ANGULARJS BOOTCAMP

LEARN THE BASICS OF ANGULARJS PROGRAMMING

DAVID MAXWELL

Angular JS Bootcamp

Learn the Basics of Angular Programming in 2 Weeks!

Contents

Introduction

I want to thank you and congratulate you for downloading the book, *"Angular JS: Learn the Basics of Angular Programming in 2 Weeks!"*

Known as an open source web framework, Angular JS is currently maintained by Google —which makes it one of the most important and prolific programming languages around, especially because it's used for single-page applications. This is exactly the reason why you have to learn how to code in the said language.

With the help of this book, you'll learn about the basics of Angular JS Programming in just a matter of 2 weeks—and understand how to code so you could spruce up those HTML pages even more.

Read this book now to find out how!

Thanks again for downloading this book, I hope you enjoy it!

Committee of the American Bar Association and a Committee of Publishers and Associations.

The information provided herein is stated to be truthful and consistent, in that any liability, in terms of inattention or otherwise, by any usage or abuse of any policies, processes, or directions contained within is the solitary and utter responsibility of the recipient reader. Under no circumstances will any legal responsibility or blame be held against the publisher for any reparation, damages, or monetary loss due to the information herein, either directly or indirectly.

The information herein is offered for informational purposes solely, and is universal as so. The presentation of the information is without contract or any type of guarantee assurance.

The trademarks that are used are without any consent, and the publication of the trademark is without permission or backing by the trademark owner. All trademarks and brands within this book are for clarifying purposes only and are the owned by the owners themselves, not affiliated with this document.

Week 1

Chapter 1: What Is Angular JS?

On the first week of the Angular JS Tutorial, it's just imperative that you really learn about what it's about first.

For starters, Angular JS, also known as Angular, is used by Google as an open source web application framework for the development of single-page applications. Angular makes it easy for both corporations and individuals to understand what's going on in various applications.

Angular is also a JavaScript framework, which means that tags used to code it also run on the JavaScript language. Therefore, a simple example of Angular would look like what's written below:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angula
r.min.js"></script>
<body>

<div ng-app="">
 <p>Name: <input type="text" ng-model="name"></p>
 <p ng-bind="name"></p>
</div>

</body>
</html>
```

What happens is that Angular reads the HTML codes first before working by itself. It makes use of embedded tag attributes, and lets those attributes bind variables that are represented by JavaScript, and could then be used for JSON Resources.

Scope

The term “scope” is used to build Computer Science fundamentals when certain kinds of binding are deemed to be valid. This means there’s a lexical environment going on.

It could also deem the literal scope of the program—which means it should follow variable scope rules, as well as the scope of the name binding, too.

Goals

The creators of Angular JS had certain goals in mind, and these are:

1. To make sure that the application's client side works well with the server side. This makes it easy for parallel programming developments to be done—so both sides could be used in the application;

2. To manipulate DOM from the application logic in a dual manner so that structured codes could be provided for the program, and;

3. To give the application building journey a structure—from the design, all the way to testing and execution.

This way, the language could be used for declarative programming, which means it could collect and connect the components of software, and could be able to create user interfaces, as well.

When this happens, the framework that you have created will be able to adapt HTML codes, and have them extended, too. This creates two-way data, and synchronizes both views and models, and also allows MVC Patterns to separate logic, data, and presentation components. With these, server-side presentations are brought out again, and client-side web applications are used, too.

Performance and Bootstrapping

When it comes to performance, bootstrapping is the number 1 thing that Angular does. This basically happens in a matter of three phases, which are as follows:

1. Directives get to be linked to the scope;

2. Directives are compiled so that DOM would be decorated, and;

3. New Injectors will be created.

This makes Angular the frontend of the *NodeJS* Runtime, together with *MEAN Stack*, and *Express.Js*.

These days, Angular JS is used on the websites of *ABC News*, *Sprint*, *Intel*, *Walgreens*, *NBC*, and *Wolphram Alpha*, amongst others, and also uses the *Libscore* Analysis for further developments.

An Angular plug-in was also created for *Google Chrome* back in 2012, which happens by means of the changing of scopes. This also makes the controller easy for watching, and so that loop iteration would be monitored.

Chapter 2: Important Angular Terminologies

Of course, there are also certain terms used in Angular JS. By knowing these terms, it would be easy for you to know what you're dealing with—and coding wouldn't ever be a problem.

For this, you could take note of the following terms:

Ng-bind. This means that DOM element texts are being set. This means that the value of the name would be seen inside the span element. Any changes made to the name would directly reflect on other elements of the DOM. For example:

