MegaInfalibleIncreibleTutotial

1-Lo primero es crear el proyecto,en el MainActivity vamos a meter una **Toolbar** personalizada para cambiar el titulo,también utilizaremos el **binding** y lo inflaremos y también **iniciaremos** un **ViewModel** el cual actualmente no tenemos creado.

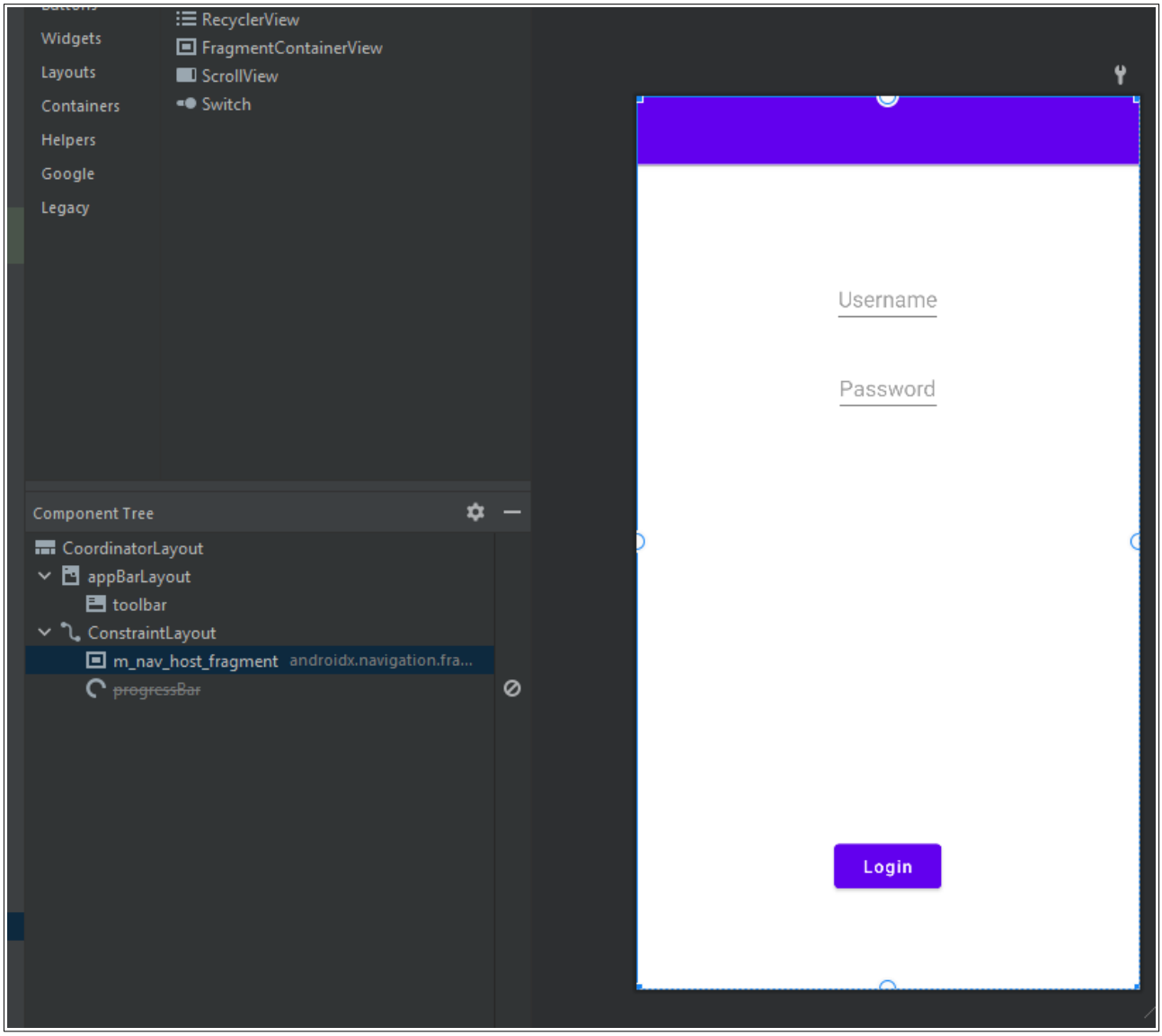
```
class MainActivity : AppCompatActivity() {

    private lateinit var binding : ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        //Cargar viewModel
        val loginViewModel= ViewModelProvider( owner: this)[LoginViewModel::class.java]
        loginViewModel.loadData()

        setTheme(R.style.Theme_Test1Ev)

        setContentView(
            ActivityMainBinding.inflate(layoutInflater).also{  it: ActivityMainBinding
            binding = it
        }.root)

        //Toolbar personalizada
        setSupportActionBar(binding.toolbar)
        binding.toolbar.title = "Test 1st Ev."



    }
}
```

2-Para que esto funcione debemos obviamente activar el viewBinding,te recuerdo aquí también la lista de dependencias de paso

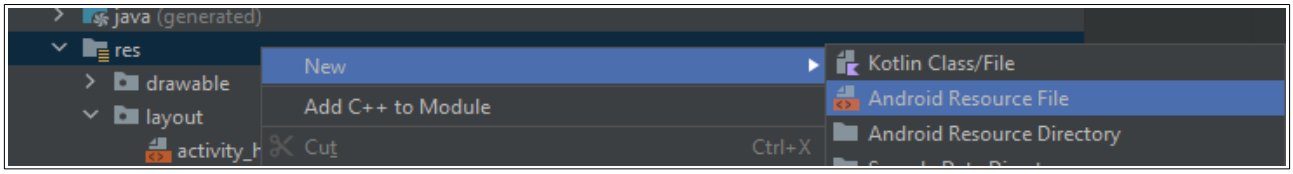
```
dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.1'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

    def viewModelVersion ="2.4.0"
    implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$viewModelVersion"
    implementation "androidx.lifecycle:lifecycle-livedata-ktx:$viewModelVersion"
    def navigationVersion = "2.3.5"
    implementation "androidx.navigation:navigation-fragment-ktx:$navigationVersion"
    implementation "androidx.navigation:navigation-ui-ktx:$navigationVersion"

    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.9")

    //GSON
    implementation 'com.google.code.gson:gson:2.8.8'

    //SafeArgs
    //apply plugin: "androidx.navigation.safeargs"

    //material
    implementation 'com.google.android.material:material:1.4.0'
}
```

```
buildFeatures{
    viewBinding true
}
```

