Modelos de programación II

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS
Acreditación Institucional de Alta Calidad

Simar Enrique Herrera Jimenez

Santiago pineda Anaya - (20222020055)

Andersson Duvan Marín Zarta - (20212020059)

Ingeniería de Sistemas

Procedimental (Anotaciones) taller 1

Bogotá D.C

4/04/2025

Introducción

En el desarrollo de aplicaciones modernas, los servicios web RESTful son fundamentales para permitir la interacción entre diferentes sistemas a través de la web. REST (Representational State Transfer) es un estilo de arquitectura que utiliza los métodos HTTP para realizar operaciones sobre los recursos (entidades) de una aplicación, como la creación, lectura, actualización y eliminación de datos.

En este contexto, los **endpoints REST** son puntos de acceso dentro de una API donde se pueden realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar). Los métodos más comunes de HTTP utilizados para estos propósitos son GET, POST, PUT y DELETE, y cada uno tiene un propósito específico dentro de un servicio RESTful.

A continuación, se describen los 5 endpoints de una API para la gestión de productos, implementados utilizando Spring Boot. Estos endpoints permiten a los usuarios interactuar con los productos de un sistema en este caso una lista de productos mediante solicitudes HTTP.

Explicación de los Endpoints

GET /api/productos/

Propósito: Este endpoint se utiliza para **obtener todos los productos** almacenados en el sistema. Permite a los clientes consultar todos los productos disponibles.

Método HTTP: GET

Descripción: Cuando el cliente realiza una solicitud GET a este endpoint, la respuesta contiene una lista de todos los productos en formato JSON. Esta operación es ideal para obtener una visión general de todos los recursos (productos) del sistema.

```
@GetMapping("/")
public List<Producto> obtenerTodos() {
   return productoService.obtenerTodos();
}
```

GET /api/productos/{id}

Propósito: Este endpoint permite **obtener un producto específico** identificado por su id.

Método HTTP: GET

Descripción: Al hacer una solicitud GET con el ID del producto, el servidor responde con los detalles del producto correspondiente o con un error 404 si no se encuentra el producto.

```
@GetMapping("/{id}")
public ResponseEntity<Producto> obtenerPorId(@PathVariable Long id) {
   Optional<Producto> producto = productoService.obtenerPorId(id);
   if (producto.isPresent()) {
      return ResponseEntity.ok(producto.get());
   } else {
      return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
   }
}
```

```java
@GetMapping("/")
public List<Producto> obtenerTodos() {
    return productoService.obtenerTodos();
}

// Obtener un producto por ID
@GetMapping("/{id}")
public ResponseEntity<Producto> obtenerPorId(@PathVariable Long id) {
    Optional<Producto> producto = productoService.obtenerPorId(id);
    if (producto.isPresent()) {
        return ResponseEntity.ok(producto.get());  // Si existe, devolver con 200 OK
    } else {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).build();  // Si no existe, devolver 404 Not Found
    }
}
```

POST /api/productos/

Propósito: Este endpoint permite **crear un nuevo producto** en el sistema.

Método HTTP: POST

Descripción: El cliente puede enviar los datos del nuevo producto en el cuerpo de la solicitud. El servidor procesará estos datos y creará un nuevo producto, asignándole un id único y respondiendo con los detalles del producto creado.

```
@PostMapping("/")
public ResponseEntity<Producto> crearProducto(@RequestBody Producto producto) {
   Producto productoCreado = productoService.crearProducto(producto);
   return ResponseEntity.status(HttpStatus.CREATED).body(productoCreado);
}
```

```java
// Crear un producto
@PostMapping("/")
public ResponseEntity<Producto> crearProducto(@RequestBody Producto producto) {
    Producto productoCreado = productoService.crearProducto(producto);
    return ResponseEntity.status(HttpStatus.CREATED).body(productoCreado);  // Devuelve el producto creado con 201 Created
}
```

PUT /api/productos/{id}

Propósito: Este endpoint permite **actualizar los datos de un producto existente** por su id.

Método HTTP: PUT

Descripción: Al hacer una solicitud PUT, el cliente puede proporcionar nuevos datos para el producto con el id especificado. Si el producto existe, se actualizarán sus datos; de lo contrario, se devolverá un error 404.

```
@PutMapping("/{id}")
public ResponseEntity<Producto> actualizarProducto(@PathVariable Long id,
@RequestBody Producto producto) {
   Producto productoActualizado = productoService.actualizarProducto(id, producto);
   if (productoActualizado != null) {
      return ResponseEntity.ok(productoActualizado);
   } else {
      return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
   }
}
```

```java
// Actualizar un producto
@PutMapping("/{id}")
public ResponseEntity<Producto> actualizarProducto(@PathVariable Long id, @RequestBody Producto producto) {
    Producto productoActualizado = productoService.actualizarProducto(id, producto);
    if (productoActualizado != null) {
        return ResponseEntity.ok(productoActualizado);  // Devuelve el producto actualizado con 200 OK
    } else {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).build();  // Si no se encuentra, devuelve 404 Not Found
    }
}
```

DELETE /api/productos/{id}

Propósito: Este endpoint se utiliza para **eliminar un producto** por su id.

Método HTTP: DELETE

Descripción: El cliente puede hacer una solicitud DELETE pasando el id del producto que desea eliminar. Si el producto existe, se eliminará de la base de datos; de lo contrario, se devolverá un error 404.

```
@DeleteMapping("/{id}")
public ResponseEntity<Void> eliminarProducto(@PathVariable Long id) {
   boolean eliminado = productoService.eliminarProducto(id);
   if (eliminado) {
      return ResponseEntity.status(HttpStatus.NO_CONTENT).build();
   } else {
      return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
```

```
    }
}
```

```java
// Eliminar un producto
@DeleteMapping("/{id}")
public ResponseEntity<Void> eliminarProducto(@PathVariable Long id) {
    boolean eliminado = productoService.eliminarProducto(id);
    if (eliminado) {
        return ResponseEntity.status(HttpStatus.NO_CONTENT).build();  // Devuelve 204 No Content
    } else {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).build();  // Si no se encuentra, devuelve 404 Not Found
    }
}
```

Los servicios web RESTful, al estar basados en los métodos HTTP estándar (GET, POST, PUT, DELETE), ofrecen una manera eficiente y clara para que los sistemas interactúen entre sí a través de la web. Estos servicios son esenciales en el desarrollo de aplicaciones modernas, donde la comunicación entre diferentes partes de un sistema es vital para su funcionamiento.

En este caso, hemos analizado la implementación de 5 endpoints utilizando Spring Boot para la gestión de productos. Cada uno de estos endpoints cumple una función específica dentro de las operaciones CRUD, permitiendo realizar tareas esenciales como obtener todos los productos, crear nuevos productos, actualizar productos existentes, y eliminar productos. Además, hemos visto cómo cada uno de estos métodos HTTP está cuidadosamente mapeado a sus respectivas funciones en la aplicación, proporcionando respuestas adecuadas según el resultado de la operación, como el retorno de un producto, la confirmación de creación, la actualización o la eliminación de un producto, o el manejo adecuado de errores como el "404 Not Found" cuando un producto no existe.

La estructura propuesta refleja una forma estándar y eficiente de implementar servicios RESTful, lo que facilita tanto el mantenimiento del código como la escalabilidad del sistema. Al seguir este patrón, los desarrolladores pueden crear aplicaciones que permitan la interacción con una base de datos de forma sencilla y efectiva, mientras cumplen con las mejores prácticas de desarrollo de APIs RESTful.