Article
Chris Stewart · Apr 28, 2017 4m read

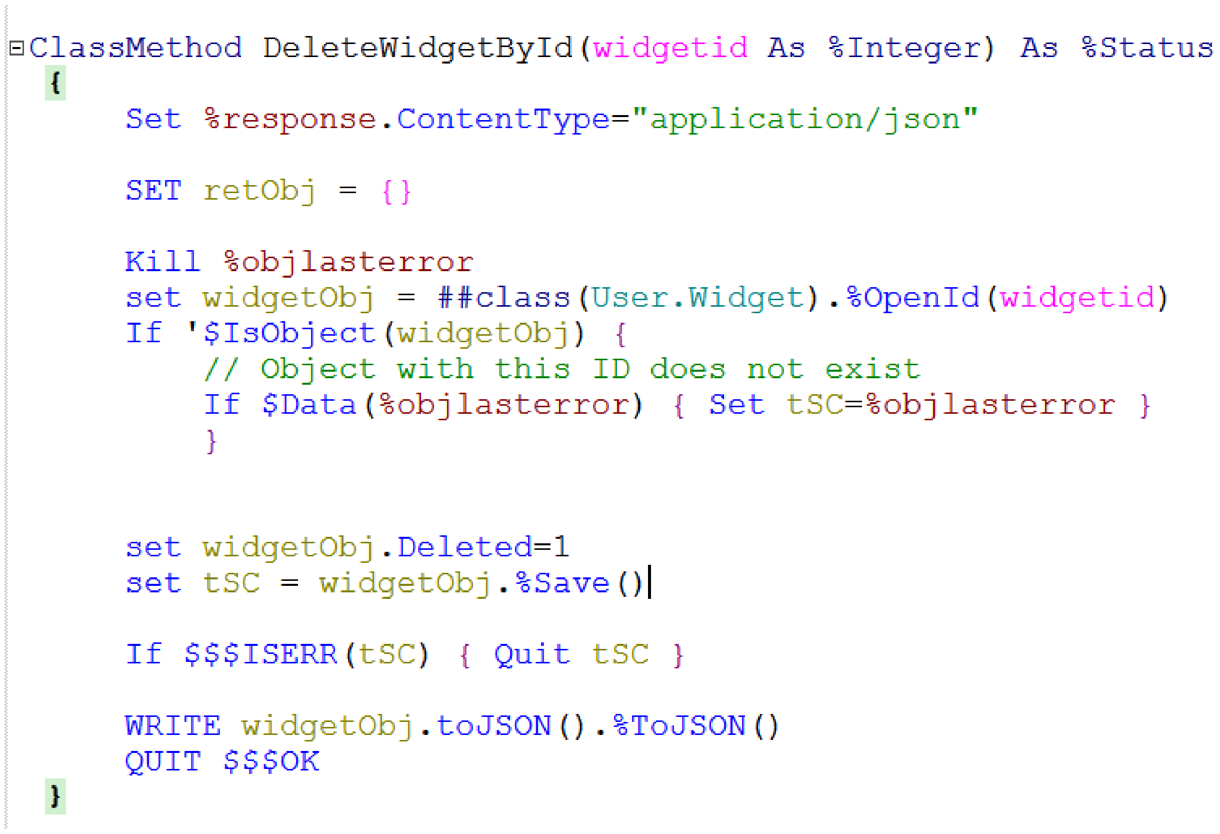
Let's write an Angular 1.x app with a Caché REST backend - Part 10

In our last lesson, we implemented a new REST Service to allow us to perform CRU operations on Widgets, and refactored our Controllers to allow the page setup to be decouple from the content.

