**Contents**

* Angular
* Getting Started
* Components
* Controller
* Directive
* Factory
* Service
* Filter (TO DO)

**AngularJS**

AngularJS provides great functionality to a website by extending the functionality html can provide. it addresses the problem with html, which is making a webpage dynamic, and can do this by loading the required JavaScript files into the browser and will evaluate the functions and expressions. This allows for some powerful features, for example 2-way data binding and dynamic web pages., this allows for less requests to the server, and makes use of the local browser evaluating the JavaScript creating a dynamic webpage.

Info @ <https://angularjs.org/>

Single page apps

<https://blog.codecentric.de/en/2014/08/angularjs-browserify/>

layout and module loading with browserify

load templates with browserify

<https://www.npmjs.com/package/grunt-angular-templates>

ng route

<https://docs.angularjs.org/api/ngRoute>

Node is server side JavaScript, and takes advantage of the browser doing some of the bulk of the work, instead of passing requests back and forward to the server. The JavaScript will be loaded along with the page and the browser will evaluate the functions and expressions. This allows for some powerful features, for example 2 way data binding and dynamic web pages.

**Get started:**

To start out we must include angular in our app

We can do this by installing it through Npm. Although Npm is the package manager for node, there are additional package managers, we are going to use bower, bower is a package manager for client side dependencies. <https://github.com/bower/bower>

We can install this by “**npm install bower -g**”

We are installing this module globally so we can use command line with it.



Then we must initialize bower, using the “**bower init**” command, this will create a bower.json, similar to the way we initialized npm and it created the package.json file for us.



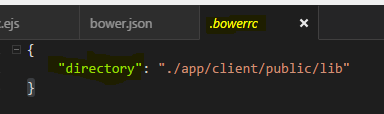
Again follow through with the setup, answering yes to them all will suffice. You will end up with something like so.



You may configure your bower with an additional file “.bowerrc”, here I am specifying what directory I wish for my dependencies to be installed into.

<https://bower.io/docs/config/>

I am installing my bower dependencies into my public folder in my app directory, as later we will use this folder to store all our files we wish to be made accessible in the browser.

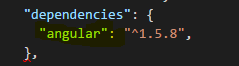


Now we are ready to start installing our client side dependencies.

Install Angular: “bower install angular –save”



This will result in adding angular to your bower.json, just like installing a dependacy with Npm (remember here NPM is for server side dependencies, while Bower is for Client side).



We must now add this angular.js file into index.egs, so when we launch our app it has angular loaded and we can use it then.

Open your index.ejs file and add this line of code. Insert this in the <head> section of our .egs file.



Now we are set up to do some angular work.

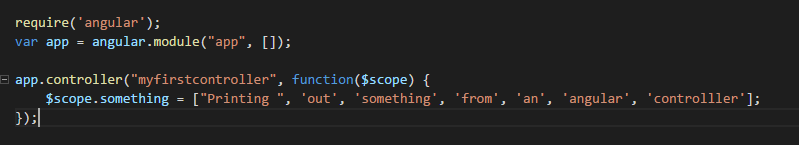
**Angular**

Creating a simple angular app and controller,

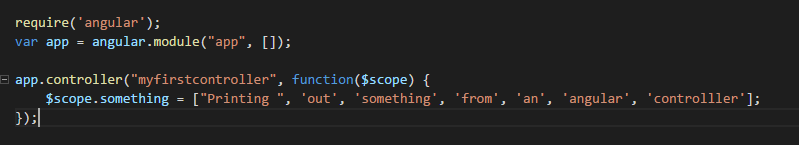
Creating the angular app module is simple and easy and can be done in 2 lines, although it doesn’t do anything, it is the entry point of the app.

Create an “app.js” file within the “app/client/ folder”

We must require angular, when this page is loaded into the browser, we have already specified in our “index.ejs” that angular must be loaded with it, but in this JavaScript file we are telling it to look for the angular file.



Next are going to create a simple controller, where we will hard code in some values and in the browser we will use angular to display this array.



