31 DE DICIEMBRE DE 2022

ULPGC

MEMORIA PAMN

MUSEO VIRTUAL

ALEJANDRO DOMINGUEZ RUIZ Y DIEGO SIERRA LÓPEZ

IDEA

Tras una serie de codelabs realizados, y visualización de distintos videos de youtube para coger ideas, decidimos que la mejor idea era implementar el desarrollo del museo virtual.
Nombre: **Virtual Informatic Museum.**

Tras una deliberación entre los dos compañeros, decidimos cual debían ser las funciones básicas que debería contener la aplicación.
Esta debería contar con 3 puntos clave:

- Lector de QR
- Tours virtuales (videos con interés relevante)
- Categorías y artículos (novedades dentro del sector e investigaciones de interés)

Con estos puntos principales sobre nuestra aplicación, decidimos el resto de las funcionalidades que deberíamos añadir para darle un interés extra a la aplicación.

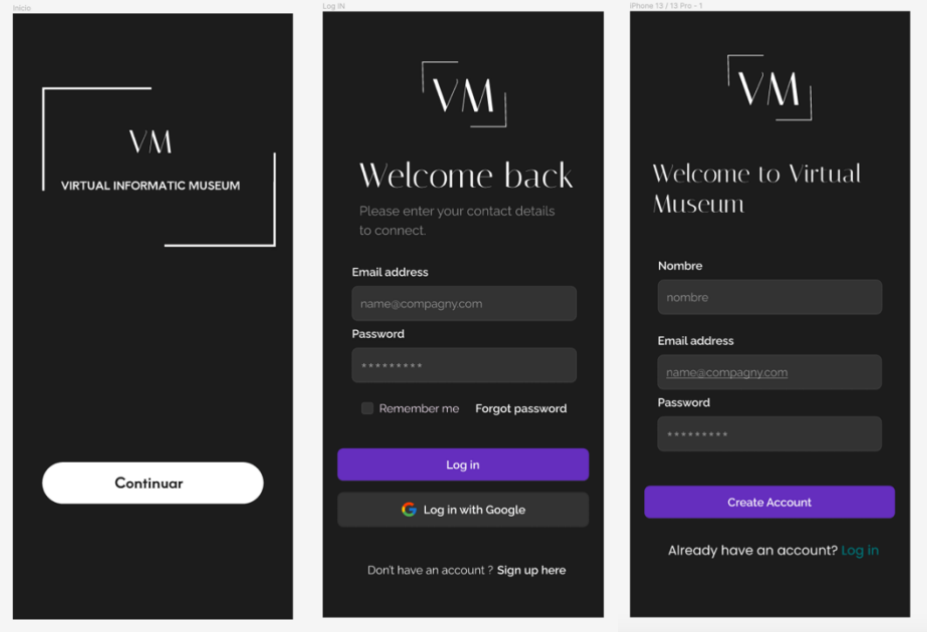
- Registro y Login
- Pantalla de perfil (se guardarán todos los videos, artículos y QR que a el usuario le gustaron).

Con estos puntos, consideramos la versión inicial sería suficiente para aportar un valor al usuario y que descargase nuestra aplicación.

DISEÑO APLICACIÓN

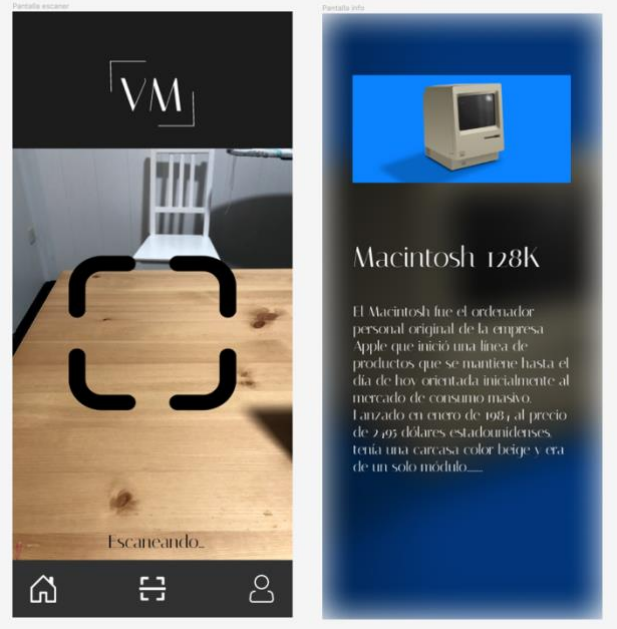
Para el desarrollo de la aplicación, utilizamos la herramienta **Figma**, creamos una premium con el correo institucional de la universidad, para poder trabajar los dos compañeros en el mismo proyecto.

Diseñamos 3 pantallas principales, una pantalla de carga, una pantalla de login y en caso de no estar registrados, una pantalla para registrase.
El color elegido para el fondo de la aplicación es un color “negro” con código 1C1C1C.

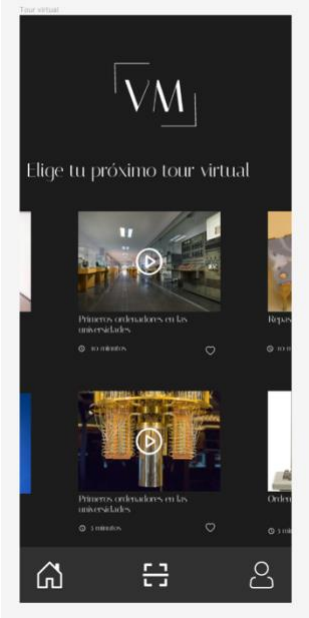


Tras registrase, accederíamos a la pantalla principal en la que tenemos las 3 funciones principales en 3 botones distintos.

- Botón QR: Activaríamos la cámara y al escanear el código QR, nos llevaría a la información sobre el dispositivo y su historia.



- Botón Tours: Mostrara una serie de videos interesantes de YouTube, a los que podremos darle “me gusta” para guardarlo.



- Botón categorías y artículos: Mostrara artículos interesantes al igual que las distintas categorías distintas donde encontraríamos artículos específicos.

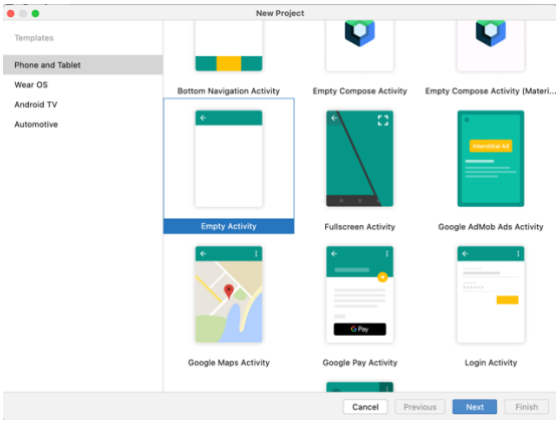
Para finalizar, tenemos una pantalla con el perfil del usuario donde podremos ver todos los videos, artículos y escáneres QR que nos hayan gustado. Además, podremos cambiar la foto de perfil.