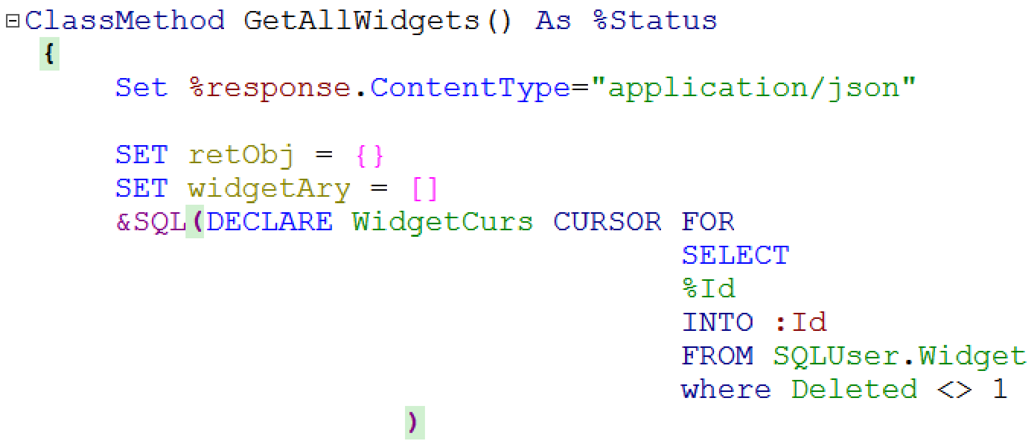
When we created our Widget Services, we did not implement a Deletion operation, which the HTTP Delete verb provides. As this is a base table for other parts of the Widgets Direct empire, we don't want to be able to do a hard Delete of the WIdget values, as this could cause issues with our ordering and billing modules. So, we will add a "Deleted" property to the class, and have the Delete operation set this Boolean flag instead. We will adjust our GetAllWidgets method to ignore any records which are marked as "Deleted"

```
/// Persistent class to hold Widgets
Class User.Widget Extends %Persistent
{

Property Name As %String;

Property Description As %String(MAXLEN = 500);

Property Price As %Currency;

Property Quantity As %Integer;

Property Deleted As %Boolean [ InitialExpression = 0 ];
```

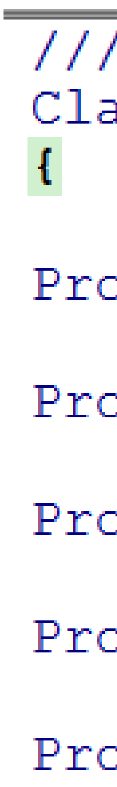
REST.Widget



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

     SET retObj = {}

     Kill %objlasterror
     set widgetObj = ##class(User.Widget).%OpenId(widgetid)
     If '$IsObject(widgetObj) {
         // Object with this ID does not exist
         If $Data(%objlasterror) { Set tSC=%objlasterror }
         }


     set widgetObj.Deleted=1
     set tSC = widgetObj.%Save()

     If $$$ISERR(tSC) { Quit tSC }

     WRITE widgetObj.toJSON().%ToJSON()
     QUIT $$$OK
 }
```



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

     SET retObj = {}
     SET widgetAry = []
     &SQL(DECLARE WidgetCurs CURSOR FOR
                                  SELECT
                                  %Id
                                  INTO :Id
                                  FROM SQLUser.Widget
                                  where Deleted <> 1
                  )
```


```
///
Cla
{

Pro

Pro

Pro

Pro

Pro
```

We now have a full CRUD stack in REST for our Widgets (though we do not currently handle any relationships between widgets and their accessories, that will come later)

So, now we have our Update and Add methods for Widgets, we need a way to access them from our web application. Since this is Widget Specific code, we will keep things neat by creating a new WidgetController.js and place it in the /modules/widget/ web folder. We will implement a function that pops up a Material Dialog window, using a custom html template. This is set to see the $scope from the calling application, and implements a Controller specific to the Dialog window, with it's own scope. Inside this controller, we implement a function to take a widget object, and PUT it to a specific Widget's Update call (there is a lot going on here, but I've tried to keep this example as simple as I can). If the update call is successful, then the Dialog window closes, if it is not, then the window stays open and an error is written to the Debugger console using the $log service

