**Top of Form**

**A Master Detail Example, Using JSON & AngularJS**

[**Michael Gledhill**](https://www.codeproject.com/script/Membership/View.aspx?mid=4555587), 18 Nov 2015

|  |  |
| --- | --- |
| https://codeproject.global.ssl.fastly.net/script/Ratings/Images/stars-fill-md.png  https://codeproject.global.ssl.fastly.net/script/Ratings/Images/stars-empty-md.png | 4.88 (37 votes) |
|  | |  |  |  | | --- | --- | --- | | Rate: | [vote 1vote 2vote 3vote 4vote 5](https://www.codeproject.com/Tips/837693/A-Master-Detail-Example-Using-JSON-AngularJS) |  | |

A simple example of using AngularJS to turn JSON into a beautiful Master-Detail view !

**Is your email address OK?** You are signed up for our newsletters but your email address is either unconfirmed, or has not been reconfirmed in a long time. Please **click**[**here**](https://www.codeproject.com/script/Membership/SendConfirmRequest.aspx?rp=%2fTips%2f837693%2fA-Master-Detail-Example-Using-JSON-AngularJS)**to have a confirmation email sent** so we can confirm your email address and start sending you newsletters again. Alternatively, you can [**update your subscriptions**](https://www.codeproject.com/script/Membership/Subscribe.aspx?rp=%2fTips%2f837693%2fA-Master-Detail-Example-Using-JSON-AngularJS).

* [**Download demo - 12.4 KB**](https://www.codeproject.com/KB/HTML/837693/AngularJSdemo.zip)

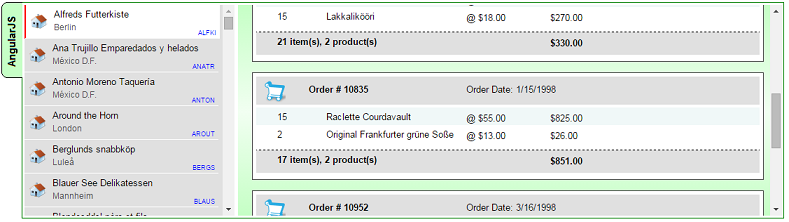
Introduction

In this short tutorial, I'm going to walk through turning the data from two boring *Northwind* JSON web services into the following Master-Detail view, using the power of *AngularJS*.

All of the CSS, HTML and JavaScript for this example can be downloaded using the download link above, you can view a "live demo" of how it looks on this page:

[Master Detail view](http://mikesknowledgebase.azurewebsites.net/pages/Services/WebServices-Page8.7.htm)

... and here's what the example will look like:



I'll be honest - there's nothing revolutionary here, just a few tips'n'tricks which you can learn from and use in your own code, and it's a nice example of how little code you need to write to create such a friendly, responsive display, from some JSON data.

Right, let's get started!

Ingredients

To create this Master-Detail view, we're going to need four ingredients:

1. Two JSON web services - one of them will fetch a list of all Customer records, and the other takes a Customer ID, and fetches a list of that customer's Orders, and the Products within that order

Here are examples of the web services which we'll use (you can click on these links, to see the JSON data which our example is based on):

* + <http://inorthwind.azurewebsites.net/Service1.svc/getAllCustomers>
  + <http://inorthwind.azurewebsites.net/Service1.svc/getBasketsForCustomer/ANATR>

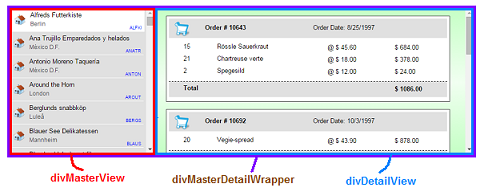
1. Some AngularJS / JavaScript code to load the data from each of these web services, and save it into variables, ready to be binded into our HTML controls
2. Some HTML
3. Some CSS to add styling to our web page and make it look funky

Two Side-by-side divs

On our web page, we are going to need *two*<div> controls: a Master view, where our list of customers will appear, and a Detail view, where a particular customer's list of orders will appear.

