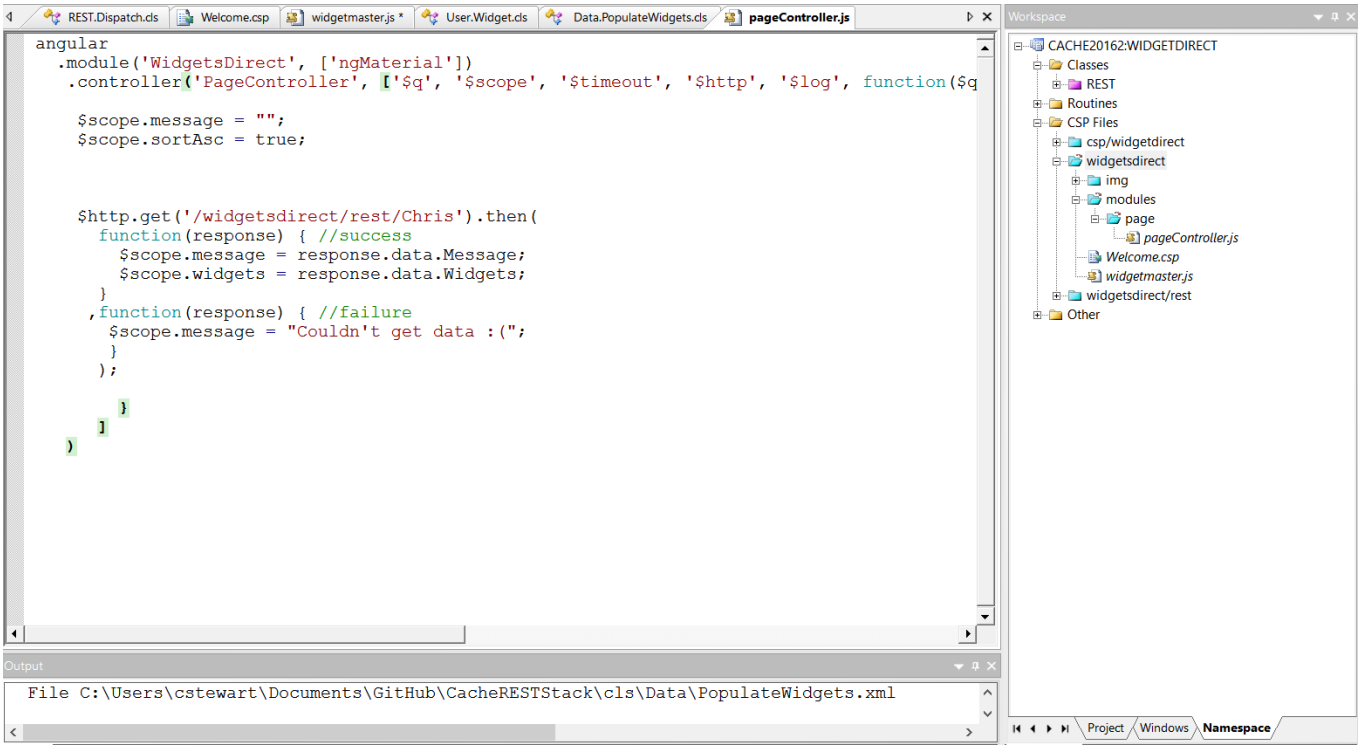
Article
Chris Stewart · Apr 24, 2017 4m read

Let's write an Angular 1.x app with a Caché REST backend - Part 7

or *"Things are going to break"*

We left our application over the weekend, secure in the knowledge that it was returning data from our primary persistent class, User.Widget. However, Widgets Direct are the premier supplier of both Widgets AND Widget Accessories, so we should really start working on adding these Accessories to our application.

