# **Directives**

Angular allows you to extent HTML with new attributes, these are called directives.

Info @ <https://docs.angularjs.org/guide/directive>

Directives allow you to reduce hardcoding in all the various JavaScript and html into the html/ejs file. This allows you to put a simple custom html tag to include the directive you wish to use. This allows for

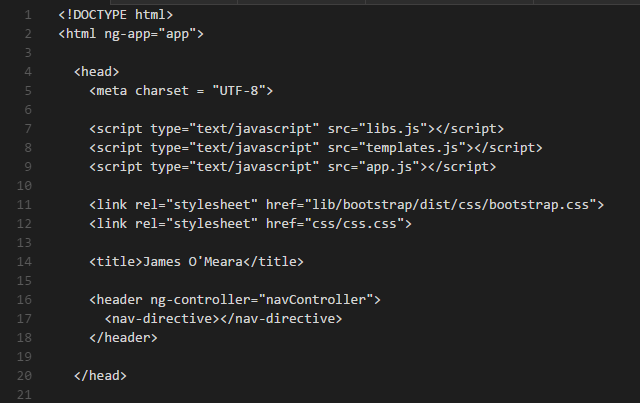
Directives are simple to create, can provide a wide range of functionality and re-use. Directive can include its own logic and can be passed in data and process it itself, e.g. pass in images with text: the directive could provide the layout for the image and surrounding text and display them in a grid.

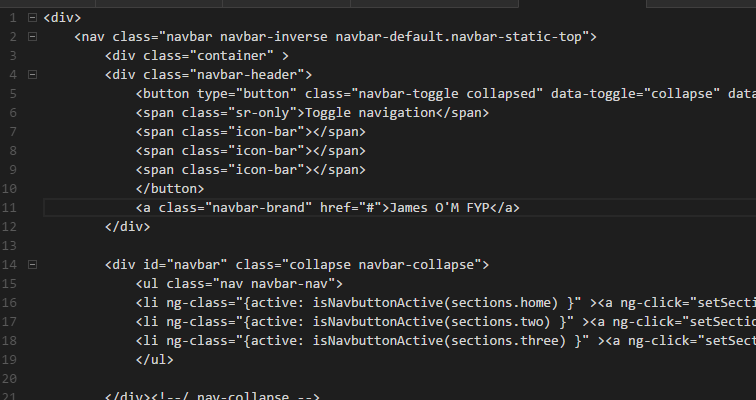
e.g.:

Here is a simple directive taking the bootstrap nav template and creating a directive for it. This will result in our index file size being significantly reduced and easier to manage.

Directive: simpleDirective = html: <simple-directive>.

This will allow our index.ejs || index.html to contain a lot less code as we can store the html required in template files for the directives, or even in the directive JavaScript file itself.





# **Creating a directive**

We can extend our angular module and create a new directive by:

**yourAngularAppName.directive( “desiredName” , [function () {**

**return {**

**};**

**}]);**

There are many different types/styles of directives that you can create, this influences the way it will replace the html as the directives are compiled or how scopes are used. In the object that the directive will return we can add multiple things.

* Controller
* Restrict
* Transclude
* Scope
* Replace
* Template
* TemplateUrl
* Link (Function)

**Controller:**

Required controller for the instance of this Directive

**Restrict:**

This is to define how the directive is triggered, this is how we call our directive in the html. Whether we call the directive by attribute name, element name class or comment

‘A’ – matches by attribute name

‘E’ – matches by attribute name

‘C’ – matches by class name

‘M’ – matches by comment

Or

‘ABC’ – will allow match by either attribute class or attribute name

Code:

**Return{**

**Restrict: ‘E’**

**}**

**Transclude:**

This makes the directive nest its scopes, meaning that any directive that uses this can access its parent scope via transclusion. Directives can use its own scope within its template, but if necessary it can transclude to access the outer scope.

More @ <http://teropa.info/blog/2015/06/09/transclusion.html>

Also: <http://angular-tips.com/blog/2014/03/transclusion-and-scopes/>

**Scope:**

Directives may have their own scope; this is called an isolate scope. By default, directives inherit their parent scope, but if you wish to isolate the scope for any given directive, add the attribute scope to the returned object in the directive:

**return{**

**scope: {}**

**}**

We can pass in data into our directive via or declaration in the html file:

**<my-directive data={{ dataInControllerForExample }}>**

Inside the directive:

**return{**

**scope: {**

**data: ‘=’**

**// the ‘=’ will keep a 2 way data bind with the data passed in**

**}**

**}**

With this way of initiating the scope within the directive, gives more power to what the directive can access and what it can accept.

More @ <http://angular-tips.com/blog/2014/03/transclusion-and-scopes/>

& <https://weblogs.asp.net/dwahlin/creating-custom-angularjs-directives-part-2-isolate-scope>

**Replace:**

This will determine whether the tag that we used to call the directive is replaced or not.

EG

If our directive has one line of html -> <p> some line of text </p>

Index page:

**<div>**

**<my-directive></my\_directive>**

**</div>**

When: **replace: true**

html = **<div>**

**<p> some line of text </p>**

**<div>**

When: **replace: false**

html = **<div>**

**<my-directive>**

**<p> some line of text </p>**

**</my-directive>**

**<div>**

**Template & TemplateUrl:**

This attribute will allow us to embed html along with the directive, either we can hardcode it into the directive or we can specify a file for the template.

**template: ‘<p>Enter html code here</p> etc',**

**templateUrl: 'location/of/directiveTemplate.ejs',**

**Link function:**

The link functions purpose is to provide any logic specific to this directive. This is similar to a controller, but this makes the functions and logic here only accessible by the inner workings of the directive and not any other controller or service/factory.

**return**

**link: function(scope, element, attrs) {**

**}**

**}**

More @ <http://websystique.com/angularjs/angularjs-custom-directives-link-function-guide/>

**Using a service or factory in a directive**

In the declaration of the directive add it as an input

**app.directive(‘directiveName’, [‘factoryName’, ‘serviceName’, function (factoryName,**

**serviceName) {**

**return {**