LABORATORIO #4,5,6
SOFTWARE DEVELOPMENT: LAYERS

Valentina Bueno Valles
Valentina Viafara Esteban

I. *Descripción de las piezas de software implementadas en cada una de las capas:*

1. Capa de Acceso a Datos.
    - Identificación de archivos modificados (*.java*, *.xml*).

      El primer archivo modificado fue el llamado *User*, se borró el atributo *balance* y sus respectivos métodos *get* y *set*. También se insertaron los atributos *mail*, *password* con sus respectivos métodos *get* y *set*.

```
@Entity
public class User implements Serializable {

    @PrimaryKey
    @ColumnInfo(name = "idU")
    public int id;

    @ColumnInfo(name = "name")
    public String name;

    @ColumnInfo(name = "mail")
    public String mail;

    @ColumnInfo(name = "password")
    public String password;

    public int getId() {
        return id;
    }
```

      Se modifico el archivo *database* ya que se agregaron las clases abstractas del Dao pertenecientes a *Manager, Count y Transaction,* de manera que estas puedan tener acceso a la base de datos para manejarla.

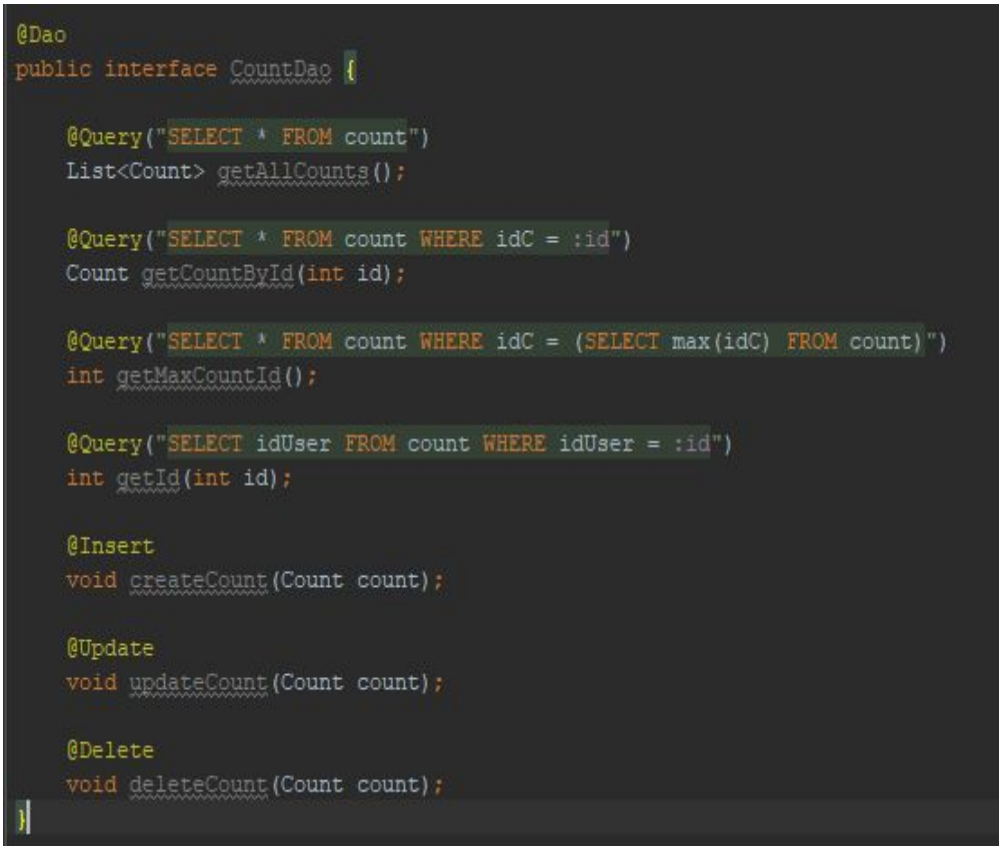
```java
package co.edu.unal.se1.dataAccess.db;

import androidx.room.RoomDatabase;

import co.edu.unal.se1.dataAccess.dao.CountDao;
import co.edu.unal.se1.dataAccess.dao.UserDao;
import co.edu.unal.se1.dataAccess.dao.ManagerDao;
import co.edu.unal.se1.dataAccess.dao.TransactionDao;
import co.edu.unal.se1.dataAccess.model.Count;
import co.edu.unal.se1.dataAccess.model.User;
import co.edu.unal.se1.dataAccess.model.Manager;
import co.edu.unal.se1.dataAccess.model.Transaction;

@androidx.room.Database(entities = {User.class, Count.class, Manager.class, Transaction.class}, version = 1)

public abstract class Database extends RoomDatabase {

    public abstract UserDao userDao();
    public abstract CountDao countDao();
    public abstract ManagerDao managerDao();
    public abstract TransactionDao transactionDao();
}
```

- Identificación de nuevos archivos agregados (*.java*, *.xml*).

