PROYECTO: INTERFAZ DE USUARIO 1

Nombre: Alejandro Morcillo Muriel
Fecha: 29/05/24
github: https://github.com/ales5muriel/ThreadExchange-1.git

INDICE

Trasladar el contenido de la “MainActivity” y del “CreditActivity” en dos fragmentos que deben permitir la técnica de “view binding”. Desde este momento todos los fragmentos que se creen deben usar el “view binding”. (0,5)

1. Crear un sistema de navegación de modo que la aplicación se inicia con el “SplashActivity” (queda fuera de la navegación) y a continuación viaja al “MainActivity”, que por defecto carga el contenido del fragmento “LoginFragment”. En el fragmento de Login se podrá introducir el nombre del usuario y llegar a un fragmento de Menú (MenuFragment), desde este fragmento se podrá navegar hasta el de créditos y viceversa. Desde el MenuFragment se podrá navegar hasta el de Login si se pulsa en el botón de “Salir” (que debe crearse). (1)

2. Realizar pruebas de uso en el emulador, indicando si encuentra fallos y sus soluciones. (0,25)

Después de ver la forma de crear temas y estilos con el uso de “Material Design” del Android Jetpack es el momento de crear un tema completamente personalizado para la aplicación. En este tema se debe: (CE 2B)

1. Personalizar completamente los colores de la aplicación, tanto para el modo estándar como para el oscuro. (1)

2. Incorporar una o varias fuentes de datos para la aplicación. (0,25)

3. Creación de estilos personalizados, al menos para los siguientes elementos: (0,75)

1. Textos.
2. Títulos.
3. Botones.

Tras haber estudiado los diferentes “Layouts” y algunos de sus elementos auxiliares como las ventanas deslizantes (ViewPager2) o las tarjetas (CardView) es el momento de crear los diferentes fragmentos que va a contener la aplicación: (CE 2A y CE 2B)

1. Los fragmentos a crear son:

1. ItemListFragment: Fragmento que contendrá la lista de elementos de los que trata la aplicación. Debe contener un “botón” para incluir el elemento en “Favoritos”. (1)
2. DetailItemFragment: Fragmento con información más detallada de un elemento de los que trata la información. (1)
3. UserInfoFragment: Fragmento con información del usuario que se encuentra en la aplicación. (1)
4. FavItemListFragment: Fragmento con la lista de elementos favoritos del usuario. Debe tener un botón para eliminar el elemento de la lista de favoritos. (0,5)
5. DetailFavItemFragment: Fragmento con información extra que el usuario puede añadir sobre los elementos de la aplicación, como por ejemplo sus anotaciones personales, su puntuación, etc. En este fragmento debe existir un botón flotante para incluir nuevos comentarios “privados” en este ítem. (0,5)

Se deben crear diseños verticales de todos los fragmentos y diseños apaisados de los fragmentos de detalles.

2. Transforma los “LinearLayout” de los fragmentos que estaban ya creados con anterioridad en “ConstraintLayout”. (0,25)

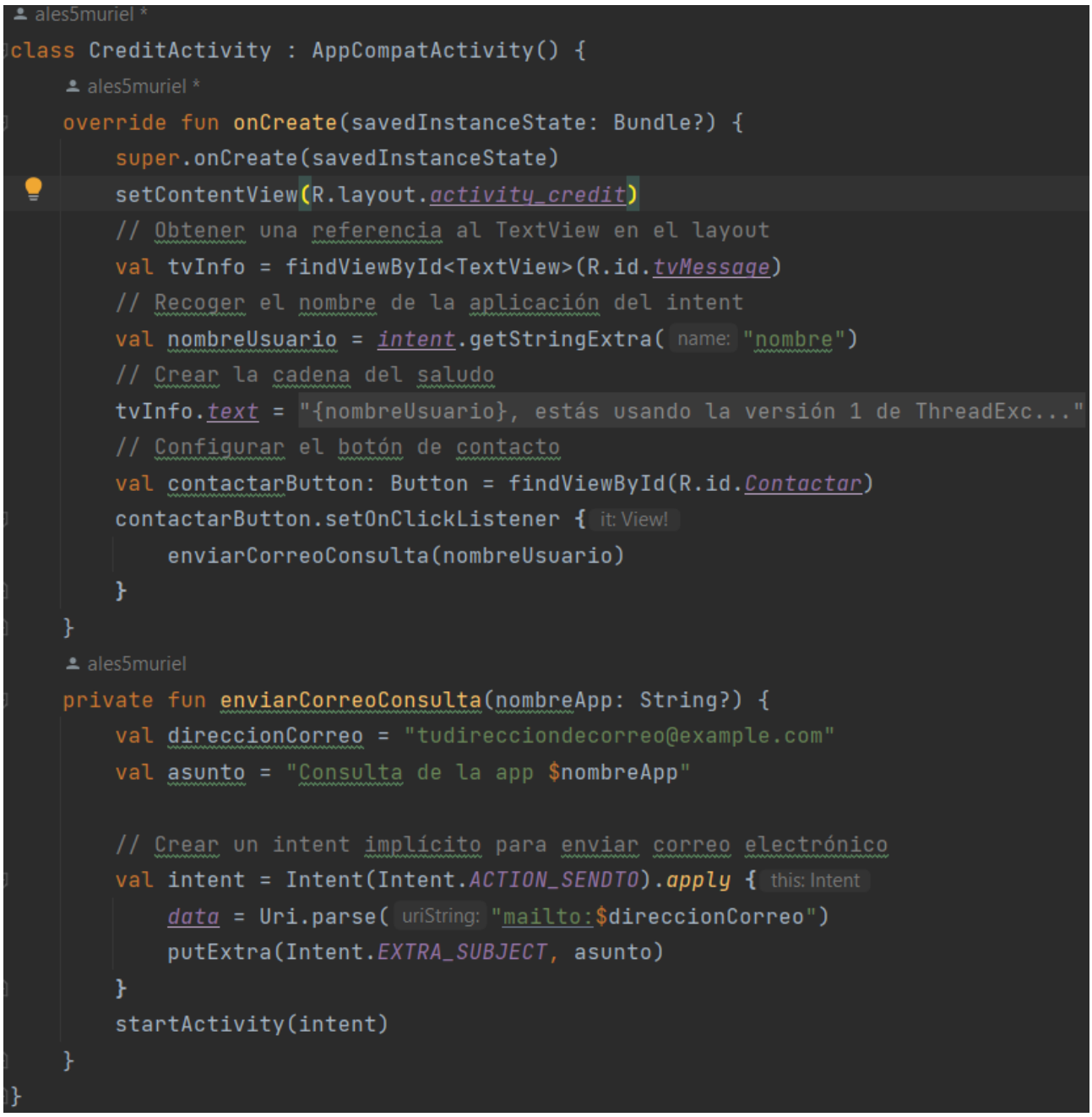
3. Crea un par de pestañas deslizantes (ViewPager2) con información sobre la aplicación que quieras resaltar. En la segunda pestaña debe existir el botón “Comenzar”. Estas pestañas deslizantes se incluirán en la lógica de navegación tras la inclusión del usuario (LoginFragment → Pestañas ViewPager2) y al pulsar sobre el botón “Comenzar” navegará al menú principal (Pestaña ViewPager2 → MenuFragment). Si desde el menú principal se pulsa en “Salir”, volvería directamente al fragmento de Login (MenuFragment → LoginFragment). (1)

