PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

PROYECTO: INTERFAZ DE USUARIO 1

android

REALIZADO POR:

Miguel Figuerola Alonso

ÍNDICE

1. 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).

2. 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”.

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

4. Incorporar una o varias fuentes de datos para la aplicación.

5. Creación de estilos personalizados, al menos para los siguientes elementos: Textos, Títulos, Botones.

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

7. DetailItemFragment: Fragmento con información más detallada de un elemento de los que trata la información.

8. UserInfoFragment: Fragmento con información del usuario que se encuentra en la aplicación.

9. Transforma los “LinearLayout” de los fragmentos que estaban ya creados con anterioridad en “ConstraintLayout”.

10. 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)

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

1. 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).

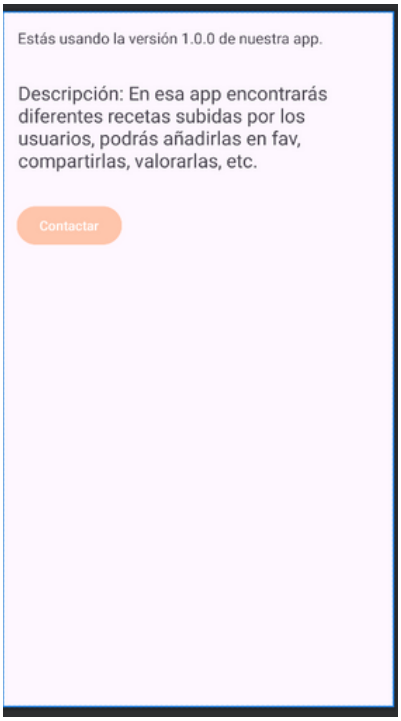
```
package com.makulky.socialcooking

import ...

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

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

Podemos observar que en el MainActivity he usado la técnica de view binding. Esto hará que una vez cargado el splash y salte al main, este cargue el fragment loginFragment, como podemos ver a continuación.

```
package com.makulky.socialcooking

import ...

class loginFragment : Fragment() {

    private var _binding: FragmentLoginBinding? = null
    private val binding get() = _binding!!
    private val svm: sharedViewModel by activityViewModels()

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = FragmentLoginBinding.inflate(inflater, container, attachToParent: false)

        binding.aceptarButton.setOnClickListener{ it: View!

            var usuario = binding.nickText.text.toString()

            if(usuario.isNotEmpty()){
                svm.usuario = usuario
                findNavController().navigate(R.id.action_loginFragment_to_menuFragment)
                Toast.makeText(requireContext(), text: "¡Bienvenido " + usuario + "!", Toast.LENGTH_SHORT).show()
            }
            else{
                Toast.makeText(requireContext(), text: "Introduce el nombre del usuario.", Toast.LENGTH_SHORT).show()
            }
        }
        return binding.root
    }

}
```

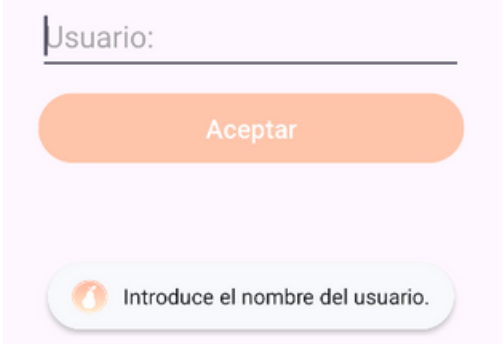
creditAcitivity al pasarlo a un fragment quedaría tal que así:

```
package com.makulky.socialcooking

import ...

class creditFragment : Fragment() {

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

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = FragmentCreditBinding.inflate(inflater,container, attachToParent: false)

        binding.contactarBoton.setOnClickListener{ it: View!
            val direccionCorreo = "mfigalo996@g.educaand.es"
            val asunto = "Consulta de la app ${"SocialCooking"}"

            val enviarEmail = Intent(Intent.ACTION_SENDTO).apply { this: Intent
                data = Uri.parse( uriString: "mailto:")
                putExtra(Intent.EXTRA_EMAIL, arrayOf(direccionCorreo))
                putExtra(Intent.EXTRA_SUBJECT, asunto)
            }
            startActivity(enviarEmail)
        }

        return binding.root
    }
}
```



2. 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”.

En el punto anterior hemos visto el contenido de loginFragment. Tengo un botón que al pulsarlo, si no hay ningún nombre de usuario introducido, saldrá un Toast en el cual le indiracará al usuario que tiene que introducir un nick. Una vez que lo hayas puesto, si le das al botón aceptar, irá al fragmento menuFragment. Como veremos a continuación.



Una vez en el menuFragment, veremos que tendremos tres botones. Inicio, Créditos y Salir.