  Se agrego un archivo con extensión .java que corresponde a la clase *Count* la cual se relaciona con la entidad *User* por medio de una llave foránea vinculada al *id* de la clase *User,* también se agregaron los atributos *idC* que corresponde a la identificación de la cuenta y *balance,* cada uno con sus respectivos métodos *get* y *set*.

```java
@Entity (foreignKeys = @ForeignKey(entity = User.class,
        parentColumns = "idU",
        childColumns = "idC",
        onDelete = ForeignKey.NO_ACTION))

public class Count implements Serializable {

    @PrimaryKey (autoGenerate = true)
    @ColumnInfo(name = "idC")
    public int id_c;

    @ColumnInfo(name = "balance")
    public double balance;

    @ColumnInfo(name = "idUser")
    public int id_user;

    public int getId() {
        return id_c;
    }

    public void setId_c(int id) {
        this.id_c = id;
    }

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }
```

Se añadió otro archivo de extensión . java que corresponde a la interfaz *CountDao*, en la cual están las operaciones básicas de sql para interactuar con la base de datos y dos consultas adicionales, la primera obtiene el *idC* más grande de manera que la asignación de los id de las cuentas sea secuencial, y la segunda filtra el usuario que está vinculada a la cuenta seleccionada.



```java
@Dao
public interface CountDao {

    @Query("SELECT * FROM count")
    List<Count> getAllCounts();

    @Query("SELECT * FROM count WHERE idC = :id")
    Count getCountById(int id);

    @Query("SELECT * FROM count WHERE idC = (SELECT max(idC) FROM count)")
    int getMaxCountId();

    @Query("SELECT idUser FROM count WHERE idUser = :id")
    int getId(int id);

    @Insert
    void createCount(Count count);

    @Update
    void updateCount(Count count);

    @Delete
    void deleteCount(Count count);
}
```

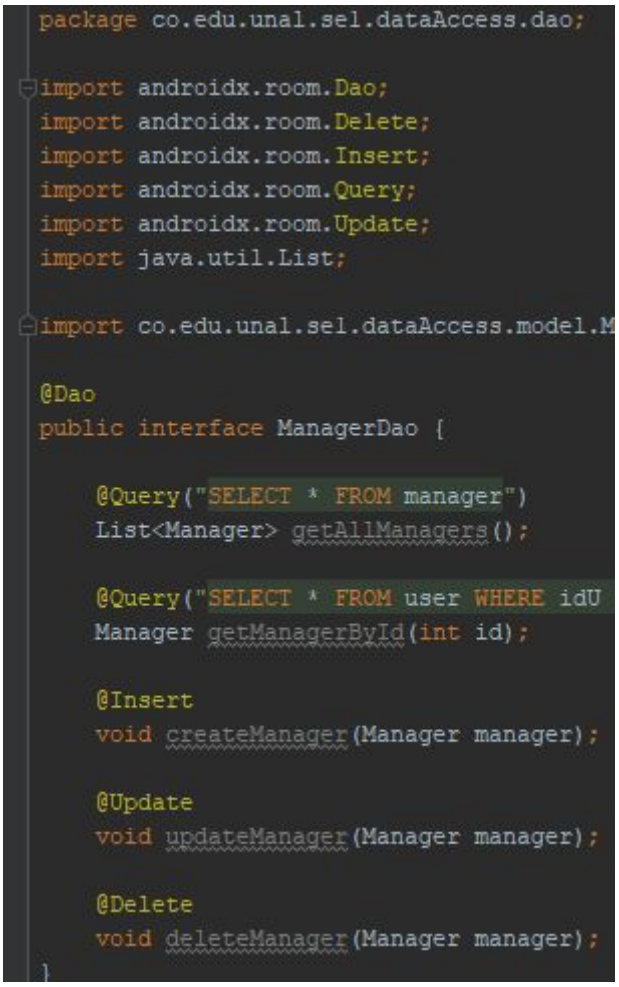
También se adicionó el archivo de extensión .java que corresponde a la clase *CountRepository* en la cual están las instancia de los métodos de las operaciones básicas de sql que operan directamente en las cuentas de los usuarios. Utiliza el *CountDao* para recuperar datos de la base de datos.

```java
public class CountRepository {

    private String DB_NAME = "sel_db_bank";

    private Database database;
    public CountRepository(Context context) {
        database = Room.databaseBuilder(context, Database.class, DB_NAME).
                allowMainThreadQueries().build();
    }

    public List<Count> getAllCounts() {
        return database.countDao().getAllCounts();
    }

    public Count getCountById(int id) {
        return database.countDao().getCountById(id);
    }

    public int getMaxCountId(){
        return database.countDao().getMaxCountId();
    }

    public User getUserByIdC(int id){
        return database.userDao().getUserById(id);
    }

    public void createCount(final Count count) {
        database.countDao().createCount(count);
    }

    public void updateCount(Count count) {
        database.countDao().updateCount(count);
```

