Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Trabajo Integrador N°1

Programación II

Docente
David Lopez

Estudiantes
Monzon, Martin
Olivero, Christian
Pichulman, Miguel
Rodriguez, Joaquin

17 de noviembre de 2025

Índice

Introducción

El proyecto consiste en el desarrollo de un sistema de gestión de usuarios y credenciales de acceso, que permita realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar), garantizando una arquitectura modular, segura y escalable mediante una relación **1→1**: **Usuario y Credencial Acceso**.

El objetivo es implementar un sistema de gestión de usuarios y credenciales aplicando POO, persistencia con JDBC/MySQL y una arquitectura en capas (Servicios, DAO, Modelo). Se desarrollarán operaciones CRUD completas, respetando la relación 1 a 1 entre entidades, y se garantizará la integridad de los datos mediante validaciones de negocio y gestión explícita de transacciones (commit/rollback). Además de nuestras inclinaciones personales hacia el proyecto, consideramos que la gestión de usuarios y credenciales de acceso constituye una necesidad transversal en prácticamente cualquier sistema informático moderno. A través de este trabajo pudimos comprobar que es posible aplicar de manera práctica los conceptos de Programación Orientada a Objetos, tales como el encapsulamiento, las asociaciones y las validaciones.

Diseño del Sistema

Explicación de la arquitectura en capas (Main, Service, DAO, Models)

Config/: Capa de Configuración
- DatabaseConnection.java: Gestión de conexiones JDBC. Inicialización estática con validación de parámetros de conexión. Evita duplicación de código y asegura cierre correcto

Entities/
- Base.java: Clase abstracta con atributos comunes (id, eliminado), provee de una herencia estandar..
- Usuario.java: Entidad principal del sistema. Campos: username, email, activo, fechaRegistro, relación 1→1 con CredencialAcceso.
- CredencialAcceso.java: Entidad secundaria. Campos: hashPassword, salt, ultimoCambio, requiereReset. Asociada a Usuario.

Dao/
- GenericDAO<T>: Interfaz genérica con operaciones CRUD (crear, leer, leerTodos, actualizar, eliminar).
- UsuarioDAO: Implementación DAO para Usuario. Queries con LEFT JOIN para incluir CredencialAcceso.
- CredencialAccesoDAO: Implementación DAO para CredencialAcceso. Manejo de inserción, actualización y baja lógica.
Uso de PreparedStatement. Evita SQL Injection y mejorar la performance.

Service/
- GenericService<T>: Interfaz genérica para servicios. Define operaciones CRUD con validaciones y transacciones.
- UsuarioService: Implementación de reglas de negocio para Usuario. Valida username y email (regex). Orquesta operaciones transaccionales compuestas: inserción/ actualización/ eliminación de Usuario junto con su CredencialAcceso.
- CredencialAccesoService: Implementación de reglas de negocio para CredencialAcceso. Valida campos obligatorios (hashPassword, salt). Gestiona cambios de contraseña y bajas lógicas.

Main/
- Main.java: Punto de entrada de la aplicación. Inicializa AppMenu.
- AppMenu.java: Orquestador del ciclo de vida del menú. Inyecta dependencias (UsuarioService, CredencialAccesoService) en MenuHandler.
- MenuHandler.java: Controlador de operaciones CRUD. Captura entradas del usuario y delega en los servicios.
    Opción 1: Crear Usuario + Credencial.
    Opción 2: Listar Usuarios.
    Opción 3: Actualizar Usuario.

Opción 4: Eliminar Usuario + Credencial.
Opción 5: Buscar Usuario por ID.
Opción 6: Buscar Usuario por Username.
Opción 7: Actualizar Contraseña.

- MenuDisplay.java: Clase utilitaria para mostrar el menú principal en consola.
- PasswordUtil.java: Utilidad para simular generación de hash y salt (solo con fines académicos).

Principios aplicados (POO, SOLID, DAO Pattern, Service Layer)

DRY (Don’t Repeat Yourself): centralización de SQL en constantes y reutilización de helpers (setParameters, mapearUsuarioCompleto).

Fail Fast: validaciones tempranas en Services y DAOs para evitar operaciones inválidas.

Transaction Script: los Services actúan como scripts transaccionales que orquestan múltiples DAOs.

