



A+ API REST

Grupo: 32

Miembro:

José Ángel Domínguez Espinaco

Daniel Lozano Portillo

José Joaquín Rodríguez Pérez

María Ruíz Gutiérrez

Miguel Ternero Algarín

Laura Vera Recacha

Índice

APIRest con un solo objeto de tipo "Material"	3
APIRest con la Collection de tipo "Material"	6
Bibliografía	8

APIRest con un solo objeto de tipo “Material”

Comenzamos realizando la API Rest que aplicaremos a un objeto de tipo “Material”, el cual tiene los siguientes atributos:

```
@Entity
@Access({AccessType.PROPERTY})
@Table(indexes = {
    @Index(columnList = "title,description,unitPrice")
})
public class Material extends DomainEntity {

    // Attributes -----

    private String title;
    private String description;
    private Double unitPrice;
    private Double quantity;
    private Double totalPrice;
```

Para realizar la API consultaremos la documentación que nos proporciona Spring en su página web: <https://spring.io/guides/gs/rest-service/>

Vamos a crear un método get que nos devolverá un Json con los materiales de nuestro sistema, pero primero intentaremos que nos devuelva solo un objeto material. Para ello, creamos un nuevo controlador y le decimos a Spring que es un controlador tipo Rest mediante la anotación `@RestController`, esto es similar a poner en Spring la anotación `@Controller` y la anotación `@ResponseBody` en los métodos de la clase.

```
@RestController
@RequestMapping(value = "/materialJSON")
public class MaterialJSONController extends AbstractController {
```

Spring serializa y deserializa Json automáticamente con la librería Jackson, para ello vamos a añadirla a maven definiéndola en el “pom.xml”.

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.5.1</version>
</dependency>
```

Por último, escribiríamos nuestro método get que será el responsable de enviar un JSON de los materiales al cuerpo de la respuesta.

```
// Listing -----
@RequestMapping(value = "/listJSON", produces = {
    MediaType.APPLICATION_JSON_VALUE
}, method = RequestMethod.GET)
public Material list() {
    Material material;

    material = this.materialService.findAll().iterator().next();
    //Quitamos la bidireccionalidad para que no nos salga un stack overflow al intentar mapearlo a json
    material.setBuys(null);
    material.getLabelMaterial().setMaterials(null);

    return material;
}
```

Como se puede comprobar no convertimos nuestro objeto java a Json en ningún momento puesto que Spring gracias a la anotación `@ResponseBody` (se pone implícitamente al poner la anotación `@RestController` como se ha dicho anteriormente) serializa el objeto java automáticamente con la librería Jackson, es decir, la convierte de objeto Java a Json. También tenemos que indicar que la respuesta es de tipo Json mediante la anotación siguiente: `produces = {MediaType.APPLICATION_JSON_VALUE}`.

Por último, indicar que, como se comenta en el método, tenemos que quitar las relaciones bidireccionales de nuestro objeto para evitar un “stack overflow error” al mapearlo a Json.

Al arrancar el servidor Tomcat, nos encontramos con el siguiente error:

Panic

Sorry for the inconvenience. The following exception was thrown and is not recoverable: `HttpMediaTypeNotAcceptableException`.

Message

Could not find acceptable representation

Stack trace

