
Artículo

[Joel Espinoza](#) · 26 ago, 2019 Lectura de 4 min

Desarrollar un backend de servicios REST para una aplicación Angular 1.x con Caché - Parte 2

"Así que su jefe le gritó por enviarle una página web con un "Hello World" sin formato"

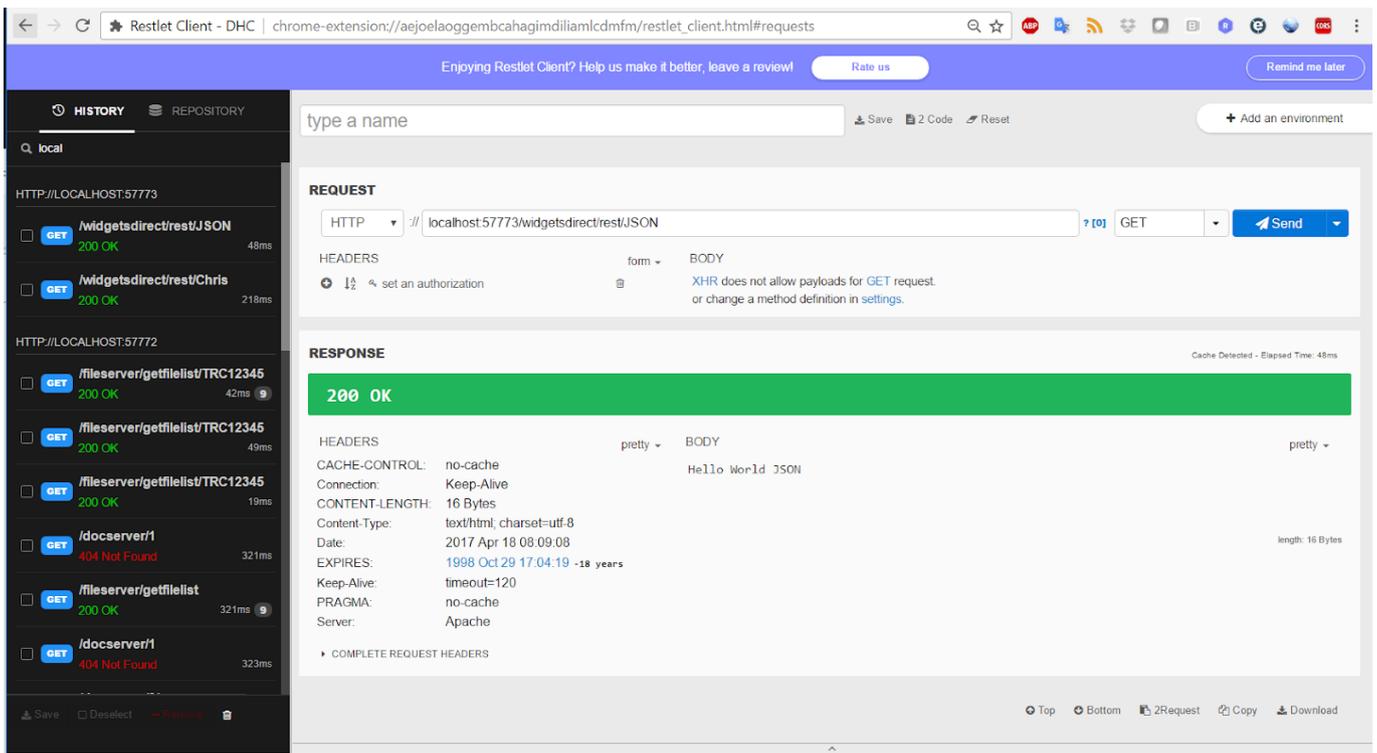
Nuestra [lección anterior](#) terminó cuando proporcionamos un valor para el Mensaje desde Caché a un servicio REST para el cliente, y utilizamos Angular interfaz de usuario. Aunque hay muchas partes móviles involucradas en el proceso, la página no es especialmente emocionante en este momento. Antes de que empecemos a añadir nuevas funciones, deberíamos retroceder un poco y revisar nuestras herramientas.

En este tutorial se utiliza la funcionalidad de JSON que está integrada en las versiones 2016.2+ de Caché. Esta funcionalidad está parcialmente disponible en la versión 2016.1 pero utiliza una sintaxis diferente, la cual puede ya no ser compatible en el futuro.

En la publicación 1, verificamos la salida de nuestra solicitud GET mediante un navegador estándar, y visualizamos la salida de texto simple en la página. Esto solamente sirve para probar los escenarios más básicos, entonces deberíamos instalar algo que nos permitiera crear peticiones HTTP personalizadas y luego asimilar las respuestas. Para ello, existen numerosas opciones disponibles, las cuales van desde la herramienta curl en la línea de comandos hasta extensiones para probar los servicios. La extensión [RESTLET](#) (antes llamado DHC) está disponible como un plugin de Chrome, y proporciona un excelente soporte para las solicitudes HTTP con JSON. Este es el cliente que usaré en todas las publicaciones que haga en el futuro, pero cualquier depurador HTTP funcionará de la misma manera.

Estas publicaciones no serán un tutorial sobre Angular 1.x o Angular en general. Para ello, existen muchísimos y excelentes tutoriales gratuitos que están disponibles en línea.