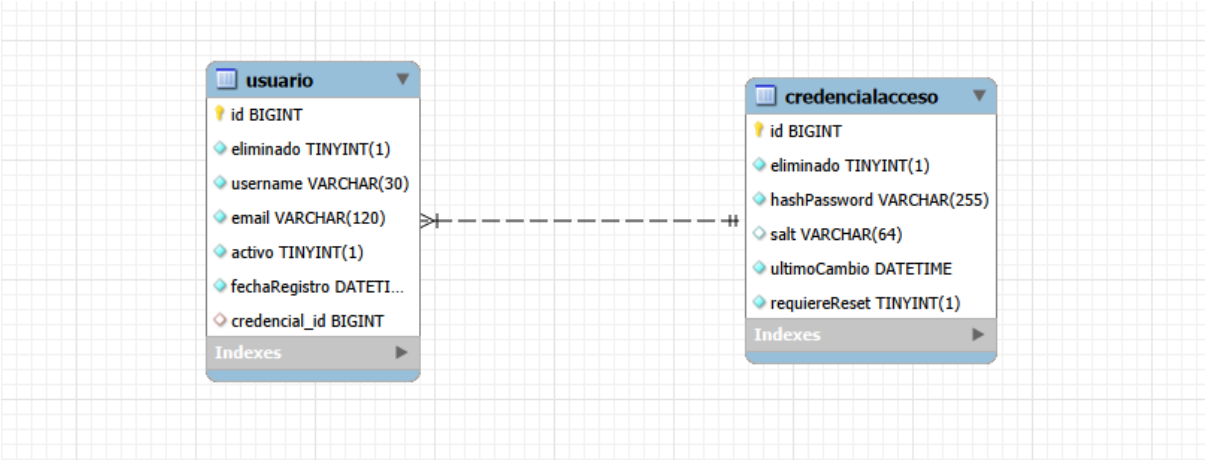
Layered Architecture: separación estricta entre presentación, servicio, persistencia y modelo.

Este diseño permite una buena escalabilidad ya que se pueden agregar nuevas entidades, roles o permisos sin romper la estructura. Las capas están aisladas lo que facilita pruebas de depuración.

Modelo de Datos y UML

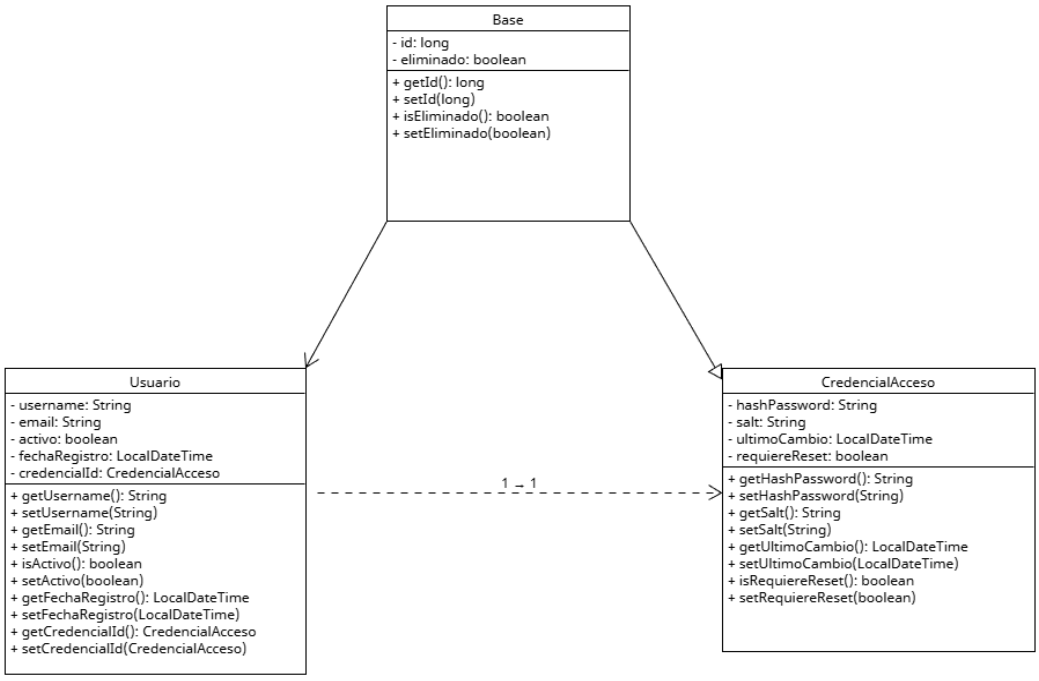
El sistema implementa un modelo de datos simple compuesto por dos entidades Usuario y CredencialAcceso, relacionadas mediante una asociación 1→1 obligatoria.
Este diseño responde a los requisitos del Trabajo Práctico Integrador, garantizando consistencia en la administración de la información vinculada a la autenticación del usuario.