4. Crear la navegación entre todos los fragmentos creados. La información que se muestran en los fragmentos se debe crear desde objetos incorporados y creados dentro de la lógica de la aplicación, en las propias clases o en un objeto (Singleton) que proporcione estos datos. En la próxima unidad, se tomarán los datos desde otras fuentes de datos. (0,5)

5. Realizar pruebas de uso en el emulador, indicando si encuentra fallos y sus soluciones. (0,5)

Trasladar el contenido de la “MainActivity” y del “CreditActivity” en dos fragmentos que deben permitir la técnica de “view binding”. Desde este momento todos los fragmentos que se creen deben usar el “view binding”. (0,5)

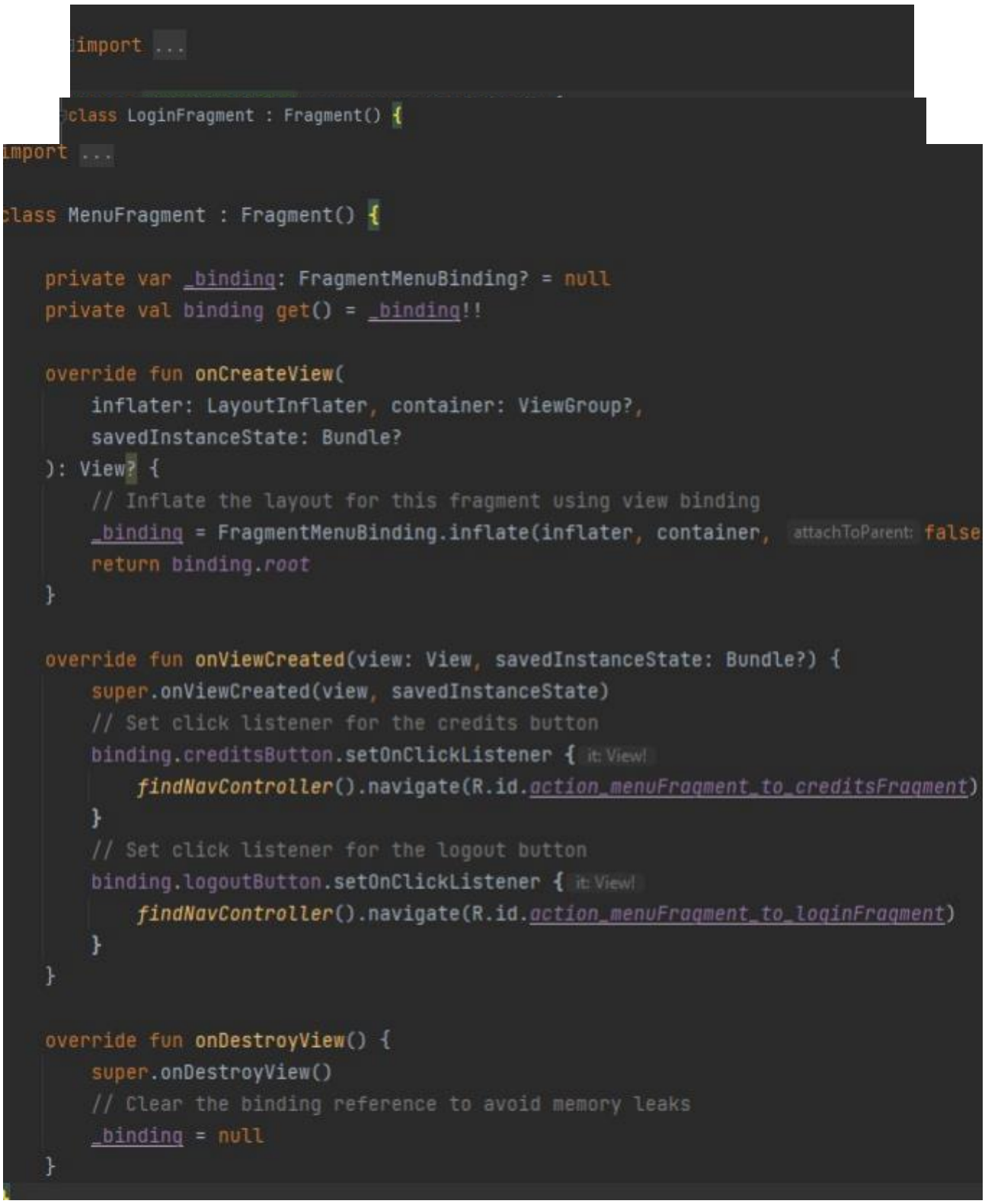
```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    ales5muriel *
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val navHostFragment =
            supportFragmentManager.findFragmentById(R.id.navHostFragment) as NavHostFragment
        val navController = navHostFragment.navController

        navController.addOnDestinationChangedListener { _, destination, _  ->
            if(destination.id == R.id.login2Fragment) {
                binding.bottomNav.visibility = View.GONE
            } else {
                binding.bottomNav.visibility = View.VISIBLE
            }
        }
        setContentView( binding.root)
        binding.bottomNav.setupWithNavController(navController)
        NavigationUI.setupActionBarWithNavController( activity: this, navController)
    }
    new *
    override fun onSupportNavigateUp(): Boolean {
        val navController = this.findNavController(R.id.navHostFragment)
        return navController.navigateUp()
    }
}
```



```
ales5muriel *
class CreditActivity : AppCompatActivity() {
    ales5muriel *
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_credit)
        // Obtener una referencia al TextView en el layout
        val tvInfo = findViewById<TextView>(R.id.tvMessage)
        // Recoger el nombre de la aplicación del intent
        val nombreUsuario = intent.getStringExtra( name: "nombre")
        // Crear la cadena del saludo
        tvInfo.text = "{nombreUsuario}, estás usando la versión 1 de ThreadExc..."
        // Configurar el botón de contacto
        val contactarButton: Button = findViewById(R.id.Contactar)
        contactarButton.setOnClickListener { it: View!
            enviarCorreoConsulta(nombreUsuario)
        }
    }
    ales5muriel
    private fun enviarCorreoConsulta(nombreApp: String?) {
        val direccionCorreo = "tudirecciondecorreo@example.com"
        val asunto = "Consulta de la app $nombreApp"

        // Crear un intent implícito para enviar correo electrónico
        val intent = Intent(Intent.ACTION_SENDTO).apply { this: Intent
            data = Uri.parse( uriString: "mailto:$direccionCorreo")
            putExtra(Intent.EXTRA_SUBJECT, asunto)
        }
        startActivity(intent)
    }
}
```

