PROYECTO : INTERFAZ DE USUARIO 2

Juan Carlos Ortega Lepe
2ºDAM

ÍNDICE

Enlace repositorio GitHub:

https://github.com/Escaroz/JuegosFase2ProyectoFinal

Introducción:

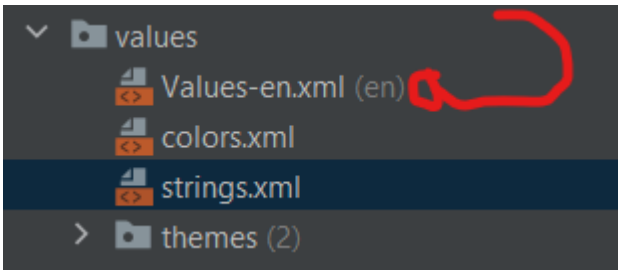
Esta práctica, es la segunda Fase de nuestra aplicación de Android Studio, a continuación se añade la explicación para cada apartado de esta práctica.

1.Idiomas

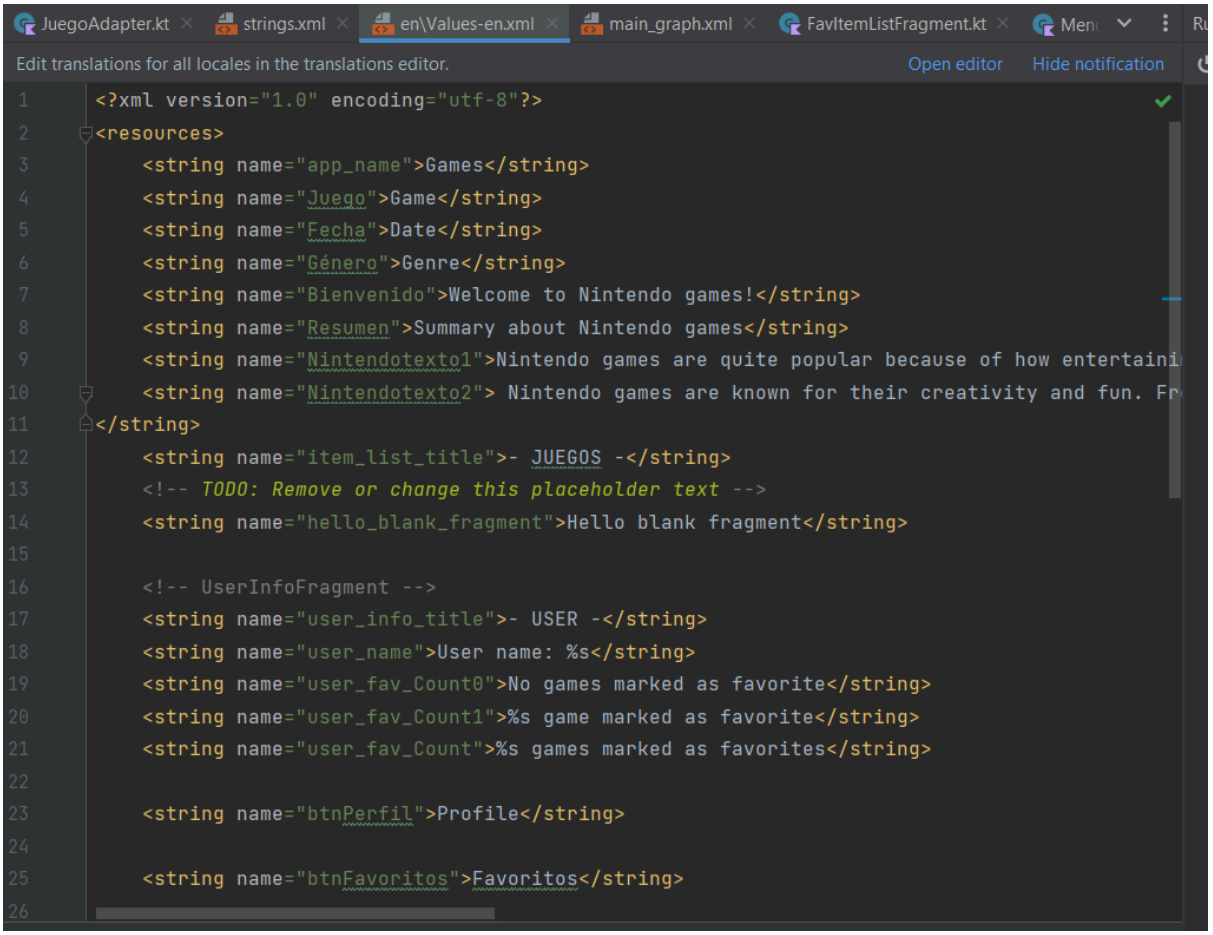
Para implementar otro idioma en nuestra aplicación, deberemos crear un nuevo archivo string.xml, pero deberemos seleccionar que el idioma esté en inglés.