Diagrama Entidad-Relación **(tablas usuarios y credenciales_acceso)**



- Un usuario no puede existir sin una credencial.
- Una credencial no puede estar asociada a más de un usuario.
- La integridad se asegura tanto a nivel de BD como en las transacciones de UsuarioService.

Diagrama UML – Modelo de Entidades



El diagrama UML representa la estructura de clases del sistema. La clase Base actúa como superclase común y provee los atributos compartidos id y eliminado, de los cuales heredan las entidades Usuario y CredencialAcceso. La relación entre Usuario y CredencialAcceso es 1→1 unidireccional, un usuario posee exactamente una credencial, mientras que la credencial no mantiene referencia inversa.

Las clases incluyen sus atributos privados y los métodos públicos correspondientes, aplicando principios de encapsulamiento, herencia y buena organización orientada a objetos.

Fragmento del script SQL

```
CREATE TABLE CredencialAcceso (

    id BIGINT PRIMARY KEY AUTO_INCREMENT,

    eliminado BOOLEAN NOT NULL DEFAULT FALSE,

    hashPassword VARCHAR(255) NOT NULL,

    salt VARCHAR(64),

    ultimoCambio DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP,

    requiereReset BOOLEAN NOT NULL DEFAULT FALSE

);



CREATE TABLE Usuario (

    id BIGINT PRIMARY KEY AUTO_INCREMENT,

    eliminado BOOLEAN NOT NULL DEFAULT FALSE,

    username VARCHAR(30) NOT NULL UNIQUE,

    email VARCHAR(120) NOT NULL UNIQUE,

    activo BOOLEAN NOT NULL DEFAULT TRUE,

    fechaRegistro DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP,

    credencial_id BIGINT UNIQUE,

    CONSTRAINT fk_credencial
```

```
        FOREIGN KEY (credencial_id)
        REFERENCES CredencialAcceso(id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Funcionalidades y Reglas de Negocio

El sistema implementa un conjunto completo de operaciones CRUD y búsquedas para las entidades Usuario y CredencialAcceso, respetando los requerimientos del Trabajo Práctico Integrador y la arquitectura en capas definida (DAO → Service → Main / Menu).

Validaciones importantes

Crear Usuario (Alta compuesta: Usuario + CredencialAcceso)

- El sistema permite registrar un nuevo usuario junto a su credencial.
- La operación es transaccional:
- Se crea primero la credencial
- Luego el usuario asociado.
- Si ocurre un error, se hace rollback y nada queda creado.

Listar Usuarios

- Muestra todos los usuarios activos (no eliminados lógicamente).
- Incluye la credencial asociada (JOIN automático desde el DAO).

Buscar Usuario por ID

- Permite obtener un usuario específico con su credencial.
- Filtra automáticamente registros eliminados.

Buscar Usuario por Nombre

- Implementada como requerimiento del TPI.
- Devuelve el usuario coincidente si existe.

Actualizar Usuario

- Permite modificar datos del usuario (email, username, fecha, estado).
- Si el usuario contiene cambios en su credencial, también se actualiza la credencial en la misma transacción.

Eliminar Usuario

- El usuario y su CredencialAcceso se eliminan lógicamente, no físicamente.
- La operación es transaccional, garantizando integridad del vínculo 1→1.

Actualizar Contraseña (Credencial)

- Permite modificar el hash y salt simulados.
- Actualiza el campo ultimoCambio automáticamente.

Reglas de Negocio Implementadas

se aplican en la capa **Service** antes de llamar a los DAO:

Validaciones de Usuario

- username obligatorio y máximo de 30 caracteres.
- email obligatorio y con formato válido.
- Un usuario debe tener siempre una credencial asociada.
- No pueden existir duplicados de email o username

Validaciones de CredencialAcceso

- hashPassword obligatorio.
- salt obligatorio.
- La fecha de último cambio se actualiza al modificar la contraseña.

Regla 1→1 entre Usuario y CredencialAcceso

- credencial_id UNIQUE en base de datos.
- Transacciones en UsuarioService.
- Baja lógica compuesta (usuario + credencial).

Manejo de Errores y Consultas Inválidas

- Captura de NumberFormatException en el menú.
- Verificación de ID válido.
- Verificación de existencia de usuario antes de eliminar.

