

PROYECTO: INTERFAZ DE USUARIO 2



Nombre: Alejandro Morcillo Muriel

Fecha: 29/05/24

github: <https://github.com/ales5muriel/ThreadExchange-Parte2.git>

Indice

Realizar que la aplicación soporte como mínimo dos idiomas. Todos los textos y opciones se deben poder traducir. (1)

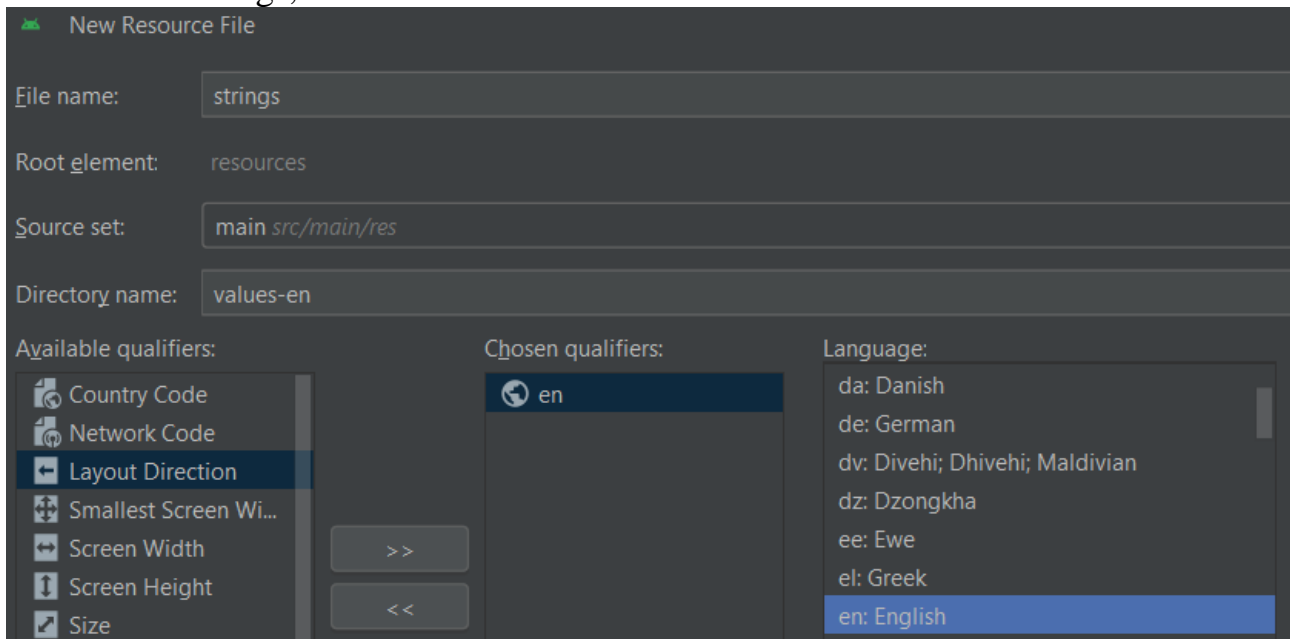
1. Incluir el UpButton en la aplicación. (1)
2. Crear un menú inferior en la aplicación con las opciones mínimas de: (1)
 1. Descubrir: Lleva a ItemListFragment.
 2. Favoritos: Lleva a FavItemListFragment.
 3. Personal: Lleva a UserInfoFragment.
 3. Al pulsar sobre un elemento de una lista se debe navegar a su fragmento de detalle en el que se obtienen los datos específicos del elemento marcado. (1)
4. Incluir mensajes emergentes de información en los casos que consideres necesarios. Se deben incluir un mínimo de dos de estos elementos. Se pueden usar “Toast” o “Snackbar”. (0,5)
5. Realizar pruebas de uso en el emulador, indicando si encuentra fallos y sus soluciones. (0,5)

Finalmente se debe realizar el trabajo de documentación y pruebas de uso de la aplicación creada en un dispositivo móvil. Se tiene que realizar: (CE 2H y CE 2I).

1. Interfaces de usuario. (1)
2. Diagrama de clases de la aplicación. (1)
3. Diagrama de casos de uso. (1)
4. Configuración de un dispositivo móvil para instalar la aplicación creada. (1)
5. Pruebas de uso de la aplicación en el móvil, mostrando su comportamiento en ambos formatos: vertical y apaisado. (1)

Realizar que la aplicación soporte como mínimo dos idiomas. Todos los textos y opciones se deben poder traducir.

Para poder realizar esta parte, nos dirigimos a **values** creamos un directorio que lo llamaremos strings, ahora crearemos dentro un archivo de **values resource file**.

