



TRABAJO FINAL INTEGRADOR: Programación II

Sistema de Gestión de Usuarios y Credenciales de Acceso

Alumnos:

4822 - Basualdo Arcati Pablo - pbasualdoarcati@gmail.com

5721 - Calcatelli Renzo - rccalcatelli@gmail.com

7589 - Kaddarian Dante - dantek91@yahoo.com

Comisión: M2025-1

Materia: Programación II

Profesor: Ariel Enferrel

 [Enlace](#) al repositorio de GitHub

 [Enlace](#) al video explicativo

Índice

1. Integrantes y Roles

- Información de integrantes y asignación de responsabilidades
- Nota sobre metodología de trabajo grupal

2. Elección del Dominio y Justificación

- 2.1. Dominio Seleccionado
- 2.2. Justificación de la Elección
- 2.3. Modelo de Datos

3. Diseño y Decisiones Técnicas

- 3.1. Relación 1:1 Unidireccional
- 3.2. FK Única vs PK Compartida
- 3.3. Diagrama UML

4. Arquitectura por Capas

- 4.1. Flujo de Datos
- 4.2. Principios de Diseño Aplicados

5. Persistencia y Transacciones

- 5.1. Estructura de la Base de Datos
- 5.2. Decisiones de Diseño de BD
- 5.3. Gestión de Transacciones
- 5.4. Orden de Operaciones
- 5.5. Seguridad en Persistencia

6. Validaciones y Reglas de Negocio

- 6.1. Validaciones de Usuario
- 6.2. Validaciones de CredencialAcceso
- 6.3. Reglas de Negocio Principales

7. Pruebas Realizadas

- 7.1. Pruebas de Funcionalidad
- 7.2. Consultas SQL Útiles
- 7.3. Menú de Consola

8. Conclusiones y Mejoras Futuras

- 8.1. Conclusiones
- 8.2. Logros del Proyecto
- 8.3. Mejoras Futuras
- 8.4. Reflexión Personal

9. Referencias y Herramientas

- 9.1. Documentación Técnica
- 9.2. Herramientas Utilizadas
- 9.3. Uso de Inteligencia Artificial

1. Integrantes y Roles

Este Trabajo Final Integrador fue desarrollado de manera individual como parte de la cursada de Programación II. A continuación se detalla la información del integrante y las responsabilidades asumidas durante el proyecto:

Integrante	Legajo	Rol Principal	Responsabilidades
Basualdo Arcati Pablo	4822	Desarrollador Full-Stack Tester	<ul style="list-style-type: none"> • Diseño y creación de base de datos <ul style="list-style-type: none"> • Diseño de la arquitectura del sistema • Implementación de entidades y relaciones • Desarrollo de capa DAO y Service <ul style="list-style-type: none"> • Pruebas
Calcatelli Renzo	5721	Desarrollador Full-Stack Investigación	<ul style="list-style-type: none"> • Desarrollo de capa DAO y Service • Diseño y creación de base de datos <ul style="list-style-type: none"> • Documentación • Pruebas • Implementación de interfaz de usuario
Kaddarian Dante	7589	Desarrollador Full-Stack Tester	<ul style="list-style-type: none"> • Diseño y creación de base de datos • Implementación de interfaz de usuario <ul style="list-style-type: none"> • Pruebas • Desarrollo de capa DAO y Service

Nota: Cabe aclarar que todos los integrantes participaron activamente y asumieron las mismas responsabilidades a lo largo de todo el trabajo, el cual se realizó de forma grupal desde el inicio. Las únicas tareas que se asignaron individualmente al finalizar fueron las pruebas y la documentación, completadas una vez terminada la implementación del proyecto en Java. Si bien la consigna original establecía equipos de cuatro integrantes, este proyecto fue desarrollado por tres personas, bajo la supervisión y aprobación del docente, asumiendo íntegramente las tareas de diseño, implementación y documentación.

2. Elección del Dominio y Justificación

2.1. Dominio Seleccionado

Para este proyecto se seleccionó el dominio **Usuario** → **CredencialAcceso**, una de las opciones propuestas en las consignas del TFI. Esta elección modela la relación entre un usuario del sistema y sus credenciales de acceso, implementando una asociación unidireccional 1 a 1.