For both views, we'll load an array of records into an AngularJS variable, and leave AngularJS to do the hard work of creating a list of <div> controls, one per record.

To make the Master & Detail views appear side-by-side, we'll wrap them an outer div, and apply some CSS.



Here's the basic HTML for this (which we'll need to modify in a minute)..

Hide   Copy Code

<div id="divMasterDetailWrapper">

<div id="divMasterView"></div>

<div id="divDetailView"></div>

</div>

...and here's the CSS we'll need, to get the divs to appear side-by-side:

Hide   Copy Code

#divMasterDetailWrapper

{

width: 100%;

height: 300px;

border: 1px solid Green;

padding: 3px;

}

#divMasterView

{

width: 300px;

height: 300px;

background-color: #E0E0E0;

margin-right: 5px;

overflow-y: auto;

float: left;

}

#divDetailView

{

height: 300px;

padding: 0;

display: block;

overflow-y: auto;

}

(I'll be honest, I always forget the styling needed to get this working properly, so this section is for my own benefit !!)

The AngularJS Code

You can download the full AngularJS script from the download link at the top of this tip, but, for now, here's the first part of the code:

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

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

*// Force AngularJS to call our JSON Web Service with a 'GET' rather than an 'OPTION'*

*// Taken from: http://better-inter.net/enabling-cors-in-angular-js/*

myApp.config(['$httpProvider', function ($httpProvider) {

$httpProvider.defaults.useXDomain = true;

delete $httpProvider.defaults.headers.common['X-Requested-With'];

}]);

myApp.controller('MasterDetailCtrl',

function ($scope, $http) {

*// We'll load our list of Customers from our JSON Web Service into this variable*

$scope.listOfCustomers = null;

*// When the user selects a "Customer" from our MasterView list, we'll set this variable.*

$scope.selectedCustomer = null;

$http.get('http://inorthwind.azurewebsites.net/Service1.svc/getAllCustomers')

.success(function (data) {

$scope.listOfCustomers = data.GetAllCustomersResult;

*// If we managed to load more than one Customer record, then select the*

*// first record by default.*

$scope.selectedCustomer = $scope.listOfCustomers[0].CustomerID;

*// Load the list of Orders, and their Products, that this Customer has ever made.*

$scope.loadOrders();

}

Our first problem is that our two JSON web services use the "GET" protocol, but by default, AngularJS will attempt to call them using the "OPTIONS" protocol, so will fail miserably to fetch our data.

To fix this, we need the following few lines, from this site:  
<http://better-inter.net/enabling-cors-in-angular-js/>

Hide   Copy Code

*// Force AngularJS to call our JSON Web Service with a 'GET' rather than an 'OPTION'*

*// Taken from: http://better-inter.net/enabling-cors-in-angular-js/*

myApp.config(['$httpProvider', function ($httpProvider) {

$httpProvider.defaults.useXDomain = true;

delete $httpProvider.defaults.headers.common['X-Requested-With'];

}]);

The code defines a new *AngularJS* controller called MasterDetailCtrl which contains a couple of variables, one for storing our list of customer records, and one to store the ID of which customer record is currently selected.

It then calls our JSON web service (using the "GET" protocol), and stores the results in a listOfCustomersvariable.

The web service to fetch all of our customer records returns JSON data like this:

Hide   Copy Code

{

GetAllCustomersResult: [

{

City: "Berlin",

CompanyName: "Alfreds Futterkiste",

CustomerID: "ALFKI"

},

{

City: "México D.F.",

CompanyName: "Ana Trujillo Emparedados y helados",

CustomerID: "ANATR"

},

Notice how (in this particular example) we actually need to set the listOfCustomers variable to the data.GetAllCustomersResult value, as the array of customer records lives there.

Hide   Copy Code

$http.get('http://inorthwind.azurewebsites.net/Service1.svc/getAllCustomers')

.success(function (data) {

$scope.listOfCustomers = data.GetAllCustomersResult;

Next, the Angular code grabs the *first*customer record, and calls a "loadOrders" function, to load that customer's orders.

