El uso de la anotación @RepositoryRestResource nos puede ser muy práctica en muchas ocasiones cuando queremos construir arquitecturas REST complejas de una forma rápida dentro de Spring Framework. Normalmente cuando trabajamos con Spring Framework es relativamente común utilizar Spring Data para automatizar la gestión de repositorios . Vamos a ver un ejemplo , para ello en primer lugar mostraremos las dependencias de Maven vía Spring Boot.

```
&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"&amp;amp;amp;gt;
&amp;amp;amp;lt;modelVersion&amp;amp;amp;gt;4.0.0&amp;amp;amp;lt;/mode
lVersion&amp;amp;amp;gt;

&amp;amp;amp;lt;groupId&amp;amp;amp;gt;com.example&amp;amp;amp;lt;/gro
upId&amp;amp;amp;gt;
&amp;amp;amp;lt;artifactId&amp;amp;amp;gt;demo&amp;amp;amp;lt;/artifac
tId&amp;amp;amp;gt;
        &amp;amp;amp;lt;version&amp;amp;amp;gt;0.0.1-
SNAPSHOT&amp;amp;amp;lt;/version&amp;amp;amp;gt;
&amp;amp;amp;lt;packaging&amp;amp;amp;gt;jar&amp;amp;amp;lt;/packaging
&amp;amp;amp;gt;

&amp;amp;amp;lt;name&amp;amp;amp;gt;demo&amp;amp;amp;lt;/name&amp;amp;
amp;gt;
        &amp;amp;amp;lt;description&amp;amp;amp;gt;Demo project for
Spring Boot&amp;amp;amp;lt;/description&amp;amp;amp;gt;
```

<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.1.0.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</artifactId>
</dependency>

<dependency>
<groupId>mysql</groupId>

```
&amp;amp;amp;lt;artifactId&amp;amp;amp;gt;mysql-connector-
java&amp;amp;amp;lt;/artifactId&amp;amp;amp;gt;
&amp;amp;amp;lt;scope&amp;amp;amp;gt;runtime&amp;amp;amp;lt;/scope&amp
;amp;amp;gt;
                &amp;amp;amp;lt;/dependency&amp;amp;amp;gt;
                &amp;amp;amp;lt;dependency&amp;amp;amp;gt;
&amp;amp;amp;lt;groupId&amp;amp;amp;gt;org.springframework.boot&amp;am
p;amp;lt;/groupId&amp;amp;amp;gt;
&amp;amp;amp;lt;artifactId&amp;amp;amp;gt;spring-boot-starter-
test&amp;amp;amp;lt;/artifactId&amp;amp;amp;gt;
&amp;amp;amp;lt;scope&amp;amp;amp;gt;test&amp;amp;amp;lt;/scope&amp;am
p;amp;gt;
                &amp;amp;amp;lt;/dependency&amp;amp;amp;gt;
        &amp;amp;amp;lt;/dependencies&amp;amp;amp;gt;

        &amp;amp;amp;lt;build&amp;amp;amp;gt;
                &amp;amp;amp;lt;plugins&amp;amp;amp;gt;
                        &amp;amp;amp;lt;plugin&amp;amp;amp;gt;
&amp;amp;amp;lt;groupId&amp;amp;amp;gt;org.springframework.boot&amp;am
p;amp;lt;/groupId&amp;amp;amp;gt;
&amp;amp;amp;lt;artifactId&amp;amp;amp;gt;spring-boot-maven-
plugin&amp;amp;amp;lt;/artifactId&amp;amp;amp;gt;
                        &amp;amp;amp;lt;/plugin&amp;amp;amp;gt;
                &amp;amp;amp;lt;/plugins&amp;amp;amp;gt;
        &amp;amp;amp;lt;/build&amp;amp;amp;gt;

&amp;amp;amp;lt;/project&amp;amp;amp;gt;
```

Una vez definidas las dependencias es momento de ver el contenido del fichero application.properties que define una serie de parametrizaciones por defecto para JPA y la

url de acceso REST para los repositorios.

spring.datasource.url=jdbc:mysql://localhost:8889/springjpa
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver.class=com.mysql.jdbc.Driver
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
spring.data.rest.basePath=/webapi

Una vez configuradas estas dos cosas el siguiente paso es construir el código necesario para trabajar con JPA y servicios REST. En primer lugar definimos la clase:

```
package com.arquitecturajava.springrest;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Persona {

	@Id
	private String nombre;
	private String apellidos;
	private int edad;
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
```