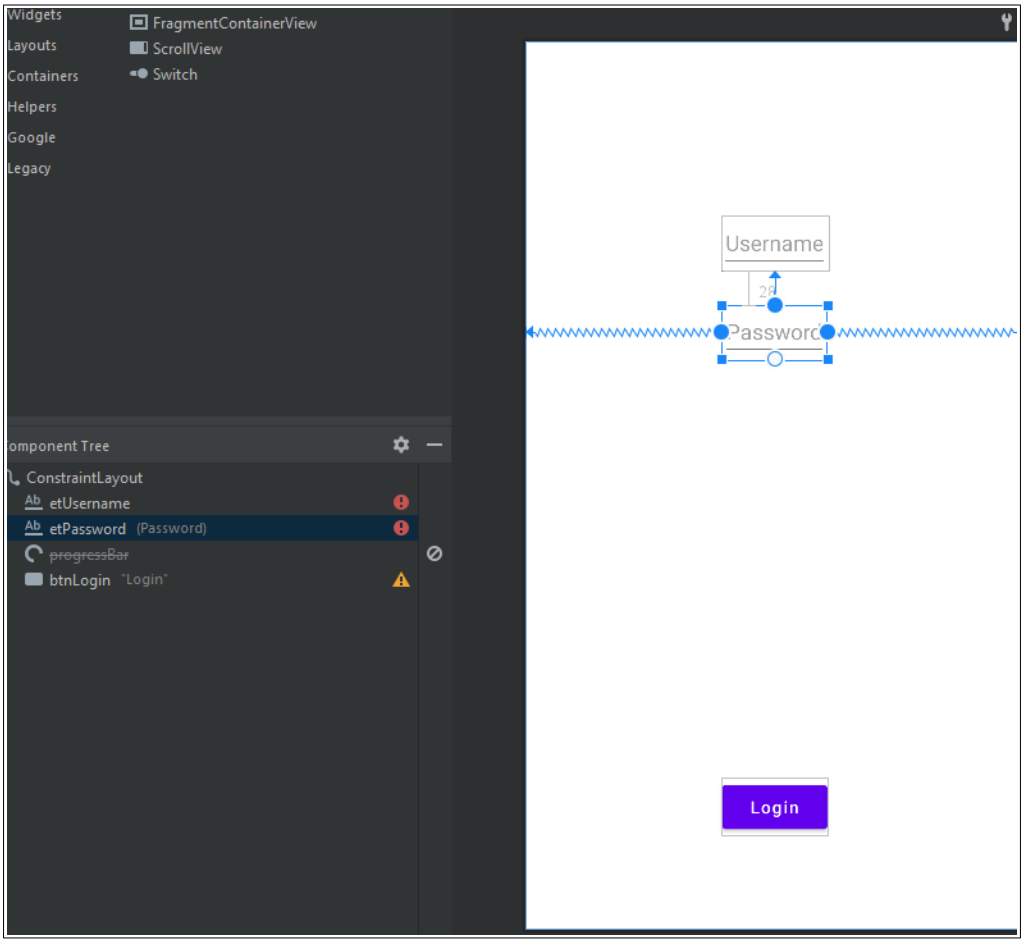
2-Ahora que tenemos eso vamos a mirar el xml del MainAcitivty,dentro hay un appBarLayout que dentro tiene la toolbar,tambiénhay un contenedor que apunta a un navGraph y por ultimo un progressBar el cual para el efecto visual de la ruedecita



3-Para crear el navGraph teníamos que hacer eso:



4-Dentro del NavGraph crearemos el LoginFragment (no voy a poner nada de esto,doble click y au),
dentro del Xml de este tendremos 2 TextView(username y password) , otra progressBar y un btn de login



5-Ahora vamos a ver el .kt del loginFragment,lo primero es el binding

```
class LoginFragment : Fragment() {

    private lateinit var binding: FragmentLoginBinding

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return FragmentLoginBinding.inflate(
            inflater,
            container,
            attachToParent: false
        ).also { it: FragmentLoginBinding
            binding = it
        }.root
    }
```

Lo siguiente es crear variables con los datos del username y la password y utilizar el viewModel.

¿Qué hacemos con el ViewModel?

1-Inicializarlo

```
val viewModel = ViewModelProvider( owner: this)[LoginViewModel::class.java]
```

2-Al clickar el botón de login utilizaremos una funcion del ViewModel que comprobara si el usuario existe en el json

```
viewModel.checkUser(username,pass)
```

3-Observaremos un MutableLiveData llamado isLoading y cuando sea True mostraremos la ruedecita

```
viewModel.isLoadingLD.observe(viewLifecycleOwner){isLoading->
    if(isLoading){
        binding.progressBar.visibility = View.VISIBLE
    }else{
        binding.progressBar.visibility = View.GONE
    }
}
```

4-También observaremos un MutableLiveData llamada UserLD, a esta variable en el viewModel cuando ejecutamos el checkUser(lo he comentado anteiormente) si el usuario existia le hacia un Post ,entonces lo que ocurre en este código es que si el usuario existia ,hacemos un Intent pasandole el id del usuario en cuestión,si no,muestra un toast diciendo que el usuario no existe

```
    viewModel.userLD.observe(viewLifecycleOwner){user->
        user?.let { it: Login.User
            val i : Intent = Intent(requireActivity(),HomeActivity::class.java)
            i.putExtra(HomeActivity.ID_PARAM, user.id)
            requireActivity().startActivity(i)
        } ?: run{ this: LoginFragment
            Snackbar.make(binding.root, text: "User doesn't exist", Snackbar.LENGTH_SHORT).show()
        }
    }
}
```

Para resumir,Intent con id de usuario y un atributo de HomeActivity y vamos a HomeActivity.