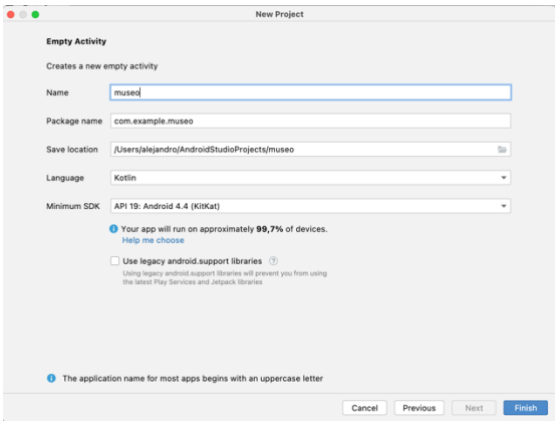
Para movernos entre todas las pantallas, utilizaremos un menú del el cual podremos acceder a todas las pantallas de una manera rápida y cómoda.

DESARROLLO APLICACIÓN

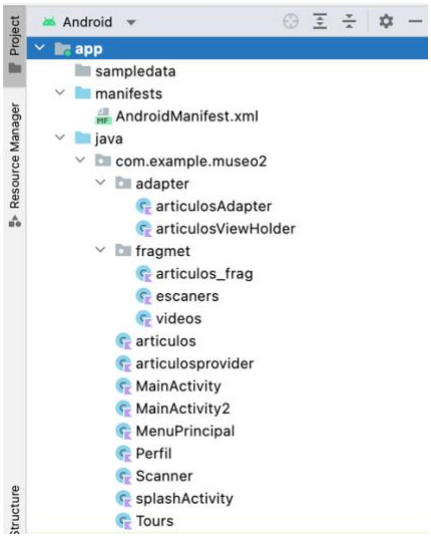
Para empezar el desarrollo de la aplicación, utilizamos el entorno Android Studio y el lenguaje de programación Kotlin.

Para empezar con el desarrollo del proyecto, creamos una nueva actividad vacía, y seleccionamos el lenguaje kotlin y el SDK “API 19 4.4” está en concreto es la versión KitKat y permite que nuestra aplicación este disponible en el 99,7% de los dispositivos.





Para la creación de la aplicación hemos utilizado acivities y fragmentos en caso de la pantalla de perfil, para reutilizar la actividad.



Configuración inicial y librerías

El primer paso, es modificar la configuración incial, es modificar el build gradel(:app), y activar multidexEnabled a true, para que no nos ocurra el error de exceder el límite de referencias de 64k.

```
android {
    namespace 'com.example.museo2'
    compileSdk 32

    defaultConfig {
        applicationId "com.example.museo2"
        minSdk 19
        targetSdk 32
        versionCode 1
        versionName "1.0"

        multiDexEnabled true
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
```

Así como añadir el viewBinding para poder trabajar de manera más cómoda al llamar a los objetos de XML.

```
    viewBinding {
        enabled = true
    }
```

Para poder utilizar la base de datos firebase, debemos seguir los pasos que nos indica el siguiente enlace, pulse aqui.

Añadimos el buildscript con el repositorio y las dependencias al build gradel del proyecto para que pueda funcionar Firebase así como dentro del build gradel de la app las dependencias para que pueda funcionar Firebase para la autenticación así como añadir cloud Firestore para poder almacenar las fotos y los “me gusta”.

```
// Top-level build file where you can add configuration options common to a
buildscript {
    repositories {
        google()  // Google's Maven repository
        mavenCentral()  // Maven Central repository

    }
    dependencies {
        // Add the dependency for the Google services Gradle plugin
        classpath 'com.google.gms:google-services:4.3.13'
    }
}

plugins {
    id 'com.android.application' version '7.3.0' apply false
    id 'com.android.library' version '7.3.0' apply false
    id 'org.jetbrains.kotlin.android' version '1.7.10' apply false
}
```

```
dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.5.1'
    implementation 'com.google.android.material:material:1.7.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'com.google.firebase:firebase-firestore-ktx:24.4.0'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.4'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.0'

    //Para la base de datos
    implementation 'com.google.firebase:firebase-analytics-ktx'
    implementation 'com.google.firebase:firebase-auth-ktx'
    implementation platform('com.google.firebase:firebase-bom:31.0.2')
    implementation 'com.google.firebase:firebase-database-ktx'
    implementation 'com.google.firebase:firebase-firestore:24.4.1'
```

Para poder utilizar el escáner, añadimos la librería Zxing, con esta librería, no necesitaríamos modificar los permisos para que la aplicación pueda acceder a la cámara.

```
58      //Para el escaner
59      implementation('com.journeyapps:zxing-android-embedded:4.3.0') { transitive = false }
60      implementation 'com.google.zxing:core:3.3.0'
```

Para poder implementar los videos en el apartado Tours, utilizamos la librería Android YouTube player, disponible toda la información en este enlace.

```
62      //Para videos de youtube desde el movil
63      implementation 'com.pierfrancescosoffritti.androidyoutubeplayer:core:11.1.0'
```

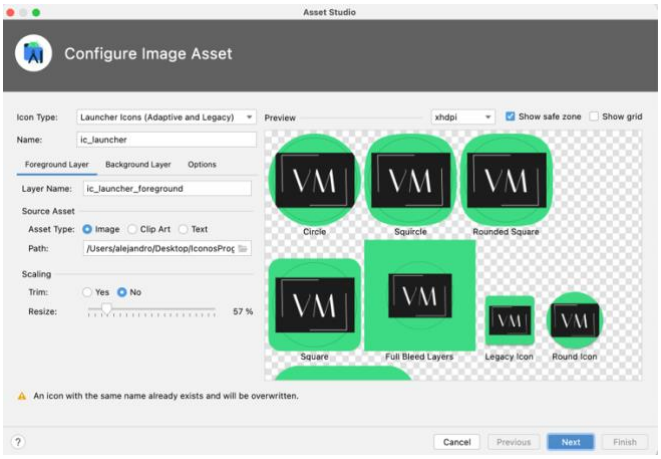
Para poder mostrar las vistas en los artículos, utilizamos la librería Glide, que nos permite mostrar y cargar imágenes de internet de forma rápida.

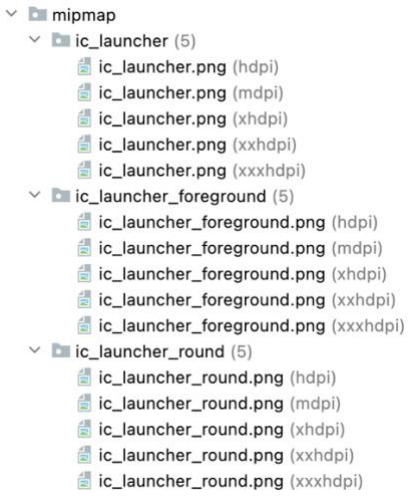
```
66      //Para fotos internet
67      implementation 'com.github.bumptech.glide:glide:4.14.2'
68      annotationProcessor 'com.github.bumptech.glide:compiler:4.14.2'
```

Por último, añadimos la librería de.hdodenhof, para poder mostrar la imagen de perfil de forma circular.

