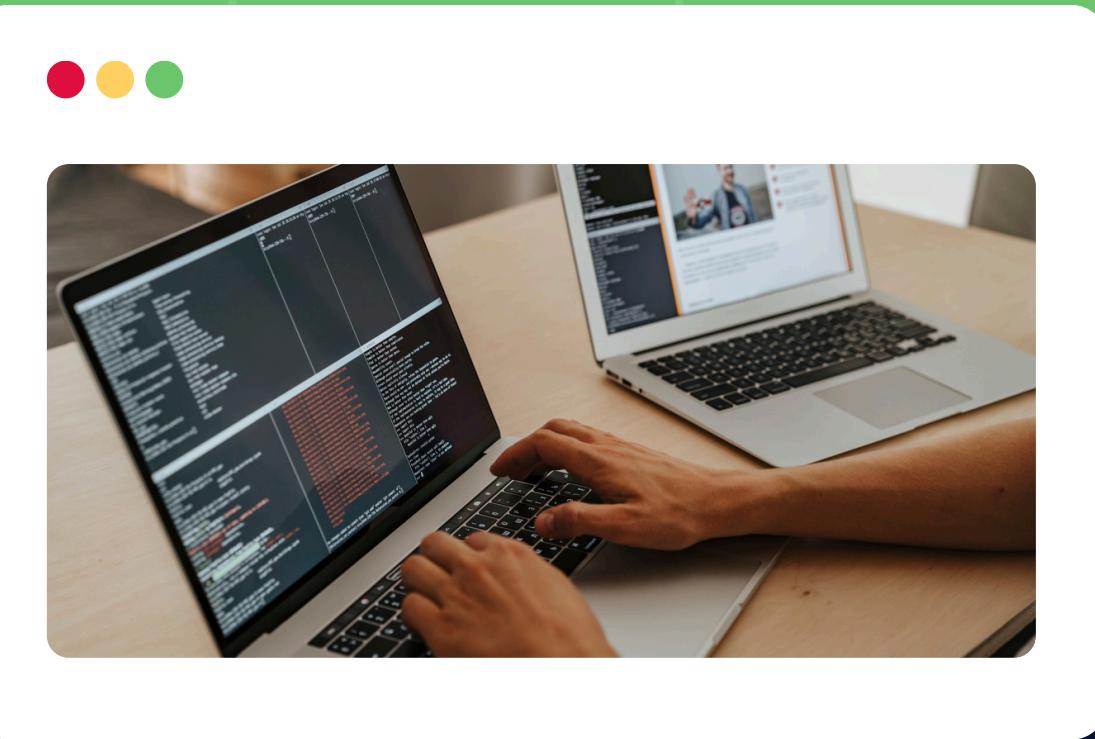


Data Access Object (DAO)

A screenshot of a code editor window with a dark theme. The code is written in React.js and shows a component structure. It includes imports for React, useState, and Fragment. The component uses state management and functional programming concepts like map and console.log.

