Guia #3

JDBC & Programación funcional

Base de Datos

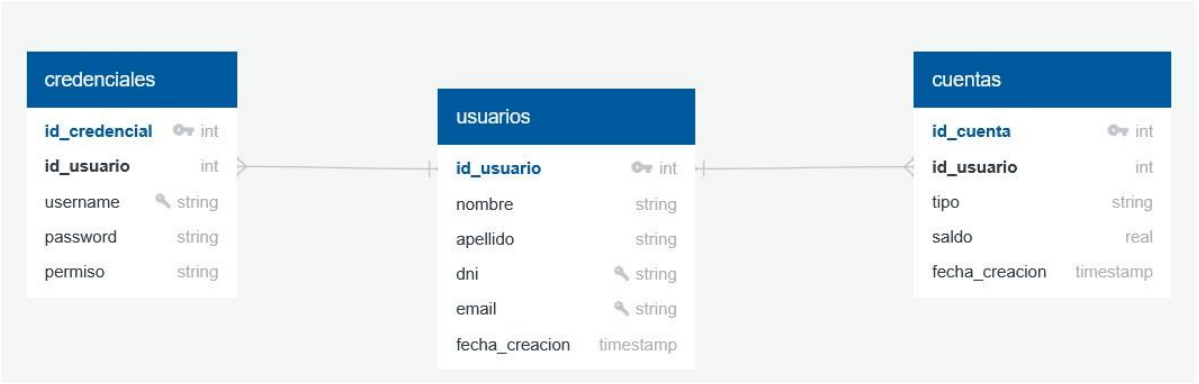
Se provee un archivo `banco.sql` que contiene la siguiente estructura de tabla:

```sql
CREATE TABLE "usuarios" (
	"id_usuario" INTEGER PRIMARY KEY AUTOINCREMENT,
	"nombre" TEXT NOT NULL,
	"apellido" TEXT NOT NULL,
	"dni" TEXT UNIQUE NOT NULL,
	"email" TEXT UNIQUE NOT NULL,
	"fecha_creacion" TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE "credenciales" (
	"id_credencial" INTEGER PRIMARY KEY AUTOINCREMENT,
	"id_usuario" INTEGER NOT NULL,
	"username" TEXT UNIQUE NOT NULL,
	"password" TEXT NOT NULL,
	"permiso" TEXT NOT NULL CHECK (permiso IN ('CLIENTE',
'ADMINISTRADOR', 'GESTOR')),
	FOREIGN KEY("id_usuario") REFERENCES "usuarios"("id_usuario") ON
DELETE CASCADE
);

CREATE TABLE "cuentas" (
	"id_cuenta" INTEGER PRIMARY KEY AUTOINCREMENT,
	"id_usuario" INTEGER NOT NULL,
	"tipo" TEXT NOT NULL CHECK (tipo IN ('CAJA_AHORRO',
'CUENTA_CORRIENTE')),
	"saldo" REAL DEFAULT 0,
	"fecha_creacion" TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
	FOREIGN KEY("id_usuario") REFERENCES "usuarios"("id_usuario") ON
DELETE CASCADE
);
```

DER



Datos Iniciales en la Base de Datos

La base de datos ya tiene algunos registros precargados:

Credenciales

id_credencial	id_usuario	username	password	permiso
1	1	juanperez	1234	CLIENTE
2	2	mariagomez	1234	ADMINISTRADOR
3	3	carloslopez	1234	GESTOR
4	4	anamartinez	1234	CLIENTE
5	5	pedrofernandez	1234	ADMINISTRADOR

Usuarios

id_usuario	nombre	apellido	dni	email	fecha_creacion
1	Juan	Pérez	12345678	juan.perez@email.com	2025-03-25 15:37:49
2	María	Gómez	87654321	maria.gomez@email.com	2025-03-25 15:37:49
3	Carlos	López	11223344	carlos.lopez@email.com	2025-03-25 15:37:49
4	Ana	Fernández	44332211	ana.fernandez@email.com	2025-03-25 15:37:49
5	Pedro	Martínez	55667788	pedro.martinez@email.com	2025-03-25 15:37:49

Cuentas

id_cuenta	id_usuario	tipo	saldo	fecha_creacion
1	1	CAJA_AHORRO	15000.5	2025-03-25 15:37:49
2	1	CUENTA_CORRIENTE	5000.75	2025-03-25 15:37:49
3	2	CAJA_AHORRO	30000	2025-03-25 15:37:49
4	3	CUENTA_CORRIENTE	12000.2	2025-03-25 15:37:49
5	4	CAJA_AHORRO	8000.9	2025-03-25 15:37:49
6	5	CUENTA_CORRIENTE	25000	2025-03-25 15:37:49

Roles/Permisos

CLIENTE: Puede gestionar sus propias cuentas y ver su saldo.

