**INITIALIZE**

1. Let's get started by making a simple AngularJS app. We'll explain each step in the next exercise.  
   In **app.js**, type in the contents exactly as you see here:

var app = angular.module("myApp", []);

1. Open up **index.html**. Modify the <body> tag so it looks like this:

<body ng-app="myApp">

1. Open up **js/controllers/MainController.js**. Type in the contents exactly as you see here:

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

$scope.title = 'Top Sellers in Books';

}]);

1. Go to **index.html**. Modify the <div class="main">tag so it looks like this:

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

1. In **index.html** inside <div class="main">, modify the <h1> element so it looks like this:

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

View the AngularJS app in the browser by visiting [http://localhost:8000](http://localhost:8000/). The "Top Sellers in Books" content appears as the heading of the page.

 In **app.js**, we created a new *module* named myApp. A *module* contains the different components of an AngularJS app.

 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.

 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.

 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">.

 Inside <div class="main"> we accessed $scope.titleusing {{ title }}. This is called an *expression*. Expressions are used to display values on the page.

 The value of title showed up when we viewed the app in the browser.

**OBJECT**

1. Let's add more data to the controller and display them in the view. In the controller, attach another property to $scope named product. Set it equal to an object with the following properties:

{ name: 'The Book of Trees', price: 19 }

1. Then, in **index.html** inside <p class="title">, access the product's name with product.name and display it using an expression.
2. In <p class="price">, access and display the product's price

**FILTER**

1. Currently the product price shows up as a number. It would be better to format it as a currency. Rather than change the data in the controller, let's use an AngularJS *filter* to format the data in the view.

In **index.html** in <p class="price">, change the expression to look like this:

{{ product.price | currency }}

We'll explain how this works in the next exercise.

Great! The product price changed from a number to a formatted currency. How does it work?

1. AngularJS gets the value of product.price.
2. It sends this number into the currency filter. The pipe symbol (|) takes the output on the left and "pipes" it to the right.
3. The filter outputs a formatted currency with the dollar sign and the correct decimal places.

In this way, filters help to separate the content in the controller from its presentation in the view.

1. AngularJS comes with [a few more built-in filters](https://docs.angularjs.org/api/ng/filter). Let's use two more.

In **MainController.js** inside $scope.product, add a third property named pubdate:

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

1. In **index.html** inside <p class="date">, display the product's pubdate.
2. Format the product's pubdate by piping it to the datefilter.
3. Format the product's name by piping it to the uppercase filter.

**LIST**

1. Let's add more data to the controller and display them in the view.

In the controller, delete the $scope.product object.

1. Attach a new property to $scope named products. Set it equal to an array of objects. Type in the contents exactly as you see here:

[

{

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'

}

]

1. In the view inside <div class="main">, delete the <div class="col-md-6"> element.  
   Replace it with this HTML. Type in the contents exactly as you see here:

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

<div class="thumbnail">

<img src="img/the-book-of-trees.jpg">

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

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

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

</div>

</div>

You'll see that both products have the same cover image. Let's fix this bug in the next exercise. Click Next to continue.

1. The problem now is that both products have the same image. Let's fix this.

In the view inside <div class="col-md-6">, replace

<img src="img/the-book-of-trees.jpg">

with

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

The ng-src is a directive that sets the <img>element's src to a property in the controller.

**DYNAMIC DATA**

1. In the controller in the $scope.products array, add a new property named likes to each element. Set all likes properties to 0.
2. In the view under <p class="date">, type in a rating element:

<div class="rating">

<p class="likes">+ </p>

</div>

Inside <p class="likes">, display a product's likesusing an expression.

1. Back in the controller after $scope.products, attach a new property to $scope named plusOne. Set it equal to function. Type in the contents exactly as you see here:

function(index) {

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

};

1. In the view modify <p class="likes"> to look like this:

<p class="likes" ng-click="plusOne($index)">

View the AngularJS app in the browser. Click on the +0 in each product tile.

Learn how to use directives to make standalone UI components.

Here's an AngularJS app for a mobile app store:

* In the controller **MainController.js**, there are three objects $scope.move, $scope.shutterbugg, and $scope.gameboard that each contain info about an app, like its title and price.
* In the view **index.html** in the .main section, each app is displayed inside a .card div.

But looking at the view, the same code is written over and over again to display each app. This is repetitive and error-prone. Let's fix this.

1. In the new file **js/directives/appInfo.js**, type in this code:

app.directive('appInfo', function() {

return

{

restrict: 'E',

scope:

{

info: '='

},

templateUrl: 'js/directives/appInfo.html'

};

});

1. Include this new JavaScript file in **index.html** in line 48 as a <script> element.
2. In the new file **js/directives/appInfo.html**. Type in this HTML to display an app's info:

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

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

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

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

1. In **index.html**, replace the contents of the first .carddiv with the new <app-info> element:

<div class="card"> <app-info info="move"></app-info> </div>

1. Do the same for the second and third .card divs. Replace their contents with <app-info info="shutterbugg"></app-info> and <app-info info="gameboard"></app-info>

View the AngularJS app in the browser by typing [http://localhost:8000](http://localhost:8000/).