2.2. Justificación de la Elección

La elección de este dominio se fundamenta en los siguientes criterios técnicos y didácticos:

- **Relevancia práctica:** El sistema de gestión de usuarios y credenciales es un componente fundamental en prácticamente cualquier aplicación moderna, lo que hace este dominio altamente relevante para el aprendizaje profesional.
- **Claridad conceptual:** La relación **Usuario-CredencialAcceso** permite demostrar de manera clara y natural la asociación unidireccional 1:1, donde un usuario conoce sus credenciales pero las credenciales no necesitan conocer al usuario.
- **Complejidad adecuada:** Este dominio ofrece la complejidad justa para implementar todos los requisitos del TFI: operaciones CRUD, transacciones, validaciones complejas y reglas de negocio significativas.
- **Aspectos de seguridad:** Permite incorporar conceptos importantes de seguridad informática como hash de contraseñas, control de intentos fallidos y validación de credenciales, ampliando el alcance educativo del proyecto.
- **Escalabilidad conceptual:** El modelo elegido puede ser fácilmente extendido en el futuro para incluir roles, permisos, autenticación multifactor, etc., demostrando buenas prácticas de diseño.

2.3. Modelo de Datos

El modelo implementado consiste en dos entidades principales:

Entidad	Atributos Principales	Propósito
Usuario	<ul style="list-style-type: none">• id (Long)• username (String)• email (String)• nombreCompleto (String)• activo (Boolean)• eliminado (Boolean)• fechaRegistro (LocalDateTime)• credencialAcceso	Representa a una persona que utiliza el sistema
CredencialAcceso	<ul style="list-style-type: none">• id (Long)• usuarioId (Long)• hashPassword (String)• salt (String)• ultimoCambio (LocalDateTime)• requiereReset (Boolean)• intentosFallidos (Integer)• ultimoLogin (LocalDateTime)• eliminado (Boolean)	Almacena información de autenticación y seguridad

3. Diseño y Decisiones Técnicas

3.1. Relación 1:1 Unidireccional

La implementación de la relación 1:1 unidireccional es el aspecto central de este proyecto. A continuación se detallan las decisiones de diseño tomadas:

Dirección de la asociación:

Usuario → CredencialAcceso (unidireccional). La clase Usuario contiene una referencia privada a CredencialAcceso, pero CredencialAcceso no conoce a Usuario. Esta decisión se basa en el principio de que "un usuario tiene credenciales" pero las credenciales no necesitan saber a qué usuario pertenecen desde la perspectiva del modelo de objetos Java.

3.2. FK Única vs PK Compartida

Se evaluaron dos alternativas para implementar la relación 1:1 en la base de datos:

Criterio	FK Única (Implementada)	PK Compartida (Alternativa)
Estructura	<code>credencial_acceso.usuario_id</code> UNIQUE + FOREIGN KEY	<code>credencial_acceso.id = usuario.id</code> (PK y FK)
Ventajas	<ul style="list-style-type: none"> • IDs independientes • Mayor flexibilidad • Más intuitiva • Mejor para migraciones 	<ul style="list-style-type: none"> • Garantía absoluta 1:1 • Ligeramente más eficiente • Menos espacio en disco
Desventajas	Requiere constraint UNIQUE adicional	<ul style="list-style-type: none"> • IDs acoplados • Menos flexible • Complejidad en inserciones

Decisión final: Se optó por implementar la relación mediante **Foreign Key única** (`usuario_id` UNIQUE en `credencial_acceso`) por su mayor flexibilidad y facilidad de mantenimiento. Esta aproximación permite que cada entidad mantenga su propio identificador independiente, facilitando futuras extensiones del modelo y reduciendo el acoplamiento entre tablas.

3.3. Diagrama UML

El diagrama UML completo del sistema se adjunta como archivo separado. El diagrama incluye:

- **Todos los paquetes del sistema** (config, entities, dao, service, main).
- **Clases con atributos completos** (tipos y visibilidad).
- **Métodos con firmas completas** (parámetros y tipos de retorno).
- **Relación 1:1 unidireccional** Usuario → CredencialAcceso claramente marcada.
- **Interfaces genéricas** (GenericDao, GenericService).
- **Relaciones de implementación, dependencia y uso entre componentes**.
- **Notas explicativas de patrones y decisiones de diseño**.

