Trabajo Integrador Final

UTN - Programación II

Joaquín Del Valle Lietti

Fecha máxima de entrega: 17/11

Link al repo

Link al video

PDF EXPLICATIVO

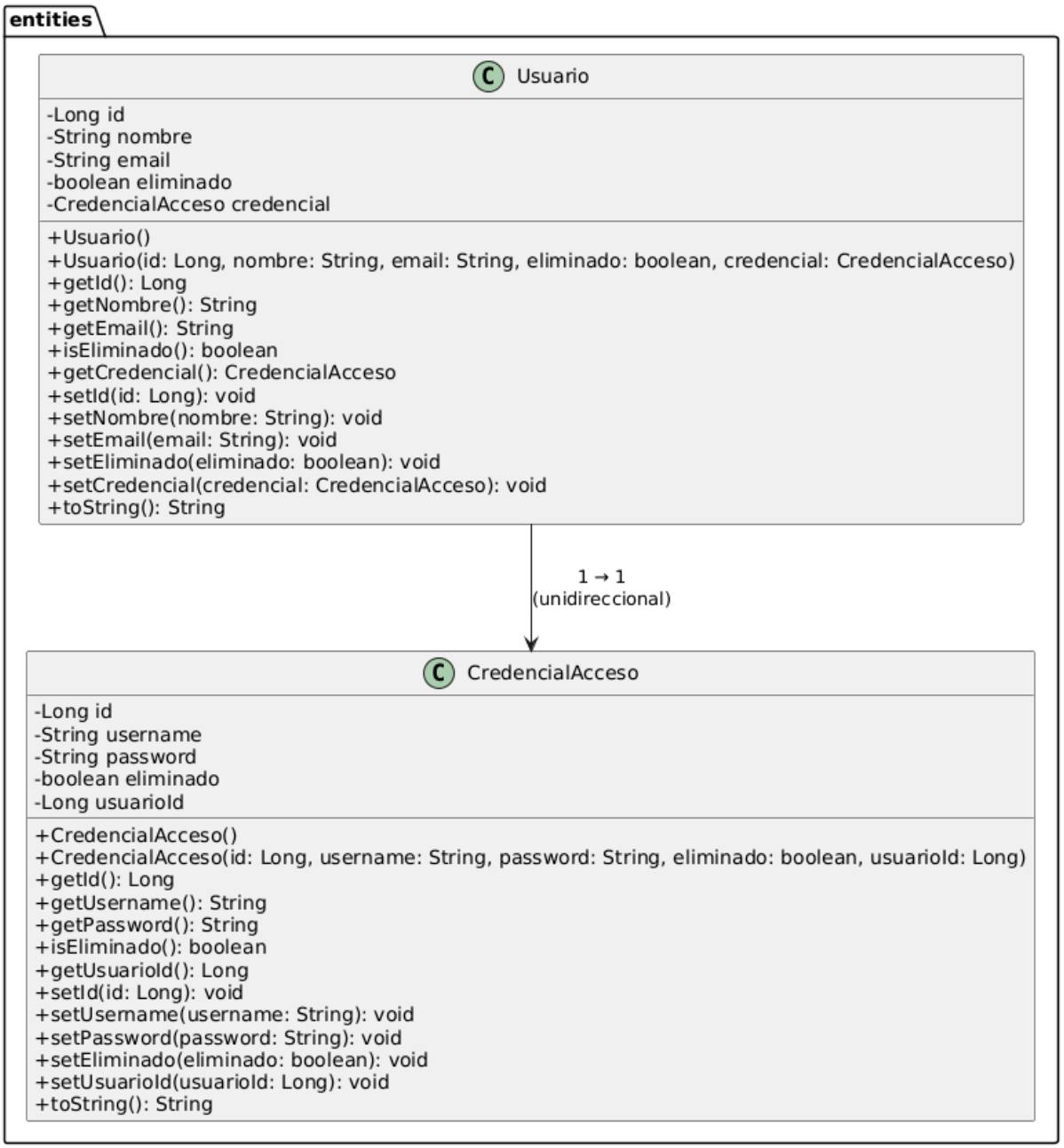
Este informe desarrolla en profundidad el sistema de gestión de usuarios y credenciales implementado en Java, siguiendo una arquitectura multicapa, con JDBC, MySQL, patrón DAO, servicios transaccionales y un menú de consola

1. Introducción

El propósito del trabajo integrador es diseñar, desarrollar y documentar una aplicación basada en arquitectura por capas, persistentemente conectada a una base de datos MySQL mediante JDBC, con manejo transaccional y operaciones CRUD completas. El objetivo es demostrar dominio de programación orientada a objetos, separación de responsabilidades, manejo adecuado de datos y consistencia referencial mediante relaciones estructuradas en la base. El dominio elegido fue la gestión de usuarios y sus credenciales. Se planteó una relación unidireccional 1→1, donde cada Usuario posee una Credencial de Acceso, pero la credencial solamente mantiene referencia al ID del usuario, respetando la unidireccionalidad. Toda la lógica de negocio se maneja dentro de la capa Service, mientras que el acceso a datos se abstrae mediante el patrón DAO.

