Developing Web Applications Using AngularJS

Lab Guide

Version 1.7

presented by

Hands On Technology Transfer, Inc.

Chelmsford, Massachusetts, (USA)

Table of Contents

Module 1 Exercises Introduction to AngularJS **5**
Lab Setup 5
Exercise 1 – Exploring the Parts of an AngularJS Application 5
Module 2 Exercises Data Binding **10**
Exercise 1 – Working with AngularJS Data Binding and Expressions 10
Exercise 2 – Tracking User Input with AngularJS Data Binding (Optional) 14
Module 3 Exercises Working with Filters **18**
Exercise 1 – Experimenting with AngularJS Filters 18
Exercise 2 – Creating an Employee Search Feature with AngularJS Filters (Optional) 21
Module 4 Exercises Directives **24**
Exercise 1 – Using AngularJS Directives 24
Exercise 2 – Experimenting with AngularJS Directives (Optional) 27
Module 5 Exercises AngularJS Modules **31**
Exercise 1 – Working with Modules and Dependency Injection 31
Module 6 Exercises Controllers **35**
Exercise 1 – Working with Controllers 35
Exercise 2 – Creating a Contact Manager Application 39
Exercise 3 – Binding Multiple Controllers to a View (Optional) 43
Module 7 Exercises Single Page Applications **48**
Exercise 1 – Creating a Single Page Application 48
Module 8 Exercises Custom Services **52**
Exercise 1 – Creating a Simple Service 52
Exercise 2 – Moving Controller Code to a Service 54
Module 9 Exercises Using AngularJS Built-in Services **57**
Exercise 1 – Lab Setup 57
Exercise 2 – Using the `$http` Service 58
Exercise 3 – Web Service Calls Using `$resource` 62
Module 10 Exercises Defining Custom Directives **67**
Exercise 1 – Registering Custom Directives 67
Module 11 Exercises Defining Custom Filters **71**
Exercise 1 – Building Custom Filters 71
Exercise 2 – Defining a Custom Search Filter (Optional) 74
Module 12 Exercises Integrating Forms with AngularJS **77**
Exercise 1 – Enhancing a Standard Form 77
Exercise 2 – Creating a Simple ToDo List in AngularJS 82
Exercise 3 – Auto Loan Calculator (Optional) 85
Module 13 Exercises Form Validation with AngularJS **89**
Exercise 1 – Form Validation 89
Exercise 2 – Use the `ngMessages` Module for Error Reporting 94
Module 14 Exercises Creating Animations in AngularJS **101**
Exercise 1 – Creating an Animated Notification Bar 101
Exercise 2 – Creating an Animated Image Gallery 104
Module 15 Exercises Using UI Bootstrap **110**
Exercise 1 – Creating a Dynamic Accordion 110
Module 16 Exercises Additional Angular UI Directives **113**
Exercise 1 – Creating a Calendar with Tooltip Display 113
Module 17 Exercises Working with AngularStrap **117**
Exercise 1 – Creating a Dynamic Modal 117
Appendix A Configuring a Virtual Directory **121**
Exercise 1 – Configuring a Virtual Directory 121

Module 1 Exercises
Introduction to AngularJS

AngularJS is a client-side JavaScript framework that is primarily used to build single page applications. AngularJS makes it easy to build interactive, modern web applications by offering features such as dependency injection, data binding, routing, separation of application logic from the data models and views, as well as Ajax services.

Lab Setup

1. Unzip the `HOTT_AngJS.zip` file from your lab CD to the root of the `C:\` directory.

2. This should create a `HOTT_AngJS` directory that consists of several module folders. Within each module folder, there are several folders:

   - `Examples` (where you'll find the student guide examples)
   - `Starters` (where you'll find a clean copy of the starter files for the lab exercises)
   - `Labs` (this is where you will save your work)
   - `Solutions` (where you will find a working solution to the lab exercise)

Several of the pages that we will be working with throughout the course use Ajax to load pages and content asynchronously. Due to the Cross Origin Policy restriction in several browsers (Firefox is the exception), you must create a virtual directory in order to use AngularJS features such as routing and Ajax requests. Once you have created the virtual directory, you should use this approach to view your lab files.

3. Using the steps detailed in Appendix A of this lab guide, create a virtual directory to point to your `HOTT_AngJS` folder.

Exercise 1 – Exploring the Parts of an AngularJS Application

In this exercise, you will explore an AngularJS application that contains several of the things that make AngularJS a great framework for developing web applications. This single page application uses data binding, controllers, filters, services, routing and animations.

1. Use Windows Explorer to copy the `AngularJSApp` directory from the `Mod01/Starters` folder into the `Mod01/Labs` folder.

Examining the Code

In the following steps, we will be exploring some of the components of a single page application using AngularJS.

2. Use Windows Explorer to examine the `scripts` folder. Notice the presence of the `angular.min.js` file. This is a minified version of the core AngularJS file. It must be referenced on any HTML page that uses AngularJS.

There are also a pair of other `js` files that contain `angular` in the name. The `angular-animate.min.js` script provides an application with the ability to add animations. The `angular-route.min.js` script is used to add routing functionality which is crucial in building a Single Page Application (SPA).

3. Open the `app.js` file in your editor.

It contains the registration of an AngularJS module. Notice the array of strings that contains `'ngRoute'` and `'ngAnimate'`. We are defining the dependencies of our application using Dependency Injection. That means that in order for our application to run, we need to access code that is defined in a pair of additional modules.

The file also contains a configuration block to specify several routes which will be used in the application. A route enables you to define different URLs for different content in the application. Each route connects a path with a JavaScript object that defines a controller and a file containing the HTML markup to use when a route is chosen.

4. Next, open the `scripts/views/employees.html` file in your editor. This file contains an HTML snippet that will be loaded into a shell page (`index.html`) when the `/employees` route is selected.

5. Locate the `<tr>` element that has an `ng-repeat` attribute. This attribute (directive) is used to iterate through the items in a collection and render a template.

In this case, a row is rendered for each employee. The row contains five cells that each use a data binding expression to output data. Here, we are outputting the properties of an employee. Notice the cells for Hire Date and Salary. We are specifying a filter to format those properties when the page is rendered.

6. Next, open the `scripts/controllers/EmployeeController.js` file in your editor. When the `/employees` route is selected, the `EmployeeController`'s constructor function is invoked. Several variables are defined on a `$scope` object. The `$scope` is a special AngularJS object that stores data and behaviors that are available to a view.