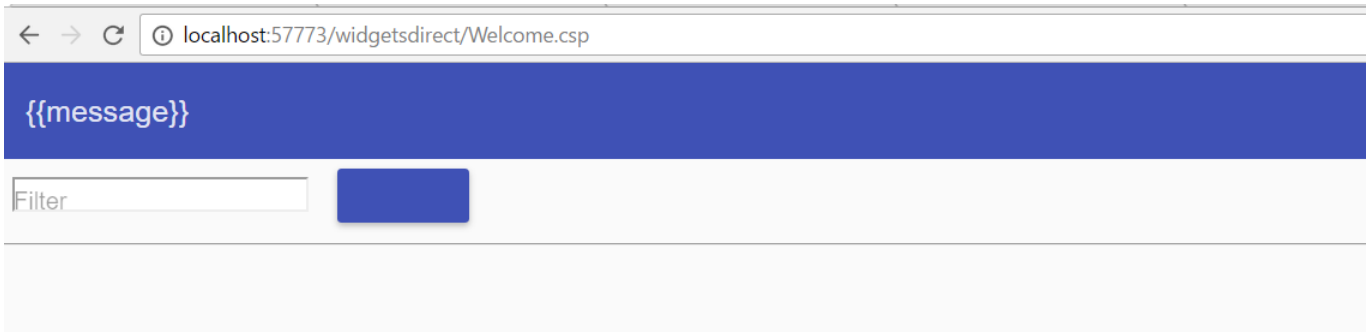
We should do some housekeeping first though. Our Page Controlller code is currently sitting in the widgetmaster.js file. As we start to build up our application and use multiple controllers, this will make the PageController hard to find, so we should refactor it into a sensible location and file name. So let's create *modules/page/PageController.js* under our web application, and paste the code in there. We can then remove the controller code from widgetmaster.js



REST.Dispatch.cls | Welcome.csp | **widgetmaster.js** | User.Widget.cls

```
angular
  .module('WidgetsDirect', ['ngMaterial'])
```

Let's save and reload our application to make sure everything works



Well, this clearly isn't good. How can we find out what went wrong though? As this is all client side code, we're not going to find any errors on the server. Instead, we need to press F12 to open our browser's debugger (these examples use Chrome, which is my personal preference as far as debuggers go, but all the major browsers have an equivilant). All errors in the runtime will log to the Console, so find this in your debugger. You may need to reload the page to trigger the error again.

```
▶Error: [ng:areq] http://errors.angularjs.org/1.4.8/ng/areq?p0=PageCont  angular.js:12520
roller&p1=not%20a%20function%2C%20got%20undefined
    at angular.js:38
    at qb (angular.js:1815)
    at Qa (angular.js:1825)
    at angular.js:9158
    at w (angular.js:8228)
    at D (angular.js:8270)
    at g (angular.js:7731)
    at g (angular.js:7734)
    at angular.js:7611
    at angular.js:1681
```

The Angular framework very helpfully includes a link to the documentation to unpack any errors returned. This error is telling us that it can not find the function definition of PageController in the Angular runtime. We just refactored it into a new file, so why can't Angular see it? Did we add the new script to the CSP page, so that Angular was able to access it?

```
        <hr>
<!-- AngularJS Libraries -->
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.m
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-an
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-ar
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-me
<!--AngularMaterial Library -->
<script src="http://ajax.googleapis.com/ajax/libs/angular_material/1.1.0/ang
<!--User Libraries -->
<script src="widgetmaster.js"></script>
<script src="modules/page/pageController.js"></script>
</body>

</html>
```

So we have learned a new lesson. The parent page must reference all scripts containing content we are trying to use. Once we add the new pagecontroller.js as a reference, we can reload our page successfully.

Onto the accessories. Accessory information is held in the User.WidgetAccessory class, and this is linked to the User.Widget class as a many to many relationship, through the use of a bridge class (please refer to the Many to Many section of the Caché documentation for more info, I'm not going to cover this in depth here). Our accessory class has a number of properties, and our bridge class has just 2, one link to Widget, and one link to Accessory, each with an Inverse property.