First in **js/directives/appInfo.js**, we made a new *directive*. We used app.directive to create a new directive named 'appInfo'. It returns an object with three options:

1. restrict specifies how the directive will be used in the view. The 'E' means it will be used as a new HTML element.
2. scope specifies that we will pass information into this directive through an attribute named info. The = tells the directive to look for an attribute named info in the <app-info> element, like this:

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

The data in info becomes available to use in the template given by templateURL.

1. templateUrl specifies the HTML to use in order to display the data in scope.info. Here we use the HTML in **js/directives/appInfo.html**.

Looking at **js/directives/appInfo.html**, we define the HTML to display details about an app, like its title and price. We use expressions and filters to display the data.

Then in **index.html** we use the new directive as the HTML element <app-info>. We pass in objects from the controller's scope ($scope.shutterbugg) into the <app-info> element's info attribute so that it displays.

1. In the controller, create a new property $scope.apps. Set it equal to an array of objects:

[

{

icon: 'img/move.jpg',

title: 'MOVE',

developer: 'MOVE, Inc.',

price: 0.99

},

{

icon: 'img/shutterbugg.jpg',

title: 'Shutterbugg',

developer: 'Chico Dusty',

price: 2.99

}

]

1. Add two more objects to the array describing your favorite apps. Make sure to define the four properties for each app.
2. In the view, use ng-repeat to loop through $scope.apps and display each element. To do this, add ng-repeat to a <div class="card">, and then use the custom directive <app-info> to display each element.

<div class=”card” ng-repeat=”item in apps”>

<app-info info=”item”></app-info>

</div>

1. In the new file **js/directives/installApp.js**, create a new directive named installApp. Refer to the appInfodirective for an example:

* use app.directive to create a new directive named installApp
* use the restrict option to create a new Element
* set the scope option to an empty object {}
* use the templateUrl option to tell this directive to use the **js/directives/installApp.html** file

app.directive(‘installApp’,function(){

return{

restrict:’E’,

scope: {},

templateUrl: ‘js/directives/installApp.html’

}

});

1. In the new file **js/directives/installApp.js**, create a new directive named installApp. Refer to the appInfodirective for an example:

* use app.directive to create a new directive named installApp
* use the restrict option to create a new Element
* set the scope option to an empty object {}
* use the templateUrl option to tell this directive to use the **js/directives/installApp.html** file

1. Include this new JavaScript file in **index.html** as a <script> element.
2. In the installApp directive, add a fourth option named link, and type in the following function:

function(scope, element, attrs)

{

scope.buttonText = "Install",

scope.installed = false,

scope.download = function()

{

element.toggleClass('btn-active');

if(scope.installed)

{

scope.buttonText = "Install";

scope.installed = false;

}

else

{

scope.buttonText = "Uninstall";

scope.installed = true;

}

}

}

app.directive('installApp',function(){

return{

restrict: 'E',

scope: {},

templateUrl: 'js/directives/installApp.html',

link: function(scope, element, attrs) {

scope.buttonText = "Install",

scope.installed = false,

scope.download = function() {

element.toggleClass('btn-active');

if(scope.installed) {

scope.buttonText = "Install";

scope.installed = false;

} else {

scope.buttonText = "Uninstall";

scope.installed = true;

}

}

}

};

});

* 1. Next, write the directive's template:

In the new file **js/directives/installApp.html**. Type in the following HTML:

<button class="btn btn-active" ng-click="download()"> {{ buttonText }} </button>

1. Finally, use the new directive in the view:

In **index.html**, add the new <install-app> element inside the .card div under the <app-info> element.

1. View the AngularJS app in the browser by typing [http://localhost:8000](http://localhost:8000/).

Use services to communicate with a server.

1. In the browser frame on the right, visit <https://s3.amazonaws.com/codecademy-content/courses/ltp4/forecast-api/forecast.json>. It's a JSON object containing a city\_name and an array days containing weather data for the next five days.
2. Create a service named forecast that fetches the weather data from the server. In the new file **js/services/forecast.js**. Type in this code exactly as you see here:

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

{

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

.success(function(data) {

return data;

})

.error(function(err) {

return err;

});

}]);

1. Include **js/services/forecast.js** in **index.html** in line 62 as a new <script> element.
2. In the controller, modify MainController by passing in the forecast service, like this:

app.controller('MainController', ['$scope', 'forecast', function($scope, forecast) { // ... }]);

1. Inside MainController, use the forecast service to save the weather data into $scope.fiveDay, like this:

forecast.success(function(data) { $scope.fiveDay = data; });

 First in **js/services/forecast.js**, we made a new *service*. We used app.factory to create a new service named forecast

 The forecast service needs to use AngularJS's built-in $http to fetch JSON from the server. Therefore, we add $http to the forecast service as a dependency, like this:

['$http', function($http) { // ... }]

Now $http is available to use inside forecast.