7. Next, open the `scripts/services/services.js` file in your editor. Within it, there is a pair of custom services which are used to provide data to a controller. One service (created using the `factory()` method) returns an array of customer objects. The other service (created using the `service()` method) returns an array of employee objects.

8. Locate the line with `this.getEmployees`. The line is defining a method that returns an array of employee objects. Each employee has properties of `name`, `phone`, `city`, `hireDate` and `salary`. Add a new employee to the array.

NOTE: In this example, the data is loaded from an array, but in a real-world application, it would probably be loaded from a remote server.

9. Now open the `index.html` file in your editor. The `<html>` element contains a special attribute named `ng-app`. In AngularJS, we call this a directive. Directives are used to teach a browser new syntax and provide new behaviors to HTML. `ngApp` is used to bootstrap or initialize an application.

10. Next, look at the links defined within the navigation bar. The `href` attribute of each link specifies a route that was configured in `app.js`. When one of the links is clicked, AngularJS knows which content to load based on the routes that we configured.

11. Next, locate the `<ng-view>` element. This is an AngularJS directive that is used to indicate where the HTML markup of the route's template should be rendered.

12. Now view the code at the bottom of the page, this is where the AngularJS files are referenced. You should reference AngularJS at the bottom of the page so the HTML has a chance to load before the scripts execute. This is not required.

NOTE: Although the components (controllers and services) could be defined within a single file, we have chosen to split them into separate folders and files for organization. You can structure your AngularJS applications as you see fit.

Testing the Application

In the following steps, we will be exploring some of the features of a single page application built using AngularJS.

13. Browse the `index.html` page by right-clicking on the file within IIS and choosing Browse.

14. Before you click on one of the hyperlinks that appear in the navigation bar, open the browser's Developer Tools and examine the HTML.

    - Within the `<html>` element, you should see an `ng-app` attribute (called a directive) and a class of `ng-scope`.

The `ngApp` directive is used to Bootstrap (or initialize) an AngularJS application. The `ng-scope` class is automatically added by AngularJS to elements where scopes are attached. In AngularJS, scope is an object that refers to the application model. Scope is used to store the data and behaviors that should be available to the view.

15. Click the Employees link on the navigation bar. Did you notice that a table of employees was displayed using an animation?

NOTE: If you do not see the employee information, make sure you are browsing using the virtual directory!

AngularJS has the ability to display animations using CSS3 or JavaScript animations. In addition, pay attention to the URL in the address bar.

16. Use the Developer Tools window to examine the HTML. Locate the `ng-view` element and examine it to see the HTML that was loaded from the route's template.

17. Now click on the Network tab (still under Developer Tools). Click the filename (`employees.html`) and then the Resources tab if using Firefox and the Preview tab if using Chrome.

You are viewing the HTML that was loaded as part of the request. Instead of loading the HTML each time a page is requested, with single page applications, the content that will be used by all pages is loaded when the shell page loads. Then, when a link is clicked specifying a view, the browser loads just the HTML required for the view.

18. Repeat the same steps for the Customers link. Pay attention to the URL in the address bar.

19. Close the Developer Tools.

20. Click to navigate between the different pages and pay attention to the URL displayed in the address bar.

21. Click to revisit the Employees page, click the column headers of the table. What happens?

The data is being sorted using the `orderBy` filter. AngularJS contains several filters that can be used to format data. The Hire Date and Salary columns are also being filtered (as a date and currency).

22. Now, click over to the Customers page. The Customers page contains hyperlinks to view the orders for each customer. Click one of the Orders links and pay attention to the address bar. Do you notice a number being passed in the URL? This is called a route parameter.

Route parameters are useful when you need to change the view's content dynamically. For example, in this application, we want to show the orders for a particular customer. If we click on a different link, the orders for a different customer are displayed.

23. Close the browser.

Now that we've seen some of the components of an AngularJS application, it is time to start talking about them in detail.

Module 2 Exercises
Data Binding

One of the most powerful features of AngularJS is data binding. AngularJS uses two-way data binding to automatically sync a view and its data. A view is a projection of the data model at all times. When the model changes, the view reflects the change, and vice versa. AngularJS uses the scope object to refer to the application model. Scope serves as an execution context for expressions. AngularJS expressions are JavaScript-like code snippets that are placed in interpolation bindings such as `{{variable}}`. They can also be used in directive attributes, such as `ng-click="getTotal()"`.

Exercise 1 – Working with AngularJS Data Binding and Expressions

In this exercise, you will use AngularJS data binding to display data in a view. At first, you will initialize and bind to a value defined in the HTML markup. Then, you will define an AngularJS controller and initialize several values on the `$scope` object. By associating the controller with the view, you will then be able to access the scope's data. You need to use data binding expressions to render the data on the view.

1. Use Windows Explorer to copy the `ACMETraders` directory from the `Mod02/Starters` directory into the `Mod02/Labs` folder.

2. Using Windows Explorer, examine the structure of the `ACMETraders` directory.

You will notice there are three folders (`css`, `images` and `scripts`) as well as an `index.html` file already present. The `css` folder contains the `bootstrap.min.css` file. The images folder contains a single image file. The `scripts` folder already contains the minified version of the AngularJS script.

3. Open the `index.html` file in your editor.

4. Add an `ng-app` attribute to the `<html>` element of the page. This attribute (known as a directive) is used to bootstrap an AngularJS application. It also marks the `<html>` tag as the root element of our AngularJS application.

5. Within the `<body>` tag, add an `ng-init` attribute. Set its value to `title='ACME Traders'`.

The `ng-init` attribute is a directive that can be used to evaluate an expression in the current scope. We are using it to initialize a value named `title`. AngularJS documentation does not recommend the use of `ngInit` because it can add unnecessary amounts of logic into the markup. We are using it here strictly for demonstration. Typically, we would place all logic in a controller.

6. Within the element with a class of container, add a <h1> element. Set its text to `Welcome to {{title}}`.

This adds a data binding expression to the page. AngularJS will interpolate the value of `title` so that when the page is rendered in the browser, the `title` variable will be evaluated and rendered on the page.

7. At the bottom of the page (just before </body>), confirm that there is a reference to the `angular.min.js` file located in the `scripts` folder.

8. Save and test the page in the browser. Do you see the message displayed in the <h1>?

Instead of defining the data for the view, it is more common to define it within a controller.

