ResponseEntity es una de las clases más habituales cuando uno utiliza Spring Framework con Servicios REST. Cuando uno comienza a trabajar con REST los ejemplos suelen ser muy directos y se hace uso de las anotaciones esenciales a la hora de publicar el Servicio . Sin embargo si queremos añadir flexibilidad y construir un servicio REST de mayor calidad será obligatorio conocer como funciona esta clase de Spring Framework . Vamos a ver un ejemplo de hola mundo que nos ayude a aclarar las cosas partiendo de un proyecto de Spring Boot estas son sus dependencias

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.5.6</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.arquitecturajava</groupId>
        <artifactId>rest</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>rest</name>
        <description>spring rest</description>
        <properties>
                <java.version>11</java.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
```

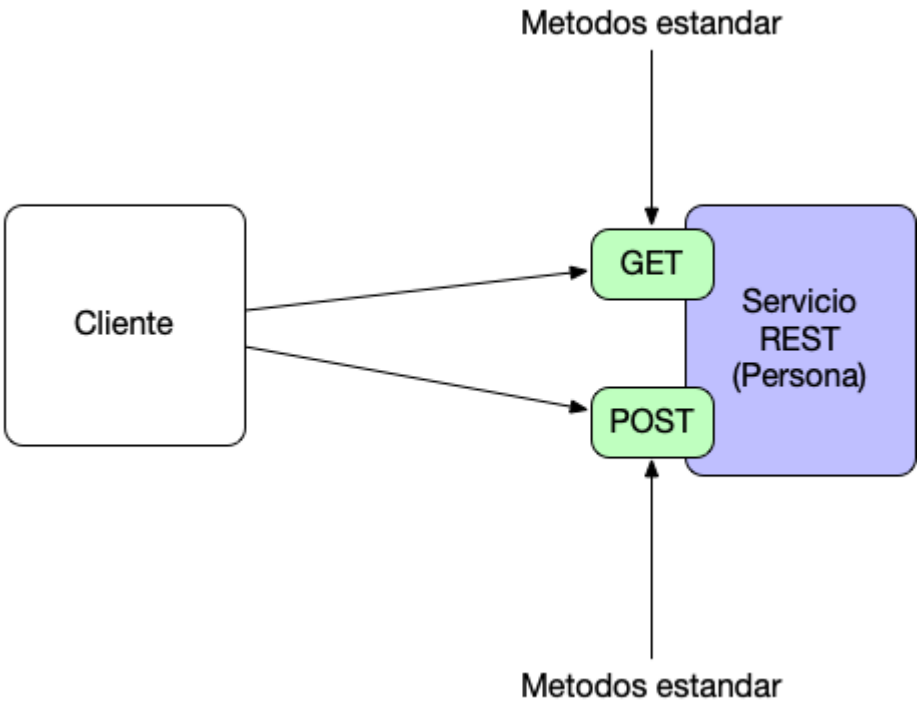
```
                        <artifactId>spring-boot-starter-
web</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
test</artifactId>
                        <scope>test</scope>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
<groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-
plugin</artifactId>
                        </plugin>
                </plugins>
        </build>

</project>
```

En este caso solo hemos incluido el starter de Web de tal forma que podamos diseñar un servicio REST y desplegarlo de forma directa. El siguiente paso es construir un servicio REST.




```
package com.arquitecturajava.rest;

import java.util.ArrayList;
import java.util.List;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/personas")
public class PersonaRESTController {