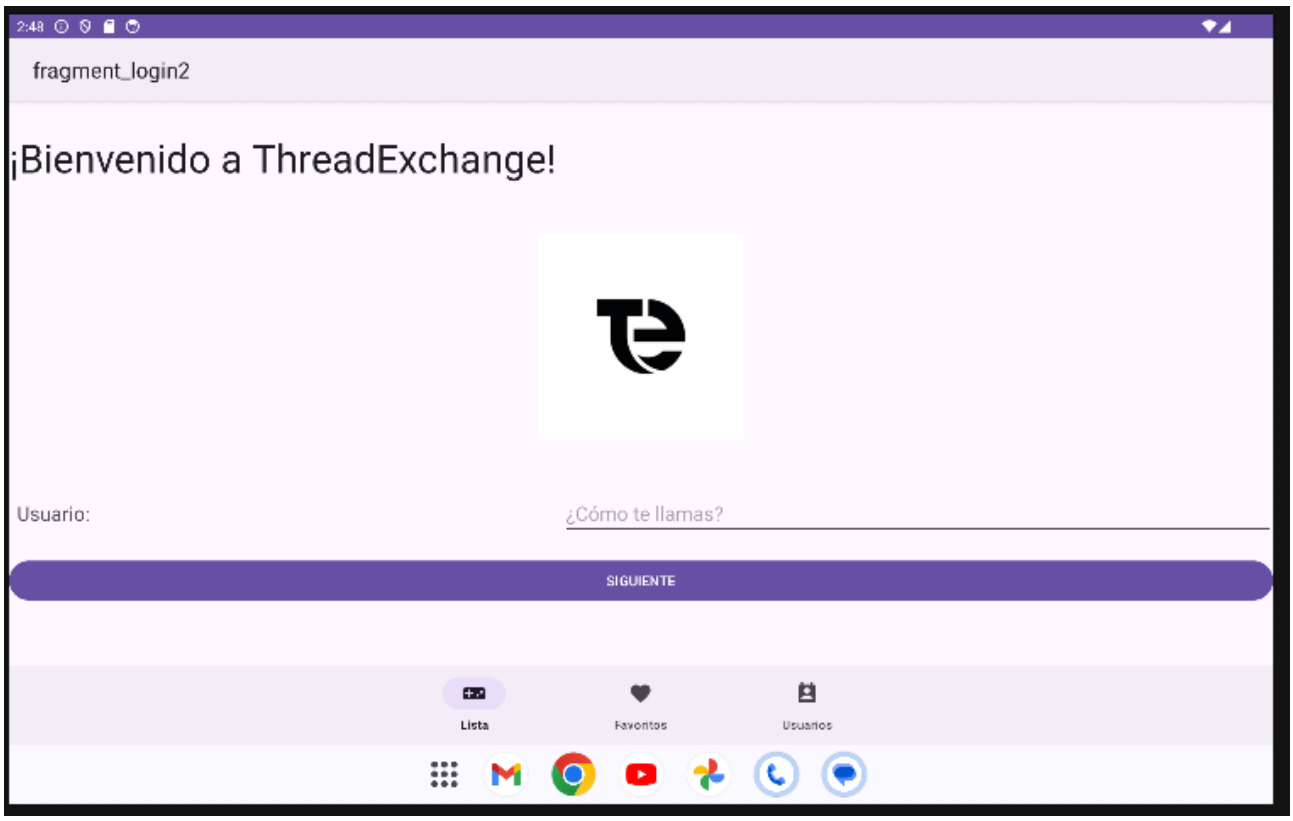
De esta forma hemos conseguido pasar el contenido del MainActivity y CreditActivity a fragmentos en los que hemos podido utilizar la técnica de view binding.

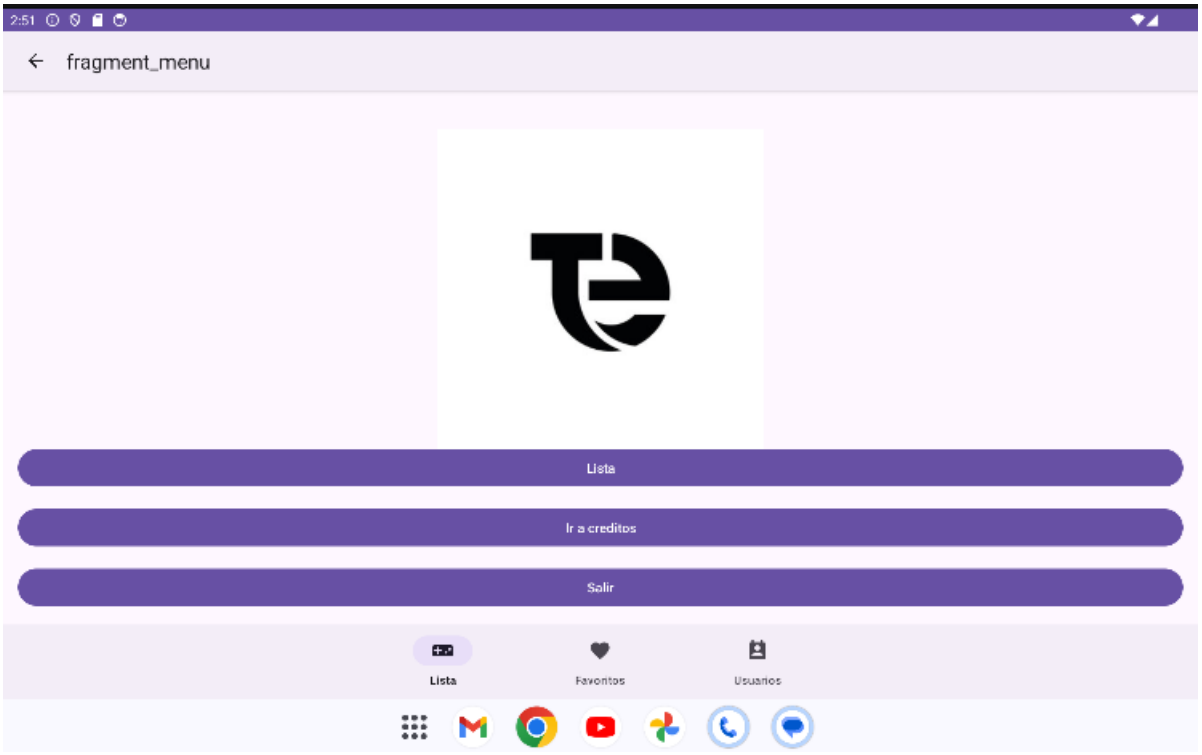
1. Crear un sistema de navegación de modo que la aplicación se inicia con el “SplashActivity” (queda fuera de la navegación) y a continuación viaja al “MainActivity”, que por defecto carga el contenido del fragmento “LoginFragment”. En el fragmento de Login se podrá introducir el nombre del usuario y llegar a un fragmento de Menú (MenuFragment), desde este fragmento se podrá navegar hasta el de créditos y viceversa. Desde el MenuFragment se podrá navegar hasta el de Login si se pulsa en el botón de “Salir” (que debe crearse). (1)



```
import ...

class LoginFragment : Fragment() {
import ...

class MenuFragment : Fragment() {

    private var _binding: FragmentMenuBinding? = null
    private val binding get() = _binding!!

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

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        // Set click listener for the credits button
        binding.creditsButton.setOnClickListener { it: View!
            findNavController().navigate(R.id.action_menuFragment_to_creditsFragment)
        }
        // Set click listener for the logout button
        binding.logoutButton.setOnClickListener { it: View!
            findNavController().navigate(R.id.action_menuFragment_to_loginFragment)
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        // Clear the binding reference to avoid memory leaks
        _binding = null
    }
```