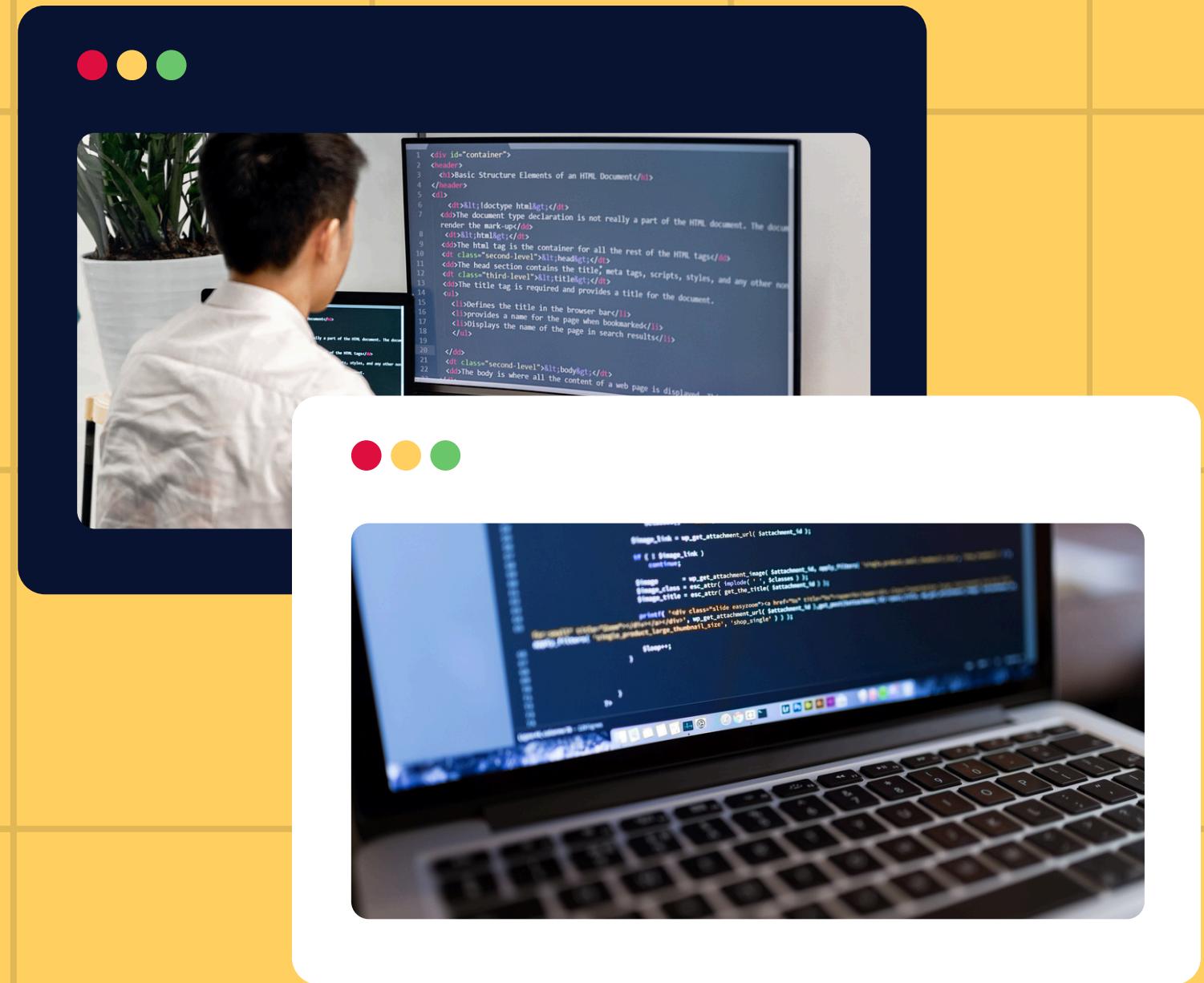
Ingeniería en Sistemas
Computacionales

**Oswaldo Colli
Rosales**

Introducción

Imagina que tienes una aplicación que necesita acceder a una base de datos para guardar y recuperar información de usuarios.

