


Open · Opened 2 days ago by  **Rubén Montero**

Un tercer endpoint... JSON

Resumen

- Escribiremos una clase `JSONGetExample` que devuelva un mensaje distinto e intentaremos que sea en formato JSON
- Hablaremos de la cabecera `Content-Type: application/json`
- Usaremos `StringRepresentation` para que nuestro servidor envíe dicha cabecera
- Serviremos el resultado en `/example3`

Descripción

Nuestros dos *endpoint* anteriores entregan contenido en *texto plano*.

Esto está bien, pero podemos hacerlo mejor.

Un buen API REST transferirá los datos usando un formato estándar, como [XML](#) o [JSON](#).

Un tercer *endpoint*

Sigue los pasos que ya conoces para añadir un tercer *endpoint*.

- Crearemos una nueva clase `JSONGetExample.java` que extienda de `ServerResource`
- En `SimpleREST.java`, la *mapearemos* al *endpoint* `/example3` con `.attach`

Retomando `JSONObject`

Dentro de `JSONGetExample.java`, en el método `toString` vamos a **fabricar** un `JSONObject` sencillo como ya sabemos. El objetivo es que sea algo así:

```
{
  "message": "Esto es un JSON de prueba"
}
```

Así que haremos...

```
JSONObject json = new JSONObject();
json.put("message", "Esto es un JSON de prueba");
```

Basta con invocar `.toString()` para que dicho `JSONObject` sea convertido en un *texto* transferible:

El ejemplo completo en `JSONGetExample.java`:

```
@Get
public String toString() {
    JSONObject json = new JSONObject();
    json.put("message", "Esto es un JSON de prueba");
    String response = json.toString();
    return response;
}
```

¿Funciona?

Si ahora navegamos a `http://localhost:8104/example3`, **veremos** la respuesta JSON esperada.

Pero esto **no funciona** suficientemente **bien**.

El *tipo de contenido* de una petición/respuesta HTTP se especifica con una cabecera `Content-Type`. Ahora mismo, Restlet está enviando al navegador la cabecera:

```
Content-Type: text/plain
```

Esto es **insuficiente** porque muchos clientes HTTP **fallarán** a la hora de interpretar la respuesta correctamente, como el JSON que es.

¿Y qué hacemos?

Debemos encargar a Restlet que **responda** usando esta cabecera:

```
Content-Type: application/json
```

¿Cómo lo conseguimos?

Ya **no** nos vale el sencillo método `toString` que veníamos usando.

Vamos a añadir un **nuevo** método que devuelva `StringRepresentation`. Esta es una clase propia de Restlet.

El nombre del método, ahora **da igual** siempre y cuando esté anotado con `@Get`:

```
import org.json.JSONObject;
import org.restlet.data.MediaType;
import org.restlet.representation.StringRepresentation;
import org.restlet.resource.Get;
import org.restlet.resource.ServerResource;

public class JSONGetExample extends ServerResource {

    // Método toString viejo
    @Get
    public String toString() { /* ... */ }

    // Nuevo método
    @Get
    public StringRepresentation getEndpointResponse() {
        // ...
    }
}
```

StringRepresentation

En dicho método vamos a añadir el código que *genera* un JSON, como antes:

```
// Nuevo método
@Get
public StringRepresentation getEndpointResponse() {
    JSONObject json = new JSONObject();
    json.put("message", "Esto es un JSON de prueba");
    String jsonString = json.toString();
    // ...
}
```

Pero, esta vez, vamos a **instanciar** un objeto `StringRepresentation` pasando nuestro `String` como parámetro al construirlo.

Así:

```
// Nuevo método
@Get
public StringRepresentation getEndpointResponse() {
    JSONObject json = new JSONObject();
    json.put("message", "Esto es un JSON de prueba");
    String jsonString = json.toString();
    StringRepresentation representation = new StringRepresentation(jsonString);
    // ...
}
```

Y, ¿por qué?

MediaType.APPLICATION_JSON

Para poder *especificar* que dicha *representación* es de tipo JSON. Lo hacemos invocando `.setMediaType` y pasando la *constante* `MediaType.APPLICATION_JSON`, así:

```
// Nuevo método
@Get
public StringRepresentation getEndpointResponse() {
    JSONObject json = new JSONObject();
    json.put("message", "Esto es un JSON de prueba");
    String jsonString = json.toString();
    StringRepresentation representation = new StringRepresentation(jsonString);
    representation.setMediaType(MediaType.APPLICATION_JSON);
    // ...
}
```

¡Listo!

Ya sólo falta *devolver* la instancia que hemos construido.

Restlet entenderá perfectamente que se trata de un JSON.

```
public class JSONGetExample extends ServerResource {
-
-   @Get
-   public String toString() {
-       JSONObject json = new JSONObject();
-       json.put("message", "Esto es un JSON de prueba");
-       String jsonString = json.toString();
-       return jsonString;
-   }
-
-   @Get
-   public StringRepresentation getEndpointResponse() {
-       JSONObject json = new JSONObject();
-       json.put("message", "Esto es un JSON de prueba");
-       String jsonString = json.toString();
-       StringRepresentation representation = new StringRepresentation(jsonString);
-       representation.setMediaType(MediaType.APPLICATION_JSON);
+       return representation;
+   }
}
```

Podemos borrar el `toString` anterior. Se trata de un método sencillo para devolver respuestas en *texto plano*, pero que **no** usaremos de ahora en adelante.

Además, **no** puede haber **más de un método** anotado con `@Get` dentro del mismo `ServerResource`. ¿Por qué crees que es así?

Ha mejorado el asunto

Si ahora visitamos <http://localhost:8104/example3> desde el navegador, veremos como *se muestra de forma diferente*.

El *contenido* de la respuesta es el mismo, pero el navegador (cada uno lo hace a su manera) decide *presentarlo* de una forma más entendible.

Por último

Verifica que el *test* funciona correctamente.

Haz `commit` y `push` para subir los cambios al repositorio.



[Rubén Montero @ruben.montero](#) changed milestone to [%Sprint 2](#) 2 days ago