2. Realizar pruebas de uso en el emulador, indicando si encuentra fallos y sus soluciones.


2:48
fragment_login2
¡Bienvenido a ThreadExchange!
Usuario:
¿Cómo te llamas?
SIGUIENTE
Lista
Favoritos
Usuarios


2:51
fragment_menu
Lista
Ir a creditos
Salir
Lista
Favoritos
Usuarios

2. Después de ver la forma de crear temas y estilos con el uso de “Material Design” del Android Jetpack es el momento de crear un tema completamente personalizado para la aplicación. En este tema se debe: (CE 2B)

1. Personalizar completamente los colores de la aplicación, tanto para el modo estándar como para el oscuro. (1)

```
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="blue">#2196F3</color>
    <color name="purple">#673AB7</color>

    <!-- Modo estandar -->
    <color name="md_theme_primary">#874B6C</color>
    <color name="md_theme_onPrimary">#FFFFFF</color>
    <color name="md_theme_primaryContainer">#FFD8E9</color>
    <color name="md_theme_onPrimaryContainer">#380726</color>
    <color name="md_theme_secondary">#725763</color>
    <color name="md_theme_onSecondary">#FFFFFF</color>
    <color name="md_theme_secondaryContainer">#FDD9E8</color>
    <color name="md_theme_onSecondaryContainer">#291520</color>
    <color name="md_theme_tertiary">#7F543B</color>
    <color name="md_theme_onTertiary">#FFFFFF</color>
    <color name="md_theme_tertiaryContainer">#FFDBC9</color>
    <color name="md_theme_onTertiaryContainer">#311302</color>
    <color name="md_theme_error">#BA1A1A</color>
    <color name="md_theme_onError">#FFFFFF</color>
    <color name="md_theme_errorContainer">#FFDAD6</color>
    <color name="md_theme_onErrorContainer">#410002</color>
    <color name="md_theme_background">#FFF8F8</color>
    <color name="md_theme_onBackground">#21191D</color>
    <color name="md_theme_surface">#FFF8F8</color>
    <color name="md_theme_onSurface">#21191D</color>
    <color name="md_theme_surfaceVariant">#F0DEE4</color>
    <color name="md_theme_onSurfaceVariant">#504349</color>
```

```
<!-- Modo oscuro -->
<color name="md_theme_primary_highContrast">#400E2D</color>
<color name="md_theme_onPrimary_highContrast">#FFFFFF</color>
<color name="md_theme_primaryContainer_highContrast">#67304F</color>
<color name="md_theme_onPrimaryContainer_highContrast">#FFFFFF</color>
<color name="md_theme_secondary_highContrast">#311C27</color>
<color name="md_theme_onSecondary_highContrast">#FFFFFF</color>
<color name="md_theme_secondaryContainer_highContrast">#543C48</color>
<color name="md_theme_onSecondaryContainer_highContrast">#FFFFFF</color>
<color name="md_theme_tertiary_highContrast">#391A05</color>
<color name="md_theme_onTertiary_highContrast">#FFFFFF</color>
<color name="md_theme_tertiaryContainer_highContrast">#603A22</color>
<color name="md_theme_onTertiaryContainer_highContrast">#FFFFFF</color>
<color name="md_theme_error_highContrast">#4E0002</color>
<color name="md_theme_onError_highContrast">#FFFFFF</color>
<color name="md_theme_errorContainer_highContrast">#8C0009</color>
<color name="md_theme_onErrorContainer_highContrast">#FFFFFF</color>
<color name="md_theme_background_highContrast">#FFF8F8</color>
<color name="md_theme_onBackground_highContrast">#21191D</color>
<color name="md_theme_surface_highContrast">#FFF8F8</color>
<color name="md_theme_onSurface_highContrast">#000000</color>
<color name="md_theme_surfaceVariant_highContrast">#F0DEE4</color>
<color name="md_theme_onSurfaceVariant_highContrast">#2B2126</color>
<color name="md_theme_outline_highContrast">#4C4045</color>
<color name="md_theme_outlineVariant_highContrast">#4C4045</color>
<color name="md_theme_scrim_highContrast">#000000</color>
<color name="md_theme_inverseSurface_highContrast">#372E32</color>
```

2. Incorporar una o varias fuentes de datos para la aplicación. (0,25)

```
// Preferences DataStore (SharedPreferences like APIs)
dependencies { this: DependencyHandlerScope
    implementation("androidx.datastore:datastore-preferences:1.0.0")

    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.7.0")
    // optional - RxJava2 support
    implementation("androidx.datastore:datastore-preferences-rxjava2:1.0.0")

    // optional - RxJava3 support
    implementation("androidx.datastore:datastore-preferences-rxjava3:1.0.0")
}

// Alternatively - use the following artifact without an Android dependency.
dependencies { this: DependencyHandlerScope
    implementation("androidx.datastore:datastore-preferences-core:1.0.0")
}
```

En **Gradle scripts**, entramos en el apartado de **build.gradle.kts(Module:app)**, en el apartado de **depedencies** podemos añadir distintas fuentes de datos para nuestra aplicación.