modules/widget/WidgetController.js

```
angular
  .module('WidgetsDirect')
   .controller('WidgetController', ['$q', '$scope', '$timeout', '$http', '$log', '$mdDialog',function($q, $scope, $timeout, $ht

      $scope.editWidget = function(ev, widget) {
         var parent = angular.element(document.body);
         $scope.widget = widget;

         $mdDialog.show({
            parent: parent,
            clickOutsideToClose: true,
            scope: $scope,
            preserveScope: true,
            targetEvent: ev,
            templateUrl: '/widgetsdirect/modules/widget/EditWidget.csp',
            controller: function DialogController($scope, $mdDialog, $http, $log) {

                $scope.saveWidget = function(widget) {
                  $http.post('/widgetsdirect/rest/widget/'+widget.Id, widget).then(
                     function(data) {

                        $mdDialog.hide();

                     },
                     function(data) {

                          $log.info('Update failed ');

                     }
                  );
```

```
Cla
 {

 }
```

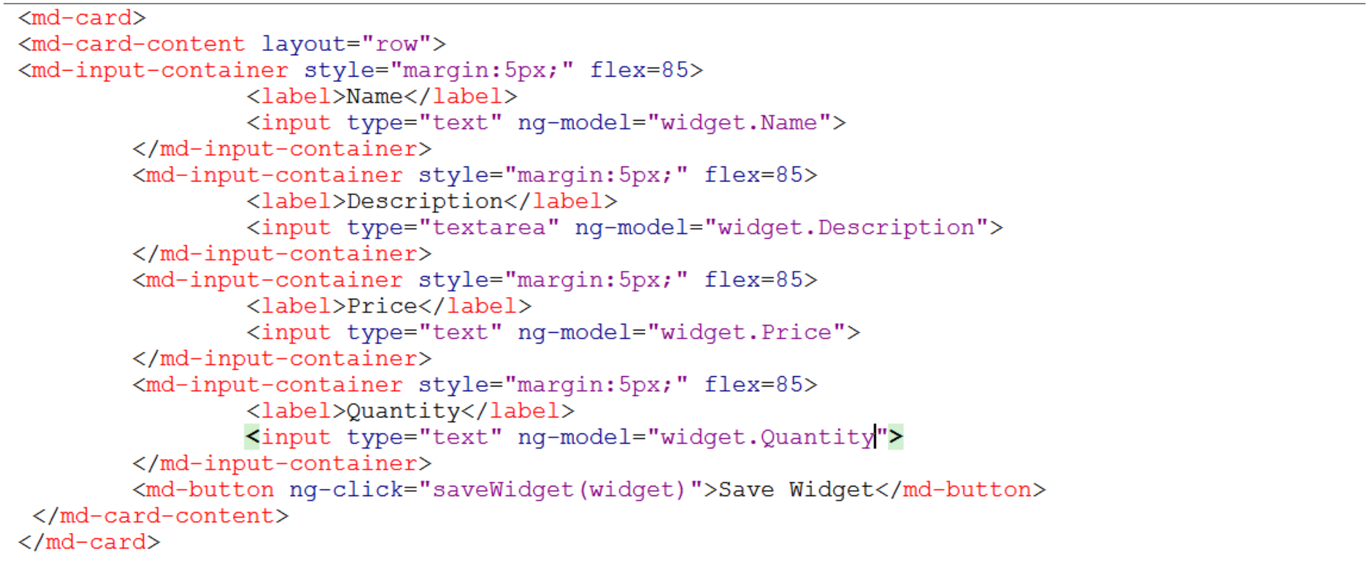
We now need to update our Widget card to be bound to this new Widget Controller, and implement an ng-click somewhere to trigger the function which displays our Dialog Window, For now, we will bind the ng-click to the Description field, so that clicking anywhere in the Description will give us the dialog to edit that record. We pass in the $event variable which allows the Angular runtime to determine where actions were triggered from.

