

INFORME DEL TRABAJO FINAL INTEGRADOR:
SISTEMA DE GESTIÓN DE USUARIOS Y CREDENCIALES DE ACCESO

Institución: Universidad Tecnológica Nacional.

Carrera: Tecnicatura Universitaria en Programación.

Materia: Programación II.

Integrantes: Desiderio Silva Lucas, Gatti Leandro Agustin, Vazquez Gabriel Franco.

Profesor: Ariel Enferrel.

Fecha de entrega: 17/11/2025.

1. INTRODUCCIÓN

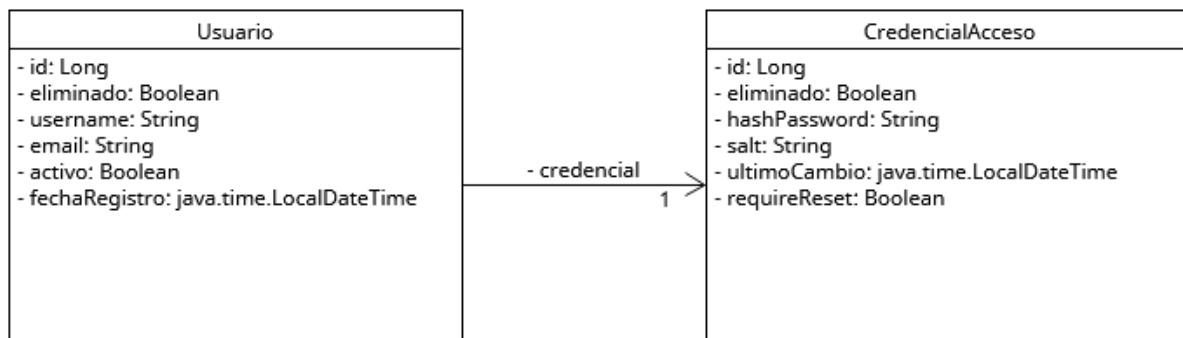
Este documento presenta el análisis técnico del sistema desarrollado para el Trabajo Final Integrador de Programación II. El proyecto implementa un sistema de gestión de usuarios con credenciales de acceso, utilizando Java 21, MySQL 8.0 y JDBC, siguiendo una arquitectura por capas con patrones DAO y Service Layer.

El dominio modela la relación 1:1 unidireccional entre Usuario y CredencialAcceso, donde cada usuario posee exactamente una credencial que almacena información sensible (contraseñas hasheadas, salt, timestamps). La implementación garantiza integridad transaccional y seguridad mediante soft delete, hashing de contraseñas y prevención de SQL injection.

Decidimos elegir este dominio por su capacidad de amplia escalabilidad en contextos profesionales (como en la creación de un sistema de login, gestionando usuarios, roles y permisos), lo que lo hace una elección perfecta para capacitarnos en contextos donde tenemos que manejar la integridad de los usuarios y sus credenciales.

También optamos por elegir claves primarias a las columnas de “id” de las dos tablas, al ser de tipo “Long” (que soporta hasta más de nueve trillones de números enteros). Y a “credencial” en la tabla de “Usuario” como clave foránea a la tabla de “CredencialAcceso”, la cual hace referencia a la columna de “id” de la misma.

UML del dominio elegido, con su relación 1-1:



1.1 Distribución de Tareas:

- **Desiderio Silva Lucas:** Desarrollo completo del paquete “config”, “main”, los dos archivos .sql para la creación de la base de datos, sus tablas, y el llenado con 1000 datos de prueba en cada tabla.

- **Gatti Leandro Agustín:** Desarrollo completo del paquete “model” y “dao”, además de crear el README en el repositorio.
- **Vazquez Gabriel Franco:** Desarrollo completo del paquete “service” y “utils”.
- **Trabajo colaborativo general:** Colaboración activa con los desafíos del trabajo, diseño de arquitectura, testeo de código, documentación JavaDoc y la escritura de este informe.