Se agregó un archivo de extensión .java que corresponde a la clase *Manager* para poder vincularlo a la transacción en caso, como se especifica en el laboratorio 2, que el usuario solicite directamente al administrador hacer una transferencia de dinero. Tiene los siguientes atributos: *idM, nameM, mailM y passwordM,* con sus respectivos métodos *get y set.*

```java
@Entity
public class Manager implements Serializable {

    @PrimaryKey
    @ColumnInfo(name = "idM")
    public int id_m;

    @ColumnInfo(name = "nameM")
    public String name_m;

    @ColumnInfo(name = "mailM")
    public String mail_m;

    @ColumnInfo(name = "passwordM")
    public String password_m;

    public int getId() {
        return id_m;
    }

    public void setId(int id) {
        this.id_m = id;
    }

    public String getName() {
        return name_m;
    }

    public void setName(String name) {
        this.name_m = name;
    }
```

Se añadió el archivo con extensión . java correspondiente a la clase *ManagerDao* que contiene las operaciones básicas de sql (CRUD) para poder manejar los datos referentes a la entidad Manager.

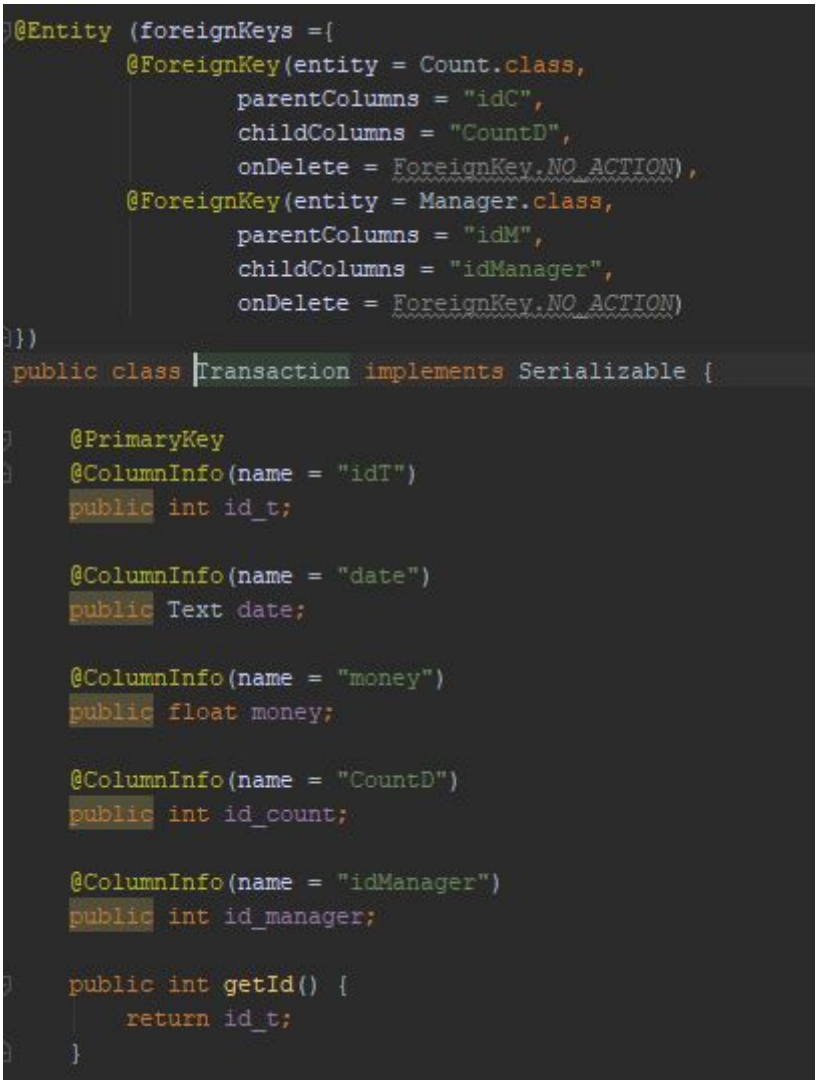


```java
package co.edu.unal.sel.dataAccess.dao;

import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.Query;
import androidx.room.Update;
import java.util.List;

import co.edu.unal.sel.dataAccess.model.M

@Dao
public interface ManagerDao {

    @Query("SELECT * FROM manager")
    List<Manager> getAllManagers();

    @Query("SELECT * FROM user WHERE idU
    Manager getManagerById(int id);

    @Insert
    void createManager(Manager manager);

    @Update
    void updateManager(Manager manager);

    @Delete
    void deleteManager(Manager manager);
}
```