3. Creación de estilos personalizados, al menos para los siguientes elementos: (0,75)

1. **Textos.**
2. **Títulos.**
3. **Botones.**

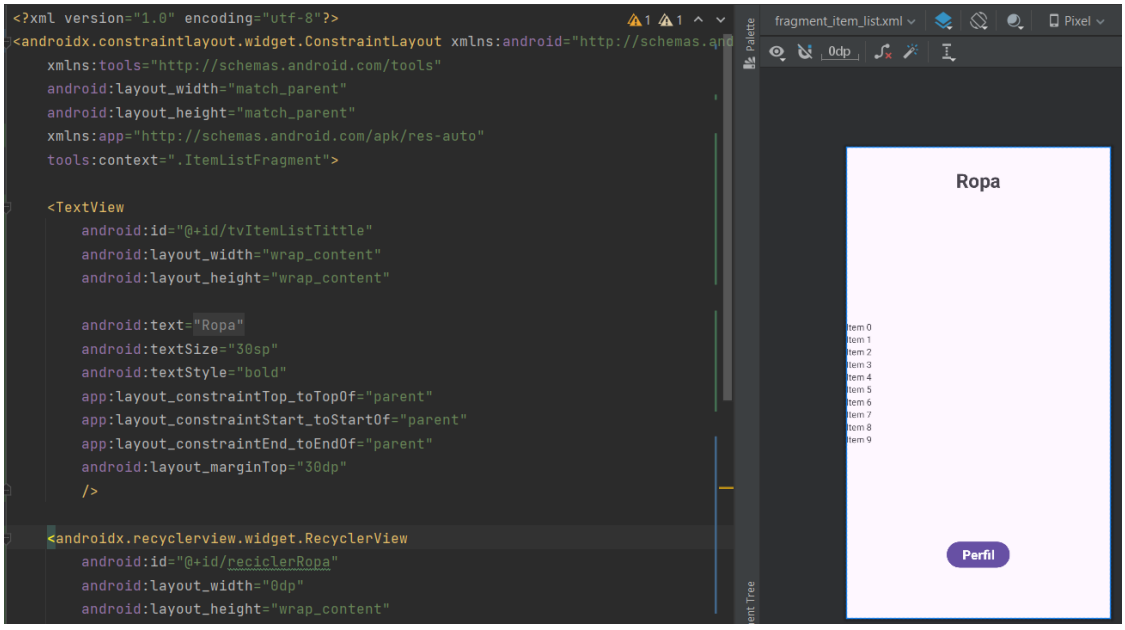
```xml
<resources>
    <!-- Estilo para textos -->
    <style name="AppTextStyle" parent="android:Widget.TextView">
        <item name="android:textColor">#FF0000</item> <!-- Color del texto -->
        <item name="android:textSize">16sp</item> <!-- Tamaño del texto -->
    </style>

    <!-- Estilo para títulos -->
    <style name="AppTitleStyle" parent="android:Widget.TextView">
        <item name="android:textColor">#0000FF</item> <!-- Color del título -->
        <item name="android:textSize">20sp</item> <!-- Tamaño del título -->
        <item name="android:textStyle">bold</item> <!-- Estilo del texto (negrita) -->
    </style>

    <!-- Estilo para botones -->
    <style name="AppButtonStyle" parent="android:Widget.Button">
        <item name="android:background">#9C27B0</item> <!-- Color de fondo del botón -->
        <item name="android:textColor">#FFFFFF</item> <!-- Color del texto del botón -->
    </style>
</resources>
```

En el apartado de **res→ values→ styles.xml.** Aqui podremos añadir los distintos estilos para nuestros botones, títulos y textos.

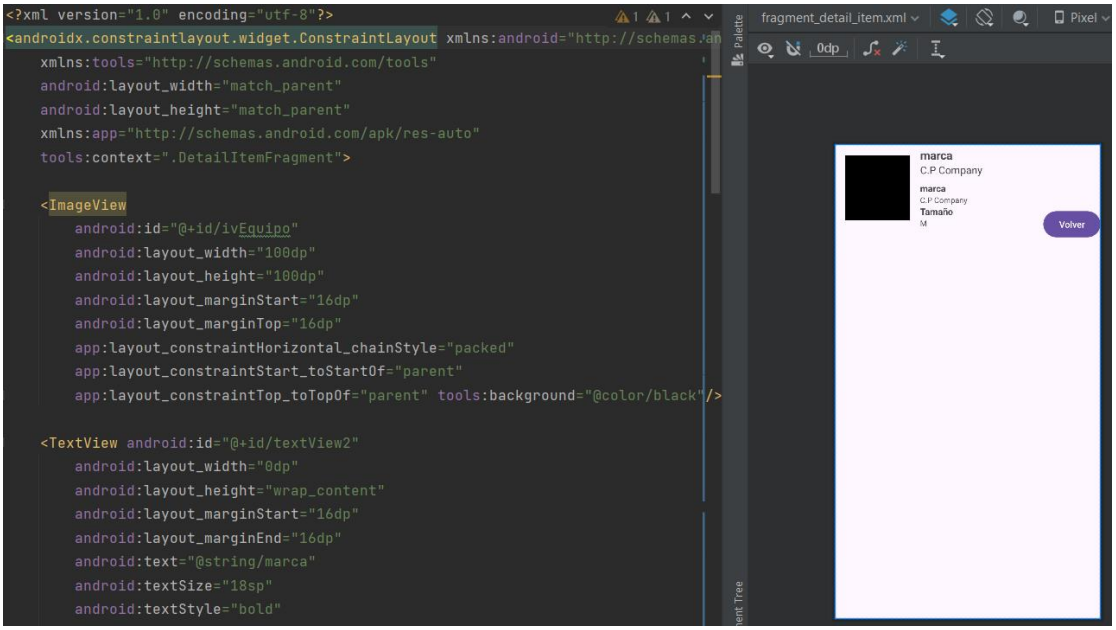
Tras haber estudiado los diferentes “Layouts” y algunos de sus elementos auxiliares como las ventanas deslizantes (ViewPager2) o las tarjetas (CardView) es el momento de crear los diferentes fragmentos que va a contener la aplicación: (CE 2A y CE 2B)