```
-----------------------------------------------
0. Salir
Ingrese una opcion: 7
Ingrese el ID del usuario cuya contrase�a desea cambiar: 2
Ingrese la NUEVA contrase�a para MartinBOX: river
Exito: Contrase�a actualizada para el usuario ID: 2

========= GESTION DE USUARIOS (TFI) =========
--- CRUD Usuario ---
1. Crear Usuario (Alta Usuario y Credencial)
2. Listar Usuarios
3. Actualizar Usuario (Datos personales)
4. Eliminar Usuario (Baja logica Usuario y Credencial)
--- Busquedas ---
5. Buscar Usuario por ID
6. Buscar Usuario por Username (Busqueda especifica)
--- CRUD Credencial ---
7. Actualizar Contrase�a (Requiere Reset)
-----------------------------------------------
0. Salir
Ingrese una opcion:
```

Transacciones y Validaciones

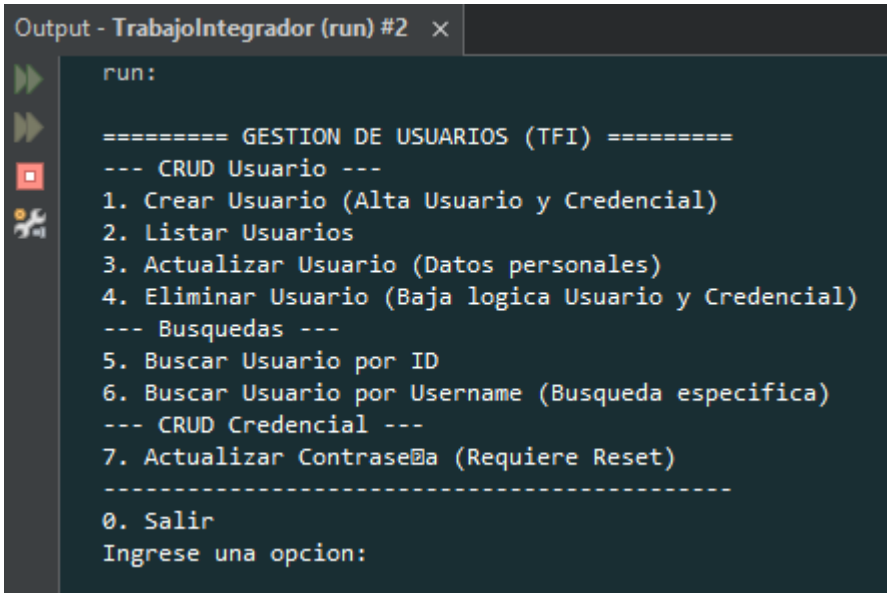
El sistema maneja transacciones y validaciones para mantener la integridad entre Usuario y CredencialAcceso, respetando la relación 1:1. Todas las operaciones que afectan a ambas entidades se ejecutan dentro de una transacción manual: se desactiva el autocommit, se realizan las operaciones y se confirma con commit(), o si algo falla se hace rollback(). Esto evita inconsistencias y garantiza que nunca quede un usuario sin credencial o una credencial huérfana.

En el alta de usuario, primero se crea la credencial y luego el usuario usando ese ID; si algo falla se revierte todo. En la actualización, primero se valida y luego se actualiza el usuario; si también hay cambios en la credencial, se actualiza dentro de la misma transacción. Si cualquier update falla, se deshace todo. En la eliminación lógica, se cargan usuario y credencial con la misma conexión, se marcan ambos como eliminados y se confirma; si ocurre un error, rollback. Así se mantiene la integridad y no se pierden datos físicamente.

Las validaciones se aplican antes de llegar a los DAO. Para Usuario: username obligatorio (sin espacios, máx. 30), email obligatorio con formato válido, unicidad de username y email, existencia de una credencial asociada y eliminación sólo lógica. Para CredencialAcceso: hashPassword y salt obligatorios, actualización de ultimoCambio y control con requiereReset.

El sistema también maneja errores comunes. En caso de claves duplicadas, como “Duplicate entry”, el Service devuelve mensajes claros (“El username ya existe”, “El email ya existe”). También se controlan errores de formato, IDs inválidos y campos vacíos para evitar caídas del programa.

Pruebas y Resultados



- Casos de prueba (crear, listar, eliminar, rollback).

  Crear Usuario:

```
Ingrese una opcion: 1
--- Creando Nuevo Usuario ---
Username: joaquin
Email: joaquin@tupad.com
Contrase@a: cordoba
? @xito: Usuario 'joaquin' creado con ID: 1
   (Credencial asociada con ID: 1)
```

  Listar Usuarios

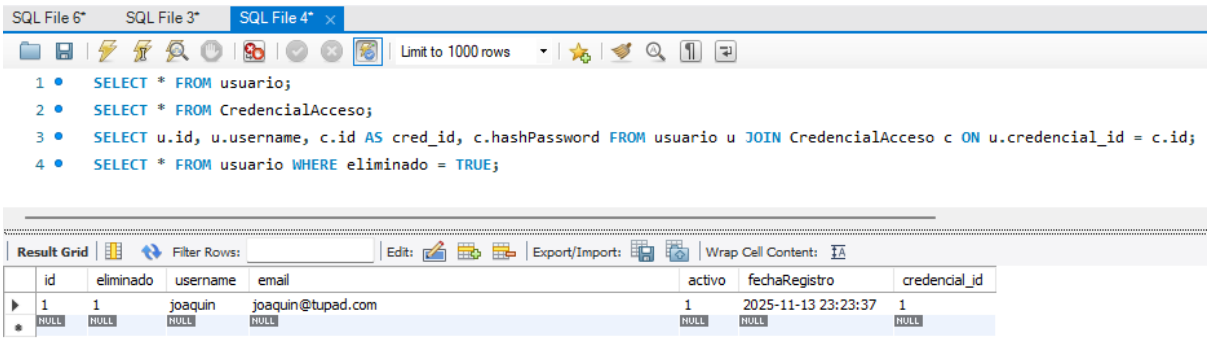
```
Ingrese una opcion: 2
--- Listando Todos los Usuarios ---
--------------------
ID: 1 | Username: joaquin
Email: joaquin@tupad.com | Activo: true
   -> Credencial ID: 1 | Requiere Reset: false
--------------------
ID: 2 | Username: martin
Email: martin@tupad.com | Activo: true
   -> Credencial ID: 2 | Requiere Reset: false
--------------------
ID: 3 | Username: christian
Email: christian@tupad.com | Activo: true
   -> Credencial ID: 3 | Requiere Reset: false
--------------------
ID: 4 | Username: miguel
Email: miguel@tupad.com | Activo: true
   -> Credencial ID: 4 | Requiere Reset: false
```

Eliminar Usuario

```
Ingrese una opcion: 4
Ingrese el ID del usuario a eliminar: 1
¿Esta seguro que desea eliminar al usuario con ID 1? (s/n): s
Exito: Usuario con ID 1 eliminado logicamente

========= GESTION DE USUARIOS (TFI) =========
--- CRUD Usuario ---
1. Crear Usuario (Alta Usuario y Credencial)
2. Listar Usuarios
3. Actualizar Usuario (Datos personales)
4. Eliminar Usuario (Baja logica Usuario y Credencial)
--- Busquedas ---
5. Buscar Usuario por ID
6. Buscar Usuario por Username (Busqueda especifica)
--- CRUD Credencial ---
7. Actualizar Contrase¿a (Requiere Reset)
-----------------------------------------------
0. Salir
Ingrese una opcion: 2
--- Listando Todos los Usuarios ---
--------------------
ID: 2 | Username: martin
Email: martin@tupad.com | Activo: true
   -> Credencial ID: 2 | Requiere Reset: false
--------------------
ID: 3 | Username: christian
Email: christian@tupad.com | Activo: true
   -> Credencial ID: 3 | Requiere Reset: false
--------------------
ID: 4 | Username: miguel
Email: miguel@tupad.com | Activo: true
   -> Credencial ID: 4 | Requiere Reset: false
```

Baja lógica (eliminado=1)



Búsqueda específica (por username) y manejo de errores