Una vez creado, pasaremos todos los strings de nuestro archivo en español, al de inglés, para que una vez ejecutemos la aplicación, se nos muestre el idioma en inglés de la aplicación.

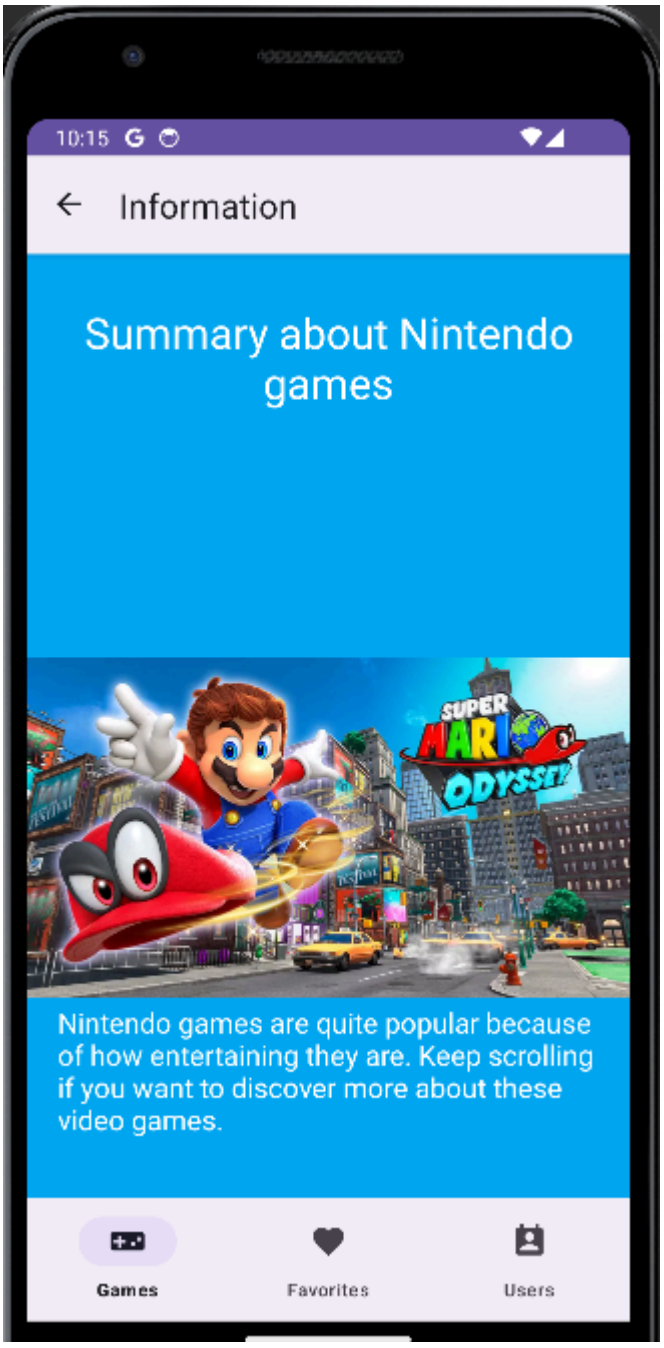
Creamos el archivo string en inglés en la carpeta values:



Convertimos todos los strings del archivo en español al de inglés



Una vez hecho esto, si ejecutamos nuestra aplicación y hemos cambiado los strings de los archivos xml correctamente, se deberá ver de la siguiente manera:


10:15
Information
Summary about Nintendo games
SUPER MARIO ODYSSEY
Nintendo games are quite popular because of how entertaining they are. Keep scrolling if you want to discover more about these video games.
Games
Favorites
Users