9. Create a new JavaScript file named `myApp.js` file from the `scripts` directory. Within it, add the following line of code:

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

This line will be used to register an AngularJS module. Modules serve as containers for AngularJS components.

10. Below that, add the following line of code to register an AngularJS controller:

```
myApp.controller('StockController', function($scope){
});
```

11. Within the controller, add the following code to define properties on the scope:

```
$scope.logo = 'images/acme-logo.png';
$scope.today = new Date();
$scope.customer = 'Some name';
```

12. Save the file and open the `index.html` file in your editor. Add a reference to the `myApp.js`. Make sure you reference this file after you have referenced the AngularJS script.

13. Change the value of the `ng-app` attribute to `stockApp` (this is the name you specified when you created your module).

14. Add the `ng-controller` attribute to the tag with a `class` of `container`. Give it a value of `StockController`.

15. Add an `<img>` tag above the `<h1>` that you added before. Include an `ng-src` attribute and set its value to `{{logo}}`. The `ng-src` directive is used to specify a URL that will be interpolated by AngularJS before the browser attempts to load the resource.

16. After the `<h1>`, add a paragraph that contains the following:

```
{{customer}}'s Portfolio - {{today}}
```

17. Save the page and then test in the browser. Do you see the image and the customer's name and the date displayed on the page?

The date is not formatted at all and renders with both the date and time information as a string. Although we could provide a more user-friendly format for the date using JavaScript code, instead we will use an AngularJS filter.

18. Modify the previous expression by including a pipe (`|`) character followed by the word `date`. It should resemble the following:

```
{{customer}}'s Portfolio - {{today | date}}
```

19. Save and test the page again in the browser. Do you notice any difference in the format of the date?

In addition to being able to store primitive types on the scope, we can also store more complex types. In the following steps, we will store an object on the scope and then bind it to the view.

20. Re-open the `myApp.js` file in your editor.

21. Within the controller, define a property on the `$scope` named `stock`. Set it equal to an object literal that defines `name`, `symbol` and `price` properties. Make up values for a stock, or you can use `Apple`, `AAPL` and `101.42`.

22. Save the file.

23. Re-open the HTML file in the editor. Add a `<table>` below the paragraph. Add the following Bootstrap classes to format the table: `table table-bordered table-striped`

24. Add a `<thead>` element with a single row. The row should contain table headers for Symbol, Name and Price.

25. Below the `<thead>`, add a `<tbody>` element. Define a single row within the `<tbody>`.

26. Add three `<td>` elements to the row. Within each row, include a data binding expression to display the properties of the `stock` object. Use the following format to display the property:

    `{{stock.property}}`

27. Save and test the page in the browser. Do the values render properly?

28. Using the same syntax that we used to format the date, include the `currency` filter to format the price.

29. Save and test the page again.

Instead of defining individual properties to store a stock, it is more likely that we need to store multiple stocks. We can do that using an array.

30. Within the controller, define another property on the `$scope` named `stocks`. Set it equal to an array of `stock` objects. Each stock should have `name`, `symbol` and `price` properties. You can either create your own or use the following data:

    - `Apple, AAPL, 101.42`
    - `Microsoft, MSFT, 52.29`
    - `Netflix, NFLX, 100.72`
    - `Jet Blue, JBLU, 21.51`
    - `Amazon, AMZN, 596.38`

NOTE: We are hard-coding the stocks, but in a real-world application, we would typically retrieve the stock information from a remote server.

31. Modify the `<tr>` tag that appears within the `<tbody>` to include the `ng-repeat` attribute. `ngRepeat` is a directive that repeats a template for each item in a collection. Set the value of `ng-repeat` to `stock in stocks`.

32. Save and test the page in the browser. What happened? You should now see multiple stocks defined in the table.

33. Close the editor.

Exercise 2 – Tracking User Input with AngularJS Data Binding (Optional)

In this exercise, you will use define several values within a controller and then link that controller to a view. Using AngularJS directives, you will bind the values defined on the controller's scope to the view. As the user enters text into a textarea field, the number of characters entered into the field will be displayed on the page. Because of the way that AngularJS's two-way data binding works, changes to the text area will be visible immediately on the page.

1. Use Windows Explorer to copy the `CheckLength` directory from the `Mod02/Starters` directory into the `Mod02/Labs` folder.

2. Using Windows Explorer, examine the contents of the `CheckLength` directory.

You will notice there is a pair of folders (`css` and `scripts`) as well as a `character-length.html` file already present. The `css` folder contains the `bootstrap.min.css` file. The `scripts` folder already contains the minified version of the AngularJS script.

3. Begin by creating a new JavaScript file in the `scripts` folder named `charlength.js`. Within it, add the following line of code:

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

This line will be used to register an AngularJS module. Modules serve as containers for AngularJS components.

4. Below that, add the following line of code to register an AngularJS controller:

```
app.controller('MainController', function($scope){
});
```

Normally, controllers are stored in a separate `controllers` directory. But for this exercise, we will just create the controller within this same file as the module.

In the following steps, you will define variables on the `$scope`. `$scope` (pronounced dollar scope) is an AngularJS service that is automatically injected into a controller. The `$scope` is used to store values (and functions) that should be available to a view using AngularJS's two-way data binding. You will use syntax similar to the following to define variables on the `$scope`.

```
$scope.variablename = value;
```

5. Within the controller's function, define the following variables and assign the specified values:

   - `WordLength = 0;`
   - `Text = '';`
   - `WordLengthStyle = { 'color' : '#ff0000' };`
   -
```
UpdateWordLength = function(){
                     $scope.WordLength = $scope.Text.length;
                   };
```

6. Save the file.

7. Now open the `character-length.html` file in your editor.

8. Add an `ngApp` directive to the `<html>` element of the page. Set its value to `myApp` (this is the name you specified when you created your module).

9. At the bottom of the page (just before `</body>`), add a reference to the `angular.min.js` file located in the `scripts` folder.

10. You will also need to reference the `charlength.js`. Make sure you reference this file after you have referenced the AngularJS script.

11. Add the `ngController` directive to the `<body>` tag. Give it a value of `MainController`.

12. Next, locate the <label> element with a for attribute of comment. Add an ng-model attribute (this is called a directive) with a value of WordLength. This will bind the value of the WordLength variable (property) to this element.