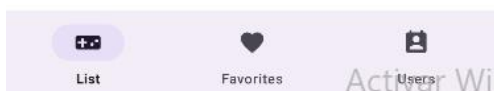


← fragment_view_pager



Thread Exchange is more than an application; it's a dynamic portal to the world of fashion. With a focus on clothing and accessories, our platform offers fashion enthusiasts a unique experience of exchange, discovery, and style.

MENÚ



1. Incluir el UpButton en la aplicación.

En nuestro caso hemos usado un actionBar el cual no servirá para viajar al fragmento anterior, según la lógica que hayamos implementado en nuestro navgraph.xml

El sistema de navegación maneja automáticamente la navegación hacia atrás cuando se utiliza NavController para navegar entre destinos.

En mi caso estoy utilizando un ActionBar, no necesitas implementar un UpButton explícitamente. En su lugar, la navegación hacia atrás se maneja automáticamente según la configuración en tu NavGraph.

2. Crear un menú inferior en la aplicación con las opciones mínimas de:

1. **Descubrir:** Lleva a ItemListFragment.
2. **Favoritos:** Lleva a FavItemListFragment.
3. **Personal:** Lleva a UserInfoFragment.

Para esto hemos tenido que crear un directorio llamado menu, donde hemos creado un **menu.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <!-- Agregar UpButton como un elemento de menú -->

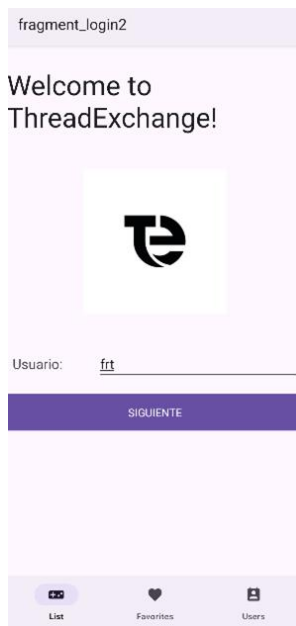
    <item
        android:id="@+id/recyclerFragment"
        android:icon="@drawable/baseline_videogame_asset_24"
        android:title="@string/lista"
        app:showAsAction="ifRoom"/>

    <item
        android:id="@+id/favItemListFragment2"
        android:icon="@drawable/baseline_favorite_24"
        android:title="@string/Favoritos"
        app:showAsAction="ifRoom"/>

    <item
        android:id="@+id/userInfoFragment2"
        android:icon="@drawable/baseline_perm_contact_calendar_24"
        android:title="@string/Usuarios"
        app:showAsAction="ifRoom"/>

</menu>
```





En este caso lo que nos quedaría sería ajustar el modo de viajar una vez seleccionamos el botón, para ello tendremos que modificar el **navgraph**, agregar las acciones pertinentes para que haga lo que deseamos al seleccionar el botón. Tendremos que asegurarnos que para cada fragmento tenga un identificador único y por último vincular los botones con las acciones del **navgraph**.

En mi actividad o fragmento que contiene el menú inferior, se implementa un método para manejar los clics en los elementos del menú.

```
package com.example.threadexchange

import android.content.Intent
import android.os.Bundle
import android.view.View
import android.widget.EditText
import androidx.appcompat.app.AppCompatActivity
import androidx.navigation.findNavController
import androidx.navigation.fragment.NavHostFragment
import androidx.navigation.ui.NavigationUI
import androidx.navigation.ui.setupWithNavController
import com.bumptech.glide.Glide
import com.example.threadexchange.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    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 {
```

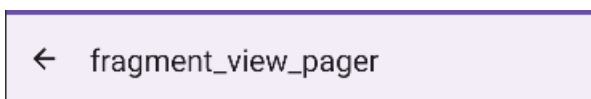
```

        binding.bottomNav.visibility = View.VISIBLE
    }
}
setContentViews( binding.root)
binding.bottomNav.setupWithNavController(navController)
NavigationUI.setupActionBarWithNavController(this, navController)
}

override fun onSupportNavigateUp(): Boolean {
    val navController = this.findNavController(R.id.navHostFragment)
    return navController.navigateUp()
}
}

```

En nuestra aplicación, hemos implementado un menú inferior que proporciona acceso rápido a diferentes secciones de la aplicación. Para lograr esto, hemos utilizado la clase **BottomNavigationView** y la hemos vinculado con el **NavController** utilizando el método **setupWithNavController(navController)**. Esto nos permite asociar cada elemento del menú con un destino de navegación en nuestro gráfico de navegación (**NavGraph**).



De esta forma se vera en nuestra **App** el UpButton

3. Al pulsar sobre un elemento de una lista se debe navegar a su fragmento de detalle en el que se obtienen los datos específicos del elemento marcado. (1)

En mi proyecto en mi **item_list.xml**, he creado un botón de detalles el cual me permite navegar al fragmento de detalle del producto correspondiente.



