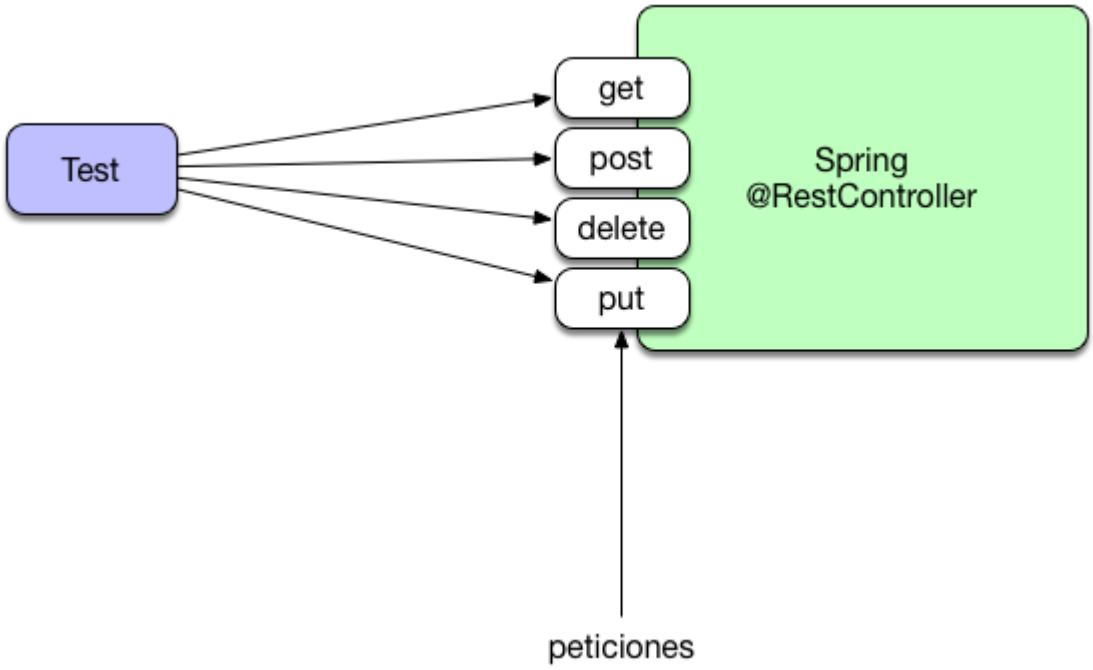
La necesidad de crear Spring REST Test cada día aumenta. Nuestras arquitecturas avanzan el uso de servicios REST se multiplica. Tenemos que comenzar a construir pruebas unitarias que de una forma sencilla puedan testear servicios REST . Ya sean servicios existentes o Mocks puros.



La necesidad de herramientas

Es imprescindible contar con alguna herramienta que nos facilite la creación de pruebas unitarias para estos servicios. REST Assured es una de estas herramientas que nos permite realizar pruebas de forma sencilla y clara. Vamos a ver como funciona ,para ello el primer paso es construir un par de Servicios REST . Vamos a utilizar Spring Spring Boot y @RestController para llevar a cabo esta tarea. A continuación se muestra el fichero Maven de partida.

```
&amp;amp;amp;lt;?xml version="1.0" encoding="UTF-8"?&amp;amp;amp;gt;
&amp;amp;amp;lt;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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</mode
lVersion>

<groupId>com.arquitecturajava&am
p;lt;/groupId>
<artifactId>rest</artifac
tId>
	<version>0.0.1-
SNAPSHOT</version>
<packaging>jar</packaging
>

<name>rest</name&
amp;gt;
	<description>Demo project for
Spring Boot</description>

	<parent>
<groupId>org.springframework.boot&am
p;amp;lt;/groupId>
		<artifactId>spring-boot-
starter-parent</artifactId>
<version>1.5.10.RELEASE</
version>
		<relativePath/>
<!-- lookup parent from repository -->
	</parent>

	<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
<java.version>1.8</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>

                <dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
<version>3.0.6</version&a

```
mp;amp;amp;gt;
&amp;amp;amp;lt;scope&amp;amp;amp;gt;test&amp;amp;amp;lt;/scope&amp;amp;amp;gt;
&amp;amp;amp;lt;/dependency&amp;amp;amp;gt;
        &amp;amp;amp;lt;/dependencies&amp;amp;amp;gt;
```

Clases y Servicios

Acabamos de definir las dependencias para SpringBoot. Es momento de ver nuestras clases Java y como trabajar con ellas. En este caso únicamente generaremos la clase Ordenador y un Servicio REST que nos devuelva información sobre ella.

```
package com.arquitecturajava.rest;

public class Ordenador {

        private String modelo;
        private String marca;
        private double precio;

        public Ordenador(String modelo, String marca, double precio) {
                super();
                this.modelo = modelo;
                this.marca = marca;
                this.precio = precio;
        }
        public Ordenador() {
                super();
        }
```

```
	public String getModelo() {
		return modelo;
	}
	public void setModelo(String modelo) {
		this.modelo = modelo;
	}
	public String getMarca() {
		return marca;
	}
	public void setMarca(String marca) {
		this.marca = marca;
	}
	public double getPrecio() {
		return precio;
	}
	public void setPrecio(double precio) {
		this.precio = precio;
	}

}



package com.arquitecturajava.rest;

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

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class OrdenadorController {
	static List&amp;amp;amp;lt;Ordenador&amp;amp;amp;gt; lista =
new ArrayList&amp;amp;amp;lt;&amp;amp;amp;gt;();
	static {

		lista.add(new Ordenador("Yoga", "Lenovo", 800));
		lista.add(new Ordenador("Air", "Apple", 1000));
	}