```
package com.makulky.socialcooking

import ...

class menuFragment : Fragment() {

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

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = FragmentMenuBinding.inflate(inflater,container, attachToParent: false)

        binding.salirButton.setOnClickListener{ it: View!
            findNavController().navigate(R.id.action_menuFragment_to_loginFragment)
        }

        binding.creditosButton.setOnClickListener{ it: View!
            findNavController().navigate(R.id.action_menuFragment_to_creditFragment)
        }

        binding.imagePerfil.setOnClickListener{ it: View!
            findNavController().navigate(R.id.action_menuFragment_to_perfilFragment)
        }

        binding.iniciarButton.setOnClickListener{ it: View!
            findNavController().navigate(R.id.action_menuFragment_to_inicioFragment)
        }

        return binding.root
    }
}
```

Como vemos en la imagen anterior. cada botón, gracias a la técnica de view binding y al navigation controller, podremos acceder a cada fragmento al pulsar su correspondiente botón. Si pulsamos el botón Salir, veremos que volveremos al loginFragment. Si pulsamos el botón Créditos, iremos a creditFragment, si pulsamos en el botón de la imagen de perfil, nos saldrá un fragmento (perfilFragment) en el cual nos dará una pequeña información sobre el usuario. Y por último, si pulsamos en el botón Iniciar, iremos al fragmento inicioFragment, en el cual nos saldrá una lista de todas nuestras recetas, como veremos a continuación en otro punto.

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

Las pruebas las he grabado en vídeo para que veas el funcionamiento de la aplicación. Lo tendrás en el directorio raíz del proyecto.

4. Incorporar una o varias fuentes de datos para la aplicación.

```
dependencies {  this: DependencyHandlerScope

    val nav_version = "2.7.0"

    // Kotlin
    implementation("androidx.navigation:navigation-fragment-ktx:$nav_version")
    implementation("androidx.navigation:navigation-ui-ktx:$nav_version")

    // Testing Navigation
    androidTestImplementation("androidx.navigation:navigation-testing:$nav_version")