```
org.springframework.web.HttpMediaTypeNotAcceptableException: Could not find acceptable representation at
org.springframework.web.servlet.mvc.method.annotation.AbstractMessageConverterMethodProcessor.writeWithMessageConverters(AbstractMessageConverterMethodProcessor.java:193) at
org.springframework.web.servlet.mvc.method.annotation.AbstractMessageConverterMethodProcessor.writeWithMessageConverters(AbstractMessageConverterMethodProcessor.java:193) at
org.springframework.web.servlet.mvc.method.annotation.RequestResponseBodyMethodProcessor.handleReturnValue(RequestResponseBodyMethodProcessor.java:193) at
org.springframework.web.servlet.mvc.method.annotation.HandlerMethodReturnValueHandlerComposite.handleReturnValue(HandlerMethodReturnValueHandlerComposite.java:71) at
org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:122) at
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandleMethod(RequestMappingHandlerAdapter.java:748) at
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:689) at
org.springframework.web.servlet.mvc.method.annotation.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:83) at
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:945) at
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:876) at
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:931) at
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:822) at
org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:807) at
org.springframework.web.servlet.http.HttpServlet.service(HttpServlet.java:728) at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:305) at
```

Esto es debido porque nuestra configuración de Spring no mapea automáticamente los objetos a Json con la librería Jackson, por ello utilizaremos como alternativa la biblioteca de Google para Json denominada Gson y una breve modificación del método para convertir los objetos manualmente dentro de los métodos en vez de dejar ese trabajo a Spring.

En primer lugar, añadimos a maven la biblioteca Gson y en segundo lugar reescribimos el método anterior.

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.8.0</version>
</dependency>

@RequestMapping(value = "/listJSON", produces = {
    MediaType.APPLICATION_JSON_VALUE
}, method = RequestMethod.GET)
public String listJson() {

    final Material material = this.materialService.findAll().iterator().next();

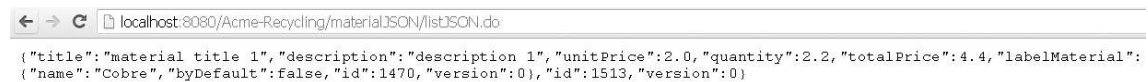
    //Quitamos la bidireccionalidad para que no nos salga un stack overflow al intentar mapearlo a json
    material.setBuys(null);
    material.getLabelMaterial().setMaterials(null);

    //Serializamos el objeto a Json
    final String materialSerialized = new Gson().toJson(material);

    return materialSerialized;
}
```

Este sería nuestro nuevo método, dicho método nos devuelve un String que es el Json de nuestro objeto material. Este String se añade directamente al cuerpo del mensaje de respuesta

gracias a la anotación `@ResponseBody` que va implícita en la anotación `@RestController` como se ha indicado anteriormente.



```
{ "title": "material title 1", "description": "description 1", "unitPrice": 2.0, "quantity": 2.2, "totalPrice": 4.4, "labelMaterial": "Cobre", "byDefault": false, "id": 1470, "version": 0 }, { "id": 1513, "version": 0 }
```

Vemos que se ha serializado correctamente nuestro objeto a Json. A continuación, escribiremos un método que consuma el Json de nuestro método get y nos muestre el objeto material en la vista de “listar los materiales”. Dicho método es el siguiente:

```
@RequestMapping(value = "/list", method = RequestMethod.GET)
public ModelAndView list() {

    ModelAndView result;

    final Collection<Material> materials;

    materials = new ArrayList<Material>();

    final RestTemplate restTemplate = new RestTemplate();
    //Si intento convertirlo a la clase Material.class no funcionaria, por lo tanto lo convertimos a una clase String y despues hacemos la conversion con Gson
    final String jsonString = restTemplate.getForObject("http://localhost:8080/Acme-Recycling/materialJSON/listJSON.do", String.class);
    //Hacemos la conversion con Gson
    final Gson gson = new Gson();
    final Material material = gson.fromJson(jsonString, Material.class);

    //Lo añadimos a la lista
    materials.add(material);

    result = new ModelAndView("material/list");
    result.addObject("materials", materials);
    result.addObject("requestURI", "materialJSON/list.do");

    return result;
}
```

En primer lugar, inicializamos la lista de materiales vacía para posteriormente añadirle el material deserializado y llamar a la vista “list” que tenemos para los materiales. Seguidamente utilizamos un objeto de tipo “RestTemplate” para consumir el Json que nos devuelve nuestro método get y lo pasamos a la clase “String” para luego por medio de un objeto Gson convertirlo en objeto java. Lo pasamos a “String” y no al tipo “Material” directamente porque de la segunda manera nos salta el siguiente error ya que Spring no nos deserializa automáticamente el Json:

Panic

Sorry for the inconvenience. The following exception was thrown and is not recoverable: `RestClientException`.