	@GetMapping("/ordenadores")
	public List&amp;amp;amp;lt;Ordenador&amp;amp;amp;gt;
buscarTodos() {

		return lista;
	}
&amp;amp;amp;lt;span data-mce-type="bookmark" style="display: inline-
block; width: 0px; overflow: hidden; line-height: 0;"
class="mce_SELRES_start"&amp;amp;amp;gt;&amp;amp;amp;lt;/span&amp;amp;
amp;gt;
&amp;amp;amp;lt;span data-mce-type="bookmark" style="display: inline-
block; width: 0px; overflow: hidden; line-height: 0;"
class="mce_SELRES_end"&amp;amp;amp;gt;&amp;amp;amp;lt;/span&amp;amp;am
p;gt;    @GetMapping("/ordenadores/{modelo}")
	public ResponseEntity&amp;amp;amp;lt;Ordenador&amp;amp;amp;gt;
buscarUno(@PathVariable String modelo) {
		Optional&amp;amp;amp;lt;Ordenador&amp;amp;amp;gt;
```

```
ordenador = lista.stream().filter(o -&amp;amp;amp;gt;
o.getModelo().equals(modelo)).findFirst();
                if (ordenador.isPresent()) {

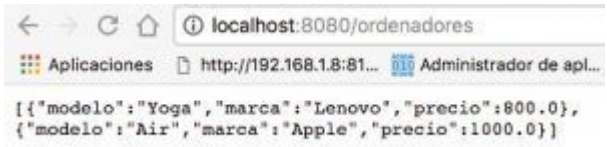
                        return new
ResponseEntity&amp;amp;amp;lt;Ordenador&amp;amp;amp;gt;(ordenador.get(
),HttpStatus.OK);
                }else {

                        return new
ResponseEntity&amp;amp;amp;lt;Ordenador&amp;amp;amp;gt;(HttpStatus.NOT
_FOUND);
                }

        }
}
```

Construcción de @RestController

Acabamos de construir un servicio REST con dos URL de acceso /ordenadores y /ordenadores/{modelo}. Si arrancamos la aplicación veremos que accedemos a los datos.



Si pedimos una url concreta con un ordenador especifico nos devolverá el ordenador:



Spring REST Test y Rest Assured

Vamos a utilizar pruebas unitarias con Rest Assured para validar los datos que nos llegan desde ambas url de forma sencilla. El primer paso es obtener las dependencias de Maven.

Una vez lo tenemos vamos a construir nuestras primeras pruebas unitarias con este framework que nos compruebe la lista de ordenadores.

```
package com.arquitecturajava.rest;

import static io.restassured.RestAssured.get;
import static org.hamcrest.Matchers.equalTo;

import org.junit.Test;

public class RESTTest {

	@Test
	public void testLista() {

		get("/ordenadores").then().body("modelo[0]",
equalTo("Yoga"));
	}

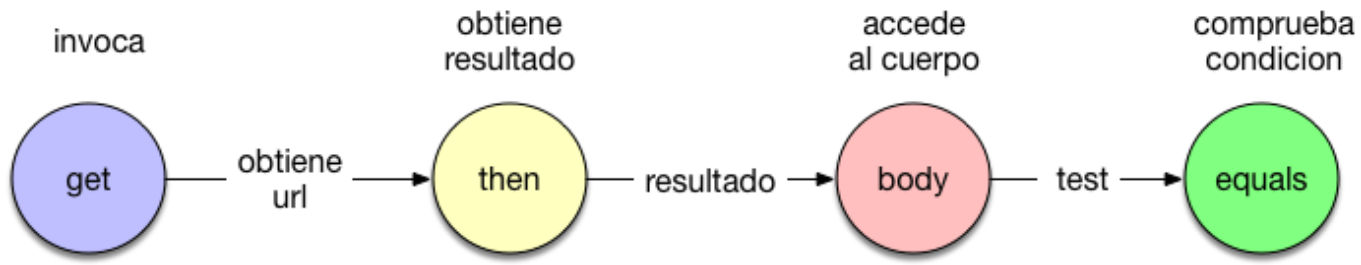
	@Test
	public void testOrdenador() {
```

```
                get("/ordenadores/Yoga").then().body("modelo",
equalTo("Yoga"));
        }

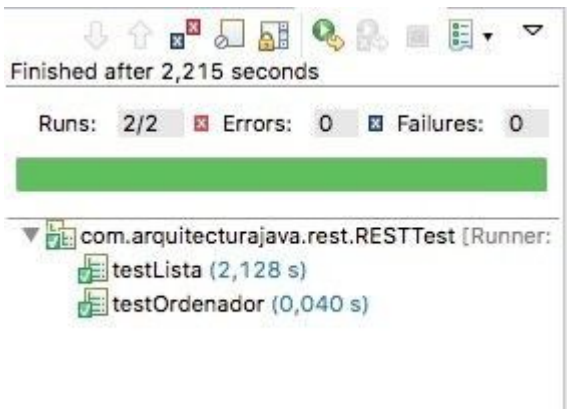
}
```

Resultado Test

Como podemos ver el DSL de Rest Assured es muy claro y permite ejecuta las pruebas unitarias de una forma muy sencilla. En este caso es suficiente con invocar el método get() que soporta programación fluida. Una vez ejecutado el método get() el método then() nos devolverá el resultado. Por último el método body() nos permite acceder a la respuesta y comprobar su contenido-



Nos queda por último ejecutar los Test y comprobar que funcionan



Acabamos de usar Rest Assured para lanzar pruebas unitarias contra servicios REST.

Otros artículos relacionados:

1. Spring REST Client con RestTemplates
2. ¿ Que es REST ?
3. Spring REST Service con @RestController
4. REST JSON y Java