2.UpButtom

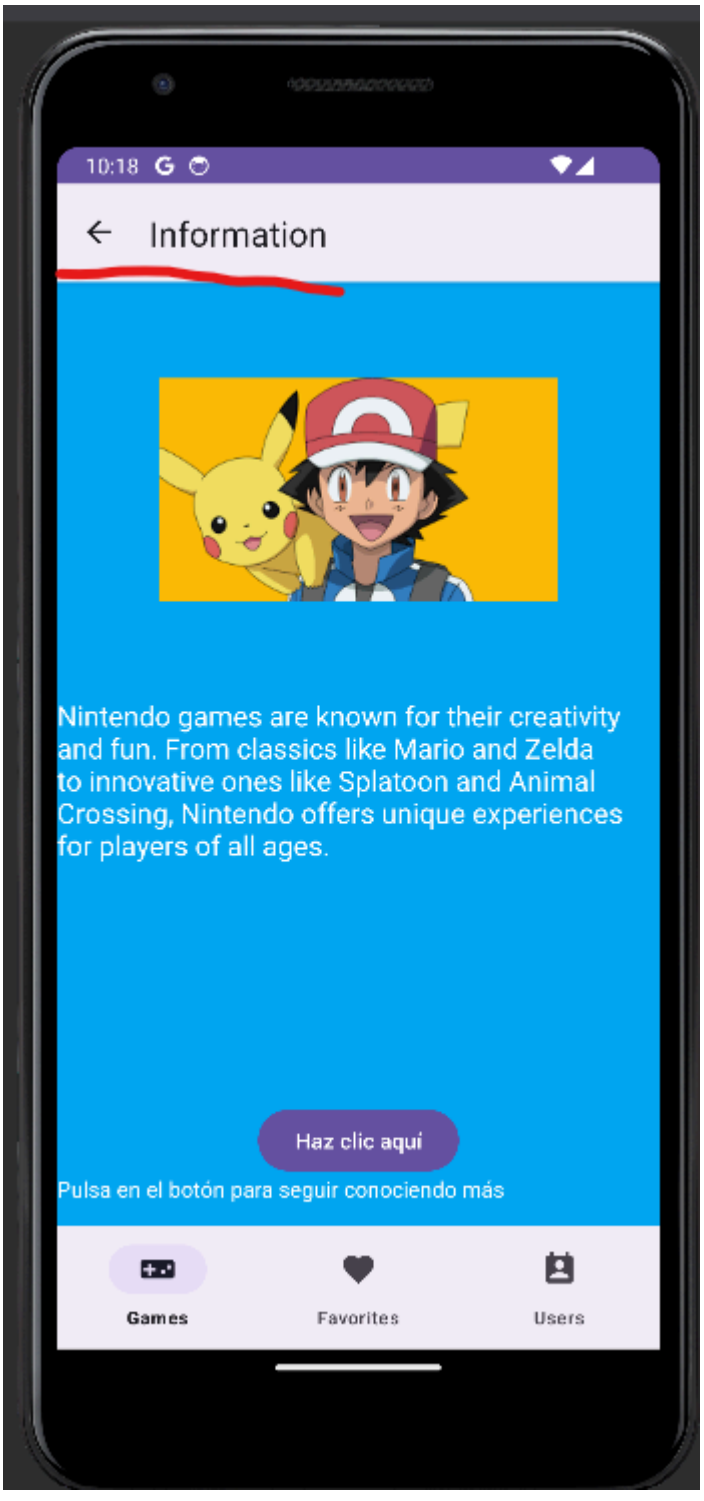
Para realizar el UpButtom, debemos agregar las siguientes líneas de código en nuestra mainActivity:

```
        binding.bottomNav.setupWithNavController(navController)

        NavigationUI.setupActionBarWithNavController( activity: this, navController)
    }

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

Una vez hecho esto, debemos quitar las líneas en themes.xml que prohíben el uso de UpButtom:

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Base.Theme.Juegos" parent="Theme.Material3.DayNight">
        <!-- Customize your light theme here. -->
        <!-- <item name="colorPrimary">@color/my_light_primary</item> -->
    </style>

    <style name="Theme.Juegos" parent="Base.Theme.Juegos" />
</resources>
```

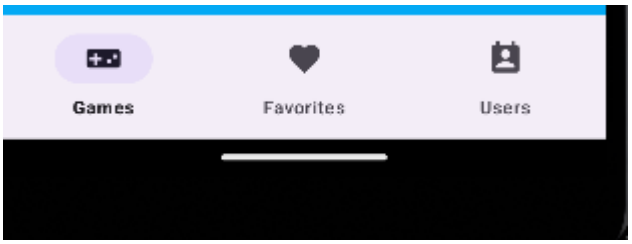
Una vez hecho esto, ejecutamos la aplicación, y funcionará correctamente:


10:18
Information
Nintendo games are known for their creativity and fun. From classics like Mario and Zelda to innovative ones like Splatoon and Animal Crossing, Nintendo offers unique experiences for players of all ages.
Haz clic aquí
Pulsa en el botón para seguir conociendo más
Games
Favorites
Users

3.Menu

para hacer el menú, deberemos crearlo en la carpeta menú, y poner este código en el xml, para que nos lleve a los fragmentos que queremos:

```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">

    <item
        android:id="@+id/itemListFragment"
        android:icon="@drawable/baseline_videogame_asset_24"
        android:title="Juegos"
        app:showAsAction="ifRoom"/>

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

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

</menu>
```



4.DetailItemListFragment

Para mostrar este fragmento, deberemos crearlo correctamente, y usar binding:

```
1   package com.marisma.juegos.adapter                                    ⚠ 8  ✓ 45
2
3   import ...
11
    Carlos Ortega
12  class DetailItemFragment : Fragment() {
13
14      private var _binding: FragmentDetailItemBinding? = null
        Carlos Ortega
15      private val binding get() = _binding!!
16
        Carlos Ortega
17      override fun onCreateView(
18          inflater: LayoutInflater, container: ViewGroup?,
19          savedInstanceState: Bundle?
20      ): View {
21          _binding = FragmentDetailItemBinding.inflate(inflater, container,  attachToParent: false)
22          return binding.root
23      }
24
        Carlos Ortega
25      override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
26          super.onViewCreated(view, savedInstanceState)
27
28          val juegoNombre = arguments?.getString( key: "juegoNombre")
29          val juegoGenero = arguments?.getString( key: "juegoGenero")
30          val juegoEstudio = arguments?.getString( key: "juegoEstudio")
31          val juegoNota = arguments?.getInt( key: "juegoNota",  defaultValue: 0) // Valor predetermi
```

Definimos los atributos de nuestra ListFragment:

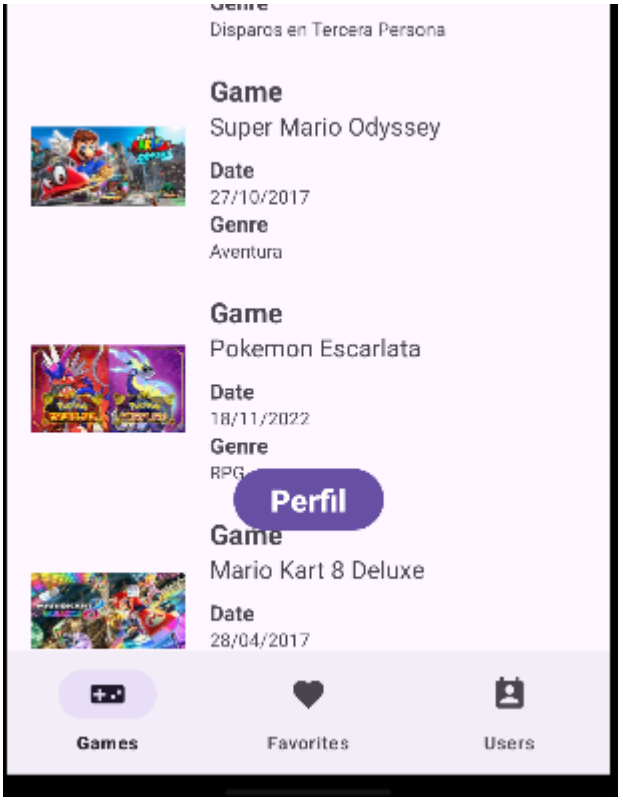
```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    val juegoNombre = arguments?.getString( key: "juegoNombre")
    val juegoGenero = arguments?.getString( key: "juegoGenero")
    val juegoEstudio = arguments?.getString( key: "juegoEstudio")
    val juegoNota = arguments?.getInt( key: "juegoNota",  defaultValue: 0) // Valor predetermi
    val juegoFoto = arguments?.getString( key: "juegoFoto")

    // Actualizar las vistas con los detalles del juego
    juegoNombre?.let { it: String
        binding.textViewJuego.text = "Nombre: $it"
    }
    juegoGenero?.let { it: String
        binding.textViewGenero.text = "Género: $it"
    }
    juegoEstudio?.let { it: String
        binding.textViewEstudio.text = "Fecha: $it"
    }
    juegoNota?.let { it: Int
        binding.textViewNota.text = "Foto: $it"
    }

    juegoFoto?.let { it: String
        // Cargar la foto utilizando Glide o Picasso
        Glide.with( fragment: this) RequestManager
            .load(juegoFoto) RequestBuilder<Drawable!>
            .placeholder(R.drawable.zelda) // Opcional: establece una imagen de marcador
```