5-Antes de ir al HomeActivity vamos a ver la Clase Login.kt,esta clase basicamente es el modelo sobre el que trabaja viewmodel,es decir,la clase que realmente tiene la funcion

```
var login : Login = Login()
```

Esto es en el viewmodel

6-Lo primero es que dentro tenemos la Data class de los usuarios con su id identificativo ,su nombre y su contraseña.

```
data class User(
    val id:Int,
    val username:String,
    val pass:String
)
```

7-Tenemos un Companion object(se comparte entre instancias) de la lista de usuarios

```
companion object{
    var listUser: List<User> = listOf()
}
```

8-Ahora tenemos una de las funciones principales, leer los datos del Json,en la propia imagen ya estan explicados los pasos mas importantes

```
fun loadFromJSON(context: Context):Boolean{
    var userList:List<User>? //creamos lista usuarios donde caera la información
    val raw = context.resources.openRawResource(R.raw.users) //json
    val rd = BufferedReader(InputStreamReader(raw))//leemos json con buffered

    val listType: Type = object : TypeToken<MutableList<User?>?>() {}.type   //

    val gson = Gson() //Inicializamos Gson
    userList = gson.fromJson(rd, listType) //utilizando TypeToken y el rd le pasamos los usuarios a la lista

    Thread.sleep(waitTime)

    userList?.let {  it: List<User>
        listUser = it // la lista de usuarios de companion object es esta
    }

    return userList!!.isNotEmpty() // devolvemos la lista si no esta vacia
}
```

9-Aquí tenemos el metodo que realmente se encarga de checkear los Users

```
//onLogin y onLoading son las dos variables observadas desde el viewModel
suspend fun checkUser(user:User, onLogin: OnLogin, onLoading: OnLoading){
    onLoading( loading: true)//Se pone en True para que muestre la ruedecita
    delay(waitTime)//Se pone un tiempo para que se muestre la rueda mas tiempo
    val list = listUser.filter { //Creamos una lista asignandole un filtro  diciendole que el usu/pass debe ser igual que el que le pasamos
        it.username == user.username && it.pass == user.pass
    }

    if(list.isEmpty()) onLogin( user: null)//si esta vacia le dices que no hay login
    else
        onLogin(list[0])//si tiene uno le das el primero

    onLoading( loading: false)//para de girar la rueda
}
```

10-Aquí tenemos algo similar ,básicamente filtramos por id

```
suspend fun getUserById(id:Int,onLogin: OnLogin, onLoading: OnLoading){
    onLoading( loading: true) //Mostramos rueda
    delay(waitTime)//esperamos
    //Filtro por id
    val list = listUser.filter { user->
        user.id == id
    }

    // el mismo rollo de la lista
    if(list.isEmpty()) onLogin( user: null)
    else
        onLogin(list[0])

    onLoading( loading: false)
}
```

1-Ahora vamos a crear el HomeActivity,primero vamos a ver su codigo.

2-Lo primero es declarar el binding y el appBar(recuerda que es lo de Navigation y hay distintos)

```
private lateinit var binding: ActivityHomeBinding

private lateinit var appBarConfiguration:AppBarConfiguration
```

3-Ahora creamos una constante dentro de un companion object (para que sea lo mismo en todas las instancias de la clase)

```
companion object{
    const val ID_PARAM:String ="ID_PARAM"
}
```

4-Ahora inflamos el binding

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(ActivityHomeBinding.inflate(layoutInflater).also {  it: ActivityHomeBinding
        binding = it
    }.root)
```

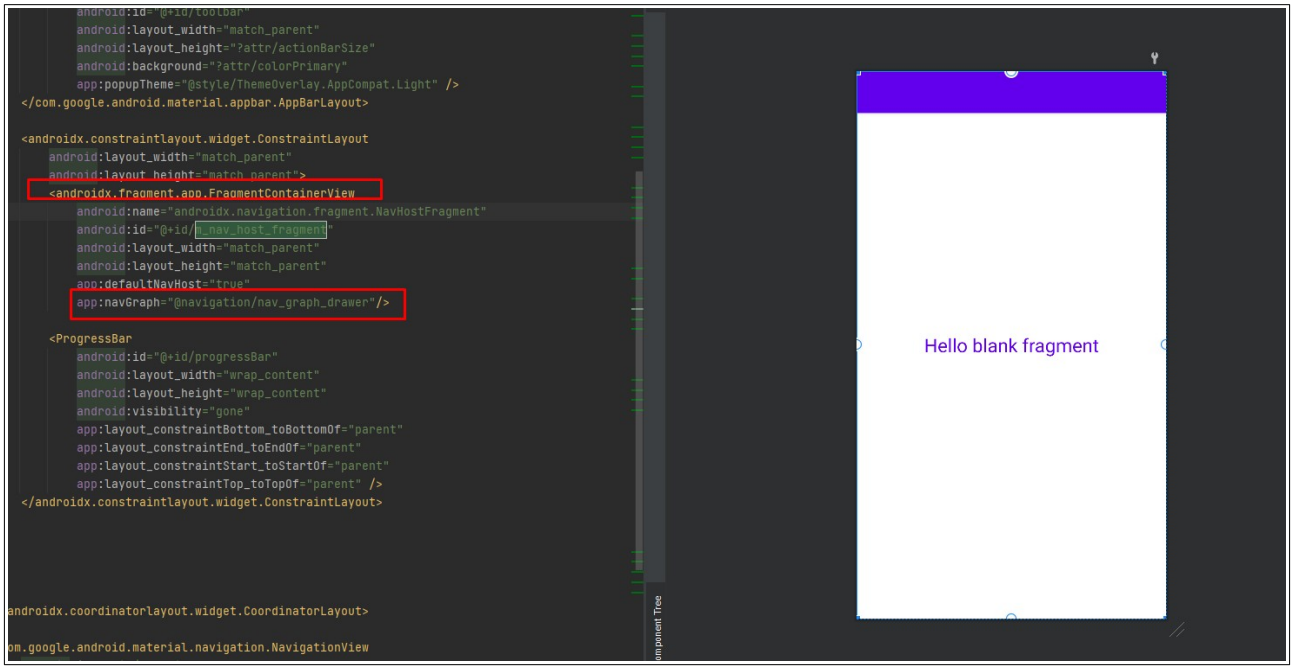
5-Le cambiamos el titulo al toolbar que habiamos creado y lo ponemos con setSuppor...

