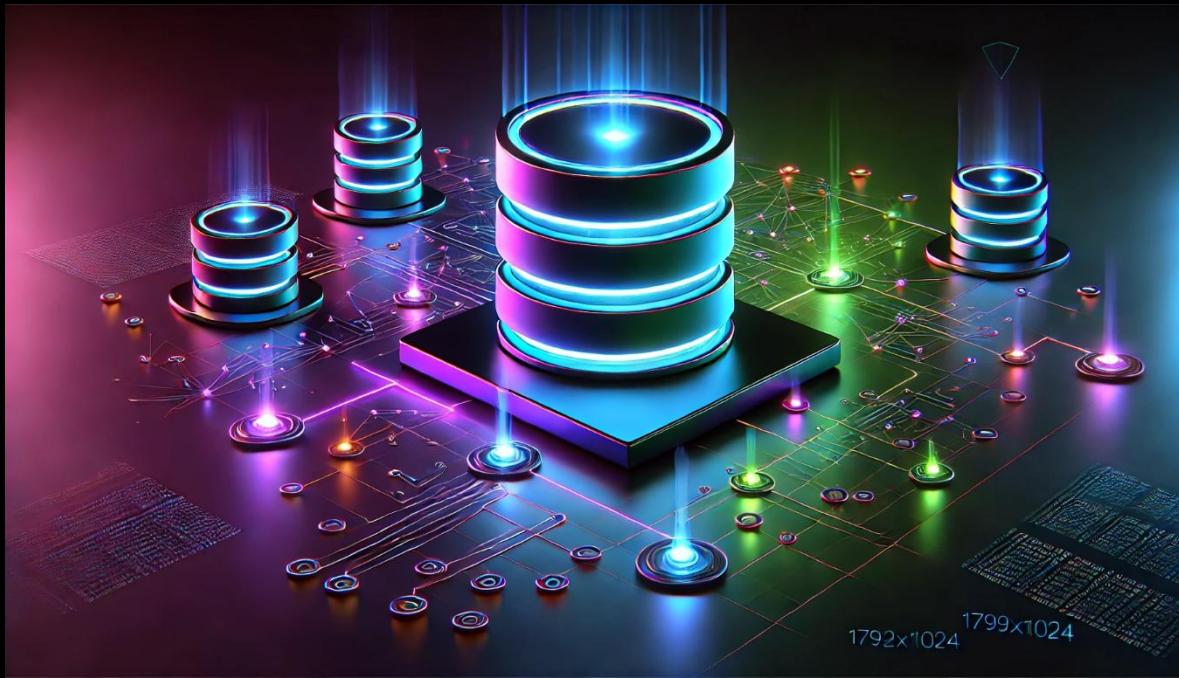


# Servicio y Repositorio con Spring Boot



## Creación de Repositorio y Servicio con Spring Boot

En una aplicación con **Spring Boot**, una buena práctica es estructurar el código en **capas**:

1. **Capa de Datos (Repositorio)** → Comunicación con la base de datos.
2. **Capa de Servicio** → Lógica de negocio.
  - **Interfaz** → Define métodos que la implementación debe cumplir.
  - **Clase de servicio** → Implementa la lógica de negocio.

---

### ◆ Paso 1: Crear la Capa de Datos (Repositorio)

❖ **Objetivo:** Definir una interfaz que maneje la persistencia de datos mediante **JPA**.

❖ **Archivo:** ProductoRepository.java

```
package gm.inventarios.repositorio;  
  
import gm.inventarios.modelo.Producto;  
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface ProductoRepository extends JpaRepository<Producto, Integer> { }
```

**Explicación:**

- `ProductoRepository` es una **interfaz** que extiende `JpaRepository<Producto, Integer>`.
  - `JpaRepository` proporciona métodos CRUD sin necesidad de implementarlos (`findAll()`, `save()`, `deleteById()`, etc.).
  - `Producto` es la entidad que se mapea a la base de datos.
  - `Integer` es el tipo de dato de la **clave primaria** (`idProducto`).
- 

## ◆ Paso 2: Crear la Interfaz de Servicio

❖ **Objetivo:** Definir métodos que la lógica de negocio debe implementar.

❖ **Archivo:** `IProductoServicio.java`

```
package gm.inventarios.servicio;

import gm.inventarios.modelo.Producto;
import java.util.List;

public interface IProductoServicio {
    List<Producto> listarProductos();

    Producto buscarProductoPorId(Integer idProducto);

    void guardarProducto(Producto producto);

    void eliminarProductoPorId(Integer idProducto);
}
```

**Explicación:**

- Define los métodos que la **clase de servicio** implementará.
  - Se usa una **interfaz** porque permite mayor flexibilidad y facilita la implementación de múltiples servicios en el futuro.
- 

## ◆ Paso 3: Crear la Implementación de la Capa de Servicio

❖ **Objetivo:** Implementar la lógica de negocio, utilizando el repositorio para acceder a la base de datos.

❖ **Archivo:** ProductoServicio.java

```
package gm.inventarios.servicio;

import gm.inventarios.modelo.Producto;
import gm.inventarios.repositorio.ProductoRepositorio;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class ProductoServicio implements IProductoServicio{

    @Autowired
    private ProductoRepositorio productoRepositorio;

    @Override
    public List<Producto> listarProductos() {
        return this.productoRepositorio.findAll();
    }

    @Override
    public Producto buscarProductoPorId(Integer idProducto) {
        Producto producto =
            this.productoRepositorio.findById(idProducto).orElse(null);
        return producto;
    }

    @Override
    public void guardarProducto(Producto producto) {
        this.productoRepositorio.save(producto);
    }

    @Override
    public void eliminarProductoPorId(Integer idProducto) {
        this.productoRepositorio.deleteById(idProducto);
    }
}
```

### ✓ Explicación:

- Se usa `@Service` para marcar esta clase como un **componente de servicio** en Spring.
- `@Autowired` inyecta el `ProductoRepositorio`, permitiendo acceder a la base de datos sin necesidad de instanciarlo manualmente.
- Implementa `IProductoServicio`, garantizando que todos los métodos definidos en la interfaz están presentes en la clase.
- Usa métodos de `productoRepositorio` como `findAll()`, `save()`, `findById()`, y `deleteById()` para interactuar con la base de datos.

## Conclusión

Este diseño sigue la **arquitectura en capas**, que ofrece varias ventajas:

- Separación de responsabilidades:** La capa de datos se encarga de la persistencia, mientras que la capa de servicio maneja la lógica de negocio.
- Facilidad de mantenimiento:** Puedes modificar la lógica de negocio sin afectar la comunicación con la base de datos.
- Reutilización de código:** Puedes cambiar la implementación de `IProductoServicio` sin afectar otras partes del código.

Con este enfoque, tu aplicación Spring Boot será escalable, limpia y fácil de extender. 

Saludos!

**Ing. Ubaldo Acosta**

Fundador de [GlobalMentoring.com.mx](#)