Manejo de errores con *@ControllerAdvice*

Desarrollo de un API REST con Spring Boot

Hasta ahora

- Hemos podido manejar excepciones.
- El mensaje de error es personalizado.
- ¿Qué nos falta?
    - Gestión global de errores.
- *@ExceptionHandler* se ubica a nivel de método y funciona a nivel de clase.

@ControllerAdvice

- Especialización de *@Component* para clases que declaran los métodos *@ExceptionHandler*, *@InitBinder* o *@ModelAttribute* para ser compartidos entre múltiples clases de *@Controller*.
- *@RestControllerAdvice* es una especialización que unifica *@ControllerAdvice* y *@ResponseBody*.
- Para manejar una excepción, se escogerá el primer método dentro de la clase *@ControllerAdvice* que esté anotado para trata la excepción (con *@ExceptionHandler*).

@ControllerAdvice

- Puede existir más de una clase anotada con *@ControllerAdvice*.
- En tal caso, puede ser recomendable el uso de *@Order* o *@Priority* para establecer una predecencia en el tratamiento de errores.
- En caso de varias opciones para una excepción dentro de una clase, escogerá la más cercana a la raíz (*FileNotFoundException* vs *IOException*).

@ControllerAdvice

- Si no indicamos nada, la anotación hace que la clase trate posibles excepciones producidas en cualquier controlador.
- Podemos acotar el radio de acción
  - `@ControllerAdvice("my.chosen.package")`
  - `@ControllerAdvice(value = "my.chosen.package")`
  - `@ControllerAdvice(basePackages = "my.chosen.package")`
  - `@ControllerAdvice(basePackageClasses = MyClass.class)`
  - `@ControllerAdvice(assignableTypes = MyController.class)`
  - `@ControllerAdvice(annotations = RestController.class)`

Un primer ejemplo

```
@RestControllerAdvice
public class GlobalControllerAdvice {
        // Manejo de errores personalizado
}
```

- Incluimos dentro todos los *@ExceptionHandler* que hemos definido hasta ahora

Algunas pequeñas mejoras en la clase modelo de error

```
@Setter @Getter @RequiredArgsConstructor @NoArgsConstructor
public class ApiError {
	@NonNull
	private HttpStatus estado;
	@JsonFormat(shape = Shape.STRING, pattern = "dd/MM/yyyy
hh:mm:ss")
	private LocalDateTime fecha = LocalDateTime.now();
	@NonNull
	private String mensaje;

}
```

Retos

- Es un buen momento para ir entrenándonos en lo que ya sabemos, y crear el servicio para las categorías. Debe tener:
  - GET /categoria
  - GET /categoria/{id}
  - POST /categoria/
  - PUT /categoria/{id}
  - DELETE /categoria/{id}

Retos

- Puedes crear algunas subclases de ApiError, que nos permitan no tener que indicar el estado, como NotFoundApiError, y que ya incluya HttpStatus.NOT_FOUND como estado.