«Talento Tech»

Back-End

# Java

Clase 13

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

# Clase 13: Operaciones CRUD con Base de Datos

## Índice



---

Buenos Aires aprende
Agencia de Habilidades para el Futuro

## Objetivos de la Clase

- Crear interfaces `Repository` para acceder a la base de datos usando Spring Data JPA.
- Implementar operaciones CRUD (Create, Read, Update, Delete) directamente en la base de datos.
- Inyectar repositorios en servicios y servicios en controladores para mantener un diseño modular.
- Refactorizar el código que antes usaba una lista en memoria, sustituyéndolo por consultas y operaciones a la base de datos.
- Probar las operaciones con Postman o cURL, verificando la persistencia real.

---

## Bienvenida a TechLab

Talento Lab

¡Les damos nuevamente la bienvenida a **TechLab**! En la clase anterior configuramos la base de datos, mapeamos la entidad `Producto` y logramos que Spring Boot se conecte a MySQL. Sin embargo, las operaciones CRUD todavía no las hacíamos contra la base. Hoy, daremos el paso definitivo: implementaremos las operaciones CRUD reales utilizando Spring Data JPA, repositorios y persistencia en MySQL.

Esto significa que cuando crees, leas, actualices o elimines un producto a través de la API, el cambio se reflejará realmente en la base de datos. De esta manera, la información persistirá más allá del ciclo de vida de la aplicación, acercándonos a una aplicación lista para producción.

---

Buenos Aires aprende
Agencia de Habilidades para el Futuro

## Situación Inicial en TechLab

Talento Lab

Hasta ahora, la API respondía con datos en memoria. Silvia quiere que los productos que agreguemos permanezcan, incluso si reiniciamos el servidor. Matías y Sabrina ya tienen una base de datos MySQL corriendo, y la entidad `Producto` mapeada. Solo falta crear el `ProductoRepository` y cambiar el servicio para usarlo.

Al finalizar esta clase, cualquier operación sobre productos se reflejará directamente en la base de datos, lo que es un paso fundamental en el camino al e-commerce productivo que TechLab desea.

---

## Creación de Interfaces Repository

**Spring Data JPA** permite definir interfaces que extienden `JpaRepository<T, ID>` para acceder a datos sin escribir sentencias SQL manuales. Por ejemplo:

```
@Repository
public interface ProductoRepository extends JpaRepository<Producto,
Integer> {
   // Podés agregar métodos personalizados aquí si son necesarios
}
```

`ProductoRepository` hereda métodos como `findAll()`, `findById(ID)`, `save(Entity)`, `deleteById(ID)`, etc.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

### @Repository y JpaRepository

- `@Repository` indica que esta interfaz es un componente Spring que maneja la capa de acceso a datos.
- `JpaRepository<Producto, Integer>` define el tipo de entidad (`Producto`) y el tipo de su llave primaria (`Integer`).

---

## Implementación de Operaciones CRUD utilizando JPA Repository

Ahora, en lugar de mantener una lista en memoria, `ProductoService` usará `ProductoRepository`:

```
@Service
public class ProductoService {
    private final ProductoRepository repo;
    @Autowired
    public ProductoService(ProductoRepository repo) {
        this.repo = repo;
    }
    public List<Producto> listarTodos() {
        return repo.findAll();
    }
    public Producto obtenerPorId(int id) {
        return repo.findById(id).orElse(null);
    }
    public Producto guardar(Producto p) {
        return repo.save(p);
    }
```

```
public Producto actualizar(int id, Producto datos) {
        Producto p = obtenerPorId(id);
        if (p != null) {
            p.setNombre(datos.getNombre());
            p.setPrecio(datos.getPrecio());
            p.setCantidadEnStock(datos.getCantidadEnStock());
            return repo.save(p);
        }
        return null;
    }

    public boolean eliminar(int id) {
        if (repo.existsById(id)) {
            repo.deleteById(id);
            return true;
        }
        return false;
    }
}
```

Ahora, `guardar()` crea un registro en la base, `listarTodos()` hace un `SELECT * FROM producto`, etc.