13. Within the label, add an ng-style attribute to the span tag. This will set the style attribute using the value stored on the $scope. Then, use a data binding expression to display the WordLengthStyle property inside the span.

14. Add the ng-model attribute to the textarea element. Assign a value of Text to bind the field to the Text property on the scope. Use the ng-change attribute to call the UpdateWordLength() function defined on the scope. This will cause the WordLength property to be synced with the value entered into the textarea.

The textarea has an ng-trim attribute set to false which indicates that whitespace should not be trimmed automatically.

15. Save and test the page in the browser. What happens when you enter text into the textarea?

16. Re-open the charlength.js file in your editor. Set the value of the Text property on the scope to some message.

17. Save and test the page again in the browser. Do you see your message within the textarea? What about the character count?

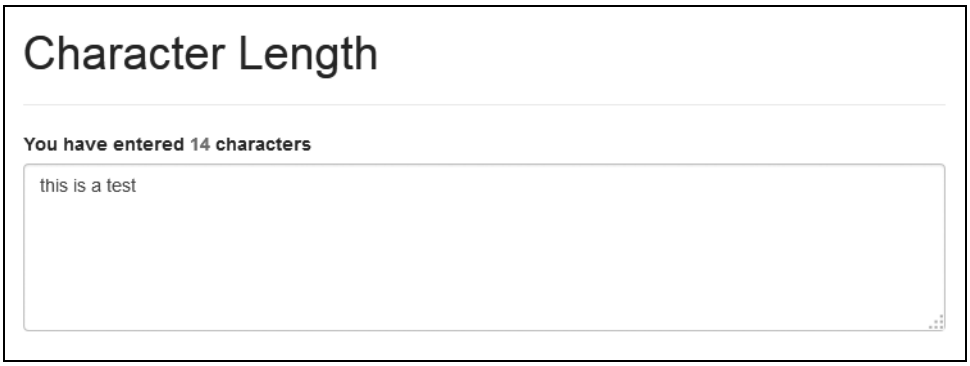
18. The character count does not reflect the text entered into the text area now because the WordLength gets updated when the value in the textarea changes. Any ideas on how to fix this so it does reflect the value when we load the page?

19. Change the value of the WordLength variable in the charlength.js file from 0 to $scope.Text.length.

20. Save and test the page. What happens? You should receive an error because you are attempting to use the Text property before it has been defined.

21. To fix this, switch the order of property declarations so the Text is defined before the WordLength.

22. Save and test the page again. Was the problem fixed?



Module 3 Exercises
Working with Filters

AngularJS filters are used to format data that is presented to the user. There are filters that can be used to format string or numeric data, as well as others that work with objects like arrays and dates. You can also define your own filters.

Filters are invoked within HTML by using the pipe (`|`) character inside a data binding expression {{}}. Some filters allow arguments to be passed. For example, you could pass an argument to the date filter to specify a specific format, or, you could pass a property or field name to the `orderBy` filter to indicate that sorting should be done on a property or field.

Exercise 1 – Experimenting with AngularJS Filters

In this exercise, you will use several AngularJS filters to format data. In addition, you will provide sorting behavior to a table so the user will be able to click a table header to sort a column in ascending or descending order.

1. Use Windows Explorer to copy the `EmployeeList` directory from the `Mod03/Starters` directory into the `Mod03/Labs` folder.

2. Using Windows Explorer, examine the contents of the `EmployeeList` directory.

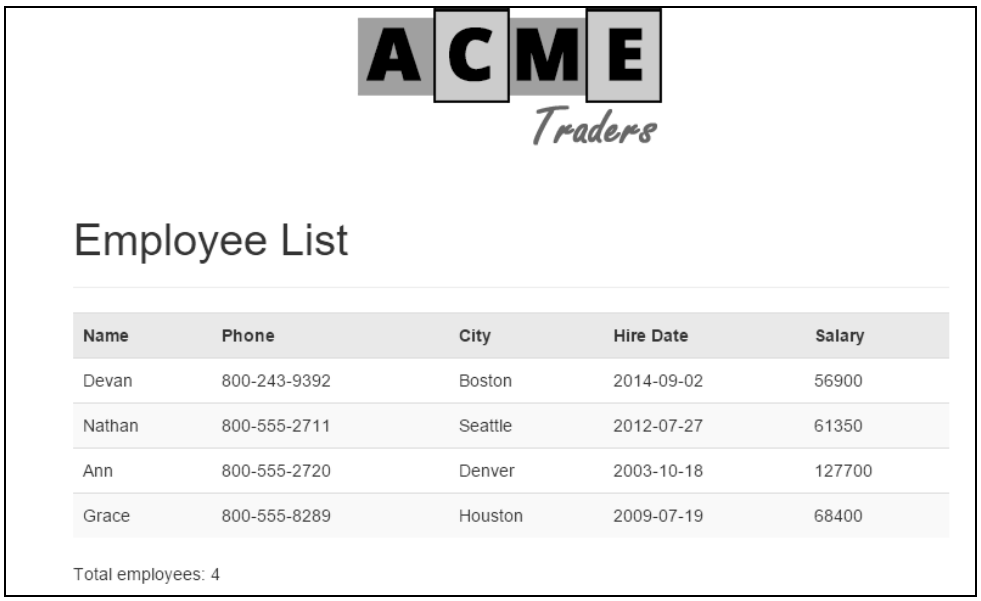
You will notice there are three folders (`css`, `images` and `scripts`) as well as an `employees.html` file already present. The `css` folder contains the `bootstrap.min.css` file. The images folder contains a single image file. The `scripts` folder already contains the minified version of the AngularJS script as well as a script defining the application.

3. Open the `employees.html` file in your editor.

This page already contains several directives and scripts that are necessary to create an AngularJS application.

- The `ngApp` directive is used on the `<html>` element to bootstrap the application. The value of `filterApp` indicates that we are using a specific AngularJS module
- The `ngController` directive is used to associate the `EmployeeController` (defined in the `scripts/controllers/EmployeeController.js` file) with a `<div>` element in the view
- The `ngRepeat` directive is used to render a template (the table row) for each employee in the `$scope.employees` property
- There are references to the AngularJS script and the `filterApp.js` and `EmployeeController.js` file

4. Open the page in the browser. It should resemble the following:



In the following steps, we will use filters to format some of the fields.

5. Use a filter to display the city field in uppercase.

6. Next, use a filter to format the hire date column using the `date` filter. Try customizing the format of the date by supplying an argument to the filter.