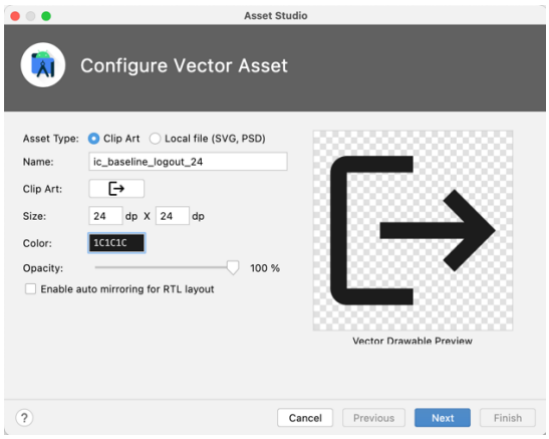
Iconos aplicaciones

Para crear el icono principal de la aplicación, añadimos un image Asset, donde añadiremos nuestro logo con diferentes formas y lo ajustamos al tamaño deseado. Esto se crea dentro de la carpeta mipmap.





Para la creación de los iconos que utilizamos dentro de la aplicación, creábamos un nuevo vector asset dentro de la carpeta drawable para la creación de estos iconos.



Activities y XML

Splash Screen

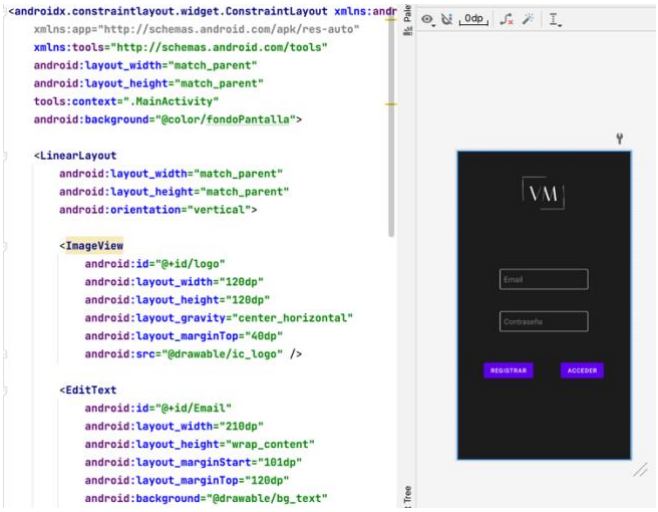
Esta será una pantalla de carga inicial que estará solamente unos segundos mientras se carga todo el contenido, por ello una vez iniciada, llamamos a la siguiente actividad que es la pantalla principal. En el XML, colocamlos el bitmap que es el logo de nuestra aplicación y modificamos el item con top, bottom, right y left para que se vea con un formato decente y no quede descuadrada.

```
class splashActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        startActivity(Intent( packageContext: this, MainActivity::class.java))

    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list
    android:opacity="opaque"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:drawable="@color/fondoPantalla"/>

    <item
        android:left="40dp"
        android:right="40dp"
        android:top="250dp"
        android:bottom="250dp"
        >
        <bitmap android:src="@drawable/ic_logo"/>
    </item>
</layer-list>
```

MainActivity (Pantalla inicio)

En el XML, utilizamos los linear layout para organizar el contenido dentro de la pantalla, los ImageView para colocar en todas las pantallas los logos y por último, los editText para poder escribir la información necesaria y Buttons para realizar las llamadas a los métodos.



```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:andr
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:background="@color/fondoPantalla">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/logo"
            android:layout_width="120dp"
            android:layout_height="120dp"
            android:layout_gravity="center_horizontal"
            android:layout_marginTop="40dp"
            android:src="@drawable/ic_logo" />

        <EditText
            android:id="@+id/Email"
            android:layout_width="210dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="101dp"
            android:layout_marginTop="120dp"
            android:background="@drawable/bg_text"
```

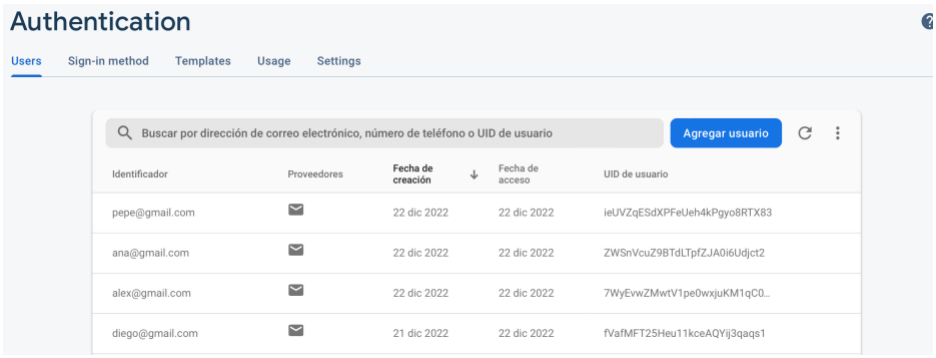
En la activity, creamos 2 variavle sque psoteriormente se utilizaran para realizar las llamadas a la base de datos para la autentificación y para la base de datos firestore. BindingView, lo utilizamos para las llamdas a los botones, una vez que se hace click.

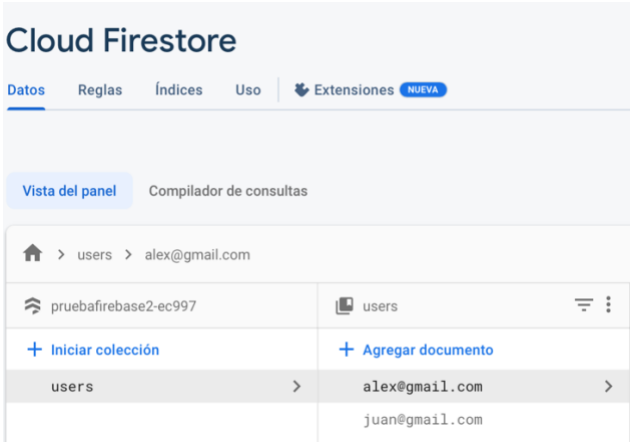
```
class MainActivity : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth

    private val db = FirebaseFirestore.getInstance()


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

        val binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        auth = FirebaseAuth.getInstance()

        binding.botonRegistrar.setOnClickListener {  it: View!
            val email = binding.Email.text.toString()
            val contraseña = binding.ContraseA.text.toString()
            println(email)
            println(contraseña)
            if (email.isNotEmpty() && contraseña.isNotEmpty()) {
                registerUser(email, contraseña)
            }
        }

        binding.BotonAcceder.setOnClickListener {  it: View!
            val email = binding.Email.text.toString()
            val contraseña = binding.ContraseA.text.toString()
            if (email.isNotEmpty() && contraseña.isNotEmpty()) {
                loginUser(email, contraseña)
            }
        }

    }
```