Message

Could not extract response: no suitable `HttpMessageConverter` found for response type [class domain.Material] and content type [application/json]

Stack trace

```
org.springframework.web.client.RestClientException: Could not extract response: no suitable HttpMessageConverter found for response type [class domain.Material] and content type [application/json] at
org.springframework.web.client.HttpMessageConverterExtractor.extractData(HttpMessageConverterExtractor.java:108) at
org.springframework.web.client.RestTemplate.doExecute(RestTemplate.java:535) at org.springframework.web.client.RestTemplate.execute(RestTemplate.java:489)
at org.springframework.web.client.RestTemplate.getForObject(RestTemplate.java:226) at
org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerMethod.invokeForRequest(AnnotationMethodHandlerMethod.java:132) at
org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerMethod.invokeAndHandle(AnnotationMethodHandlerMethod.java:104) at
org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter.invokeHandleMethod(AnnotationMethodHandlerAdapter.java:748) at
org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter.handleInternal(AnnotationMethodHandlerAdapter.java:689) at
```

Por lo tanto, gracias a nuestro pequeño ajuste, comprobamos que ya nos muestra el material en nuestro list correctamente:

One item found.

Title	Description	Unit price	Quantity	Total price	Label of material
material title 1	description 1	€2	2.2	€4.4	Cobre

If you wanna buy some material... [click here to register as a buyer](#)

APIRest con la Collection de tipo “Material”

Para finalizar, serializaremos y deserializaremos la colección de todos los materiales que están en nuestro sistema para asemejarlo a la vista que tienen los usuarios no autenticados para ver los materiales de nuestro sistema, pero consumiendo nuestra API desde el controlador. El método para serializar el objeto a Json es el siguiente:

```
@RequestMapping(value = "/listJSON", produces = {
    MediaType.APPLICATION_JSON_VALUE
}, method = RequestMethod.GET)
public String listJSON() {

    final Collection<Material> materials = this.materialService.findAll();
    //Quitamos la bidireccionalidad para que no nos salga un stack overflow al intentar mapearlo a json
    for (final Material m : materials) {
        m.setBuys(null);
        m.getLabelMaterial().setMaterials(null);
    }
    final String serialize = new Gson().toJson(materials);
    return serialize;
}
```

Esta vez tenemos que recorrer con un bucle for todos los materiales para quitar la bidireccionalidad para que así no nos salga el siguiente error:

HTTP Status 500 - Handler processing failed; nested exception is java.lang.StackOverflowError

type Exception report
message Handler processing failed; nested exception is java.lang.StackOverflowError
description The server encountered an internal error that prevented it from fulfilling this request.
exception

```
org.springframework.web.util.NestedServletException: Handler processing failed; nested exception is java.lang.StackOverflowError
org.springframework.web.servlet.DispatcherServlet.triggerAfterCompletionWithError(DispatcherServlet.java:1284)
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:965)
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:876)
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:931)
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:822)
javax.servlet.http.HttpServlet.service(HttpServlet.java:621)
org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:807)
javax.servlet.http.HttpServlet.service(HttpServlet.java:728)
org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:330)
org.springframework.security.web.access.intercept.FilterSecurityInterceptor.invoke(FilterSecurityInterceptor.java:118)
org.springframework.security.web.access.intercept.FilterSecurityInterceptor.doFilter(FilterSecurityInterceptor.java:84)
org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:342)
org.springframework.security.web.access.ExceptionTranslationFilter.doFilter(ExceptionTranslationFilter.java:113)
org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:342)
org.springframework.security.web.session.SessionManagementFilter.doFilter(SessionManagementFilter.java:103)
org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:342)
org.springframework.security.web.authentication.AnonymousAuthenticationFilter.doFilter(AnonymousAuthenticationFilter.java:113)
org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:342)
org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter.doFilter(SecurityContextHolderAwareRequestFilter.java:154)
org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:342)
```