Se agregó un archivo de extensión .java que corresponde a la clase *ManagerRepository* en el cual está la sección de código que describe las operaciones básicas de sql por medio del uso de la clase *ManagerDao*.

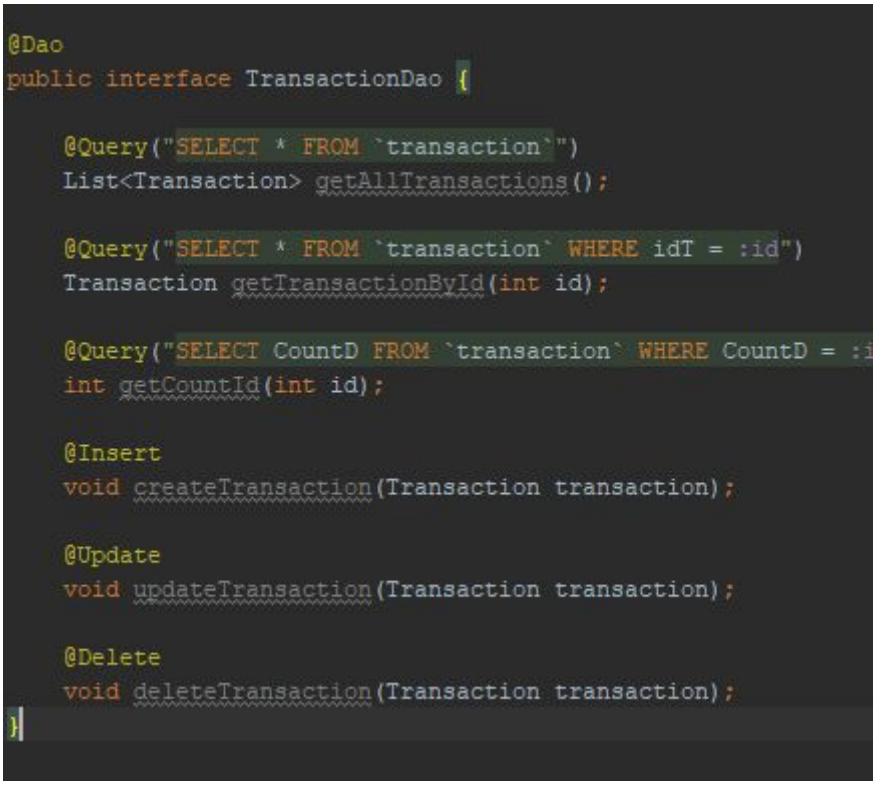
```java
public class ManagerRepository {

    private String DB_NAME = "sel_db_bank";

    private Database database;
    public ManagerRepository(Context context) {
        database = Room.databaseBuilder(context, Database.class, DB_NAME).
                allowMainThreadQueries().build();
    }

    public List<Manager> getAllManagers() {
        return database.managerDao().getAllManagers();
    }

    public Manager getManagerById(int id) {
        return database.managerDao().getManagerById(id);
    }

    public void createManager(final Manager manager) {
        database.managerDao().createManager(manager);
    }

    public void updateManager(Manager manager) {
        database.managerDao().updateManager(manager);
    }

    public void deleteManager(int id) {
        Manager manager = database.managerDao().getManagerById(id);
        database.managerDao().deleteManager(manager);
    }
}
```