    implementation("androidx.datastore:datastore-preferences:1.1.1")
    implementation("com.github.bumptech.glide:glide:4.12.0")
    annotationProcessor("com.github.bumptech.glide:glide:4.12.0")
    implementation("androidx.fragment:fragment-ktx:1.7.1")
    implementation("androidx.core:core-ktx:1.9.0")
    implementation("androidx.appcompat:appcompat:1.6.1")
    implementation("com.google.android.material:material:1.11.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")
}
```

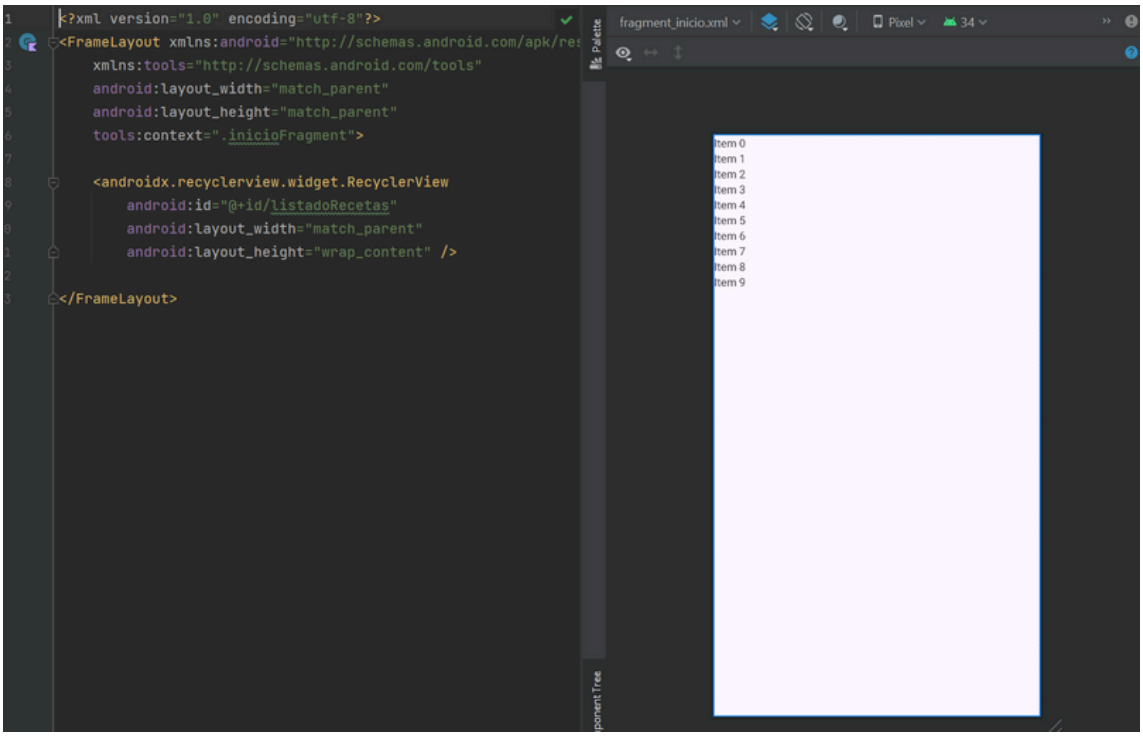
5. Creación de estilos personalizados, al menos para los siguientes elementos: Textos, Títulos, Botones.

Creamos un archivo de recursos llamado styles dentro del directorio values. Allí crearemos los estilos.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <style name="AppTextStyle" parent ="android:Widget.TextView">
        <item name="android:textColor">#FF0000</item>
        <item name="android:textSize">17sp</item>
    </style>

    <style name="AppTitleStyle" parent ="android:Widget.TextView">
        <item name="android:textColor">#0000FF</item>
        <item name="android:textSize">25sp</item>
        <item name="android:textStyle">bold</item>
    </style>

    <style name="AppButtonStyle" parent ="android:Widget.Button">
        <item name="android:backgroundTint">#FFFEC5AB</item>
    </style>

</resources>
```

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

Dentro del fragmento pondremos nuestro RecyclerView, para luego más adelante cargar cada elemento de la lista.



Crearemos una clase en la cual haremos un constructor con los datos del elemento que queramos añadir. En mi caso lo voy a hacer de recetas, por tanto tendrá 5 atributos, como veremos a continuación.

```
data class datosRecetas(
    val id: Int,
    val nombre: String,
    val autor: String,
    val dificultad: String,
    val urlFoto: String
)
```

Esto lo hacemos para que luego, en otra clase que crearemos, pondremos los datos de cada elemento en una lista dentro de un companion object, para poder llamarlo desde otras clases. A esta clase la he llamado recetasProvider.

```kotlin
class recetasProvider {

    companion object{
        val listaRecetas= listOf<datosRecetas>(
            datosRecetas(
                id: 0,
                nombre: "Tarta de queso",
                autor: "Manuel",
                dificultad: "Dificultad: 2/5",
                urlFoto: "https://recetasdecocina.elmundo.es/wp-content/uploads/2017/02/tarta-de-queso-fr%C3%ADa-casera.jpg"
            ),
            datosRecetas(
                id: 1,
                nombre: "Carrilleras al vino tinto",
                autor: "Francisca",
                dificultad: "Dificultad: 3/5",
                urlFoto: "https://i.blogs.es/115a45/carrilleras_vino/1366_2000.jpg"
            ),
            datosRecetas(
                id: 2,
                nombre: "Pato a la naranja",
                autor: "Daniel",
                dificultad: "Dificultad: 4/5",
                urlFoto: "https://t1.uc.ltmcdn.com/es/posts/2/2/1/como_hacer_pato_a_la_naranja_30122_orig.jpg"
            ),
            datosRecetas(
                id: 3,
                nombre: "Cerdo agridulce",
                autor: "María",
                dificultad: "Dificultad: 3/5",
                urlFoto: "https://assets.unileversolutions.com/recipes-v2/247009.jpg"
            ),
```

