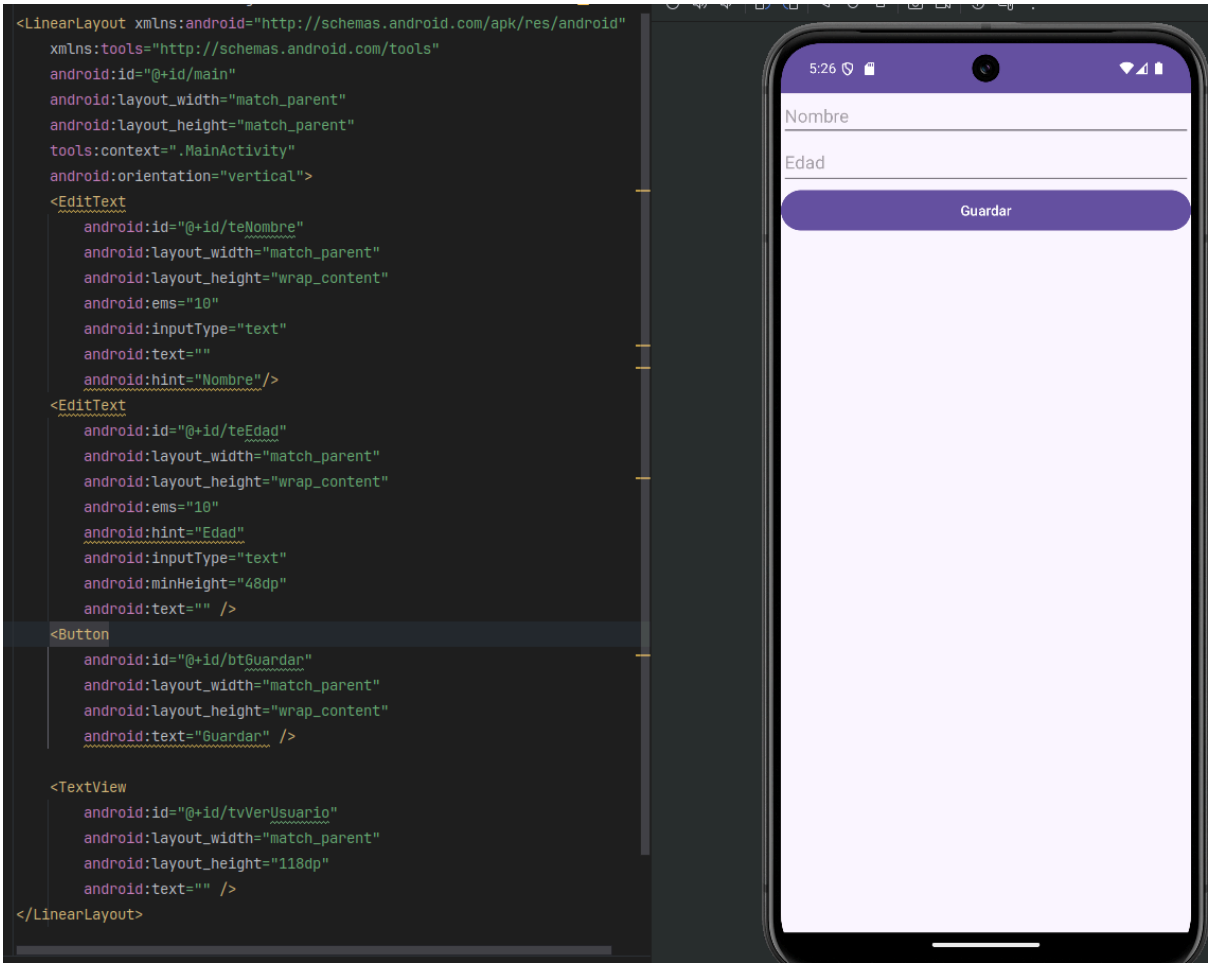
TP 4 - ViewModel + Live - Lista de Usuarios

Objetivo: comprender cómo *`LiveData`* actualiza de manera automática los componentes activos de la UI cuando los datos cambian, sin necesidad de que ustedes manejen manualmente el ciclo de vida o realicen comprobaciones adicionales.