        static List<Persona> lista= new ArrayList<Persona>();
        static {
```

```
                lista.add(new Persona ("pepe",20));
                lista.add(new Persona ("juan",30));
        }
        @GetMapping
        public List<Persona> buscarTodos() {
                return lista;
        }
        @PostMapping
        public void insertar(@RequestBody Persona persona) {
                lista.add(persona);
        }
}
```

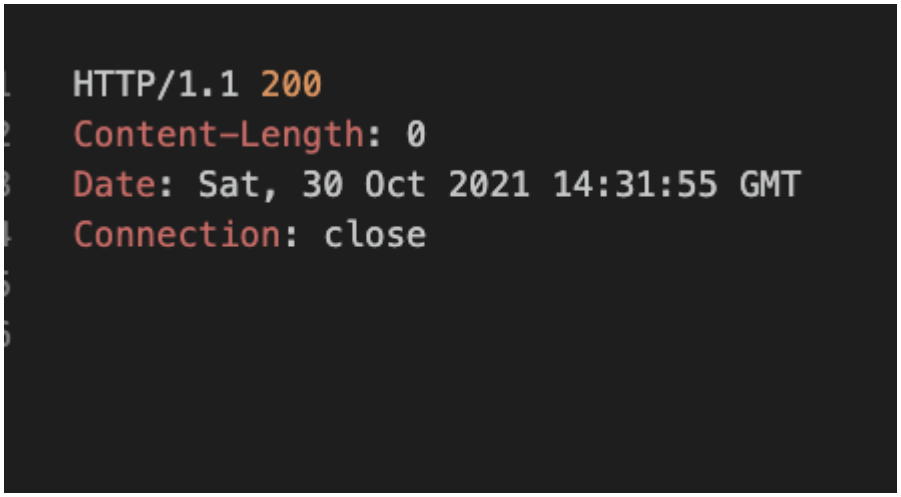
Acabamos de construir un servicio REST con dos operaciones . La primera una operación de tipo GET que nos devuelve una lista de Personas . La segunda una operación de tipo POST que inserta nuevas Personas en la lista . La operación de tipo GET es muy fácil de invocar desde el navegador y obtendremos la lista de personas.

[{"nombre":"pepe","edad":20},{"nombre":"juan","edad":30}]

@PostMapping y REST

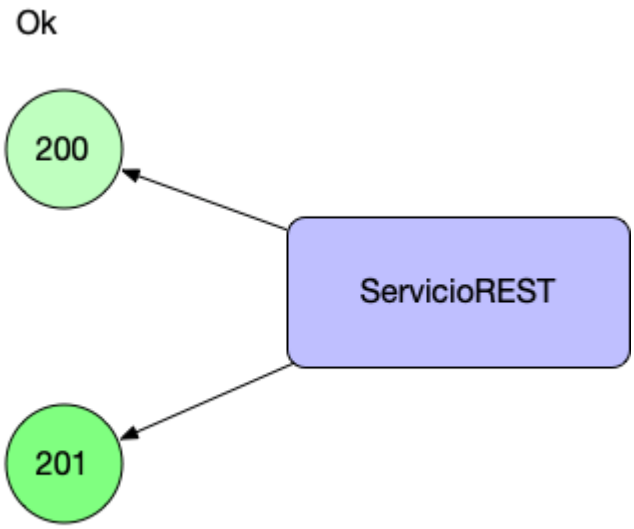
Realizar una petición POST es ya algo más complejo y necesitaremos de alguna aplicación cliente tipo PostMan o Visual Studio Code REST Client para que podamos enviar un objeto e insertarlo en la lista. En este caso he usado Visual Studio Code y REST Client como plugin lo que permite escribir el verbo POST con la URL y adjuntar los datos en formato JSON de forma muy directa.

La petición REST se realiza y obtendremos como resultado que todo ha ido bien . En este caso se generara un status code de 200. Que quiere decir que todo se ha ejecutado correctamente



ResponseEntity y anotaciones

Todo funciona correctamente pero no es suficiente ya que cuando uno realiza una petición POST debería recibir un status code de 201 si la petición se ha realizado de forma correcta . Un status code de 200 indica que todo ha ido correctamente pero un 201 es más especifico e indica que un nuevo recurso se ha añadido a la colección.



Ok + inserción

Por lo tanto el comportamiento de Spring por defecto no es el más deseado . ¿Cómo podemos cambiar este comportamiento? . Haciendo uso de la clase ResponseEntity que nos permite afinar más las respuestas.

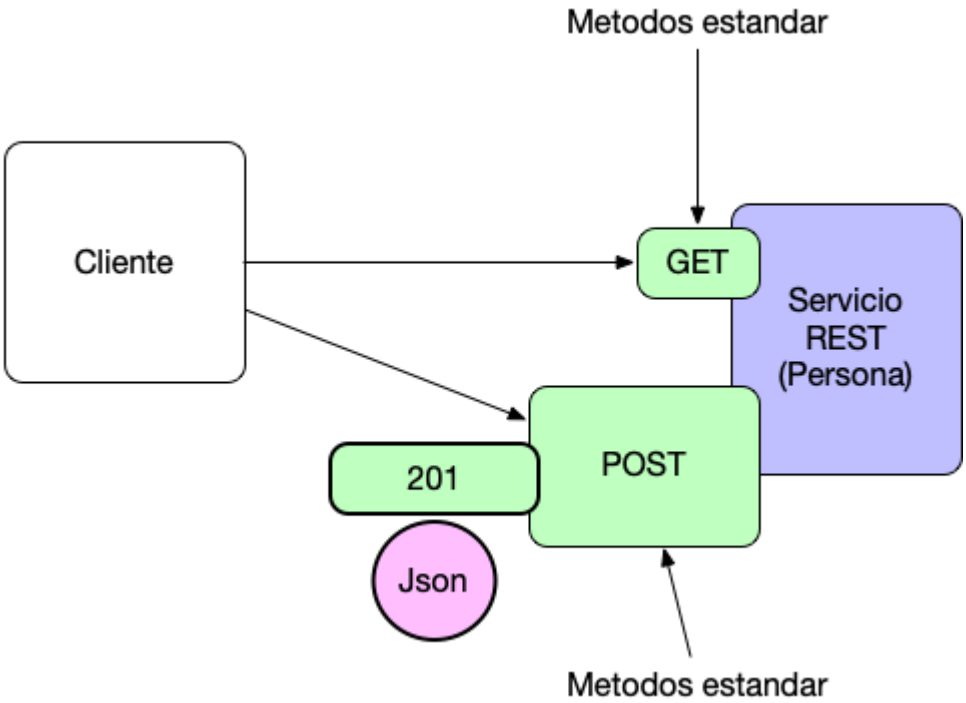
```
@PostMapping
        public ResponseEntity<Persona> insertar(@RequestBody Persona
persona) {
                lista.add(persona);
                return new
ResponseEntity<Persona>(persona,null,HttpStatus.CREATED);
        }
```

De esta forma cuando nosotros gestionemos la petición al servicio REST realizando una petición POST este servicio nos devolverá un nuevo resultado. En este caso el diseñador del API ha decidido que como respuesta de una petición de inserción se devuelve el registro insertado así como un nuevo código HTTP de 201 que significa que un recurso nuevo ha sido creado.

```
HTTP/1.1 201
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 30 Oct 2021 14:51:44 GMT
Connection: close

{
  "nombre": "gema",
  "edad": 30
}
```

Podemos ver como la respuesta en este caso incluye tanto el status code de 201 como el retorno de un objeto JSON con la información que habíamos enviado.



Otros artículos relacionados

- ¿Qué es Spring Boot?

- JSON API y Arquitecturas REST
- Introducción a Spring Data y JPA
- JAX-RS Servicios RESTFull en Java
- REST HTTP return codes y sus curiosidades