2. Los fragmentos a crear son:

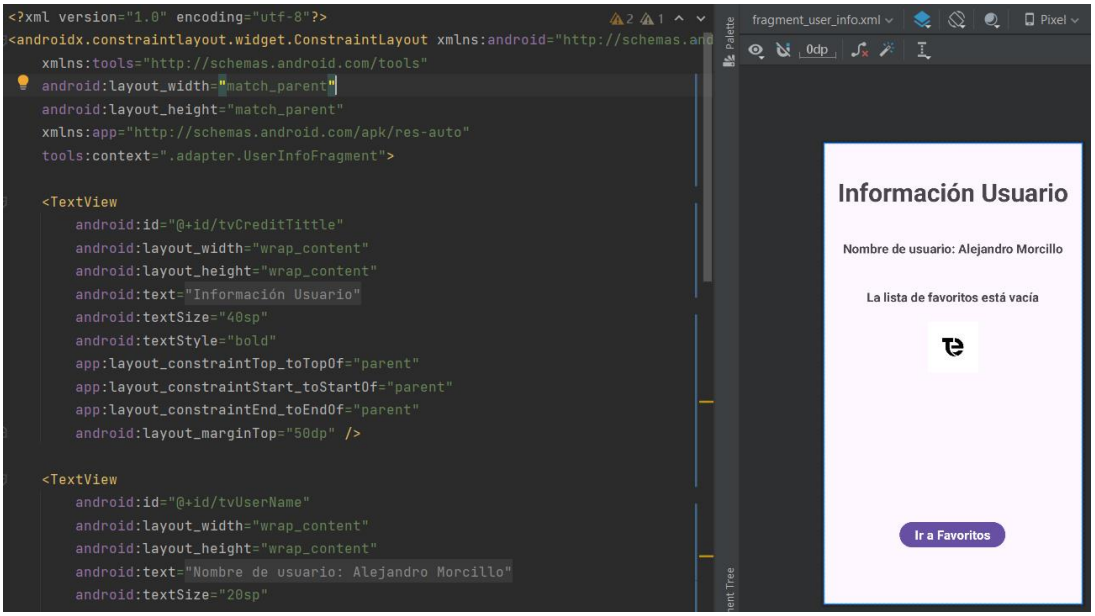
1. **ItemListFragment**: Fragmento que contendrá la lista de elementos de los que trata la aplicación. Debe contener un “botón” para incluir el elemento en “Favoritos”. (1)



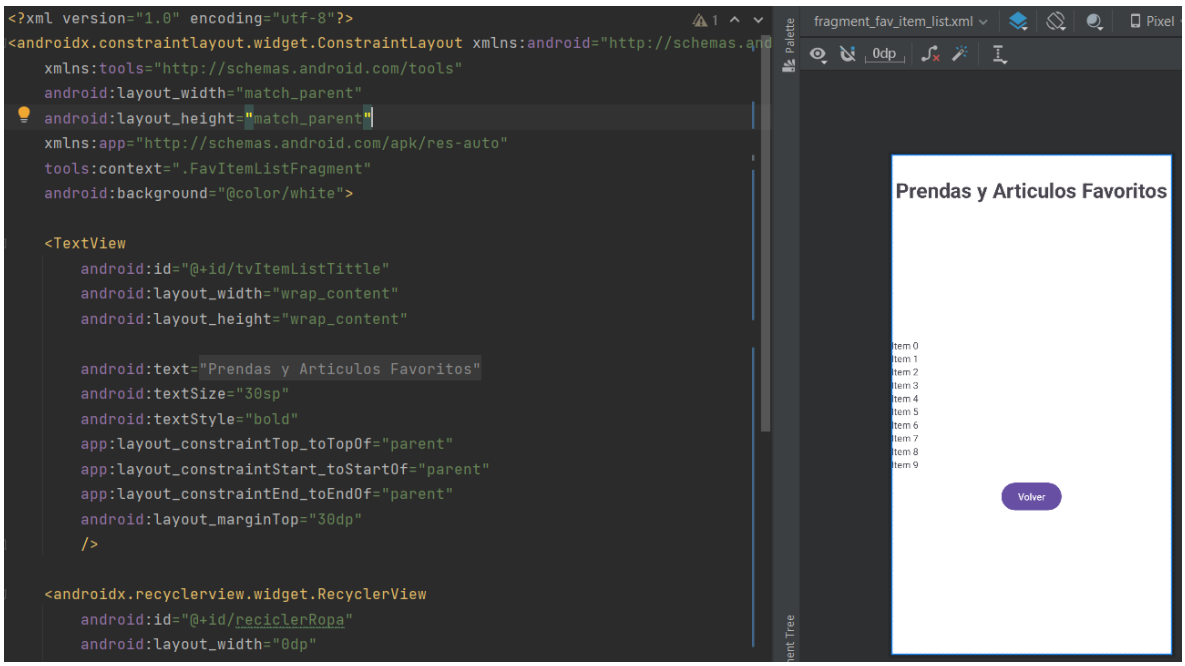
2. **DetailItemFragment**: Fragmento con información más detallada de un elemento de los que trata la información. (1)



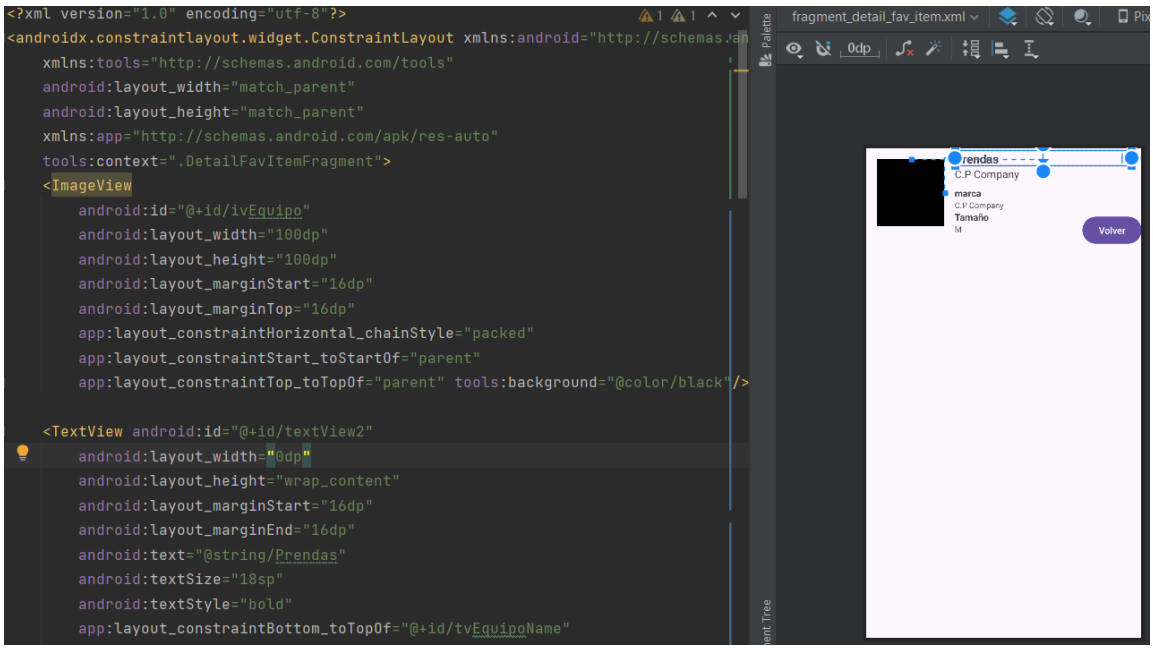
3. **UserInfoFragment**: Fragmento con información del usuario que se encuentra en la aplicación. (1)