Se creó el archivo de extensión .java que corresponde a la clase *Transaction* que está vinculada a las entidades *Manager* y *Count* por medio de una llave foránea que se relaciona con las llaves primarias de las dos entidades. Tiene como atributos *idT, date, money* con sus respectivos métodos get y set.



```
@Entity (foreignKeys ={
        @ForeignKey(entity = Count.class,
                parentColumns = "idC",
                childColumns = "CountD",
                onDelete = ForeignKey.NO_ACTION),
        @ForeignKey(entity = Manager.class,
                parentColumns = "idM",
                childColumns = "idManager",
                onDelete = ForeignKey.NO_ACTION)
})
public class Transaction implements Serializable {

    @PrimaryKey
    @ColumnInfo(name = "idT")
    public int id_t;

    @ColumnInfo(name = "date")
    public Text date;

    @ColumnInfo(name = "money")
    public float money;

    @ColumnInfo(name = "CountD")
    public int id_count;

    @ColumnInfo(name = "idManager")
    public int id_manager;

    public int getId() {
        return id_t;
    }
```

Se añadió un archivo de extensión .java correspondiente a la clase *TransactionDao* que tiene las operaciones básicas de sql que manejan los datos de la entidad *Transaction*.



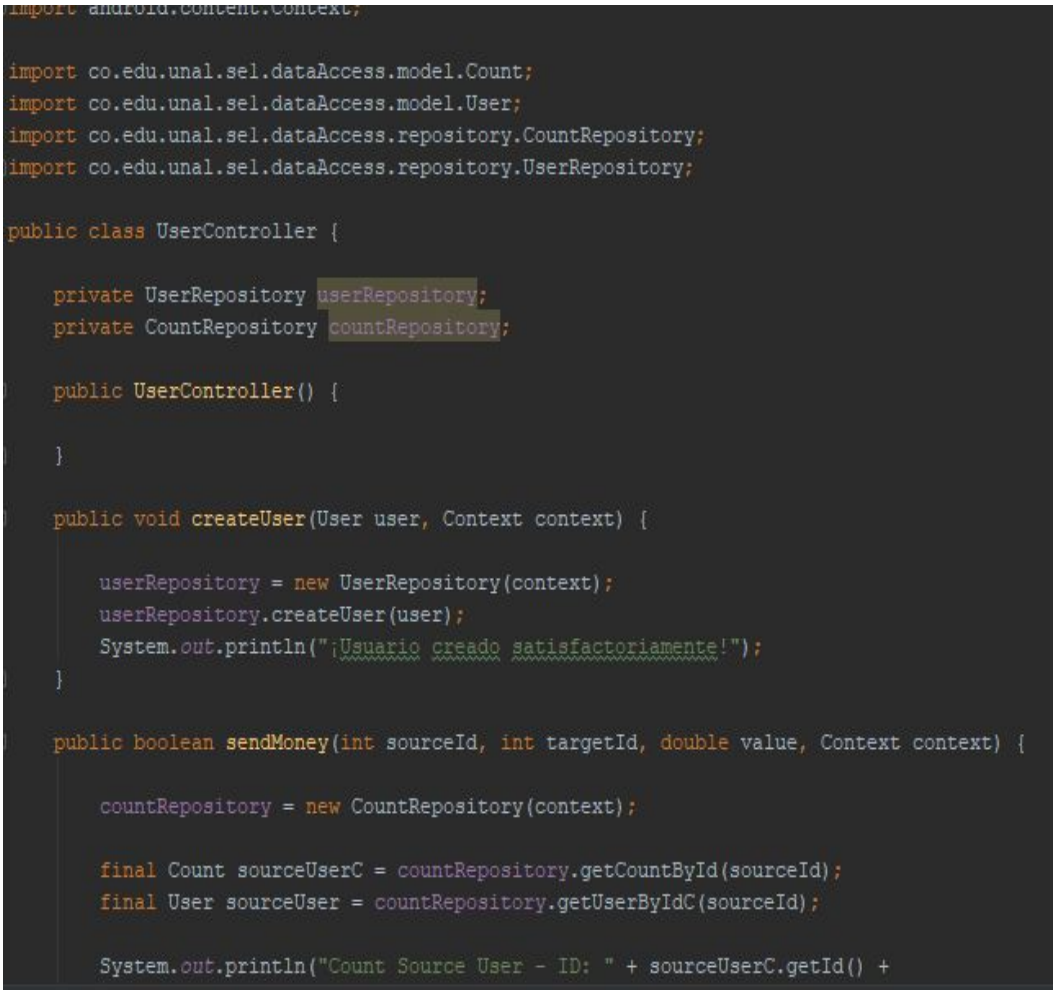
```java
@Dao
public interface TransactionDao {

    @Query("SELECT * FROM `transaction`")
    List<Transaction> getAllTransactions();

    @Query("SELECT * FROM `transaction` WHERE idT = :id")
    Transaction getTransactionById(int id);

    @Query("SELECT CountD FROM `transaction` WHERE CountD = :i
    int getCountId(int id);

    @Insert
    void createTransaction(Transaction transaction);

    @Update
    void updateTransaction(Transaction transaction);

    @Delete
    void deleteTransaction(Transaction transaction);
}
```

