El uso REST Nested Resources es cada día más necesario cuando construimos arquitecturas REST . En muchos casos estas arquitecturas pueden funcionar de una forma razonable utilizando los clásicos Recursos y verbos HTTP . Ahora bien el uso de recursos REST standard no siempre solventa todos los problemas que este tipo de Arquitecturas puede generar. Imaginemos que estamos ante una situación en la que tenemos dos Recursos REST Personas y Deportes. Para ello el primer paso es crear las clases y relacionarlas , más adelante las convertiremos con Spring Boot en recursos.

```
package com.arquitecturajava.embebidos;

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

import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonInclude.Include;

public class Persona {

  private String nombre;
  private String apellido;
  private int edad;
  @JsonInclude(Include.NON_EMPTY)
  private List<Deporte> deportes = new ArrayList<>();

  public List<Deporte> getDeportes() {
    return deportes;
  }

  public void setDeportes(List<Deporte> deportes) {
    this.deportes = deportes;
  }
```

```
public void addDeporte(Deporte deporte) {

  this.deportes.add(deporte);
}

public Persona(String nombre, String apellido, int edad) {
  super();
  this.nombre = nombre;
  this.apellido = apellido;
  this.edad = edad;
}

public Persona(Persona p) {

  this(p.getNombre(), p.getApellido(), p.getEdad());
}

public String getNombre() {
  return nombre;
}

public void setNombre(String nombre) {
  this.nombre = nombre;
}

public String getApellido() {
  return apellido;
}

public void setApellido(String apellido) {
  this.apellido = apellido;
```

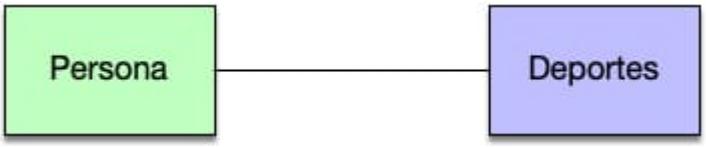
```
  }

  public int getEdad() {
    return edad;
  }

  public void setEdad(int edad) {
    this.edad = edad;
  }
}
```

Podemos observar que estamos ante una Persona que incluye una lista de Deportes . En nuestro caso la clase Deporte será muy muy básica y solo contendrá el nombre del Deporte que practicamos.

```
package com.arquitecturajava.embebidos;

import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonInclude.Include;

@JsonInclude(Include.NON_NULL)
public class Deporte {

  private String nombre;

  public String getNombre() {
    return nombre;
  }

  public void setNombre(String nombre) {
    this.nombre = nombre;
  }
```

```
  public Deporte(String nombre) {
    super();
    this.nombre = nombre;
  }
}
```

Ambas clases se encuentran relacionados a través del ArrayList de Deportes que contiene la Persona .



Es momento de construir un ejemplo con Spring Boot que nos publique la información de estos conceptos como API REST. Para ello como siempre usaremos Spring Initializr y nos construiremos un proyecto Web.

```
<?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>2.1.8.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.arquitecturajava</groupId>
  <artifactId>embebidos</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>embebidos</name>
```

```
  <description>Demo project for Spring Boot</description>

  <properties>
    <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>
  </dependencies>

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

Creaomos RestController que habilite la posibilidad de seleccionar un listado de Personas.

```
package com.arquitecturajava.embebidos;

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

```
import java.util.Optional;
import java.util.stream.Collectors;

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 PersonaController {
  static List<Persona> lista=new ArrayList<>();
  static {
    Persona p1=new Persona("juan","sanchez",20);
    p1.addDeporte(new Deporte ("padel"));
    lista.add(p1);
    Persona p2= new Persona("ana","gomez",30);
    p2.addDeporte(new Deporte ("natacion"));
    lista.add(p2);
    Persona p3= new Persona("juan","sanchez",20);
    p3.addDeporte(new Deporte ("futbol"));
    lista.add(p3);
  }

  @GetMapping("/personas")
  public List<Persona> personas() {
    return
lista.stream().map(Persona::new).collect(Collectors.toList());
  }
  @GetMapping("/personas/{nombre}")
  public Persona persona(@PathVariable String nombre) {
```

```
    Optional<Persona>
oPersona=lista.stream().filter(p->p.getNombre().equals(nombre)).findFi
rst();
    if (oPersona.isPresent()) {
      return oPersona.get();
    }
    return null;
  }
  @GetMapping("/personas/{nombre}/deportes")
  public ResponseEntity<List<Deporte>> deportesPersona(@PathVariable
String nombre) {
    Optional<Persona>
oPersona=lista.stream().filter(p->p.getNombre().equals(nombre)).findFi
rst();
    if (oPersona.isPresent()) {
      return new
ResponseEntity<List<Deporte>>(oPersona.get().getDeportes(),
HttpStatus.OK);
    }
    return new ResponseEntity<>(new
ArrayList<Deporte>(),HttpStatus.NOT_FOUND);
  }
  @GetMapping("/personas/-/deportes")
  public List<Persona> personasConDeportes() {
    return lista;
  }
```

