



## **TRABAJO FINAL INTEGRADOR: Programación II**

### **Sistema de Gestión de Usuarios y Credenciales de Acceso**

#### **Alumnos:**

4822 - Basualdo Arcati Pablo - [pbasualdoarcati@gmail.com](mailto:pbasualdoarcati@gmail.com)

5721 - Calcatelli Renzo - [rcalcatelli@gmail.com](mailto:rcalcatelli@gmail.com)

7589 - Kaddarian Dante - [dantek91@yahoo.com](mailto: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"><li>• Diseño y creación de base de datos<ul style="list-style-type: none"><li>• Diseño de la arquitectura del sistema</li></ul></li><li>• Implementación de entidades y relaciones</li><li>• Desarrollo de capa DAO y Service</li><li>• Pruebas</li></ul>
Calcatelli Renzo	5721	Desarrollador Full-Stack Investigación	<ul style="list-style-type: none"><li>• Desarrollo de capa DAO y Service</li><li>• Diseño y creación de base de datos<ul style="list-style-type: none"><li>• Documentación</li><li>• Pruebas</li></ul></li><li>• Implementación de interfaz de usuario</li></ul>
Kaddarian Dante	7589	Desarrollador Full-Stack Tester	<ul style="list-style-type: none"><li>• Diseño y creación de base de datos</li><li>• Implementación de interfaz de usuario<ul style="list-style-type: none"><li>• Pruebas</li></ul></li><li>• Desarrollo de capa DAO y Service</li></ul>

**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
<b>Usuario</b>	<ul style="list-style-type: none"><li>• id (Long)</li><li>• username (String)</li><li>• email (String)</li><li>• nombreCompleto (String)</li><li>• activo (Boolean)</li><li>• eliminado (Boolean)</li><li>• fechaRegistro (LocalDateTime)</li><li>• credencialAcceso</li></ul>	Representa a una persona que utiliza el sistema
<b>CredencialAcceso</b>	<ul style="list-style-type: none"><li>• id (Long)</li><li>• usuarioid (Long)</li><li>• hashPassword (String)</li><li>• salt (String)</li><li>• ultimoCambio (LocalDateTime)</li><li>• requiereReset (Boolean)</li><li>• intentosFallidos (Integer)</li><li>• ultimoLogin (LocalDateTime)</li><li>• eliminado (Boolean)</li></ul>	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.

### Implementación en Java:

```
public class Usuario {
    private Long id;
    private boolean eliminado;
    private String username;
    private String email;
    private String nombreCompleto;
    private boolean activo;
    private LocalDateTime fechaRegistro;
    private CredencialAcceso credencialAcceso; // Relación unidireccional

    // Constructores, getters, setters...
}

public class CredencialAcceso {
    private Long id;
    private boolean eliminado;
    private Long usuarioId; // FK hacia usuario
    private String hashPassword;
    private String salt;
    private LocalDateTime ultimoCambio;
    private boolean requiereReset;
    private int intentosFallidos;
    private LocalDateTime ultimoLogin;
    // NO tiene referencia al objeto Usuario

    // Constructores, getters, setters...
}
```

## 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"> <li>IDs independientes</li> <li>Mayor flexibilidad <ul style="list-style-type: none"> <li>Más intuitiva</li> </ul> </li> <li>Mejor para migraciones</li> </ul>	<ul style="list-style-type: none"> <li>Garantía absoluta 1:1</li> <li>Ligeramente más eficiente</li> <li>Menos espacio en disco</li> </ul>
Desventajas	Requiere constraint UNIQUE adicional	<ul style="list-style-type: none"> <li>IDs acoplados</li> <li>Menos flexible</li> <li>Complejidad en inserciones</li> </ul>

**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.

#### Implementación en MySQL:

```
CREATE TABLE credencial_acceso (  
  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    usuario_id BIGINT NOT NULL UNIQUE, -- FK única garantiza 1:1  
    nombre_usuario VARCHAR(50) NOT NULL UNIQUE,  
    password_hash VARCHAR(64) NOT NULL,  
    salt VARCHAR(32) NOT NULL,  
    intentos_fallidos INT DEFAULT 0,  
    ultimo_intento DATETIME,  
    eliminado BOOLEAN DEFAULT FALSE,  
    FOREIGN KEY (usuario_id)  
        REFERENCES usuario(id)  
        ON DELETE CASCADE  
  
);
```