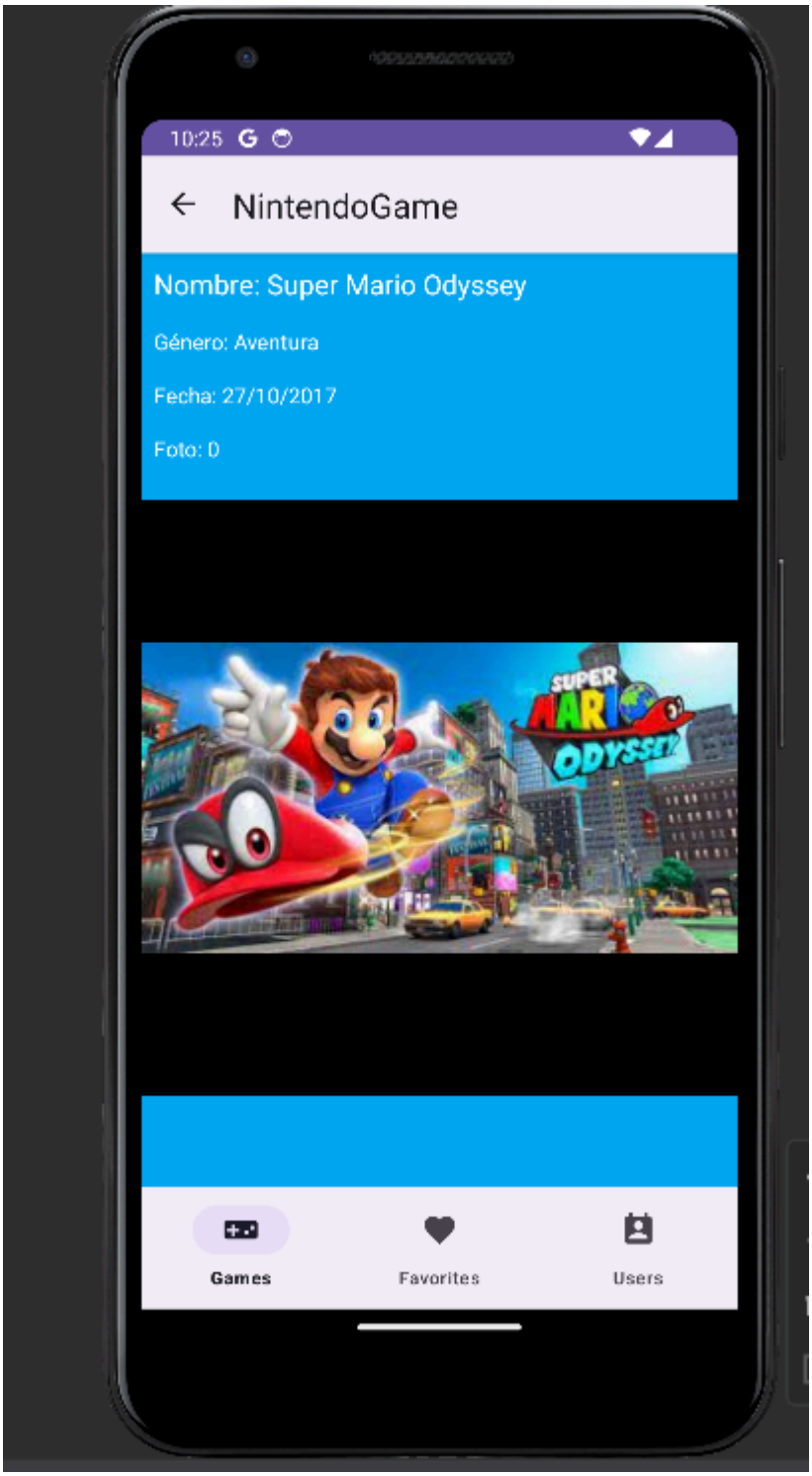
```
        juegoFoto?.let { it: String
            // Cargar la foto utilizando Glide o Picasso
            Glide.with( fragment: this) RequestManager
                .load(juegoFoto) RequestBuilder<Drawable!>
                .placeholder(R.drawable.zelda) // Opcional: establece una imagen de marcador
                .error(R.drawable.zelda) // Opcional: establece una imagen de error si la ca
                .into(binding.ivJuego)
        }
    }

    Carlos Ortega
    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Una vez hecho, deberemos hacer en nuestro ItemListFragment, un findNavController para abrir este fragmento desde nuestra lista:

```
47
    Carlos Ortega
48  private fun onItemSelected(datos: Datos) {
49      val bundle = Bundle().apply { this: Bundle
50          putString("juegoNombre", datos.juego)
51          putString("juegoGenero", datos.genero)
52          putString("juegoEstudio", datos.fecha)
53          putString("juegoFoto", datos.foto)
54      }
55      findNavController().navigate(R.id.action_itemListFragment_to_detailItemFragment, bundle)
56  }
57
    Carlos Ortega
58  override fun onDestroyView() {
59      super.onDestroyView()
60      _binding = null
61  }
62
63
```

Y así se ve el resultado:




10:25
NintendoGame
Nombre: Super Mario Odyssey
Género: Aventura
Fecha: 27/10/2017
Foto: 0
Games
Favorites
Users

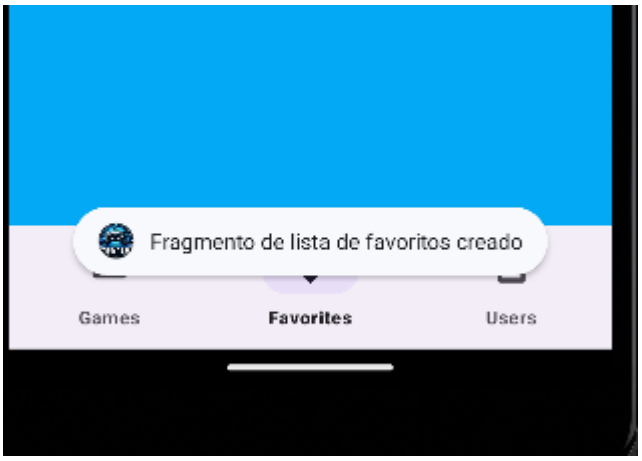
5.Toast

Para hacer un toast, deberemos seleccionar el fragmento donde lo queramos realizar, y agregaremos las siguientes líneas de código:

En mi caso, lo e hecho en mi FavItemListFragment:

```
    Carlos Ortega
    private fun showToast(message: String) {
        Toast.makeText(requireContext(), message, Toast.LENGTH_SHORT).show()
    }