Por último se agrego el archivo de extensión .java que corresponde a la clase *TransactionRepository* que usa el *Dao* de *Transaction* para recuperar datos y manejar operaciones básicas del sql en la entidad.

```
private Database database;
public TransactionRepository(Context context) {
    database = Room.databaseBuilder(context, Database.class, DB_NAME).
            allowMainThreadQueries().build();
}

public List<Transaction> getAllTransactions() {
    return database.transactionDao().getAllTransactions();
}

public Transaction getTransactionById(int id) {
    return database.transactionDao().getTransactionById(id);
}

public Count getCountByIdC(int id){
    return database.countDao().getCountById(id);
}

public Manager getManagerByIdC(int id){
    return database.managerDao().getManagerById(id);
}

public void createTransaction(final Transaction transaction) {
    database.transactionDao().createTransaction(transaction);
}

public void updateTransaction(Transaction transaction) {
    database.transactionDao().updateTransaction(transaction);
}

public void deleteTransaction(int id) {
    Transaction transaction = database.transactionDao().getTransactionById(id);
```

2. Capa de Lógica de Negocio.

    - Identificación de archivos modificados (*.java*, *.xml*).
      El archivo disponible es de extensión .java *UserController* en el cual se modifica ligeramente el método sendMoney para poder comunicar los datos recibidos por el usuario en la capa de presentación con la base de datos, teniendo en cuenta que las entidades *User* y *Count* se relacionan entre sí. También como se hizo la separación lógica de estas entidades es necesario importar las dos clases para poder acceder a sus atributos y métodos.



```
import android.content.Context;

import co.edu.unal.sel.dataAccess.model.Count;
import co.edu.unal.sel.dataAccess.model.User;
import co.edu.unal.sel.dataAccess.repository.CountRepository;
import co.edu.unal.sel.dataAccess.repository.UserRepository;

public class UserController {

    private UserRepository userRepository;
    private CountRepository countRepository;

    public UserController() {

    }

    public void createUser(User user, Context context) {

        userRepository = new UserRepository(context);
        userRepository.createUser(user);
        System.out.println("¡Usuario creado satisfactoriamente!");
    }

    public boolean sendMoney(int sourceId, int targetId, double value, Context context) {

        countRepository = new CountRepository(context);

        final Count sourceUserC = countRepository.getCountById(sourceId);
        final User sourceUser = countRepository.getUserByIdC(sourceId);

        System.out.println("Count Source User - ID: " + sourceUserC.getId() +
```

- Identificación de nuevos archivos agregados (*.java*, *.xml*).
  Se agrega por cada nueva entidad un Controller, es decir, para las nuevas entidades que se agregaron en el punto anterior *Count*, *Manager* y *Transaction*, se crean los archivos *CountController*, *ManagerController* y *TransactionController*. Para cada uno de estos nuevos Controladores se reconoce la necesidad de implementar los métodos propuestos en el laboratorio 2 para cada clase con el fin de garantizar el correcto funcionamiento de la aplicación.

```java
package co.edu.unal.se1.businessLogic.controller;

import android.content.Context;

import co.edu.unal.se1.dataAccess.model.Count;
import co.edu.unal.se1.dataAccess.repository.CountRepository;



public class CountController {

    private CountRepository countRepository;


    public CountController() {

    }

    public void createCount(Count count, Context context) {

        countRepository = new CountRepository(context);
        countRepository.createCount(count);
        System.out.println("¡Cuenta creada satisfactoriamente!");
    }
}
```

```java
package co.edu.unal.se1.businessLogic.controller;

import android.content.Context;

import co.edu.unal.se1.dataAccess.model.Manager;
import co.edu.unal.se1.dataAccess.repository.ManagerRepository;



public class ManagerController {

    private ManagerRepository managerRepository;


    public ManagerController() {

    }

    public void createManager(Manager manager, Context context) {

        managerRepository = new ManagerRepository(context);
        managerRepository.createManager(manager);
        System.out.println("¡Administrador creado satisfactoriamente!");
    }
}
```

```java
package co.edu.unal.se1.businessLogic.controller;

import android.content.Context;

import co.edu.unal.se1.dataAccess.model.Transaction;
import co.edu.unal.se1.dataAccess.repository.TransactionRepository;



public class TransactionController {

    private TransactionRepository transactionRepository;


    public TransactionController() {

    }

    public void createTransaction(Transaction transaction, Context context) {

        transactionRepository = new TransactionRepository(context);
        transactionRepository.createTransaction(transaction);
        System.out.println("¡Transacción creada satisfactoriamente!");
    }
}
```