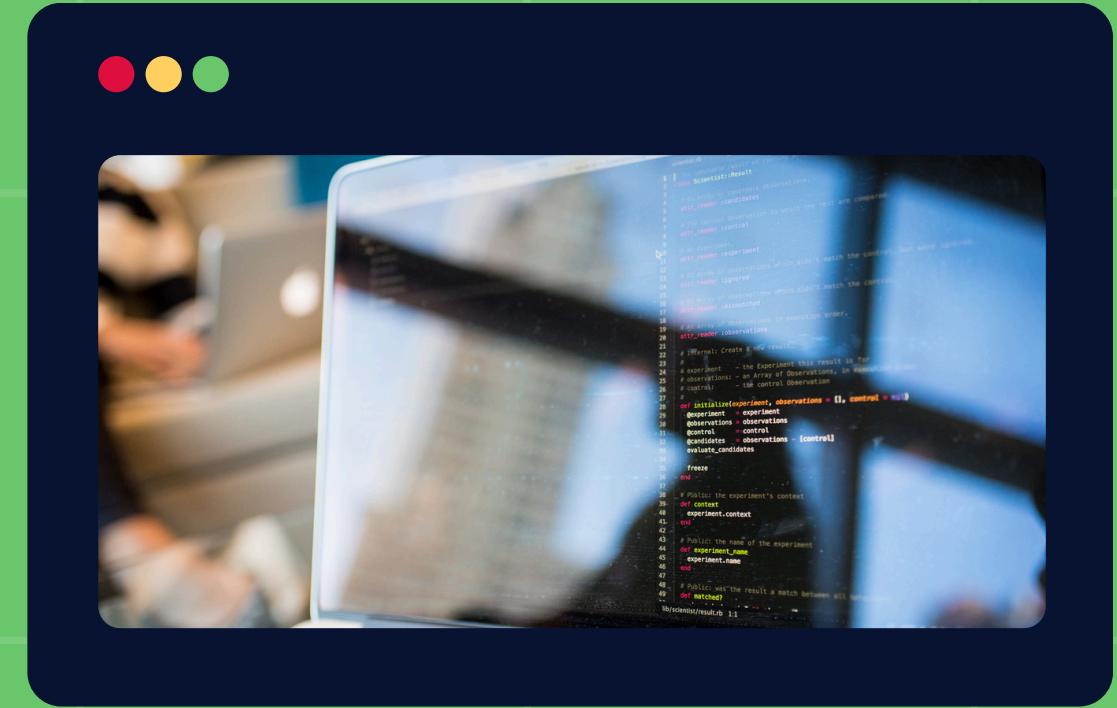
Sin el patrón DAO, el código de acceso a la base de datos estaría mezclado con la lógica de la aplicación, lo que haría que todo sea más difícil de modificar y mantener.



El patrón DAO (Data Access Object) actúa como un intermediario entre la aplicación y la base de datos. Su función es manejar la conexión, consultas y modificaciones de datos sin que la aplicación tenga que preocuparse por estos detalles.

¿Por qué usar DAO?

1. Separa la lógica de negocio del acceso a datos.
2. Hace que el código sea más organizado y fácil de mantener.
3. Permite cambiar la base de datos sin modificar la aplicación.