Welcome.csp

```
  <!-- List of Widgets starts here -->
  <md-card ng-repeat="widget in widgets | filter : widgetFilterText |  orderBy: !sortAsc ? 'Id' : '-Id'" ng-controller="WidgetController">
   <md-card-header>
    <md-card-avatar>
     <img src="img/logo.svg"/>
    </md-card-avatar>
    <md-card-header-text>
      <span class="md-title">Widgets Direct - {{widget.Id}}</span>
      <span class="md-subhead">{{widget.Name}} - ${{widget.Price}}</span>
      <span class="md-subhead">There are {{widget.Accessories.length}} compatible accessories </span>
    </md-card-header-text>
   </md-card-header>
   <md-card-content ng-click="editWidget($event,widget)">
    <p>
      {{widget.Description}}
    </p>
   </md-card-content>
 </md-card>
```

Finally, we need to create the Template file we specified in our mdDialog setup. We create EditWidget in the modules/widget/ folder, and can make a (very) basic form from some Material Input components. We also implement a Button which will trigger the saveWidget method that is located in the Dialog window's scope. We will come back and tidy up this form later. Notice that we do not have to create a full HTML page for the Form template, we can just provide the UI Components that are needed for the form

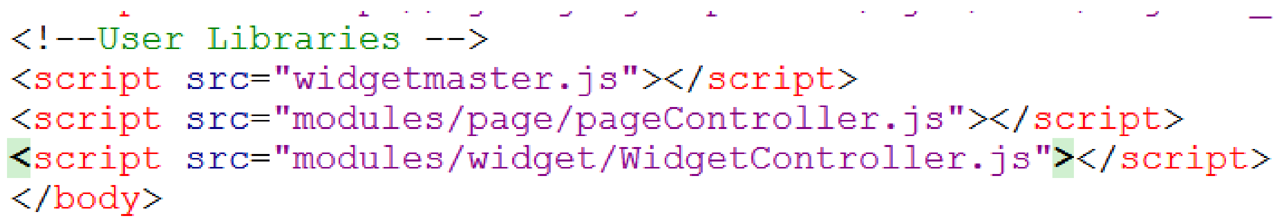
modules/widget/EditWidget.csp



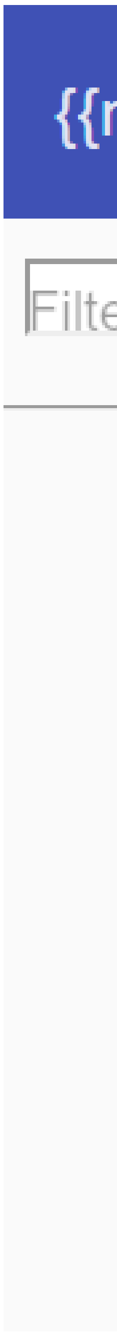
```
<md-card>
<md-card-content layout="row">
<md-input-container style="margin:5px;" flex=85>
                <label>Name</label>
                <input type="text" ng-model="widget.Name">
        </md-input-container>
        <md-input-container style="margin:5px;" flex=85>
                <label>Description</label>
                <input type="textarea" ng-model="widget.Description">
        </md-input-container>
        <md-input-container style="margin:5px;" flex=85>
                <label>Price</label>
                <input type="text" ng-model="widget.Price">
        </md-input-container>
        <md-input-container style="margin:5px;" flex=85>
                <label>Quantity</label>
                <input type="text" ng-model="widget.Quantity">
        </md-input-container>
        <md-button ng-click="saveWidget(widget)">Save Widget</md-button>
 </md-card-content>
</md-card>
```