Métodos que destacar

Para iniciar sesión, realizamos la llamada a firebaseAuth, y esta comprueba en la base de datos si se encuentran los usuarios con ese email y contraseña.
En caso de no estar registrado, la llamada a firebaseAuth, crea el usuario y además crea ese usuario dentro de FirebaseFirestore.

```
private fun loginUser(email: String, contraseña: String) {
    FirebaseAuth.getInstance().signInWithEmailAndPassword(email, contraseña)
        .addOnCompleteListener { it: Task<AuthResult!>
            if (it.isSuccessful){...} else {
                showAlert()
            }
        }
}

private fun registerUser(email: String, contraseña: String) {
    FirebaseAuth.getInstance().createUserWithEmailAndPassword(email, contraseña)
        .addOnCompleteListener { it: Task<AuthResult!>
            if (it.isSuccessful) {
                db.collection( collectionPath: "users").document(email)
                abrir(email)
            } else {
                showAlert()
            }
        }
}
```





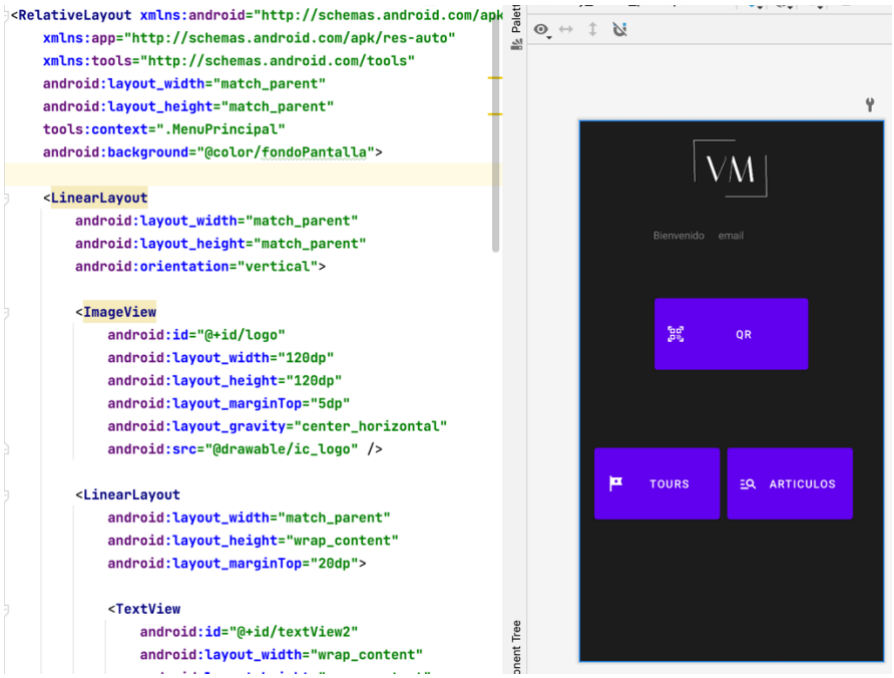
Menu

Creamos un XML de tipo Menu para poder utilizar ese menú para acceder a cada distinta pantalla de la aplicación.

```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/Home"
        android:title="Home"
        app:showAsAction="never" />

    <item
        android:id="@+id/Tours"
        android:title="ToursVirtuales"
        app:showAsAction="never" />

    <item
        android:id="@+id/Articulos"
        android:title="Articulos"
        app:showAsAction="never" />

    <item
        android:id="@+id/escaner"
        android:title="Escaner"
        app:showAsAction="never" />

    <item
        android:id="@+id/perfil"
        android:title="Perfil"
        app:showAsAction="never" />

</menu>
```

MenuPrincipal

En el XML, volvemos a utilizar los relativeLayout y linear layout para colocar el contenido.



En el activity, podemos destacar el paso del intent “email”, una vez iniciamos sesión, cogemos ese email para “saludarlo” y darle la bienvenida a ese usuario. Ese String “email”, lo colocamos en el textView.

```
class MenuPrincipal : AppCompatActivity() {

    private lateinit var binding: ActivityMenuPrincipalBinding

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


        binding = ActivityMenuPrincipalBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val bundle:Bundle? = intent.extras
        val email:String? = bundle?.getString( key: "email")

        setup()

        binding.botonQR.setOnClickListener { abrirEscaner() }
        binding.botonCategorias.setOnClickListener { abrirCat() }
        binding.botonTours.setOnClickListener { abrirTours() }


    }
```

Métodos que destacar

Método para seleccionar otra pantalla a través del menú. Este método se encuentra en todas las clases, y una vez que seleccionamos otra pantalla llamamos a su método e iniciamos su actividad.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    return when (item.itemId) {
        R.id.Home-> true
        R.id.Articulos -> {
            abrirCat()
            true
        }
        R.id.perfil -> {
            abrirPerfil()
            true
        }
        R.id.Tours -> {
            abrirTours()
            true
        }
        R.id.escaner -> {
            abrirEscaner()
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}
```

En caso de seleccionar el perfil, creamos un intent para iniciar la actividad y le pasamos ese email del usuario.

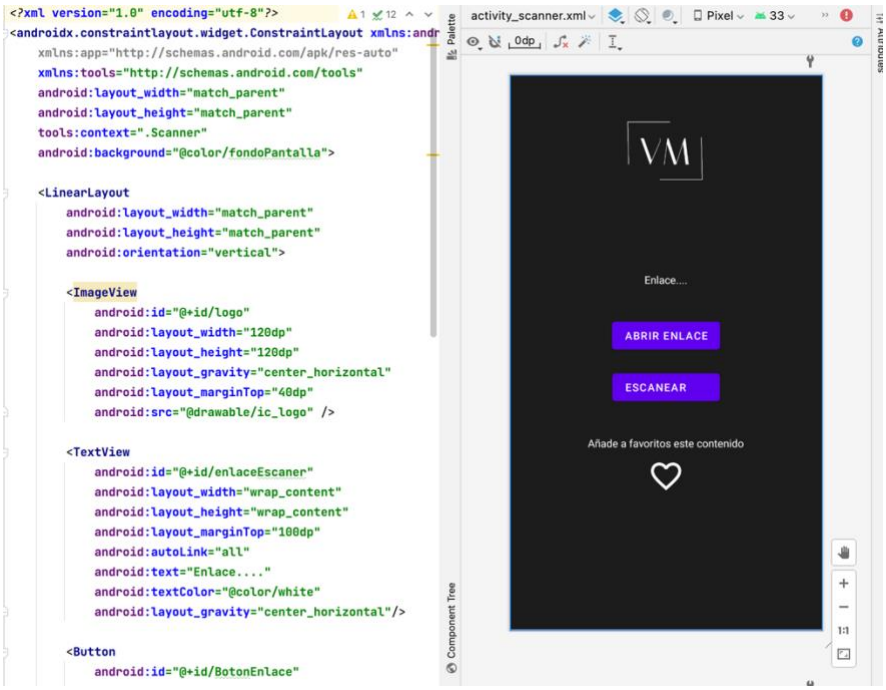
```
private fun abrirPerfil() {

    val mail = intent.getStringExtra( name: "email")