Welcome.csp | widgetmaster.js | User.Widget.cls | Data.PopulateWidgets.cls | pageController.js | **User.WidgetAccessory.cls ***

```
/// Class to hold our widget accessories
Class User.WidgetAccessory Extends %Persistent
{
Property Name As %String;

Property Description As %String(MAXLEN = 500);

Property Price As %Currency;

Property Quantity As %Integer;

Property FirstManufactured as %Date;

Property InProduction as %Boolean;

Property Import as %Boolean;

Property SKU as %String;

}
```

```
Class User.WidgetAccessoryLink Extends %Persistent
{

Relationship Widget As User.Widget [ Cardinality = one, Inverse = Accessories ];

Relationship Accessory As User.WidgetAccessory [ Cardinality = one, Inverse = Widgets ];

Index WidgetIndex On Widget;

Index AccessoryIndex On Accessory;
```

To start with, we would like to output all compatible accessories with our Widgets, so we need to add these as part of our toJSON() method, as an array. We can do this very neatly, by iterating over the relationship using GetNext to return all Accessory objects, and return the toJSON output from each. We can then push this output onto a JSON array, and attach this to a property of the Widget JSON.

```
Method toJSON() As %String
  {

      set jsonReturn = {}
      set jsonReturn.Id                  = ..%Id()
      set jsonReturn.Name                = ..Name
      set jsonReturn.Description         = ..Description
      set jsonReturn.Price               = ..Price
      set jsonReturn.Quantity            = ..Quantity

      set accessorykey = ""
      set accessoryList = []
        Do {
            set accLink = ..Accessories.GetNext(.accessorykey)
            If (accLink '= "") { do accessoryList.%Push(accLink.Accessory.toJSON()) }
        } While (accessorykey '= "")
        set jsonReturn.Accessories = accessoryList

      quit jsonReturn
  }

Storage Default
```

Let's compile this, and reload our page to check that everything is still rendering OK.

Well, this clearly isn't good. That's the error message we put in to trap any failures from the REST Service. This means we have an error on the server. However, we should still start with our trusty friend, the F12 debugger. This will display the error **as it was returned to the client.** This may not seem any more useful than loading the service using a REST debugger, but when the setup becomes more complex (with Authentication, for example), it can be very useful to trap the exact failure scenario that the client recorded. So, we load up the debugger, and check the console.

We have an error on the GET. In the Chrome debugger we can actually click this link to take us to the Timeline view, which will show all server calls, and highlight the failed calls in red. We can then click this link to get the error message returned from the server.

Developer Tools - http://localhost:57773/widgetsdirect/Welcome.csp

Elements Console Sources Network Timeline Profiles Application Security

View: Preserve log Disable cache Offline No throttling

Filter Regex Hide data URLs All XHR JS CSS Img Media Font Doc WS

5 ms 10 ms 15 ms 20 ms 25 ms 30 ms 35 ms 40

Name	Status
Welcome.csp	200
angular-material.min.css	200
logo.svg	200
angular.min.js	200
angular-animate.min.js	200
angular-aria.min.js	200
angular-messages.min.js	200
angular-material.min.js	200
widgetmaster.js	200
pageController.js	200
Chris	500

11 requests | 1.9 KB transferred | Finish: 763 ms

Clicking on the red link gives us

```
▼{errors: [{code: 5002, domain: "%ObjectErrors",…}],…}
  ▶errors: [{code: 5002, domain: "%ObjectErrors",…}]
   summary: "ERROR #5002: Cache error: <METHOD DOES NOT EXIST>ztoJSON+11^User.Widget.1 *toJSON,User.WidgetAccessory"
```