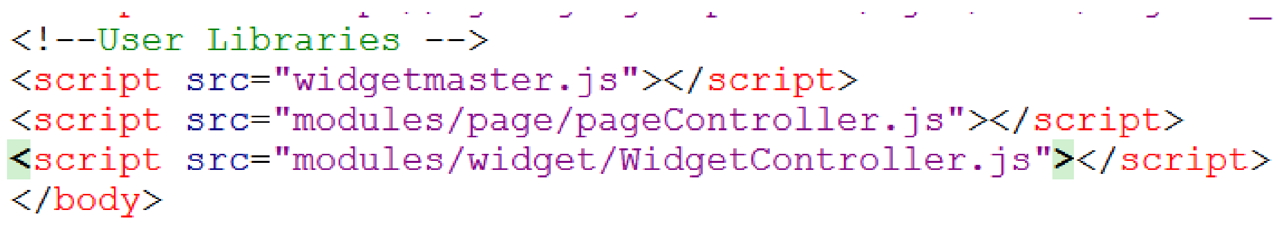
Now, we can add the new controller to the list of User scripts on the Welcome page, compile everything and refresh our page, so we can start editing!

Welcome.csp



```
<!--User Libraries -->
<script src="widgetmaster.js"></script>
<script src="modules/page/pageController.js"></script>
<script src="modules/widget/WidgetController.js"></script>
</body>
```

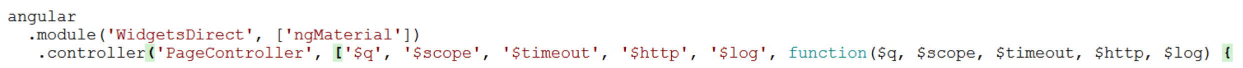




```
<!--User Libraries -->
<script src="widgetmaster.js"></script>
<script src="modules/page/pageController.js"></script>
<script src="modules/widget/WidgetController.js"></script>
</body>
```

This looks bad. It appears the whole Angular runtime is unhappy. This is actually a trap I laid a few lessons back, when refactoring the Controllers. The syntax for defining an Angular Module is

modules/page/pageController.js



```
angular
  .module('WidgetsDirect', ['ngMaterial'])
   .controller('PageController', ['$q', '$scope', '$timeout', '$http', '$log', function($q, $scope, $timeout, $http, $log) {
```

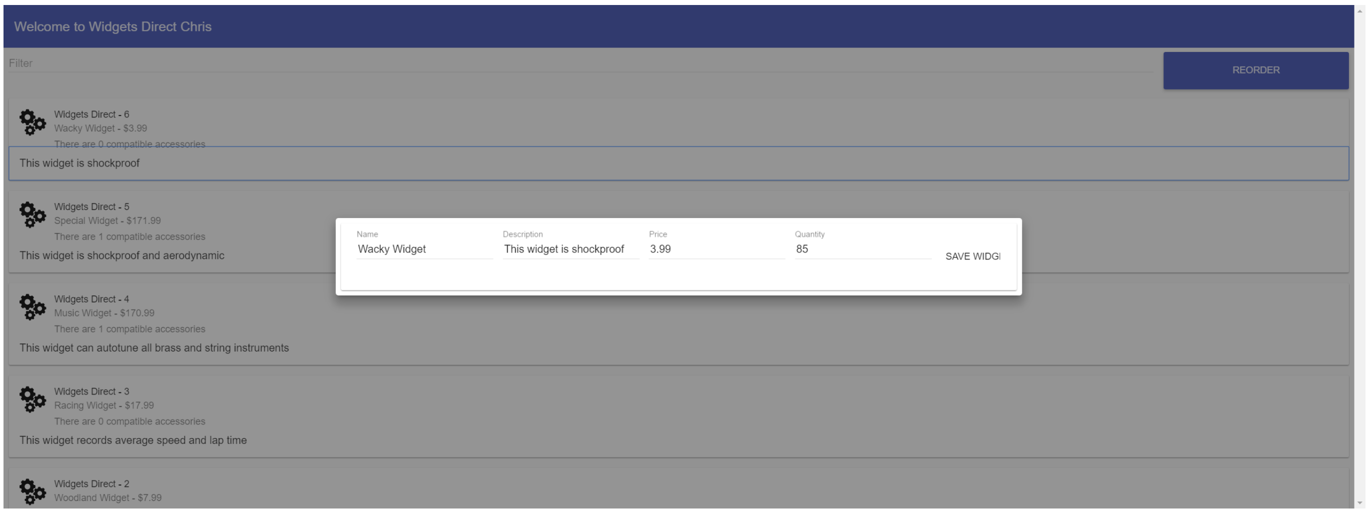
and the syntax for attaching config to an existing module is:

modules/widget/WidgetController.js

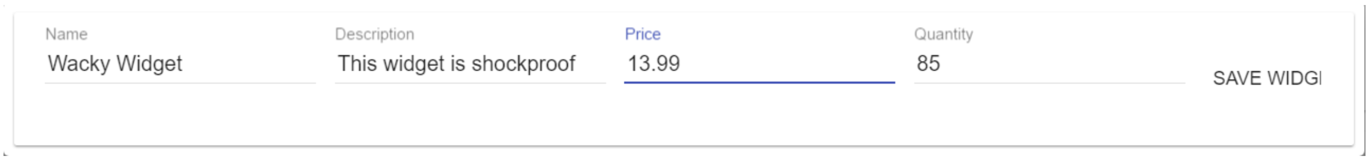
```
angular
  .module('WidgetsDirect')
   .controller('WidgetController', ['$q', '$scope', '$timeout', '$http', '$log', '$mdDialog',function($q, $scope, $timeout, $http, $log, $mdDialog) {
```

There can only be one module definition per application, but we have the injection dependency list *['ngMaterial']* in all of our controllers. We need to remove the second argument from our .module lines in both the Widget and Page Controller, and leave the Module definition in the widgetmaster.js. This is a nasty bug to have in your code, and will pretty much prevent all functionality from running. Once you've encountered this once, you will never fall for it again, so this is a public service message rather so you can learn from my mistakes :)

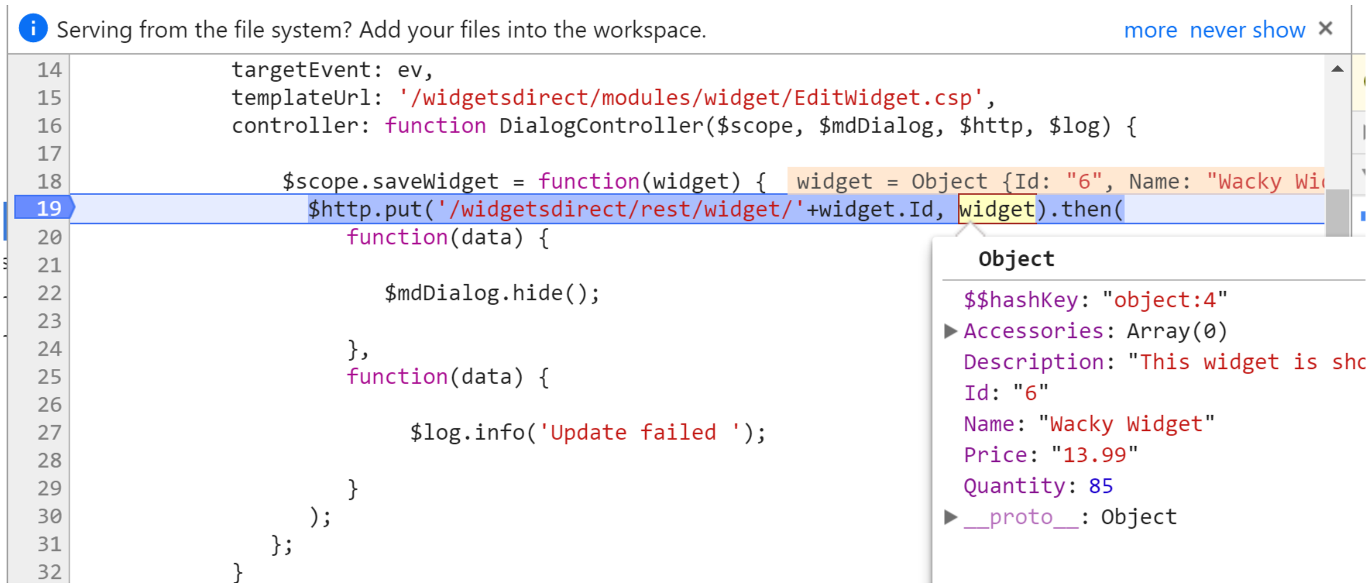
With our Module definition corrected, we can now load our page, and should be able to click a Widget Description to get an Edit form