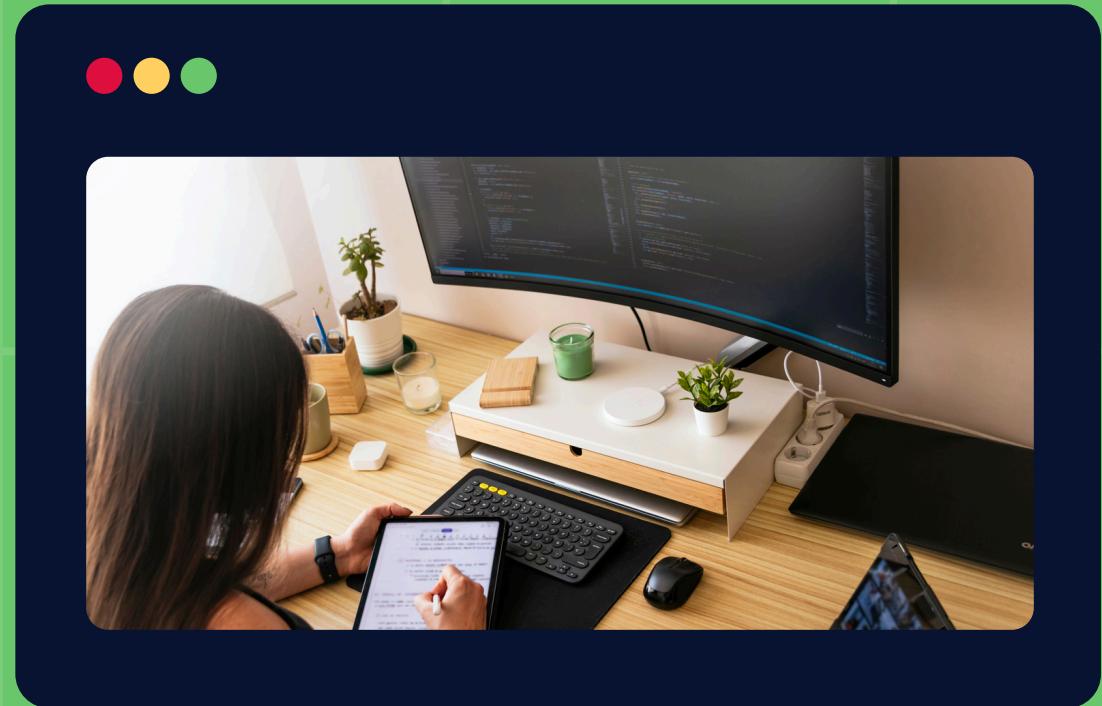


¿Qué es DAO?

El patrón de diseño Data Access Object (DAO) es un modelo estructural que proporciona una abstracción sobre la capa de acceso a datos. Su principal objetivo es separar la lógica de negocio de la gestión de la persistencia, facilitando la reutilización del código y mejorando el mantenimiento del sistema.



- Permite que la lógica de negocio no se vea afectada por los cambios en la capa de acceso a datos.
- Proporciona una interfaz para realizar operaciones CRUD (Create, Read, Update, Delete) sin depender de la implementación específica de la base de datos.
- Facilita la migración a diferentes proveedores de bases de datos y permite reutilizar el código DAO en diferentes partes de la aplicación.
- La separación de la capa de persistencia permite realizar pruebas unitarias sin necesidad de interactuar con la base de datos real.