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

**Tabla usuario:**

```
CREATE TABLE usuario (  
  id BIGINT AUTO_INCREMENT PRIMARY KEY,  
  eliminado BOOLEAN NOT NULL DEFAULT FALSE,  
  username VARCHAR(50) NOT NULL,  
  email VARCHAR(100) NOT NULL,  
  nombre_completo VARCHAR(100) NOT NULL,  
  activo BOOLEAN NOT NULL DEFAULT TRUE,  
  fecha_registro DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
```

```
CONSTRAINT uq_usuario_username UNIQUE (username),
CONSTRAINT uq_usuario_email UNIQUE (email),

INDEX idx_usuario_username (username),
INDEX idx_usuario_email (email),
INDEX idx_usuario_activo (activo),
INDEX idx_usuario_eliminado (eliminado)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

## 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.

## Tabla credencial\_acceso:

```
CREATE TABLE credencial_acceso (
  id BIGINT AUTO_INCREMENT PRIMARY KEY,
  eliminado BOOLEAN NOT NULL DEFAULT FALSE,
  usuario_id BIGINT NOT NULL,
  hash_password VARCHAR(255) NOT NULL,
  salt VARCHAR(64),
  ultimo_cambio DATETIME,
  requiere_reset BOOLEAN NOT NULL DEFAULT FALSE,
  intentos_fallidos INT NOT NULL DEFAULT 0,
  ultimo_login DATETIME,

  CONSTRAINT fk_credencial_usuario
    FOREIGN KEY (usuario_id)
    REFERENCES usuario(id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,

  CONSTRAINT uq_credencial_usuario UNIQUE (usuario_id),
```

```
INDEX idx_credencial_usuario (usuario_id),  
INDEX idx_credencial_eliminado (eliminado)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

## 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, resets 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. Gestión de Transacciones

Las transacciones son fundamentales para mantener la consistencia de datos. El sistema implementa transacciones en la capa Service siguiendo este patrón:

```
public Usuario crearUsuarioConCredencial(
    Usuario usuario,
    CredencialAcceso credencial,
    String passwordTextoPlano
) {
    Connection conn = null;
    try {
        // 1. Obtener conexión
        conn = DatabaseConfig.getConnection();
        // 2. Iniciar transacción
        conn.setAutoCommit(false);
        // 3. Validar datos
        validarUsuario(usuario);
        validarCredencial(credencial);
        // 4. Crear usuario
        Usuario usuarioCreado = usuarioDao.crear(usuario, conn);
        // 5. Procesar contraseña
        String salt = generarSalt();
        String hash = hashPassword(passwordTextoPlano, salt);
        credencial.setSalt(salt);
        credencial.setPasswordHash(hash);
        // 6. Crear credencial vinculada
        credencial.setUsuarioId(usuarioCreado.getId());
        CredencialAcceso credCreada =
            credencialDao.crear(credencial, conn);
        // 7. Asociar en memoria
        usuarioCreado.setCredencialAcceso(credCreada);
        // 8. COMMIT si todo OK
        conn.commit();
        return usuarioCreado;
    } catch (Exception e) {
        // 9. ROLLBACK ante cualquier error
        if (conn != null) {
            try {
                conn.rollback();
            } catch (SQLException ex) {
```

```
        // log error
    }
}
throw new RuntimeException("Error: " + e.getMessage());
} finally {
    // 10. Restaurar autoCommit y cerrar
    if (conn != null) {
        try {
            conn.setAutoCommit(true);

            conn.close();
        } catch (SQLException e) {
            // log error
        }
    }
}
}
```

## 5.4. 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

## 5.5. 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 y Reglas de Negocio

El sistema implementa múltiples niveles de validación para garantizar la integridad de los datos.

### 6.1. 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.

## 6.2. 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.

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

### 7.2. Consultas SQL Útiles

Durante las pruebas, se utilizaron las siguientes consultas para verificar el estado de la base de datos:

**Listar usuarios con credenciales:**

```
SELECT u.id, u.nombre, u.apellido, u.email,
       c.nombre_usuario, c.intentos_fallidos
FROM usuario u
LEFT JOIN credencial_acceso c ON u.id = c.usuario_id
WHERE u.eliminado = FALSE;
Verificar relación 1:1:
SELECT u.id, COUNT(c.id) as cant_credenciales
FROM usuario u
LEFT JOIN credencial_acceso c ON u.id = c.usuario_id
GROUP BY u.id
HAVING cant_credenciales > 1;
```

**Usuarios bloqueados:**

```
SELECT u.nombre, u.email, c.intentos_fallidos,
       c.ultimo_intento
FROM usuario u
JOIN credencial_acceso c ON u.id = c.usuario_id
WHERE c.intentos_fallidos >= 5;
```

**Auditoría de eliminados:**

```
SELECT 'Usuario' as tipo, id, nombre as detalle,
       'Eliminado' as estado
FROM usuario WHERE eliminado = TRUE
UNION ALL
SELECT 'Credencial', id, nombre_usuario, 'Eliminado'
FROM credencial_acceso WHERE eliminado = TRUE;
```

## 7.3. 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.

-

```
=====
                          MENÚ PRINCIPAL
=====
1. Crear Usuario con Credencial
2. Buscar Usuario por ID
3. Buscar Usuario por Username
4. Listar Todos los Usuarios
5. Actualizar Usuario
6. Activar/Desactivar Usuario
7. Eliminar Usuario (Baja Lógica)
8. Salir
=====
Selecione una opción:
```

## 8. Conclusiones y Mejoras Futuras

### 8.1. 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.

## 8.2. Logros del Proyecto

- **Cumplimiento total de requisitos:** El proyecto cumple con el 100% de las consignas del TFI.
- **Código limpio y documentado:** Todas las clases incluyen comentarios explicativos y javadoc.
- **Seguridad robusta:** Implementación de SHA-256 + salt, PreparedStatement y validaciones exhaustivas.
- **Funcionalidad completa:** CRUD completo para ambas entidades, búsquedas, validación de login y más.
- **Manejo profesional de errores:** Try-catch en todas las operaciones críticas con mensajes descriptivos.

## 8.3. Mejoras Futuras

Si bien el proyecto cumple con todos los requisitos, se identificaron las siguientes mejoras potenciales para versiones futuras:

- **Interfaz gráfica (GUI):** Migrar de consola a JavaFX o Swing para mejor experiencia de usuario.
- **Pool de conexiones:** Implementar HikariCP o similar para gestión eficiente de conexiones en producción.
- **Logging profesional:** Incorporar SLF4J + Logback para registro de eventos y debugging.
- **Sistema de roles y permisos:** Extender el modelo para incluir roles (admin, usuario, guest) y permisos granulares.
- **Autenticación multifactor (MFA):** Agregar segundo factor de autenticación vía email o TOTP.
- **API RESTful:** Exponer funcionalidades mediante REST API con Spring Boot o similar.
- **Testing automatizado:** Implementar JUnit 5 para pruebas unitarias y de integración.
- **Recuperación de contraseña:** Sistema de reset de contraseña vía email con tokens temporales.
- **Auditoría completa:** Tabla de logs con todas las operaciones (quién, qué, cuándo).
- **Migración a ORM:** Evaluar JPA/Hibernate para reducir código boilerplate (manteniendo conocimiento de JDBC).
- **Validación de contraseñas comunes:** Verificar contra lista de contraseñas débiles conocidas.
- **Sesiones de usuario:** Implementar tokens de sesión con expiración automática.

## 8.4. 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. Referencias y Herramientas

### 9.1. 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.

### 9.2. Herramientas Utilizadas

Categoría	Herramienta	Versión	Propósito
Lenguaje	Java SE	21 LTS	Desarrollo de aplicación
Base de Datos	MySQL	8.0	Persistencia de datos
Driver JDBC	MySQL Connector/J	8.0.33	Conexión Java-MySQL
IDE	IntelliJ IDEA / Eclipse	Última	Desarrollo y debugging
Control de versiones	Git	2.x	Gestión de código fuente
Documentación	PlantUML / Mermaid	Última	Diagramas UML
Cliente BD	MySQL Workbench	8.0	Administración de BD

### 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.