

# MANEJO DE EXCEPCIONES

PROGRAMACIÓN 3  
PROF. MANGO EDUARDO



# CONTENIDO

01

@ExceptionHandler

02

@ControllerAdvice

03

Uso de mensajes personalizados



# Manejo de excepciones

## ¿QUE ES?

El manejo de excepciones es una parte fundamental en el desarrollo de aplicaciones robustas con Spring Boot. Una forma efectiva de manejar errores es mediante la anotación `@ExceptionHandler`, que permite capturar y gestionar excepciones sin necesidad de atraparlas donde surjan con el típico bloque try-catch, lo que genera un código mucho menos verboso y más legible.

# @ExceptionHandler

Podemos definir un método dentro de un controlador para manejar una excepción específica utilizando @ExceptionHandler. Este método se ejecutará automáticamente cuando ocurra la excepción indicada.



```
@RestController
@RequestMapping("/api")
public class UsuarioController {

    @GetMapping("/usuario/{id}")
    public Usuario obtenerUsuario(@PathVariable Long id) {
        return usuarioService.obtenerPorId(id)
            .orElseThrow(() -> new
UsuarioNoEncontradoException("Usuario no encontrado con ID: " + id));
    }

    @ExceptionHandler(UsuarioNoEncontradoException.class)
    public ResponseEntity<String>
manejarUsuarioNoEncontrado(UsuarioNoEncontradoException ex) {
        return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
    }
}
```



# @ControllerAdvice

@ControllerAdvice nos permite manejar de manera global cualquier excepción que surja en un controlador o que sean propagadas hasta uno de estos.

Si una excepción es lanzada fuera de este contexto, no será atrapada y manejada correctamente por los @ExceptionHandler, por lo que debe ser manejada utilizando otra estrategia.

The word "ADVICE" is rendered in large, bold, 3D yellow block letters. It is positioned on a blue surface that reflects the letters, creating a subtle shadow. The background is a solid blue color. The letters are slightly offset from the bottom right corner of the frame.

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(UsuarioNoEncontradoException.class)
    public ResponseEntity<String>
manejarUsuarioNoEncontrado(UsuarioNoEncontradoException ex) {
        return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> manejarExcepcionGeneral(Exception ex)
{
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Ocurrió un
error inesperado: " + ex.getMessage());
    }
}
```



# Mensajes de error personalizados

Para ofrecer mejores respuestas ante errores, es posible declarar una clase que contenga la información que se desea devolver en caso de una excepción. Es importante que se devuelva una información detallada de porque fallo la petición.

```
name" => null
username" => null
username" => "admin"
gender" => null
email" => "info@mecanb
email_verified_at" =>
password" => "$2y$10$1
isActive" => 1
user_role" => "Adminis
avatar" => "assets/ime
remember_token" => "0c
created_at" => "2022-1
updated_at" => "2022-1
```



```
@ExceptionHandler(UsuarioNoEncontradoException.class)
public ResponseEntity<ErrorDetalles>
manejarUsuarioNoEncontrado(UsuarioNoEncontradoException ex, WebRequest
request) {
    ErrorDetalles errorDetalles = new ErrorDetalles(ex.getMessage(),
request.getDescription(false));
    return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(errorDetalles);
}
```

Esto devolverá una respuesta como:

```
{
  "timestamp": "2025-03-27T12:34:56",
  "mensaje": "Usuario no encontrado con ID: 1",
  "detalles": "uri=/api/usuario/1"
}
```

# MANEJO DE EXCEPCIONES

PROGRAMACIÓN 3  
PROF. MANGO EDUARDO



Universidad Tecnológica Nacional de Mar del Plata