

UD 07

PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES 22/23
CFGs DAM

LIBRERÍAS MULTIMEDIA (PARTE 1) IMPLEMENTACIÓN DE ELEMENTOS MULTIMEDIA EN ANDROID STUDIO

Autor: Carlos Espinosa
c.espinosamoreno@edu.gva.es
Fecha: 2022/2023
Licencia Creative Commons
versión 4.0

Reconocimiento – NoComercial – CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Índice

Objetivos de la unidad.....	1
Imágenes en Android.....	1
Unidades de medida en Android.....	2
Captura de imágenes.....	3
Hacer la foto con nuestra aplicación.....	3
Opciones de configuración de la cámara.....	4
Implementar una vista previa de la imagen.....	6
Captura de imágenes.....	7
¿Cómo tomar una foto?.....	8
Documentación adicional actualizada.....	9
Reproducción de audio y vídeo.....	10
Conceptos básicos de MediaPlayer.....	10
Contenido multimedia aceptado.....	11
Recomendaciones sobre codificación de video.....	12
Controles multimedia.....	13
Apoyo a la reanudación de la reproducción.....	14
Implementación MediaBrowserService.....	14
BIBLIOGRAFÍA.....	15

1. Objetivos de la unidad

- Incorporar sonidos y materiales audiovisuales a las aplicaciones
- Modelar y aplicar contenidos multimedia
- Manejar las propiedades de las pantallas de dispositivos móviles y controlar los eventos de pantalla.
- Usar las librerías de gestión del sonido de Android.
- Integrar objetos como recurso de audio y vídeo
- Comprender e implementar el uso de sensores de cámara en la aplicación.

2. Imágenes en Android
















Para el uso de imágenes hay que tener unos conceptos básicos como codificación de imágenes. Las imágenes pueden ser mapas de bits o imágenes vectoriales. Un mapa de bits es una matriz de bits que especifican el color de cada píxel en esa matriz. El número de bits dedicados a un píxel individual determina el número de colores que se pueden asignar a ese píxel.

Ejemplo: si cada píxel se representa mediante 4 bits, a un píxel determinado se le puede asignar uno de los 16 colores diferentes ($2^4 = 16$).

Además, un píxel puede tener propiedades de brillo, de esta forma es como se conforma la imagen en la pantalla y la calidad de un mapa de bits depende de la resolución.

Por otro lado, tenemos imágenes vectoriales, basadas en fórmulas matemáticas, y que no se dividen en unidades mínimas de información, sino en manchas de color y líneas. Las imágenes vectoriales son independientes de la resolución y mantienen la nitidez y la definición aunque se amplíen: no hay pérdida de calidad y pueden adaptarse a cualquier forma.

Cada color de la tabla se representa mediante un número de 24 bits: 8 bits para rojo, 8 bits para verde y 8 bits para azul. Los números se muestran en formato hexadecimal (base 16): A = 10, B = 11, C = 12, D = 13, E = 14 y F = 15.

3	3	3	3	3	3	3	3	0	000000	
0	1	4	1	4	1	4	0	1	FF0000	
0	4	1	4	1	4	1	0	2	00FF00	
0	5	5	5	5	5	5	0	3	0000FF	
0	5	5	5	5	5	5	0	4	FFFFFF	
0	1	4	1	4	1	4	0	5	FFFF00	
0	4	1	4	1	4	1	0	6	FF00FF	
2	2	2	2	2	2	2	2	7	00FFFF	
								8	FF0080	
								9	FF8040	
								A	804000	
								B	008080	
								C	800000	
								D	800080	
								E	8080FF	



Puedes ver más sobre los diferentes formatos de imágenes aquí:

<https://learn.microsoft.com/es-es/dotnet/desktop/winforms/advanced/types-of-bitmaps?view=netframeworkdesktop-4.8>

En cuanto al diseño de app, el uso de imágenes va ligado también con la importancia de elegir una buena paleta de color puede cambiar nuestro diseño de la app significativamente. Es necesario buscar colores que le den aspecto actual y transmita las sensaciones para las que nuestra app está creada.