4. Arquitectura por Capas

El sistema implementa una arquitectura en capas bien definida, separando las responsabilidades en cinco paquetes principales:

Capa/Paquete	Responsabilidades	Clases Principales
config	Configuración del sistema <ul style="list-style-type: none"> Gestión de conexiones a BD <ul style="list-style-type: none"> Lectura de propiedades <ul style="list-style-type: none"> Patrón Singleton Recursos compartidos 	DatabaseConfig
entities	Modelo de dominio <ul style="list-style-type: none"> Representación de objetos <ul style="list-style-type: none"> Encapsulación de datos Relaciones entre entidades <ul style="list-style-type: none"> POJOs con baja lógica 	Usuario CredencialAcceso
dao	Acceso a datos <ul style="list-style-type: none"> Operaciones CRUD Mapeo objeto-relacional <ul style="list-style-type: none"> Queries SQL Gestión de ResultSet 	GenericDao UsuarioDao CredencialAccesoDao
service	Lógica de negocio <ul style="list-style-type: none"> Validaciones Transacciones Reglas de negocio Orquestación de DAOs 	GenericService UsuarioService CredencialAccesoService
main	Interfaz de usuario <ul style="list-style-type: none"> Interacción con usuario <ul style="list-style-type: none"> Menús de consola Manejo de entrada/salida <ul style="list-style-type: none"> Punto de entrada 	Main AppMenu

4.1. Flujo de Datos

El flujo típico de una operación en el sistema sigue este patrón:

1. AppMenu recibe entrada del usuario y la valida.
2. AppMenu invoca el método correspondiente en Service.
3. Service aplica validaciones de negocio.
4. Service inicia transacción (**setAutoCommit(false)**).
5. Service invoca operaciones en DAO, pasando la Connection.
6. DAO ejecuta SQL mediante PreparedStatement.
7. DAO mapea ResultSet a objetos.
8. Service realiza **commit()** o **rollback()** según resultado.
9. Service retorna resultado a AppMenu.

10. AppMenu muestra resultado al usuario.

4.2. Principios de Diseño Aplicados

- **Separación de Responsabilidades (SRP):** Cada clase tiene una única razón para cambiar. Los DAOs solo manejan persistencia, los Services solo lógica de negocio, etc.
 - **Inversión de Dependencias (DIP):** Las capas superiores dependen de abstracciones (interfaces GenericDao y GenericService) no de implementaciones concretas.
 - **Abierto/Cerrado (OCP):** El sistema es extensible mediante herencia de interfaces genéricas sin modificar código existente.
 - **Don't Repeat Yourself (DRY):** Código común está centralizado en interfaces genéricas y métodos compartidos.
-

5. Persistencia y Transacciones

5.1. Estructura de la Base de Datos

La base de datos MySQL implementa el modelo relacional siguiendo estos principios:

Decisiones de diseño implementadas:

- **username vs nombre/apellido:** Se decidió usar un campo único `username` para login y un campo `nombre_completo` para la información personal. Esto simplifica la autenticación y evita problemas con nombres compuestos o culturalmente diversos.
- **Tamaño de email:** Se usa VARCHAR(100) en lugar de VARCHAR(255) como optimización. 100 caracteres es suficiente para el 99.9% de emails reales y mejora el rendimiento de los índices.
- **Campo activo:** Permite desactivar cuentas temporalmente sin eliminarlas. Un usuario puede estar inactivo (activo=FALSE) sin estar eliminado, útil para suspensiones.
- **fecha_registro con DEFAULT CURRENT_TIMESTAMP:** Se asigna automáticamente, evitando errores de omisión y garantizando auditoría precisa.
- **Múltiples índices:** Se crean índices en todos los campos usados frecuentemente en WHERE, JOIN y ORDER BY para optimizar consultas.
- **NOT NULL en campos críticos:** Todos los campos esenciales son NOT NULL para garantizar integridad de datos desde la base de datos.

