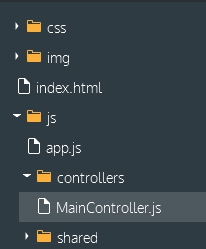
**Building an app with Angular**

Angular is a framework. It allows you to organize your front end code in a very clear and cool manner. We are gonna create the following folder:



We start off by creating an empty app in app.js. the 6 key words are:

* ***Module****: contains the different components of an Angular app*
* *Scope: is where the app will live (here, in <body>)*
* ***Controlle****r: will manage the app data. Here, we store a string in the property title*
* ***Directive****: helps us define where we attach our controllers (their scope)*
* *Expression: allows to display values on a page ( {{ title }} )*
* ***Service****: function that contacts the server for data and brings it back.*

The **module** basically says to the browser: “Ok, if you wanna know what data to display and how to display it, you should ask these guys, they are called controllers. I might direct you to different controllers depending the page the user is on”. The **controller** says “here is the data I want you to use in the view, and here is how I wanna organize it and call it. The **directive** says “here is how I want you to display data when I am called by my tag. The **service** says: “here is where you should fetch the data. Sometimes I will tell you to fetch only one part of it. You’ll know”.

Var app = angular.module(“myApp”, []);

Then we open our index.html and modify the body:

<body ng-app="myApp">

Then we go to MainController.js:

app.controller(‘Main Controller’, [‘Scope’, function($scope){

$scope.title = ‘Top seller in Books’;

}]);

After this, we go back to index.html and modify accordingly the “main” balise:

<div class="main" ng-controller="MainController">

Our app is set up. Finally, we call it into the html so that our h1 displays our title:

<h1> {{ title }} </h1>

What did we do?

1. In **app.js**, we created a new *module* named MyApp. A *module* contains the different components of an AngularJS app.
2. Then, in **index.html** we added <body ng-app="myApp">. The ng-app is called a *directive*. It tells AngularJS that the MyApp module will live within the <body> element, termed the application's *scope*. In other words, we used the ng-app directive to define the application scope.
3. In **MainController.js** we created a new *controller* named MainController. A *controller* manages the app's data. Here we use the property title to store a string, and attach it to $scope.
4. Then, in **index.html**, we added <div class="main" ng-controller="MainController">. Like ng-app, ng-controller is a *directive* that defines the controller scope. This means that properties attached to $scope in MainController become available to use within <div class="main">.
5. Inside <div class="main"> we accessed $scope.title using {{ title }}. This is called an *expression*. Expressions are used to display values on the page.
6. The value of title showed up when we viewed the app in the browser.

Now. You can store strings in the controllers, but also object:

$scope.product = {

name: "The book of Trees",

price: 19

};

And then, we are gonna access the name property of the object in html:

<p> {{ product.name }} </p>

And then, we are gonna specify the price, but we are gonna use a built-in filter to display it in a currency format:

<p> {{ product.price | currency }}

Now. You can create a property which contains several chunks of data, even a whole databse! Here we create a property that contains two books:

$scope.products = [

{

name: 'The Book of Trees',

price: 19,

pubdate: new Date('2014','03','08'),

cover: 'img/the-book-of-trees.jpg'

},

{

name: 'Program or be Programmed',

price: 8,

pubdate: new Date('2013','08','01'),

cover: 'img/program-or-be-programmed.jpg'

}

Then, the HTML will automatically generate as many blocks as there is books with the same properties (just like with PHP):

<div ng-repeat="product in products" class="col-md-6">

<div class="thumbnail">

<img ng-src="{{product.cover}}">

<p class="title">{{ product.name }}</p>

<p class="price">{{ product.price | currency }}</p>

<p class="date">{{ product.pubdate | date }}</p>

</div>

</div> ]

This is thanks to the directive: ng-repeat = “product in products”. Let’s talk about directives for a second. We also use ng-src in the <img> element.

Directives bind behavior to HTML elements. When the app runs, AngularJS walks through each HTML element looking for directives. When it finds one, AngularJS triggers that behavior (like attaching a scope or looping through an array).

We are now going to set up a counter that is going to count the likes of each product when we click that button. We start off by adding a property “likes: 0 “ to each product and set the counter to 0. We then add a new property to the controller, which we make equal to a function:

$scope.plusOne = function(index){

$scope.products[index].likes += 1;

We then add a new element in html that displays {{products.likes}}

And finally, add some directives to the element:

<div class="rating">

<p class="likes" ng-click="plusOne($index)"> +{{product.likes}} </p>

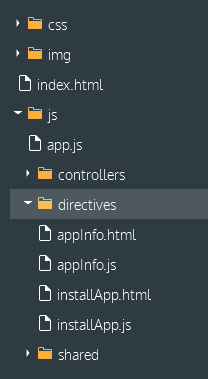
</div>

Ng-click is a directive that activates a certain function when the element is clicked. [index] serves to point out to the browser that he has to act on only the book that corresponds to the button that has been pushed, not all of them.