Ejemplo

900	#311B92	900	#1A237E	900	#0D47A1
A100	#B388FF	A100	#8C9EFF	A100	#82B1FF
A200	#7C4DFF	A200	#536DFF	A200	#448AFF
A400	#651FFF	A400	#3D5AFE	A400	#2979FF
A700	#6200EA	A700	#304FFE	A700	#2962FF

https://desarrollador-android.com/material-design/disenio-material-design/estilo/color/#Paleta_de_color

3. Unidades de medida en Android

En Android, al programar debemos acostumbrarnos a utilizar unidades de medida que se adapten a las pantallas como:

- points (PT): son la unidad de medida más pequeña utilizada en tipografía. En una pulgada hay 72pt y en centímetros 28 pt.
- píxels (px): equivale a un pixel real. No recomendado → varía mucho según el dispositivo.
- density-independent pixels (dp): basada en la densidad física. Considerada como la solución más adaptada para las diferentes pantallas.
- scale-independent pixels (sp): es similar a los dp, pero aplicados a textos. Se escala según el tamaño de fuente. Es el recomendable cuando especificamos el tamaño de la fuente en nuestra aplicación.

Una vez sabemos cómo se utilizan las unidades de medida vamos a ver cómo podemos aplicarlos a la hora de trabajar con imágenes en nuestras pantallas, cuyas dimensiones expresaremos siempre en pulgadas (1inch=2.54cm), que expresan el tamaño de la diagonal visible.

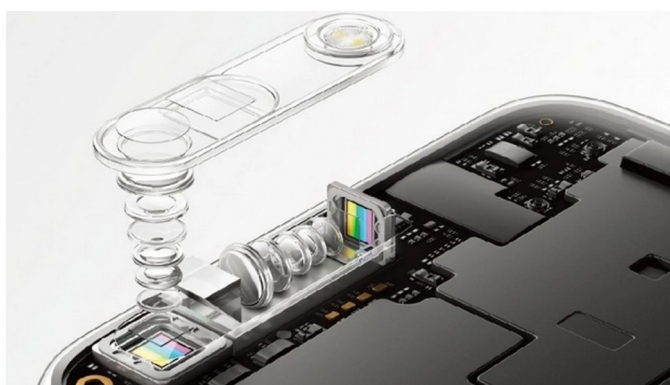
En cuanto a la resolución, esta indica el número de líneas verticales y horizontales que puede mostrar la pantalla y se expresa en ppp (píxeles por pulgada).

Importante: Un número alto de resolución implica imágenes de gran calidad pero también un incremento en el uso de los recursos.

4. Captura de imágenes

Como sabes, todos los smartphones disponen de una cámara de fotos integrada, de mejor o peor calidad que podremos utilizar con nuestra aplicación.

Podemos hacer uso de esta cámara de dos formas, mediante un **Intent** o programando nosotros una aplicación para el control de la cámara. El segundo método es sensiblemente más complejo y nos permitirá hacer un ajuste de los balances, del uso del flash, etc.



4.1 Hacer la foto con nuestra aplicación

Existen diferentes métodos para el uso e implementación de la cámara en nuestra app. El apartado Camera y Camera han entrado en desuso y en la actualidad se utiliza CameraX.

CameraX es una biblioteca de Jetpack creada para que el desarrollo de una apps de cámara sea más fácil. Para las apps nuevas, es recomendable que comiences con CameraX puesto que proporciona una API coherente y fácil de usar que funcione en la gran mayoría de los dispositivos

CameraX es compatible con dispositivos que **ejecutan Android 5.0 (nivel de API 21)** y versiones posteriores, lo que representa más del **98%** de los dispositivos Android existentes.

En ella destacan los casos de uso, que te permiten concentrarte en la tarea que debes completar en lugar de administrar variaciones específicas del dispositivo. Se admiten los casos de uso de la cámara más comunes:

- **Vista previa:** Permite obtener una imagen en la pantalla.
- **Análisis de imágenes:** Permite acceder a un búfer sin inconvenientes a fin de utilizarlo en tus algoritmos, por ejemplo, para pasar contenido a ML Kit.
- **Captura de imágenes:** Permite guardar imágenes.
- **Captura de video:** Permite guardar videos y audio.

