








UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA





Universidad Nacional de Colombia - sede Bogotá  
Facultad de Ingeniería  
Departamento de Ingeniería de Sistemas  
Curso: Ingeniería de Software I



## Patrón de Diseño DAO



El patrón Data Access Object (DAO) es un patrón de diseño estructural  que tiene como finalidad separar la lógica de acceso a datos de la lógica de negocio en una aplicación. Esto se logra mediante una clase o conjunto de clases que encapsulan las operaciones necesarias para interactuar con una fuente de datos , como lo es en este caso la base de datos relacional.



Este patrón permite que los objetos de negocio no se encarguen directamente de conectarse, consultar o modificar la fuente de datos, sino que deleguen esas responsabilidades a los DAOs . Esto favorece la independencia entre capas del sistema, facilita el mantenimiento , mejora la reutilización del código  y permite cambiar la fuente de datos sin afectar otras capas de la aplicación.

Implementación :



En el sistema desarrollado, el patrón DAO fue implementado mediante la clase UserDao, ubicada en el paquete org.catatunbo.spynet.dao . Esta clase encapsula la lógica de autenticación de usuarios , accediendo a la base de datos mediante consultas SQL  parametrizadas, y devuelve objetos del tipo User si las credenciales son válidas .

El método authenticate(String username, String password) es responsable de establecer la conexión, preparar y ejecutar la consulta SQL , procesar el resultado, y construir un objeto User a partir de los datos obtenidos. Todo esto se realiza dentro del DAO, liberando así a la capa de negocio de las responsabilidades de interacción directa con la base de datos .

La conexión se obtiene a través de la clase DatabaseConnection , la cual sigue el patrón Singleton  para garantizar una única instancia de conexión. Esta integración permite una gestión eficiente de recursos y evita la duplicación de código.

La capa de servicios o lógica de negocio, por su parte, simplemente invoca el método authenticate(...) del DAO, y toma decisiones según el resultado. Esto demuestra una clara separación de responsabilidades  y una arquitectura limpia .

Justificación :

El uso del patrón DAO en este proyecto responde a la necesidad de estructurar el sistema de manera modular y mantenible . Gracias a este patrón, se logra separar claramente las responsabilidades entre las distintas capas del sistema. La clase UserDao se encarga exclusivamente del acceso a datos , mientras que las clases de negocio (como el

PasswordHasher que aun no se ha implementado bien 😞) se ocupan de aplicar reglas ⚖️, tomar decisiones 🎯 y coordinar el flujo de información ↻.

Además, esta separación promueve una alta cohesión 🤝 en cada componente y un bajo acoplamiento (clave para que corra el programa cuando la embarramos con la base de datos 😬) entre ellos, lo cual es un principio fundamental del diseño orientado a objetos 🧠. Si se requiere cambiar la estructura de la base de datos , modificar la tecnología de acceso o incluso reemplazar la fuente de datos 💾 , solo será necesario ajustar la implementación del DAO, sin afectar las capas superiores 🚀.

Desde una perspectiva de mantenimiento y evolución 🔧, el patrón DAO ofrece una ventaja significativa. Toda la lógica de consulta , transformación de resultados ↻ y manejo de errores técnicos ⚠️ se encuentra centralizada en una clase concreta , por lo que cualquier cambio necesario se puede aplicar de manera localizada y controlada.

Para la siguiente entrega se espera tener más códigos con patrones DAO implementados, pues en nuestro proyecto es fundamental el uso activo de los datos 🤖.