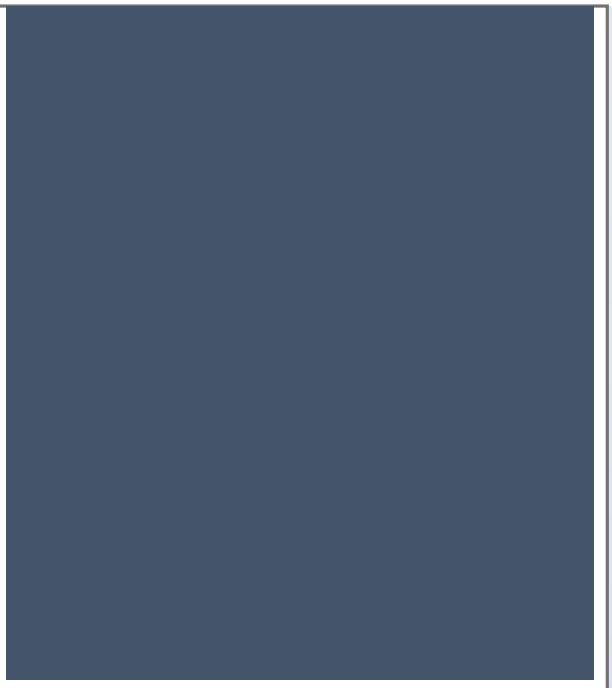


# Programación multimedia y dispositivos móviles

## Actividad 3.4. Acceso y toma de imágenes

Francisco José García Cutillas | 2FPGS_DAM

# Índice

# Ejercicio 1

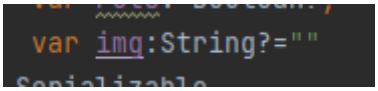
**Vamos a completar la aplicación de parques añadiendo imágenes de estos. Para ello, tomaremos fotografías utilizando la cámara de nuestro dispositivo solicitaremos los permisos adecuados según la operación a realizar y finalmente almacenaremos la información en la base de datos.**

**CONSEJOS:**

**La solución de este ejercicio implica tocar varios aspectos de toda la aplicación. Entre ellos destacar:**

- **Modificar el modelo, de modo que después de tomar la captura almacenemos otro campo de tipo String con la ubicación de la imagen en el dispositivo. Además , este cambio implicará migrar la versión de la base de datos.**

  Para modificar el modelo, debemos modificar, valga la redundancia, la clase Park contenida dentro del paquete db.
  A esta clase le vamos a añadir una nueva variable en la que se guardará un dato de tipo String, el cual será el nombre de la fotografía tomada. Pero por ahora tendrá por defecto una cadena vacía.



Como se ha comentado en el enunciado, la adición de un nuevo campo a nuestra base de datos conlleva a la actualización de la versión de la misma. Por lo tanto, nos vamos a la clase ParksDB.kt también dentro del paquete db y realizamos lo siguiente:

```kotlin
package com.mgh.pmdm.parques.db

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import androidx.room.migration.Migration
import androidx.sqlite.db.SupportSQLiteDatabase

//Cambiamos la versión de la base de datos por la versión 2
@Database(entities = [Park::class], version = 2)

abstract class ParksDB : RoomDatabase() {

    abstract fun parkDao(): ParkDAO
}

// Creamos la migración de la versión 1 a la 2
val MIGRATION_1_2 = object : Migration(1, 2) {
    override fun migrate(database: SupportSQLiteDatabase) {
        database.execSQL( sql: "ALTER TABLE Park ADD img VARCHAR")
    }
}

object DatabaseBuilder {

    private var INSTANCE: ParksDB? = null

    fun getInstance(context: Context): ParksDB {
        if (INSTANCE == null) {
            synchronized(ParksDB::class) {
                INSTANCE = buildRoomDB(context)
            }
        }
        return INSTANCE!!
    }

    //Al constructor le vamos a añadir la migración de la base de datos
    private fun buildRoomDB(contexto: Context) =
        Room.databaseBuilder(
            contexto.applicationContext,
            ParksDB::class.java,
             name: "parks-db"
        ).addMigrations(MIGRATION_1_2).build()

}
```