Next go back to the “index.egs” file and now we can add this file to be loaded

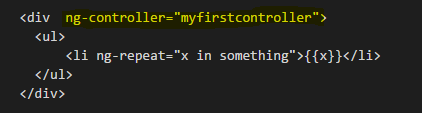


Now we have an index.js file that loads our app file and angular file we can now start doing things with angular now in the “.ejs” file

First we must declare what app we want, we called our app module “app” earlier now we are telling our body tag that everything inside it will be able to access this module.



In another tag we can call on our angular controller and inside the “**{{ }}**” we can write angular expressions. This will iterate through the array that the angular controller has initialized and print them out in the browser.



**Angular Components**

Angular uses a few different components and methods to keep the layout makeup of the app clean and organized. These components have distinct differences and uses.

**Directives**

Creating custom elements and html tags, these can make your html code extremely re-useable and extensible.

**Controllers**

Controllers are there to place any business logic of the app or part you are working on into.

**Services**

Allow to share information, variables and other functionality between controllers

**Factories**

Are there to generate your data, whether this be reading from a file, sending requests to server, database or whatever it may be.

**Filters**

Are used to generate an array’ , from array, example, creating a very simple filter to change all words into uppercase etc..

**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

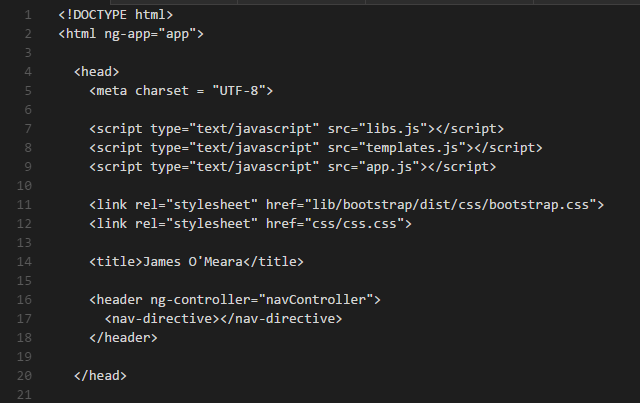
Directives are simple to create, can provide a wide range of functionality and re-use. Directive can include its own logic and can be passed in data and process it itself, e.g. pass in images with text: the directive could provide the layout for the image and surrounding text and display them in a grid.

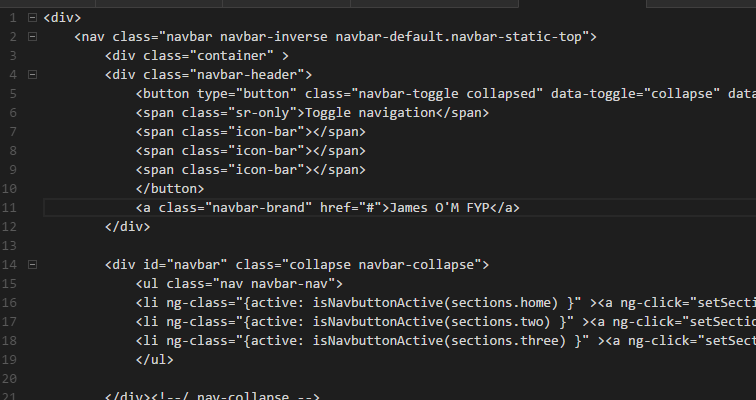
e.g.:

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 {**

**Controller**

Controllers add Business logic of your app, they should only control the logic for any one view, so if you have another view create a new controller to keep the logic separate. Controllers can be used in directives, or in any html elements. Controllers also allow access to $scope and provide access to variables and functions to be executed in real-time in the browser. This is how we give html and our web pages dynamic elements.

More info @ <https://docs.angularjs.org/guide/controller>

Inside html we use the angular braces to open and close when we are trying to do some angular work

“ **{{** “ & then “ **}}** “

**How to create a simple Controller**

Start of by creating an angular app if you haven’t already:

**var app = angular.module("app", [**

**]);**

Then create a controller, give it a name and pass in any variables in which you would like to use, for example $scope in this instance. We can pass in services and factories also; this will be shown later.

