Manejo de Excepciones con Spring Boot

DigitalHouse>

Certified Tech Developer
The Ultimate Degree

Índice

1 Anotación @ExceptionHandler

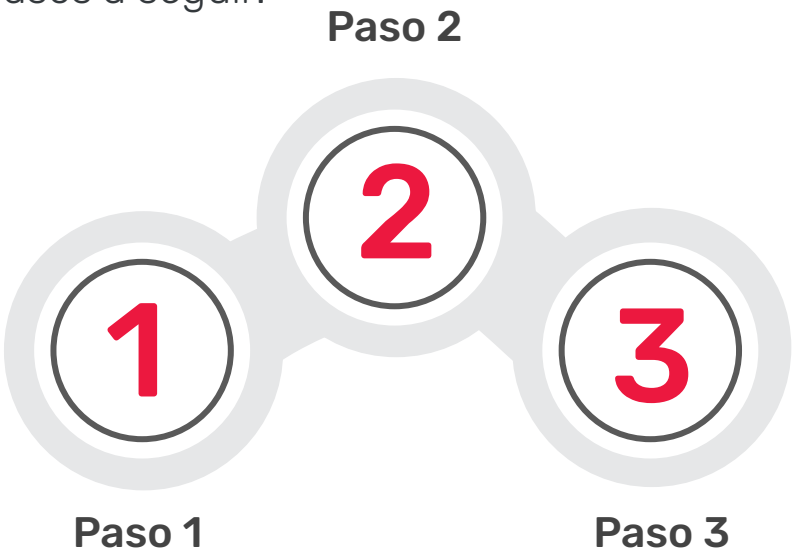
@ExceptionHandler

La primera opción es utilizar esta anotación para *manejar las excepciones a nivel de controller,* es decir, dentro de nuestro controller implementamos un método que se ejecutará en caso de producir un error.
Por ejemplo cuando tenemos un error donde el recurso no se ha localizado, aparece el error 404:

```
{
       "timestamp": 1512713804164,
      "status": 404,
      "error": "Not Found",
      "message": "No message available",
      "path": "/some-url"
}
```

Podemos especificar el estado de respuesta para una excepción específica, junto con la definición de la excepción con la anotación **@ResponseStatus**.

Veamos cuáles son los pasos a seguir:



Paso 1

Si necesitamos capturar una excepcion propia creada por nosotros debemos primero crear nuestra propia Excepcion.

```
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends Exception{

  public ResourceNotFoundException(String  mensaje){
    super(mensaje);
  }
}
```

Paso 2

Desde algún método del controller podría lanzarse una excepción del tipo ResourceNotFoundException

```
@RestController
public class MiController{
  @RequestMapping(value = "/files/{id}")
   public String getFile(@PathVariable("id")long id){

    if(id <= 0){
      String mensajeError = "No se encuentra ningun archivo con
id" + id;
      throw new ResourceNotFoundException(mensajeError);
    }else{
        ...
    }
  }
}
```

Paso 3

Luego, capturamos la excepcion con @ExceptionHandler

```
@RestController
public class MiController{


  @ExceptionHandler(Exception.class)
  private ResponseEntity<?> exception(ResourceNotFoundException ex, WebRequest request)
}
```

Este método captura las excepciones de tipo Exception o sea cualquier excepcion de Java.
Con Exception Handler podemos capturar tanto excepciones propias de Java (Excepcion, NullPointerException, etc) como excepciones propias creadas por nosotros.

Es buena práctica loguear los errores con log4j:

```
log.error(ex);
return new ResponseEntity<>("Error manejado por exception Handler",
HttpStatus.NOT_FOUND);
```

Captura las excepciones de tipo ResourceNotFoundException que se den dentro del controller MiController.

```
@ExceptionHandler(ResourceNotFoundException.class)
public ResponseEntity<?>
resourceNotFoundException(ResourceNotFoundException ex,
                          WebRequest request) {
}
```

Es buena práctica loguear los errores con log4j:

```
log.error(ex);
return new ResponseEntity<>("Error manejado por exception Handler",
HttpStatus.NOT_FOUND);
```

Este enfoque tiene un gran inconveniente: el método anotado @ExceptionHandler solo está activo para ese controlador en particular, no globalmente para toda la aplicación. Es decir tendríamos que copiar y pegar el código en todos los controllers para poder procesar este error. Por supuesto, agregar esto a cada controlador hace que no sea adecuado para un mecanismo general de manejo de excepciones.

Podemos solucionar esta limitación haciendo que todos los controladores hereden una clase de controlador base. Sin embargo, esta solución puede ser un problema para aplicaciones donde, por cualquier motivo, eso no es posible. Por ejemplo, los controladores ya pueden extenderse desde otra clase base o puede que ellos mismos no sean modificables directamente.

2 Excepciones globales @ControllerAdvice

@ControllerAdvice

La anotación @ControllerAdvice nos permite juntar nuestros múltiples @ExceptionHandlers dispersos en un solo componente global de manejo de errores.

El mecanismo es extremadamente simple pero también muy flexible:

- Nos da un control total sobre el cuerpo de la respuesta (Response body), así como sobre el código de estado (response status).
- Proporciona mapeo de varias excepciones al mismo método, para ser manejadas juntas.
- Además podemos usar ResponseEntity para el response.

Veamos un ejemplo:

```
@ControllerAdvice
public class GlobalExceptionHandler {

      @ExceptionHandler(ResourceNotFoundException.class)
      public ResponseEntity<?> resourceNotFoundException(ResourceNotFoundException ex,
WebRequest request) {

            return new ResponseEntity<>(“Error manejado por Exception Handler”,
HttpStatus.NOT_FOUND);
      }

      @ExceptionHandler(Exception.class)
      public ResponseEntity<?> globleExcpetionHandler(Exception ex, WebRequest request) {

            return new ResponseEntity<>(“Error manejado por Exception Handler”,
HttpStatus.INTERNAL_SERVER_ERROR);
      }
}
```

3 Excepciones a nivel de método

ResponseStatusException

En el caso que necesitemos tratar un error particular, y utilizar @ExceptionHandler a nivel de clase no nos sea eficiente, podemos usar la clase ResponseStatusException directamente en un método en particular de un controller. Veamos un ejemplo:

```
@GetMapping(value = "/{id}")
public Persona findById(@PathVariable("id")Long id, HttpServletResponse response){
  try {
       Persona persona = service.findOne(id);

      return persona;
      }

    catch (MyResourceNotFoundException ex){
       throw new ResourceNotFoundException(HttpStatus.NOT_FOUND, "Persona Not Found", ex);

    }
}
```

DigitalHouse>