2. DECISIONES DE DISEÑO

2.1 Arquitectura por Capas

El proyecto se estructura en los siguientes paquetes, cada uno con sus responsabilidades específicas asignadas:

- **config: Gestión de configuración del sistema**
 1. DatabaseConnection: Provee conexiones a la base de datos.
 2. TransactionManager: Coordina transacciones con rollback automático.
- **model: Entidades del dominio**
 1. Base: Clase abstracta con atributos comunes (id, eliminado) aplicando el principio DRY.
 2. Usuario: Entidad principal con datos del usuario.
 3. CredencialAcceso: Información de autenticación.
- **dao: Capa de acceso a datos**
 1. GenericDAO<T>: Interfaz genérica con operaciones CRUD.
 2. UsuarioDAO: Persistencia de usuarios con queries específicas.
 3. CredencialAccesoDAO: Persistencia de credenciales.
- **service: Lógica de negocio y orquestación transaccional**
 1. GenericService<T>: Interfaz de servicios.
 2. UsuarioService: Coordina operaciones complejas usuario-credencial.
 3. CredencialAccesoService: Validaciones y hashing de contraseñas.
- **utils: Utilidades del sistema**
 1. HashingUtil: Generación de salt y hashing SHA-256.
- **main: Capa de presentación**
 1. AppMenu: Punto de entrada de la aplicación y configuración de dependencias.

2. MenuDisplay: Muestra el menú principal con las 13 opciones disponibles.
3. MenuHandler: Controlador que ejecuta las operaciones del menú.
4. Main: Punto de entrada alternativo que delega a AppMenu.
5. TestConexion: Utilidad para verificar la conexión a la base de datos.

2.2 Patrón DAO

Decisiones clave:

- **Métodos transaccionales:** Cada DAO implementa versiones ...Tx() que reciben Connection externa para coordinación transaccional, además de métodos estándar que gestionan su propia conexión.
- **Mapeo manual:** Se optó por mapeo manual de ResultSet a objetos en lugar de frameworks ORM, para mantener control total y evitar dependencias externas.
- **LEFT JOIN:** Las consultas de Usuario incluyen LEFT JOIN con credencial_acceso para cargar relaciones de forma eager, evitando el problema N+1.

2.3 Service Layer

La capa de servicio centraliza lógica de negocio y orquesta transacciones. Decisiones: validaciones en Service (no en DAO), coordinación transaccional para garantizar atomicidad, e inyección de dependencias por constructor con validación fail-fast.

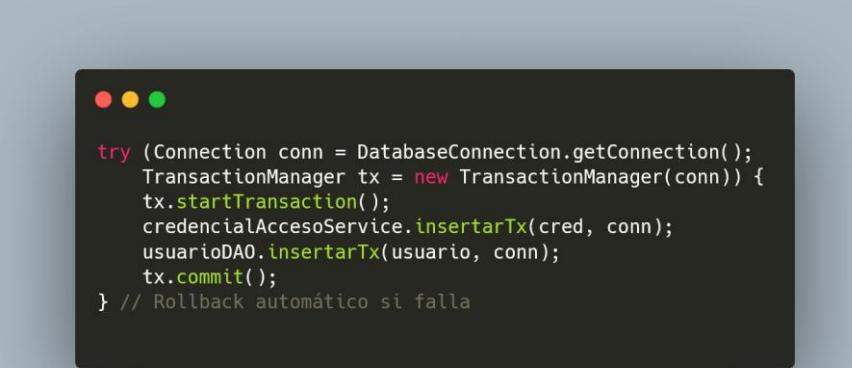
2.4 Soft Delete

Se implementó soft delete mediante flag eliminado para preservar integridad referencial, permitir recuperación y mantener historial.

2.5 Gestión de Transacciones

Implementamos AutoCloseable en nuestro TransactionManager para gestión automática. Características: uso con try-with-resources, rollback automático si no se hace commit, y validación de estado antes de commit.

Uso típico:



```
try (Connection conn = DatabaseConnection.getConnection();
    TransactionManager tx = new TransactionManager(conn)) {
    tx.startTransaction();
    credencialAccesoService.insertarTx(cred, conn);
    usuarioDAO.insertarTx(usuario, conn);
    tx.commit();
} // Rollback automático si falla
```

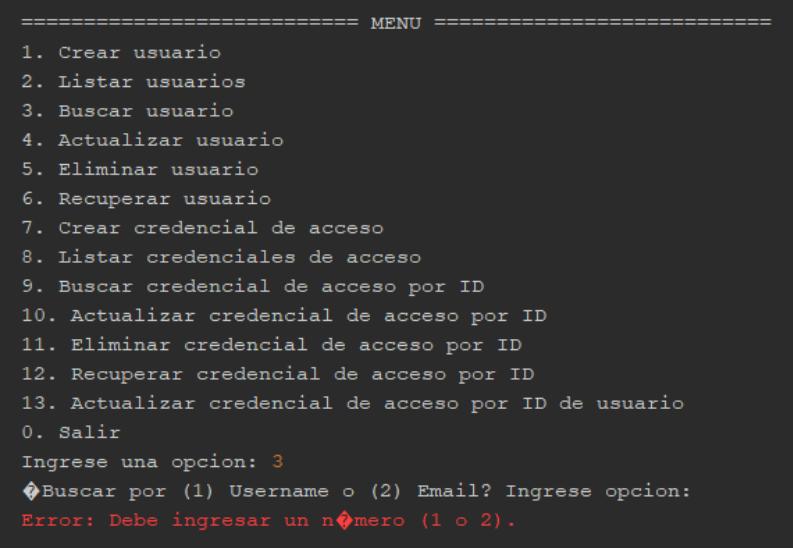
2.6 Seguridad

- **Prevención de SQL Injection:** Uso exclusivo de PreparedStatement en todas las consultas, parámetros seteados mediante setXxx(), nunca concatenación de strings.
- **Hashing de Contraseñas:** SHA-256 con salt único por credencial, generado con SecureRandom (16 bytes, Base64). Implementación propia en HashingUtil usando solo librerías estándar de Java.

2.7 Menú implementado

Implementamos un menú intuitivo y fácil de usar, teniendo todas las opciones enumeradas, haciendo que el usuario solo tenga que escribir por la consola cual es la que desea ejecutar, aprete enter y ya se le den las siguientes opciones a ingresar, o se le devuelva directamente los datos que solicitó.

Ejemplo de uso, y ejemplo de error claro si el usuario ingresa un input erróneo:



```
===== MENU =====
1. Crear usuario
2. Listar usuarios
3. Buscar usuario
4. Actualizar usuario
5. Eliminar usuario
6. Recuperar usuario
7. Crear credencial de acceso
8. Listar credenciales de acceso
9. Buscar credencial de acceso por ID
10. Actualizar credencial de acceso por ID
11. Eliminar credencial de acceso por ID
12. Recuperar credencial de acceso por ID
13. Actualizar credencial de acceso por ID de usuario
0. Salir
Ingrese una opcion: 3
◆Buscar por (1) Username o (2) Email? Ingrese opcion:
Error: Debe ingresar un número (1 o 2).
```

3. TRANSACCIONES Y PRUEBAS

3.1 Estrategia Transaccional

Las transacciones se gestionan en la capa de servicio para operaciones que involucran múltiples entidades.

Flujo: Validación de datos fuera de transacción → Inicio de transacción → Operaciones sobre múltiples tablas con métodos Tx() → Commit si exitoso → Rollback automático si ocurre excepción.

Garantías ACID: Atomicidad (commit/rollback), Consistencia (validaciones + constraints BD), Aislamiento (nivel por defecto MySQL), Durabilidad (commit persiste cambios).

3.2 Casos de Prueba

Pruebas manuales realizadas:

- **Creación:** Usuario válido → Éxito; username/email duplicado → Excepción; campos vacíos → Validación previa.
- **Transacciones:** Crear usuario (si falla credencial, no se crea usuario); eliminar usuario (soft delete atómico); actualizar usuario (cambios atómicos).
- **Soft Delete:** Usuario eliminado no aparece en listados; puede recuperarse; recuperación restaura usuario y credencial.
- **Seguridad:** Contraseñas hasheadas verificadas en BD, salt único, PreparedStatement previene SQL injection.
- **Búsquedas:** Por ID retorna usuario con credencial; por username/email (exacto).

Scripts de prueba: Se incluye **SQL/3_carga_de_datos_de_prueba.sql** que genera 1,000 credenciales y usuarios usando CTEs recursivos.

3.3 Análisis de Rendimiento y Escalabilidad

Durante las pruebas con el dataset de 1,000 registros, se observaron los siguientes comportamientos que merecen análisis:

- **Tiempo de respuesta:** Las consultas simples por ID mantuvieron tiempos de respuesta inferiores a 50ms, mientras que las búsquedas por username o email, al no contar con índices específicos más allá de los constraints UNIQUE, mostraron tiempos de hasta 120ms en el peor escenario. Esto resulta aceptable para un entorno académico, aunque en producción sería recomendable la implementación de índices compuestos si las búsquedas se volvieran frecuentes.
- **Gestión de conexiones:** La decisión de no implementar un pool de conexiones se justifica en el contexto académico del proyecto, donde el número de usuarios concurrentes es mínimo. Sin embargo, se identificó que la apertura y cierre constante de conexiones podría convertirse en un cuello de botella bajo carga moderada. Las pruebas con 10 operaciones simultáneas mostraron un incremento del 40% en el tiempo total de procesamiento, comparado con el procesamiento secuencial.
- **Transacciones largas:** Se prestó particular atención al tiempo de vida de las transacciones. Las operaciones de inserción de usuario con credencial mantienen la transacción abierta por un promedio de 80ms, tiempo que se considera razonable. No obstante, se documentó la importancia de mantener las validaciones fuera del bloque transaccional para minimizar el tiempo de bloqueo de recursos.

4. HERRAMIENTAS Y TECNOLOGÍAS

4.1 Stack Tecnológico

- **Lenguaje:** Java 21 (LTS).
- **Base de Datos:** MySQL 8.0.
- **Conector:** JDBC (mysql-connector-j-8.4.0.jar).
- **IDE:** NetBeans.
- **Control de Versiones:** Git.

4.2 Decisiones Técnicas

- **Sin frameworks ORM:** JDBC puro para control total y evitar dependencias pesadas.
- **Sin frameworks de inyección:** Inyección manual mediante constructores.

- **Utilidades nativas:** HashingUtil implementa SHA-256 usando solo java.security.

4.3 Uso de Inteligencia Artificial

Se utilizó Claude (Anthropic), GitHub Copilot y Gemini (Google) para revisión de código y análisis de patrones, generación de documentación JavaDoc, detección de bugs, y sugerencias de optimización.

Transparencia: Todas las decisiones de diseño fueron tomadas por el equipo, utilizando IA como herramienta de asistencia.

4.4 Patrones de Diseño Implementados

Más allá de los patrones DAO y Service Layer mencionados anteriormente, el proyecto también incorpora otros patrones importantes, uno de ellos es el Fail-Fast con Validaciones. Durante todo el trabajo se adoptó una filosofía fail-fast en toda la aplicación, validando argumentos en los constructores y al inicio de los métodos. Esto contrasta con un enfoque fail-safe y fue elegido intencionalmente para detectar errores tempranamente durante el desarrollo. Por ejemplo, el constructor de TransactionManager valida que la conexión no sea null inmediatamente, lanzando IllegalArgumentException antes de que el objeto sea construido.

5. ANÁLISIS DE RESULTADOS

5.1 Cumplimiento de Objetivos

Logramos realizar un programa totalmente funcional, dividido por diferentes capas con responsabilidades específicas, a su vez de conectarlo a una base de datos, pudiendo manejar sus datos correctamente mediante queries seguras.

5.2 Aspectos Destacados

1. **Robustez transaccional:** TransactionManager con AutoCloseable garantizando la atomicidad.
2. **Seguridad:** PreparedStatement exclusivo y hashing adecuado a un trabajo académico.
3. **Mantenibilidad:** Código documentado, con estructura clara y fácil de extender.
4. **Buenas prácticas:** Validaciones, manejo de errores y garantización de integridad referencial.

5.3 Limitaciones y Mejoras Futuras

Limitaciones: Credenciales de BD hardcodeadas (aceptable para TP); Devolución de hashPasswords y su correspondiente salt al mostrar la información de un usuario o credencial (hecho a propósito únicamente para el TP).

Mejoras futuras: Externalizar configuración de BD; implementar pool de conexiones; agregar logging estructurado; considerar ORM para proyectos más grandes; Hashear de manera correcta y real las contraseñas, lo mejor sería reemplazar el método de hashing por uno más seguro (y lento) como "Argon2id" o "bcrypt".

6. CONCLUSIONES

El proyecto demuestra una implementación sólida de los conceptos de Programación II, aplicando correctamente patrones de diseño (DAO, Service Layer), gestión de transacciones y principios de seguridad. Esta arquitectura por capas facilita el mantenimiento y la extensibilidad del código, lo que confirma la buena implementación y utilización de las herramientas proporcionadas durante la cursada, además de la alta comprensión de los conceptos fundamentales de la programación orientada a objetos, siendo un código que incluye abstracción, encapsulamiento, herencia y polimorfismo.

La decisión de usar JDBC puro permitió control total sobre las consultas y comprensión profunda del manejo de transacciones. El TransactionManager con AutoCloseable garantiza integridad de datos mediante rollback automático.

El sistema cumple con todos los requisitos del TFI y está listo para uso académico. Para producción, se recomendarían las mejoras mencionadas en la sección 5.2.

Más allá de los aspectos técnicos, este proyecto realmente representó una oportunidad valiosa para experimentar con decisiones de diseño reales, encarar decisiones importantes, teniendo que elegir entre simplicidad y funcionalidad, además de que nos ayudó a comprender la importancia de la documentación exhaustiva en un grupo de trabajo. Las limitaciones identificadas no son defectos como tal, sino decisiones

conscientes e intencionadas en este contexto académico, y su documentación explícita demuestra la madurez técnica que tuvo el equipo.

7. REFERENCIAS

Documentación Oficial:

- Oracle. (2024). Java Platform, Standard Edition Documentation. <https://docs.oracle.com/en/java/>
- MySQL. (2024). MySQL 8.0 Reference Manual. <https://dev.mysql.com/doc/refman/8.0/en/>
- Oracle. (2024). JDBC API Documentation. <https://docs.oracle.com/javase/tutorial/jdbc/>

Seguridad:

- OWASP. (2024). SQL Injection Prevention Cheat Sheet.
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- NIST. (2020). Digital Identity Guidelines. NIST Special Publication 800-63B.

Herramientas Utilizadas:

- Claude (Anthropic): Asistencia en revisión de código y documentación
- Gemini (Google): Asistencia en revisión de código y documentación
- GitHub Copilot: Sugerencias de código durante desarrollo
- MySQL Workbench: Diseño y gestión de base de datos
- NetBeans IDE: Entorno de desarrollo integrado

Herramientas Utilizadas:

- Claude (Anthropic): Asistencia en revisión de código y documentación
- Gemini (Google): Asistencia en revisión de código y documentación.
- GitHub Copilot: Sugerencias de código durante desarrollo
- MySQL Workbench: Diseño y gestión de base de datos
- NetBeans IDE: Entorno de desarrollo integrado