Hide   Copy Code

$scope.loadOrders = function () {

*// The user has selected a Customer from our Drop Down List. Let's load this Customer's records.*

$http.get('http://inorthwind.azurewebsites.net/Service1.svc/getBasketsForCustomer/' + $scope.selectedCustomer)

.success(function (data) {

$scope.listOfOrders = data.GetBasketsForCustomerResult;

})

.error(function (data, status, headers, config) {

$scope.errorMessage = "Couldn't load the list of Orders, error # " + status;

});

}

As you can see, all this does is call our second JSON web service, and stores the results in a listOfOrdersvariable.

And, shockingly, *that's it* for the JavaScript.

Our JavaScript coding work is done.

Well, almost. We *will*need to come back to write a couple of small function to calculate the "totals" for each order later, but we'll get to that later.

The HTML

Now, it's time to make the link between our Angular code, and our webpage.

First, we need to tell our web page that it's going to be using the myApp module in our Angular code.

Hide   Copy Code

<html ng-app='myApp' >

Our webpage needs to include a link to the AngularJS library. You either download a copy from Angular's website:

<https://angularjs.org/>

Or you can include this line in your code, to include the current version (v1.2.26):

Hide   Copy Code

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.26/angular.min.js"></script>

Our Master & Detail views will both use data from the MasterDetailCtrl control in our myApp Angular module, so we need to add the ng-controller directive to our outer <div>.

Hide   Copy Code

<div id="divMasterDetailWrapper" ng-controller='MasterDetailCtrl'>

To populate the Master view with a list of customers, we need to change our divMasterView to use the ng-repeat directive, to get AngularJS to iterate through our listOfCustomers array, and create a set of <div>controls for each item it finds:

Take a deep breath:

Hide   Copy Code

<div id="divMasterView">

<div id="{{customer.customerid}}" ng-repeat='Customer in listOfCustomers'

class="cssOneCompanyRecord ng-class="{cssCompanySelectedRecord:

Customer.CustomerID == selectedCustomer}" ng-click="selectCustomer(Customer);">

<div class="cssCompanyName">{{Customer.CompanyName}}</div>

<div class="cssCompanyCity">{{Customer.City}}</div>

<div class="cssCustomerID">{{Customer.CustomerID}}</div>

<img src='/images/icnOffice.png' class="cssCustomerIcon" />

</div>

</div>

Wow. That's quite a mouthful.

The key to this is the ng-repeat directive in the first inner div.

Hide   Copy Code

ng-repeat='Customer in listOfCustomers'

This tells AngularJS to iterate through the records in the listOfCustomers variable, and for each one, it'll create a set of the controls within that <div>:

Hide   Copy Code

<div class="cssCompanyName">{{Customer.CompanyName}}</div>

<div class="cssCompanyCity">{{Customer.City}}</div>

<div class="cssCustomerID">{{Customer.CustomerID}}</div>

<img src='/images/icnOffice.png' class="cssCustomerIcon" />

We also used the ng-click directive on our customer <div>s...

Hide   Copy Code

ng-click="selectCustomer(Customer);"

...so when the user clicks on one a customer <div>s, we can call a small function to set that CustomerID as the being the "selected" customer...

Hide   Copy Code

$scope.selectCustomer = function (val) {

$scope.selectedCustomer = val.CustomerID;

$scope.loadOrders();

}

As I said before, our JSON web service returns an array of Customer records, each of which looks like this:

Hide   Copy Code

{

City: "Berlin",

CompanyName: "Alfreds Futterkiste",

CustomerID: "ALFKI"

}

So, you can see how the ng-repeat takes the Customer record, and can bind to its three properties.