    val homeIntent = Intent( packageContext: this, Perfil::class.java).apply {  this: Intent
        putExtra( name: "email", mail)
    }
    startActivity(homeIntent)
}
```

Mismo método para abrir, Tours, Artículos y Escáner.

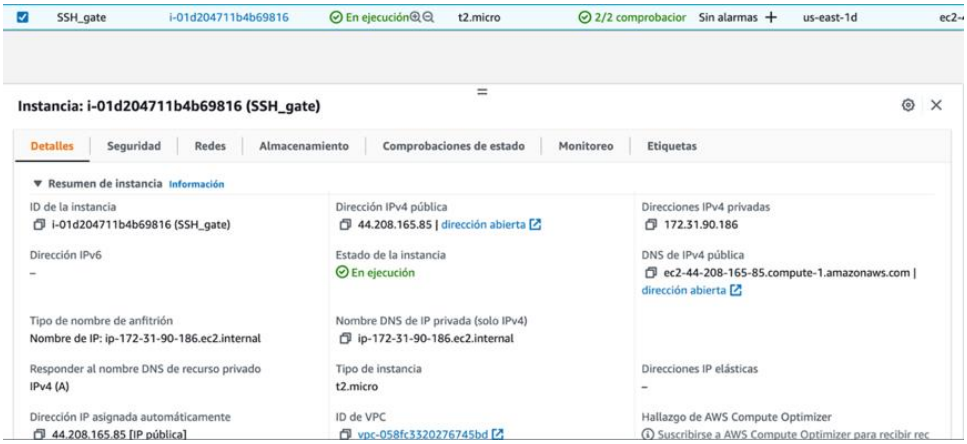
```
private fun abrirTours() {

    val mail = intent.getStringExtra( name: "email")

    val homeIntent = Intent( packageContext: this, Tours::class.java).apply {  this: Intent
        putExtra( name: "email", mail)
    }
    startActivity(homeIntent)
}
```

Escáner (Scanner)

Una vez, pulsamos el botón de QR, se activa la cámara que detectara solo códigos QR, una vez detecta uno, en el textView aparece la url del enlace y para acceder solo hay que pulsar el botón abrir enlace.



Para esta aplicación, desarrollamos una series de códigos QR que nos llevasen a un servidor web donde tendríamos la información sobre el dispositivo escaneado.
Este servidor web fue creado con las instancias de EC2, que nos proporcionaba la cuenta de estudiante de Amazon.



Una vez creada la instancia, solo hay que crear el servidor web y crear un HTML dentro de esta. El resultado final es:

En el activity, una vez que se crea la actividad el primer método que llamamos es escaner(), este inicia el escaner de forma directa para que se escanee el código QR. En caso de tocar el botón abrir enlace, llamaremos al método abrirEnlace(), y en caso de querer volver a abrir la cámaram tenemos que tocar el boton escanear.

```
import ...

class Scanner : AppCompatActivity() {

    private lateinit var binding: ActivityScannerBinding

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

        binding = ActivityScannerBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val bundle:Bundle? = intent.extras
        val email:String? = bundle?.getString( key: "email")


        //llamada metodo escaner nada mas iniciar activity
        escaner()

        binding.BotonEnlace.setOnClickListener { it: View!
            abrirEnlace()
        }
        binding.likeBoton.setOnClickListener{ it: View!
            añadirEscaner()
        }
        binding.AbrirScanner.setOnClickListener { it: View!
            escaner()
        }


    }

```

Métodos a destacar

Una vez que iniciamos la actividad, llamamos al método escanear, este activara la cámara para detectar los códigos QR. Ponemos por defecto que el flas no salte y suene un beep cuando detecte un código qr.
Una vez lo tenemos escaneado, transformamos el resultado a un texto que colocaremos en el textview.
En caso de cancelar el escaner, mostramos un mensaje en la pantalla con el texto “cancelado”.

```
    private fun escaner(){
        IntentIntegrator( activity: this)
            .setDesiredBarcodeFormats(IntentIntegrator.QR_CODE)
            .setPrompt("Escaneando....")
            .setTorchEnabled(false)
            .setBeepEnabled(true)
            .initiateScan()
    }
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        val result = IntentIntegrator.parseActivityResult(requestCode, resultCode,data)

        if(result != null){
            if(result.contents !== null){
                //el text view donde se muestra el enlace.
                binding.enlaceEscaner.text = result.contents

            }else{
                Toast.makeText( context: this,  text: "cancelado" , Toast.LENGTH_LONG).show()
                }
            }
            super.onActivityResult(requestCode, resultCode, data)
    }
```

Al pulsar el botón abrirEnlace, comprobamos primero que tenemos un link en el text view, en caso de que se encuentre, transformamos ese string en un intent.data e iniciamos la actividad con el intent.
En este caso, se nos abrirá el url con el HTML que hemos creado.

```
47      private fun abrirEnlace() {
48
49          if(binding.enlaceEscaner.text.toString().equals("")){
50              Toast.makeText( context: this,  text: "No hay ningun enlace escaneado" , Toast.LENGTH_LONG).show()
51          }else{
52              val intent = Intent(Intent.ACTION_VIEW)
53              intent.data = Uri.parse(binding.enlaceEscaner.text.toString())
54              startActivity(intent)
55
56          }
57      }
58
```

Tours

En el XML, hemos utilizado la técnica del ScrollView. Esta nos permite deslizarnos en la pantalla. Dentro del XML también utilizamos el Linear Layout para colocar todos los videos con el mismo formato y utilizamos la Liberia Android YouTube player para mostrar los videos.
Con esta librería, solo tenemos que indicar el el app:videoId, los últimos caracteres del url del video de youtube y de forma automática lo busca y lo pone a reproducir.

```
<ScrollView xmlns:android="http://schemas.android.com/apk/re
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Tours"
    android:orientation="vertical"
    android:background="@color/fondoPantalla">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

    <com.pierfrancescosoffritti.androidyoutubeplayer.core.pl
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:videoId="lsAeTXNQyIA"
        app:autoPlay="false"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_marginTop="10dp">
    </com.pierfrancescosoffritti.androidyoutubeplayer.core.p

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="5dp"
```

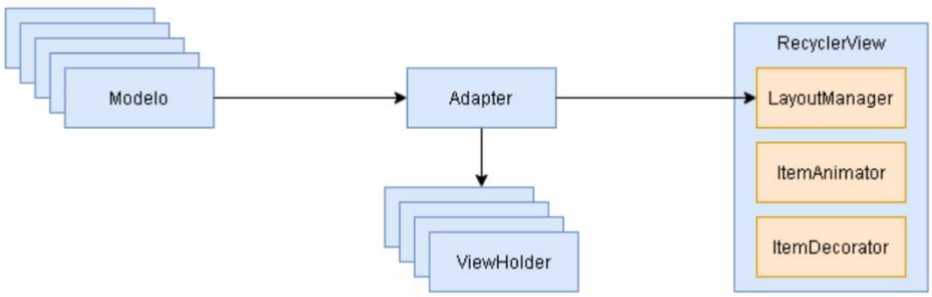
En cuanto al activity, nos encontramos los métodos simples para todos, ya que solo le pasos el intent del email y nos encontramos con el menú para cambiar las pantallas.