Entonces, regresemos a nuestro código. Comencemos a probar nuestro nuevo depurador HTTP. Cargue la url del servicio y envíe la solicitud. Debería ver algo como esto:



En este punto, probablemente se esté preguntando "pensé que íbamos a usar JSON para devolver nuestros datos desde REST? ¿Estos datos son solamente texto simple?" Eso es correcto, y necesitamos trabajar más en nuestro servicio REST para que nuestra salida sea un poco más útil. Volvamos al Studio y abramos nuestra clase REST.Dispatch. Necesitamos crear un objeto en JSON (usamos el atajo {} para hacer esto), y luego estableceremos una propiedad para este objeto de modo que contenga nuestro mensaje (Message). Después, haremos impresión de la data del mensaje (usando Write), pero usando la función %ToJSON(), esto generará el formato correcto a partir del objeto. Entonces, al igual que lo hicimos antes, haremos una salida con el estado OK

```
ClassMethod HelloWorld(Name As %String) As %Status
{
    SET retObj = {}
    SET retMessage = "Hello World " _ Name
    SET retObj.Message = retMessage
    WRITE retObj.%ToJSON()
    QUIT $$$OK
}
```

Solicitémoslo nuevamente desde nuestro servicio:

The screenshot shows a REST client interface. Under the 'REQUEST' tab, the URL is 'localhost:57773/widgetsdirect/rest/JSON' and the method is 'GET'. A 'Send' button is visible. Under the 'RESPONSE' tab, the status is '200 OK'. The response headers are listed on the left, and the body is shown as a JSON object: {"Message": "Hello World JSON"}. The interface also shows a 'Cache Detected - Elapsed Time: 61ms' message.

Ahora tenemos una cadena JSON, pero no implementa una bonita impresión. Es casi como si el cliente no supiera que esto supuestamente es JSON. Si nos fijamos en el tipo de contenido (Content-Type), la respuesta del mensaje está en text/html. Aunque algunos clientes podrán descifrar el contenido en JSON automáticamente, debemos tener claro que aprobamos JSON como respuesta. Volvamos a nuestra clase, y agreguemos el tipo de contenido (Content-Type) a nuestra respuesta (%response). También cambiaremos nuestro mensaje para darle una bienvenida más apropiada a Widgets Direct

```
ClassMethod HelloWorld(Name As %String) As %Status
{
    Set %response.ContentType="application/json" ✓

    SET retObj = {}|
    SET retMessage = "Welcome to Widgets Direct " _Name
    SET retObj.Message = retMessage
    WRITE retObj.%ToJSON()
    QUIT $$$OK
}
```

Si ahora cargamos nuevamente nuestra solicitud, veremos correctamente y bien impreso el JSON.

The screenshot shows a REST client interface. At the top, under the 'REQUEST' tab, the URL is 'localhost:57773/widgetsdirect/rest/JSON' and the method is 'GET'. A 'Send' button is visible. Below the request, the 'RESPONSE' tab is active, showing a '200 OK' status. The response headers include 'no-cache', 'Keep-Alive', '30 Bytes', 'application/json', and a date. The response body is a JSON object: `{ "Message": "Hello World JSON" }`. A green bar at the top of the response section indicates the status.

Ahora presentamos JSON a nuestro cliente. Volvamos a cargar nuestra página de Bienvenida para ver si esto hizo alguna diferencia

The screenshot shows a browser window with the address bar displaying 'localhost:57773/widgetsdirect/Welcome.csp'. The developer console shows the following JSON response: `{"Message": "Welcome to Widgets Direct Chris"}`.

Hay una diferencia, pero no es exactamente lo que esperábamos. Ahora mostramos todo el objeto JSON, debido a que nuestro controlador puede vincular toda la sección de datos con el mensaje (`$scope.message`). Necesitamos modificar nuestro controlador para desempaquear correctamente el campo del mensaje (`Message`).

```
$http.get('/widgetsdirect/rest/Chris').then(  
  function(response) { //success  
    $scope.message = response.data.Message;  
  }  
);
```

Ahora que nuestro mensaje se desempaqueó y estableció correctamente, podemos actualizar nuestra página para obtener nuestro nuevo mensaje de Bienvenida

The screenshot shows a browser window with the address bar displaying 'localhost:57773/widgetsdirect/Welcome.csp'. The page content now displays 'Welcome to Widgets Direct Chris'.

Recapitulemos

En esta lección nosotros:

1. Aprendimos que no debemos enviarle páginas con un "Hello World" a nuestro jefe
2. Revisamos nuestras herramientas
3. Visualizamos nuestra salida REST en un depurador HTML
4. Convertimos nuestro servicio REST para que mostrara JSON correctamente
5. Actualizamos el controlador de nuestra página para que desempaquetara la respuesta JSON

En nuestra [próxima lección](#) haremos lo siguiente:

- Añadiremos un conjunto de JSON a nuestro servicio
- Añadiremos en nuestra página una visualización repetitiva para un conjunto de elementos

[#Angular](#) [#API REST](#) [#CSP](#) [#Frontend](#) [#HTML](#) [#JavaScript](#) [#JSON](#) [#Tutorial](#) [#Caché](#)

URL de
fuente: <https://es.community.intersystems.com/post/desarrollar-un-backend-de-servicios-rest-para-una-aplicaci%C3%B3n-angular-1x-con-cach%C3%A9-parte-2>