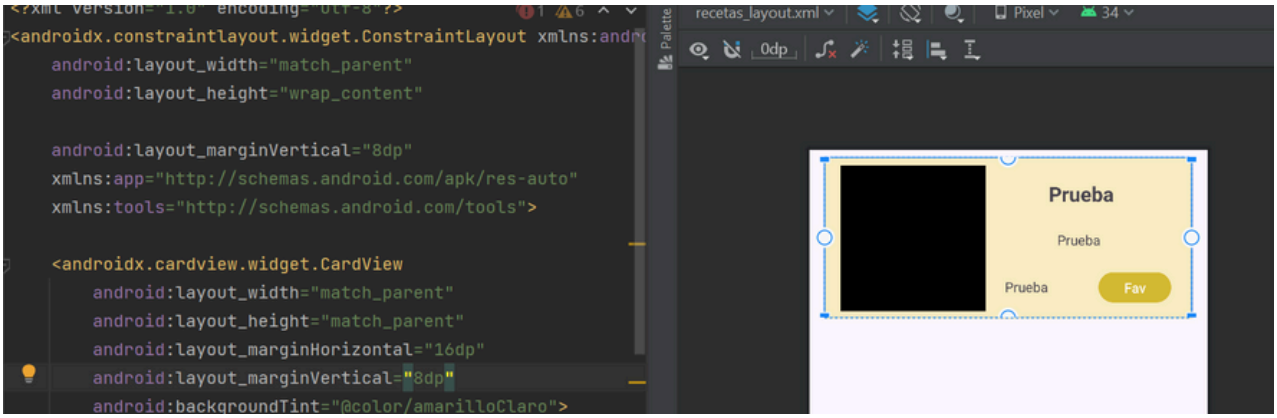
Esto lo hacemos para que luego, en otra clase que crearemos, pondremos los datos de cada elemento en una lista dentro de un companion object, para poder llamarlo desde otras clases. A esta clase la he llamado recetasProvider.

A continuación crearemos un par de clases, una clase que funcionará de adaptador, el cual actúa como un puente entre los datos que le hemos proporcionado con recetasProvider y el recyclerView. Para su correcto funcionamiento necesitariamos crear la otra clase, que funcionará como viewHolder. La principal función de esta clase será contener las vistas para cada elemento de forma individual en el recyclerView.

```kotlin
package com.makulky.socialcooking.adapter

import ...

class recetasAdapter(private val listaRecetas:List<datosRecetas>, private val navController: NavController) : RecyclerView.Ada

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): recetasViewHolder {
        val layoutInflater = LayoutInflater.from(parent.context)
        return recetasViewHolder(layoutInflater.inflate(R.layout.recetas_layout, parent, attachToRoot: false), navController)
    }

    override fun getItemCount(): Int {
        return listaRecetas.size
    }

    override fun onBindViewHolder(holder: recetasViewHolder, position: Int) {
        val item = listaRecetas[position]
        holder.render(item)
    }
}
```

```kotlin
package com.makulky.socialcooking.adapter

import ...

class recetasViewHolder(view: View, private val navController: NavController) : RecyclerView.ViewHolder(view) {

    val binding = RecetasLayoutBinding.bind(view)

    fun render(datosRecetasModel: datosRecetas) {
        binding.nombreRecetaTextView.text = datosRecetasModel.nombre
        binding.autorRecetaTextView.text = datosRecetasModel.autor
        binding.calificacionRecetaTextView.text = datosRecetasModel.dificultad
        Glide.with(binding.imagenReceta.context).load(datosRecetasModel.urlFoto).into(binding.imagenReceta)

        itemView.setOnClickListener { it: View!
            val bundle = Bundle().apply { this: Bundle
                putString("nombre", datosRecetasModel.nombre)
                putString("autor", datosRecetasModel.autor)
                putString("dificultad", datosRecetasModel.dificultad)
                putString("urlFoto", datosRecetasModel.urlFoto)
            }
            navController.navigate(R.id.action_inicioFragment_to_contenidoCeldaFragment, bundle)
        }

        binding.favButton.setOnClickListener { it: View!
            Toast.makeText(itemView.context, text: "Se ha añadido a favoritos!", Toast.LENGTH_SHORT).show()
        }
    }
}
```

Dentro del layout recetas_layout.xml tendremos el diseño de cada elemento del recycler.



El contenido de inicioFragmento, donde contenemos nuestro recyclerView, quedaría tal que así.

