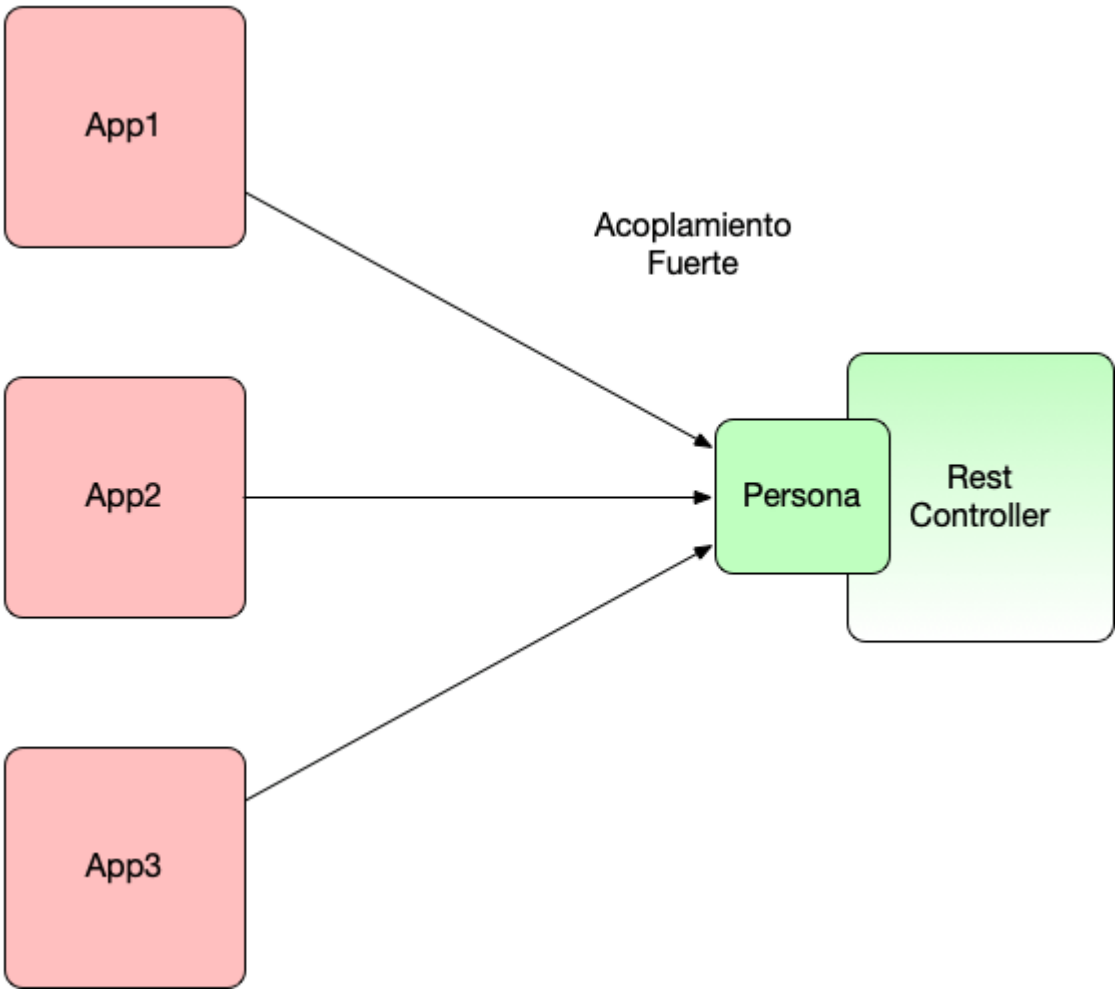
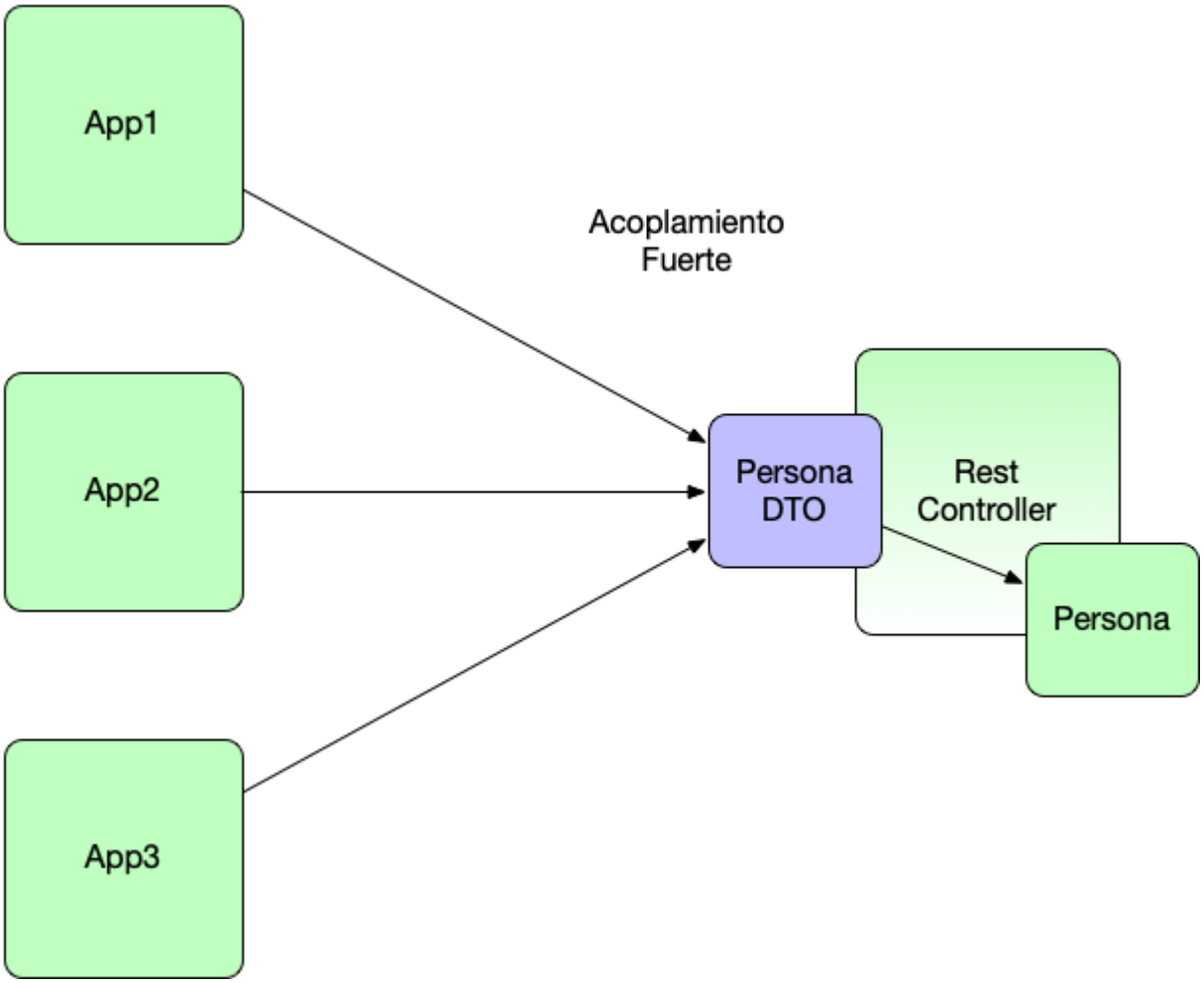
Manejar REST Java Record es una de las cosas que poco a poco se irán implantando en las aplicaciones Web desarrolladas con Java sea el framework que sea. Todos usamos REST a la hora de publicar datos que otras aplicaciones van a consumir. La pregunta que mucha gente se hace a veces cuando vamos a publicar datos en REST es si publicamos directamente objetos de negocio como Persona o publicamos PersonasDTO. Cualquier opción es posible pero recordemos que si publicamos directamente objetos Persona las aplicaciones cliente que conectemos a nuestro servidor quedaran muy fuertemente acopladas.