---

## Inyección de Dependencias (@Autowired)

Como se ve, `ProductoService` inyecta `ProductoRepository`. El controlador inyectará `ProductoService`:

```
@RestController
@RequestMapping("/productos")
```

```java
public class ProductoController {

    private final ProductoService productoService;

    @Autowired
    public ProductoController(ProductoService productoService) {
        this.productoService = productoService;
    }

    @GetMapping
    public List<Producto> listarProductos() {
        return productoService.listarTodos();
    }

    @GetMapping("/{id}")
    public Producto obtenerProducto(@PathVariable int id) {
        return productoService.obtenerPorId(id);
    }

    @PostMapping
    public Producto crearProducto(@RequestBody Producto nuevo) {
        return productoService.guardar(nuevo);
    }

    @PutMapping("/{id}")
    public Producto actualizarProducto(@PathVariable int id,
@RequestBody Producto datos) {
        return productoService.actualizar(id, datos);
    }

    @DeleteMapping("/{id}")
    public void eliminarProducto(@PathVariable int id) {
        productoService.eliminar(id);
    }
}
```

## Refactorización de Controladores y Servicios para usar la Base de Datos

Lo que antes era una lista en memoria ahora es una base real. Ya no necesitamos una lista estática. Todo pasa por el repositorio. Esto simplifica el código y lo hace más escalable.

Si reiniciamos la aplicación, los datos se mantienen en la base. Si agregamos un producto con `POST /productos`, podemos verificar en MySQL Workbench que se insertó un nuevo registro en la tabla `producto`.

---

## Pruebas de Funcionalidad con Datos Reales

- **Crear un producto:** `POST /productos` con un JSON del estilo `{"nombre":"Café", "precio":250.0, "cantidadEnStock":100}`.
- **Listar productos:** `GET /productos` debe mostrar el producto recién creado.
- **Actualizar:** `PUT /productos/1` con un JSON que cambie el precio.
- **Eliminar:** `DELETE /productos/1`.

Verificar con MySQL Workbench que la tabla `producto` refleje estos cambios.

---

Buenos Aires aprende
Agencia de Habilidades para el Futuro

## Requerimiento en TechLab

En **Sibelius**, nuestro cliente gourmet, se ha incrementado el catálogo de productos, y el equipo de marketing solicita **más información** de cada ítem (descripciones, categorías, etc.) para diseñar campañas enfocadas. Además, **Silvia** la Product Owner, ve la necesidad de **filtrar productos** según distintos criterios (nombre, categoría), para que sea más fácil gestionar el inventario y exhibir en su plataforma.

Para cumplir con estas nuevas demandas, Matías y Sabrina (en conjunto con vos) deben **extender el modelo** y **afinar el acceso a la base de datos**:

## Ejercicios prácticos

1. **Agregar más atributos a Producto:**
    a. Por ejemplo, `descripcion` o `categoria`.
    b. Verificar que se agregue la columna en la base al iniciar la aplicación.
2. **Crear Consultas Personalizadas:**
    a. Agregar un método en `ProductoRepository` como `List<Producto> findByNombre(String nombre)` y probarlo en el servicio.
3. **Probar con más datos:**
    a. Crear varios productos y listarlos para confirmar que la base de datos los almacena correctamente.

Con la ayuda de **Spring Data JPA**, se busca garantizar que estas consultas sean sencillas de implementar, y que el repositorio pueda seguir ampliándose para futuras exigencias. Así, el equipo en TechLab dará un **salto de calidad** en la administración de su catálogo, acercando a Sibelius la agilidad que tanto necesita en su plataforma de e-commerce.

A continuación, se detallan los **ejercicios prácticos** para esta clase, invitándote a incorporar más atributos a `Producto`, crear métodos personalizados en tu repositorio y confirmar que la inserción, consulta y actualización de los datos funcionen correctamente en la base de datos. ¡Manos a la obra!

Buenos Aires aprende
Agencia de Habilidades para el Futuro

---

## Materiales y Recursos Adicionales

- Documentación Oficial de Spring Data JPA:
- Tutoriales en YouTube sobre integración de Spring Boot, JPA y MySQL.

---

## Preguntas para Reflexionar

- ¿De qué forma el uso de repositorios JPA simplifica las operaciones CRUD en comparación con escribir SQL manualmente?
- ¿Cómo influye la persistencia en la base de datos en la escalabilidad y confiabilidad del sistema de e-commerce?

---


Buenos Aires Ciudad