```
9  class Tours : AppCompatActivity() {
10     override fun onCreate(savedInstanceState: Bundle?) {
11         super.onCreate(savedInstanceState)
12         setContentView(R.layout.activity_tours)
13
14         val bundle:Bundle? = intent.extras
15         val email:String? = bundle?.getString( key: "email")
16
17     }
18
19     override fun onCreateOptionsMenu(menu: Menu): Boolean {
20         menuInflater.inflate(R.menu.menu,menu)
21         return true
22     }
23
```

Artículos

Para la realización del apartado artículos, hemos utilizado la técnica del reclycled view, para ello debemos tener en cuenta las siguientes clases y XML.

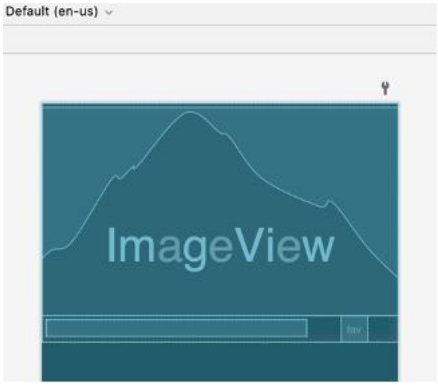
- Datos: Determinan la colección de artículos que se quiere mostrar sobre los ítems (listas).
- RecyclerView: Es el contenedor donde se mostraran el resultado final.
- RecyclerView.ViewHolder: Contiene la referencia del view creado para un ítem y la información de su lugar en el RecyclerView. Esto te permitirá vincular directamente tu información sobre los widgets del ítem.
- RecyclerView.Adapter: Colabora con el ViewHolder para convertir una colección de datos en una lista de views que serán añadidos al RecyclerView y enlazados con dichos datos.
- RecyclerView.LayoutManager: Es el responsable de medir y posicionar los ítems dentro del RecyclerView. Además, determinar cuando los ítems no son visible para el usuario. Existen varias implementaciones de esta clase para permitirnos crear listas, grillas o estructuras escalonadas.



Item_Articulos

En este XML, creamos la vista de cómo se va a visualizar el artículo, este está formado por un ImageView, un textView para el título del artículo y un botón de like.

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <ImageView
        android:id="@+id/idImagen"
        android:layout_width="match_parent"
        android:layout_height="240dp"
        android:layout_marginTop="5dp"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <TextView
            android:id="@+id/idArticulo"
            android:layout_width="300dp"
            android:layout_height="wrap_content"
            android:layout_marginTop="5dp"
            android:layout_marginLeft="5dp"
            android:textColor="@android:color/tertiary_text_light"
            android:textColorHint="@android:color/tertiary_text_light"/>

        <ImageButton
            android:id="@+id/fav"
            android:layout_width="30dp"
            android:layout_height="30dp"
```



Artículos

Esta clase “data”, sirve para almacenar los valores de los artículos que estarán compuestos por una foto, un titulo y su url de internet.

Scanner.kt | articulos.kt | activity_

```kotlin
package com.example.museo2

data class articulos (
    val titulo:String,
    val foto:String,
    val url:String)
```

ArticulosProvider

En esta clase, creamos una lista con el total de artículos y atributos para más tarde poder cogerlos y mostrarlos.

```kotlin
package com.example.museo2

class articulosprovider {
    companion object{
        var articulosLista = listOf<articulos>(
            articulos(
                titulo: "noticia1",
                foto: "https://www.fundacionaquae.org/wp-content/uploads/2020/06/superordenador.jpg",
                url: "https://www.technologyreview.es//s/14835/el-ultimo-desafio-de-la-ia-calcular-su-propia-huella-de-carbono
            ),
            articulos(
                titulo: "noticia2",
                foto: "https://img.blogs.es/intel/wp-content/uploads/2019/12/microcomputer-ibm-model-5140-1986-672541-medium-1
                url: "https://territoriointel.xataka.com/historia-ordenador-experiencia-nueve-profesionales-ambitos-muy-dispar
            ),
            articulos(
                titulo: "noticia3",
                foto: "https://d2jhl5pzkfi24b.cloudfront.net/article_image/0001/04/3e27a83dfb644c9f0dfe01ea4221bcd6c1f3e4ac.jp
                url: "https://www.technologyreview.es//s/14835/el-ultimo-desafio-de-la-ia-calcular-su-propia-huella-de-carbono
            ),
```

ArtículosAdapter

Esta clase de manera simple permite coger un listado en este caso de artículos y transfórmarlos a un recyclerview. Esta clase extiende de RecyclerView.adapter y a su vez de viewHolder.

```kotlin
class articulosAdapter(private val articulosLista:List<articulos>) : RecyclerView.Adapter<articulosViewHolder>(){


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

    override fun onBindViewHolder(holder: articulosViewHolder, position: Int) {
        val item = articulosLista[position]
        holder.render(item)

    }

    override fun getItemCount(): Int = articulosLista.size

}
```

Métodos que destacar

onCreateViewHolder, devuelve al view holder cada item de articulos

```kotlin
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): articulosViewHolder {
    val layoutInflater = LayoutInflater.from(parent.context)
    return articulosViewHolder(layoutInflater.inflate(R.layout.item_articulos,parent, attachToRoot: false))
}
```

onBindViewHolder, pasa por cada ítem y llama a la función render de la clase viewHolder.

```
override fun onBindViewHolder(holder: articulosViewHolder, position: Int) {
    val item = articulosLista[position]
    holder.render(item)

}
```

ArticulosViewHolder

```
import ...

class articulosViewHolder (view: View) : RecyclerView.ViewHolder(view ){

    val binding = ItemArticulosBinding.bind(view)


    fun render(articulosModel: articulos) {
        binding.idArticulo.text = articulosModel.titulo
        Glide.with(binding.idImagen.context).load(articulosModel.foto).into(binding.idImagen)


        itemView.setOnClickListener() { it: View!
            val intent = Intent(Intent.ACTION_VIEW)
            intent.data = Uri.parse(articulosModel.url)

            itemView.context.startActivity(intent)

        }
    }
}
```

Métodos a destacar

Esta función se llama por cada ítem de artículos que se pasa desde el adapter y asignamos a los text e imágenes sus partes de los “articulosModel”

```
fun render(articulosModel: articulos) {
    binding.idArticulo.text = articulosModel.titulo
    Glide.with(binding.idImagen.context).load(articulosModel.foto).into(binding.idImagen)


    itemView.setOnClickListener() { it: View!
        val intent = Intent(Intent.ACTION_VIEW)
        intent.data = Uri.parse(articulosModel.url)

        itemView.context.startActivity(intent)

    }
}
```

MainActivity2(clase principal articulos)

En esta clase, se encuentra el XML donde va a aparecer todos los artículos asi como las llamadas a los métodos de las clases adapter y viewholder.
Como hemos dicho, utilizamos la técnica del recyclerview para mostrar todo el contenido.

```xml
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity2"
    android:background="@color/fondoPantalla">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerArticulos"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

    </androidx.recyclerview.widget.RecyclerView>

```

Para la clase, utilizamos una nueva actividad desde la cual llamaremos a los métodos del adaptador

```kotlin
package com.example.museo2

import ...

class MainActivity2 : AppCompatActivity() {


    private lateinit var binding: ActivityMain2Binding


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMain2Binding.inflate(layoutInflater)
        setContentView(binding.root)

        val bundle:Bundle? = intent.extras
        val email:String? = bundle?.getString( key: "email")


        initRecycledView()


    }
```