CameraX tiene una API de Extensions opcional que te permite acceder a las mismas funciones y capacidades que la app de cámara nativa de un dispositivo con pocas líneas de código.

4.2 Opciones de configuración de la cámara

En este apartado vamos a ver como se configura CameraX para controlar diferentes aspectos de sus operaciones. Existen diferentes métodos, es recomendable consultar la documentación específica para la implementación más eficiente en nuestra app.

Por ejemplo, con el caso de uso de captura de imágenes simple, se puede establecer una relación de aspecto y un modo de flash. El siguiente código muestra un ejemplo:

```
val imageCapture = ImageCapture.Builder()
    .setFlashMode(...)
    .setTargetAspectRatio(...)
    .build()
```

Para simplificar, CameraX tiene configuraciones predeterminadas, como ejecutores internos y controladores que son adecuados para la mayoría de los casos de uso.

Sin embargo, si tu aplicación tiene requisitos especiales o prefiere personalizar esas configuraciones, CameraXConfig es la interfaz para ese propósito.

Con **CameraXConfig**, una aplicación puede hacer lo siguiente:

- Optimizar la latencia de inicio con **setAvailableCameraLimiter()**
- Proporcionar el ejecutor de la aplicación a CameraX con **setCameraExecutor()**
- Reemplazar el controlador de programador predeterminado por **setSchedulerHandler()**
- Cambiar el nivel de registro con **setMinimumLoggingLevel()**

Vamos a ver un pequeño ejemplo:

- Crea un objeto CameraXConfig con tus configuraciones personalizadas.
- Implementa la interfaz CameraXConfig.Provider en tu Application y muestra tu objeto CameraXConfig en getCameraXConfig().
- Agrega la clase Application al archivo AndroidManifest.xml, como se describe [aquí](#).

```
class CameraApplication : Application(), CameraXConfig.Provider {
    override fun getCameraXConfig(): CameraXConfig {
        return CameraXConfig.Builder.fromConfig(Camera2Config.defaultConfig())
            .setMinimumLoggingLevel(Log.ERROR).build()
    }
}
```

Mantén una copia local del objeto CameraXConfig si tu aplicación necesita saber la configuración de CameraX después de establecerlo.

Selección automática del hardware.

Automáticamente, CameraX proporciona funciones específicas para el dispositivo en el que se ejecuta tu app. Por ejemplo, CameraX determinará automáticamente la mejor resolución si no especificas una o si la que especificas no es compatible. Todo esto lo administra la biblioteca, lo que elimina la necesidad de escribir un código específico del dispositivo.

El objetivo de CameraX es inicializar con éxito una sesión de la cámara. Por lo tanto, CameraX ajusta la resolución y las relaciones de aspecto según la capacidad del dispositivo por los siguientes motivos:

- El dispositivo no admite el tipo de resolución solicitado.
- El dispositivo tiene problemas de compatibilidad, como los dispositivos heredados que requieren determinadas resoluciones para funcionar correctamente.
- En algunos dispositivos, ciertos formatos solo están disponibles en determinadas relaciones de aspecto.
- El dispositivo tiene preferencia por un "mod16 más cercano" para JPEG o codificación de video.

Se puede consultar la resolución con `SCALER_STREAM_CONFIGURATION_MAP` para obtener más información de la cámara. Aunque CameraX crea y administra la sesión, siempre debes verificar en el código los tamaños de imagen que se muestran en los resultados del caso de uso y ajustarlos en consecuencia.

En caso de querer usar un dispositivo diferente, hay algunas opciones:

- Solicita la cámara frontal predeterminada con `CameraSelector.DEFAULT_FRONT_CAMERA`.
- Solicita la cámara posterior predeterminada con `CameraSelector.DEFAULT_BACK_CAMERA`.
- Filtra la lista de dispositivos disponibles por su `CameraCharacteristics` con `CameraSelector.Builder.addCameraFilter()`.

```
fun selectExternalOrBestCamera(provider: ProcessCameraProvider):CameraSelector? {
    val cam2Infos = provider.availableCameraInfos.map {
        Camera2CameraInfo.from(it)
    }.sortedByDescending {
        // HARDWARE_LEVEL is Int type, with the order of:
        // LEGACY < LIMITED < FULL < LEVEL_3 < EXTERNAL
        it.getCameraCharacteristic(CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL)
    }

    return when {
        cam2Infos.isNotEmpty() -> {
            CameraSelector.Builder()
                .addCameraFilter {
                    it.filter { camInfo ->
                        // cam2Infos[0] is either EXTERNAL or best built-in camera
                        val thisCamId = Camera2CameraInfo.from(camInfo).cameraId
                        thisCamId == cam2Infos[0].cameraId
                    }
                }
            }.build()
        }
    }
```

```
    }  
    else -> null  
  }  
}  
  
// create a CameraSelector for the USB camera (or highest level internal camera)  
val selector = selectExternalOrBestCamera(processCameraProvider)  
processCameraProvider.bindToLifecycle(this, selector, preview, analysis)
```

Dependiendo de la aplicación puedes también establecer resoluciones específicas cuando compilas casos de uso con el método `setTargetResolution(Size resolution)`, como se muestra en la siguiente muestra de código:

```
val imageAnalysis = ImageAnalysis.Builder()  
    .setTargetResolution(Size(1280, 720))  
    .build()
```

4.3 Implementar una vista previa de la imagen

Una utilidad del uso de la cámara es agregar una vista previa a tu app, usando `PreviewView`, que es una `View` que se puede recortar, escalar y rotar para mostrarse correctamente.

La vista previa de la imagen se transmite a una superficie dentro de `PreviewView` cuando se activa la cámara.

La implementación de una vista previa de `CameraX` con `PreviewView` implica los siguientes pasos, que se explican en secciones posteriores:

- De manera opcional, configura un `CameraXConfig.Provider`.
- Agrega un elemento `PreviewView` a tu diseño.
- Solicita un elemento `ProcessCameraProvider`.
- Durante la creación de `View`, comprueba la presencia de `ProcessCameraProvider`.
- Selecciona una cámara y vincula el ciclo de vida y los casos de uso.

El uso de `PreviewView` tiene algunas limitaciones. Cuando usas `PreviewView`, no puedes hacer nada de lo siguiente:

- Crear un elemento `SurfaceTexture` para configurar en `TextureView` y `Preview.SurfaceProvider`
- Recuperar el elemento `SurfaceTexture` de `TextureView` y configurarlo en `Preview.SurfaceProvider`
- Obtener el elemento `Surface` de `SurfaceView` y configurarlo en `Preview.SurfaceProvider`
- Si se produce alguno de estos problemas, `Preview` detendrá la transmisión de marcos a `PreviewView`.

¿Cómo agregar un elemento PreviewView a tu diseño?

```
<FrameLayout
    android:id="@+id/container">
    <androidx.camera.view.PreviewView
        android:id="@+id/previewView" />
</FrameLayout>
```

En el siguiente código, se muestra cómo solicitar un elemento CameraProvider

```
import androidx.camera.lifecycle.ProcessCameraProvider
import com.google.common.util.concurrent.ListenableFuture

class MainActivity : AppCompatActivity() {
    private lateinit var cameraProviderFuture : ListenableFuture<ProcessCameraProvider>
    override fun onCreate(savedInstanceState: Bundle?) {
        cameraProviderFuture = ProcessCameraProvider.getInstance(this)
    }
}
```

Una vez que hayas creado y confirmado el elemento CameraProvider, haz lo siguiente:

- Crea un elemento Preview.
- Especifica la opción de cámara LensFacing deseada.
- Vincula la cámara seleccionada y los casos de uso al ciclo de vida.
- Conecta el elemento Preview a PreviewView.