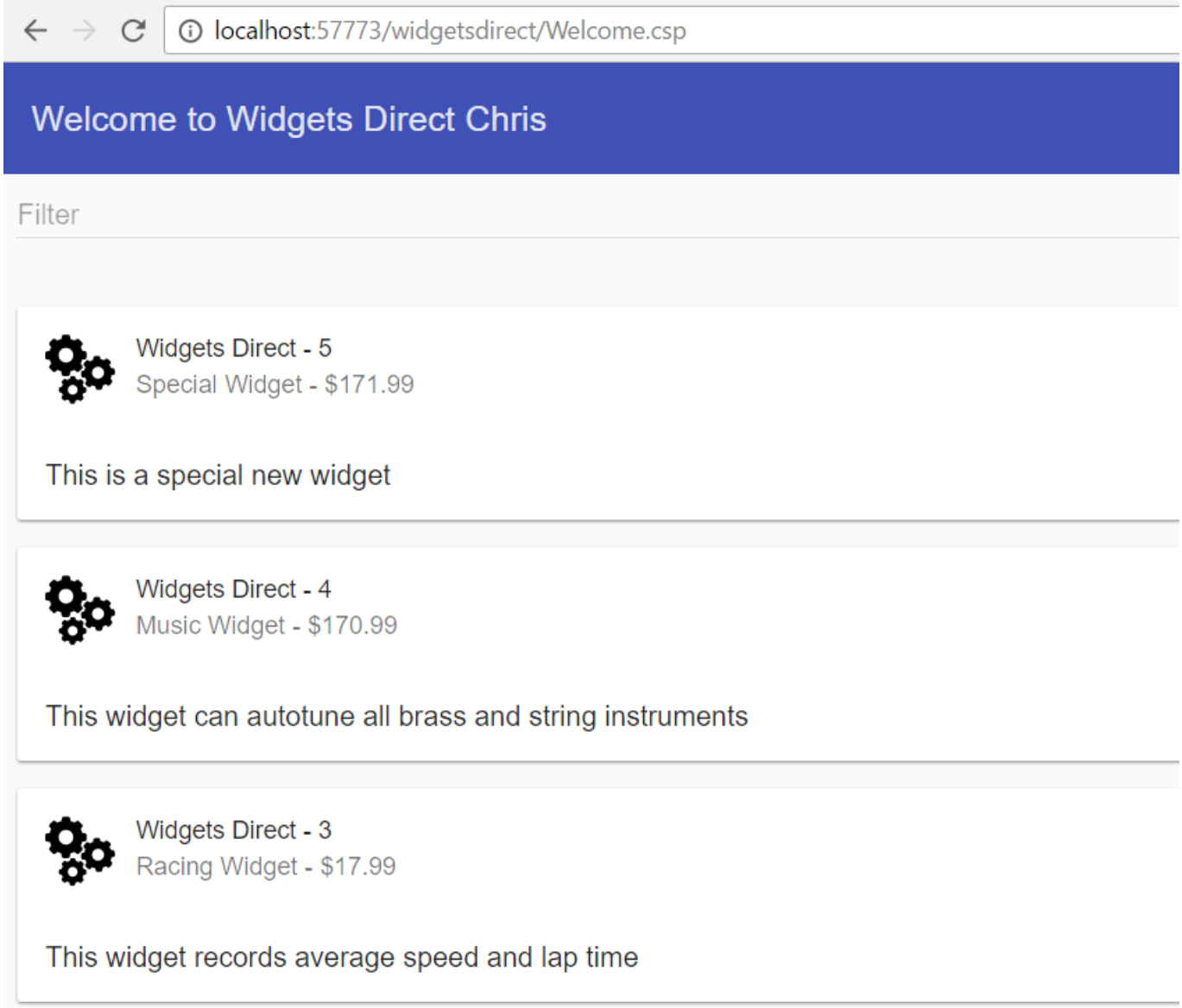
So, it looks like we got a bit excited earlier and referenced our Access ory class before we defined a toJSON method for it. Let's go and define a basic one now (you may notice something missing from this, it's on purpose and will be covered next time)

```
Method toJSON() As %String
{

    set jsonReturn = {}
    set jsonReturn.Id                = ..%Id()
    set jsonReturn.Name              = ..Name
    set jsonReturn.Description       = ..Description
    set jsonReturn.Price             = ..Price
    set jsonReturn.Quantity          = ..Quantity
    set jsonReturn.FirstManufactured = $zd(..FirstManufactured,4)
    set jsonReturn.InProduction      = ..InProduction
    set jsonReturn.Import            = ..Import
    set jsonReturn.SKU               = ..SKU
    quit jsonReturn

}
```

So, we can now compile our Accessory class and reload our page. Hopefully everything should now be returning correctly from the REST service and the page should display



We have a working page again, but we're not displaying anything about Accessories. Let's check our JSON output has some information about these accesssories

```
BODY
{
  Message: "Welcome to Widgets Direct JSON",
  Widgets: [
    {
      Id: "1",
      Name: "Waterproof Widget",
      Description: "This widget is waterproof to 100m depth for a time of up to 7 hours",
      Price: 10.99,
      Quantity: 17,
      Accessories: [
        {Id: "1", Name: "Flotation Aid", Description: "This accessory helps the widget to float", Price: 18.54,…},
        {Id: "2", Name: "Flight Aid", Description: "This accessory helps the widget to fly",…},
        {Id: "3", Name: "Slip Cover", Description: "This accessory protects the widget from scratches",…}
      ]
    },
    {Id: "2", Name: "Woodland Widget", Description: "This widget identifies plant and tree species",…},
    {Id: "3", Name: "Racing Widget", Description: "This widget records average speed and lap time",…},
    {Id: "4", Name: "Music Widget", Description: "This widget can autotune all brass and string instruments",…},
    {Id: "5", Name: "Special Widget", Description: "This is a special new widget",…}
  ]
}
```


We have our linked accessories populating arrays for each widget. As this has been a pretty long lesson so far, let's just do something simple and display a count of compatible accessories for each widget. We will add this to the header of each card, and just return the length of each array to serve as our count, and we can start doing more interesting things with them next time. As in our last lesson, we don't need to do anything special to start using these new data element, we just need to reference them.

```
<md-card-header-text>
  <span class="md-title">Widgets Direct - {{widget.Id}}</span>
  <span class="md-subhead">{{widget.Name}} - ${{widget.Price}}</span>
  <span class="md-subhead">There are {{widget.Accessories.length}} compatible accessories </span>
```

After a quick compile, we now have our Accessories linking to our Widgets

Widgets Direct - 5
Special Widget - $171.99
There are 1 compatible accessories
This is a special new widget

Widgets Direct - 4
Music Widget - $170.99
There are 1 compatible accessories
This widget can autotune all brass and string instruments

Widgets Direct - 3
Racing Widget - $17.99
There are 0 compatible accessories
This widget records average speed and lap time

Widgets Direct - 2
Woodland Widget - $7.99
There are 2 compatible accessories
This widget identifies plant and tree species

Widgets Direct - 1
Waterproof Widget - $10.99
There are 3 compatible accessories
This widget is waterproof to 100m depth for a time of up to 7 hours

Recap

In this lesson we:

1. Broke our application by not referencing our controller JS
2. Fixed our application by using our browser debugger
3. Implemented a new Accessory class, and a relationship class to connect it to our Widget class
4. Broke our application by failing to add a toJSON to our new class
5. Fixed our application by using our browser debugger to identify the issue
6. Added a summary of linked objects using the javascript length function

In our next lesson we will:

- Expand our data model by completing our Accessory toJSON

This article is part of a multi-part series on using Angular on top of Caché REST services. The listing of the full series can be found at the Start Here page

#Angular #CSP #HTML #JavaScript #REST API #Frontend #Caché

Source URL:https://community.intersystems.com/post/lets-write-angular-1x-app-cach%C3%A9-rest-backend-part-7