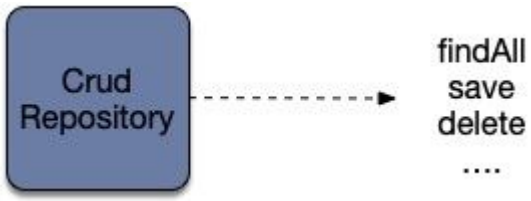
```
	}
	public String getApellidos() {
		return apellidos;
	}
	public void setApellidos(String apellidos) {
		this.apellidos = apellidos;
	}
	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}
	public Persona(String nombre, String apellidos, int edad) {
		super();
		this.nombre = nombre;
		this.apellidos = apellidos;
		this.edad = edad;
	}
	public Persona() {
		super();
	}

}
```

@RepositoryRestResource

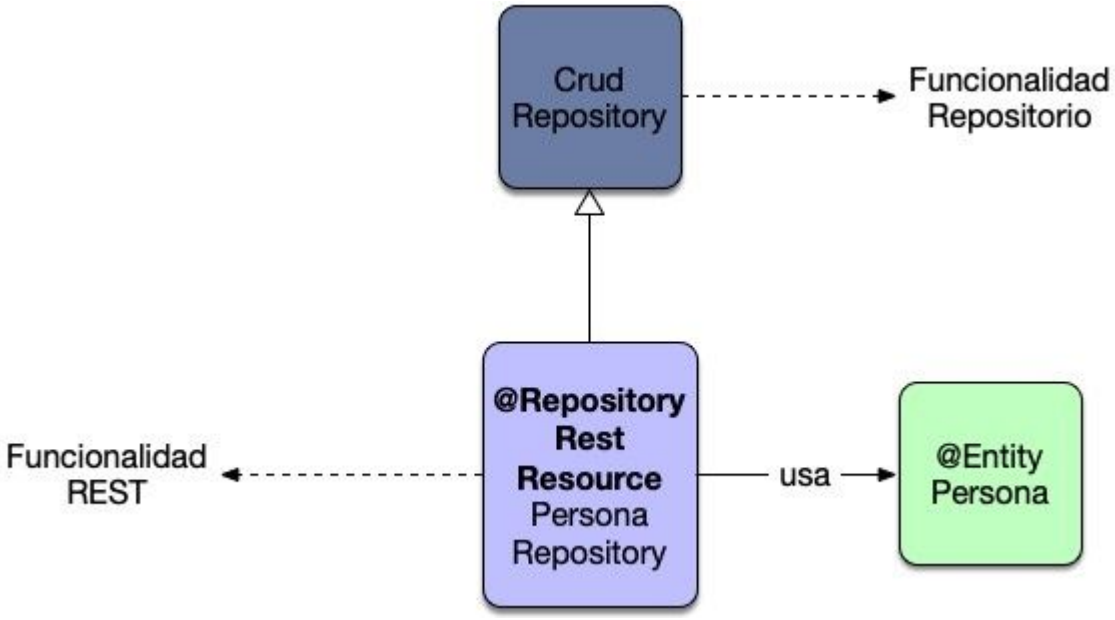
En segundo lugar tenemos que implementar el interface de Spring Data que se encargará de automatizar la creación del repositorio y publicarlo vía REST.

```
package com.arquitecturajava.springrest;

import org.springframework.data.repository.CrudRepository;
import
org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource(path="/personas")
public interface PersonaRepository extends
CrudRepository&amp;amp;amp;lt;Persona, String&amp;amp;amp;gt;{

}
```

Usamos el interface CrudRepository para implementar de forma automática las operaciones de persistencia más básicas.



Añadimos a este interface la anotación @RepositoryRestResource esta anotación se encargará de publicar el repositorio cómo servicio REST y acceder a las operaciones fundamentales que todo repositorio implementa.



Una vez hecho esto el siguiente paso es arrancar la aplicación de Spring Boot y comprobar que en la URL personas accedemos la información de estas sin problema.

```
{
  "_embedded" : {
    "personae" : [ {
      "apellidos" : "sanchez",
      "edad" : 30,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/webapi/personas/ana"
        },
        "persona" : {
          "href" : "http://localhost:8080/webapi/personas/ana"
        }
      }
    }, {
      "apellidos" : "perez",
      "edad" : 20,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/webapi/personas/pedro"
        },
        "persona" : {
          "href" : "http://localhost:8080/webapi/personas/pedro"
        }
      }
    } ]
  },
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/webapi/personas"
    },
    "profile" : {
      "href" : "http://localhost:8080/webapi/profile/personas"
    }
  }
}
```


Como vemos se accede a la información y ademas disponemos de links directos a un mayor detalle . El uso de Spring Data REST puede ser muy útil en nuestros proyectos si tenemos enfoque de arquitecturas SPA por ejemplo. Aprendamos a usar este framework y eliminemos muchos de los problemas y dudas que nos aparecen cuando construimos este tipo de arquitecturas.

Otros artículos relacionados:

1. Spring REST Service con @RestController
2. Spring MVC @RequestMapping

3. Spring MVC Flash Attributes
4. Swagger documentando nuestro API REST
5. Arquitecturas REST y sus niveles
6. Spring Data REST