```

<action
    android:id="@+id/action_recyclerFragment_to_detailItemFragment3"
    app:destination="@id/detailItemFragment" />

```

De esta manera cuando pulsemos el boton de detalle asociado a cada uno de los items de nuestra lista, viajara al fragmento asociado correspondiente.

4. Incluir mensajes emergentes de información en los casos que consideres necesarios. Se deben incluir un mínimo de dos de estos elementos. Se pueden usar “Toast” o “Snackbar”. (0,5)

En mi App he implementado varios Toast que se pueden ver y comprobar mediante la emulación de la App, En primer lugar encontramos el Toast de inicio de sesión, el cual controla que si el usuario no ha ingresado ningún usuario no podrá iniciar y pasar al siguiente fragmento.

```
// Mostrar un Toast si el campo de usuario está vacío.  
Toast.makeText(requireContext(), text: "Tienes que rellenar el usuario para poder pasar al siguiente fragmento", Toast.LENGTH_SHORT).show()
```

En la App, también podremos encontrar un toast el cual cuando marquemos o pulsemos un elemento en la lista de productos, nos mostrara el nombre del producto seleccionado.

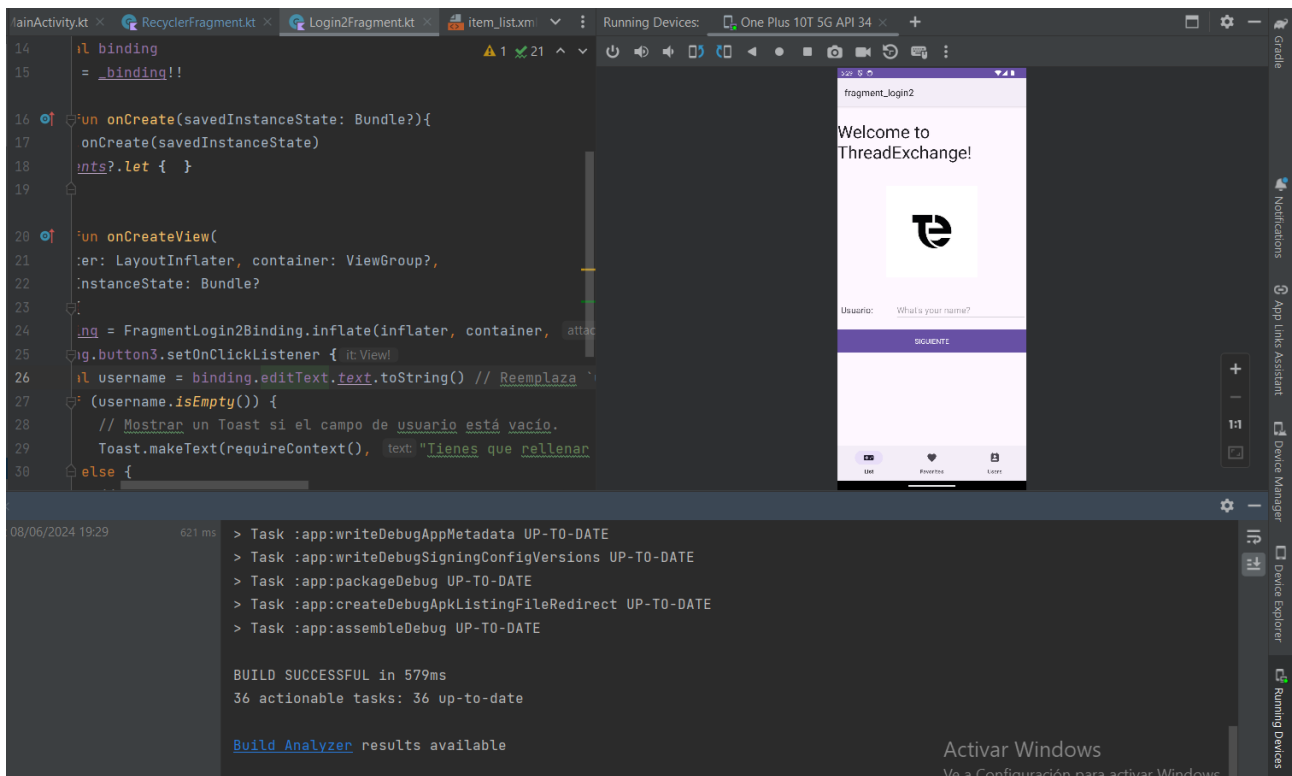
```
Toast.makeText(requireContext(),  
    Ropa.Nombre,  
    Toast.LENGTH_SHORT  
) .show()
```

Por ultimo, hemos añadido un Toast en el fragmento de detalle el cual nos indicara cual sera el mensaje o comentario privado que hemos redactado o enviado sobre ese producto.