```
Ingrese una opcion: 6
Ingrese el Username del usuario a buscar: joaquin
Error: No se encontro un usuario con Username: 'joaquin'
```

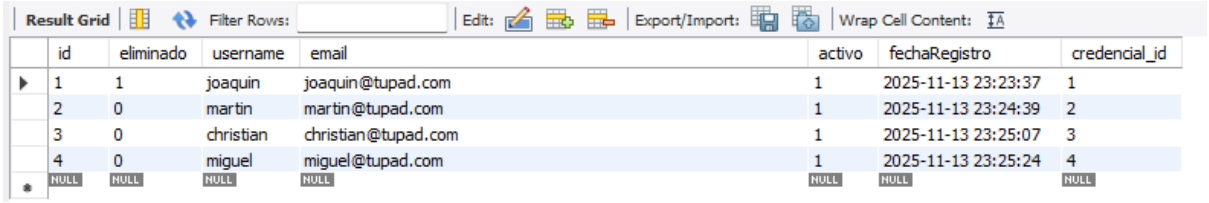
```
Ingrese una opcion: 6
Ingrese el Username del usuario a buscar: martin
--- Usuario Encontrado ---
ID: 2 | Username: martin
Email: martin@tupad.com | Activo: true
```

Entrada invalida

```
========= GESTION DE USUARIOS (TFI) =========
--- CRUD Usuario ---
1. Crear Usuario (Alta Usuario y Credencial)
2. Listar Usuarios
3. Actualizar Usuario (Datos personales)
4. Eliminar Usuario (Baja logica Usuario y Credencial)
--- Busquedas ---
5. Buscar Usuario por ID
6. Buscar Usuario por Username (Busqueda especifica)
--- CRUD Credencial ---
7. Actualizar Contrase@a (Requiere Reset)
-----------------------------------------
0. Salir
Ingrese una opcion: abc
Error: Entrada invalida. Por favor, ingrese solo numeros
```

Violacion de unicidad - rollback- (intentar crear un usuario con un username ya existente)

```
Ingrese una opcion: 1
--- Creando Nuevo Usuario ---
Username: miguel
Email: otromiguel@utn.com
Contrase@a: otromendoza
? Error de Base de Datos: Error: El 'username' ya existe.
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	eliminado	username	email	activo	fechaRegistro	credencial_id
1	1	joaquin	joaquin@tupad.com	1	2025-11-13 23:23:37	1
2	0	martin	martin@tupad.com	1	2025-11-13 23:24:39	2
3	0	christian	christian@tupad.com	1	2025-11-13 23:25:07	3
4	0	miguel	miguel@tupad.com	1	2025-11-13 23:25:24	4
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Conclusiones

Este proyecto nos permitió integrar y aplicar todos los conocimientos adquiridos a lo largo de la materia, resultando una experiencia muy enriquecedora para nuestro aprendizaje. Durante el desarrollo del trabajo pudimos poner en práctica distintas competencias importantes, como:

- Diseñar un sistema con una arquitectura ordenada y coherente.
- Aplicar patrones de diseño adecuados a cada problema.
- Manejar recursos, validaciones y excepciones de forma correcta.
- Trabajar en equipo coordinando ideas, uniendo código y resolviendo conflictos.

En general, el trabajo nos ayudó a entender cómo se combinan todos estos conceptos en un desarrollo real.

Consideramos que en futuras versiones podríamos mejorar la organización del tiempo y la coordinación entre integrantes. Al ser nuestro primer proyecto con un enfoque más profesional, hubo etapas que llevaron más tiempo del esperado.

De todas formas, este proceso nos dejó mejor preparados para enfrentar trabajos futuros con mayor eficiencia, seguridad y planificación.