- **Invocar las acciones correspondientes para obtener la fotografía desde la cámara y guardarla en el dispositivo gestionando los permisos de forma adecuada.**

  Ahora para poder realizar fotos con la cámara, necesitamos tener acceso a la misma y también poder escribir en el sistema de archivos para guardar las fotografías. Por ello, vamos a necesitar un cargador para comprobar los permisos y otro para hacer las fotos.
  En primera instancia vamos a configurar en nuestro fichero AndroidManifest.xml un proveedor de almacenamiento en nuestra aplicación, añadiendo lo siguiente:

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="${applicationId}.provider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/provider_paths" />
</provider>
```

  Además de eso, vamos a crear dentro del directorio res otro llamado xml. Dentro de xml vamos a crear el fichero provider_paths.xml, al que le añadiremos:

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-files-path name="external_files" path="."/>
</paths>
```

  Con esto le estamos indicando al FileProvider qué contenido se le puede solicitar. En este caso será desde el raíz del área de almacenamiento externo de la aplicación.

  Una vez realizado todo lo anterior, debemos dar los permisos correspondientes para poder realizar fotografías con la cámara. Todo esto lo haremos en primer lugar añadiendo lo siguiente a nuestro AndroidManifest.xml:

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

El siguiente paso será ir a la clase SecondFragment.kt dentro del paquete view -> ui y añadimos lo siguiente:

```
// Con esto vamos a comprobar los permisos de la cámara y el almacenamiento
private val cameraPermissionRequest = registerForActivityResult(
    ActivityResultContracts.RequestMultiplePermissions()
) { permissions ->
    if (permissions.getOrDefault(android.Manifest.permission.CAMERA, defaultValue: false) &&
        permissions.getOrDefault(android.Manifest.permission.WRITE_EXTERNAL_STORAGE, defaultValue: false)
    )
    // Si el permiso está ok, llamamos al método de tomar foto
        takePhoto()
    else {
        // Si no, lanzamos un error
        Log.w( tag: "[ Permisos de cámara ]", msg: "No tienes permiso de cámara")
    }
}
```

Con esto vamos a controlar si la aplicación tiene o no permisos de cámara y almacenamiento. Debemos tener en cuenta que para poder realizar esto, nuestro minSdk debe ser el 24 o superior, de lo contrario no podríamos utilizar el método getOrDefault.

Ahora dentro del método onViewCreated vamos a añadir la opción para que al pulsar la imagen, se determine si hay o no permisos de cámara y almacenamiento.

```
//Cuando pulsamos sobre la imagen, se comprueban los permisos
binding.Image.setOnClickListener { it: View!
    cameraPermissionRequest.launch(
        arrayOf(
            android.Manifest.permission.CAMERA,
            android.Manifest.permission.WRITE_EXTERNAL_STORAGE
        )
    )
}
```

Una vez realizado todo lo anterior, vamos a proceder con el código (seguimos en la clase SecondFragment.kt) necesario para tomar una foto. Primero vamos a crear dos variables en las que se guardarán la ruta de la imagen en formato URI y en tipo string.

```
// Variables para guardar la ruta de la imagen
private var latestTmpUri: Uri? = null
private var tmpUri = ""
```

Una vez hecho esto, creamos el método para tomar una foto.

```
// Método para tomar una foto
private fun takePhoto() {
    /* Al igual que en el ejercicio anterior, vamos a crear una variable en la que
    *  guardaremos la instancia del directorio donde guardar las fotos */
    val dir = requireActivity().getExternalFilesDir(Environment.DIRECTORY_PICTURES)

    // Generamos un fichero temporal con un nombre, el cual será el prefijo y una extensión
    val file = File.createTempFile( prefix: "parques",  suffix: ".jpg", dir)

    // Obtenemos la ruta en la que se va a almacenar la foto
    latestTmpUri =
        FileProvider.getUriForFile(
            requireActivity(),
            authority: requireActivity().applicationContext.packageName + ".provider",
            file
        )

    // Le proporcionamos la ruta al cargador de la cámara para que guarde ahí la imagen
    cargadorCamara.launch(latestTmpUri)
}
```