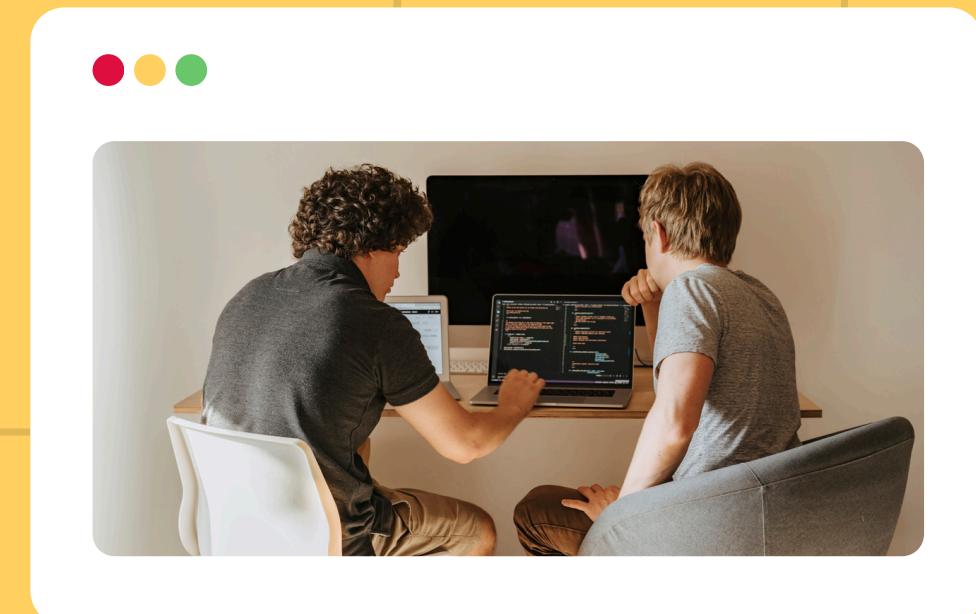
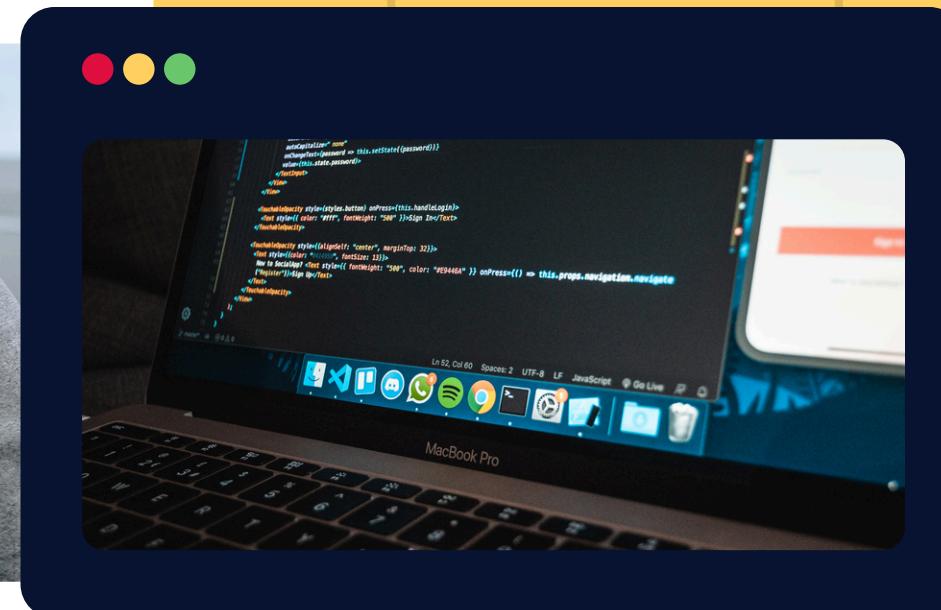
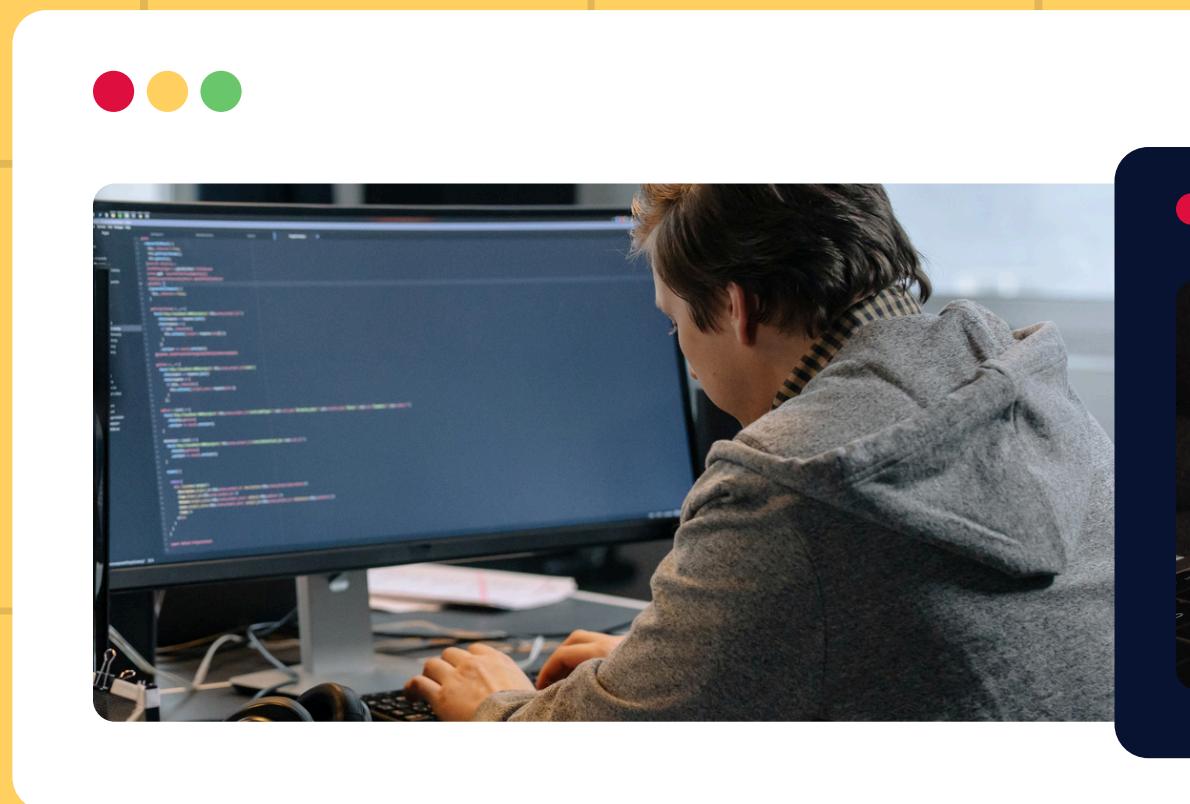