Fuentes y Herramientas

- Java 17, MySQL MySQL Workbench, MySQL Connector/J, Apache NetBeans 27,
- IA y herramientas: ChatGPT, UMletino, documentación Java y MySQL, material de clase.
- Repositorio GitHub

Anexo

Historias de Usuario

HU-001 – Crear Usuario con su Credencial

Como administrador del sistema, quiero registrar un nuevo usuario junto con su credencial de acceso para que pueda quedar almacenado en la base de datos y disponible para el sistema.

Criterios de aceptación:

- Se debe ingresar username, email y contraseña (hash simulada).
- El email y el username no pueden repetirse.
- La credencial se crea primero, luego el usuario (transacción completa).
- Si falla uno, se hace rollback.

HU-002 – Listar Todos los Usuarios Activos

Como administrador, quiero ver un listado de usuarios no eliminados para consultar su información básica y su credencial asociada.

Criterios de aceptación:

- Solo se muestran usuarios con eliminado = FALSE.
- Se muestra username, email, estado y datos de la credencial.
- Si no hay usuarios, mostrar mensaje correspondiente.

HU-003 – Buscar Usuario por ID

Como administrador quiero buscar un usuario por su ID, para obtener rápidamente su información completa.

Criterios de aceptación:

- Si el ID existe y no está eliminado, se muestra toda la información.
- Si no existe, se muestra mensaje de no encontrado.

HU-004 – Buscar Usuario por Username

Como administrador quiero buscar un usuario por su nombre de usuario, para verificar si está registrado y consultar sus datos.

Criterios de aceptación:

- Si el username existe y el usuario no está eliminado, se muestran sus datos.
- Si el username no existe, se muestra un mensaje indicando que no fue encontrado.
- La comparación debe hacerse con el username exactamente como fue ingresado.

HU-005 – Actualizar Datos del Usuario

Como administrador quiero actualizar los datos de un usuario, para mantener su información actualizada.

Criterios de aceptación:

- Se pueden modificar username, email y estado activo.
- El username y el email nuevos no pueden estar repetidos.
- Si el ID no existe, debe informarse.
- Los datos deben validarse antes de actualizar.

HU-006 – Eliminar Usuario (Baja Lógica)

Como administrador quiero eliminar un usuario, para que no aparezca más en el sistema.

Criterios de aceptación:

- La eliminación es lógica (se marca eliminado = TRUE).
- También debe eliminarse lógicamente la credencial asociada.
- La operación debe realizarse dentro de una transacción.
- Si alguno de los pasos falla, se hace rollback.

HU-007 – Actualizar Contraseña del Usuario

Como administrador quiero cambiar la contraseña de un usuario, para mantener la seguridad del sistema.

Criterios de aceptación:

- Se debe generar un nuevo hash y un nuevo salt.
- Se debe actualizar la fecha del último cambio.
- Si la credencial no existe o está eliminada, debe informarse.

HU-008 – Listar Todas las Credenciales Activas

Como administrador quiero ver todas las credenciales activas, para controlar su estado general.

Criterios de aceptación:

- Solo se muestran credenciales con eliminado = FALSE.
- Se muestra hashPassword, salt, último cambio y si requiere reset.

HU-009 – Buscar Credencial por ID

Como administrador quiero consultar una credencial por su ID, para revisar su información detallada.

Criterios de aceptación:

- Si la credencial existe y no está eliminada, se muestra.
- Si no existe, se informa que no fue encontrada.

HU-010 – Resetear Contraseña (Marcar requiereReset)

Como administrador quiero marcar una credencial para que requiera un cambio de contraseña, para reforzar la seguridad.

Criterios de aceptación:

- Se cambia el campo requiereReset a TRUE.
- Debe existir la credencial.
- Si la credencial no existe, se debe mostrar un mensaje.

HU-011 – Relación 1:1 entre Usuario y Credencial

Como sistema quiero mantener la relación 1 a 1 entre usuario y credencial, para evitar inconsistencias en la base de datos.

Criterios de aceptación:

- Un usuario solo puede tener una credencial.
- Una credencial solo puede pertenecer a un usuario.
- La clave foránea debe ser única.
- Si se intenta asignar una credencial ya utilizada, se debe impedir.