Trabajo Final Integrador – Programación II

Grupo:

- Lautaro Zullo
- Walter Verdún
- Ignacio Salazar

Tutora: Flor Gubiotti

Profesora: Cinthia Rigoni

Dominio elegido: Usuario → CredencialAcceso

1. Integrantes y roles

- **Ignacio Salazar**: desarrollo de la capa *Service*, manejo de transacciones (commit/rollback), pruebas de integración y revisión de persistencia.

- **Walter Verdun**: diseño y diagramación UML, definición de entidades, estructura de la base de datos y configuración del esquema relacional.

- **Lautaro Zullo**: implementación de la capa *DAO*, manejo de la conexión JDBC y operaciones CRUD sobre las entidades.

El desarrollo se realizó en conjunto bajo una modalidad colaborativa, organizando las tareas en función de los tiempos y las fortalezas técnicas de cada integrante. Si bien parte del código fue implementado por un integrante con mayor experiencia, todos participaron en el análisis, la documentación y las pruebas del sistema.

---

2. Elección del dominio y justificación

El dominio seleccionado fue **Usuario → CredencialAcceso**, una relación que representa un caso muy común en aplicaciones reales: cada usuario registrado posee exactamente una credencial de acceso asociada, que contiene información sensible como la contraseña, su *hash* y el estado de la cuenta.

La elección se basó en los siguientes motivos:

- Es un dominio **simple pero representativo** de sistemas con autenticación de usuarios.

- Permite **demostrar claramente** la relación 1→1 unidireccional entre entidades.

- Facilita la aplicación del patrón DAO, la validación de datos y el manejo de transacciones en operaciones compuestas.

- Tiene **aplicaciones prácticas reales** en sistemas de login, gestión de permisos y administración de cuentas.

Este modelo combina claridad conceptual con desafíos técnicos, como la persistencia transaccional y el control de unicidad entre usuarios y credenciales.

---

3. Diseño del sistema y decisiones clave

El proyecto implementa una relación 1⟶1 unidireccional, donde la clase Usuario contiene una referencia a la clase CredencialAcceso, pero no a la inversa. Es decir, cada usuario posee una sola credencial, y cada credencial pertenece exclusivamente a un usuario.

3.1 Decisión sobre la clave foránea

Se optó por la estrategia de clave foránea única (FK UNIQUE) en lugar de clave primaria compartida.
La tabla Usuarios contiene el campo credencial_id, que actúa como foreign key referenciada a CredencialAcceso(id) y está declarada como UNIQUE para garantizar la relación 1⟶1.
Este diseño ofrece varias ventajas:

- Simplifica las operaciones de inserción y actualización, evitando dependencias circulares.

- Permite mantener la independencia de ambas tablas, facilitando el mantenimiento y la reutilización.

- Asegura integridad referencial con la opción ON DELETE CASCADE, eliminando la credencial asociada si el usuario se da de baja.

3.2 Diagrama UML

El diagrama UML (adjunto al repositorio) muestra las dos clases principales (Usuario y CredencialAcceso) junto con las capas del sistema.

Usuario incluye los atributos id, username, email, activo, fechaRegistro, rol, eliminado y una referencia a CredencialAcceso.

Por su parte, CredencialAcceso posee id, hashPassword, salt, ultimoCambio, requiereReset y eliminado.

Las relaciones y dependencias entre paquetes siguen el patrón clásico por capas: main → service → dao → entities → config.

---

4. Arquitectura por capas

El proyecto sigue una **arquitectura multicapa**, donde cada paquete tiene responsabilidades claramente definidas:

- config/

  Contiene la clase DatabaseConnection, responsable de obtener una conexión JDBC a la base de datos MySQL.

  La configuración se maneja mediante propiedades externas, permitiendo cambiar los parámetros sin modificar el código.

- entities/

  Incluye las clases del modelo de dominio (Usuario y CredencialAcceso).

  Cada entidad define sus atributos, constructores, métodos de acceso y el campo eliminado para manejar bajas lógicas.

  Se mantiene coherencia con el modelo UML y con las restricciones de la base.

- dao/

  Implementa el patrón DAO (Data Access Object).

  Contiene una interfaz genérica GenericDao<T> y las clases UsuarioDao y CredencialAccesoDao.

  Cada DAO utiliza PreparedStatement para ejecutar operaciones CRUD seguras, evitando

inyección SQL.

Además, las operaciones reciben una Connection externa, lo que permite participar en transacciones coordinadas.