**Creating HTML elements:**

We have previously inserted data (strings, objects) into our html file via AngularJS. Now, let’s take it a step further. We are gonna insert whole chunks of html code into the HTML file (elements, data) via AngularJS. Watch and learn, baby.

First we are gonna create more folders:



Then, in the js/directives/appInfo.js, we are gonna create our new directive (a buffed-up version of classical directives where we can precisely determine how we want to display the date:

app.directive(‘appInfo’, function(){

return{

restrict: ‘E’,

scope: {

info: ‘=’

},

templateUrl = ‘js/directives/appInfo.html

};

});

Note on the scope:   
Chaque propriété définie dans le scope isolé va être liée par un binding d'un certain type à un attribut de l'élément HTML.  
  
On va avoir trois utilisations différentes pour les attributs de l'élément :

* des attributs de **type texte**, qui peuvent éventuellement contenir une ou plusieurs expressions entre doubles accolades {{...}} (comme la directive *ngSrc*)
* des attributs de **type expression** qui servent à faire un binding sur la **valeur** de l'expression (comme la directive *ngBind*)
* des attributs de **type expression** qui servent à déclencher une **action** (comme la directive *ngClick*)

Ces trois cas d'utilisation correspondent aux trois symboles utilisables dans la définition des propriétés du scope isolé :

* '**@**' pour un **attribut texte**
* '**=**' pour une **expression valeur**
* '**&**' pour une **expression action**

Then, we determine how we want the chunk of HTML to be. in appInfo.html, we write:

<img class="icon" ng-src="{{ info.icon }}">

<h2 class="title">{{ info.title }}</h2>

chunk #1

<p class="developer">{{ info.developer }}</p>

<p class="price">{{ info.price | currency }}</p>

Eventually, we use our new directive to integrate it into our html:

<div class=”app-chunk”>

<app-info info=”app number1”></app-info>

</div>

Done! now the app-info element will display chunk#1 when the page is loaded (chunk#1 adapted to the first element of the data in the controller, ie app number 1, that is). Be sure to notice the importance of the attribute info. We set it up in appInfo.js, and tell the browser to look for an attribute named like that in our new element. Then, we switch to appInfo.html and what do we see?

{{info.title}}

the value of the attribute info in each of the elements is translated to the MainController.js and chznged into each of the data elements (here, the different apps) stored in there. It does an array swipe, basically!

Now, let’s take it a step further. we are gonna mix in traditional ng directives and our new one.

We create an array ($scope.apps) in our controller that lists all our apps.

We are gonna need to access it through our app-Info element to give the info attribute something to chew on. Here is how it’s done:

<div class="card" ng-repeat="app in apps">

<app-info info="app"></app-info>

</div>

equals

<div class="card ng-scope" ng-repeat="app in apps">

<app-info class="ng-isolate-scope" info="app"><img src="img/move.jpg" class="icon" ng-src="img/move.jpg">

<h2 class="title ng-binding">MOVE</h2>

<p class="developer ng-binding">MOVE, Inc.</p>

<p class="price ng-binding">$0.99</p><script type="text/javascript" src="http://external-production.codecademy.com/assets/relay.js"></script></app-info>

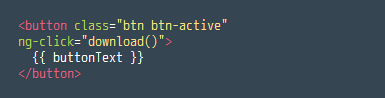
</div>

Now something else. we add a property “link” to our directive:



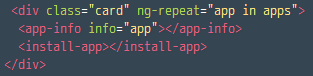
1. The link is used to create interactive directives that respond to user actions. scope refers to the directive's scope. Any new properties attached to $scope will become available to use in the directive's template.
2. element refers to the directive's HTML element.
3. attrs contains the element's attributes.