7. Format the salary field as currency.

8. Save and test the page in the browser. Does it resemble the following?

Now we will provide sorting behavior the table by defining a custom function in the controller. To use the `orderBy` filter, we will need to supply an expression that determines how the sorting will be performed. We will then provide the ability to reverse the sort order.

9. Add the `orderBy` filter to the expression that outputs the employee's name. Following the filter, supply: `'name'` to indicate that sorting should be performed on the name.

10. Save and test the page in the browser. Are the records now sorted by the name?

The records should be sorted in ascending order. In order to sort the records in descending order, we need to add another parameter to the filter.

The `orderBy` filter defines an optional `reverse` parameter that will reverse the sort order if set to `true`.

11. Add another argument to the `orderBy` filter with a value of `true`.

12. Save and test the page in the browser again. Do you see that the name is now sorted in reverse order?

13. Instead of just sorting by the name, it might be nice to allow the user to choose the column to sort on.

14. Add a property to the `$scope` defined in the controller named `sortBy`. Set its value to `'name'`.

15. Define another property named `reverse` on the `$scope` and set it to `false`. We will use this to specify the sort order (ascending or descending).

16. Next, define another property named `doSort` on the `$scope`. Set it equal to an anonymous function that defines a single parameter, named `propName`.

17. Within the function, set the `sortBy` property to the `propName` parameter that is passed in. In addition, toggle the `reverse` property.

18. Open the `employees.html` file in your editor.

19. Change the arguments that are being passed to the `orderBy` filter. Instead of just passing `name`, use the `sortBy` property. Instead of the value `true`, use the `reverse` property.

20. Save and test the page in the browser. The records should be sorted by name in reverse order.

We need a way to invoke the `doSort` function and toggle the `reverse` property. We will do that by adding the `ngClick` directive to the table headers.

21. Add an `ng-click` attribute to each of the `<th>` elements. Call the `doSort()` function on each of the headers, passing in the name of the field.

22. Save and test the page in the browser. Are you able to sort by clicking on the column headers? What if you click a second time on the header, is the sort order reversed?

Exercise 2 – Creating an Employee Search Feature with AngularJS Filters (Optional)

In this exercise, you will use an AngularJS filter to provide the user to search through a set of records. After you provide the search feature, you will then modify the search criteria to search on a single field.

1. Use Windows Explorer to copy the `EmployeeSearch` directory from the `Mod03/Starters` directory into the `Mod03/Labs` folder.

2. Use Windows Explorer to examine the contents of the `EmployeeSearch` directory.

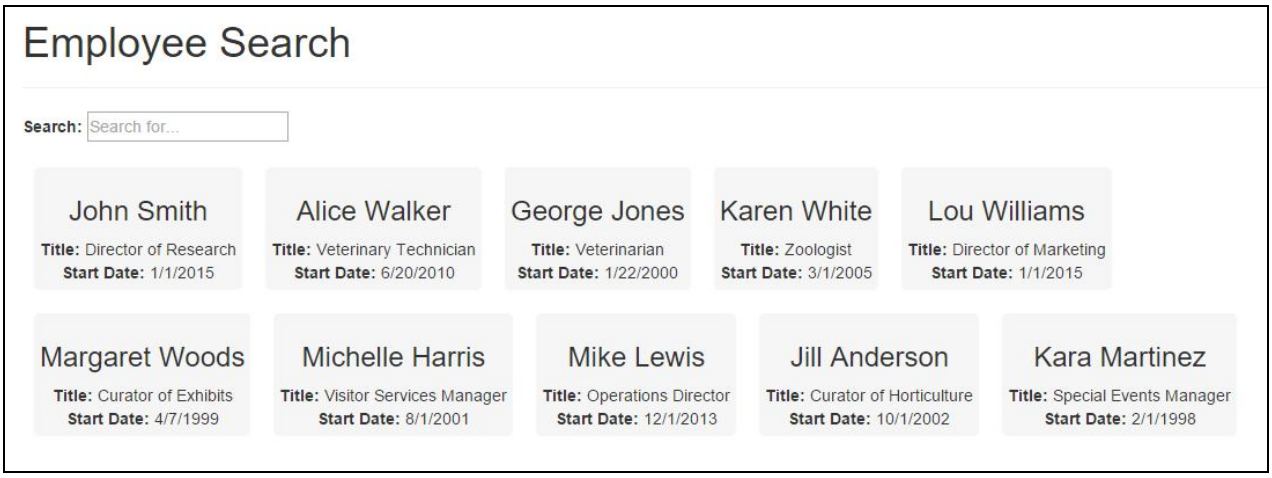
You will notice there are two folders (`css` and `scripts`) as well as an `employeedirectory.html` file already present. The `css` folder contains the `bootstrap.min.css` file. The `scripts` folder already contains a minified version of the AngularJS script as well as a script defining our application.

3. Open the `employeedirectory.html` file in the editor.

This page already contains several directives and scripts that are necessary to create an AngularJS application.

- The `ngApp` directive is used on the `<html>` element to bootstrap the application. The value of `filterApp` indicates that we are using a specific AngularJS module
- The `ngController` directive is used to associate the `EmployeeController` (defined in the `scripts/controllers/EmployeeController.js` file) with a `<div>` element in the view
- The `ngRepeat` directive is used to render a template (the table row) for each employee in the `$scope.employees` property
- There are references to the AngularJS script and the `filterApp.js` and `EmployeeController.js` file