Finalmente crearemos el código responsable de lanzar la cámara.

```
private val cargadorCamara = registerForActivityResult(
    ActivityResultContracts.TakePicture()
) { result: Boolean ->
    if (result) {
        binding.Image.setImageURI(latestTmpUri)
        tmpUri = latestTmpUri.toString()
        viewModel.currentPark.value?.img=tmpUri;
    }
}
```

- **Modificar el segundo fragmento para ofrecer algún mecanismo para tomar una imagen, como, por ejemplo, hacer clic sobre la imagen, y hacer que esta se muestre en lugar de la imagen por defecto.**

  Con lo realizado anteriormente ya tendríamos la imagen tomada y guardada en el almacenamiento del teléfono. Pero nos faltaría guardar la misma en la base de datos de la aplicación.
  Siguiendo en la clase SecondFragment.kt, en el método onPositiveClick, cuando creemos un nuevo parque en el campo img, pondremos lo siguiente:

```
img = tmpUri?:""
```

  Con esto ya podríamos guardar la imagen en la base de datos.

Una vez hecho esto, tenemos que cargar la imagen tanto en la interfaz de parques como en la de editar parque. Primero, puesto que estamos en la clase SecondFragment.kt, vamos a empezar por ahí. Para cagar la imagen, tenemos que actualizar el contenido del ImageView dentro del método prepareObservers:

```
fun prepareObservers() {
    //Observador de la vista
    viewModel.currentPark.observe(viewLifecycleOwner) { it: Park?
        it?.let { mypark ->
            // Si el texto de la imagen no está vacío, carga imagen
            if (mypark.img!="") binding.Image.setImageURI(Uri.parse(mypark.img))
            // Si no, cargas imagen por defecto
            else binding.Image.setImageResource(R.drawable.appimg)
```

Ahora vamos a la clase ParkViewHolder.kt dentro del paquete viewmodel, y le añadimos el atributo:

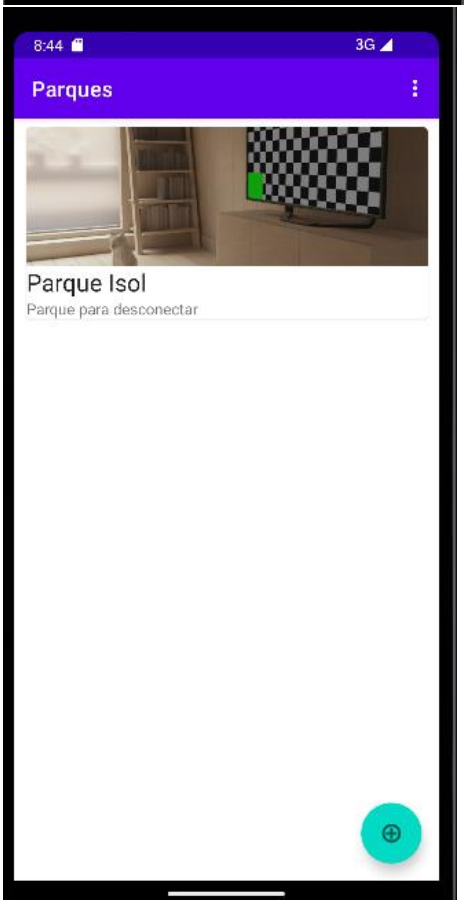
```
val img=itemView.findViewById(R.id.parkImageView) as ImageView
```

Finalmente, dentro del método bind, añadimos:

```
if (parque.img!="") img.setImageURI(Uri.parse(parque.img))
```


8:43
3G
Editar parque
Parque Isol
Parque para desconectar
658958456
www.isol.es
Opening Time
Closing Time
24h
24h
Activities
Sports
Children's Park


8:43
3G
Editar parque
Parque Isol
Parque para desconectar
658958456
www.isol.es
Opening Time
Closing Time
24h
24h
Activities
Sports
Children's Park


8:44
3G
Parques
Parque Isol
Parque para desconectar

Nota: para poder utilizar la cámara le hemos tenido que dar permiso anteriormente. Entrando en settings del device -> apps y en nuestra aplicación.