REST Resources y Agregaciones

Lo primero que podemos observar en este Controlador es que tenemos la posibilidad de obtener una lista de Personas con @GetMapping(“/personas”) . Esto nos devolvera la lista de Personas con sus datos.

```
// 20190927154254
// http://localhost:8080/personas

[
  {
    "nombre": "juan",
    "apellido": "sanchez",
    "edad": 20
  },
  {
    "nombre": "ana",
    "apellido": "gomez",
    "edad": 30
  },
  {
    "nombre": "juan",
    "apellido": "sanchez",
    "edad": 20
  }
]
```

Esta es la url más habitual y nos devuelve una lista de Recursos. El segundo caso más habitual es cuando nosotros queremos localizar un único recurso y pasamos a este URL como información adicional el identificador de este.

```
// 20190927154430
// http://localhost:8080/personas/juan

{
  "nombre": "juan",
  "apellido": "sanchez",
  "edad": 20,
```

```
  "deportes": [
    {
      "nombre": "padel"
    }
  ]
}
```

Como podemos ver obtenemos la persona y su deporte (esto último sería opcional) . Es momento de acceder a los Deportes de cada una de las Personas @GetMapping("/personas/{nombre}/deportes") . Esto nos devuelve vía un agregado los deportes de una persona en concreto.

```
// 20190927154630
// http://localhost:8080/personas/juan/deportes

[
  {
    "nombre": "padel"
  }
]
```

REST Nested Resources

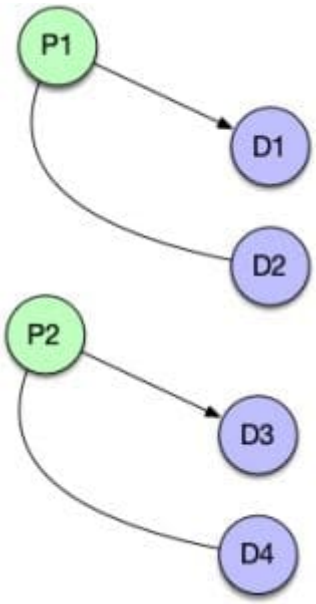
Todo es correcto . Sin embargo hay situaciones en las que necesitamos obtener de golpe ambos recursos anidados (REST Nested Resources) . ¿Cómo podemos hacer esto? . Bueno normalmente siempre hay un paso previo en el cual a través de fetchAPI o alguna solución similar de nuestro framework de persistencia obtenemos el grafo de objetos . Una vez que ese grafo le tenemos disponible es cuestión de publicarlo vía una url que informe sobre la relación . Es el caso de @GetMapping("/personas/-/deportes") . Esta url nos indica que son las personas con sus deportes.

```
// 20190927194617
```

```
// http://localhost:8080/personas/-/deportes

[
  {
    "nombre": "juan",
    "apellido": "sanchez",
    "edad": 20,
    "deportes": [
      {
        "nombre": "padel"
      }
    ]
  },
  {
    "nombre": "ana",
    "apellido": "gomez",
    "edad": 30,
    "deportes": [
      {
        "nombre": "natacion"
      }
    ]
  },
  {
    "nombre": "juan",
    "apellido": "sanchez",
    "edad": 20,
    "deportes": [
      {
        "nombre": "futbol"
      }
```

```
      ]
   }
]
```

Así pues habremos abordado con garantías el concepto de REST Nested Resources lo que nos permitirá reducir las peticiones a nuestro API REST. Ya que una sola llamada devuelve el grafo de objetos completo.



Otros artículos relacionados

1. REST DTO y JSON Arquitecturas Web y objetos
2. JSON API y Arquitecturas REST
3. ¿ Que es REST ?
4. https://jsonapi.org/