2. Modelado UML

El modelado UML permite comprender de manera visual la estructura del dominio: entidades, atributos, métodos y relaciones. En este caso, se diseñó un diagrama UML que muestra las clases Usuario y CredencialAcceso junto con sus atributos privados, métodos públicos y la relación 1→1 unidireccional establecida entre ambas.


entities
Usuario
-Long id
-String nombre
-String email
-boolean eliminado
-CredencialAcceso credencial
+Usuario()
+Usuario(id: Long, nombre: String, email: String, eliminado: boolean, credencial: CredencialAcceso)
+getId(): Long
+getNombre(): String
+getEmail(): String
+isEliminado(): boolean
+getCredencial(): CredencialAcceso
+setId(id: Long): void
+setNombre(nombre: String): void
+setEmail(email: String): void
+setEliminado(eliminado: boolean): void
+setCredencial(credencial: CredencialAcceso): void
+toString(): String
1 → 1
(unidireccional)
CredencialAcceso
-Long id
-String username
-String password
-boolean eliminado
-Long usuarioId
+CredencialAcceso()
+CredencialAcceso(id: Long, username: String, password: String, eliminado: boolean, usuarioId: Long)
+getId(): Long
+getUsername(): String
+getPassword(): String
+isEliminado(): boolean
+getUsuarioId(): Long
+setId(id: Long): void
+setUsername(username: String): void
+setPassword(password: String): void
+setEliminado(eliminado: boolean): void
+setUsuarioId(usuarioId: Long): void
+toString(): String

3. Arquitectura en Capas

El proyecto se organizó siguiendo una arquitectura clara y mantenible, separada en capas que aíslan responsabilidades y facilitan el mantenimiento y la escalabilidad del sistema. A continuación se detalla la función de cada capa: 1. Capa config/: contiene la clase DatabaseConnection, responsable de leer las propiedades externas y generar conexiones JDBC. 2. Capa entities/: conformada por las clases Usuario y CredencialAcceso. Ambas incluyen ID, atributo de baja lógica, getters, setters y métodos toString. Representan la estructura principal del dominio. 3. Capa dao/: define interfaces genéricas y sus variantes específicas. Se encuentra aquí GenericDao, UsuarioDao y CredencialAccesoDao. 4. Capa dao/jdbc/: contiene las implementaciones concretas usando JDBC. Todas las consultas SQL usan PreparedStatement para evitar inyección SQL y mejorar rendimiento. 5. Capa service/: implementa lógica de negocio, validaciones y manejo de transacciones. Garantiza consistencia y aplica reglas como impedir que un usuario tenga más de una credencial. 6. Capa main/: implementa la interfaz de consola. El menú permite gestionar completamente las entidades promoviendo interacción con el usuario final de manera clara

4. Patrón DAO El patrón DAO

se empleó para separar el acceso a los datos del resto de la aplicación. Esto garantiza que las consultas SQL se encuentren encapsuladas y que el resto de la aplicación no dependa de detalles de persistencia. Cada DAO implementa CRUD completo: crear, leer, leer todos, actualizar y eliminar. Los DAOs fueron diseñados para recibir una Connection externa, lo que permite la ejecución de transacciones compuestas provenientes de la capa de servicio. Este diseño resulta especialmente útil cuando se deben ejecutar operaciones relacionadas atómicamente, como en el caso de crear un usuario y su credencial correspondiente dentro de la misma transacción.

5. Capa Service y Manejo Transaccional

La capa de servicios representa uno de los elementos centrales del proyecto. Aquí se implementa la lógica empresarial, se coordinan los DAO mediante una única conexión compartida y se garantiza la integridad de los datos mediante transacciones JDBC. Las transacciones se manejan manualmente: se llama a `setAutoCommit(false)` para iniciar una transacción global y luego a `commit()` o `rollback()` según el resultado de las operaciones internas. El cierre y restauración del autocommit se realiza dentro de bloques try-finally para asegurar la liberación de recursos incluso ante fallos inesperados. Ejemplo de una operación transaccional compuesta: Crear una credencial para el usuario. Asignar el ID generado a la entidad Usuario. Crear el usuario en la base de datos. Confirmar con commit o revertir todo si algo falla.

6. Menú de Consola

Como interfaz principal del sistema, el menú de consola permite al usuario final interactuar fácilmente con las funciones implementadas. Proporciona opciones claras, validaciones de

entrada, mensajes descriptivos y ejecución de cada operación de manera guiada. El menú incluye: Creación de usuarios con su credencial asociada. Lectura por ID. Listar todos los usuarios. Actualizar usuarios. Eliminar lógicamente usuarios. Búsqueda por email. Gestión completa de credenciales.

7. Conclusión

El sistema implementado refleja el cumplimiento riguroso de las consignas asignadas: modelado UML, arquitectura en capas, validaciones del dominio, relación 1→1 correctamente implementada, persistencia mediante JDBC, operaciones transaccionales y un menú funcional e intuitivo. El desarrollo permitió aplicar múltiples conceptos de programación orientada a objetos, técnicas de diseño robustas y procedimientos de persistencia profesional. El resultado es una aplicación completa, funcional y extensible que demuestra una comprensión sólida de los contenidos de la materia.