Welcome to Widgets Direct Chris
Filter
REORDER
Widgets Direct - 6
Wacky Widget - $3.99
There are 0 compatible accessories
This widget is shockproof
Widgets Direct - 5
Special Widget - $171.99
There are 1 compatible accessories
This widget is shockproof and aerodynamic
Name
Wacky Widget
Description
This widget is shockproof
Price
3.99
Quantity
85
SAVE WIDGI
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

Excellent, let's load our debugger with F12, and place a breakpoint on our $http.put so we can see our request as it is sent. We will then change the price of the Widget, and hit Save to see our request being sent to the server



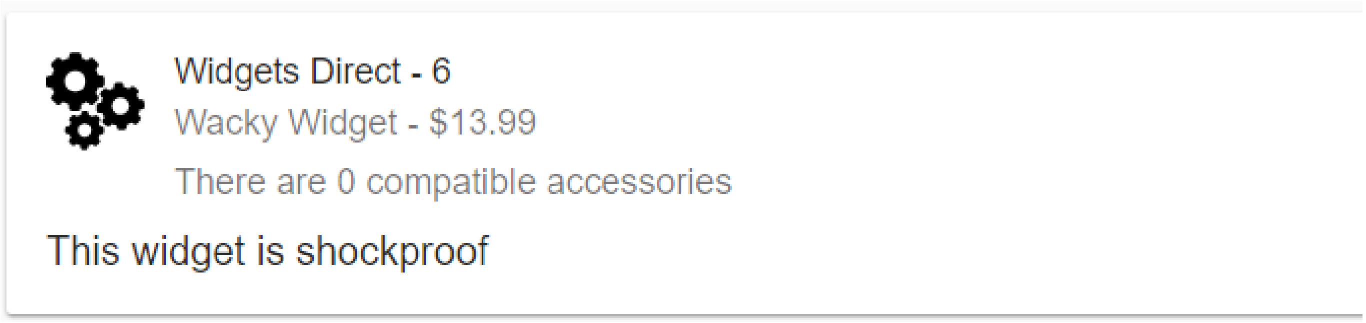
WidgetController.js


Serving from the file system? Add your files into the workspace.

```
            targetEvent: ev,
            templateUrl: '/widgetsdirect/modules/widget/EditWidget.csp',
            controller: function DialogController($scope, $mdDialog, $http, $log) {

                $scope.saveWidget = function(widget) {  widget = Object {Id: "6", Name: "Wacky Wi
                    $http.put('/widgetsdirect/rest/widget/'+widget.Id, widget).then(
                        function(data) {

                            $mdDialog.hide();

                        },
                        function(data) {

                            $log.info('Update failed ');

                        }
                    );
                };
            }
```

```
Object
$$hashKey: "object:4"
▶ Accessories: Array(0)
Description: "This widget is shc
Id: "6"
Name: "Wacky Widget"
Price: "13.99"
Quantity: 85
▶ __proto__: Object
```

We can see that our update is in the payload object, which is great, but we are also sending all fields, including the Accessories array which seems a bit wasteful (especially when we start adding more properties to our Services). If we hit Play, then the debugger should resume, and we should see our Dialog window close, and our update persisted to the Widget


Widgets Direct - 6
Wacky Widget - $13.99
There are 0 compatible accessories
This widget is shockproof

We now have a method to update our Widgets, though our Form is very very basic, and our update payload isn't optimised. We will cover this in future lessons

Today we:

1. Implemented the DELETE verb for Widgets
2. Created a templated Dialog window
3. Created a new Widget Controller
4. Corrected our Module definition
5. Performed our first REST PUT call

Next time we will:

- Tidy up our form, by adding styling and validation

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