To better understand this code, you can run this example and look at the ng-repeat's output using Google Chrome:



Compare that with the ng-repeat <div> element:

Hide   Copy Code

<div id="{{customer.customerid}}"

ng-repeat='Customer in listOfCustomers'

class="cssOneCompanyRecord

ng-class="{cssCompanySelectedRecord: Customer.CustomerID == selectedCustomer}"

ng-click="selectCustomer(Customer);">

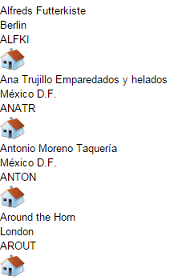
Each of the <div>s has been given the class cssOneCompanyRecord, an ID assigned to it, which is the customer's ID, then we have three further <div>s and an <img>, with various other values from that customerrecord.

How cool is that !

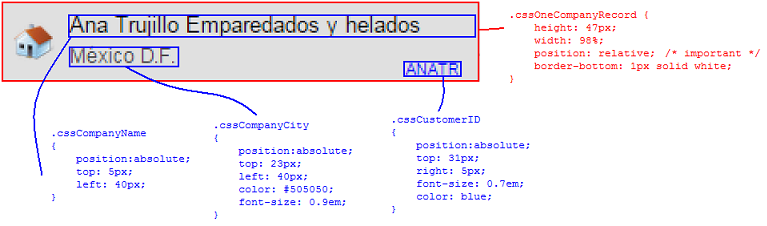
The CSS

Now, the ingredient we haven't mentioned so far is the CSS. But, hey, does this *really*matter ?

Well, yes. Without the correct CSS, our ng-repeat will create exactly the same set of <div>s which Chrome showed us above, but they'll just appear one after another, each on a separate line:



So, how do we create one "outer" <div> per customer, of a fixed width & height, then a set of inner <div>s, where we precisely position the customer name, ID, city name & our "house" icon ?



This is a very cool CSS trick.

If you set an "outer" <div> to have "position:relative", then you can insert "inner" <div>s into it, set their position to absolute, and position them at exact (x,y) positions within the <div>.

So, using CSS, we can set the Company Name to position (40, 5) within the outer customer <div>, the City name at position (40, 23), and so on.

The Master View

We use exactly the same "tricks" in the Master view.

The Master view (right-hand view) works in exactly the same way as the Details view, but this time, we have *two*ng-repeat controls within each other.

We need to iterate through the list of Orders for a particular customer and *then*iterate through the list of Products in each Order.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

<div id="divDetailView">

<div id="Order\_{{Order.OrderID}}"

ng-repeat="Order in listOfOrders" class="cssOneOrderRecord">

<div class="cssOneOrderHeader">

<div class="cssOrderID">Order # {{Order.OrderID}}</div>

<div class="cssOrderDate">Order Date: {{Order.OrderDate}}</div>

</div>

<div class="cssOneProductRecord"

ng-repeat='Product in Order.ProductsInBasket'

ng-class-odd="'cssProductOdd'"

ng-class-even="'cssProductEven'" >

<div class="cssOneProductQty">{{Product.Quantity}}</div>

<div class="cssOneProductName">{{Product.ProductName}}</div>

<div class="cssOneProductPrice">@ {{Product.UnitPrice | currency}}</div>

<div class="cssOneProductSubtotal">

{{Product.UnitPrice \* Product.Quantity | currency}}</div>

</div>

<div class="cssOneOrderTotal">

<div class="cssOneProductQty">

{{Order.ProductsInBasket|countItemsInOrder}} item(s),

{{Order.ProductsInBasket.length}} product(s)

</div>

<div class="cssOneProductSubtotal">

{{Order.ProductsInBasket|orderTotal | currency}}

</div>

</div>

</div>

</div>

Did you notice the nested ng-repeats ?

Hide   Copy Code

<div id="divDetailView">

<div id="Order\_{{Order.OrderID}}"

ng-repeat="Order in listOfOrders" class="cssOneOrderRecord">

...

<div class="cssOneProductRecord"

ng-repeat='Product in Order.ProductsInBasket >

...

</div>

...

</div>

</div>

Once again, we use the CSS trick of applying...

