**PATRÓN DE DISEÑO: DAO (DATA ACCESS OBJECT)**

**1. ¿Qué es el patrón DAO?**

El patrón DAO (Data Access Object) es un patrón de diseño estructural que separa la lógica de acceso a datos de la lógica de negocio. Su propósito principal es aislar la aplicación de los detalles de persistencia de datos, proporcionando una API limpia y reutilizable para interactuar con la base de datos.

**Propósitos del patrón DAO:**

* Encapsular el código que interactúa con la base de datos
* Facilitar cambios en el motor de persistencia sin afectar la lógica del sistema
* Promover la reutilización y pruebas de los componentes de persistencia
* Centralizar el manejo de transacciones y conexiones
* Proporcionar una interfaz uniforme para operaciones CRUD

**2. Implementación en el proyecto Bogotravel**

En el proyecto Bogotravel, el patrón DAO fue implementado para todas las entidades persistentes del sistema.

**Estructura del paquete DAO:**  
bogotravel/  
└── dao/  
├── UsuarioDAO.java  
├── EntradaDAO.java  
├── LugarTuristicoDAO.java  
└── PorVisitarDAO.java

**Ejemplo de implementación (UsuarioDAO.java):**

public class UsuarioDAO {  
private static final String SQL\_INSERT =  
"INSERT INTO usuarios (nombre, email, password) VALUES (?, ?, ?)";

text

public boolean registrar(Usuario usuario) {

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(SQL\_INSERT)) {

stmt.setString(1, usuario.getNombre());

stmt.setString(2, usuario.getEmail());

stmt.setString(3, hashPassword(usuario.getPassword()));

return stmt.executeUpdate() > 0;

} catch (SQLException e) {

System.err.println("Error al registrar usuario: " + e.getMessage());

// Considerar lanzar una excepción personalizada

return false;

}

}

// Otros métodos CRUD (buscarPorId, actualizar, eliminar, etc.)

}

**Características de la implementación:**

* Cada DAO maneja una entidad específica del dominio
* Uso de PreparedStatement para prevenir inyecciones SQL
* Conexiones manejadas mediante try-with-resources
* Clase DBConnection centralizada para gestión de conexiones PostgreSQL
* Consultas SQL definidas como constantes

**3. Justificación del uso en el sistema**

La implementación del patrón DAO en Bogotravel se justifica por:

**Ventajas obtenidas:**

* Desacoplamiento: Separación clara entre lógica de negocio y acceso a datos
* Mantenibilidad: Facilidad para modificar la capa de persistencia sin afectar otras partes
* Reusabilidad: Operaciones CRUD centralizadas y accesibles desde cualquier componente
* Escalabilidad: Preparado para futuros cambios de motor de base de datos
* Testabilidad: Permite mockear DAOs para pruebas unitarias

**Beneficios específicos para el proyecto:**

* Manejo consistente de transacciones
* Centralización del manejo de errores SQL
* Implementación uniforme de operaciones comunes
* Seguridad mejorada contra inyecciones SQL

**4. Presencia del patrón en el código fuente**

El patrón DAO está implementado en las siguientes clases:

Clase DAO Entidad Operaciones implementadas  
UsuarioDAO Usuario CRUD completo + métodos específicos  
EntradaDAO Entrada CRUD básico + consultas especializadas  
LugarTuristicoDAO Lugar CRUD + búsquedas geolocalizadas  
PorVisitarDAO Favoritos CRUD + listados personalizados

Cada implementación sigue las mismas buenas prácticas:

* Uso de parámetros en consultas SQL
* Manejo adecuado de recursos (Connection, Statement, ResultSet)
* Tratamiento consistente de excepciones
* Métodos atómicos por operación

**5. Conclusión**

La implementación del patrón DAO en Bogotravel ha demostrado ser una solución efectiva para:

1. Organizar la capa de persistencia de manera estructurada
2. Reducir el acoplamiento entre componentes
3. Facilitar el mantenimiento y evolución del sistema
4. Mejorar la seguridad en el acceso a datos
5. Permitir pruebas más efectivas de la aplicación

Esta arquitectura sigue las mejores prácticas recomendadas para aplicaciones Java empresariales y proporciona una base sólida para futuras extensiones del sistema.