Chapter 3

AngularJS

In the previous chapter we saw how to style your web page using Twitter Bootstrap to create menus and HTML tables. Although it was pretty simple to generate the page, it still involved a fair amount of HTML. In this chapter we’ll see how to simplify that HTML by using AngularJS and some simple JSON data structures to generate the HTML dynamically.

AngularJS allows you to easily bind your data with HTML to generate a web page on the client instead of on the server. As your data changes, Angular will automatically update the HTML to reflect the change. In addition, if you type something into a form which is bound to data, AngularJS will also update the internal data within the JavaScript as well as update any other HTML bound to the data. This goes beyond just generating a web page from a template and is referred to as two-way data binding. Let’s illustrate this with a simple “hello world” version of an AngularJS web page.

For this chapter we’ll be using examples in the 03\_angular project folder. As in the previous chapters you can download the source from <https://github.com/ddgarrett/beam-stack>. If you download the zip file from that web address or clone the Github project you’ll find a “03\_angular” project folder. If you then open that directory in WebStorm you can open a terminal window and type in the commands shown below. If you need help installing WebStorm, using the Terminal window, or running Node.js see Chapter 1 on Node.js. In a terminal window who’s current directory is the “03\_angular” project, type the following:

…03\_angular > npm install

…03\_angular > node 03\_server.js

You should see something like the WebStorm Terminal window shown below in figure 3.1 containing the message “Listening on port 3000”.

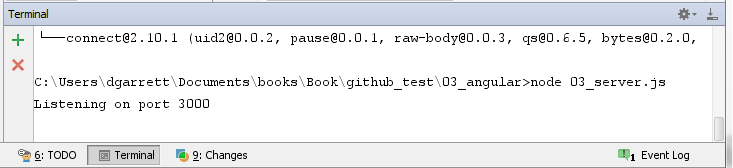


Figure 3.1 – Terminal Window after Starting Node.js

If everything is working correctly, you should then be able to navigate to <http://localhost:3000/03a_angular_world.html> and see the web page shown below in figure 3.2.

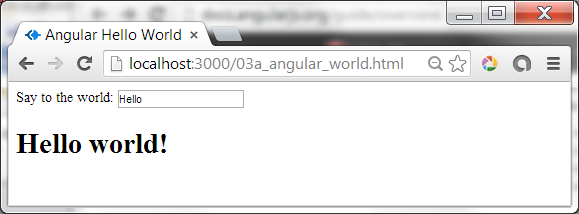


Figure 3.2 – AngularJS Hello World

Try typing something into the input field “Say to the world:” and you’ll see the text below the input field change as you type.



Figure 3.3 – AngularJS Two-Way Data Binding

This is the magic of AngularJS: two-way data binding, and declarative statements. We’ve talked about two-way data binding. A declarative statement simply refers to the fact that we declare what we want and not how to accomplish it. Let’s look at the source for 03a\_angular\_world.html to see how it accomplishes two way data binding and supports declarative statements.

If you open the file “03a\_angular\_world.html” in WebStorm you should see the window in figure 3.4. On line 2 of the example we see a line which tells Angular what part of the HTML it should be concerned about using an “**ng-app**” directive. In this case we’ve put the tag on the initial HTML tag, but it could be done just to the body or other portion of the page. AngularJS directives always being with **“ng-“.**

On line 6 we’ve added the link to the Content Delivery Network (CDN) source for our version of AngularJS We’ve also inserted a little JavaScript starting on line 10. Eventually we’ll move our custom code into a separate JavaScript file, but for now we’ve put the code inline for readability. This code defines the Angular “controller” named “HelloCntl” that we’ll be using for this page. All the controller is doing in this case is initializing the value of a variable named “$scope.say” on line 12 of the HTML source.

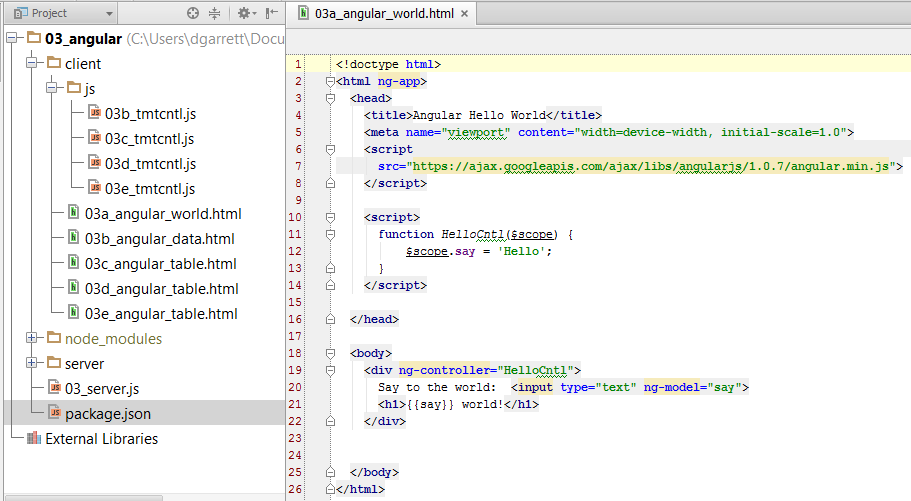


Figure 3.4 – Angular Hello World HTML Source

On line 19 we define a div whose content will be controlled by the “HelloCntl” controller using the tag ng-controller="HelloCntl". To complete the demo, on lines 20 and 21 we put the lines which allow us to input and display the variable. The Angular directive ng-model="say" defines the name of the attribute which will be used for the form input tag. On the following line, the tag “{{say}}” defines where the value of “say” should be displayed. The ng-model=”say” specifies two-way data binding, meaning that it not only displays the value of “say” but can also update the value. When updated, the new value of “say” is shown in the display of the tag “{{say}}”.

Try This

Change the value assigned to $scope.say to something else, even a non-string such as:

$scope.say = new Date();

Refresh the web page after you save your change. What happens?

Now that you’ve see the basics of how Angular works, let’s start defining some of the data structures that we’ll need for our example application, and see how Angular easily and elegantly handles repeating groups of data.

## Angular Data Structures

For our next example we’ll be adding JSON to our controller to define the data needed for our example application. Since this code will take more than a few lines of code we’ll be putting it into a separate JavaScript file. If you open the “js” folder within our project you should be able to see the source for our controller in the “03b\_tmtcnlt.js” file, as show in figure 3.5.

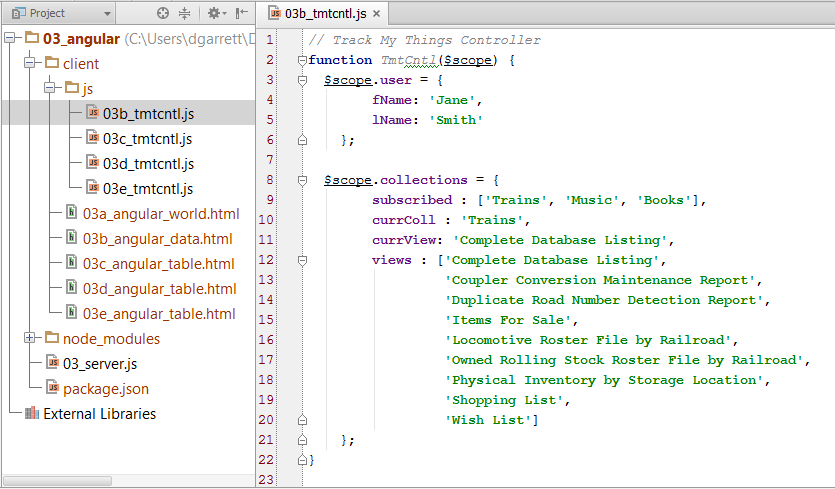


Figure 3.5 - 03b\_tmtcntl.js

For now we’ll be hard coding the data to be displayed. Eventually we’ll read this data from a MongoDB. At line 3 and line 8 we define our two data structures, “$scope.user” and “$scope.collections”. Note that in our JavaScript we prefix the name of the fields with “$scope”, but in the HTML we leave off “$scope”.

function TmtCntl($scope) {

$scope.user = {

fName: 'Jane',

lName: 'Smith'

};

$scope.collections = {

subscribed : ['Trains', 'Music', 'Books'],

currColl : 'Trains',

currView: 'Complete Database Listing',

views : ['Complete Database Listing',

…

'Wish List']

};

}

You can see what this data looks like when displayed in the HTML if you open the following web link <http://localhost:3000/03b_angular_data.html>. You should see something like the web page in figure 3.6.

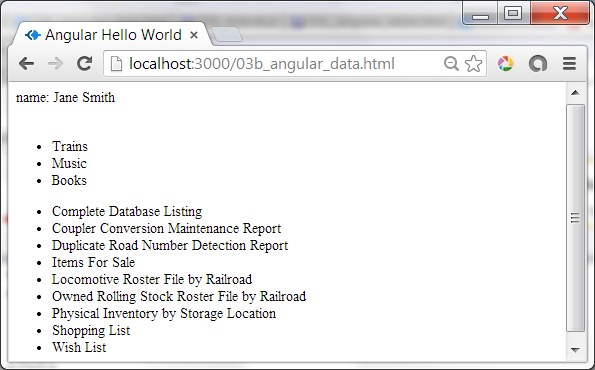


Figure 3.6 – Controller Defined Data Displayed on Web Page

If you look at the HTML source in “03b\_angular\_data.html”, as shown in figure 3.7, you’ll see that we displayed the user’s name using the code “name: {{user.fName}}{{user.lName}}”

Try This

Your customer has requested that you add an email address to the customer information. How would you do this? Try the following:

1. Add the new field to the definition of the customer information in 03b\_tmtcntl.js:

$scope.user = {

fName: 'Jane',

lName: 'Smith',

email: 'j.smith@nowhere.com'

};

2. Add the new field to the display in 03b\_angular\_data.html:

name: {{user.fName}} {{user.lName}}, email: {{user.email}}

Save your modifications then refresh the web page to see your changes. Not too bad! And as we’ll see later, with MongoDB it won’t be much more difficult to also add the new field to the database as well!

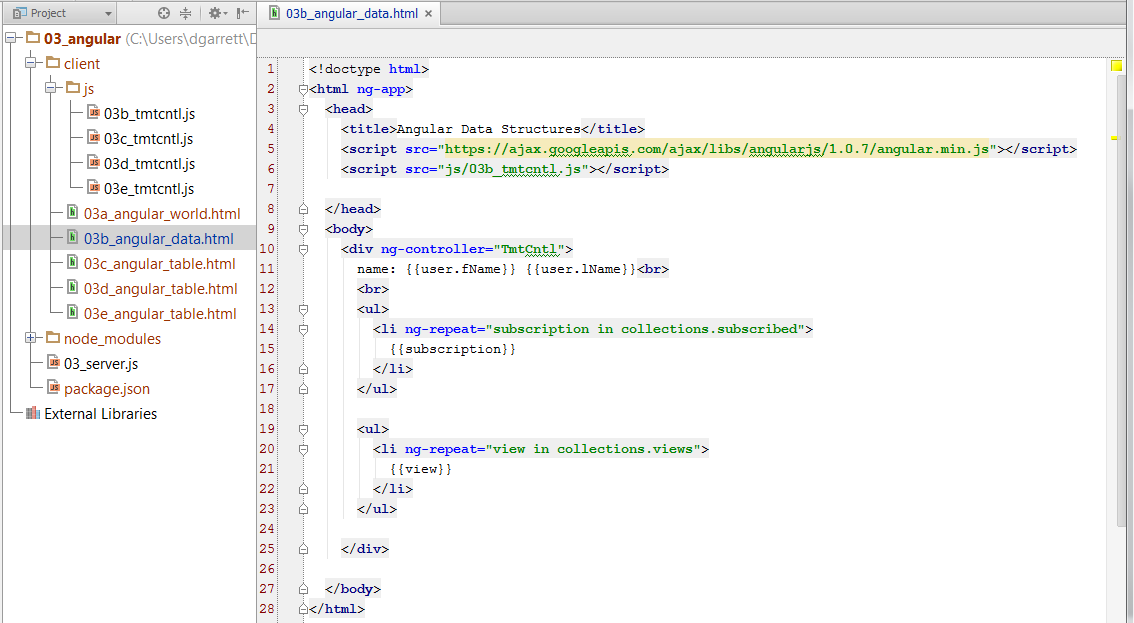


Figure 3.7 – 03b\_angular\_data.html Source

On line 14 we show how Angular displays repeating data using the “ng-repeat” directive to display a list item for each entry in the “collections.subscribed” array.

<li ng-repeat="subscription in collections.subscribed">

{{subscription}}

</li>

We then complete the web page by adding a list of views for the currently selected collection, starting at line 20.

<li ng-repeat="view in collections.views">

{{view}}

</li>

Not much HTML needed to display quite a bit of data. Later we’ll see how to read the data from a RESTful web service instead of coding it into the controller, but being able to code “example” data in the controller makes it easy for us to mock up user interfaces with realistic user data. It will also give us a better idea of what data will be needed from our server and how the data might be structured in the MongoDB.

## Angular Tables

Now that you have a basic understanding of how Angular displays data, let’s add the table showing the collection data. We’ll add back just enough of the Bootstrap libraries and classes covered in Chapter 2 to make the table more readable.

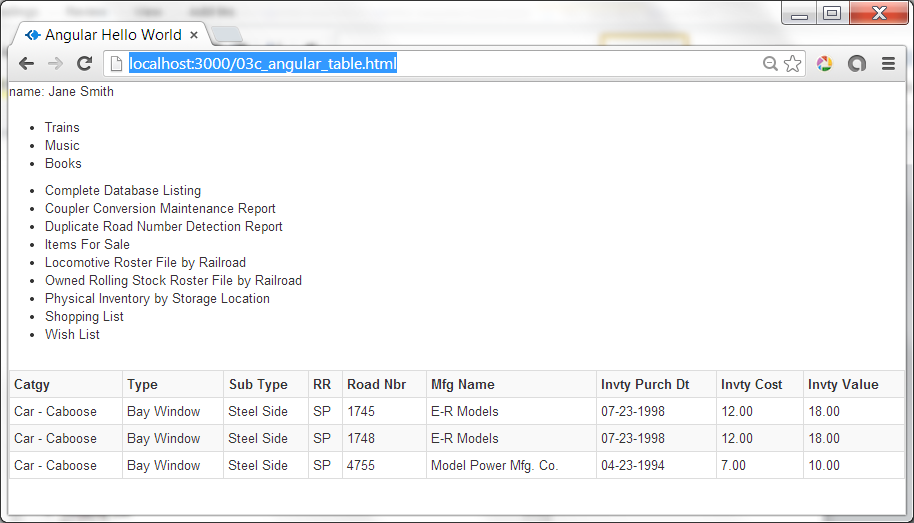


Figure 3.8 – Angular Display of Dynamically Defined Table

If you point your browser at <http://localhost:3000/03c_angular_table.html> you should see the Bootstrap table shown in Figure 3.8. To add this table to the web page we simply added the following as taken from lines 26 to 35 of the filed “03c\_angular\_table.html”. For clarity the code below shows the **Twitter Bootstrap** code in **blue** and the **AngularJS** code in **green**.

<table

class="**table table-striped table-bordered table-hover table-condensed**">

<tr>

<th **ng-repeat="field in collectionData.fields"**>

**{{field}}**

</th>

</tr>

<tr **ng-repeat="row in collectionData.rows"**>

<td **ng-repeat="field in row"**> **{{field}}** </td>

</tr>

</table>

If you look at the controller “03c\_tmtctl.js” in WebStorm you’ll see where the data is defined on line 23 of the controller. First we define the names to show at the top of the table in the “fields” attribute. We then define an array of data for each row in the table, as shown in figure 3.9.

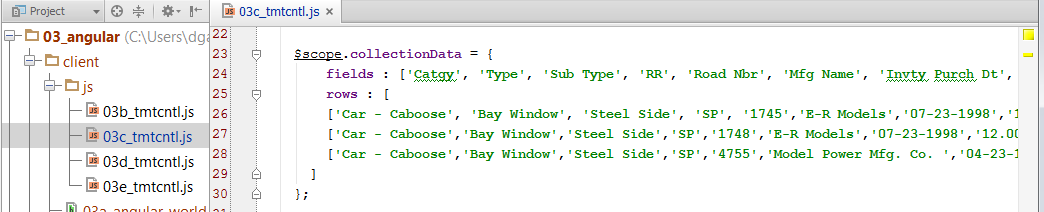


Figure 3.9 – Collection Data

## Angular Paging

Before we integrate the Angular code with the previously covered Bootstrap HTML classes, let’s add one more feature to simulate what paging would look like. We’ll also illustrate the use the Angular “ng-click” directive. If you point your browser at <http://localhost:3000/03d_angular_table.html> you’ll see the screen shown in Figure 3.10. We’ve added a “Page” button at the bottom left side of the web page to show how the table will appear as a user pages through multiple pages. We won’t actually be paging through multiple pages, but rather just alternating between two sets of data.

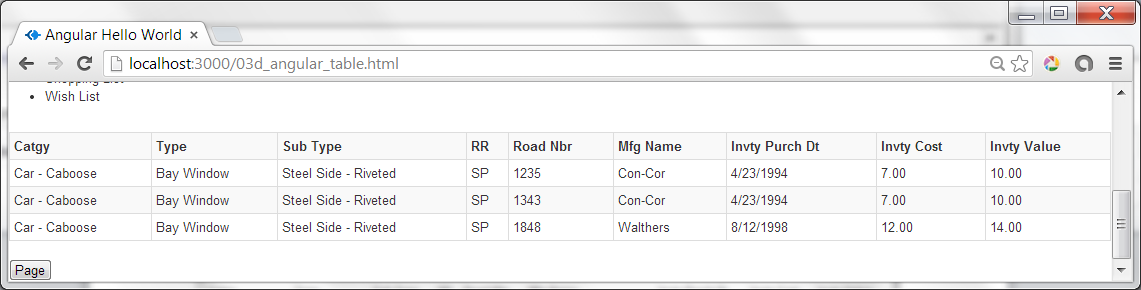


Figure 3.10 – Table with “Page” button

If you look at the HTML in “03**d**\_angular\_table.html” you’ll see a single new tag at line 37.

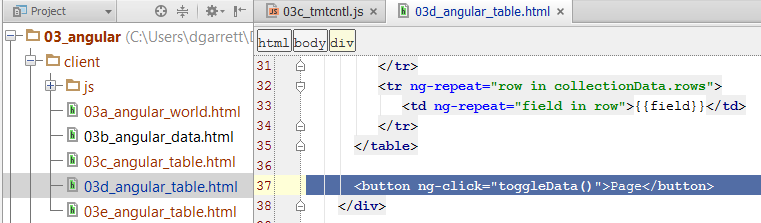


Figure 3.11 – 03d\_angular\_table.html Source

The “ng-click” directive allows us to call a function previously defined within the controller. In this case we’ve defined a function called “toggleData()” in the controller file “03d\_tmtcntl.js”. As previously mentioned, the controller is simulating paging by switching the display between two sets of data. The function defined at line 41 of the controller, the “toggleData” function, consists of code to swap the data being displayed between two sets of data.

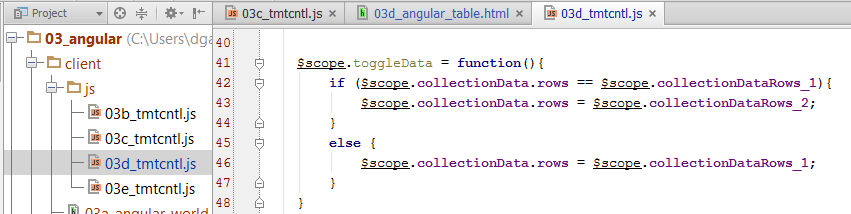


Figure 3.12 – 03d\_tmtcntl.js toggleData() Function

We define the two sets of data starting at line 24 with the following:

$scope.collectionDataRows\_1 = [

['Car - Caboose', 'Bay Window', 'Steel Side', 'SP', '1745','E-R Models','07-23-1998','12.00','18.00'],

['Car - Caboose','Bay Window','Steel Side','SP','1748','E-R Models','07-23-1998','12.00','18.00'],

['Car - Caboose','Bay Window','Steel Side','SP','4755','Model Power Mfg. Co. ','04-23-1994','7.00','10.00']

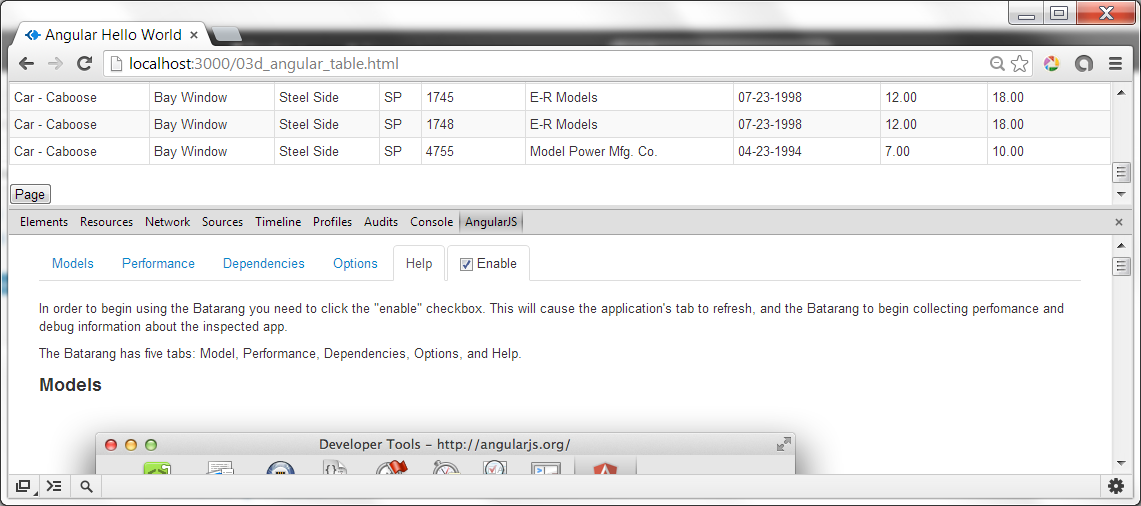
];

This illustrates the power of Angular. We only have to set the rows to another collection of data and Angular takes care of the rest, rebuilding the html dynamically for the entire table. As you might imagine, this will make it very simple for us to page through a collection using a web service to provide the next page of data, without having to re-read and regenerate the entire web page on the server.

## Debugging AngularJS App

As you might guess, once you begin reading data from a service it can be much more difficult to debug your Angular program when the wrong data appears. One popular tool available to help debug these issues as well as other Angular related problems is Batarang. Because Batarang runs only under Chrome, we’ll be covering how to debug your AngularJS app, and JavaScript in general, using the Chrome JavaScript Console. There is a Batarang GitHub site, <https://github.com/angular/angularjs-batarang>, which describes Batarang in more detail and also covers how to use it under the Apple and Windows. To install it from the Chrome store, go to <https://chrome.google.com/webstore/detail/ighdmehidhipcmcojjgiloacoafjmpfk>

Once you’ve installed Batarang in Chrome you can see the Batarang console using the Chrome “customize and control menu” (those three lines in the upper right part of the browser window). From that menu choose “Tools” and then “JavaScript console”. If you then choose the AngularJS tab, you should see a Batarang page similar to the one shown in on the bottom half of Figure 3.13.

Figure 3.13 – Chrome Batarang Extension for AngularJS

The first thing you’ll need to do is check an “Enable” checkbox to enable Batarang, as shown in figure 3.13. The help text shown in the initial window, or under the “AngularJS”, “Help” tab provides a good introduction to using Batarang. One of the tabs that I found very useful during the development of these examples was the “Models” tab. Figure 3.14 shows the Models tab for our last example web page, showing the data we defined in our controller. Although we can currently look at our data in the controller JavaScript, once we start reading the data from a server the Angular Models tab will be an easy way to examine what data has been received from the server.

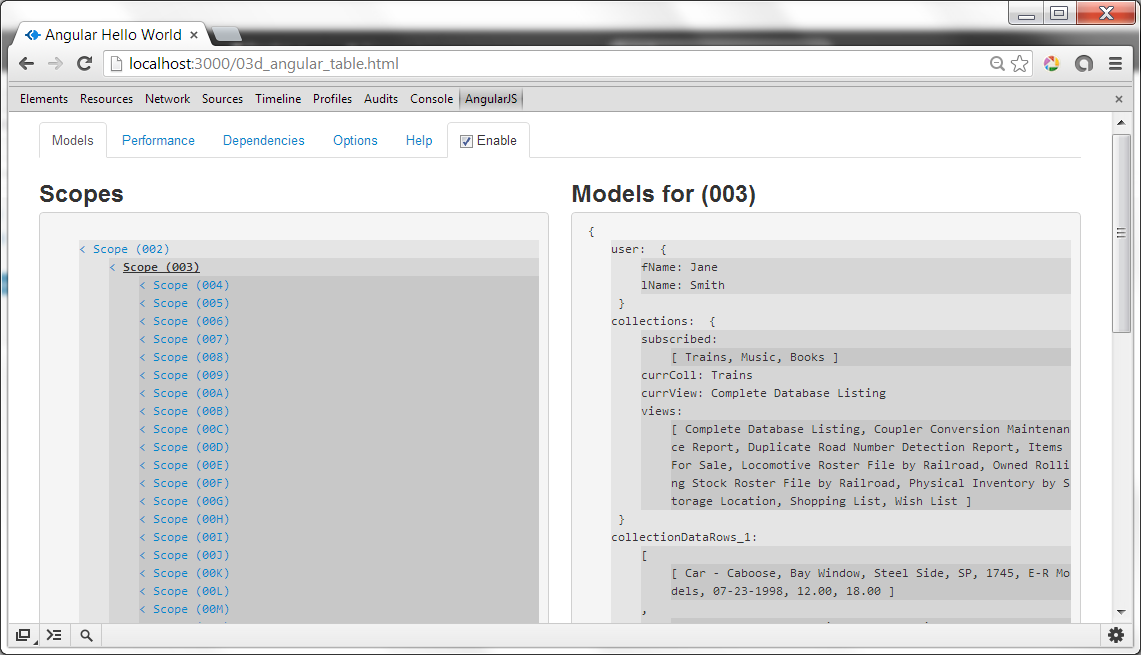


Figure 3.14 – Batarang Models Tab

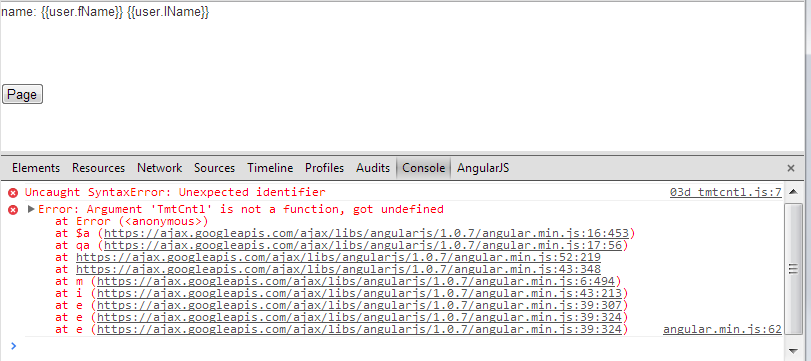
Try This

Introduce an error into your JavaScript. For the screen capture shown below we’ve modified “03d\_tmtcntl.js” to remove the comma (‘,’) after user.fName. Notice how the Angular bracketed, values such as “{{user.fName}}”, are not replaced with a value.

In the JavaScript console you’ll also see multiple errors. Usually you’ll want to focus on the first error since often the first error causes other errors as well.

What happens if you click the link to the right of the first error message, “03d\_tmtcntl.js:7”? Does this help fix the problem?

What do you see in the Angular tab and the display of the Angular Scopes?



## Angular Plus Bootstrap

Now let’s take all that we’ve learned about AngularJS and use it to generate the Bootstrap prototype that we developed in Chapter 2 but this time using the data defined in the Angular controller. If you point your browser to <http://localhost:3000/03e_angular_table.html> you’ll see the Bootstrap demo page from the last chapter, but now using Angular to generate the menus and table. You should see something like the web page shown in figure 3.15.

We’ve also started to add some of the actions that we’ll need to select a collection and a collection view from the menus. The pagination toolbar at the bottom of the page also toggles the table contents and doesn’t yet perform real paging. This amount of functionality can often be helpful while prototyping a new page with users. For many users this will give them a better idea of how the page works, and they can even test drive it, albeit to a very limited extent. This can often get across ideas much better than static drawings or static web pages, and we end up with a prototype that we can then extend to build a full blown application.

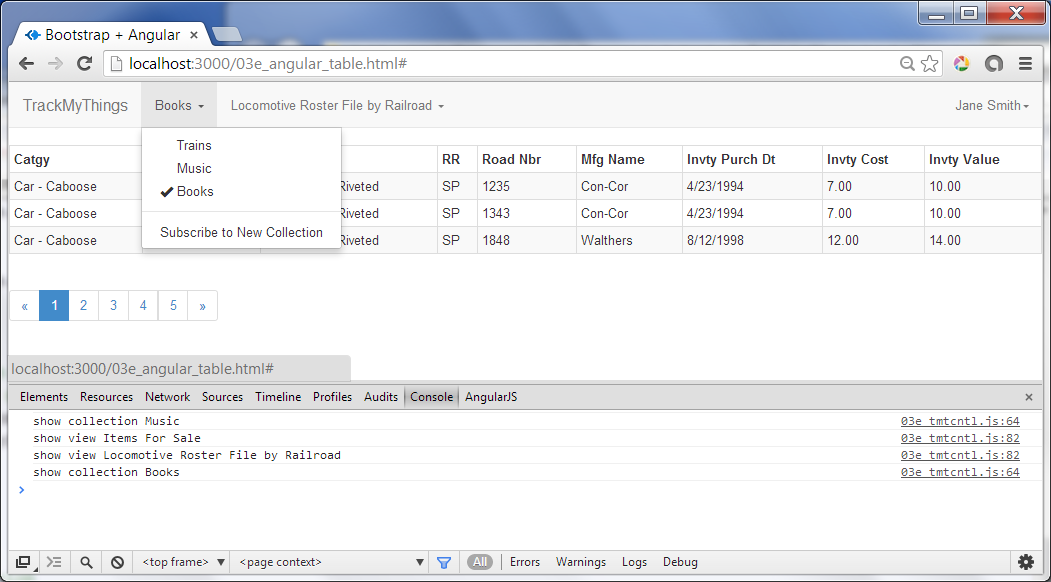


Figure 3.15 – Bootstrap Plus AngularJS

If you look at the HTML source file at “03e\_angular\_table.html”, most of it should look very familiar. For example, on line 31 we’ve taken the Bootstrap menu that we defined in Chapter 2 and replaced the multiple <li …> tags with a single Angular tag as shown below.

<li ng-repeat="subscription in collections.subscribed" >

<a href="#" ng-click="showCollection(subscription)" >

<span class="glyphicon glyphicon-ok

{{getSubsCollCheckmarkFont(subscription)}}">

</span>

{{subscription}}

</a>

</li>

Examining the tag in a bit more detail we see that:

1. We start the <li … tag with an “ng-repeat” directive, as we had in our previous Angular example.
2. We then have an “ng-click” directive which calls a JavaScript function “showCollection(subscription)” when the user selects the menu
3. We can also use a function when defining a substitution values, as we’ve done with “{{getSubsCollCheckmarkFont(subscription)}}”.
4. We complete the <li … tag with the “{{subscription}}” variable, as we had in our previous Angular example.

Let’s take a closer look at the two function calls contained in this tag. If you look at the JavaScript controller source at “03e\_tmtcntl.js” you’ll see the definitions for the two functions at lines 50 and 59. The “getSubsCollCheckmarkFont” is shown below. This function simply returns a blank or “font-white” if we want to either show or hide the checkmark in front of a menu option. We need to do this to maintain proper spacing in the displayed menu items.

// If the passed subscription is the "currColl" (current collection)

// return empty string, else return "font-white" to hide the menu Checkmark

$scope.getSubsCollCheckmarkFont = function(subscription){

if ($scope.collections.currColl === subscription){

return '';

}

return 'font-white'

}

The second new function “showCollection” is shown below. This function is just a “stub” which we’ll complete later once we’ve connected the application to MongoDB. For now, this function will set the current collection, and then write to the console.

// Show the specified collection

// TODO: finish this function

$scope.showCollection = function(subscription){

if (!($scope.collections.currColl === subscription)){

$scope.collections.currColl = subscription;

console.log('show collection ' + subscription);

}

}

We’ve added similar functions for the “Views” menu to display or hide the checkmark and to take action when a view is selected. You’ll find those functions on lines 68 and 77 of the controller. We’ll be adding to these functions as we go, but for now there’s quite a bit of “live feel” to the web page for just a couple of dozen more lines of code.

You’ve learned how to style a web page with Twitter Bootstrap and how to integrate that page with a data source using AngularJS. We now have the pieces of a functioning prototype even without having a database. In the next chapter we’ll show you how to define this data in a MongoDB. We’ll even begin to user Expres routes to being serving up some of that data in a RESTful way.

## Additional Angular Resources

There are a number of resources available on the web to help you learn more about AngularJS. Here’s a few from a list at <https://github.com/angular/angular.js>. There’s also a getting started overview at <http://docs.angularjs.org/misc/started>.

* Web site: [http://angularjs.org](http://angularjs.org/)
* Tutorial: <http://docs.angularjs.org/tutorial>
* API Docs: <http://docs.angularjs.org/api>
* Developer Guide: <http://docs.angularjs.org/guide>