4. **FavItemListFragment**: Fragmento con la lista de elementos favoritos del usuario. Debe tener un botón para eliminar el elemento de la lista de favoritos. (0,5)



5. **DetailFavItemFragment**: Fragmento con información extra que el usuario puede añadir sobre los elementos de la aplicación, como por ejemplo sus anotaciones personales, su puntuación, etc. En este fragmento debe existir un botón flotante para incluir nuevos comentarios "privados" en este ítem. (0,5)

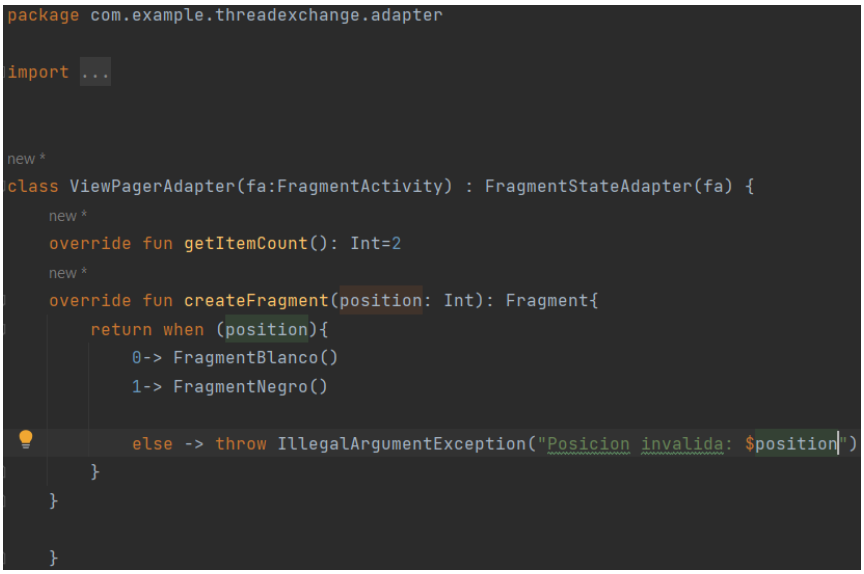


Se deben crear diseños verticales de todos los fragmentos y diseños apaisados de los fragmentos de detalles.

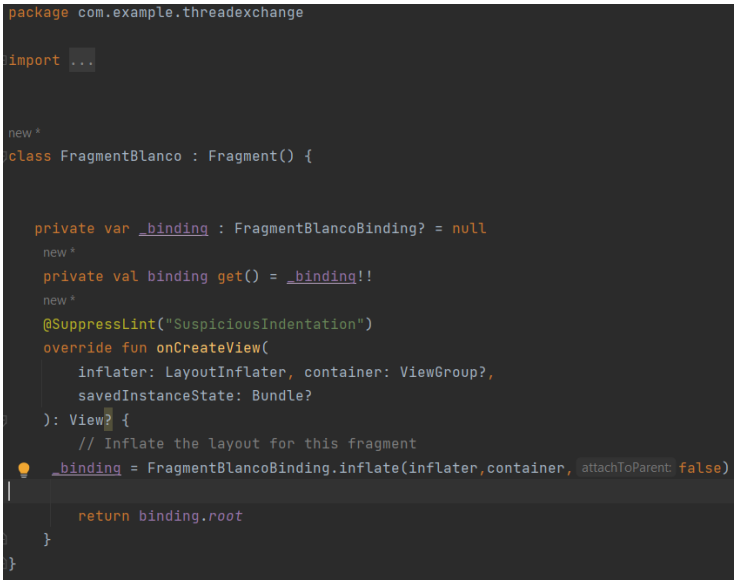
2. Transforma los "LinearLayout" de los fragmentos que estaban ya creados con anterioridad en "ConstraintLayout". (0,25)

En las fotos anteriores se puede comprobar que hemos pasado de tener un **LinearLayout** una vez generado los fragmentos a tener **constraintLayout**

3. Crea un par de pestañas deslizantes (ViewPager2) con información sobre la aplicación que quieras resaltar. En la segunda pestaña debe existir el botón “Comenzar”. Estas pestañas deslizantes se incluirán en la lógica de navegación tras la inclusión del usuario (LoginFragment → Pestañas ViewPager2) y al pulsar sobre el botón “Comenzar” navegará al menú principal (Pestaña ViewPager2 → MenuFragment). Si desde el menú principal se pulsa en “Salir”, volvería directamente al fragmento de Login (MenuFragment → LoginFragment). (1)



