INFORME: TFI PROGRAMACIÓN 2

Sistema de Gestión de Usuarios y Credenciales

Asignatura: Programación II

Fecha: 20 de Noviembre de 2025

Tecnologías: Java, MySQL, JDBC, Patrón DAO

Integrantes y Roles

A continuación, se detallan los miembros del equipo y las responsabilidades durante el desarrollo del proyecto:

Integrante	Descripción de Tareas
Saul Hillar	Diseño del diagrama de clases, definición de paquetes, lógica de negocio, transacciones (UsuarioServiceImpl), seguridad (Hashing) y validaciones.
Enzo Chavez	Implementación de la capa de acceso a datos (UsuarioDAOImpl, CredencialAccesoDAOImpl)
Michael Chiappone	Desarrollo del AppMenu, pruebas unitarias manuales y control de flujo de la aplicación.

Elección del Dominio y Justificación

Para este Trabajo Práctico, hemos seleccionado la relación de dominio:

Usuario -> CredencialAcceso.

Justificación:

La separación entre la entidad Usuario (datos de perfil, contacto y estado) y CredencialAcceso (datos de autenticación sensibles) es una práctica fundamental en el diseño de software seguro y escalable.

1. **Seguridad:** Permite aislar la información crítica (hash de contraseñas y salt) en una estructura dedicada, facilitando políticas de seguridad específicas sin exponer datos personales del usuario.
2. **Escalabilidad:** Si en el futuro se requiere cambiar el método de autenticación (ej. OAuth, Tokens), solo se modifica la entidad CredencialAcceso sin alterar la estructura base del Usuario.
3. **Coherencia con la consigna:** Representa una relación 1 a 1 fuerte. Un usuario solo puede tener un conjunto de credenciales activas, y esas credenciales pertenecen única y exclusivamente a ese usuario.

Diseño y Modelado (UML)

El diseño del sistema se basa en una orientación a objetos pura, utilizando herencia y composición para estructurar las entidades.

3.1. Descripción del Modelo de Clases

Se implementó una clase abstracta Base que centraliza los atributos comunes de persistencia: id (identificador único) y eliminado (bandera para baja lógica).

- **Usuario (Entidad A):** Hereda de Base. Contiene atributos de negocio como username, email, activo y fechaRegistro. Es la entidad "Dueña" de la relación, manteniendo una referencia a CredencialAcceso.
- **CredencialAcceso (Entidad B):** Hereda de Base. Contiene la información de seguridad: hashPassword, salt, ultimoCambio y userId. Aunque en la base de datos esta tabla contiene la *Foreign Key*, en el modelo de objetos Java, la navegación principal se diseñó desde Usuario.

3.2. Relación 1 a 1 Unidireccional

Se respetó la restricción de unidireccionalidad requerida. La clase Usuario posee un atributo private CredencialAcceso credencial;, lo que permite obtener las credenciales desde el usuario. La clase CredencialAcceso no posee un objeto Usuario como atributo, rompiendo el ciclo de dependencia en memoria.

Arquitectura por Capas

El proyecto se estructuró siguiendo una arquitectura en capas estricta para garantizar la separación de responsabilidades y el código limpio:

1. **Paquete config:**
   - Contiene la clase DatabaseConnectionPool. Se implementó utilizando la librería **HikariCP** para manejar un pool de conexiones eficiente, mejorando el rendimiento frente a la apertura/cierre manual de conexiones.
   - Lee las credenciales desde un archivo db.properties externo para no hardcodear datos sensibles en el código.
2. **Paquete models (Entities):**
   - Contiene los Objects que representan las tablas de la base de datos. Incluye Usuario, CredencialAcceso y la clase padre Base.
3. **Paquete dao (Data Access Object):**
   - Define la interfaz GenericDAO<T> para estandarizar operaciones CRUD.
   - Las implementaciones UsuarioDAOImpl y CredencialAccesoDAOImpl manejan las sentencias SQL utilizando PreparedStatement para prevenir inyección SQL.
   - **Característica Clave:** Los métodos DAO están sobrecargados para aceptar un objeto Connection externo, permitiendo que la capa de servicio controle la transacción.
4. **Paquete service:**
   - Actúa como intermediario entre la vista y el DAO. Aquí reside la lógica transaccional.
   - UsuarioServiceImpl orquesta el registro compuesto (Usuario + Credencial).
   - Incluye el uso de **BCrypt** para el hashing de contraseñas antes de enviarlas al DAO.
5. **Paquete main:**
   - Contiene AppMenu, que gestiona la interacción con el usuario vía consola, validación de entradas y captura de excepciones.

Persistencia y Base de Datos

La persistencia se realiza sobre una base de datos MySQL (gestion_usuarios_2).

Estructura de Tablas

- **Tabla usuario:** PK id (AUTO_INCREMENT), username, email, eliminado (BIT).
- **Tabla credencial_acceso:** PK id, user_id (FK UNIQUE), hash_password, salt.
  - La columna user_id tiene una restricción UNIQUE para garantizar a nivel de base de datos que un usuario no pueda tener más de una credencial (relación 1 a 1).
  - Se configuró ON DELETE CASCADE para la integridad referencial.

Manejo de Transacciones (ACID)

Uno de los puntos más fuertes del desarrollo es el manejo programático de transacciones en UsuarioServiceImpl.registrarUsuario. El flujo implementado es:

1. Se obtiene una conexión del pool: conn = DatabaseConnectionPool.getConnection().
2. Se desactiva el autocomit: conn.setAutoCommit(false).
3. **Paso A:** Se inserta el objeto Usuario. El DAO recupera el ID generado por la base de datos y lo asigna al objeto.
4. **Paso B:** Se crea el objeto CredencialAcceso, se le asigna el ID del usuario recién creado y se inserta en su tabla correspondiente.
5. **Commit:** Si ambos pasos son exitosos, se ejecuta conn.commit().
6. **Rollback:** Si ocurre cualquier excepción (ej. fallo de red, validación SQL), el bloque catch ejecuta conn.rollback(), asegurando que no quede un usuario registrado sin credenciales.

```java
@Override
// Metodos sobrecargados, para poder manear las transacciones
public void insert(Connection conn, Usuario user) throws SQLException {

    try (PreparedStatement stmt = conn.prepareStatement(INSERT_SQL, Statement.RETURN_GENERATED_KEYS)) {

        stmt.setString(1, user.getUsername());
        stmt.setString(2, user.getEmail());

        int rowsAffected = stmt.executeUpdate();
        // si no hay, no lo insertamos
        if (rowsAffected == 0) {
            throw new SQLException("No se pudo insertar el usuario.");
        }

        try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
            if (generatedKeys.next()) {
                int nuevoId = generatedKeys.getInt(1);
                user.setId(nuevoId);
            } else {
                throw new SQLException("Se inserto el usuario, pero no se pudo recuperar el ID.");
            }
        }
```

Validaciones y Reglas de Negocio

Se implementaron validaciones defensivas tanto en la capa de Servicio como en la Base de Datos:

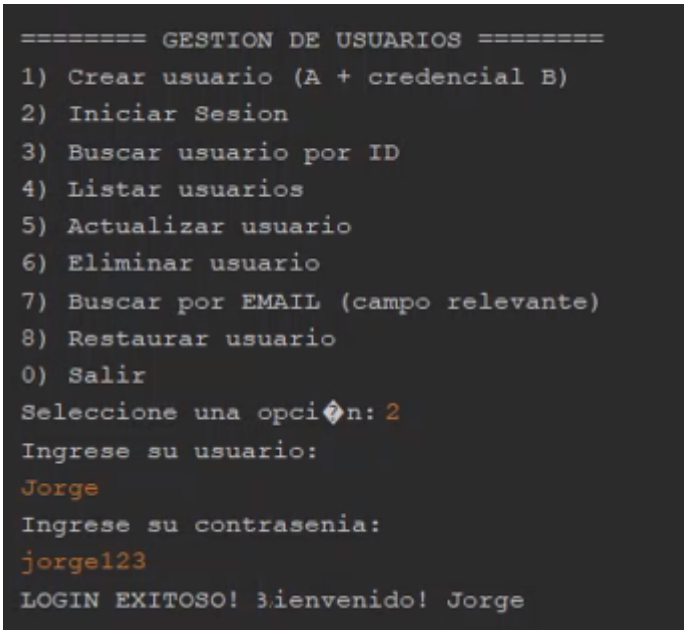
1. **Hashed Passwords:** No se almacenan contraseñas en texto plano. Se utiliza la librería BCrypt para generar un *salt* aleatorio y hashear la contraseña. Esto se verifica en el método login comparando el texto plano ingresado con el hash almacenado.
2. **Campos Obligatorios:** El servicio valida que username, email y password no sean nulos ni vacíos antes de llamar al DAO.
3. **Validación de Email:** Se incluye una verificación básica de formato (debe contener '@') en findByEmail.

4. **Baja Lógica:** Al eliminar un usuario, no se borra el registro físico (DELETE), sino que se ejecuta un UPDATE seteando el campo eliminado = TRUE. Las consultas de búsqueda (SELECT) filtran automáticamente por eliminado = FALSE para ocultar estos registros.

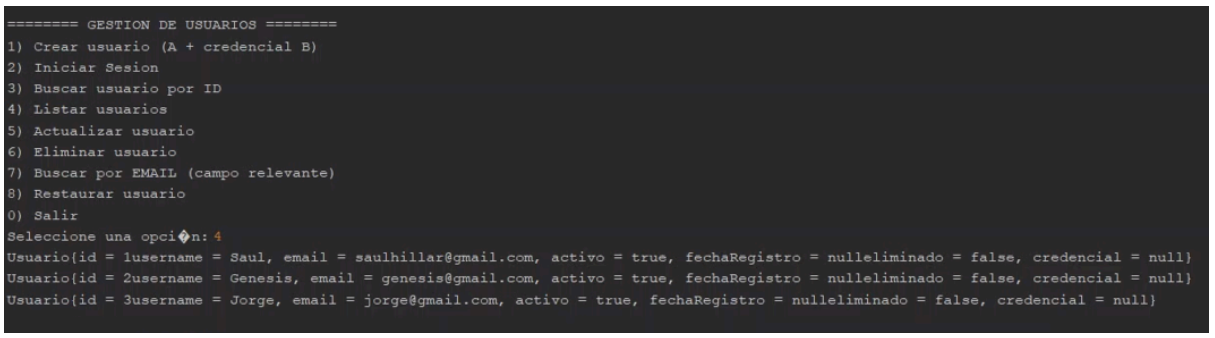
Pruebas Realizadas

Se llevaron a cabo pruebas manuales exhaustivas mediante el menú de consola (AppMenu.java).

- **Caso de Prueba 1: Inicio de Sesion luego de Registro de Usuario**



- **Caso de Prueba 2: Listar Usuarios**



- **Caso de Prueba 3: Eliminacion de Usuario**

```
======== GESTION DE USUARIOS ========
1) Crear usuario (A + credencial B)
2) Iniciar Sesion
3) Buscar usuario por ID
4) Listar usuarios
5) Actualizar usuario
6) Eliminar usuario
7) Buscar por EMAIL (campo relevante)
8) Restaurar usuario
0) Salir
Seleccione una opci□n: 6
ID a eliminar: 3
Usuario eliminado logicamente.
```

	id	username	email	fecha_registro	activo	eliminado
	1	Saul	saulhillar@gmail.com	2025-11-17 14:14:59	1	0
	2	Genesis	genesis@gmail.com	2025-11-17 16:19:02	1	0
▶	3	Jorge	jorge@gmail.com	2025-11-20 21:52:09	1	1
*	NULL	NULL	NULL	NULL	NULL	NULL

- **Caso de Prueba 4: Recuperación del Usuariio (Baja Lógica)**

```
======== GESTION DE USUARIOS ========
1) Crear usuario (A + credencial B)
2) Iniciar Sesion
3) Buscar usuario por ID
4) Listar usuarios
5) Actualizar usuario
6) Eliminar usuario
7) Buscar por EMAIL (campo relevante)
8) Restaurar usuario
0) Salir
Seleccione una opci□n: 8
ID a restaurar: 3
Usuario restaurado.
```

	id	username	email	fecha_registro	activo	eliminado
	1	Saul	saulhillar@gmail.com	2025-11-17 14:14:59	1	0
	2	Genesis	genesis@gmail.com	2025-11-17 16:19:02	1	0
▶	3	Jorge	jorge@gmail.com	2025-11-20 21:52:09	1	0
*	NULL	NULL	NULL	NULL	NULL	NULL

Conclusiones y Mejoras Futuras

El desarrollo de este Trabajo Práctico permitió consolidar los conocimientos sobre JDBC y el patrón de diseño DAO. Logramos desacoplar exitosamente la lógica de negocio del acceso a datos, implementando una relación 1 a 1 robusta con integridad transaccional y todas las capas.

Puntos Fuertes:

- Uso de **HikariCP** para la gestión profesional de conexiones.
- Seguridad implementada con **BCrypt**.
- Manejo correcto de transacciones (commit/rollback) en la capa de servicios.

Mejoras Futuras:

1. **Interfaz Gráfica:** Migrar la consola a una interfaz Swing o Web (Spring Boot).
2. **Recuperación de Contraseña:** Implementar el flujo de envío de email real, utilizando el campo requireReset del modelo CredencialAcceso.
3. **Roles y Permisos:** Extender el modelo para soportar roles (Admin/User) y restringir opciones del menú.