En muchas ocasiones se opta por añadir una clase intermediaria que se denomina PersonaDTO y reduce el acoplamiento ya que el cliente conoce el DTO pero no la estructura interna de la entidad



A partir de ahora podemos diseñar estas clases como Java Records y conseguir una simplificación del código y una inmutabilidad del objeto que transferimos:

REST Java Records

Por lo tanto tendremos que diseñar tanto la clase Persona como su Record:

```
package com.arquitecturajava.records;

public class Persona {

        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 record PersonaRecord(String nombre, String apellidos,int edad)
{

}
```

Una vez hechas operaciones diseñaremos una clase de Servicio que devuelve un objeto de negocio para que finalmente el controlador lo transforme en un Java Record:

```
package com.arquitecturajava.records;

import java.util.List;

import org.springframework.stereotype.Service;

@Service
public class PersonaService {

	public List<Persona> buscarTodos() {
		return List.of (new Persona("pepe","perez",20),new
Persona ("ana","gomez",30));
	}
}
```

```
package com.arquitecturajava.records;

import java.util.List;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class PersonaController {

	private PersonaService servicio;

	public PersonaController(PersonaService servicio) {
		super();
```

```
                this.servicio = servicio;
        }
        @RequestMapping("/personas")
        public List<PersonaRecord> buscarTodos() {
                return servicio.buscarTodos().stream().map((p)->new
PersonaRecord(p.getNombre(), p.getApellidos(),p.getEdad())).toList();
        }

}
```

Acabamos de publicar la información como REST Java Record si ejecutamos el proyecto de Spring Boot e invocamos la URL nos funcionara sin ningún problema como si hubieramos publicado un DTO clásico .

```
// 20230831101943
// http://localhost:8080/personas

[
  {
    "nombre": "pepe",
    "apellidos": "perez",
    "edad": 20
  },
  {
    "nombre": "ana",
    "apellidos": "gomez",
    "edad": 30
  }
]
```

Recordemos que los Java Records han sido diseñados para gestionar datos puros.

Otros Artículos relacionados

- Java Record Class y JDK 14
- Servicios REST (@FormParam,@PathParam)
- Entity to DTO y Java 8
- REST Nested Resources y su utilidad
- Curso Spring REST
- Java Records Oracle