MAS RESTFUL QUE NUNCA

Summary

Put vs. Patch. *

Validación.

Documentación.

Paginado y filtros *

Error Handling.

Los ejemplos que realizaremos de los temas marcados con * serán hechos con las herramientas que conocemos hasta ahora para pensar y ganar más habilidad con las mismas. Voy a dejar las herramientas que yo use para realizarlos las cuales utilicé o las vi siendo utilizadas por compañeros.

Put vs. Patch

Vamos a utilizar Patch cuando queramos recibir el objeto solamente con los atributos que se necesiten modificar.

Esto nos ahorra enviar la entidad completa a la hora de actualizar. Nos da más flexibilidad ya que no tenemos necesidad de conocer la entidad completa y facilita el trabajo con entidades muy grandes.

Para investigar: `JsonMergePatch` de `javax.json-api`

Validation

JSR 380 es una especificación de la API de Java para la validación de beans, parte de Jakarta EE y JavaSE, que garantiza que las propiedades de un bean cumplan criterios específicos, utilizando anotaciones como `@NotNull`, `@NotBlank` `@Min` y `@Max`. Esto, en conjunto con la anotación `@Valid` nos permitirá validar los campos de una manera más sencilla.

Generalmente, ya viene implementado con Hibernate, ya que ellos tienen su implementación del JSR 380. También puede ser utilizado agregando la siguiente dependencia.

```
<dependency>
  <groupId>javax.validation</groupId>
  <artifactId>validation-api</artifactId>
  <version>2.0.0.Final</version>
</dependency>
```

Error Handling

Spring nos provee de anotaciones para poder manejar la respuesta de error que dan nuestras Exceptions. Poder interceptarlas y dar la respuesta que necesitemos.

`@ExceptionHandler`: Nos permite tratar Exceptions a nivel Controller.

`@ControllerAdvice`: Nos permite tratar Exceptions a nivel Proyecto.

`@ResponseStatus`: Nos permite setear un StatusCode a nuestras Exceptions.

`throw new ResponseStatusException()`: Una Exception que viene a partir de Spring 5 la cual nos permite enviarle un Status Code por constructor.

Paginado

El paginado separa los resultados de una request en páginas todas de un tamaño. Toda esta información puede ser customizada por el cliente.

¿Por qué necesitamos paginado?

Por la eficiencia. Siempre es más barato procesar menos datos.

Por la experiencia de usuario (UX). Damos una respuesta mas rapida y prolija.

Spring nos proveé de la Interfaz `Pageable` que, ahora nos ayudará a recibir los datos necesarios para paginar, pero más adelante nos ahorrará mucho trabajo en conjunto con Spring Data.

Documentación

Una herramienta muy útil es Swagger que nos permite documentar nuestra API de una manera rápida y sencilla.

```
@EnableSwagger2
```

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.9.2</version>
</dependency>
```

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.9.2</version>
</dependency>
```