1. Crear un proyecto y reutilizaremos parte del código del proyecto anterior.



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">
    <EditText
        android:id="@+id/teNombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="text"
        android:text=""
        android:hint="Nombre"/>
    <EditText
        android:id="@+id/teEdad"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Edad"
        android:inputType="text"
        android:minHeight="48dp"
        android:text="" />
    <Button
        android:id="@+id/btGuardar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Guardar" />

    <TextView
        android:id="@+id/tvVerUsuario"
        android:layout_width="match_parent"
        android:layout_height="118dp"
        android:text="" />
</LinearLayout>
```


2. Crear una clase usuario con dos atributos nombre y edad, el constructor que inicializa esos atributos y los getters necesarios para mostrarlos.
3. Crear un nuevo ViewModel que utiliza LiveData, que nos actualizará mientras la activity esté activa.
   MutableLiveData es una clase que extiende **LiveData** y permite que su valor cambie. En este caso, está diseñada para almacenar y observar una lista de usuarios (List<Usuario>).
   getUserList() devuelve la instancia de *LiveData<List<Usuario>>*. Como se devuelve LiveData y no MutableLiveData, cualquier componente externo que observe esta lista puede leerla, pero no modificarla directamente.

```java
public class UserLiveViewModel extends ViewModel {  no usages

    private MutableLiveData<List<Usuario>> listaUsuariosLiveData;  4 usages
    private List<Usuario> listaUsuarios;  3 usages

    public LiveData<List<Usuario>> getUserList(){  no usages
        if (listaUsuariosLiveData==null){
            listaUsuariosLiveData=new MutableLiveData<>();
            listaUsuarios=new ArrayList<>();
        }
        return listaUsuariosLiveData;
    }

    public void addUser(Usuario usuario){  no usages
        listaUsuarios.add(usuario);
        listaUsuariosLiveData.setValue(listaUsuarios);
    }
}
```

4. En el MainActivity crear el binding y asociar el viewModel creado,

```java
public class MainActivity extends AppCompatActivity {
    private ActivityMainBinding binding;  13 usages
    private UserLiveViewModel userLiveViewModel;  3 usages
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding=ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
        userLiveViewModel=new ViewModelProvider( owner: this).get(UserLiveViewModel.class);
        tarea();
    }
    public void tarea(){...}
}
```

5. Desarrollar en el método tarea los listeners para el botón Guardar y un Observer. La idea clave del **Observer** es que permite que la vista reaccione automáticamente cuando los datos observados cambian.

```
public void tarea(){  1 usage
    binding.btGuardar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String nombre=binding.teNombre.getText().toString();
            int edad;
            if (!nombre.isEmpty() && !binding.teEdad.getText().toString().isEmpty()) {
                try {
                    edad = Integer.parseInt(binding.teEdad.getText().toString());
                    Usuario usuario = new Usuario(nombre, edad);
                    userLiveViewModel.addUser(usuario);
                    binding.teNombre.setText("");
                    binding.teEdad.setText("");
                } catch (NumberFormatException e) {
                    binding.tvVerUsuario.setText("Error de formato en la edad");
                }
            } else {
                binding.tvVerUsuario.setText("ERROR");
            }
            binding.teNombre.setText("");
            binding.teEdad.setText("");
        }
    });
  final Observer<List<Usuario>> listaObservada=new Observer<List<Usuario>>() {
      @Override
      public void onChanged(List<Usuario> usuarios) {
          String texto="";
          for (Usuario u: usuarios){
              texto+="Nombre: "+u.getNombre()+"Edad: "+u.getEdad()+"\n";
          }
          binding.tvVerUsuario.setText(texto);
      }
  };
  userLiveViewModel.getUserList().observe( owner: this, listaObservada);
}
```