        "note":"",
        "completed":false,
        "labelName":"Inbox"
    },
    {
        "id":2,
        "name":"Learn Angular JS 1.x",
        "dueDate":"12/30/1981",
        "note":"",
        "completed":false,
        "labelName":"Work"
    },
    {
        "id":3,
        "name":"Purchase Grocery",
        "dueDate":"12/30/1981",
        "note":"",
        "completed":true,
        "labelName":"Home"
    }
]}
```

Each task has an id, name, completed status and labelName. Rest of the elements we will not be implementing in this series,this is something that you should complete integrating by yourself at the end of the series.

Lets define this in our controller and name it as `data1` then assign it to the `$scope` object. This should look like `$scope.data1={ "tasks": [ ... ] }`

Now open the `index.html` and make following changes.

In the `list-group` `ul` element, keep only the first `li` element and delete rest of them.

Add the following code.

```
<ul class="list-group">
    <li class="list-group-item">
        <input type="checkbox" id="{{task.name}}" />
        <label for="{{task.name}}">{{task.name}}</label>
    </li>
</ul>
```

We are looping through the `data1.tasks` array and populating the task name in the label element. Open & test the `index.html` in Chrome.

Note : We need to use `{{}}` in the id attribute of the input element since id is not an angular related and it's a default html attribute.

You will not see any tasks getting displayed !!! There are also no error in the console. So whats wrong here ?

As we learnt in Part 2, every view needs to be tied to at least one controller so that the view can access the `$scope object` of the controller. If you look closely, the `labelController` we added in the `#sidebar-wrapper` `div` is already closed before `#page-content-wrapper` `div`. So `labelController` will not be accessible by our code written outside of the `#sidebar-wrapper` `div` element.

Now we shall create a new controller named `taskListController` and define it to the `#page-body` `div` element. Lets do that in our `app.js` file. Move the `data1` in this `taskListController`.

```
module.controller('taskListController',function($scope){
    $scope.data1={
        "tasks":[
            {...

        ]};
});
```

Add the `taskListController` to the `#page-body` `div` element.

```
 ..
```

Now if you refresh the index.html page you can see all 3 tasks getting displayed there.

Before we finish this part, lets clean up some of the code.

If we keep adding controllers in the `app.js` then very soon the app.js will be very big and difficult to maintain. Lets create few more folders.

- Create two folders named `label` and `task` inside our `app` folder.
- Create a js file named `label.js` inside the label folder and move the `labelController` inside this file.
- Create a js file named `task.js` inside the label folder and move the `taskListController` inside this file.
- Add the new js files after the app.js in the index.html.

Here are the files:

```
var module=angular.module('taskTracker');

module.controller('labelController',function($scope){
    $scope.data={"labels": [ ... ]};
});


var module=angular.module('taskTracker');

module.controller('taskListController',function($scope){
    $scope.data1={"tasks":[ ... ]};
});

var module=angular.module('taskTracker',[]);


...
```

If we don't have the 2nd argument while creating a module then Angular will not create a new module, rather it will assume that the module already exists and it will add the new components to the existing module. In `label.js` and `task.js` we didn't define the 2nd argument in the `angular.module()`

function so Angular will just add the controllers to the taskTracker module we created in our app.js.

The code should work as it already was. This will be the end of Part 3.

Related

How to Learn and Master Angular easily – Part2
In "Angular 1.x"

How to Learn and Master Angular easily – Part4
In "Angular 1.x"

How to Learn and Master Angular easily – Part5
In "Angular 1.x"

Filed Under: Angular 1.x, JavaScript Tagged With: Angular, Angular JS, Code, Controller, example, JavaScript, Learn, master, Module Dependency, ng-class, ng-repeat, Programming, Service, step by step, tutorial

Comments

Alex says
February 13, 2016 at 6:39 pm

(Edit)

The NG-CLASS title:
The following paragraph must also contain the reference of adding the ng-controller=”labelController”
// In the list-group ul element [add ng-controller=’labelController’], keep only the first li element and delete rest of them.

Reply

Alex says
February 13, 2016 at 6:41 pm

(Edit)

Ohh, my bad. Please remove these two comments. The next text explains it well. I stopped reading there and started looking for the bug. Maybe a hint earlier could have saved me some time :>

Reply

A Developer Diary says
February 14, 2016 at 5:38 am

(Edit)

> Not a problem Alex … let me know in case you face any issue again.
>
> Reply

Leave a Reply

Logged in as Abhisek Jana. Edit your profile. Log out? Required fields are marked *

Comment *

Post Comment

This site uses Akismet to reduce spam. Learn how your comment data is processed.