Decisiones de diseño implementadas:

- **Eliminación de nombre_usuario:** Se eliminó la redundancia. El username está en la tabla usuario. La credencial se vincula mediante usuario_id (FK).
- **hash_password VARCHAR(255):** Tamaño generoso para soportar diferentes algoritmos de hashing (BCrypt, Argon2, etc.) sin necesidad de migración futura.
- **salt VARCHAR(64):** Aunque algunos algoritmos incluyen el salt en el hash, lo almacenamos por separado para flexibilidad. 64 caracteres soporta salts de hasta 32 bytes en hexadecimal.
- **ultimo_cambio:** Permite implementar políticas de expiración de contraseñas (ej: "cambiar cada 90 días").

- **requiere_reset:** Útil para contraseñas temporales, reseteos forzados por administrador, o políticas de seguridad.
- **ultimo_login vs ultimo_intento:** Se usa `ultimo_login` para registrar accesos exitosos, no intentos fallidos. Los intentos fallidos se cuentan con `intentos_fallidos`.
- **ON UPDATE CASCADE:** Si por alguna razón se cambia el ID de un usuario (muy raro pero posible en migraciones), la FK se actualiza automáticamente.
- **Constraint nombrado:** `CONSTRAINT fk_credencial_usuario` facilita el debugging y la documentación de errores de integridad referencial.

5.2. Decisiones de Diseño de BD

- **ENGINE=InnoDB:** Para soporte de transacciones ACID y foreign keys.
- **CHARSET=utf8mb4:** Para soporte completo de caracteres Unicode, incluyendo emojis.
- **AUTO_INCREMENT en id:** IDs generados automáticamente por la base de datos.
- **UNIQUE en usuario_id:** Garantiza la relación 1:1 a nivel de base de datos.
- **UNIQUE en email y nombre_usuario:** Previene duplicados y mejora búsquedas.
- **ON DELETE CASCADE:** Si se elimina un usuario, su credencial también se elimina automáticamente.
- **Índices estratégicos:** En campos de búsqueda frecuente (email, nombre_usuario, eliminado).
- **usuario_id en credencial_acceso:** Foreign Key que establece la relación 1:1 con la tabla usuario. El constraint UNIQUE garantiza que cada usuario solo pueda tener una credencial.
- **Eliminación de nombre_usuario de credencial_acceso:** Para evitar redundancia, el username se almacena únicamente en la tabla usuario. La credencial se vincula mediante usuario_id.
- **Campo activo:** Permite desactivar usuarios temporalmente sin eliminarlos, útil para suspensiones o mantenimiento de cuentas.
- **Campo fecha_registro:** Timestamp automático de creación del usuario para auditoría.
- **Campo ultimo_cambio:** Registra la última modificación de contraseña para políticas de seguridad.
- **Campo requiere_reset:** Flag para forzar cambio de contraseña en el próximo login.

5.3. Orden de Operaciones

El orden de las operaciones en transacciones compuestas es crítico:

- **Creación:** 1. Usuario (padre) → 2. CredencialAcceso (hija con FK)
- **Actualización:** 1. Validar existencia → 2. Usuario → 3. CredencialAcceso
- **Eliminación lógica:** 1. Usuario.eliminado=true → 2. CredencialAcceso.eliminado=true
- **Eliminación física:** CASCADE elimina automáticamente credencial al eliminar usuario

5.4. Seguridad en Persistencia

- PreparedStatement en TODAS las operaciones SQL (prevención de SQL Injection).
- Contraseñas hasheadas con SHA-256 + salt único por usuario.
- Nunca se almacenan contraseñas en texto plano.
- Salt aleatorio generado con SecureRandom.
- Control de intentos fallidos de login (máximo 5).

- Registro de timestamp del último intento de acceso.
-

6. Validaciones de Usuario

- **Campos obligatorios:** Username, email y nombreCompleto no pueden ser null ni vacíos.
- **Formato de email:** Validación mediante expresión regular RFC 5322.
- **Unicidad de email:** Verificación en base de datos antes de insertar/actualizar.
- **Unicidad de username:** Verificación en base de datos antes de insertar/actualizar.
- **Longitud de username:** Entre 3 y 50 caracteres.
- **Longitud de nombreCompleto:** Entre 2 y 100 caracteres.
- **Caracteres permitidos en username:** Solo letras, números, guiones y guiones bajos.
- **Caracteres permitidos en nombreCompleto:** Letras, espacios y caracteres acentuados.
- **Estado activo:** Por defecto true al crear un usuario nuevo.
- **Fecha de registro:** Se asigna automáticamente al momento de creación.