```
setSupportActionBar(binding.toolbar)
binding.toolbar.title = "Home"
```

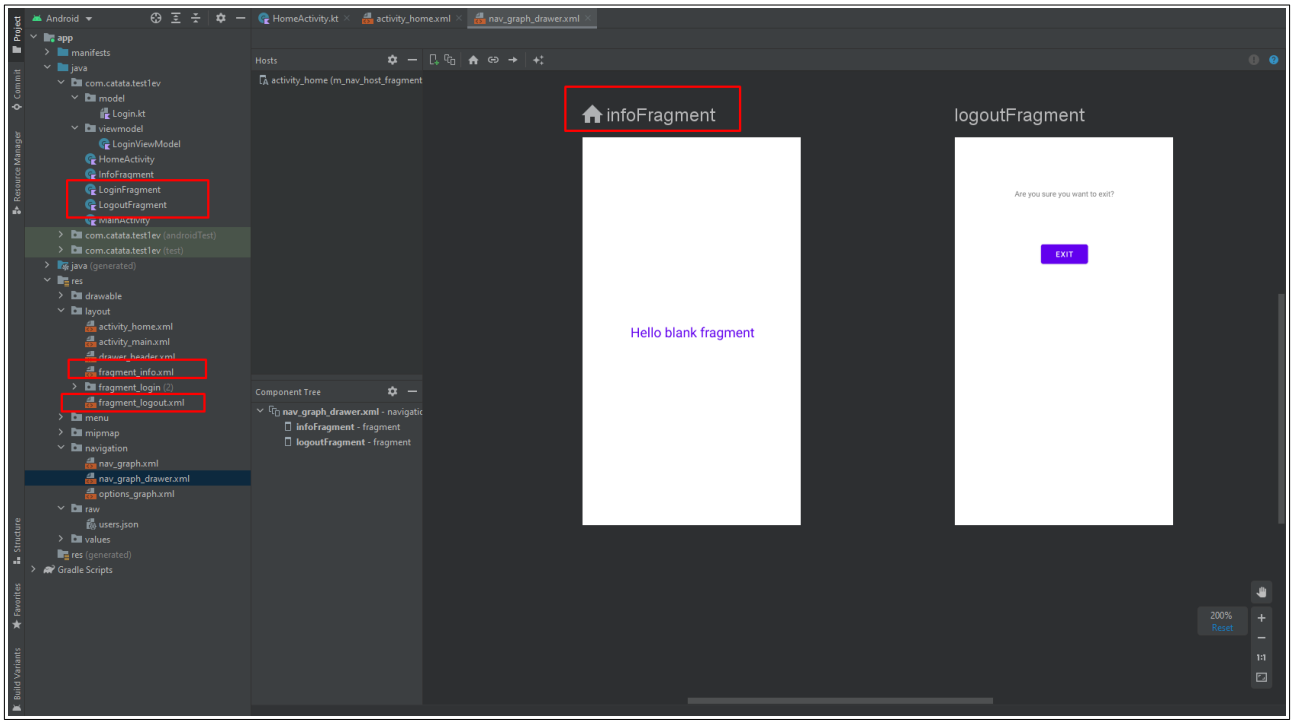
6-Ahora tenemos explicado todo lo referente al drawer menu

```
val drawerLayout: DrawerLayout = binding.drawerLayout //DONDE ESTA TODO
val navView: NavigationView = binding.navView //navView el de la izquierda

//Hace referencia a un contenedor del xml de home que tiene dentro el nav_graph de la ventana de info y el logout
val navHosFragment = supportFragmentManager.findFragmentById(R.id.m_nav_host_fragment) as NavHostFragment
val navController = navHosFragment.navController//cogemos el controllador de ese contenedor

//trabajamos con su appbar
appBarConfiguration = AppBarConfiguration.Builder(navController.graph)
    .setOpenableLayout(drawerLayout)
    .build()

//añadimos los fragmentos de destino
appBarConfiguration.topLevelDestinations.add(R.id.infoFragment)
appBarConfiguration.topLevelDestinations.add(R.id.logoutFragment)