Estructura del patrón DAO



El patrón DAO está compuesto por los siguientes elementos:

- Entidad (Model): Representa la estructura de los datos.
- Interfaz DAO: Define los métodos CRUD.
- Implementación DAO: Contiene la lógica de acceso a la base de datos.
- Cliente o Servicio: Utiliza el DAO para manipular los datos sin preocuparse por los detalles de persistencia.



Ejemplo de implementación

Ejemplo

Este ejemplo maneja una lista de productos usando el patrón DAO.

1. Crear la Clase Producto (Entidad)

```
public class Producto {  
    private int id;  
    private String nombre;  
    private double precio;  
  
    public Producto(int id, String nombre, double precio) {  
        this.id = id;  
        this.nombre = nombre;  
        this.precio = precio;  
    }  
  
    public int getId() { return id; }  
    public void setId(int id) { this.id = id; }  
  
    public String getNombre() { return nombre; }  
    public void setNombre(String nombre) { this.nombre = nombre; }  
  
    public double getPrecio() { return precio; }  
    public void setPrecio(double precio) { this.precio = precio; }  
}
```

Ejemplo



2. Definir la Interfaz ProductoDAO

```
import java.util.List;

public interface ProductoDAO {
    void agregarProducto(Producto producto);
    Producto obtenerProducto(int id);
    List<Producto>
obtenerTodosLosProductos();
    void actualizarProducto(Producto
producto);
    void eliminarProducto(int id);
}
```



3. Implementar ProductoDAO (Base de Datos Simulada)

```
import java.util.ArrayList;
import java.util.List;

public class ProductoDAOImpl implements ProductoDAO {
    private List<Producto> productos = new ArrayList<>();

    @Override
    public void agregarProducto(Producto producto) {
        productos.add(producto);
    }

    @Override
    public Producto obtenerProducto(int id) {
        return productos.stream().filter(p -> p.getId() == id).findFirst().orElse(null);
    }

    @Override
    public List<Producto> obtenerTodosLosProductos() {
        return productos;
    }

    @Override
    public void actualizarProducto(Producto producto) {
        Producto p = obtenerProducto(producto.getId());
        if (p != null) {
            p.setNombre(producto.getNombre());
            p.setPrecio(producto.getPrecio());
        }
    }

    @Override
    public void eliminarProducto(int id) {
        productos.removeIf(p -> p.getId() == id);
    }
}
```