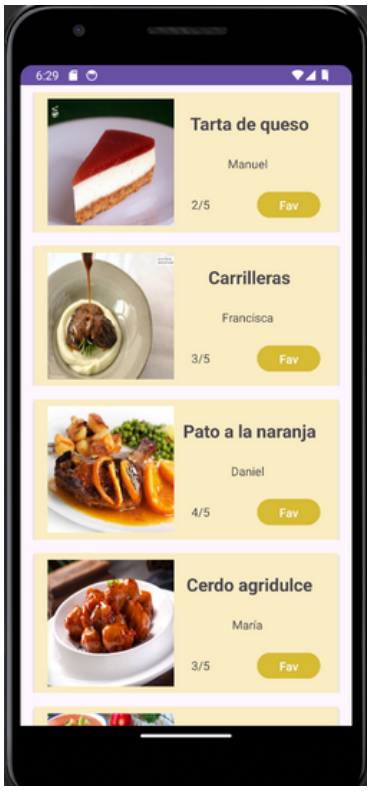
```kotlin
package com.makulky.socialcooking

import ...

class inicioFragment : Fragment() {

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

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = FragmentInicioBinding.inflate(inflater, container, attachToParent: false)

        initRecyclerView()

        return binding.root
    }

    private fun initRecyclerView() {
        val recyclerView = binding.listadoRecetas
        recyclerView.layoutManager = LinearLayoutManager(requireContext())
        recyclerView.adapter = recetasAdapter(recetasProvider.listaRecetas, findNavController())
    }

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

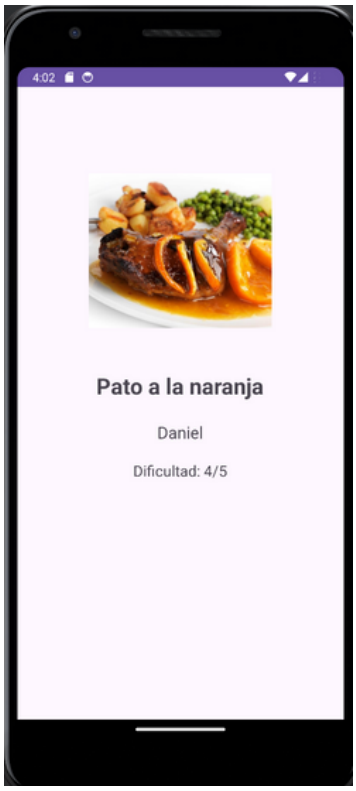
7. DetailItemFragment: Fragmento con información más detallada de un elemento de los que trata la información.

Como hemos visto en nuestro holder, con la función itemView.setOnClickListener, cada vez que pulsemos sobre una celda de nuestro recycler, nos llevará al fragmento que almacena el contenido del elemento seleccionado habiéndolo guardado previamente con un Bundle.

```kotlin
itemView.setOnClickListener { it: View!
    val bundle = Bundle().apply { this: Bundle
        putString("nombre", datosRecetasModel.nombre)
        putString("autor", datosRecetasModel.autor)
        putString("dificultad", datosRecetasModel.dificultad)
        putString("urlFoto", datosRecetasModel.urlFoto)
    }
    navController.navigate(R.id.action_inicioFragment_to_contenidoCeldaFragment, bundle)
}
```

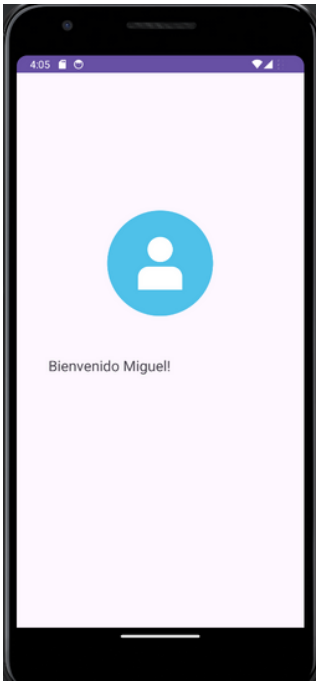
```kotlin
import ...

class contenidoCeldaFragment : Fragment() {

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

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = FragmentContenidoCeldaBinding.inflate(inflater, container, attachToParent: false)

        val nombre = arguments?.getString( key: "nombre")
        val autor = arguments?.getString( key: "autor")
        val dificultad = arguments?.getString( key: "dificultad")
        val urlFoto = arguments?.getString( key: "urlFoto")

        binding.tituloRecetaCelda.text = nombre
        binding.autorRecetaCelda.text = autor
        binding.dificultadRecetaCelda.text = dificultad
        Glide.with(binding.imagenRecetaCelda.context).load(urlFoto).into(binding.imagenRecetaCelda)

        return binding.root
    }

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





8. UserInfoFragment: Fragmento con información del usuario que se encuentra en la aplicación.

```
package com.makulky.socialcooking

import ...

class perfilFragment : Fragment() {

    private var _binding: FragmentPerfilBinding? = null
    private val binding get() = _binding!!
    private val svm: sharedViewModel by activityViewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = FragmentPerfilBinding.inflate(inflater,container, attachToParent: false)

        binding.nickTextView.text = "Bienvenido " + svm.usuario + "!"

        return binding.root
    }
}
```