//añadimos ese controlador al lavView
NavigationUI.setupWithNavController(navView, navController)
//añadimos el rstp
NavigationUI.setupWithNavController(binding.toolbar, navController, appBarConfiguration)
```

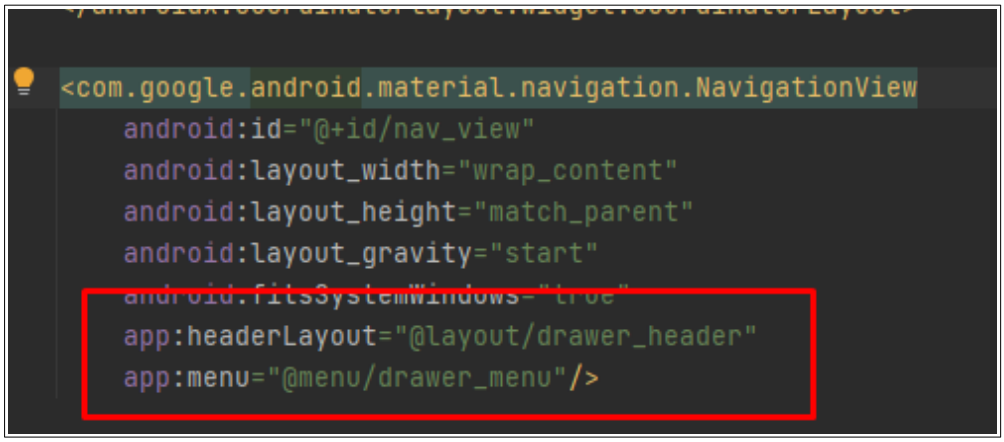
7-Ahora tenemos aquí el XML del homeactivity(esto se copia y au) el cual tiene pues eso,un drawer,la prograssbar,un appbar… y lo más importante, un contenedor con el nav_graph de info+logout

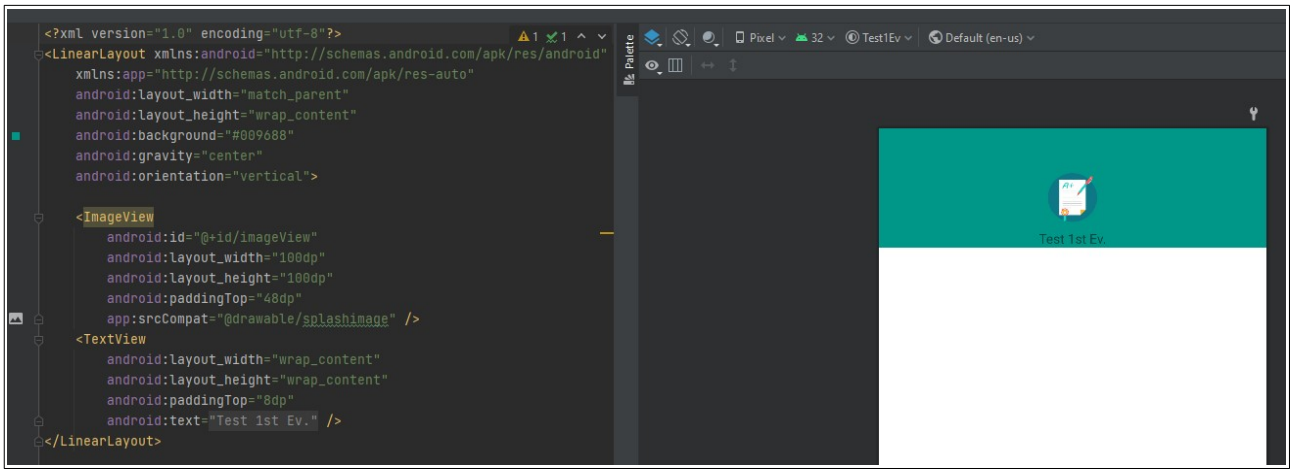


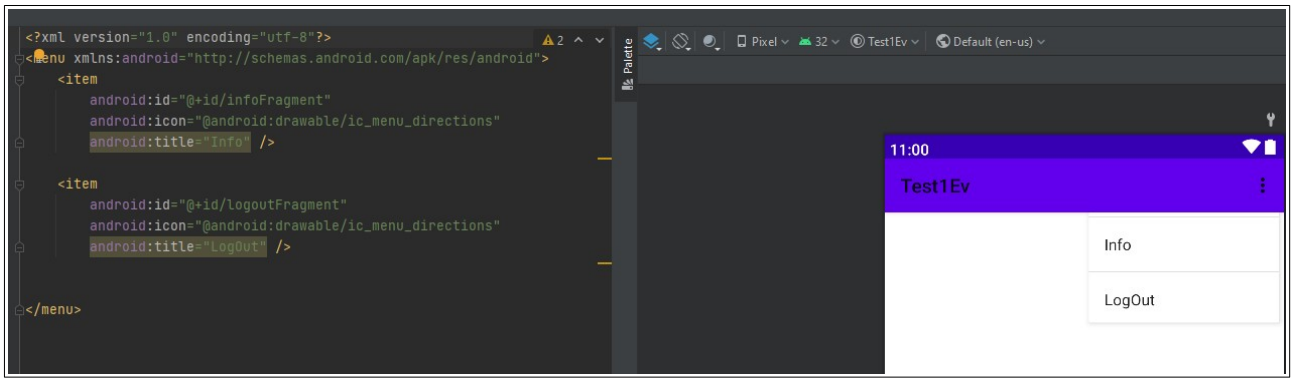
8-Aquí tenemos la estructura del navGraph,se puede ver que primero tenemos un texto el cual será odificado con los datos del usuario y que son fragmentos con su .kt y xml



