INFORME TÉCNICO

Sistema de Gestión de Usuarios y Credenciales de Acceso

Trabajo Final Integrador - Programación II

Tecnicatura Universitaria en Programación - UTN

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

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.

2**. DECISIONES DE DISEÑO**

2.1 **Arquitectura por Capas**

Se adoptó una arquitectura de cuatro capas: Presentación → Servicio → DAO → Modelo. Esta estructura permite testabilidad, mantenibilidad y escalabilidad, facilitando la incorporación de nuevas funcionalidades sin afectar capas existentes.

2.2 **Patrón DAO**

Se implementó mediante la interfaz genérica GenericDAO<T> y clases concretas UsuarioDAO y CredencialAccesoDAO. 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 **Clase Base y Soft Delete**

Se creó la clase Base con atributos comunes (id, eliminado) extendida por Usuario y CredencialAcceso, aplicando el principio DRY. Se implementó soft delete mediante flag eliminado para preservar integridad referencial, permitir recuperación y mantener historial.

2.5 **Gestión de Transacciones**

Se diseñó TransactionManager que implementa AutoCloseable 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.

3. **TRANSACCIONES Y PRUEBAS**

3.1 **Estrategia Transaccional**

- Las transacciones se gestionan en la capa de servicio para operaciones que involucran múltiples entidades. Flujo: (1) Validación de datos fuera de transacción, (2) Inicio de transacción, (3) Operaciones sobre múltiples tablas con métodos ...Tx(), (4) Commit si exitoso, (5) 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.

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) y GitHub Copilot 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.

5. **ANÁLISIS DE RESULTADOS**

5.1 **Cumplimiento de Objetivos**

1. Arquitectura por capas con separación clara de responsabilidades
2. Patrón DAO con interfaz genérica y clases concretas
3. Service Layer con lógica de negocio centralizada y gestión transaccional
4. Transacciones con rollback automático
5. Seguridad: prevención de SQL injection y hashing de contraseñas
6. Soft Delete implementado correctamente
7. CRUD completo para ambas entidades

5.2 Aspectos Destacados

1. Robustez transaccional: TransactionManager con AutoCloseable garantiza atomicidad.

2. Seguridad: PreparedStatement exclusivo y hashing adecuado.

3. Mantenibilidad: Código documentado, estructura clara, fácil de extender.

4. Buenas prácticas: Validaciones, manejo de errores, convenciones.

5.3 Limitaciones y Mejoras Futuras

Limitaciones: Credenciales de BD hardcodeadas (aceptable para TP); método getRequireReset() debería ser isRequireReset(); validación de email solo verifica no vacío.

Mejoras sugeridas: Externalizar configuración de BD; implementar pool de conexiones; agregar logging estructurado; considerar ORM para proyectos más grandes.

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. La arquitectura por capas facilita el mantenimiento y la extensibilidad.
- 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 5.3.

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
- GitHub Copilot: Sugerencias de código durante desarrollo
- MySQL Workbench: Diseño y gestión de base de datos
- NetBeans IDE: Entorno de desarrollo integrado