4. Open the page in the browser. It should resemble the following:

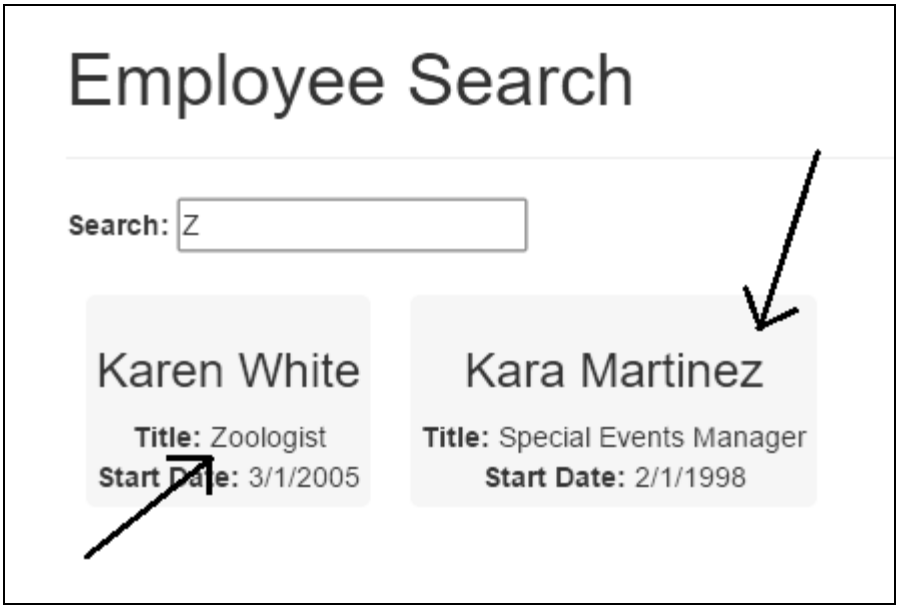


In the following steps, we will add the search and filter functionality to our application using the `filter` filter. We will also use the `orderBy` filter to sort the results by the employee's name.

5. Add an `ng-model` attribute to the search text field. Give it a value of `emp` which is the same value that we used to represent an individual employee in the `ngRepeat` directive.

6. Now, add the `filter` filter and provide the argument of `emp`. In addition, chain the `orderBy` filter to sort the results by the employee's name.

7. Save and test the page in the browser. What happens as you enter characters into the search field?



The filter is being performed on any character in any property of the employee. Question: How would we limit the search to a specific property? Answer: We would have to change the `ng-model` to the object's property you want to filter on. (For example: Set the `ng-model` to `emp.title`.)

8. Change the `ng-model` to `emp.title`.

9. Save and test the page in the browser. Now does the page work as expected?

Module 4 Exercises
Directives

AngularJS uses directives to extend HTML by providing new elements and attributes. Directives are markers on a DOM element that tells AngularJS's HTML compiler to attach a specified behavior to that DOM element. Directives can be used to attach event handlers, or to transform the DOM element and its children. AngularJS defines several built-in directives, but you can also define your own. Built-in directives are prefixed with `ng-`. AngularJS

Exercise 1 – Using AngularJS Directives

In this exercise, you will use several AngularJS directives to retrieve and display an array of album objects that are defined on the `$scope`. These albums are the "Top 10 Greatest Albums of All-Time" according to Rolling Stone magazine.

1. Use Windows Explorer to copy the `TopAlbums` directory from the `Mod04/Starters` directory into the `Mod04/Labs` folder.

2. Use Windows Explorer to examine the contents of the `TopAlbums` directory.

You will notice there are three folders (`images`, `css` and `scripts`) as well as an `albums.html` file already present. The `images` folder contains several image files. The `css` folder contains the `bootstrap.min.css` file. The `scripts` folder already contains a minified version of the AngularJS script as well as another script defining our application.

3. Begin by creating a new JavaScript file in the `scripts` folder named `albums.js`. Within it, add the following line of code:

```
var app = angular.module('top10App', []);
```

This line will be used to register an AngularJS module. Modules serve as containers for AngularJS components.

4. Next, add another JavaScript file named `AlbumController.js`. Save this file to the `scripts/controllers` folder.

5. Within this file, add the following line of code to register an AngularJS controller:

```
app.controller('AlbumController', function($scope){
});
```

6. Open the `albumstarter.txt` file from the `Starters` folder in Notepad. Copy the contents of the file and paste within the function you just added.

7. Save the file.

8. Now open the `albums.html` file in your editor.

9. Add an `ngApp` directive to the `<html>` element of the page. Set its value to `top10App` (this is the name you specified when you created your module).

10. At the bottom of the page (just before `</body>`), add a reference to the `angular.min.js` file located in the `scripts` folder.

11. You will also need to reference the `albums.js` and `AlbumController.js` files as well. Make sure you reference these after you have referenced the AngularJS script.

12. Add the `ngController` directive to the `<body>` tag. Give it a value of `AlbumController`.

13. Next, locate the `<button>` element with the text `"Get Albums"`. Add an `ngClick` directive which will call the `getAlbums()` method that was added to the scope.

This method will assign an array of album objects to an `albums` property on the scope.

14. Within the `<tbody>` element, locate the `<tr>` element and use the `ngRepeat` directive to render a template (the row with its different cells) for each of the albums in the array.

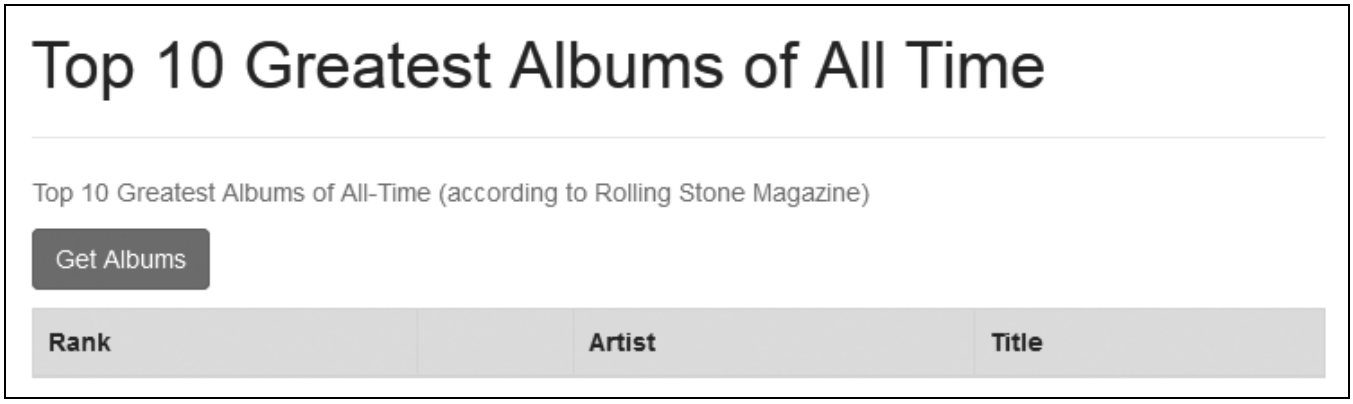
15. Use AngularJS data binding expressions to bind the properties of each album within the appropriate cells. Display the properties as follows:

    - Display the album's `rank` in the first cell
    - Display the album's `cover` in the second cell using an `img` tag. The images are located in the `images` folder. Use an expression to output the `cover` property.
    - Display the `artist` in the third cell
    - In the fourth cell, you will need to display (in order) the `title`, `label` and `released` properties in separate `<span>` tags

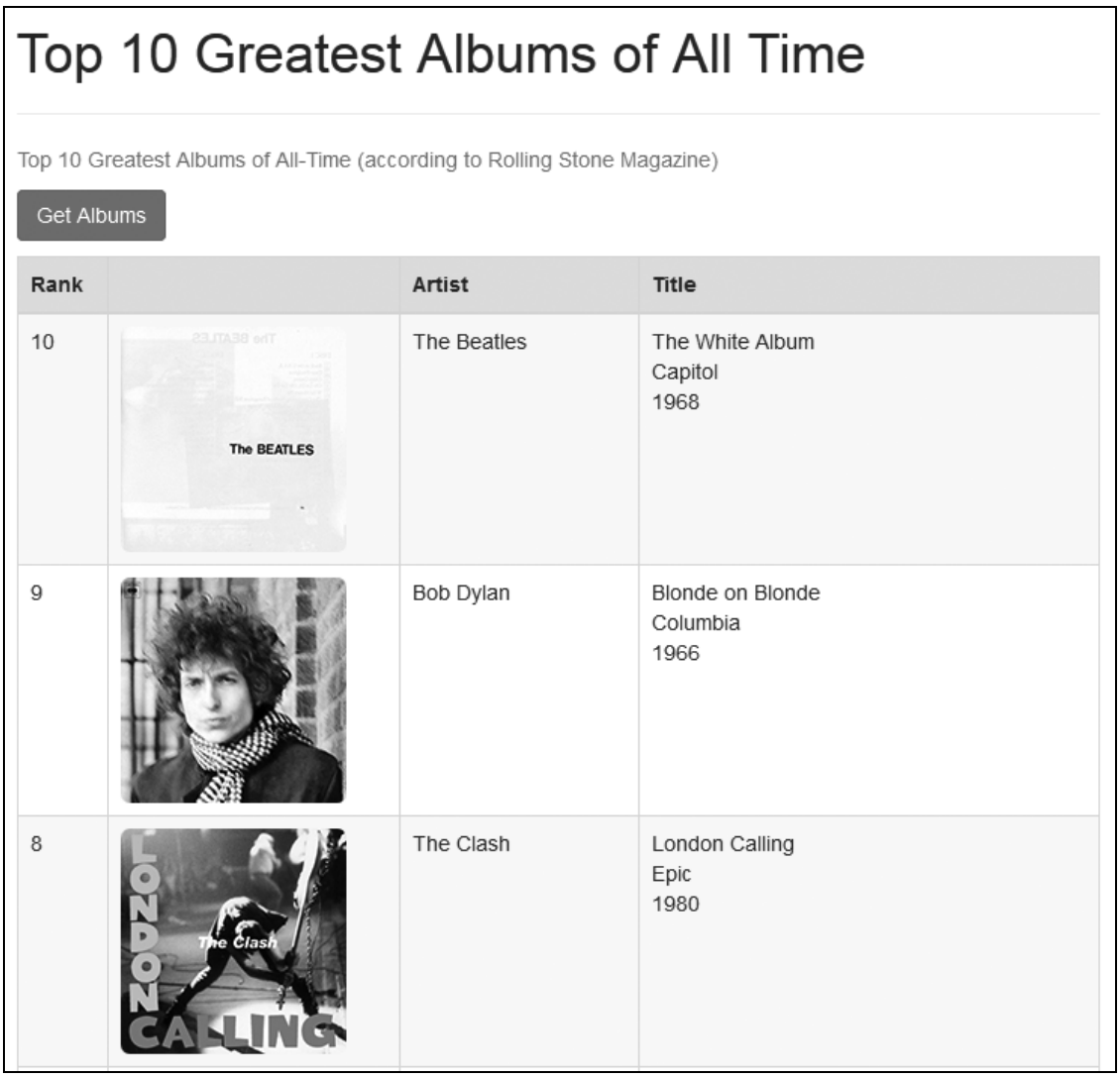
16. Save and test the page in your browser.

NOTE: When you test the page, use the Developer Tools and refresh. Did the browser attempt to load an image using a URL containing the curly braces? This will occur if you tried to use the `src` attribute instead of the `ng-src` directive. If you remembered to use the directive, good job! If you didn't, try switching the attribute in the `img` tag and test again.

On page load:



After clicking Get Albums button:



Exercise 2 – Experimenting with AngularJS Directives (Optional)

In this exercise, you will use several AngularJS directives to implement the functionality of a basic shopping cart. Users will be able to add and remove cart items by using buttons to change an item's quantity. As items are added or removed from the cart, a total will be displayed. When there is a quantity of 0 for a particular item, its decrease button will be disabled. This will all be done using AngularJS directives.

NOTE: The majority of functionality will be provided for you in the controller. You will simply need to use some of the built-in directives to access the properties and behaviors defined on the scope.

1. Use Windows Explorer to copy the `AngularCart` directory from the `Mod04/Starters` directory into the `Mod04/Labs` folder.

2. Using Windows Explorer, examine the structure of the `AngularCart` directory.

You will notice there are four folders (`images`, `fonts`, `css` and `scripts`) as well as a `cart.html` file already present. The `fonts` folder is used to supply glyphicons to the Bootstrap CSS. The `images` folder contains several image files. The `css` folder contains the `bootstrap.min.css` and a `cart.css` files. The `scripts` folder already contains a minified version of the AngularJS script, as well as a `StoreController.js` file (located under `controllers`).

3. Take a moment to examine the `StoreController.js` file in your editor. Within it, you should see several properties defined. A few of these will be used to store the cart's total and quantity, as well as the products that are available. More importantly, there are several methods that provide the cart's behavior. There are methods to increase and decrease the quantity of a particular product in the cart, one to clear the cart and others to update the Total and Quantity.

4. Add a new JavaScript file to the `scripts` folder named `cart.js`. Within it, add the following line:

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

This line will be used to register an AngularJS module. Modules serve as containers for AngularJS components.

5. Now open the `cart.html` file in your text editor.

6. Add an `ngApp` directive to the `<html>` element of the page. Set its value to `storeApp` (this is the name you specified when you created your module).