```
<span ng-bind="name"></span>
```

Ng-app. This derives the Angular's root element so that behavior and directives would be defined, as well.

Ng-model-options. This allows updates for the model to be done by means of fine tuning.

Ng-model. This is almost the same as Ng-bind so that there'd be a bond between the scope and the view.

Ng-controller. This means that HTML expressions are evaluated with the help of the JavaScript controller class.

Ng-class. These are dynamically-loaded class attributes.

Ng-show and Ng-hide. This shows or hides the elements conditionally, which always depends on the Boolean expression formula. This could be achieved by using CSS styles.

Ng-repeat. This instantiates elements for a collection at least once.

Ng-view. This is responsible for handling JSON routes for the base directives which are also specified by certain controllers.

Ng-switch. This helps you choose from a set of templates depending on the values of the expressions.

Ng-animate. This helps provide enough support for CSS3 Keyframe, CSS3 Transition, and JavaScript. This also provides enough custom and core directives, together with animation keyhooks, too.

Ng-aria. This one creates common ARIA attributes, and also serves as a module of accessibility and support.

Chapter 3: Angular Expressions

In coding the Angular language, you of course have to make use of expressions. These are written in double braces {{}}, but could also be contained within a directive, just like this one below:

```
Ng-bind= “expression”
```

The expression will then be resolved by Angular itself, and have results returned when they are ready. They pretty much mirror JavaScript expressions, so if you know something about that, this language would not be too hard for you.

This also means that Angular Expressions are:

1. Angular Expressions could contain filters—but JavaScript expressions do not have them.

2. They do not support exceptions, loops, and coordinates.

3. The expressions could be written inside HTML.

4. Angular Expressions could also be variables, operators, and literals.

For example:

```
{{ 10 + 5 }} or {{ name + ” ” + nickname }}
```

This means that a basic angular expression code would look like the following:

```
<!DOCTYPE html>
<html>
<script src=“http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js”>
</script>
<body>

<div ng-app=””>
  <p>Supergirl Versus Bizarro: {{ Supergirl +Bizzaro }}</p>
</div>

</body>
</html>
```

By taking away the *ng* directive, HTML would not be able to solve the expression, but just show it as it is, which would just basically look the same as this:

```
<!DOCTYPE html>
<html>
<script src=“http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js”>
</script>
<body>
```

```
<div ng-app=””>
  <p>Supergirl Versus Bizarro: {{ Supergirl +Bizzaro }}</p>
</div>

</body>
</html>
```

Another example would be if you want to change certain colors and styles in CSS. For example, if you want to change the color of a box as pink, you could try the following:

```
<div ng-app=”” ng-init=“myCol=‘pink’”>

<input style=“background-color:{{myCol}}” ng-
model=“myCol” value=”{{myCol}}”>

</div>
```

Angular Numerical Expressions

Numerical Expressions are also used in Angular JS Programming. For this, you could use *ng-bind,* which would give you the following expressions:

```
<div ng-app=”” ng-init=“quantity=2;cost=10”>

<p>Total in dollar: <span ng-bind=“quantity * cost”></span></p>

</div>
```

Or, you could also choose not to use the ng-bind function, and get the following result:

```
<div ng-app=”” ng-init=“quantity=2;cost=10”>

<p>Total in dollar: {{ quantity * cost }}</p>

</div>
```

Angular Arrays

This works like JavaScript Arrays. If you're going to use *ng-bind*, it would look something like this one below:

```
<div ng-app="" ng-init="points=[2,4,6,8]">

<p>The third result is <span ng-bind="points[2]"></span></p>

</div>
```

But, if you're going to take away ng-bind, it would look something like this one:

```
<div ng-app="" ng-init="points=[2,4,6,8]">

<p>The third result is {{ points[2] }}</p>

</div>
```

Angular Objects

You could also take note that Angular Objects work like JavaScript objects, especially with *ng-bind*. It would basically show up like what you see below:

```
<div ng-app="" ng-init="person={firstName:'Harry',lastName:'Styles'}">

<p>The name is <span ng-bind="person.lastName"></span></p>

</div>
```

Without binding, it would look like this:

```
<div ng-app="" ng-init="person={firstName:'Harry',lastName:'Styles'}">

<p>The name is {{ person.lastName }}</p>

</div>
```

Angular Strings

Strings connect things together, even in programming. Angular Strings also work like JavaScript. It could go two ways—just take a look at the examples below:

```
<div ng-app="" ng-init="firstName='Mark';lastName='Ruffalo'">

<p>The name is <span ng-bind="firstName + ' ' + lastName"></span></p>

</div>
```

Or:

```
<div ng-app="" ng-init="firstName='Mark';lastName='Ruffalo'">

<p>The name is {{ firstName + " " + lastName }}</p>

</div>
```

Chapter 4: Angular Modules

Basically, Angular Modules are the ones that define applications, and serve as containers for different parts of the application. They also serve as containers for various application controllers, and they always belong to modules.

In order to create a module, you have to make use of the *angular.mobile* function. What's inside the parenthesis would then determine the element where the application would run in, so that you could add filters, directives, and controllers to the application. An example is shown below:

```
<div ng-app="FavoriteApp">…</div>
<script>
var app = angular.module(”FavoriteApp”, []);
</script>
```

Adding Directives

Now, you could add directives to improve the functionality of the app you are trying to create. Directives are pretty much the terminologies given in Chapter 2, but could also be any of the following:

1. **Validation Properties**, such as: *$error, $invalid*, and *$dirty.*

2. **angular.isDate.** Returns value if date is the reference.

3. **angular.isArray**. Pertains to array values.

4**. angular.isNumber.** Pertains to numerical values.

5. **angular.isFunction**. Pertains to functional values.

6. **angular.isElement**. Pertains to DOM Elements.

7. **angular.isDefined**. Pertains to references that are defined.

8. **angular.equals**. Pertains to equal values on the program.

9. **angular.isUndefined**. Pertains to undefined reference values.

10. **angular.isObject**. Pertains to object references.

11**. angular.forEach().** This helps elements execute various functions whether in arrays, or objects.

12**. angular.Copy()**. This gives you deep copies of arrays or objects.

13. **angular.uppercase().** This changes values to uppercase letters

14**. angular.lowercase().** This changes values to lowercase letters

15. **<textarea>.** Modifies the text area's behaviors.

16**. <select>.** Helps you create selections on the program.

17. **<script>.** This modifies the behavior's script.

18. **<input>.** This helps you input different functions to the program.

19. **<form>.** This helps you create forms in the program.

20. **<a>.** This one helps input various variables in the program.

21. **angular.module().** This helps you create, retrieve, or register modules.

22. **angular.element().** This makes use of JQuery Elements to wrap the HTML Element with.

23. **angular.bootstrap().** This manually starts Angular JS.

You could also use your own directives, which means the code would look something like the one below:

```
<div ng-app="FavoriteApp" w3-test-directive></div>
<script>
```

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

```
app.directive(“w3TestDirective”, function() {
    return {
        template : “I was made in a directive constructor!”
    };
});
</script>
```

Adding Controllers

Of course, you could also add controllers to your codes by using the *ng-controller* directive. Basically, it would show up like this:

```
<div ng-app=”FavoriteApp“ ng-controller=“myCtrl”>
{{ AppFirstName + ” ” + ApplastName }}
</div>
<script>
```

```
var app = angular.module(“FavoriteApp”, []);

app.controller(“myCtrl”, function($scope) {
   $scope.firstName = “Harry”;
   $scope.lastName = “Styles”;
});
```

```
</script>
```

File Modules and Controllers

Then, you could create modules and controllers inside the files so that your applications would even be more defined. For this one, you could copy what's shown below:

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
<body>
<div ng-app="FavoriteApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>
<script src="FavoriteApp.js"></script>
<script src="myCtrl.js"></script>
</body>
</html>
```

Loading the Library

To load the Angular Library, you have to load it before the <head> and before the <body> tag.

For this, you have to use the *angular.module* directive and you'd get something like the one below:

```
<!DOCTYPE html>
<html>
<body>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>

<div ng-app="FavoriteApp" ng-controller="myCtrl">
{{ AppfirstName + " " + ApplastName }}
</div>

<script>
var app = angular.module("FavoriteApp", []);
app.controller("myCtrl", function($scope) {
   $scope.firstName = "Harry";
   $scope.lastName = "Styles";
});
</script>

</body>
</html>
```

Week 2

Chapter 5: Angular Models

The next thing you have to learn about are Angular Modules during the second week of the tutorial.

This is where you'd be using the *ng-model* directive for. Basically, what happens here is that you're allowing field values to be inputted together so they'd create Angular Variables.

It also means that you have to use the following during coding:

1. ng-pristine
2. ng-pending
3. ng-dirty
4. ng-invalid
5. ng-untouched
6. ng-touched
7. ng-not-empty
8. ng-empty

For this, you could follow the codes below:

```
<div ng-app="FavoriteApp" ng-controller="myCtrl">
   Name: <input ng-model="name">
</div>

<script>
var app = angular.module('FavoriteApp', []);
app.controller('myCtrl', function($scope) {
   $scope.name = "Mark Ruffalo";
});
</script>
```

Validate User Input

For this one, you should also use the *ng-model*. This is done to validate various types of data on the program. For example, e-mail addresses or e-mails, just like what you see after you have signed up for certain websites.