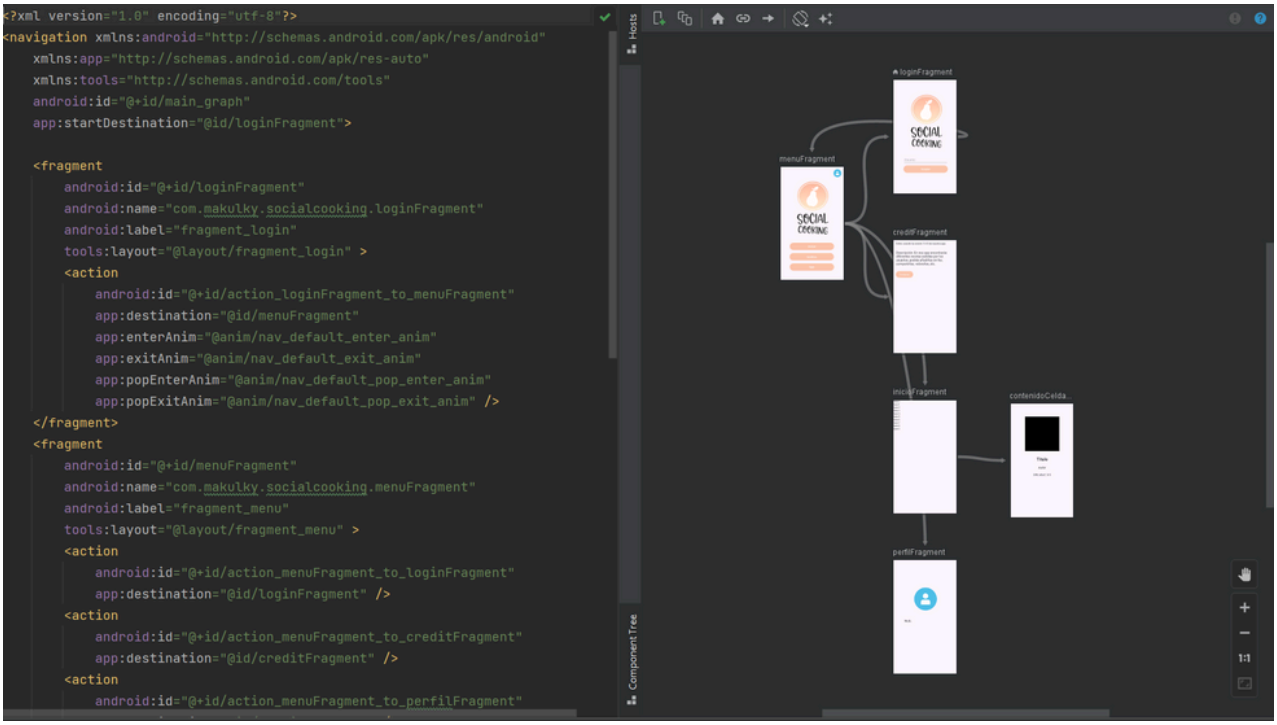
9. Transforma los “LinearLayout” de los fragmentos que estaban ya creados con anterioridad en “ConstraintLayout”.

```
<androidx.constraintlayout.widget.ConstraintLayout x
    xmlns:app="http://schemas.android.com/apk/res-au
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".loginFragment">
```

```
<androidx.constraintlayout.widget.ConstraintLayout xml
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto
    tools:context=".menuFragment">
```

Aquí un par de ejemplo como los LinearLayout los he pasado a ConstraintLayout. Excepto en el fragment que carga RecyclerView, que lo tengo como un FrameLayout, porque lo he visto más oportuno que así sea.

10. 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)



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

Las pruebas las he grabado en vídeo para que veas el funcionamiento de la aplicación. Lo tendrás en el directorio raíz del proyecto.