 Then, inside forecast, we use $http to construct an HTTP GET request for the weather data. If the request succeeds, the weather data is returned; otherwise the error info is returned.

 Next in the controller, we used the forecast service to fetch data from the server. First we added forecast into MainController as a dependency so that it's available to use. Then within the controller we used forecast to asynchronously fetch the weather data from the server and store it into $scope.fiveDay

 As before, any properties attached to $scope become available to use in the view. This means in **index.html**, we can display the city\_name using an expression as done before.

1. In the browser, visit <https://s3.amazonaws.com/codecademy-content/courses/ltp4/forecast-api/forecast.json>. Looking at the format of the data in the days array, each day has four pieces of data - datetime, icon, high, and low.
2. Visit [http://localhost:8000](http://localhost:8000/) to view the AngularJS app. Let's finish the view so that it displays the weather for each day.

Notice in the view, we're using ng-repeat to loop through each item in the days array. Inside this loop, display a day's four pieces of data.

Remember to use ng-src to display an image.

1. Use the date filter to format the datetime

<div class=”forecast” ng-repeat=”day in fiveDay.dats==ys”>

<div class=”day row”>

<div class=”weekday col-xs-4”>  
 {{day.datetime | date}}

</div>

<div class=”weather col-xs-3>

<img ng-src=”{{day.icon}}”>

</div>

<div class=”col-xs-1”></div>

<div class=”high col-xs-2”>

{{day.high}}

</div>

<div class=”low col-xs-2”>

{{day.low}}

</div>

</div>

</div<

Add routes to build powerful single-page applications.

1. In **index.html** under the .header section, type in the code exactly as you see here:

<div ng-view></div>

**<div class="header">**

**<div ng-view></div>**

**<div class="container">**

**<a href="/#/">**

**<img src="img/logo.svg" width="80" height="80"> &#12501; &#65387; &#12488; &#12501; &#65387; &#12488;**

**</a>**

**</div>**

**</div>**

1. View the AngularJS app in the browser by typing <http://localhost:8000/>.
2. In **app.js** under the angular.module, type in this code:

app.config(function ($routeProvider)

{

$routeProvider .when('/',

{

controller: 'HomeController',

templateUrl: 'views/home.html'

})

.otherwise({

redirectTo: '/'

});

});

 In **app.js** inside the app.config() method, we use Angular's $routeProvider to define the application routes.

 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 <https://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.

 Otherwise if a user accidentally visits a URL other than /, we just redirect to / using .otherwise().

 Now when a user visits /, a view will be constructed by injecting home.html into the <div ng-view></div> in **index.html**.

1. In **app.js** between the .when() and .otherwise(), add another .when() that maps the URL /photos/:id to the controller PhotoController and the template **views/photo.html**.
2. Visit <http://localhost:8000/> and click on a photo. A detail page about that photo should appear
3. In **app.js**, we mapped a URL to PhotoController and photo.html. We added a variable part named id to the URL, like this: /photos/:id.
4. In **PhotoController**, we used Angular's $routeParams to retrieve id from the URL by using $routeParams.id. Notice that we injected both $routeParams and the photos service into the PhotoController dependency array to make them available to use inside the controller.
5. Then to fetch an individual photo, we first used the photos service to fetch the array of photos from the server, and then used $routeParams.id to access the specific photo by its index.
6. As before, any properties attached to $scope become available to use in the view. This means in **photo.html**, we can display the photo's detail using expressions as done before.

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.

1. Finish the template so that it displays a photo's details.

First, in the browser, visit <https://s3.amazonaws.com/codecademy-content/courses/ltp4/photos-api/photos.json>. Looking at the format of the data in the array, each photo has six pieces of data - title, author, url, pubdate, upvotes, and views.

1. In the template **photo.html**, display a photo's remaining five pieces of data. Then visit <http://localhost:8000/> in the browser and click on a photo. A detail page about that photo should appear.
2. Use the number filter to format the views and upvotes.
3. Use the date filter to format the pubdate.
4. View the result in the browser. The photo's views, upvotes, and date should now be formatted.

<div class="photo">

<div class="container">

<img ng-src="{{ detail.url }}">

<h2 class="photo-title">{{ detail.title }}</h2>

<p class="photo-author"> {{ detail.author }}</p>

<p class="photo-views"> {{ detail.views | number }}</p>

<p class="photo-upvotes"> {{ detail.upvotes | number }}</p>

<p class="photo-pubdate"> {{ detail.pubdate | date }}</p>

</div>

</div>