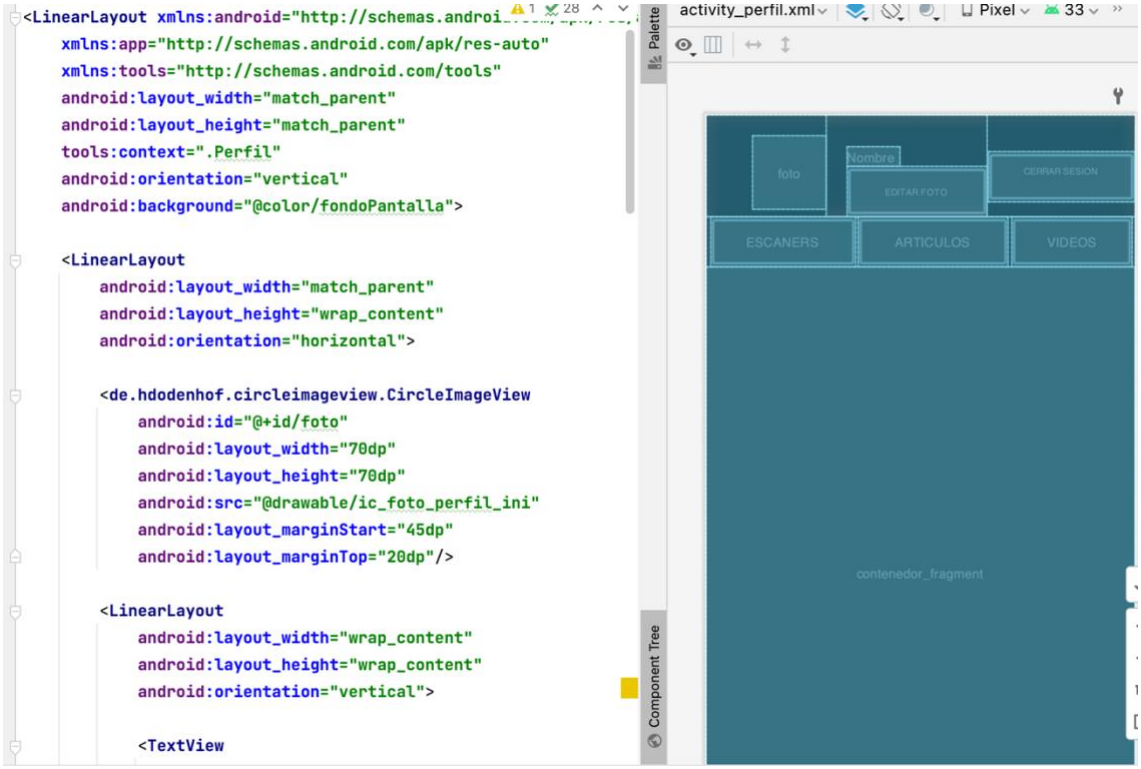
Métodos que destacar

La función initRecycledView se ejectua nada más se carga la activity, recupera el recyclerview y llamamos a la función de adapter donde le pasamos la lista por parámetros.

```
fun initRecycledView(){

    val manager = LinearLayoutManager( context: this)
    val decoration = DividerItemDecoration( context: this,manager.orientation)
    binding.recyclerArticulos.layoutManager = manager
    binding.recyclerArticulos.adapter  = articulosAdapter(articulosprovider.articulosLista)

    binding.recyclerArticulos.addItemDecoration(decoration)


}
```

Perfil

El XML de la activity perfil, esta formado por diferentes linearLayout para organizar los botones, textView y el fragment. Para el fragment inferior utilizamos un frameLayout, este ira cambiando en función de pulsar el boton escaner, artículos y vídeos.



Para esta pantalla como hemos dicho, utilizamos una activity y 3 fragmentos para mostrar los articulos, esceners y videos guardados.
Una vez se carga la actividad, cargamos el email del usuario y cargamos la foto del usuario con el método obtenerfoto(email). A parte iniciamos el fragmento y ponemos videos_frag como inicial.
Iniciamos la base de datos firebaseFirestore para poder obtener los datos de la foto y cargar todos los archivos que se han guardado.

```
val escaner_frag = escaners()
val articulos_frag = articulos_frag()
val videos_frag = videos()

private lateinit var binding :ActivityPerfilBinding
private val db = FirebaseFirestore.getInstance()

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

    val bundle:Bundle? = intent.extras
    val email:String? = bundle?.getString( key: "email")
    binding.textViewNombre.text = email

    val fragmentManager = supportFragmentManager
    val fragmentTransaction = fragmentManager.beginTransaction()
    fragmentTransaction.replace(R.id.contenedor_fragment, videos_frag)
    fragmentTransaction.addToBackStack( name: null)
    fragmentTransaction.commit()

    obtenerfoto(email)

    binding.cambiarFoto.setOnClickListener{ it: View!
        tomarfoto()
    }
    binding.cerrar.setOnClickListener { it: View!
        cerrar()
    }
```

Métodos que destacar

Para obtener la foto de perfil, debemos llamar a la base de datos, y coger la imagen. Esta se encuentra almacenada como un string, por tanto debemos codificar ese string en base 64 para formar un bitmap de la imagen. Una vez ya la tenemos, la mostramos en el circleImageView.