    Carlos Ortega
    companion object {
        Carlos Ortega
        @JvmStatic
        fun newInstance(param1: String, param2: String) =
            FavItemListFragment().apply { this: FavItemListFragment
                arguments = Bundle().apply { this: Bundle
                    putString(ARG_PARAM1, param1)
                    putString(ARG_PARAM2, param2)
                }
            }
    }
}
```

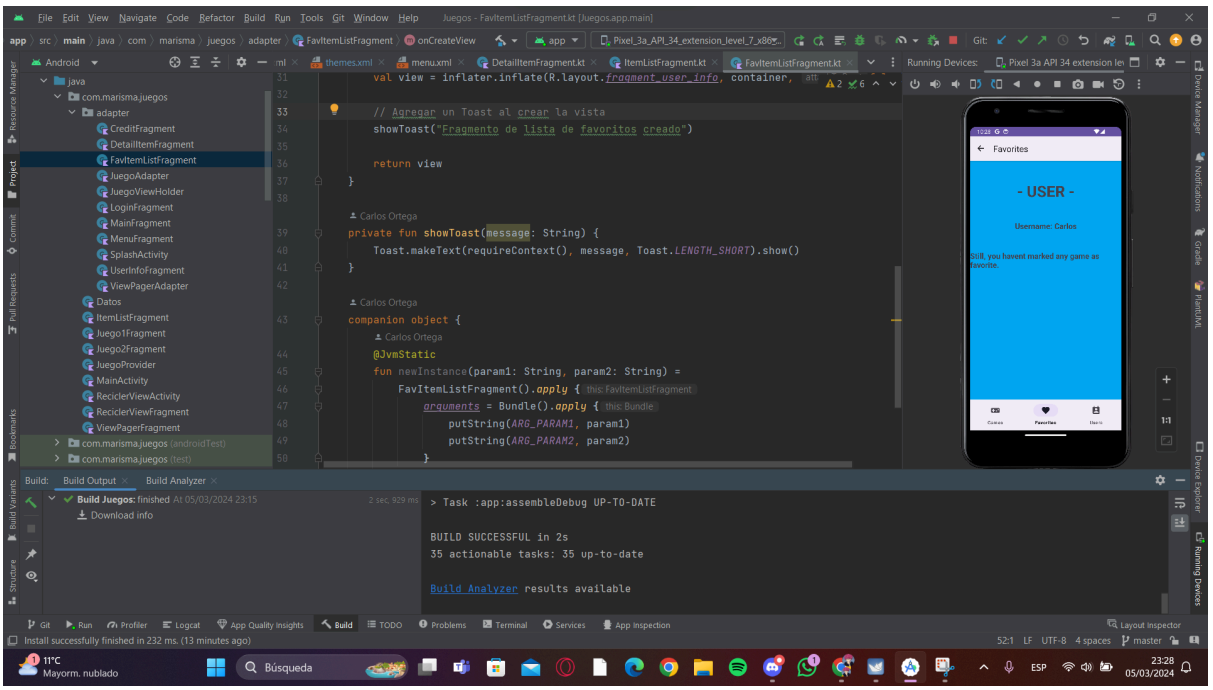
Una vez añadido esto, cuando accedemos a este fragmento de nuevo saldrá un mensaje abajo en el centro.



6.Pruebas

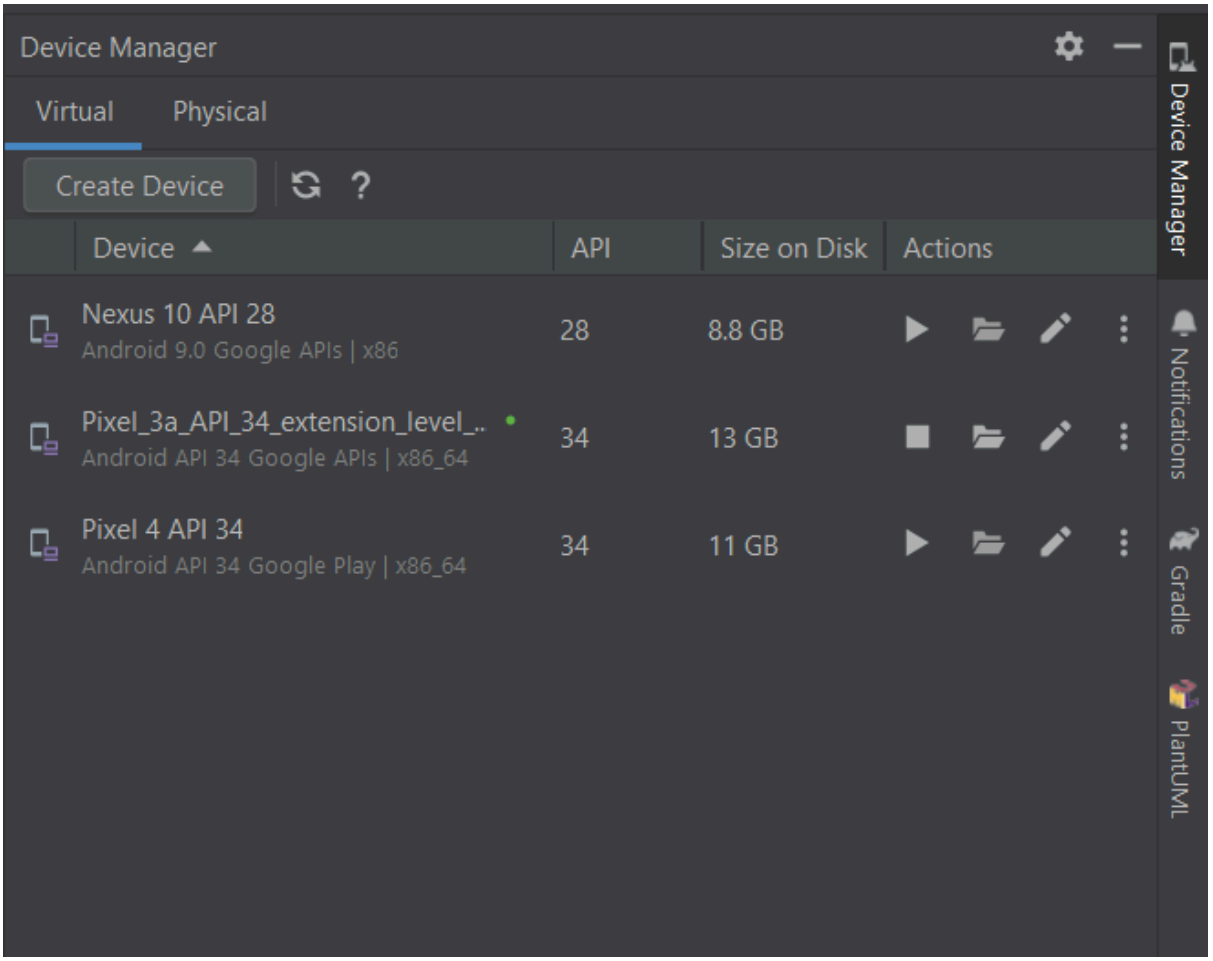
Sobre los pruebas de uso del emulador, si abrimos la consola, no nos saldrá ningún error porque he realizado correctamente la aplicación.

Si hubiera algún problema, nos lo marcaría de color rojo, y nos indicaría donde estaría el error.



7.Interfaces de usuario

Los móviles que he utilizado para la prueba de la aplicación, son los siguientes:



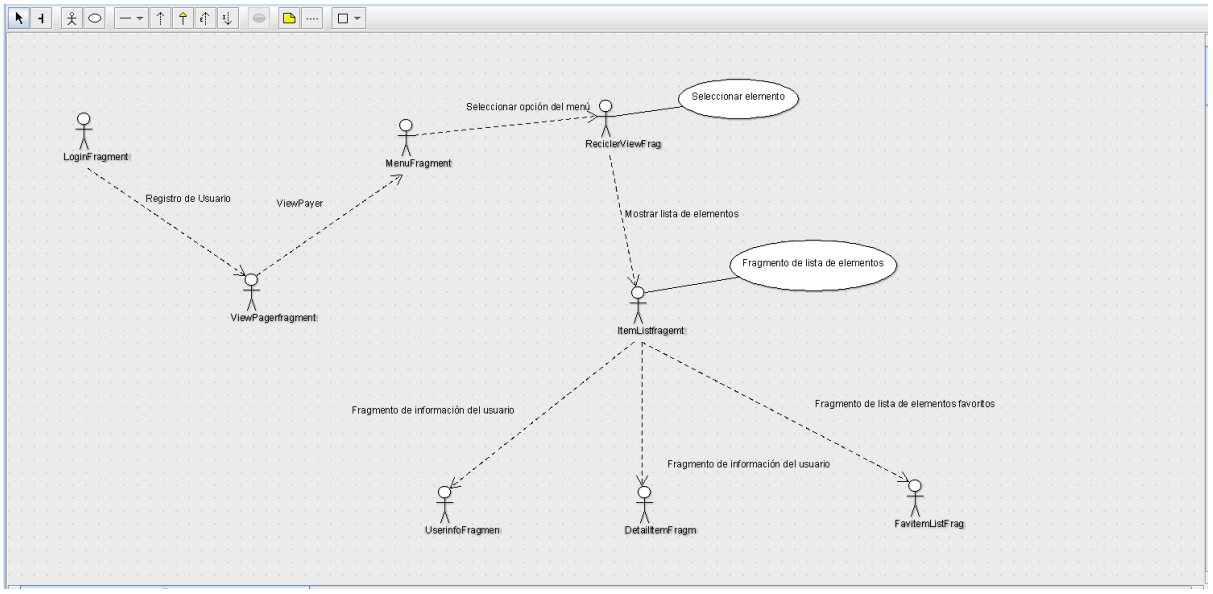
recomiendo el Pixel_3a, ya que es el que viene por defecto, y funciona bastante bien.

8.Diagrama de clases.

El diagrama de clases de mi aplicación, lo en este zip, y el archivo se llama DiagramaClase.svg

9.Diagrama de casos de uso.

A continuación se muestra el diagrama de casos de uso de mi aplicación:



10. Configuración del dispositivo móvil.

El dispositivo móvil, está configurado para que se ejecute en vertical, pero en horizontal también funciona la aplicación pero se ve algo cortado. En los videos de GitHub muestro la aplicación en vertical y en horizontal.

11. Pruebas de uso de la aplicación

Las pruebas de uso de la aplicación las he realizado en unos videos, que están junto al proyecto de GitHub, a continuación añado de nuevo el enlace a mi repositorio de GitHub.

https://github.com/Escaroz/JuegosFase2ProyectoFinal