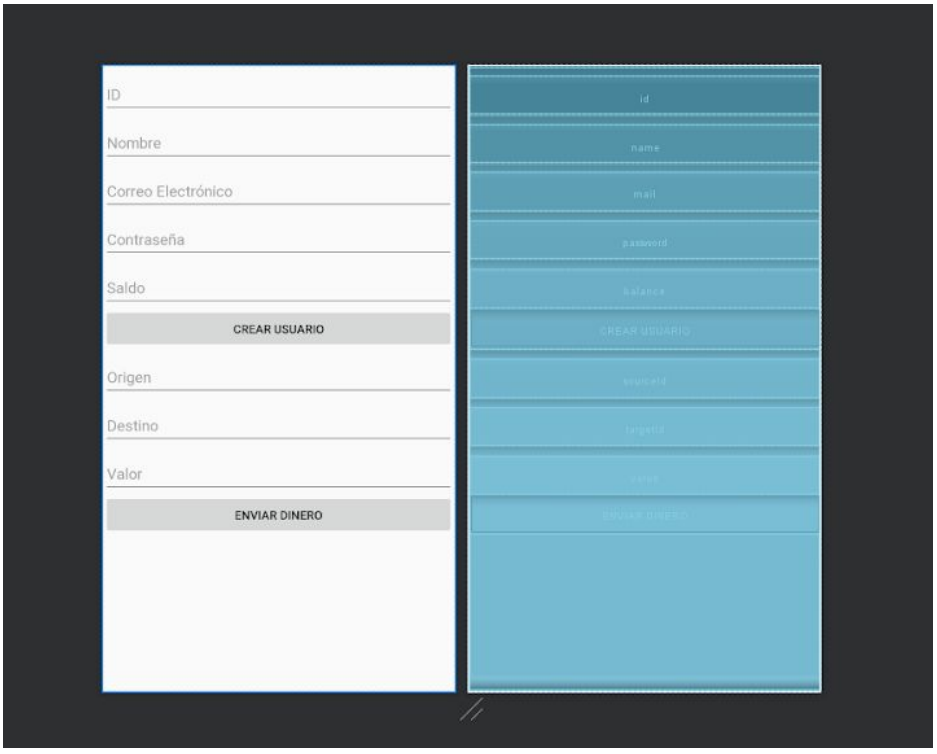
3. Capa de presentación.