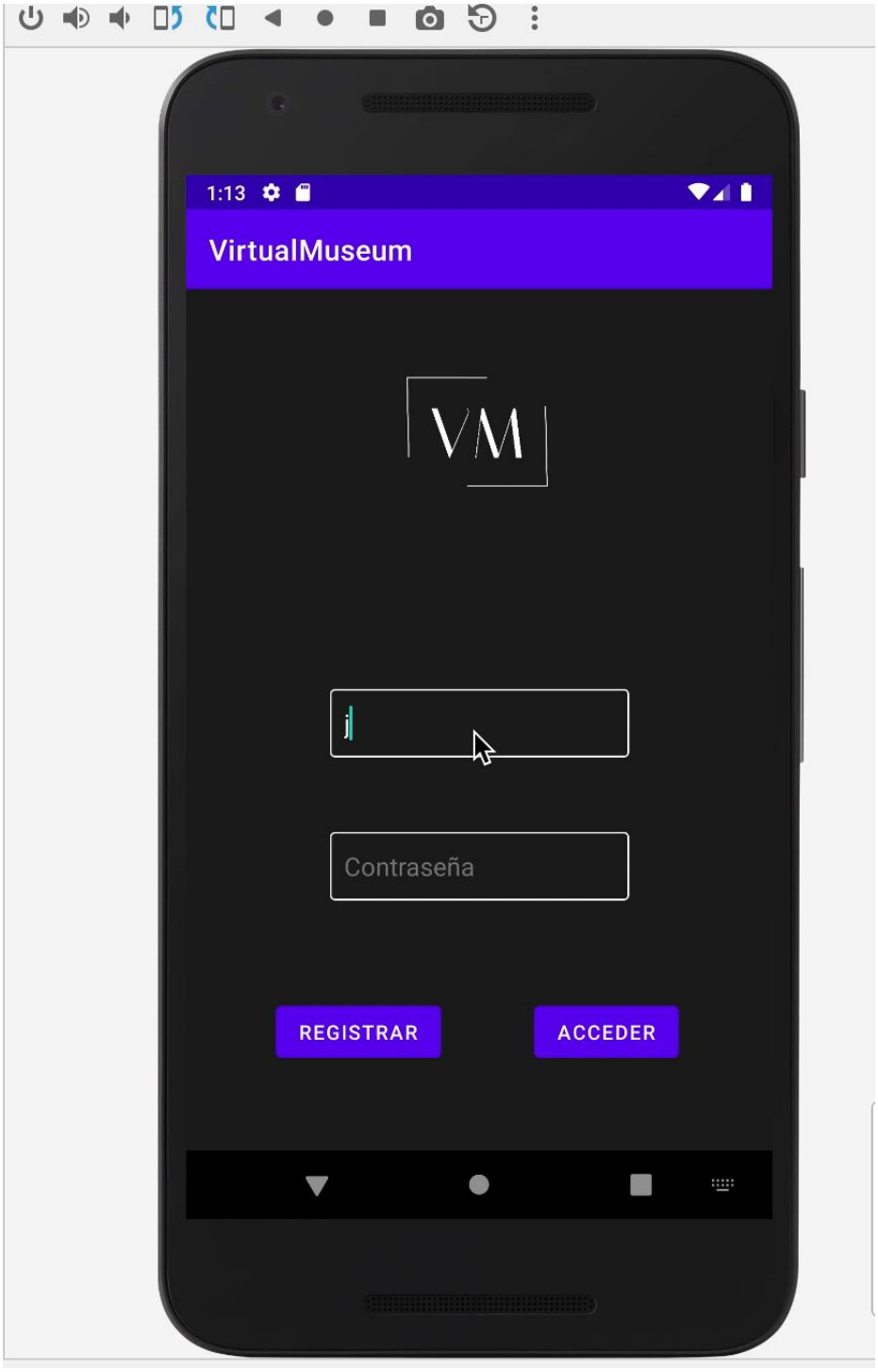
```
private fun obtenerfoto(email: String?) {
    if (email != null) {
        db.collection( collectionPath: "users").document(email).get().addOnSuccessListener { it: DocumentSnapshot!

            val imageBytes = android.util.Base64.decode(it.get("imagen").toString(), flags: 0)
            val image = BitmapFactory.decodeByteArray(imageBytes, offset: 0, imageBytes.size)

            println(image)

            binding.foto.setImageBitmap(image)

        }
    }
}
```

En caso de querer tomar una foto nueva, llamamos al método tomarfoto(), este realizara una llamada al método startForResult.Launch, que nos abrirá la cámara para poder realizarnos una foto. Una vez tenemos esa foto, se guarda como un bitmap y por tanto debemos transformala a string para poder guardarla en la base de datos.
Para ello llamao al método BitMapToString que transforma ese bitmap de base 64 en un string para poder guardarlo de forma correcta en la base de datos.
Una vez ya lo tenemos transformado, llamamos a la base de datos a la colección de usuarios y que tiene ese email y le añadimos al hasMapOf únicamente la foto.
Este hasMapOf se crea para poder añadir mas atributos a ese usuario.

```
    private fun tomarfoto() {
        startForResult.launch(Intent(MediaStore.ACTION_IMAGE_CAPTURE))
    }
    private val startForResult = registerForActivityResult(ActivityResultContracts.StartActivityForResult()){
        result: ActivityResult ->
        if(result.resultCode == Activity.RESULT_OK){
            val intent = result.data
            val imageBitmap = intent?.extras?.get("data") as Bitmap
            binding.foto.setImageBitmap(imageBitmap)
            guardarbd(imageBitmap)
        }
    }
    private fun guardarbd(imageBitmap: Bitmap) {
        val mail = intent.getStringExtra( name: "email")

        val imagen = BitMapToString(imageBitmap)

        if (mail != null) {
            db.collection( collectionPath: "users").document(mail).set(
                hashMapOf("imagen" to imagen)
            )
        }
    }

    fun BitMapToString(bitmap: Bitmap): String {
        val baos = ByteArrayOutputStream()
        bitmap.compress(Bitmap.CompressFormat.PNG,  quality: 100, baos)
        val b = baos.toByteArray()
        return android.util.Base64.encodeToString(b,  flags: 0)
    }
```

Aplicación en funcionamiento

Aquí mostramos con el emulador el uso de la aplicación final
En caso que no se pueda visualizar, dejamos el video en el repositorio.



Algunos problemas, soluciones y conclusiones

Los problemas que nos hemos ido encontrado a lo largo de este proyecto han ido varios. En un primer momento empezamos con la realización de codelabs propuestos en clase. Después de una semanas practicando las cosas básicas para la creación de un proyecto, decidimos inicializar el proyecto de nuestra aplicación.
Como veíamos que los codelabs requerían demasiada cantidad de tiempo, nos pasamos a la visualización de videos de YouTube para implementar todas las funciones necesarias en nuestra aplicación.
Como problema principal que tuvimos, destacamos la conexión a la base de datos firebase y firestore que no conseguíamos conectar. Esto se debió a que no estábamos modifacion bien el archivo buildgrade de la app. Tras solucionar estos problemas, nos dimos cuenta de la facilidad que nos proporcionaba firebase.
Por otro lado, tuvimos bastantes problemas para implementar los videos de YouTube, esto se debe a que la API de YouTube estaba algo anticuada y nos daba diferentes problemas que solucionamos con una nueva librería Android YouTube player. En este punto, decidimos implementar un scrollview ya que nos parecía la opción más fácil en un principio.
A media que íbamos avanzando en el proyecto, aprendíamos nuevas técnicas y más especializadas como en este caso el recyclerview, que utilizamos para la visualización de artículos.
Como problema final que nos encontramos, fue subir a la base de datos la imagen (foto de perfil del usuario). Esto se debe a que no se podía subir como un bitmap de imagen y por tanto había que decodificarlo, el problema nos lo encontramos al usar la versión de Android 4.4, esto se debe a que las versiones más recientes permiten de una manera más simple decodificarlo y codificarlo.
Por último, nos encontramos el problema de poder guardar los escáneres, artículos y videos con el botón de like para poder mostrarlo más tarde en los fragmentos. Este problema no lo conseguimos resolver debido a la falta de tiempo y al conocimiento nulo sobre kotlin, firebase y Android studio y por tanto debimos obligados a dedicar más tiempo a poder entender cómo funcionaba todo.
Como resultado de todo esto, podemos llegar a la conclusión de que kotlin es uno de los lenguajes que más auge está cogiendo en estos años, el desarrollo de aplicaciones móviles nativas no es tan complejo como parece debido a la gran cantidad de librerías y bases de datos que nos permiten esta creación de una manera sencilla.
En nuestro caso hemos aprendido desde 0 a cómo funciona todo Android studio y con más dedicación podríamos llegar a realizar aplicaciones más complejas de una manera sencilla y rápida.