7. At the bottom of the page (just before `</body>`), add a reference to the `angular.min.js` file located in the `scripts` folder.

8. You will also need to reference the `cart.js` and `StoreController.js` files as well. Make sure you reference these after you have referenced the AngularJS script.

9. Add the `ngController` directive to the `<body>` tag. Give it a value of `StoreController`.

10. Locate the `<!—cart items -->`comment within the table. Below it, add the `ngRepeat` directive on the `<tr>` to display a template for each of the products defined on the scope. Use the `orderBy` filter to allow sorting of the products by their `name` property.

11. Next, use a data binding expression to display the `name` of the product in the first cell of the row.

12. Then, add the `ngModel` directive to the `input[type=tel]` field and bind the field to the `qty` property of the product. Also add the `ngChange` directive to invoke the `update()` function defined on the scope. This will update the item's and cart's total and quantity if the user types directly into the field.

13. Next, use the `ngDisabled` directive on the "increase" button. Set its value to "`p.qty >= 50`". This will cause the button to become disabled if the product's quantity is greater than or equal to 50. Use the `ngClick` directive to invoke the `increase()` function from the scope. Pass `$index` as an argument.

NOTE: We are using `p` to represent the product, but this should be the same value you used in the `ngRepeat` directive to represent a single product.

14. Do the same for the "decrease" button. Set its value to "`p.qty < 1`". This will cause the button to become disabled if the product's quantity is less than 1. Use the `ngClick` directive to invoke the `decrease()` function from the scope. Again, pass `$index` as an argument.

15. Within the `<td class="tdRight">` element, use a data binding expression to display the product's `price` and format as currency.

16. Within the `<td class="tdCenter">` element, add an `ngClick` directive to the "remove" hyperlink that calls the `clear()` method. Pass `$index` as an argument.

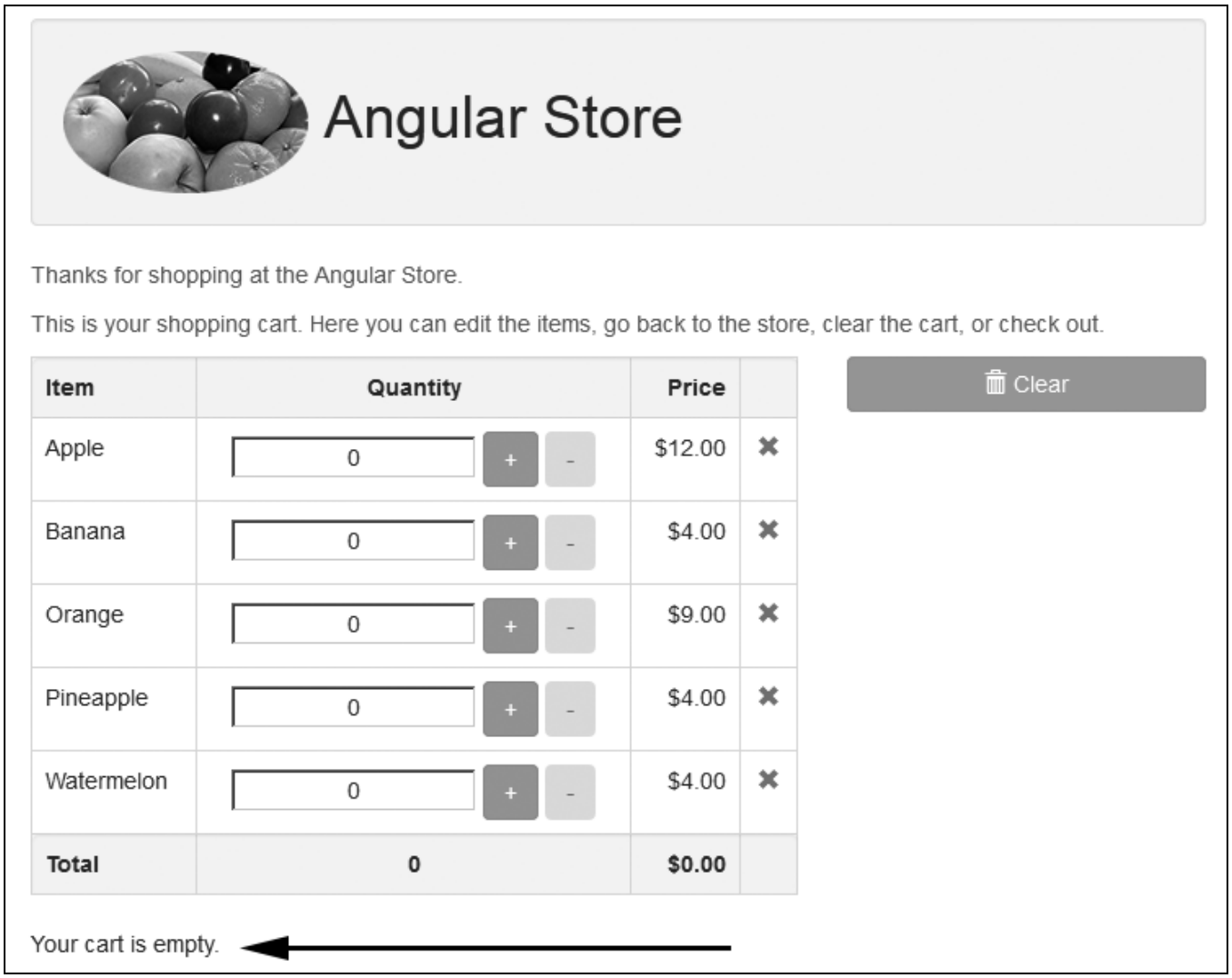
17. Next, within the `<tr class="well">`, display the `Quantity` and `Total` properties from the scope using data binding expressions. Display the `Quantity` in the cell with a `class` of `tdCenter` and the `Total` in the cell with a `class` of `tdRight`. Format the `Total` as currency.

18. Locate the `<!—empty cart message -->`comment. Add the `ngHide` directive to the paragraph that hides the paragraph when the `Total` is greater than 0.

19. Finally, add the `ngClick` directive to the "Clear" button. When the button is clicked, call the `clearAll()` method. In addition, add the `ngDisabled` directive. The button should be disabled when the `Quantity` is less than 1.

20. Save and test the page in the browser. Are you able to add and remove products from the cart? Do the buttons become enabled and disabled when expected?

    On page load:



After adding items to cart: