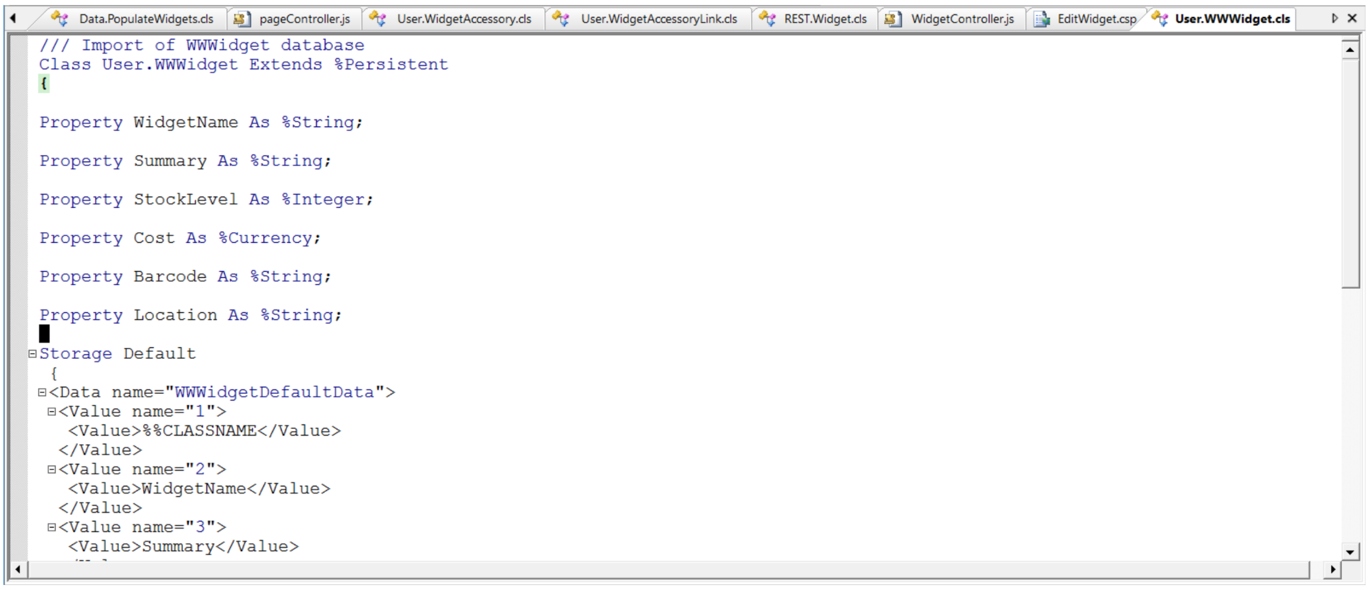
Article
Chris Stewart · May 8, 2017 3m read

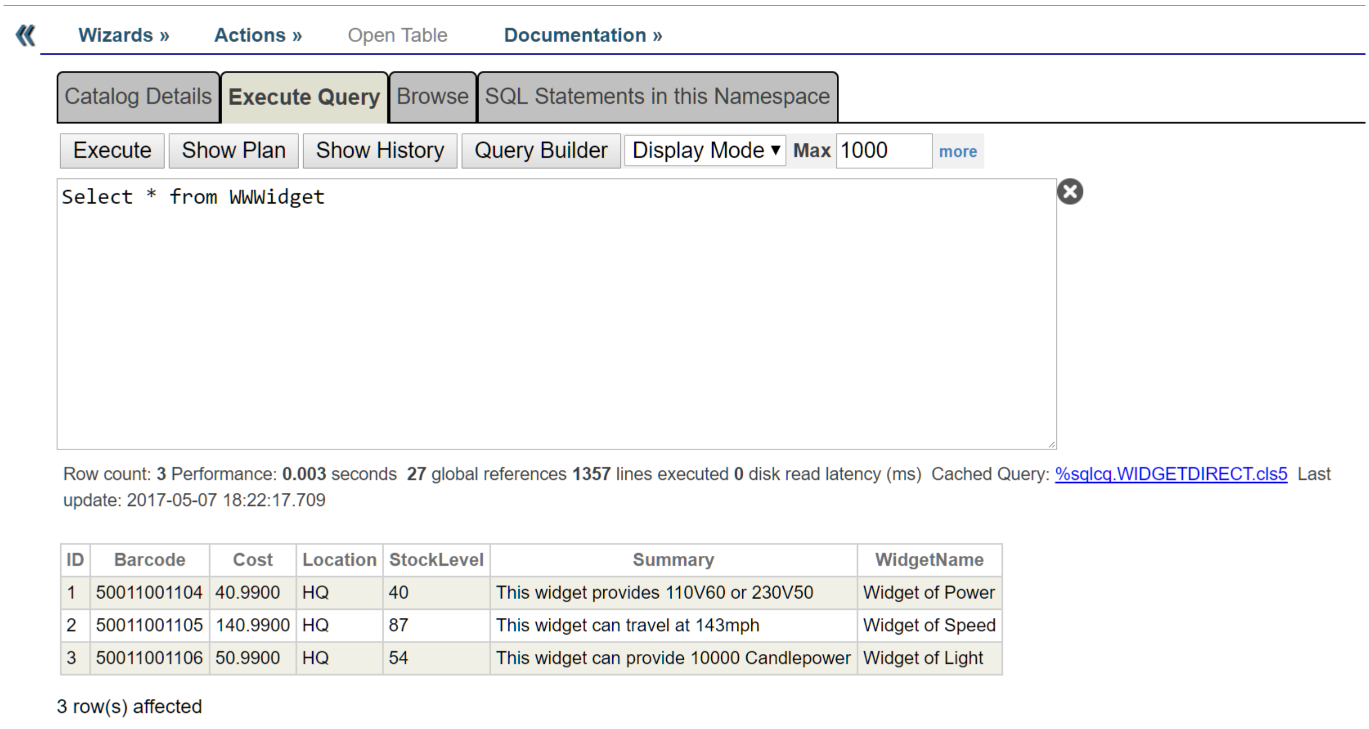
Let's write an Angular 1.x app with a Caché REST backend - Part 12

In our last lesson, we added some formatting and validation to our Edit Widget form. So, now we are ready to add the ability to add new Widgets to our application. However, the great Widget Wars have come to an abrupt end, as Widget Direct has purchased its biggest competitor, WorldWideWidgets. In order to maintain some continuity, we need to display their catalog on our new application.

So, we have good news and bad news. The good news is at that WorldWideWidgets also use Caché, but the bad news is that their Widget table has different properties with different names than our Widget class, and we need to keep the catalogs seperate for the time being. WorldWideWidgets don't have a WidgetAccessory catalog (and they wonder why they lost the Widget War), so we don't need to worry about Accessories for now



```
/// Import of WWWidget database
Class User.WWWidget Extends %Persistent
{

Property WidgetName As %String;

Property Summary As %String;

Property StockLevel As %Integer;

Property Cost As %Currency;

Property Barcode As %String;

Property Location As %String;

Storage Default
 {
 <Data name="WWWidgetDefaultData">
  <Value name="1">
   <Value>%%CLASSNAME</Value>
  </Value>
  <Value name="2">
   <Value>WidgetName</Value>
  </Value>
  <Value name="3">
   <Value>Summary</Value>
```



```
Select * from WWWidget
```

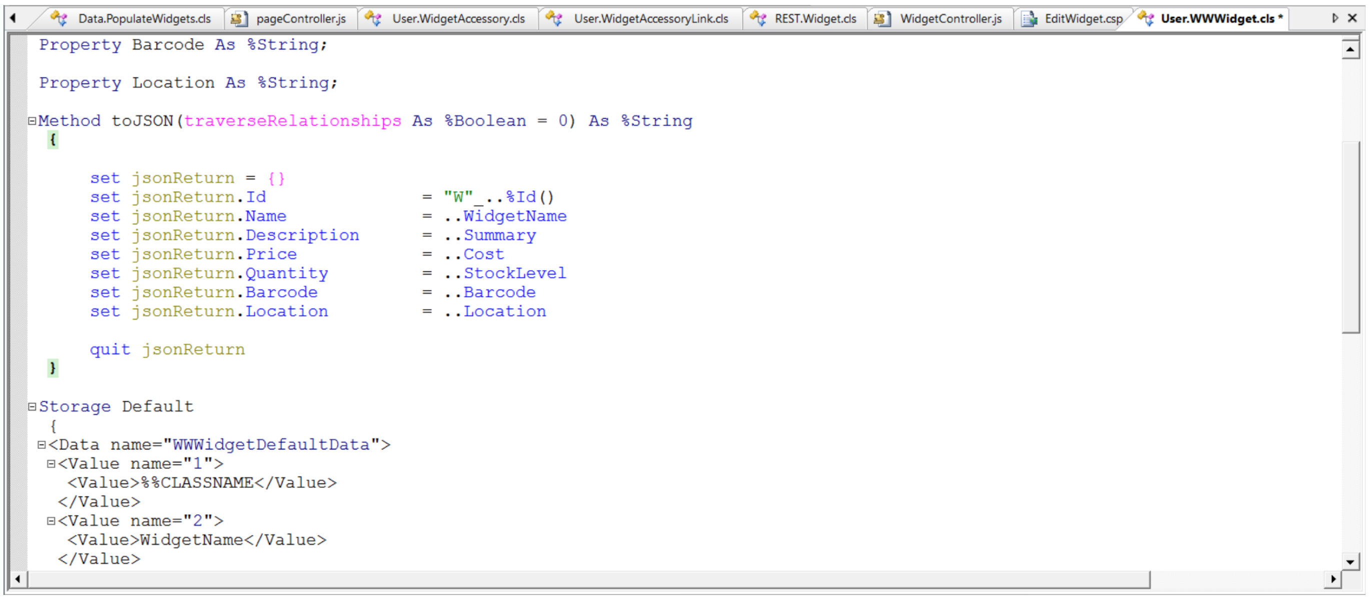
Row count: **3** Performance: **0.003** seconds **27** global references **1357** lines executed **0** disk read latency (ms) Cached Query: %sqlcq.WIDGETDIRECT.cls5 Last update: 2017-05-07 18:22:17.709

ID	Barcode	Cost	Location	StockLevel	Summary	WidgetName
1	50011001104	40.9900	HQ	40	This widget provides 110V60 or 230V50	Widget of Power
2	50011001105	140.9900	HQ	87	This widget can travel at 143mph	Widget of Speed
3	50011001106	50.9900	HQ	54	This widget can provide 10000 Candlepower	Widget of Light

3 row(s) affected

So, this is a serious problem, it looks like we need to implement 2 sets of pages, components and services. Or do we? Since the JSON representation is not tightly linked to the persistent class, we can actually export these Widgets with the property names of the original Widget class, which will then allow us to display and update them using our existing components and controllers.

So first, we implement the toJSON on the WWWidget class. We map the property names to the existing names, implement the additional properties with their actual names, and to differentiate these Widgets from our original catalog, we will prepend the ID value with a 'W'

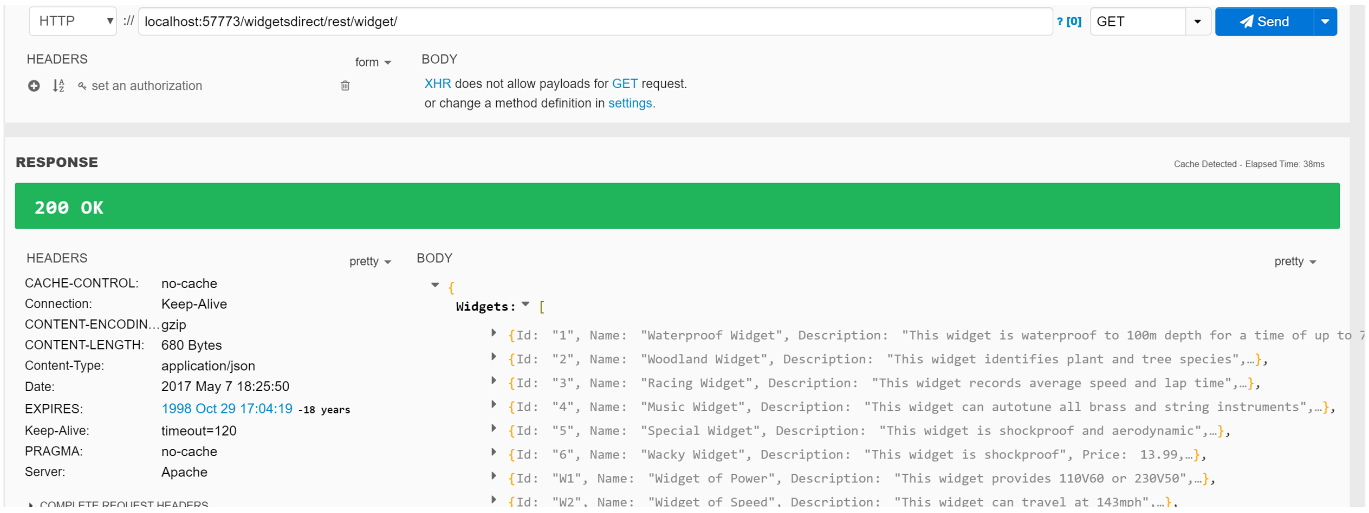


```
Property Barcode As %String;

Property Location As %String;

Method toJSON(traverseRelationships As %Boolean = 0) As %String
{

    set jsonReturn = {}
    set jsonReturn.Id               = "W"_..%Id()
    set jsonReturn.Name             = ..WidgetName
    set jsonReturn.Description      = ..Summary
    set jsonReturn.Price            = ..Cost
    set jsonReturn.Quantity         = ..StockLevel
    set jsonReturn.Barcode          = ..Barcode
    set jsonReturn.Location         = ..Location

    quit jsonReturn
}

Storage Default
{
<Data name="WWWidgetDefaultData">
<Value name="1">
<Value>%%CLASSNAME</Value>
</Value>
<Value name="2">
<Value>WidgetName</Value>
</Value>
```

and then we will alter our REST.Widget class to return all WWWidgets in addition to our Widgets. We will implement a second cursor, and push the toJSON representations of these widgets onto our widget array.

```
For {  &SQL(FETCH WidgetCurs)
       Quit:SQLCODE
       set widgetObj = ##class(User.Widget).%OpenId(Id)
       do widgetAry.%Push(widgetObj.toJSON(1))
    }
&SQL(CLOSE WidgetCurs)

// let's get the WWWidgets
&SQL(DECLARE WWWidgetCurs CURSOR FOR
                                SELECT
                                %Id
                                INTO :Id
                                FROM SQLUser.WWWidget
            )

&SQL(OPEN WWWidgetCurs)

For {  &SQL(FETCH WWWidgetCurs)
       Quit:SQLCODE
       set widgetObj = ##class(User.WWWidget).%OpenId(Id)
       do widgetAry.%Push(widgetObj.toJSON(1))
    }
&SQL(CLOSE WWWidgetCurs)
SET retObj.Widgets = widgetAry

WRITE retObj.%ToJSON()
QUIT $$$OK
```

We can check our REST Service to check that the Widgets from both classes are being returned.


HTTP :// localhost:57773/widgetsdirect/rest/widget/ GET Send

HEADERS form
set an authorization

BODY
XHR does not allow payloads for GET request.
or change a method definition in settings.

RESPONSE Cache Detected - Elapsed Time: 38ms

200 OK

HEADERS pretty

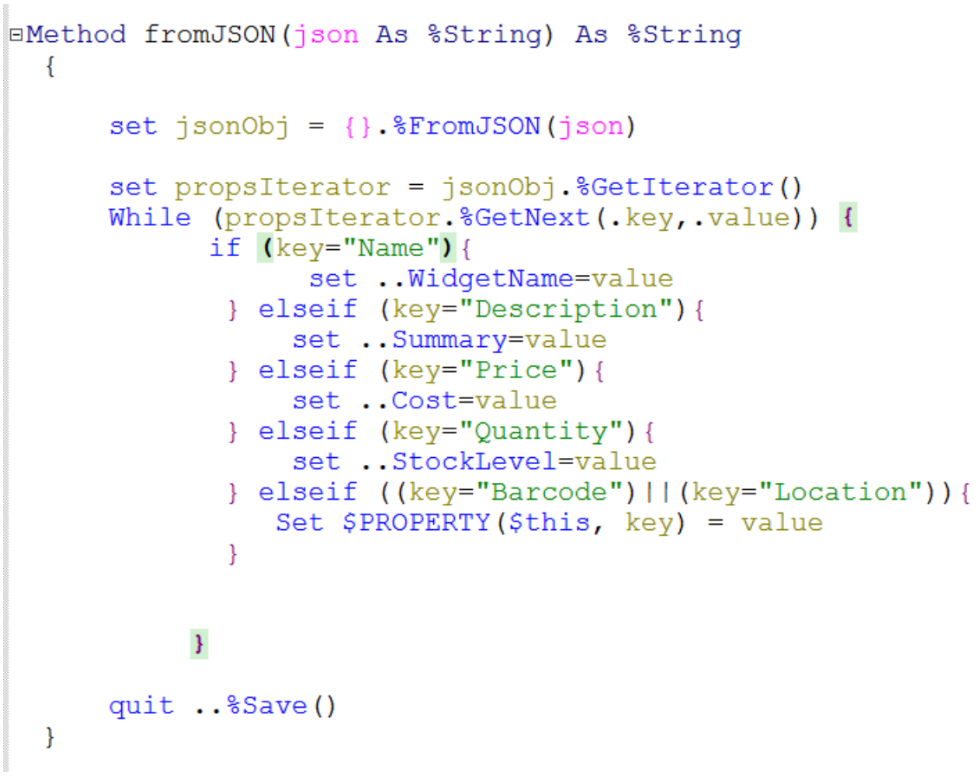
```
CACHE-CONTROL:      no-cache
Connection:         Keep-Alive
CONTENT-ENCODIN...  gzip
CONTENT-LENGTH:     680 Bytes
Content-Type:       application/json
Date:               2017 May 7 18:25:50
EXPIRES:            1998 Oct 29 17:04:19 -18 years
Keep-Alive:         timeout=120
PRAGMA:             no-cache
Server:             Apache
```

COMPLETE REQUEST HEADERS

BODY pretty

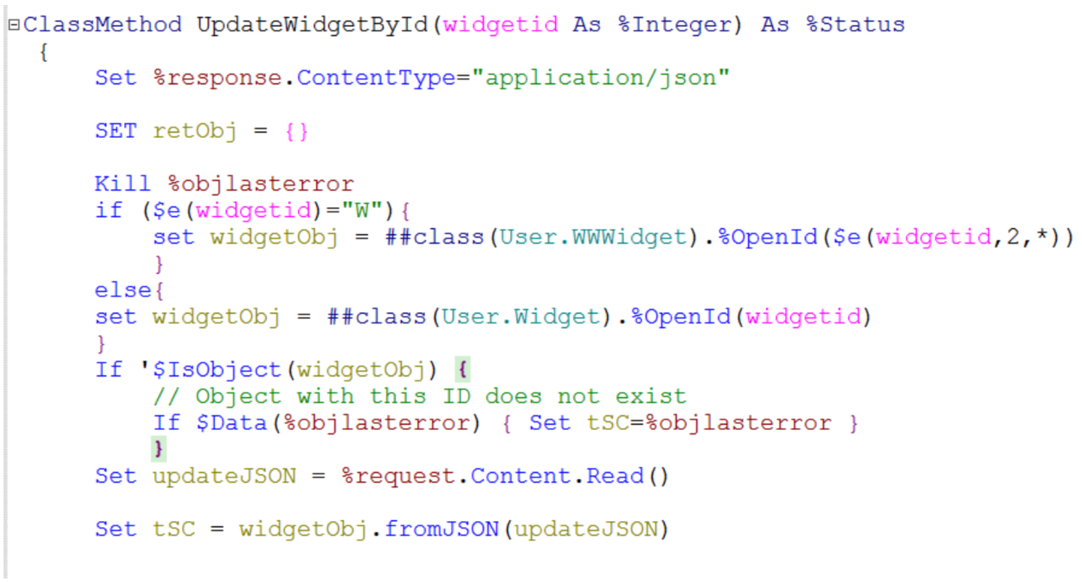
```
{
  Widgets: [
    {Id: "1", Name: "Waterproof Widget", Description: "This widget is waterproof to 100m depth for a time of up to 7
    {Id: "2", Name: "Woodland Widget", Description: "This widget identifies plant and tree species",…},
    {Id: "3", Name: "Racing Widget", Description: "This widget 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 widget is shockproof and aerodynamic",…},
    {Id: "6", Name: "Wacky Widget", Description: "This widget is shockproof", Price: 13.99,…},
    {Id: "W1", Name: "Widget of Power", Description: "This widget provides 110V60 or 230V50",…},
    {Id: "W2", Name: "Widget of Speed", Description: "This widget can travel at 143mph",…},
```

Now that we have our GET working, we can work on allowing Update of our new class. This is a reverse of the toJSON, where we will iterate over all properties we received, and manually map them to the actual properties of the class of WWWidgets.



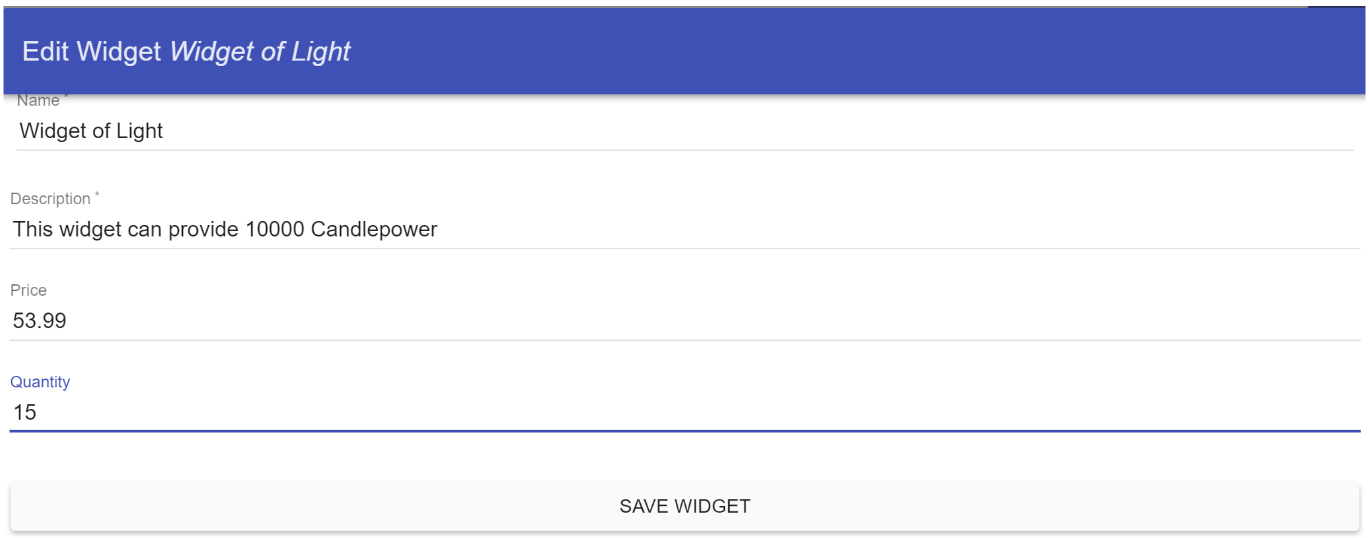
```
Method fromJSON(json As %String) As %String
 {

     set jsonObj = {}.%FromJSON(json)

     set propsIterator = jsonObj.%GetIterator()
     While (propsIterator.%GetNext(.key,.value)) {
           if (key="Name"){
                 set ..WidgetName=value
             } elseif (key="Description"){
                set ..Summary=value
             } elseif (key="Price"){
                set ..Cost=value
             } elseif (key="Quantity"){
                set ..StockLevel=value
             } elseif ((key="Barcode")||(key="Location")){
               Set $PROPERTY($this, key) = value
             }


         }

     quit ..%Save()
 }
```

We now need to make sure our PUT service is able to choose between our 2 Widget classes. Luckily, we prepended the ID of the WWWidget class with a W, so we have an easy way to differentiate between each class. If the first character of the ID is a 'W' then we %OpenId with the rest of the ID value.



```
ClassMethod UpdateWidgetById(widgetid As %Integer) As %Status
 {
     Set %response.ContentType="application/json"

     SET retObj = {}

     Kill %objlasterror
     if ($e(widgetid)="W"){
         set widgetObj = ##class(User.WWWidget).%OpenId($e(widgetid,2,*))
         }
     else{
     set widgetObj = ##class(User.Widget).%OpenId(widgetid)
     }
     If '$IsObject(widgetObj) {
         // Object with this ID does not exist
         If $Data(%objlasterror) { Set tSC=%objlasterror }
         }
     Set updateJSON = %request.Content.Read()

     Set tSC = widgetObj.fromJSON(updateJSON)
```

So, to prove that our updates are now working, we can use our Welcome.csp page. Let's open up W3, and update the price, and check that everything updates correctly


Edit Widget *Widget of Light*

Name *
Widget of Light

Description *
This widget can provide 10000 Candlepower

Price
53.99

Quantity
15

Success! We can now update Widgets in both tables seamlessly. We can implement a similar condition on our GetWidgetById method to load instances of either class, based on the W prefix of the ID.

We should also implement a method to add new WWWidgets next? Well, since we are winding down that catalog, then we will not implement any methods to add new entries. We now have full WWWidget support as part of our application

Today we:

1. Imported another Widget class
2. Implement toJSON and fromJSON methods
3. Implement logic to return and set WWWidget entries

Next time we will:

- implement an "Add New" form for Widgets

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 #HTML #JavaScript #JSON #REST API #Frontend #Caché

Source URL:https://community.intersystems.com/post/lets-write-angular-1x-app-cach%C3%A9-rest-backend-part-12