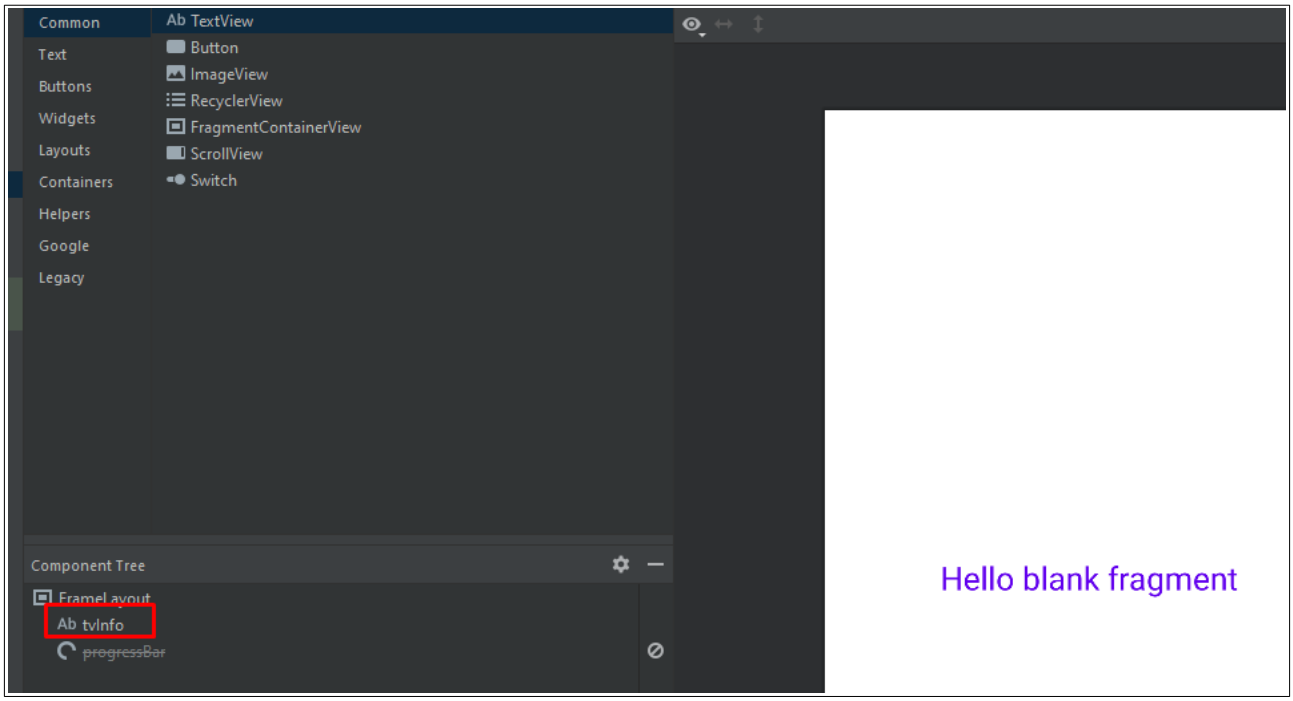
Antes de pasar al infofragment recordar que tenemos el navigation view con dis referecias,una al header y otra al menu ,allí se modificarán sus cosas







9-El XML de infofragment es muy simple,un textview y al carrer



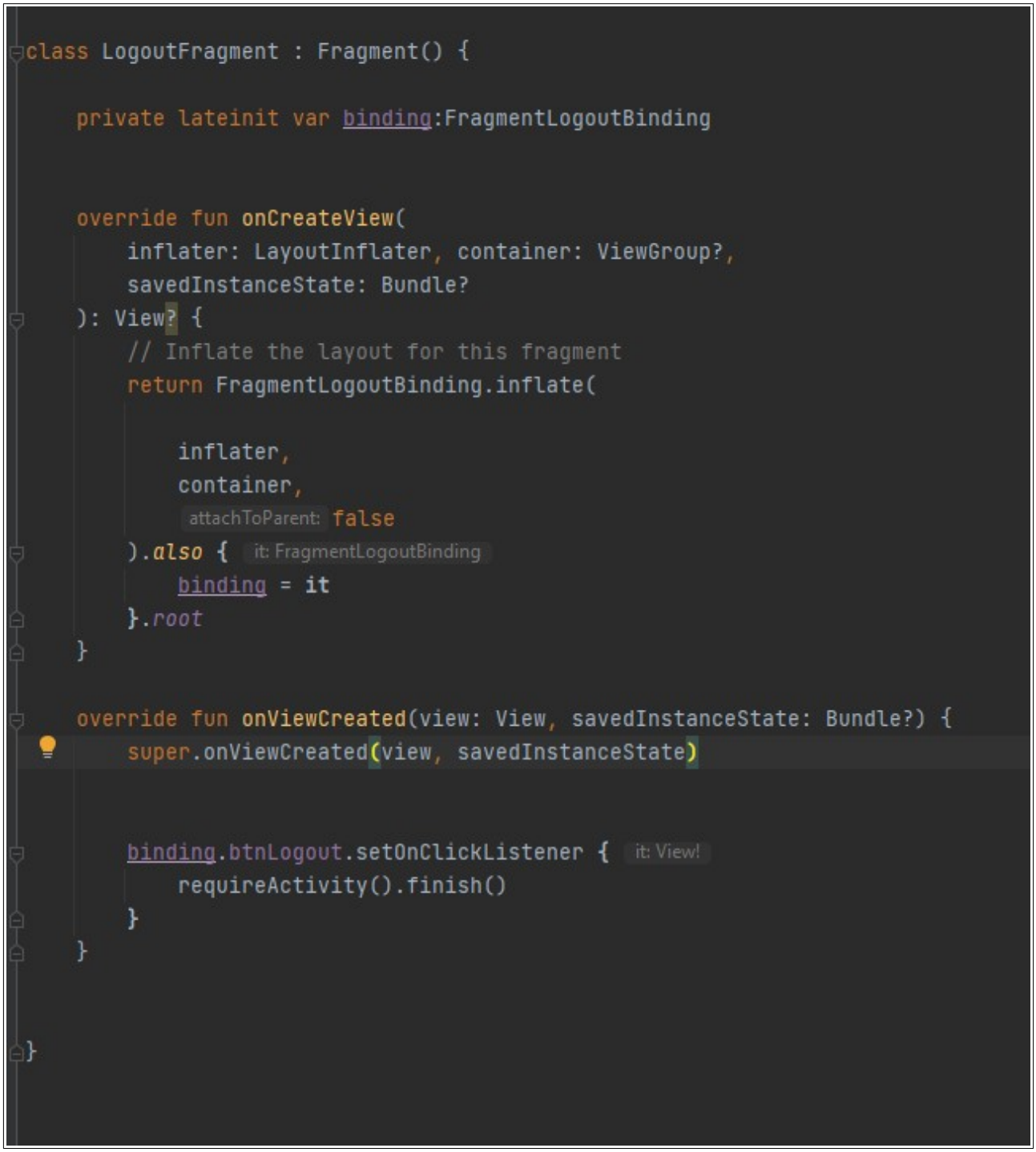
10-Primero tenemos el create view donde inflamos como siemprs

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    // Inflate the layout for this fragment
    return FragmentInfoBinding.inflate(
        inflater,
        container,
        attachToParent: false
    ).also { it: FragmentInfoBinding
        binding = it
    }.root
}
```