**app.controller("controllerName", function($scope) {**

**$scope.someVariable = ‘Hello world!’;**

**$scope.someFunction = function(){**

**return ‘hello world from a function’;**

**}**

**});**

And inside the HTML

**<html ng-app="app">**

**<body ng-controller="controllerName">**

**<p>Displaying text from controller {{ someVariable }}</p>**

**<p>Displaying text from controller {{ someFunction() }}</p>**

**</body>**

**</html>**

**Using a service or factory in a controller**

Simply add the service name or factory name into the input of the controller function. Then call the functions of the service as you would any regular object.

More info @ <http://fdietz.github.io/recipes-with-angular-js/controllers/sharing-code-between-controllers-using-services.html>

**app.controller("controllerName", function($scope, serviceName, factoryName) {**

**$scope.getSelectedTab = function(){**

**return serviceName.getTabSelected();**

**}**

**$scope.getSelectedTab = function(){**

**return factoryName.doSomething();**

**}**

**});**

**Service**

A service is there to allow your app to organize, share code, variables and functions.

“Essentially, factories are functions that *return the object*, while services are *constructor functions of the object* which are instantiated with the new keyword.”

Quote from: <http://www.codelord.net/2015/04/28/angularjs-whats-the-difference-between-factory-and-service/>

Here is where some ambiguity comes in, services can appear to be similar to factories, as in it will provide a service to the app. Yes a service can be used similary to a factory for a http request, but generally a services purpose is to provide a ‘service’, or a way of sharing code around your app, a factory is a ‘provider’ i.e. getting data from database or server or whatever it may be.

A service should be used for commutation between controllers, as controllers cannot directly access each other.

Useful links:

[**https://ilikekillnerds.com/2014/11/angularjs-call-controller-another-controller/**](https://ilikekillnerds.com/2014/11/angularjs-call-controller-another-controller/)

[**https://daveceddia.com/sharing-data-between-controllers-best-practice-use-a-service/**](https://daveceddia.com/sharing-data-between-controllers-best-practice-use-a-service/)

[**https://docs.angularjs.org/guide/services**](https://docs.angularjs.org/guide/services)

**Creating a Service**

Start of by creating an angular app if you haven’t already:

**var app = angular.module("app", [**

**]);**

And create the service definition like so:

**app.service("serviceName", function() {**

**return {**

**}**

**});**

But how do I add a function or variable to the service. As the service function returns an object we must return our service object or the methods we wish to make public

**app.service("sharedService", function() {**

**value = 1;**

**getValue = function(){**

**return value;**

**}**

**return {**

**getValue: getValue**

**}**

**});**

**Similarly, we could return an object, and this will hold all the service functions and variables that we are allowing to be accessed. (just a different way in JavaScript to work this out)**

**app.service("sharedService", function() {**

**service = {};**

**value = 1; //basically making this a private variable – see getter method below**

**service. getValue = function(){**

**return value;**

**}**

**return service;**

**});**

**Factory**

Factories, Services and providers are quite similar, but it’s the way in that they should be used is what makes them different. Factories are more flexible and powerful than a service. They can return almost anything.

“Essentially, factories are functions that *return the object*, while services are *constructor functions of the object* which are instantiated with the new keyword.”

Quote from: <http://www.codelord.net/2015/04/28/angularjs-whats-the-difference-between-factory-and-service/>

More info @ <https://docs.angularjs.org/guide/providers>

Great explanation here: <https://toddmotto.com/factory-versus-service> also

<http://www.codelord.net/2015/04/28/angularjs-whats-the-difference-between-factory-and-service/>

**How to create a factory**

Start of by creating an angular app if you haven’t already:

**var app = angular.module("app", [**

**]);**

And create the service definition like so:

**app.factory("factoryName", function($http) {**

**return {**

**}**

**});**

Similar to Services we can layout the factory, creating function and variable definitions and returning them. In here we will be adding a function to return a http request.

**app.factory("factoryName", function($http) {**

**var url = '/';**

**var factory = {};**

**factory.getSomething = function () {**

**return $http.get(url + 'getSomething')**

**};**

**return factory;**

**});**

**Filter**

Provide some sort of filtering functionality

E.g. turn all the words first letter into uppercase or filter by some term.