```
// Realizar la acción deseada con el comentario, por ejemplo, mostrarlo en un Toast  
Toast.makeText(requireContext(), text: "Comentario: $comment", Toast.LENGTH_SHORT).show()
```










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

En el apartado de interfaces de usuario podemos ver todas las capturas necesarias de la aplicación donde se ve imagen a imagen el funcionamiento de la misma.



1. Interfaces de usuario. (1)

Los distintos dispositivos que he utilizado para probar la App

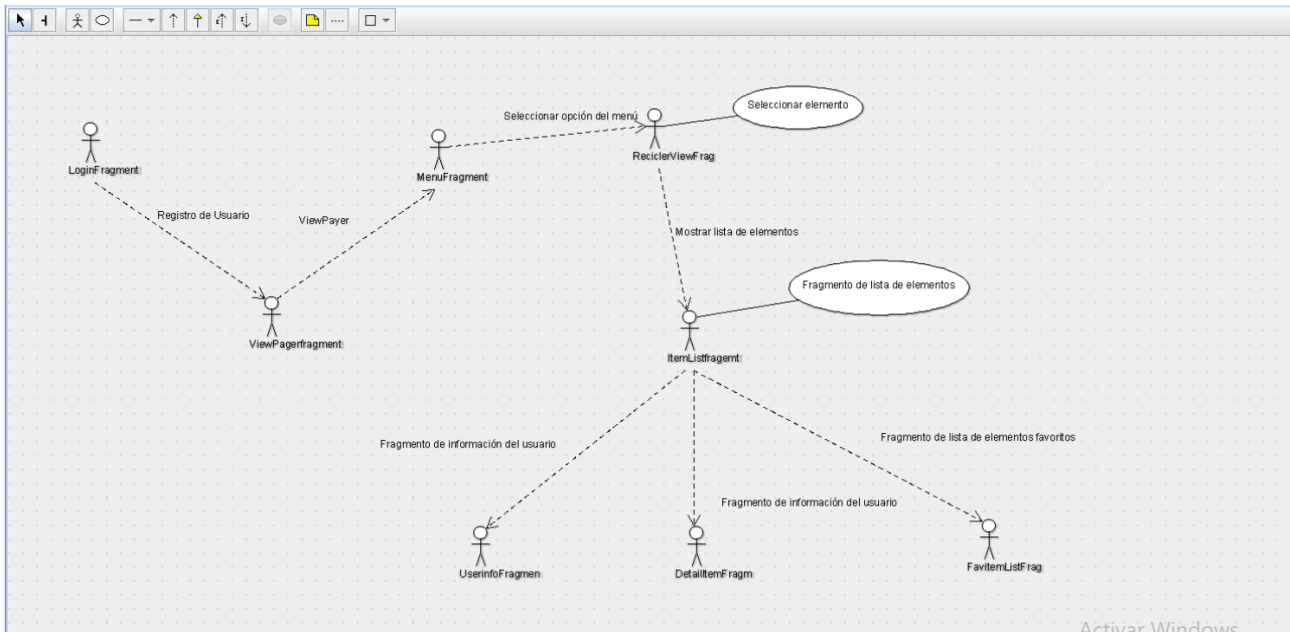
Device Manager				
	Name	API	Type	
	Nexus 10 API 34 Android 14.0 ("UpsideDownCake") x86_64	34	Virtual	 
	One Plus 10T 5G API 34 Android 14.0 ("UpsideDownCake") x86_64	34	Virtual	 
	Pixel 3a API 34 extension_level_7 x86_64 Android 14.0 ("UpsideDownCake") x86_64	34	Virtual	 

2. Diagrama de clases de la aplicación. (1)

El diagrama de clases se encuentra dentro del proyecto, el nombre del archivo es diagrama.svg y el diagrama.uml

3. Diagrama de casos de uso. (1)

El diagrama de caso de uso lo hemos creado mediante dia:



4. Configuración de un dispositivo móvil para instalar la aplicación creada. (1)

EL dispositivo móvil, ha sido configurado para que se puede ejecutar tanto en vertical como en horizontal, en los videos se puede ver el funcionamiento de la aplicación en el dispositivo.

5. Pruebas de uso de la aplicación en el móvil, mostrando su comportamiento en ambos formatos: vertical y apaisado. (1)

Las prueba de uso de la aplicación podrás encontrarlas en unos videos que he realizado, se encuentran en el Proyecto de github.

Enlace: <https://github.com/ales5muriel/ThreadExchange-Parte2.git>