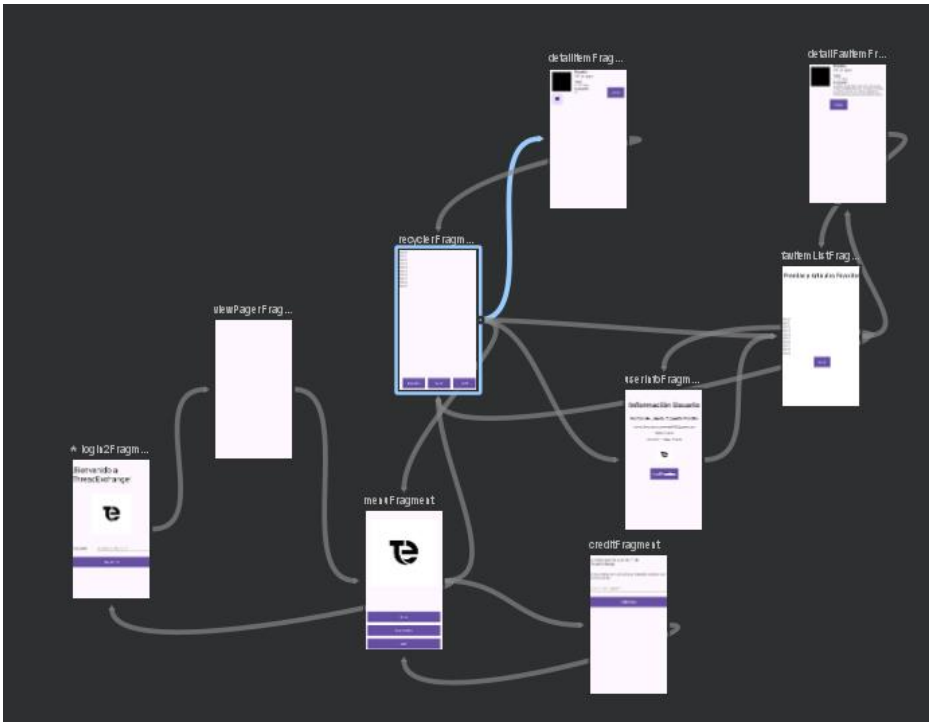
```
package com.example.threadexchange.adapter

import ...


new *
class ViewPagerAdapter(fa:FragmentActivity) : FragmentStateAdapter(fa) {
    new *
    override fun getItemCount(): Int=2
    new *
    override fun createFragment(position: Int): Fragment{
        return when (position){
            0-> FragmentBlanco()
            1-> FragmentNegro()

            else -> throw IllegalArgumentException("Posicion invalida: $position")
        }
    }

    }
```



```
package com.example.threadexchange

import ...


new *
class FragmentBlanco : Fragment() {


    private var _binding : FragmentBlancoBinding? = null
    new *
    private val binding get() = _binding!!
    new *
    @SuppressLint("SuspiciousIndentation")
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentBlancoBinding.inflate(inflater,container, attachToParent: false)

        return binding.root
    }
}
```

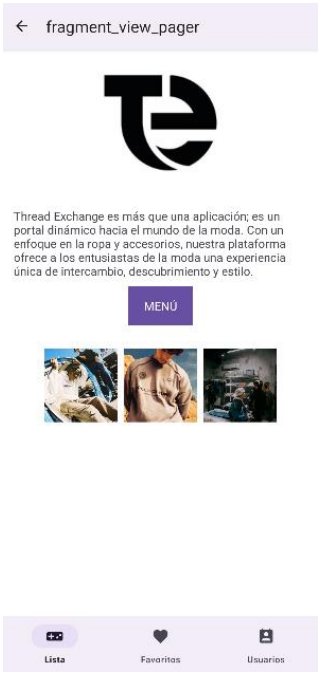
```kotlin
import ...
new *
class FragmentNegro : Fragment() {

    private var _binding: FragmentNegroBinding? = null
    new *
    private val binding get() = _binding!!
    new *
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        _binding = FragmentNegroBinding.inflate(inflater,container, attachToParent: false)
        val root: View = binding.root

        binding.button4.setOnClickListener{ it: View!
            findNavController().navigate(R.id.action_viewPagerFragment_to_menuFragment)
        }
        return root
    }
    new *
    override fun onDestroyView() {
        super.onDestroyView()
        _binding= null
    }
    new *
    private fun closeFragment(){
        requireActivity().supportFragmentManager.popBackStack()
    }
}
```

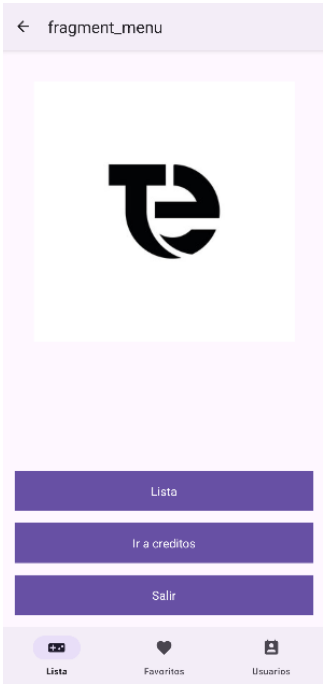
4. Crear la navegación entre todos los fragmentos creados. La información que se muestran en los fragmentos se debe crear desde objetos incorporados y creados dentro de la lógica de la aplicación, en las propias clases o en un objeto (Singleton) que proporcione estos datos. En la próxima unidad, se tomarán los datos desde otras fuentes de datos. (0,5)

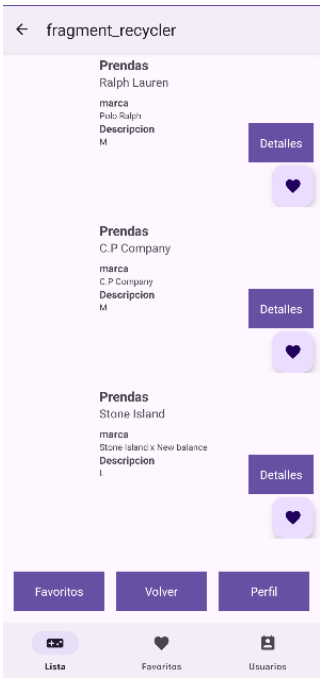


5. Realizar pruebas de uso en el emulador, indicando si encuentra fallos y sus soluciones. (0,5)


4:04
fragment_view_pager
DESLIZA AQUÍ PARA USAR LA APP
¡Bienvenido a ThreadExchange!
Lista
Favoritas
Usuarios


fragment_view_pager
Thread Exchange es más que una aplicación; es un portal dinámico hacia el mundo de la moda. Con un enfoque en la ropa y accesorios, nuestra plataforma ofrece a los entusiastas de la moda una experiencia única de intercambio, descubrimiento y estilo.
MENÚ
Lista
Favoritas
Usuarios


fragment_menu
Lista
Ir a creditos
Salir
Lista
Favoritas
Usuarios


fragment_recycler
Prendas
Ralph Lauren
marca
Polo Ralph
Descripcion
M
Detalles
Prendas
C.P Company
marca
C.P Company
Descripcion
M
Detalles
Prendas
Stone Island
marca
Stone Island x New balance
Descripcion
I
Detalles
Favoritos
Volver
Perfil
Lista
Favoritas
Usuarios