For example, you can take a look at this one below:

```
<form ng-app=”” name=“AnswerThisForm”>
   Email:
   <input type=“email” name=“InputAddress” ng-model=“text”>
   <span ng-show=“AnswerThisForm.myAddress.$error.email”>Not a valid e-mail address</span>
</form>
```

Two-Way Binding

This means that binding would serve both client and server, where the value would be seen inside the input fields. This would look like what's shown below:

```
<div ng-app="FavoriteApp" ng-controller="myCtrl">
  Name: <input ng-model="name">
   <h1>Youentered: {{name}}</h1>
</div>
```

CSS Classes

You could also use models to tweak CSS Classes. This will provide all the attributes you need to make HTML Elements work, depending on how you want them to work. This should look like what's shown below:

```
<style>
```

```
input.ng-invalid {
   background-color:pink;
}
```

```
</style>
<body>

<form ng-app="" name="AnswerThisForm">
   Enter your name:
   <input name="InputMyAddress" ng-model="text" required>
</form>
```

Application Status

Another thing where you could use Angular Models for is the Application Status. This involves *dirty*, *invalid*, *error*, or *touched*, and should look like the one below:

```
<form ng-app=”” name=“AnswerThisForm” ng-init=“myName =
‘post@mysite.com’”>
   Email:
   <input type=“email” name=“myAddress” ng-model=“myName” required></p>
   <h1>Status</h1>
   {{myForm.myAddress.$valid}}
   {{myForm.myAddress.$dirty}}
   {{myForm.myAddress.$touched}}
</form>
```

Chapter 6: Angular Scopes

In Angular Programming, Scopes are equivalent to arguments. This happens because scopes bind JavaScript and HTML Element, which means that at some point, there would be some form of conflict, too.

Scopes have available methods and abilities, and are available for both controller and the view. The view depicts HTML, while controller is JavaScript—which is the one responsible for controlling, editing, or deleting data.

Scopes are also models with both properties and methods available for the user to use. It means that when changes are made, both controller and view will be affected—and updated.

Here are some examples:

```
<div ng-app="TheHungerGames" ng-controller="myCtrl">

<h1>{{TheHungerGames}}</h1>

</div>

<script>
```

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

app.controller('movies', function($scope) {
   $scope.thehungergames= "TheHungerGames";
});
```

```
</script>
```

```
   <div ng-app="TheHungerGames" ng-controller="myCtrl">

<input ng-model="thehungergames">

<h1>Movie is {{thehungergames}}</h1>

</div>

<script>
```

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

app.movies('myCtrl', function($scope) {
   $scope.name = "thehungergames";
```

```
});
</script>
```

Limit of Scopes

You also should understand where the scopes begin, and when they end. This will make your program more coherent, and help you understand what needs to be done. It’s like knowing what’s going on around you so you could easily deal with them. This is not usually an issue unless you’re dealing with large DOM Applications. The *ng-repeat* directive is essential for this one.

For this, you could follow the code below:

```
<div ng-app=“TheFirstApp” ng-controller=“myCtrl”>

<ul>
   <li ng-repeat=“x in thefirstapp”>{{x}}</li>
</ul>

</div>

<script>
```

```
var app = angular.module(‘TheFirstApp’, []);

app.controller(‘myCtrl’, function($scope) {
   $scope.names = [“the”, “first”, “app”];
});
```

```
</script>
```

Root Scope

There’s also such a thing as Root Scope (also known as *$rootscope*). This uses the *ng-app* directive and is available for the whole application—so you can say that it’s a really important part of the tutorial. This also uses the scope that’s currently used.

For example:

```
<body ng-app=“TheFirstApp”>
<p>The rootScope’s favorite color:</p>
<h1>{{pink}}</h1>
<div ng-controller=“myCtrl”>
   <p>The scope of the controller’s favorite color:</p>
   <h1>{{purple}}</h1>
</div>
<p>The rootScope’s favorite color is still:</p>
<h1>{{pink}}</h1>
<script>
```

```
var app = angular.module(‘TheFirstApp’, []);
app.run(function($rootScope) {
   $rootScope.color = ‘pink’;
});
app.controller(‘myCtrl’, function($scope) {
   $scope.color = “purple”;
});
```

```
</script>
</body>
```

Chapter 7: Angular Controllers

Then, you have to learn about Angular Controllers. These are important because they control the application's data, and are also known as regular JavaScript objects.

The *ng-controller* directive is essential here because it defines the controller of the application, and also takes care of object constructors and obstructers, as well.

You'll later notice that applications are the ones that run inside the function, and that there are also control objectives around. You can invoke controllers with the use of *$scope* because it would serve as both function and variable, and creates two variables or properties (i.e., first and last name, etc.). Finally, you could end up with the controller in the controller properties.