- **service/**

  Define la capa de negocio y orquesta las operaciones transaccionales.

  En esta capa se crean los métodos para insertar, actualizar y eliminar entidades, controlando *commit* y *rollback* ante errores.

  También se gestionan reglas de negocio, como evitar la creación de un usuario sin credencial asociada.

- main/

  Contiene la clase AppMenu, que brinda una interfaz de consola para interactuar con el sistema.

  Permite crear, listar, buscar, modificar y eliminar usuarios, mostrando mensajes claros ante errores o acciones exitosas.

Esta organización promueve **modularidad, separación de responsabilidades y mantenibilidad** del código.

---

5. Persistencia y transacciones

5.1 Estructura de la base de datos

El modelo de datos se implementó en MySQL con las siguientes tablas principales:

```
CREATE TABLE CredencialAcceso (
 id INT AUTO_INCREMENT PRIMARY KEY,
 eliminado BOOLEAN DEFAULT FALSE,
```

```
  hashPassword VARCHAR(255) NOT NULL,
  salt VARCHAR(64),
  ultimoCambio TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  requiereReset BOOLEAN NOT NULL
);

CREATE TABLE Usuarios (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(30) NOT NULL UNIQUE,
  email VARCHAR(120) NOT NULL UNIQUE,
  eliminado BOOLEAN DEFAULT FALSE,
  activo BOOLEAN DEFAULT TRUE NOT NULL,
  fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  rol VARCHAR(20) NOT NULL DEFAULT 'Usuario' CHECK (rol IN ('Usuario', 'Moderador',
'Administrador')),
  credencial_id INT NOT NULL UNIQUE,
  FOREIGN KEY (credencial_id) REFERENCES CredencialAcceso(id)
  ON DELETE CASCADE
  ON UPDATE CASCADE
);
```

La clave foránea credencial_id con restricción UNIQUE garantiza la asociación 1⟶1.

Se implementa baja lógica mediante el campo eliminado, evitando eliminar físicamente los registros.

5.2 Orden de operaciones y transacciones

En la capa Service, cada operación compuesta (por ejemplo, crear un usuario junto a su credencial) se ejecuta dentro de una transacción:

1. Se obtiene una conexión y se desactiva el autocommit (setAutoCommit(false)).

2. Se inserta primero la credencial en la tabla CredencialAcceso.

3. Se recupera el id generado y se asocia al nuevo usuario.

4. Se inserta el usuario en la tabla Usuarios.

5. Si todo es correcto, se ejecuta commit().
   En caso de error, se hace rollback() y se muestra un mensaje adecuado.

Esta secuencia asegura **consistencia de datos**, evitando usuarios sin credencial o credenciales huérfanas.

---

6. Validaciones y reglas de negocio

En esta versión del proyecto se implementaron validaciones básicas centradas en la integridad de las operaciones, tales como:

- Verificación de unicidad de username y email al registrar un nuevo usuario.

- Confirmación de existencia de los IDs antes de realizar actualizaciones o eliminaciones.

- Impedimento de crear una segunda credencial para un mismo usuario.

Si bien no se aplicaron validaciones de formato (como verificación de correos válidos o fortaleza de contraseñas), la estructura del proyecto permite incorporar fácilmente estas mejoras dentro de la capa *Service*.

---

7. Conclusiones y mejoras futuras

El trabajo permitió aplicar de manera práctica los conceptos de arquitectura por capas, persistencia con JDBC y manejo transaccional en Java.
La relación 1⟶1 unidireccional se implementó correctamente y cumple los principios de integridad referencial.

Entre las mejoras futuras se consideran:

- Incorporar validaciones de formato y reglas de negocio más estrictas.
- Implementar hashing real de contraseñas con sal y funciones como *PBKDF2* o *bcrypt*.
- Añadir logs de actividad y manejo más detallado de excepciones.
- Evolucionar el menú de consola a una interfaz gráfica básica.
- Introducir pruebas unitarias automáticas para asegurar la calidad del código.

---

8. Fuentes y herramientas utilizadas

- **Lenguaje:** Java 21
- **Base de datos:** MySQL 8
- **Entorno:** IntelliJ IDEA / NetBeans
- **Diseño UML:** Draw.io

- **Control de versiones:** GitHub

- **IA utilizada:** ChatGPT (asistencia en redacción del informe, explicación de conceptos técnicos y revisión de estructura del documento).

- **Documentación adicional:** apuntes de la cátedra y material oficial de Oracle sobre JDBC.