    - Identificación de archivos modificados (*.java*, *.xml*).
      Se modifica ligeramente la clase reconocida por el .java *MainActivity* para hacer coincidir todo el modelo del laboratorio 2 con la aplicación que se desea presentar.

```java
package co.edu.unal.se1.presentation.activity;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import com.google.android.material.textfield.TextInputEditText;

import co.edu.unal.se1.R;
import co.edu.unal.se1.businessLogic.controller.CountController;
import co.edu.unal.se1.businessLogic.controller.UserController;
import co.edu.unal.se1.dataAccess.model.User;
import co.edu.unal.se1.dataAccess.model.Count;
import co.edu.unal.se1.dataAccess.repository.UserRepository;
import co.edu.unal.se1.dataAccess.repository.CountRepository;

public class MainActivity extends AppCompatActivity {

    private UserController userController;
    private CountController countController;
    private CountRepository countRepository;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        final TextInputEditText idInput = findViewById(R.id.id);
        final TextInputEditText nameInput = findViewById(R.id.name);
        final TextInputEditText mailInput = findViewById(R.id.mail);
        final TextInputEditText passwordInput = findViewById(R.id.password);
        final TextInputEditText balanceInput = findViewById(R.id.balance);

        Button createButton = findViewById(R.id.createButton);
        createButton.setOnClickListener((v) -> {
```

```
                User user = new User();
                Count count = new Count();
                user.setId(Integer.parseInt(idInput.getText().toString()));
                user.setName(nameInput.getText().toString());
                user.setMail(mailInput.getText().toString());
                user.setPassword(passwordInput.getText().toString());
                count.setId_c(countRepository.getMaxCountId()+1);
                count.setIdUser(user.getId());
                count.setBalance(Double.parseDouble(balanceInput.getText().toString()));

                userController = new UserController();
                userController.createUser(user, getApplicationContext());
                countController = new CountController();
                countController.createCount(count, getApplicationContext());
            });

            final TextView sourceIdInput = findViewById(R.id.sourceId);
            final TextView targetIdInput = findViewById(R.id.targetId);
            final TextView valueInput = findViewById(R.id.value);

            Button sendMoneyButton = findViewById(R.id.sendMoneyButton);
            sendMoneyButton.setOnClickListener((v) -> {

                int sourceId = Integer.parseInt(sourceIdInput.getText().toString());
                int targetId = Integer.parseInt(targetIdInput.getText().toString());
                double value = Double.parseDouble(valueInput.getText().toString());

                boolean transaction = userController.sendMoney(sourceId, targetId, value, getApplicationContext());

                if (transaction) {
                    System.out.println("¡Transacción satisfactoria!");
                } else {
                    System.out.println("¡Transacción no satisfactoria!");
                }
            });
        }
    }
```

También se modificó ligeramente el archivo .xml *activity_main* que corresponde a la interfaz gráfica que reconoce el usuario en el momento de ejecutar la aplicación en su teléfono móvil.



- Identificación de nuevos archivos agregados (*.java*, *.xml*).
  Se reconoce la necesidad de agregar archivos .xml y .java para cada una de las “ventanas” que presentan funcionalidades para

el usuario. Adicionalmente, se requiere separar la funcionalidad de crear usuario de enviar dinero en la misma venta que actualmente está presente en el archivo *MainActivity*.

II. *Descripción del flujo de cada **funcionalidad** del sistema de software a través de las tres capas.*

1) Funciones importantes

A. *createUser* que se cataloga como método de User opera de la siguiente manera:
   1. Capa de Presentación: El usuario en su celular llena los campos que se le piden (id, nombre, correo electrónico, contraseña) para poder tener un usuario en la aplicación y hacer uso de las otras funcionalidades.
   2. Capa de Lógica de Negocio: Los datos llegan a través de *UserController* que se comunica con *UserRepository* en la capa de acceso a datos.
   3. Capa de Acceso a Datos: *UserRepository* se comunica con *UserDao* para llevar a cabo los querys necesarios para modificar correctamente la base de datos.

B. *sendMoney* que se cataloga como método de User opera de la siguiente manera:
   1. *Capa de Presentación:* El usuario en su celular llena los campos que se le piden para poder enviar el dinero (id origen, id destino, valor a enviar)
   2. *Capa de Lógica de Negocio:* Estos valores pasan a la capa de lógica de negocio donde reposa el *UserController* que se comunica con la capa de acceso a datos por medio de *UserRepository*.
   3. *Capa de Acceso a Datos: UserRepository* se comunica entonces con *UserDao* para llevar a cabo los querys necesarios según lo que solicita el usuario y realiza los cambios correspondientes en la base de datos.

2) Descripción funcionamiento sistema

A. Capa de Acceso a datos: Tenemos las entidades/objetos principales que describen nuestros modelos de entidad- relación y de clases, User, Transaction, Manager y Count, con sus respectivos atributos definidos en el laboratorio 2.

Tenemos la base de datos que se encarga de almacenar todos los registros de las entidades. También está el Dao de cada una que contiene las operaciones para relacionar y modificar las bases de datos por medio de sql, permite recuperar y obtener los datos. El repository es la abstracción de esos datos, permite leer esos datos por medio del Dao.

B. Capa de Lógica de Negocio: El Controller se encarga de unificar las entidades y clases del Acceso a Datos. Usa métodos con el fin de transferir dinero y crear usuarios, las principales funcionalidades de la aplicación que interactúan con el usuario.

C. Capa de Presentación: Provee la interfaz que encaja todas las piezas del software para que el usuario pueda interactuar de manera fácil con la aplicación, por medio de ventanas que le permite utilizar las funcionalidades principales del programa.

III. Enlace al repositorio público creado y en el cual se desarrolló el laboratorio.

https://github.com/un-sei2019ii-labs-valealcuadrado/bank.git