Hide   Copy Code

position: relative

...to the "outer" <div> controls - then *absolutely*position the inner <div> controls.

Once again, the exact format of the variables we're binding to must exactly match the field names returned from our JSON web service.

Here's a typical order for one of our customers:

Hide   Copy Code

GetBasketsForCustomerResult: [

{

OrderDate: "9/18/1996",

OrderID: 10308,

ProductsInBasket: [

{

ProductID: 69,

ProductName: "Gudbrandsdalsost",

Quantity: 1,

UnitPrice: 36

},

{

ProductID: 70,

ProductName: "Outback Lager",

Quantity: 5,

UnitPrice: 15

}

]

},

You can see how this corresponds with the Product field names in our HTML:

Hide   Copy Code

<div class="cssOneProductRecord" ng-repeat='Product in Order.ProductsInBasket'

ng-class-odd="'cssProductOdd'" ng-class-even="'cssProductEven'" >

<div class="cssOneProductQty">{{Product.Quantity}}</div>

<div class="cssOneProductName">{{Product.ProductName}}</div>

<div class="cssOneProductPrice">@ {{Product.UnitPrice | currency}}</div>

<div class="cssOneProductSubtotal">{{Product.UnitPrice \* Product.Quantity | currency}}</div>

</div>

Maintainability

One downside of using AngularJS, and JavaScript itself, is that this binding makes sense to us *now*, as we're working through it, and writing the code.

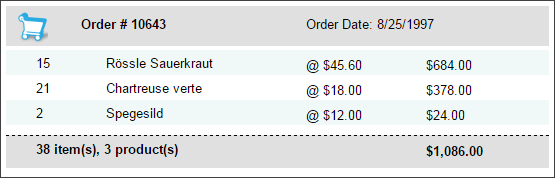
But if someone was to pick up this code in a year's time, it doesn't give them much help as to what that "Product" thing refers to. They can see from this HTML that the Product object has a ProductName, Quantityand UnitPrice to bind to. but what other fields are available? And would they really know what to do if they wanted to add a ProductCode field?

I often copy examples of the JSON records which I'm expecting from my web services, and paste them, as comments, directly into my AngularJS class, so other developers are able to see at a glance what fields are available.

It really adds to the maintainability and readability of the code (although, of course, you need to keep such comments up-to-date, if the web service changes).

Useful AngularJS tricks - Alternating Classes

In the list of products in each order, we alternate the CSS classes, so every other row appears with a light blue background:



This is a cool feature of ng-repeat directive:

Hide   Copy Code

<div class="cssOneProductRecord"

ng-repeat='..' ng-class-odd="'cssProductOdd'"

ng-class-even="'cssProductEven'" >

We use the ng-class-odd and ng-class-even directives, which lets us apply one CSS class to odd-numbered rows in our ng-repeat, and a different CSS class to the even numbered rows. But we *also*have a regular class declared, so *all*of the rows *also*receive the cssOneProductRecord class.

Useful AngularJS Tricks - Selected Item

As I said earlier, when the user clicks on a customer record, we want that record to be shown as "selected".

Without AngularJS, this would involve writing a bit of code to add a particular CSS class to the <div> that they clicked on, and to make sure no other customers are *also* shown with this class.

*With*AngularJS, it's easier than that.

Thanks to our selectCustomer function, we *know*the ID of our selected customer (we store it in the $scope.selectedCustomer variable), so in our ng-repeat which iterates through our customer record, we simply need to use an ng-class directive:

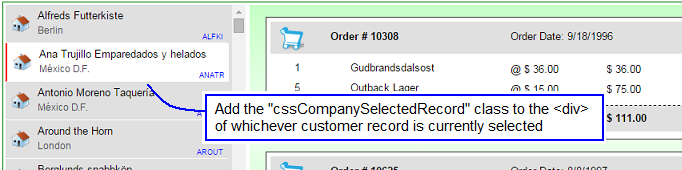
Hide   Copy Code

<div ng-repeat='Customer in listOfCustomers'

ng-class="{cssCompanySelectedRecord: Customer.CustomerID == selectedCustomer}"

...