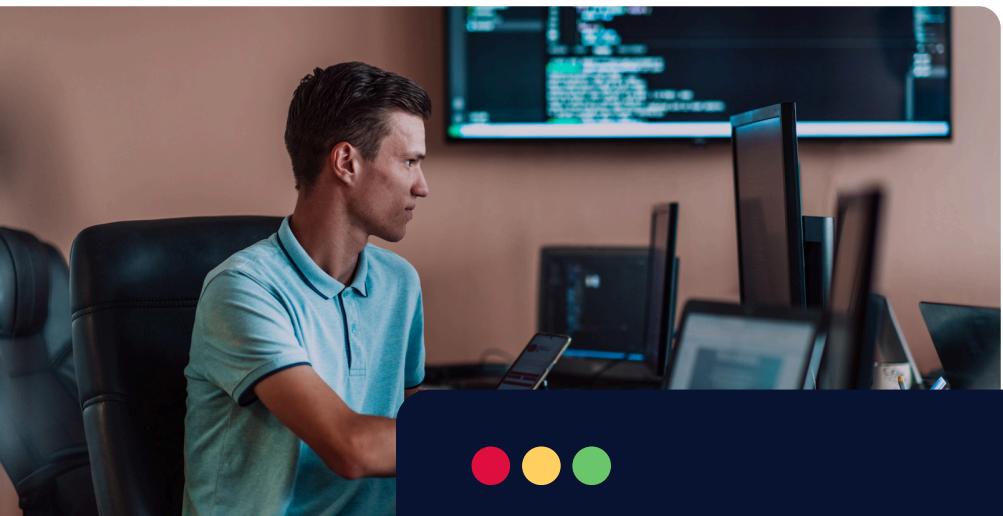
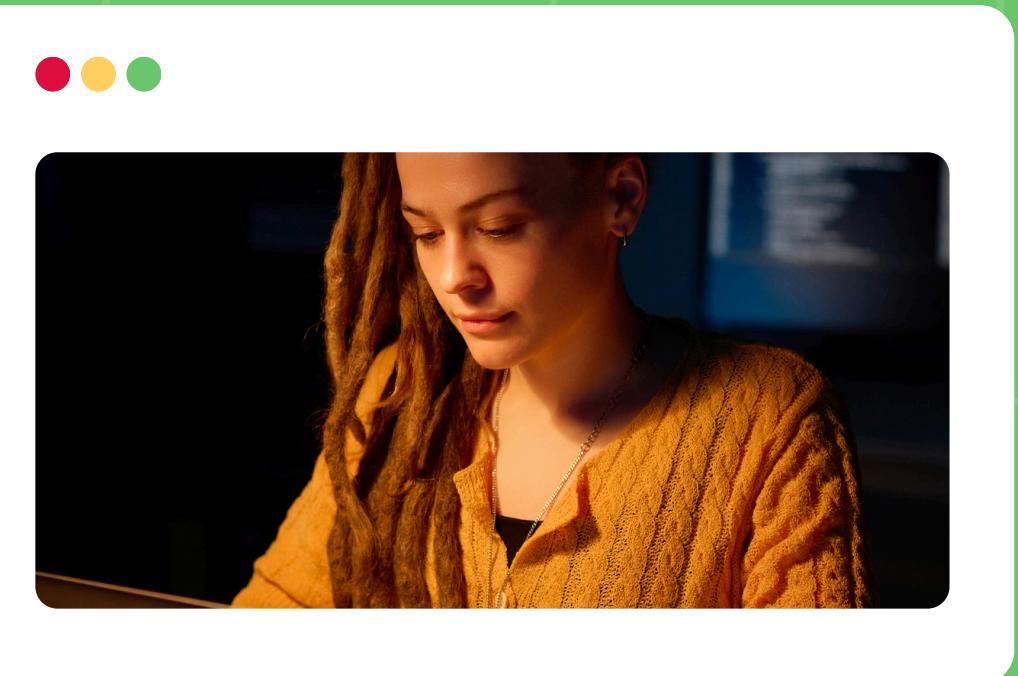
4. Uso del DAO en la Aplicación

```
public class Main {  
    public static void main(String[] args) {  
        ProductoDAO productoDAO = new ProductoDAOImpl();  
  
        // Agregar productos  
        productoDAO.agregarProducto(new Producto(1, "Laptop", 1200.50));  
        productoDAO.agregarProducto(new Producto(2, "Mouse", 25.99));  
  
        // Obtener y mostrar todos los productos  
        System.out.println("Lista de productos:");  
        for (Producto p : productoDAO.obtenerTodosLosProductos()) {  
            System.out.println(p.getNombre() + " - $" + p.getPrecio());  
        }  
  
        // Actualizar un producto  
        Producto producto = productoDAO.obtenerProducto(1);  
        if (producto != null) {  
            producto.setPrecio(1100.00);  
            productoDAO.actualizarProducto(producto);  
        }  
  
        // Eliminar un producto  
        productoDAO.eliminarProducto(2);  
  
        // Mostrar la lista actualizada  
        System.out.println("Productos después de la actualización y  
eliminación:");  
        for (Producto p : productoDAO.obtenerTodosLosProductos()) {  
            System.out.println(p.getNombre() + " - $" + p.getPrecio());  
        }  
    }  
}
```

Conclusión



El patrón DAO hace que la gestión de productos sea más estructurada y fácil de manejar. La lógica de negocio no necesita preocuparse por cómo se almacenan los datos, solo interactúa con el DAO para obtener, actualizar y eliminar productos.





Gracias