A good example would be this one:

```
<div ng-app="MyFavoriteApp" ng-controller="myCtrl">

First Name: <input type="text" ng-model="nameofapp"><br>
Last Name: <input type="text" ng-model="yourname"><br>
<br>
Full Name: {{nameofapp+ " " + yourname}}

</div>

<script>
var app = angular.module('myFavoriteApp', []);
app.controller('myCtrl', function($scope) {
   $scope.firstName = "MyFavoriteApp";
   $scope.lastName = "Jane Doe";
});
</script>
```

Methods of the Controller

Again, this is mostly about the two essential properties of the controllers—in this case, they are *1) the name of the app*, and *2) your name*. They work as both functions and variables, which makes them non-negotiables while coding.

Here's a good example:

```
<div ng-app="myFirstApp" ng-controller="personCtrl">

First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{givefullfirstname()}}

</div>

<script>
var app = angular.module('myApp', []);
app.controller('personCtrl', function($scope) {
    $scope.firstName = "Amelia";
    $scope.lastName = "Earhart";
    $scope.fullName = function() {
        return $scope.firstName + " " + $scope.lastName;
    };
});
</script>
```

External File Controllers

Of course, if you're dealing with larger applications, it's just imperative that you'd save your data in external files. For example, zip or rar files, or those cracks and patches you see in most computer games.

The elements should then go inside the script tags *<script>*. Take a look at the examples below and try coding them. After coding, make sure to save them as *namesController.js.*

```
angular.module('myApp', []).controller('namesCtrl', function($scope) {
    $scope.names = [
        {name:'Harry',house:'Gryffindor'},
        {name:'Draco',house:'Slytherin'},
        {name:'Luna',house:'Ravenclaw'}
    ];
});
```

```
<div ng-app="myFavoriteApp" ng-controller="personCtrl">

First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}
</div>
<script src="personController.js"></script>
```

```
<div ng-app="myFavoriteApp" ng-controller="personCtrl">

First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}

</div>

<script src="personController.js"></script>
```

Chapter 8: Angular Filters

No, this has nothing to do with adding filters to your photos to spruce them up. Filters give your applications more life—give them some form of body and soul—and could allow you to code what you want to happen to the app. This means you have to use the following variables:

1. **uppercase**. This helps you format a string to uppercase letters.

2. **orderBy.** This arranges the array based on the expressions used.

3**. number.** This helps format numbers into strings.

4. **lowercase**. This then formats the string into lowercase letters.

5. **limitTo.** This one limits strings or arrays into characters or elements that have been specified for them.

6. **json.** This helps format objects into JSON Strings.

7. **filter**. This helps you select subsets from arrays.

8. **date**. This helps format dates into formats that you have specified for them.

9. **currency**. This then helps you format currency numbers.

Expression Filters

The pipe character (|) is used to add expressions inside filters. For this, you could try the examples below:

```
<div ng-app=“myFavoriteApp” ng-controller=“personCtrl”>

<p>The name is {{ firstName | uppercase }}</p>

</div>
```

```
<div ng-app=“myFavoriteApp” ng-controller=“personCtrl”>

<p>The name is {{ lastName | lowercase }}</p>

</div>
```

Currency Filters

Now, you could try transforming a number into currency by making use of currency filters, such as the one you see below:

```
<div ng-app="myFavoriteApp" ng-controller="costCtrl">

<h1>How Much: {{ price | currency }}</h1>

</div>
```

Directive Filters

Of course, you could also tweak your directives by making use of directive filters. The *ng-repeat* directive is essential here, as well as the pipe (|).

For example, if you want your variables to be in order, you should use *orderB*y, and code like this:

```
<div ng-app="myFavoriteApp" ng-controller="namesCtrl">

<ul>
  <li ng-repeat="x in names | orderBy:'hogwartshouses'">
    {{ x.name + ', ' + x.hogwartshouse }}
  </li>
</ul>

</div>
```

Filter Filter

Wait, what?

Well, this is actually a thing! What happens here is that you’re allowed to choose a subset from a certain array, but you have to take note that you can only use what you have chosen for the same array. For example:

```
<div ng-app=“myFavoriteApp” ng-controller=“namesCtrl”>

<ul>
  <li ng-repeat=“x in names | filter : ‘i’”>
    {{ x }}
  </li>
</ul>

</div>
```

Arrays Based on Input

Now, here's the somewhat complicated but fun part. You can test your variables by arranging them in array mode, based on what you have inputted on the program. The names will then grow or shrink, depending on the match that you have made. The *ng-model* directive is essential here.

For example, let's use names of *Harry Potter* characters. You could list them like this:

Harry Potter

Hermione Granger

Ron Weasley

Luna Lovegood

Cedric Diggory

Fred Weasley

Cho Chang

And so on. Now, you could try the grow and shrink test by coding like this:

```
<div ng-app="myFavoriteApp" ng-controller="namesCtrl">

<p><input type="text" ng-model="test"></p>

<ul>
  <li ng-repeat="x in names | filter : test">
    {{ x }}
  </li>
</ul>

</div>
```

Try listing them again, but with their respective houses this time. For example:

Harry Potter – Gryffindor

Hermione Granger – Gryffindor

Ron Weasley – Gryffindor

Luna Lovegood – Ravenclaw

Draco Malfoy – Slytherin

Cho Chang – Ravenclaw

Cedric Diggory – Hufflepuff

Pansy Parkinson – Slytherin

Fred Weasley – Gryffindor

If you're going to use the *ng-click* directive, you'll be able to sort the names the way you

want. For example:

```
<div ng-app="myFavoriteApp" ng-controller="namesCtrl">

<table border="1" width="100%">
 <tr>
  <th ng-click="orderByMe('name')">Name</th>
  <th ng-click="orderByMe('house')">House</th>
 </tr>
 <tr ng-repeat="x in names | orderBy:myOrderBy">
  <td>{{x.name}}</td>
  <td>{{x.house}}</td>
 </tr>
</table>

</div>

<script>
angular.module('myFavoriteApp', []).controller('namesCtrl', function($scope) {
 $scope.names = [
  {name:'Harry',house:'Gryffindor'},
  {name:'Hermione',house:'Gryffindor'},
  {name:'Ron',house:'Gryffindor'},
  {name:'Luna',house:'Ravenclaw'},
  {name:'Cedric',house:'Hufflepuff'},
  {name:'Cho',house:'Ravenclaw'},
  {name:'Fred',house:'Gryffindor'},
  {name:'Draco',house:'Slytherin'},
  {name:'Pansy',house:'Slytherin'}
 ];
 $scope.orderByMe = function(x) {
  $scope.myOrderBy = x;
 }
});
</script>
```

Custom Filters

And of course, you could also personalize or customize filters the way you exactly want. For this, you could try the following:

```
<ul ng-app="myApp" ng-controller="namesCtrl">
  <li ng-repeat="x in names">
    {{x | myFormat}}
  </li>
</ul>

<script>
```

```
var app = angular.module('myApp', []);
app.filter('myFormat', function() {
  return function(x) {
    var i, c, txt = "";
    x = x.split("")
    for (i = 0; i < x.length; i++) {
      c = x[i];
      if (i % 2 == 0) {
        c = c.toUpperCase();
      }
      txt += c;
    }
    return txt;
  };
});
app.controller('namesCtrl', function($scope) {
  $scope.names =
['Harry', 'Hermione', 'Ron', 'Luna', 'Cedric', 'Cho', 'Draco', 'Fred','Pansy'];
});
```

```
</script>
```

Chapter 9: Angular HTTP and Services

And lastly, in this final chapter of the book, you'd learn how to code Angular HTTP and Services!

Angular HTTP, also known as *$http*, is essential because it helps you get and read data from remote servers. Not only that, it also allows responses to be sent to you—making the process a two-way, understandable experience.

You also have to make use of certain methods which are as follows:

1. .put()

2. .post()

3. .patch()

4. .jsonp()

5. .head()

6. .get()

7. .delete()

In order to make a simple request to the server, you could try this could below:

```
<div ng-app="myFavoriteApp" ng-controller="myCtrl">

<p>Welcome to my Blog:</p>
<h1>{{Welcome to my Blog}}</h1>

</div>

<script>
```

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
   $http.get("welcome.htm")
   .then(function(response) {
      $scope.myWelcome = response.data;
   });
});
```

```
</script>
```

To use the given methods above, you could code something that looks like the one below:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
   $http({
```

```
        method : “GET”,
        url : “welcome.htm”
    }).then(function mySucces(response) {
        $scope.myWelcome = response.data;
    }, function myError(response) {
        $scope.myWelcome = response.statusText;
    });
});
```

HTTP/Service Properties

There are also certain properties that have to be observed if you want to code HTTP or services. These are the ones listed below:

1. .**statusText**. This is a string that defines the HTTP Status.

2. .**status**. This is a number that defines the HTTP Status.

3. .**headers.** This function is used to get information regarding the function.

4. .**data.** This is an object or string that carries server responses.

5. .**config.** This is used to generate requests from the objects you have onscreen.

To understand this better, you could check out the examples below:

```
var app = angular.module(‘myFavoriteApp’, []);
app.controller(‘myCtrl’, function($scope, $http) {
   $http.get(“welcometomyblog.htm”)
   .then(function(response) {
      $scope.content = response.data;
      $scope.statuscode = response.status;
      $scope.statustext = response.statustext;
   });
});
```

```
var app = angular.module(‘myFavoriteApp’, []);
app.controller(‘myCtrl’, function($scope, $http) {
   $http.get(“wrongfilename.htm”)
   .then(function(response) {
      //First function handles success
      $scope.content = response.data;
   }, function(response) {
      //Second function handles error
      $scope.content = “Something is wrong”;
   });
});
```

JSON Object

Meanwhile, the JSON Object is all about how you could transport data from JavaScript to Angular, and vice-versa. The *ng-repeat* directive should be looped for this function. For example, you want to import data from a *Blogspot* blog to a *WordPress* one, it means that you are making use of a JSON Object.

You should also take note that *$http.get()* is all about reading the JSON Data, while *$http()* is mostly about the *XMLHttpRequestObject.* Here's a good example:

```
<div ng-app="myFavoriteApp" ng-controller="studentsCtrl">

<ul>
  <li ng-repeat="x in myData">
    {{ x.Name + ', ' + x.house }}
  </li>
</ul>

</div>

<script>
```

```
var app = angular.module('myApp', []);
app.controller('studentsCtrl', function($scope, $http) {
    $http.get("students.php").then(function(response) {
        $scope.myData = response.data.records;
    });
});
```

```
</script>
```

Services

It’s also important to use services because they allow certain functions to be limited only for your application so other people would not go ahead and copy them, and have you dealing with copyright claims.

There are a lot of services that Angular offers, but the most important one is *$location*. This allows you to save certain objects to the cloud and keep them safe, so that even if your computer or external drives break down, you know your data would still be around—which is important especially if you’re dealing with programs and applications. Here are some examples:

```
var app = angular.module(‘myFavoriteApp’, []);
app.controller(‘studentsCtrl’, function($scope, $location) {
   myUrl = $location.absUrl();
});
```

You could also use the *$timeout* service or *window.setTimeout* to help you create countdowns—these are mostly used for events, shows, and the like. For example:

```
var app = angular.module(‘myFavoriteApp’, []);
app.controller(‘myCtrl’, function($scope, $timeout) {
   $scope.myHeader = “Hi People!”;
   $timeout(function () {
      $scope.myHeader = “How are you doing today?”;
   }, 2016);
});
```

Meanwhile, the $http service could be used to obtain data coming from the server. Here’s a good example:

```
var app = angular.module(‘myFavoriteApp’, []);
app.controller(‘myCtrl’, function($scope, $http) {
   $http.get(“welcometomyblog.htm”).then(function (response) {
      $scope.myWelcome = response.data;
   });
});
```

You could also try customizing your services by doing the following:

```
var app = angular.module(‘myFavoriteapp’, []);
app.controller(‘myCtrl’, function($scope, $timeout) {
   $scope.myHeader = “Hi everyone!”;
   $timeout(function () {
      $scope.myHeader = “How are you doing today?”;
   }, 2000);
});
```

```
var app = angular.module(‘myFavoriteapp’, []);
app.controller(‘myCtrl’, function($scope, $interval) {
   $scope.theTime = new Date().toLocaleTimeString();
   $interval(function () {
```

```
        $scope.theTime = new Date().toLocaleTimeString();
    }, 1010);
});
```

```
app.controller('myCtrl', function($scope, hexafy) {
```

```
    $scope.hex = hexafy.myFunc(255);
```

```
});
```

```
app.service('hexafy', function() {
    this.myFunc = function (x) {
        return x.toString(16);
    }
});
```

```
app.filter('myFormat',['hexify', function(hexify) {
    return function(x) {
        return hexify.myFunc(x);
    };
}]);
```

```
app.controller('myCtrl', function($scope, hexafy) {
    $scope.hex = hexafy.myFunc(259);
});
```

Angular Forms

Of course, there are also times when you need to add forms to your website or application so you could get information from your subscribers, just to make sure that they are not spambots. For this, the following input elements are used:

1. text area elements

2. button elements

3. select elements

4. input elements

You should also take note of the following:

1. *Ng-click* is used to invoke the reset () method directly but only when the user clicks the button.

2. The *reset ()* method makes master and user objects equals.

3. The *formCTRL* controller is all about controlling the forms that you have on your application or website.

4. The *ng-model* binds two user elements together.

5. The *ng-app* then defines the main Angular application.

To make this clearer, you could try the example given below:

```
<div ng-app="myFavoriteApp" ng-controller="formCtrl">
  <form novalidate>
    First Name:<br>
    <input type="text" ng-model="user.firstName"><br>
    Last Name:<br>
    <input type="text" ng-model="user.lastName">
    <br><br>
    <button ng-click="reset()">RESET</button>
  </form>
  <p>form = {{user}}</p>
  <p>master = {{master}}</p>
</div>

<script>
var app = angular.module('myFavoriteApp', []);
app.controller('formCtrl', function($scope) {
    $scope.master = {firstName: "Harry", lastName: "Potter"};
    $scope.reset = function() {
        $scope.user = angular.copy($scope.master);
    };
    $scope.reset();
});
</script>
```

Now, if you want to create *Select Boxes* for your forms, you could do that by trying the example below. These boxes are important because they allow people to choose what they want.

```
<form>
Select a topic:
<select ng-model="myVar">
   <option value="">
   <option value="stdnts">Students
   <option value="cats">Cats
   <option value="house">houses
</select>
</form>
```

Meanwhile, you could also put radio buttons by using the *ng-model* directive. This means that while what people see onscreen could have different values, only the selected items would reflect—making what's going on in the app easier to understand. For example, if your select boxes are:

```
<form>
Select a topic:
<select ng-model="myVar">
   <option value="">
   <option value="stdnts">Students
   <option value="cats">Cats
   <option value="house">houses
</select>
</form>
```

Then it means that your values should be based on the variables below:

```
 <form>
Pick a topic:
<input type="radio" ng-model="myVar" value="stdnts">Students
<input type="radio" ng-model="myVar" value="cats">cats
<input type="radio" ng-model="myVar" value="house">houses
</form>
```

As for checkboxes, it basically means that you want to know whether something has a value of *true* or *false*. When you apply the *ng-model* directive, the value will then show up in the application that you are using. In this case, you have to code like what you see below:

```
 <form>
   Check to show your favorite header:
   <input type="checkbox" ng-model="myVar">
</form>

<h1 ng-show="myVar">My Favorite Header</h1>
```

Angular Tables

And of course, to explain things better, you could also add tables to your applications or websites. The *ng-repeat* directive is essential here. To display data in a table, you could try doing the following:

```
<div ng-app="myFavoriteApp" ng-controller="studentsCtrl">

<table>
  <tr ng-repeat="x in names">
    <td>{{ x.Name }}</td>
    <td>{{ x.houses}}</td>
  </tr>
</table>

</div>

<script>
var app = angular.module('myFavoriteApp', []);
app.controller('customersCtrl', function($scope, $http) {
    $http.get("http://www.hogwartshouses.com/angular/customers.php")
    .then(function (response) {$scope.names = response.data.records;});
});
</script>
```

To keep your information in order, you have to use the *orderBy* directive. A good example could be seen below:

```
<table>
  <tr ng-repeat="x in names | orderBy : 'house'">
    <td>{{ x.Name }}</td>
    <td>{{ x.house }}</td>
  </tr>
</table>
```

You could also display CSS style elements on your table. To do it, you could try coding the elements below:

```
<style>
table, th , td {
  border: 1px purple;
  border-collapse: collapse;
  padding: 8px;
}
table tr:nth-child(odd) {
  background-color: #f1f1f1;
}
table tr:nth-child(even) {
  background-color: #ffffff;
```

```
}
</style>
```

And finally, to customize it even more, you could change letters to uppercase, enter table index, and use *$odd* and *$even* factors. Take a look at the examples below:

```
<table>
<tr ng-repeat="x in names">
<td ng-if="$odd" style="background-color:#f1f1f1">{{ x.Name }}</td>
<td ng-if="$even">{{ x.Name }}</td>
<td ng-if="$odd" style="background-color:#f1f1f1">{{ x.house }}</td>
<td ng-if="$even">{{ x.house }}</td>
</tr>
</table>
```

```
   <table>
  <tr ng-repeat="x in names">
    <td>{{ $index + 1 }}</td>
    <td>{{ x.Name }}</td>
    <td>{{ x.house }}</td>
  </tr>
</table>
```

```
<table>
  <tr ng-repeat="x in names">
    <td>{{ x.Name }}</td>
    <td>{{ x.house| uppercase }}</td>
  </tr>
</table>
```

Conclusion

Thank you again for downloading this book!

I hope this book was able to help you to understand Angular JS Programming—and that you now know how to create a basic application with it!

The next step is to practice coding by following what you've learned in this book. This way, you could master Angular JS Programming in just two weeks!

Finally, if you enjoyed this book, then I'd like to ask you for a favor, would you be kind enough to leave a review for this book on Amazon? It'd be greatly appreciated!

Bonus: Claim Your Free Bonus Books Now!

I'd like to thank you for taking time to read my book. As a token of my gratitude I'd like to offer you some of my #1 Best Seller Books for FREE!

You will also receive lots of great & free content in the future as well!! Simply click the link below and enter your name and email address to get your FREE content today!

Download Your FREE Books