I also fell off my chair the first time I saw this work.



ng-class doesn't just decide whether to add the cssCompanySelectedRecord class whilst *creating* our list of customer <div>s, but, thanks to the power of binding, it keeps it up-to-date, adding and removing the class to our <div>s as customers become selected and unselected, all based on our selectedCustomer variable.

Useful AngularJS Tricks - Totals

I had a few issues creating the "totals row" which gets shown at the bottom of each order.

Hide   Copy Code

<div class="cssOneOrderTotal">

<div class="cssOneProductQty">

{{Order.ProductsInBasket|countItemsInOrder}} item(s), {{Order.ProductsInBasket.length}} product(s)

</div>

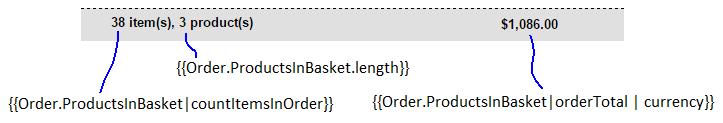
<div class="cssOneProductSubtotal">

{{Order.ProductsInBasket|orderTotal | currency}}

</div>

</div>

There are three totals in use here.



The second of these totals is the easiest.  
We can just *count*how many Product records are in each Order.

Hide   Copy Code

{{Order.ProductsInBasket.length}} product(s)

For the other two totals, we need to write a little code.

Here's how we create a total of the total number of individual items in this order.

Hide   Copy Code

myApp.filter('countItemsInOrder', function () {

return function (listOfProducts) {

*// Count how many items are in this order*

var total = 0;

angular.forEach(listOfProducts, function (product) {

total += product.Quantity;

});

return total;

}

});

And here's a similar function to calculate the total value of the order:

Hide   Copy Code

myApp.filter('orderTotal', function () {

return function (listOfProducts) {

*// Calculate the total value of a particular Order*

var total = 0;

angular.forEach(listOfProducts, function (product) {

total += product.Quantity \* product.UnitPrice;

});

return total;

}

});

An alternative was to do this is to use a more generic "sumByKey" function, which you'll find in numerous other articles on the internet, such as this one:

* <https://gist.github.com/Cacodaimon/7309268>

I chose to write these non-generic functions (i.e. which would need modifying if I wanted to sum JSON records with different field names) for maintainability. If I wanted to use this countItemsInOrder on a different screen, I wouldn't need to quote the JSON field name again.

Either method is fine though.

Summary

All of the CSS, HTML, and JavaScript files are attached, and you can view a "live" version of this Master-Detail view on this page:

* <http://mikesknowledgebase.azurewebsites.net/pages/Services/WebServices-Page8.7.htm>

This site also describes how I created the JSON web services used in this demo, and gives examples of using such data in jqGrid, and in an iPhone app using XCode.

AngularJS is an incredibly powerful library, and I've only scratched the surface of what it can do in this tip. You could also easily add a search facility, to search for customers of a particular name, and there are plenty of well-written articles out there to demonstrate how to do this.

I hope this article has nudged you to experiment with AngularJS, and see what it's capable of.

Don't forget to leave a rating at the top of this tip, to say if you found it useful.

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License (CPOL)](http://www.codeproject.com/info/cpol10.aspx)

Share

[**Michael Gledhill**](https://www.codeproject.com/Members/MikeGledhill)

|  |  |
| --- | --- |
| [Google+](https://plus.google.com/+MikeGledhill) [LinkedIn](https://ch.linkedin.com/in/mikegledhill) | Software Developer Partner Reinsurance |
| Switzerland Switzerland |

I'm a software developer, working with Visual Studio and XCode, working for a financial company.  
  
Author of the PartnerReSearch iPad app, winner of a "Business Insurance Innovation Award" in 2013, and a TechAward2014 "Innovation of the year" award in 2014.  
  
Living near Zurich, Switzerland, with my Swiss-English family.

Bottom of Form