GESTOR: Puede gestionar usuarios y cuentas, pero no modificar credenciales ni eliminar administradores.

ADMINISTRADOR: Tiene acceso total a la base de datos y a la gestión de usuarios y cuentas.

Actividades

El sistema debe contar con un único menú para todos los usuarios. Al intentar realizar una acción, el sistema debe verificar que se tenga el permiso. En caso de que no se tenga, se debe lanzar una excepción NoAutorizadoException y mostrar un mensaje apropiado.

1. Conectarse a la Base de Datos

- Importar la librería/dependencia de mysqlConnector
- Crear una clase `DBConnection` que maneje la conexión a la base de datos MySQL.

2. Permitir la creación de un nuevo usuario

- El usuario debe poder registrarse con su nombre, apellido, DNI y email.
- Se deben generar automáticamente sus credenciales con un username único y su password.
- Por defecto, el usuario tendrá el permiso de `CLIENTE`.

- El usuario puede crear una cuenta nueva de tipo `CUENTA_CORRIENTE`. Al registrarse, se le debe crear automáticamente una `CAJA_AHORRO`.
- **Restricción:** Cada usuario puede tener solo una `CAJA_AHORRO`, pero puede poseer múltiples `CUENTA_CORRIENTE`.

3. Permitir el inicio de sesión

- El sistema, previo a operar, debe permitirle al usuario iniciar sesión y validar sus credenciales en la base de datos
- Usar `Optional<Usuario>` para representar el usuario autenticado..

4. Obtener todos los usuarios registrados

- **Solo GESTORES y ADMINISTRADORES** pueden listar los usuarios del sistema.

5. Buscar un usuario por DNI o email

- **Solo GESTORES y ADMINISTRADORES** pueden buscar usuarios en la base de datos.
- Utilizar `Optional<Usuario>` para evitar valores nulos.
- Aplicar `filter` en un `Stream` para buscar por DNI o email.

6. Modificar los datos de un usuario

- **CLIENTES** pueden modificar **solo sus propios datos** (nombre, apellido, email).
- **GESTORES** pueden modificar los datos de cualquier **CLIENTE**.
  ADMINISTRADORES pueden modificar los datos de cualquier usuario.

7. Eliminar un usuario

- **GESTORES** pueden eliminar solo **CLIENTES.**
- **ADMINISTRADORES** pueden eliminar cualquier usuario**.**
  Elimina también las credenciales y cuentas asociadas (`ON DELETE CASCADE`).

8. Listar todas las cuentas de un usuario

- **CLIENTES** pueden listar sus propias cuentas.
- **GESTORES y ADMINISTRADORES** pueden listar las cuentas de cualquier usuario.
- Utilizar `Stream` para transformar los resultados en una colección.

9. Obtener el saldo total de un usuario

- **CLIENTES** pueden ver solo su saldo.
- **GESTORES y ADMINISTRADORES** pueden ver el saldo de cualquier usuario.
- Utilizar `Stream` y `reduce` para calcular el saldo total.

10. Realizar un depósito en una cuenta

- **CLIENTES** pueden depositar en sus propias cuentas.
- **GESTORES** pueden depositar en cuentas de **CLIENTES**.
- **ADMINISTRADORES** pueden depositar en cualquier cuenta.
- Utilizar `Optional` para verificar la existencia de la cuenta.

11. Realizar una transferencia entre cuentas

- **CLIENTES** pueden transferir entre sus propias cuentas o a otros usuarios.
- **GESTORES y ADMINISTRADORES** pueden transferir dinero entre cuentas de diferentes usuarios.
- La transferencia debe validar saldo suficiente antes de ejecutarse.
- Implementar la lógica con `Stream` para evitar bucles explícitos.

12. Obtener la cantidad de usuarios por tipo de permiso

- `GESTORES` y `ADMINISTRADORES` pueden acceder a esta información.
- Contar cuántos usuarios hay en el sistema agrupados por su tipo de permiso (`CLIENTE`, `GESTOR`, `ADMINISTRADOR`).
  Utilizar `Collectors.groupingBy` y `Collectors.counting()` para obtener el resultado en un `Map<String, Long>`.

13. Obtener la cantidad total de cuentas por tipo y mostrarlas

- **GESTORES y ADMINISTRADORES** pueden acceder a este dato.
  Utilizar `Collectors.groupingBy` y `Collectors.counting()` para contar las cuentas por tipo.

14. Obtener el usuario con mayor saldo total

- **Solo ADMINISTRADORES** pueden acceder a esta información.
- Utilizar `Stream`, `mapToDouble` y `max` para encontrar el usuario con el mayor saldo.

15. Listar los usuarios ordenados por su saldo total (de mayor a menor)

- **Solo ADMINISTRADORES** pueden acceder a esta información.
- Usar `Stream` para calcular el saldo total de cada usuario.