11-Después tenemos el onviewcreated el cual se encarga de mostrar los datos del usuario en el textview y muestrar la ruedita?

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    val viewModel = ViewModelProvider( owner: this)[LoginViewModel::class.java]

    //Cogemos la id pasada con el intent a  HomeActivity que se hace en LoginFragment
    //Esta utiliza el atributo ID_PARAM de HomeActivity
    val id = requireActivity().intent.getIntExtra(
        HomeActivity.ID_PARAM, defaultValue: -1
    )

    //cogemos el usuario por la id
    viewModel.getUserById(id)

    //y utilizamos un observ para cambiar el texto
    viewModel.userLD.observe(viewLifecycleOwner){
        user ->
        user?.let { it: Login.User
            binding.tvInfo.text = "User: ${it.username}\nWith password: ${it.pass}"
        }
    }

    //esto es pa la ruedita
    viewModel.isLoadingLD.observe(viewLifecycleOwner){
            isLoading ->
        if(isLoading)
            binding.progressBar.visibility = View.VISIBLE
        else
            binding.progressBar.visibility = View.GONE
    }
}
```

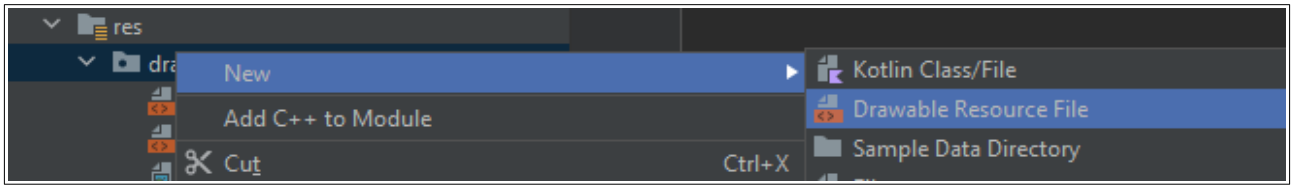
12-El logOut en si solo tiene un boton asi que solo mostraremos el código en el cual lo unico que hay es un binding y un finish de la activity en el boton

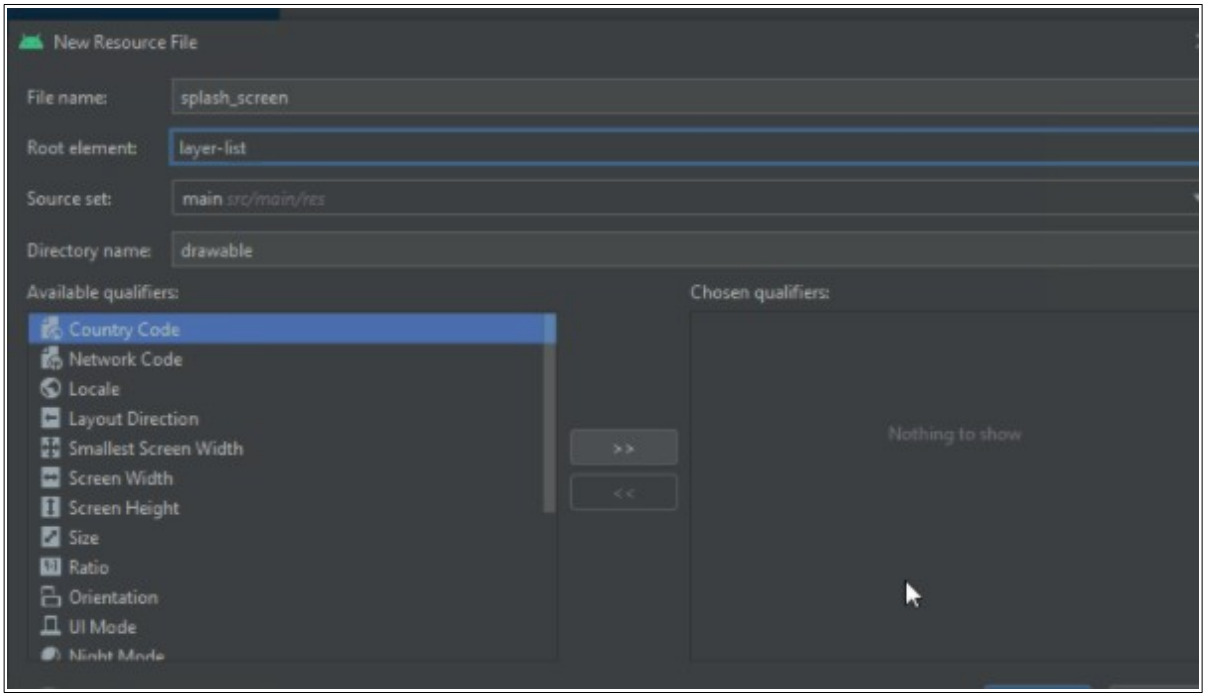


```
class LogoutFragment : Fragment() {

    private lateinit var binding:FragmentLogoutBinding


    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return FragmentLogoutBinding.inflate(

            inflater,
            container,
            attachToParent: false
        ).also {  it: FragmentLogoutBinding
            binding = it
        }.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)


        binding.btnLogout.setOnClickListener {  it: View!
            requireActivity().finish()
        }
    }


