

Caso Buscar Producto por ID con Spring



Implementación del caso para buscar un producto por ID en Spring Boot

En esta guía, se explica cómo agregar la funcionalidad para **buscar un producto por su ID** en una API desarrollada con **Spring Boot**. Se realizaron modificaciones en el **controlador**, se agregó una **clase de excepción**, y al final se prueba el funcionamiento con **Postman**.

Paso 1: Modificación de `ProductoControlador.java` para agregar el método `obtenerProductoPorId()`

Código actualizado de `ProductoControlador.java`

```
package gm.inventarios.controlador;

import gm.inventarios.expcion.RecursoNoEncontradoExcepcion;
import gm.inventarios.modelo.Producto;
import gm.inventarios.servicio.ProductoServicio;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
```

```
import java.util.List;

@RestController
//http://localhost:8080/inventario-app
@RequestMapping("inventario-app")
@CrossOrigin(value = "http://localhost:4200")
public class ProductoControlador {

    private static final Logger logger =
        LoggerFactory.getLogger(ProductoControlador.class);

    @Autowired
    private ProductoServicio productoServicio;

    //http://localhost:8080/inventario-app/productos
    @GetMapping("/productos")
    public List<Producto> obtenerProductos(){
        List<Producto> productos = this.productoServicio.listarProductos();
        logger.info("Productos obtenidos:");
        productos.forEach((producto -> logger.info(producto.toString())));
        return productos;
    }

    @PostMapping("/productos")
    public Producto agregarProducto(@RequestBody Producto producto){
        logger.info("Producto a agregar: " + producto);
        return this.productoServicio.guardarProducto(producto);
    }

    @GetMapping("/productos/{id}")
    public ResponseEntity<Producto> obtenerProductoPorId(
        @PathVariable int id){
        Producto producto =
            this.productoServicio.buscarProductoPorId(id);
        if(producto != null)
            return ResponseEntity.ok(producto);
        else
            throw new RecursoNoEncontradoExpcion("No se encontro el id: " + id);
    }
}
```

Explicación del código y cambios

1. Se creó el método `obtenerProductoPorId(int id)`:

- La anotación `@GetMapping("/productos/{id}")` define que este método manejará solicitudes **GET** para obtener un producto específico por su ID.
- Se usa `@PathVariable int id` para obtener el valor del parámetro `{id}` de la URL.
- Se llama al método `buscarProductoPorId(id)` del servicio para obtener el producto.
- **Si el producto existe**, se devuelve con `ResponseEntity.ok(producto)`, lo que responde con un código **200 OK**.

- Si el producto no existe, se lanza una excepción RecursoNoEncontradoExcepcion, lo que devuelve un error 404 Not Found.
2. Uso de `ResponseEntity<Producto>`:
- `ResponseEntity` permite manejar mejor las respuestas HTTP, asegurando que se devuelvan códigos de estado correctos (200 si existe, 404 si no).
-

Paso 2: Creación de la clase de excepción

`RecursoNoEncontradoExcepcion.java`

Código de `RecursoNoEncontradoExcepcion.java`

```
package gm.inventarios.excepcion;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class RecursoNoEncontradoExcepcion extends RuntimeException{
    public RecursoNoEncontradoExcepcion(String mensaje){
        super(mensaje);
    }
}
```

Explicación del código

1. Extiende `RuntimeException`:
 - Permite lanzar esta excepción cuando un producto no se encuentra en la base de datos.
 2. Anotación `@ResponseStatus(HttpStatus.NOT_FOUND)`:
 - Indica que cuando esta excepción se lanza, el servidor debe responder con un código HTTP 404 (Not Found).
 3. Constructor personalizado:
 - Permite enviar un mensaje con detalles sobre el error, como "No se encontró el ID: 10".
-

Paso 3: Volver a ejecutar la aplicación

Para aplicar los cambios, se debe **reiniciar** la aplicación Spring Boot.

Pasos:

1. En IntelliJ IDEA:
 - Haz clic derecho sobre la clase `InventariosApplication.java`.

- Selecciona "**Run InventariosApplication**".
2. Espera a que la aplicación se ejecute correctamente y vuelva a estar disponible en <http://localhost:8080/>
-

Paso 4: Pruebas con Postman

Ahora podemos probar el nuevo endpoint en **Postman** para buscar un producto por ID.

Prueba 1: Buscar un producto existente

1. Abre **Postman**.
2. Selecciona **GET**.
3. En la barra de dirección, ingresa:

<http://localhost:8080/inventario-app/productos/1>

4. Presiona **Send**.

Respuesta esperada (200 OK):

```
{  
    "idProducto": 1,  
    "descripcion": "Pantalón Rojo M",  
    "precio": 500.0,  
    "existencia": 100  
}
```

Prueba 2: Buscar un producto que no existe

1. En **Postman**, cambia la URL a:

<http://localhost:8080/inventario-app/productos/10>

2. Presiona **Send**.

Respuesta esperada (404 Not Found):

```
{  
    "timestamp": "02:01:40.276+00:00",  
    "status": 404,  
    "error": "Not Found",  
    "path": "/inventario-app/productos/10"  
}
```

Conclusión

- Se agregó el método `obtenerProductoPorId()` en `ProductoControlador` para buscar un producto por su ID.
- Se utilizó `ResponseEntity<Producto>` para devolver un código HTTP correcto.
- Se creó la excepción `RecursoNoEncontradoExcepcion.java` para manejar errores cuando el producto no existe.
- Se probó la funcionalidad en Postman, verificando respuestas 200 OK y 404 Not Found.

Este cambio permite que el sistema de inventarios **maneje correctamente las búsquedas por ID**, mejorando la experiencia del usuario y el manejo de errores en la API. 

Saludos!

Ing. Ubaldo Acosta

Fundador de [GlobalMentoring.com.mx](https://www.globalmentoring.com.mx)