En el siguiente código, se muestra un ejemplo:

```
fun bindPreview(cameraProvider : ProcessCameraProvider) {
    var preview : Preview = Preview.Builder()
        .build()

    var cameraSelector : CameraSelector = CameraSelector.Builder()
        .requireLensFacing(CameraSelector.LENS_FACING_BACK)
        .build()

    preview.setSurfaceProvider(previewView.getSurfaceProvider())

    var camera = cameraProvider.bindToLifecycle(this as LifecycleOwner, cameraSelector, preview)
}
```

5. Captura de imágenes

El caso de uso de captura de imágenes está diseñado para capturar fotos de alta calidad y resolución, y proporciona balance de blancos automático, exposición automática y enfoque automático (3A), además de los controles de cámaras simples y manuales. El emisor tiene la responsabilidad de decidir cómo usar la imagen capturada. Estas son algunas de las opciones:

- **takePicture(Executor, OnImageCapturedCallback):** Este método proporciona un búfer en la memoria de la imagen capturada.
- **takePicture(OutputFileOptions, Executor, OnImageSavedCallback):** Este método guarda la imagen capturada en la ubicación de archivos proporcionada.

Hay dos tipos de ejecutores personalizables en los que se ejecuta ImageCapture: el ejecutor de devolución de llamada y el ejecutor de IO.

El ejecutor de devolución de llamada es el parámetro de los métodos takePicture. Se usa para ejecutar el elemento OnImageCapturedCallback() proporcionado por el usuario.

Si el emisor decide guardar la imagen en una ubicación de archivo, puedes especificar un ejecutor para que realice el proceso de IO. Para configurar el ejecutor de IO, llama a ImageCapture.Builder.setIoExecutor(Executor). Si el ejecutor está ausente, de forma predeterminada, CameraX usará un ejecutor de IO interno para la tarea.

¿Cómo configurar la captura de imágenes?

La captura de imágenes proporciona controles básicos para tomar fotos, como flash, enfoque automático continuo, retraso sin obturador y mucho más.

Usa **ImageCapture.Builder.setCaptureMode()** para configurar el modo de captura cuando tomes una foto:

CAPTURE_MODE_MINIMIZE_LATENCY: Optimiza la captura de imágenes para la latencia.

CAPTURE_MODE_MAXIMIZE_QUALITY: Optimiza la captura de imágenes para mejorar su calidad.

El modo de captura predeterminado es CAPTURE_MODE_MINIMIZE_LATENCY. Para obtener más información, consulta la documentación de referencia de setCaptureMode().

5.1 ¿Cómo tomar una foto?

En la siguiente muestra de código, se ve cómo configurar la app para que tome una foto:

```
val imageCapture = ImageCapture.Builder()
    .setTargetRotation(view.display.rotation)
    .build()

cameraProvider.bindToLifecycle(lifecycleOwner, cameraSelector, imageCapture,
    imageAnalysis, preview)
```

Ten en cuenta que bindToLifecycle() muestra un objeto Camera. Consulta esta guía a fin de obtener más información para controlar la salida de la cámara, como el zoom y la exposición.

Una vez configurada la cámara, el siguiente código toma una foto en función de la acción del usuario.

```
fun onClick() {  
    val outputFileOptions = ImageCapture.OutputFileOptions.Builder(File(...)).build()  
    imageCapture.takePicture(outputFileOptions, cameraExecutor,  
        object : ImageCapture.OnImageSavedCallback {  
            override fun onError(error: ImageCaptureException)  
            {  
                // insert your code here.  
            }  
            override fun onImageSaved(outputFileResults: ImageCapture.OutputFileResults) {  
                // insert your code here.  
            }  
        })  
}
```

El método de captura de imágenes es compatible con el formato JPEG. Para ver un código de muestra en el que se indica cómo convertir un objeto Media.Image del formato YUV_420_888 a un objeto Bitmap RGB, consulta YuvToRgbConverter.kt.

5.2 Documentación adicional actualizada

- Uso de CameraX:
<https://developer.android.com/training/camerax>
- Arquitectura de la cámara:
<https://developer.android.com/training/camerax/architecture>
- Configuración:
<https://developer.android.com/training/camerax/configuration>
- Vista previa:
<https://developer.android.com/training/camerax/preview>
- Tomar fotos:
<https://developer.android.com/training/camerax/take-photo>
- Análisis de imágenes:
<https://developer.android.com/training/camerax/analyze>

6. Reproducción de audio y vídeo

La clase encargada de la reproducción de audio y video es “**MediaPlayer**”.