}
```

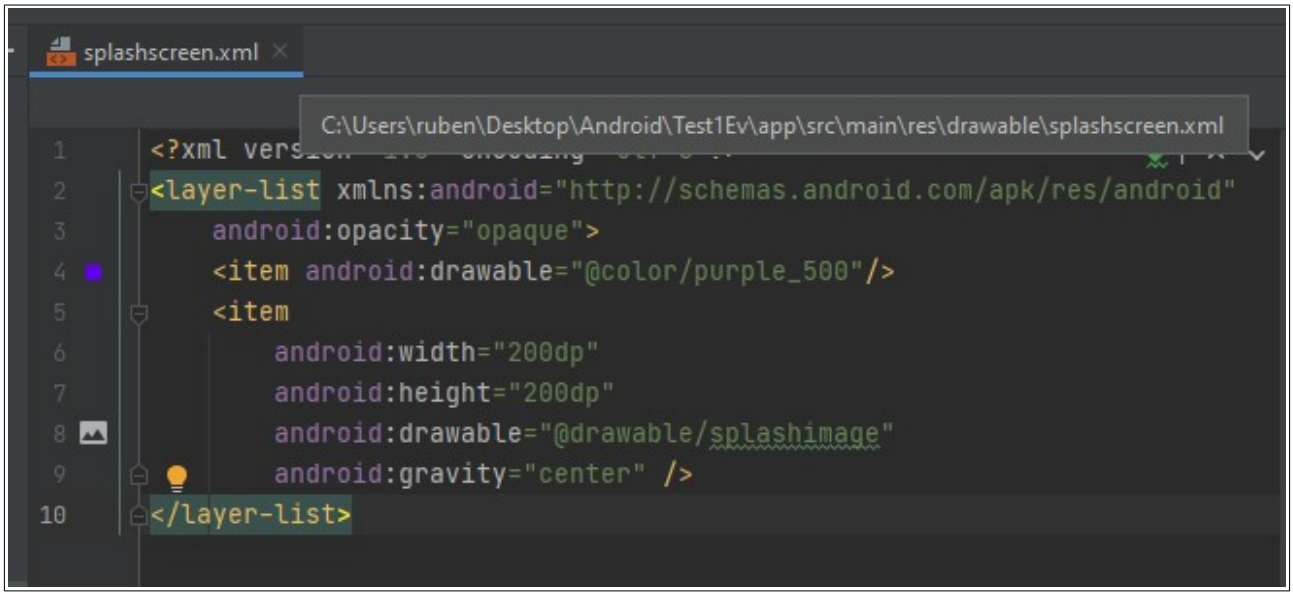
SPLASH SCREEN

1-Lo primero es en <drawable> crear un New → Drawable Resource file

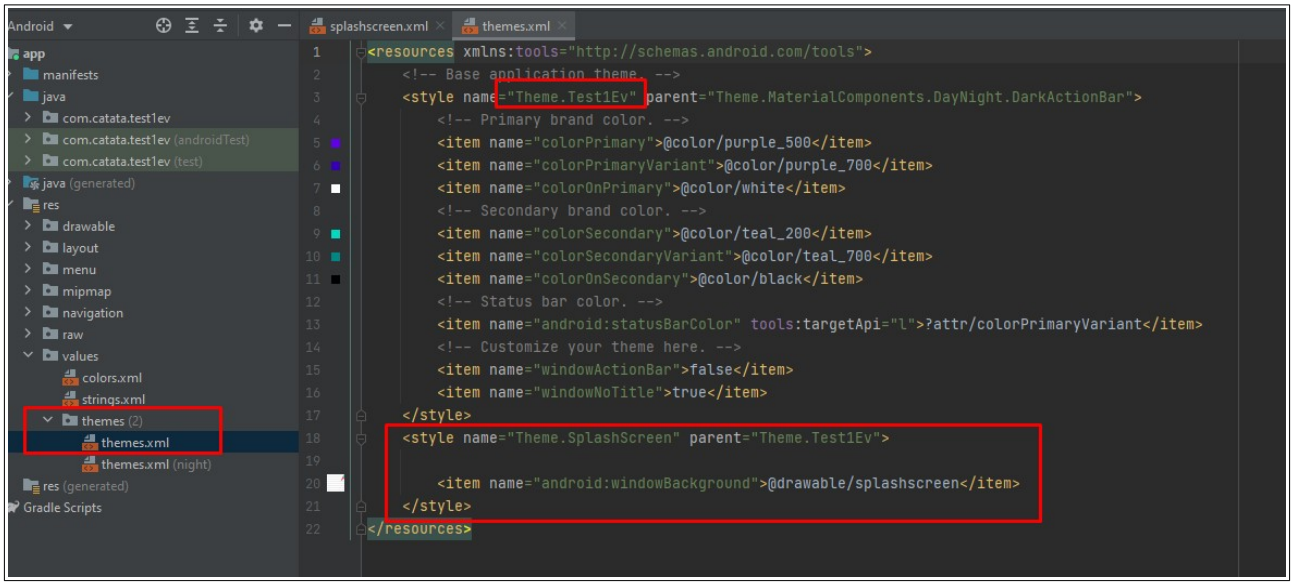




2-Dentro tiene el siguiente código,el cual referencia a una img,esto se copia y pega y au



3-Ahora crearemos un style donde pondremos como fondo el splash_screen



4-Ahora en el manifest pondremos el tema que acabamos de crearemos

```
    package="com.catata.test1ev">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Test1Ev"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Test1Ev">
        <activity
            android:name=".Home2Activity"
            android:exported="false" />
        <activity
            android:name=".HomeActivity"
            android:exported="false" />
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:theme="@style/Theme.SplashScreen">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

5-Y ahora en el MainAcitivty pondremos el tema normal