We complete this by writing in our appInstall.html:

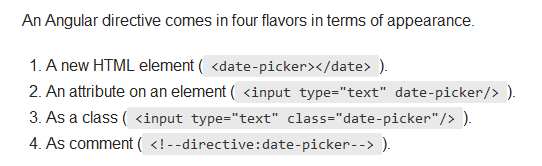


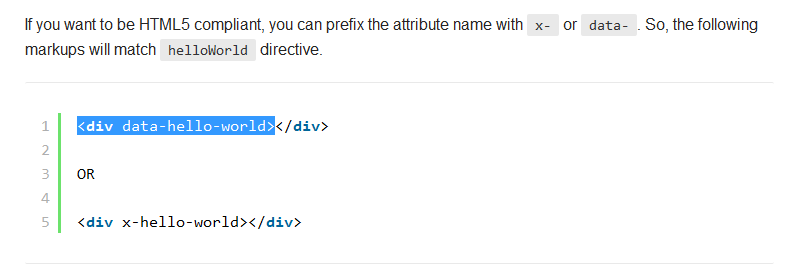
On click, thx to the ng-click directive, it will launch the download property of the appInstall.js

Finally we get this:



a button that toggles install/uninstall on click.





*A word of conclusion on directives:*

Directives are a powerful way to create self-contained, interactive components. Unlike jQuery which adds interactivity as a layer on top of HTML, AngularJS treats interactivity as a native component of HTML.

**Reading data from a server (the merging of Angular and PHP)**

The main controller is gonna change because now, we are gonna make him fetch the data in our JSON or our database:

app.controller('MainController', ['$scope','forecast',function($scope,forecast){

forecast.success(function(data){

$scope.fiveDay = data;

})

}]);

this data we see inside the controller corresponds to a service called “forecast,”, which we also specified in the module. The service looks like this:

app.factory('forecast', ['$http', function($http) {

return $http.get('http://s3.amazonaws.com/codecademy-content/courses/ltp4/forecast-api/forecast.json')

.success(function(data) {

return data;

})

.error(function(err) {

return err;

});

}]);

Main controller tells the data it received from the service to be called fiveDay. Then, we integrate it into our view, and specify we talk about the data called fiveDay

<div class="forecast" ng-repeat="day in fiveDay.days">

<div class="day row">

<div class="weekday col-xs-4">

{{day.datetime | weekday}}

</div>

<!-- icon -->

<div class="weather col-xs-3">

<img ng-src="{{day.icon}}"/>

</div>

<div class="col-xs-1"></div>

<!-- high -->

<div class="high col-xs-2">

{{day.high}}

</div>

<!-- low -->

<div class="low col-xs-2">

{{day.low}}

</div>

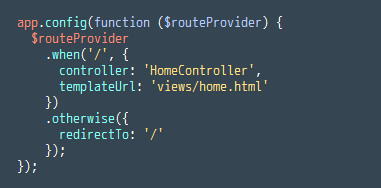
</div>

</div>

**Routing**

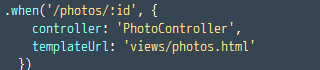
All this is fine if our website only has one page, but what if it is has a lot? it would get messy really quick trying to fit everything into one app manually. That’s why we are gonna build up our app.js in order to take into account the address where the user currently is and redirect the browser to a specific controller.

1. In **app.js** inside the app.config() method, we use Angular's $routeProvider to define the application routes.
2. We used .when() to map the URL / to to the controller HomeController and the template home.html. The HomeController uses the service **js/services/photos.js** to fetch the array of all photos from <http://s3.amazonaws.com/codecademy-content/courses/ltp4/photos-api/photos.json> and stores it into $scope.photos. The home.html uses ng-repeat to loop through each item in the photos array and display each photo.
3. Otherwise if a user accidentally visits a URL other than /, we just redirect to / using .otherwise().
4. Now when a user visits /, a view will be constructed by injecting home.html into the <div ng-view></div> in **index.html**.



Note that this app.config is defined OUTSIDE of the angular.module method. Jeez.

If we wanna use dynamic protocols, like GET[id]’s, we can tell it to the app simply by displaying :id in the address.



**Calendar:**

We plug in the routing addon by putting ngRoute as a dependency in the module:

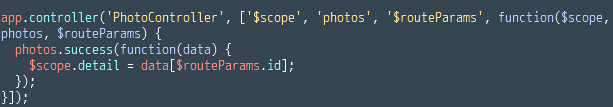
The controller looks like this. We can see it now takes into account the route Parameters to fetch the relevant id ($routeParams.id).



Let’s stop on the difference between fetching all the data and a dynamic set, part of the data with the controllers. Here is the main controller:



And here is the controller for a dynamic page meant to display only the photo we clicked on:



The difference is that one $routeParams that focuses the browser on one photo only.

*Notice that when you click on links, the app doesn't do a full reload. Only the part of the view specified by <div ng-view></div> changes.*

<https://code.angularjs.org/1.2.28/angular-route.min.js>"

* You need to include this additional JavaScript file in order to add routing to your app because it does not come with the AngularJS library by default.