

# Actualizar un Producto con Spring



## Modificación de `ProductoControlador.java` para actualizar un producto en Spring Boot (Parte 4 de la edición de un producto)

En esta cuarta parte, agregaremos el método `actualizarProducto()` en **Spring Boot**, permitiendo modificar un producto existente mediante una petición **PUT**. También explicaremos **por qué usamos PUT en lugar de POST**, y cómo probar la funcionalidad en **Postman**.

---

### Paso 1: Modificar `ProductoControlador.java` para agregar `actualizarProducto()`

#### Código actualizado

```
package gm.inventarios.controlador;

import gm.inventarios.excepcion.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);
    }

    @PutMapping("/productos/{id}")
    public ResponseEntity<Producto> actualizarProducto(
        @PathVariable int id,
        @RequestBody Producto productoRecibido){
        Producto producto = this.productoServicio.buscarProductoPorId(id);
        producto.setDescripcion(productoRecibido.getDescripcion());
        producto.setPrecio(productoRecibido.getPrecio());
        producto.setExistencia(productoRecibido.getExistencia());
        this.productoServicio.guardarProducto(producto);
        return ResponseEntity.ok(producto);
    }
}
```

---

## Paso 2: Explicación del método `actualizarProducto()`

```

@PutMapping("/productos/{id}")
public ResponseEntity<Producto> actualizarProducto(
    @PathVariable int id,
    @RequestBody Producto productoRecibido){
    Producto producto = this.productoServicio.buscarProductoPorId(id);
    producto.setDescripcion(productoRecibido.getDescripcion());
    producto.setPrecio(productoRecibido.getPrecio());
    producto.setExistencia(productoRecibido.getExistencia());
    this.productoServicio.guardarProducto(producto);
    return ResponseEntity.ok(producto);
}

```

## Explicación del código

1. `@PutMapping("/productos/{id}")`
  - Define que este método **manejará solicitudes PUT** para la URL `/productos/{id}`.
  - PUT es el método estándar para **actualizar recursos existentes** en una API REST.
2. **Obtención del ID desde la URL**

```
@PathVariable int id
```

- Extrae el ID del producto desde la URL.

3. **Recibir los nuevos datos en el cuerpo de la solicitud**

```
@RequestBody Producto productoRecibido
```

- Recibe el **objeto JSON** con los nuevos valores del producto.

4. **Verificación de existencia del producto**

```

Producto producto = this.productoServicio.buscarProductoPorId(id);
if (producto == null) {
    throw new RecursoNoEncontradoExcepcion("No se encontró el ID: " + id);
}

```

- Si el producto **no existe**, se lanza una **excepción RecursoNoEncontradoExcepcion** con un mensaje de error.

5. **Actualizar los valores del producto**

```

producto.setDescripcion(productoRecibido.getDescripcion());
producto.setPrecio(productoRecibido.getPrecio());
producto.setExistencia(productoRecibido.getExistencia());

```

- Se **asignan los nuevos valores** al producto encontrado.

6. **Guardar el producto actualizado**

```
this.productoServicio.guardarProducto(producto);
```

- Se **guarda el producto actualizado** en la base de datos.

7. **Devolver la respuesta con el producto actualizado**

```
return ResponseEntity.ok(producto);
```

- Se envía una **respuesta HTTP 200 OK** con los datos del producto modificado.
- 

## Paso 3: ¿Por qué usamos `PUT` en lugar de `POST`?

### Método HTTP

### Uso

<b>POST</b>	Para <b>crear</b> un nuevo recurso en la base de datos.
<b>PUT</b>	Para <b>actualizar</b> un recurso existente, reemplazando su contenido.

- **POST crea un nuevo recurso**, mientras que **PUT actualiza un recurso ya existente**.
  - **PUT es idempotente**, lo que significa que si se envía la misma solicitud varias veces, el estado del recurso no cambia más allá de la primera actualización.
  - **PUT requiere un ID en la URL** para indicar **qué recurso se va a modificar**, mientras que **POST no lo necesita** porque crea un nuevo recurso con un ID generado automáticamente.
- 

## Paso 4: Reiniciar la aplicación Spring Boot

Para aplicar los cambios, es necesario **detener y volver a ejecutar** la aplicación **Spring Boot**.

1. **En IntelliJ IDEA:**
    - **Clic derecho** sobre la clase `InventariosApplication.java`.
    - Selecciona "**Run InventariosApplication**".
  2. Espera a que la aplicación se inicie correctamente.
- 

## Paso 5: Prueba en Postman

Para verificar que la funcionalidad de edición funciona correctamente, realizamos una **petición PUT** en Postman.

### Prueba 1: Editar un producto existente

1. **Abrir Postman.**
2. **Seleccionar PUT en el tipo de solicitud.**

3. Ingresar la URL (Al final se agrega el valor del Id Producto que se desea modificar):

http://localhost:8080/inventario-app/productos/5

4. Ir a la pestaña "Body" → seleccionar "raw" → elegir "JSON".
5. Enviar el siguiente JSON con los nuevos valores:

```
{  
    "idProducto": 4,  
    "descripcion": "Florero L Cambio",  
    "precio": 1500.0,  
    "existencia": 400  
}
```

6. Presionar "Send".

## Respuesta esperada (200 OK)

```
{  
    "idProducto": 4,  
    "descripcion": "Florero L Cambio",  
    "precio": 1500.0,  
    "existencia": 400  
}
```

- Confirma que los datos han sido **actualizados correctamente**.

The screenshot shows a web browser window with the following details:

- Address Bar:** localhost:8080/inventario-app/ | localhost:4200/productos
- Title Bar:** InventarioApp
- Header:** Sistema de Inventarios | Inicio | Agregar Producto
- Main Content:** Sistema de Inventarios
- Table:** A table listing products with columns: Id, Descripción del Producto, Precio, Existencias, and Acciones (with an 'Editar' button).
- Data in Table:**

Id	Descripción del Producto	Precio	Existencias	Acciones
1	Pantalon Rojo M	500	100	<button>Editar</button>
2	Blusa Amarilla L	300	50	<button>Editar</button>
3	Vaso Tequilero	200	30	<button>Editar</button>
4	Florero L Cambio	1500	400	<button>Editar</button>

## Conclusión

- Se agregó el método `actualizarProducto()` en `ProductoControlador.java` para modificar productos existentes.
- Se usó `PUT` en lugar de `POST`, ya que `PUT` es el método correcto para actualizar recursos.
- Se reinició la aplicación `Spring Boot` para aplicar los cambios.
- Se probó en `Postman` confirmando que los productos se pueden modificar correctamente.

Con esto, el backend ya permite **editar productos**, y en la siguiente parte implementaremos la funcionalidad en **Angular**. 

Saludos!

Ing. Ubaldo Acosta

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