6.1. Validaciones de CredencialAcceso

- **Contraseña:** Mínimo 8 caracteres. La validación de complejidad se puede extender según políticas de seguridad.
- **Hash seguro:** SHA-256 con salt único de 16 bytes generado con SecureRandom.
- **Unicidad de salt:** Cada usuario tiene su propio salt único.
- **Intentos fallidos:** Se incrementa con cada login fallido. Máximo 5 intentos.
- **Bloqueo temporal:** Después de 5 intentos fallidos, se debe esperar o contactar administrador.
- **Relación 1:1 obligatoria:** Un usuario solo puede tener una credencial activa, garantizado por constraint UNIQUE en usuario_id.
- **Último login:** Se registra automáticamente en cada autenticación exitosa.
- **Requiere reset:** Permite forzar cambio de contraseña en el próximo acceso.
- **Último cambio:** Registra la fecha de la última modificación de contraseña para auditoría y políticas de expiración.

6.2. Reglas de Negocio Principales

- **Integridad referencial:** Una credencial no puede existir sin un usuario asociado. Si se elimina el usuario, se elimina la credencial (CASCADE).
- **Baja lógica:** Los registros no se eliminan físicamente, se marca el campo 'eliminado' como true, permitiendo auditoría y recuperación de datos.
- **Unicidad de credenciales:** Un usuario solo puede tener una credencial activa. Si se intenta crear otra, el sistema lanza una excepción.
- **Validación de login:** Se incrementa el contador de intentos fallidos. Después de 5 intentos, se debe esperar 15 minutos o contactar administrador.
- **Cambio de contraseña:** Requiere la contraseña actual. Se genera nuevo salt y hash para cada cambio.
- **Estado de usuario activo/inactivo:** Un usuario puede estar inactivo sin ser eliminado.
- **Los usuarios inactivos no pueden hacer login:** pero sus datos permanecen en el sistema.
- **Fecha de registro inmutable:** La fecha de registro se establece al crear el usuario y

no debe modificarse posteriormente.

- **Vinculación obligatoria:** Toda credencial debe estar vinculada a un usuario mediante usuario_id. No pueden existir credenciales huérfanas.
- **Último cambio de contraseña:** Se actualiza automáticamente cada vez que se modifica el hash de la contraseña, permitiendo implementar políticas de expiración.
- **Reset de intentos fallidos:** Los intentos fallidos se resetean a 0 cuando hay un login exitoso.

7. Pruebas Realizadas

Se realizaron pruebas exhaustivas de todas las funcionalidades del sistema, incluyendo casos normales, casos límite y manejo de errores.

7.1. Pruebas de Funcionalidad

Funcionalidad	Casos Probados	Resultado
Crear Usuario	<ul style="list-style-type: none"> • Datos válidos • Email duplicado • Campos vacíos • Email inválido 	✓ Exitoso
Crear Usuario con Credencial	<ul style="list-style-type: none"> • Creación completa • Transacción con rollback <ul style="list-style-type: none"> • Usuario duplicado • Contraseña débil 	✓ Exitoso
Actualizar Usuario	<ul style="list-style-type: none"> • Actualización normal <ul style="list-style-type: none"> • ID inexistente • Email a uno existente • Campos inválidos 	✓ Exitoso
Eliminar Usuario	<ul style="list-style-type: none"> • Eliminación lógica <ul style="list-style-type: none"> • ID inexistente • CASCADE a credencial • Recuperar eliminado 	✓ Exitoso
Validar Login	<ul style="list-style-type: none"> • Credenciales correctas • Contraseña incorrecta <ul style="list-style-type: none"> • Usuario no existe • 5+ intentos fallidos 	✓ Exitoso
Búsquedas	<ul style="list-style-type: none"> • Por ID • Por email • Por nombre usuario • Listar todos 	✓ Exitoso

7.2. Menú de Consola

El sistema implementa un menú interactivo en consola con las siguientes opciones:

1. **Crear Usuario con Credencial:** Operación transaccional que crea simultáneamente un usuario y su credencial de acceso asociada. Solicita username, email, nombre completo y contraseña.
2. **Buscar Usuario por ID:** Permite localizar un usuario específico mediante su identificador único. Muestra toda la información del usuario incluyendo sus credenciales asociadas.
3. **Buscar Usuario por Username:** Búsqueda de usuario utilizando el nombre de usuario como criterio. Retorna la información completa del usuario y sus credenciales.
4. **Listar Todos los Usuarios:** Muestra un listado completo de todos los usuarios registrados en el sistema, incluyendo su estado de activación y eliminación. Presenta información resumida de cada usuario.
5. **Actualizar Usuario:** Permite modificar los datos de un usuario existente (email y nombre completo). El sistema muestra los datos actuales y solicita los nuevos valores, manteniendo los existentes si no se ingresan cambios.
6. **Activar/Desactivar Usuario:** Gestiona el estado de activación de un usuario. Permite alternar entre estados activo e inactivo sin eliminar datos, útil para suspensiones temporales de acceso.
7. **Eliminar Usuario (Baja Lógica):** Realiza una eliminación lógica del usuario, marcándolo como eliminado sin remover físicamente los datos de la base de datos. Los usuarios eliminados no aparecen en listados pero sus datos se conservan para auditoría.
8. **Salir:** Cierra la aplicación de manera ordenada.

El menú incluye validación de entrada de datos, manejo de excepciones para errores de formato y base de datos, y confirmaciones para operaciones críticas como eliminación o cambio de estado.

8. Conclusiones

El desarrollo de este Trabajo Final Integrador permitió consolidar los conocimientos adquiridos durante la cursada de Programación II, específicamente en:

- Implementación práctica de relaciones 1:1 unidireccionales tanto en modelo de objetos Java como en base de datos relacional, comprendiendo las diferencias y decisiones de diseño.
- Aplicación del patrón DAO para separar la lógica de acceso a datos, facilitando el mantenimiento y las pruebas del código.
- Uso de la capa Service para centralizar la lógica de negocio y gestionar transacciones ACID, garantizando la consistencia de datos.
- Implementación de arquitectura en capas con separación clara de responsabilidades, siguiendo principios SOLID.
- Manejo robusto de excepciones y validaciones en todas las capas del sistema.
- Aplicación de buenas prácticas de seguridad como hash de contraseñas, prevención de SQL Injection y control de accesos.
- Comprensión profunda de JDBC y gestión manual de conexiones, transacciones y mapeo objeto-relacional sin frameworks ORM.

8.1. Reflexión Personal

Este proyecto representó un desafío significativo que permitió integrar conceptos teóricos con implementación práctica. El proceso de diseño, implementación y pruebas reforzó la importancia de la planificación, la arquitectura limpia y las buenas prácticas de programación. La experiencia adquirida será invaluable para futuros proyectos profesionales.

9. Documentación Técnica

- Oracle. (2024). *Java SE 21 Documentation*. <https://docs.oracle.com/en/java/javase/21/>
- Oracle. (2024). *JDBC API Documentation*.
<https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>
- MySQL. (2024). *MySQL 8.0 Reference Manual*. <https://dev.mysql.com/doc/refman/8.0/en/>
- Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.
- Universidad Tecnológica Nacional. (2025). *Programación II - Material de Cátedra*. Consignas del TFI.

9.3. Uso de Inteligencia Artificial

Durante el desarrollo de este proyecto se utilizó asistencia de IA de la siguiente manera:

- **Claude (Anthropic)**: Consultas sobre mejores prácticas de Java, revisión de sintaxis SQL, sugerencias de estructura de código, generación de diagramas UML y asistencia en documentación.
- **Uso responsable**: La IA se utilizó como herramienta de consulta y aprendizaje, no para generar código completo. Todo el código fue escrito, comprendido y probado personalmente.
- **Aprendizaje**: Las sugerencias de IA fueron analizadas críticamente y adaptadas según las necesidades del proyecto y los requisitos académicos.
- **Transparencia**: Se declara abiertamente el uso de IA, cumpliendo con los principios de integridad académica.