Por lo tanto, el nuevo Json que enviamos como respuesta es el siguiente:

```
localhost:8080/Acme-Recycling/materialJSON/listJSON.do

[{"title": "material title 1", "description": "description 1", "unitPrice": 2.0, "quantity": 2.2, "totalPrice": 4.4, "labelMaterial": {"name": "Cobre", "byDefault": false, "id": 1470, "version": 0}, {"id": 1513, "version": 0}, {"title": "material title 2", "description": "description 2", "unitPrice": 2.0, "quantity": 2.2, "totalPrice": 4.4, "labelMaterial": {"name": "Cobre", "byDefault": false, "id": 1470, "version": 0}, {"id": 1514, "version": 0}, {"title": "material title 3", "description": "description 3", "unitPrice": 3.1, "quantity": 3.2, "totalPrice": 9.92, "labelMaterial": {"name": "Aluminio", "byDefault": false, "id": 1471, "version": 0}, {"id": 1515, "version": 0}, {"title": "material title 4", "description": "description 4", "unitPrice": 4.2, "quantity": 4.2, "totalPrice": 17.64, "labelMaterial": {"name": "Aluminio", "byDefault": false, "id": 1471, "version": 0}, {"id": 1516, "version": 0}, {"title": "material title 5", "description": "description 5", "unitPrice": 5.0, "quantity": 5.2, "totalPrice": 26.0, "labelMaterial": {"name": "Bronce", "byDefault": false, "id": 1472, "version": 0}, {"id": 1517, "version": 0}, {"title": "material title 6", "description": "description 6", "unitPrice": 12.0, "quantity": 3.2, "totalPrice": 38.4, "labelMaterial": {"name": "Bronce", "byDefault": false, "id": 1472, "version": 0}, {"id": 1518, "version": 0}]
```

Seguidamente, consumiremos dicho JSON con nuestro método list(), para ello el único problema que hemos tenido es a la hora de intentar deserializar una Collection de un objeto, el cual lo hemos solucionado por medio de un objeto del tipo “Type” como se muestra en la imagen de nuestro método:

```
@RequestMapping(value = "/list", method = RequestMethod.GET)
public ModelAndView list() {

    ModelAndView result;

    final RestTemplate restTemplate = new RestTemplate();
    //Si intento convertirlo a la clase Material.class no funcionaria, por lo tanto lo convertimos a una clase String y despues hacemos la conversion con Gson
    final String string = restTemplate.getForObject("http://localhost:8080/Acme-Recycling/materialJSON/listJSON.do", String.class);
    //Hacemos la conversion con Gson
    final Gson gson = new Gson();
    final Type listType = new TypeToken<ArrayList<Material>>() {
    }.getType();
    final List<Material> materials = gson.fromJson(string, listType);

    result = new ModelAndView("material/list");
    result.addObject("materials", materials);
    result.addObject("requestURL", "materialJSON/list.do");

    return result;
}
```

Una vez escrito el método, pasamos a arrancar Tomcat y comprobar que funciona correctamente ingresando en la url que se muestra en la imagen:

Materials

6 items found, displaying 1 to 5.
[First/Prev] 1 2 [Next/Last]

Title	Description	Unit price	Quantity	Total price	Label of material
material title 1	description 1	€2	2.2	€4.4	Cobre
material title 2	description 2	€2	2.2	€4.4	Cobre
material title 3	description 3	€3.1	3.2	€9.92	Aluminio
material title 4	description 4	€4.2	4.2	€17.64	Aluminio
material title 5	description 5	€5	5.2	€26	Bronce

If you wanna buy some material... [click here to register as a buyer](#)

Con esta imagen, comprobamos que aparecen los materiales que estamos consumiendo del JSON que nos devuelve la url: <http://localhost:8080/Acme-Recycling/materialJSON/listJSON.do>

Bibliografía

- <https://spring.io/guides/gs/rest-service/>
- <https://spring.io/guides/gs/consuming-rest/>
- <https://stackoverflow.com/questions/29416804/creating-spring-rest-services-without-using-spring-boot>
- <http://www.baeldung.com/jackson-vs-gson>
- <https://stackoverflow.com/questions/5554217/google-gson-deserialize-listclass-object-generic-type>
- <https://stackoverflow.com/questions/38262055/how-to-post-a-json-payload-to-a-requestparam-in-spring-mvc>