Manejo de errores con ResponseStatusException

Desarrollo de un API REST con Spring Boot

¿Hemos dado un paso atrás?

- Bueno….
- *ResponseStatusException* nos recuerda a *@ResponseStatus*
- No son lo mismo, pero están relacionados.
- El modelo para manejar errores que presentamos en este vídeo es compatible con lo visto hasta ahora.

ResponseStatusException

- Disponible desde Spring 5
- Se trata como cualquier otra excepción (*throw new …*)
- Nos permite indicar
  - Estado (HttpStatus) (obligatorio)
  - razón (String) (opcional).
  - causa (Throwable) (opcional).

Ejemplo

```
public ResponseEntity<?> obtenerTodos() {
     List<Producto> result = productoRepositorio.findAll();
     if (result.isEmpty()) {
          throw new ResponseStatusException(
               HttpStatus.NOT_FOUND, "No hay productos registrados");
     } else {
          // Resto del código
     }
}
```

Manejo junto a otras excepciones

- Nos permite seguir reutilizando nuestras excepciones

```
@GetMapping("/producto/{id}")
public Producto obtenerUno(@PathVariable Long id) {
          try {
                    return productoRepositorio.findById(id)
                              .orElseThrow(() -> new
ProductoNotFoundException(id));
          } catch (ProductoNotFoundException ex) {
                    throw new ResponseStatusException(
               HttpStatus.NOT_FOUND, ex.getMessage());
          }
}
```

Ventajas

- Muy bueno al empezar a desarrollar nuestra aplicación
  - Manejo de errores con poco esfuerzo
- Un tipo de excepción puede llevar asociados, en diferentes lugares, diferentes tipos de código de estado .
- No necesitamos tantas clases de excepción personalizadas.
- Más control del manejo de excepciones (se lanzan programáticamente).

Desventajas

- Perdemos la globalidad ganada con @ControllerAdvice
- Duplicación de código.
- El modelo de error vuelve a ser el estándar.

Conclusión

- Podemos combinar *@ControllerAdvice* para elementos globales, con *ResponseStatusException* para elementos puntuales o más específicos.
- Cuidado con manejar un tipo de excepción más de una vez (ResponseStatusException + @ControllerAdvice).

Modelo de error

- Hemos vuelto al modelo estándar. ¿No podemos hacer nada?
- Modificar el modelo estándar.
- Debemos crear un @Component que extienda a *DefaultErrorAttributes*.
- Sobreescribimos el método *getErrorAttributes*. Este devuelve un map a partir del cual se generará el JSON.

Retos

- Extender este tipo de manejo de error al resto de métodos del controlador.
- Jugar con los posibles campos en el modelo de error.
    - Si queremos anidar objetos en nuestro JSON, no tenemos más que añadir un Map como valor para una de las entradas del Map más general.