El marco de trabajo de contenido multimedia de Android se admite la reproducción de diversos tipos de contenido multimedia comunes para que puedas integrar audio, video e imágenes con facilidad en tus apps. Puedes reproducir audio o video desde archivos multimedia almacenados en los recursos de tu app (recursos sin procesar), desde archivos independientes del sistema de archivos o desde un flujo de datos que llega a través de una conexión de red, todo mediante diferentes API de MediaPlayer.

En este documento, se muestra cómo escribir una app de reproducción de contenido multimedia que interactúa con el usuario y el sistema para obtener un buen rendimiento y una experiencia del usuario agradable

Solo puedes reproducir los datos de audio en el dispositivo de salida estándar. Por el momento, este es el altavoz del dispositivo móvil o los auriculares Bluetooth.

6.1 Conceptos básicos de MediaPlayer

Las siguientes clases se usan para reproducir sonido y video en el marco de trabajo de Android:

- **MediaPlayer:** Esta clase es la API principal para reproducir sonido y video.
- **AudioManager:** Esta clase administra fuentes y salidas de audio en un dispositivo.

Uno de los componentes más importantes del marco de trabajo de medios es la clase MediaPlayer y será en el que nos centraremos. Un objeto de esta clase puede recuperar, decodificar y reproducir audio y video con una **configuración mínima**.

La principal ventaja es que es compatible con varias fuentes de medios diferentes, entre las que se incluyen las siguientes:

- **Recursos locales**
- URI internos, como uno que se pueda obtener de un agente de resolución de contenido
- URL externas (transmisión)

Para obtener una lista de los formatos de medios compatibles con Android, consulta la documentación *Formatos de medios compatibles* pero lo más comunes si que están soportados.

A continuación, se muestra un ejemplo de cómo reproducir un archivo de audio que está disponible como recurso local sin procesar (guardado en el directorio **res/raw/** de tu app):

```
var mediaPlayer: MediaPlayer? = MediaPlayer.create(context, R.raw.sound_file_1)
mediaPlayer?.start() // no need to call prepare(); create() does that for you
```

En este caso, un recurso "sin procesar" es un archivo que el sistema no intenta analizar de ninguna manera en particular. Sin embargo, el contenido de este recurso no debe ser audio sin formato. Debe ser un archivo multimedia con el formato y la codificación adecuados en uno de los **formatos admitidos**.

A continuación, se muestra cómo puedes reproducir desde un URI disponible de forma local en el sistema (que obtuviste mediante un agente de resolución de contenido, por ejemplo):

```
val myUri: Uri = .... // initialize Uri here
val mediaPlayer: MediaPlayer? = MediaPlayer().apply {
    setAudioStreamType(AudioManager.STREAM_MUSIC)
    setDataSource(applicationContext, myUri)
    prepare()
    start()
}
```

La reproducción desde una URL remota a través de la transmisión HTTP tiene el siguiente aspecto:

```
val url = "http://....." // your URL here
val mediaPlayer: MediaPlayer? = MediaPlayer().apply {
    setAudioStreamType(AudioManager.STREAM_MUSIC)
    setDataSource(url)
    prepare() // might take long! (for buffering, etc)
    start()
}
```

Si transmites una URL para transmitir un archivo multimedia en línea, el archivo debe poder descargarse progresivamente

7. Contenido multimedia aceptado

En este documento, se describe el códec de medios, el contenedor y la compatibilidad con el protocolo de red que proporciona la plataforma de Android.

Como desarrollador de aplicaciones, puedes usar cualquier códec disponible en dispositivos con Android, tanto los proporcionados por plataforma de Android como los de plataformas específicas del dispositivo. Sin embargo, se recomienda usar perfiles de codificación de medios que se puedan usar en cualquier dispositivo.

En las tablas que se muestran a continuación, se describe el soporte de formatos de medios integrado en la plataforma de Android. Los códecs cuya disponibilidad no está garantizada en todas las versiones de la plataforma de Android se indican entre paréntesis, por ejemplo: (Android 3.0 o posterior). Ten en cuenta que cualquier dispositivo móvil puede admitir otros formatos o tipos de archivo que no se enumeran en la tabla.

En la sección 5 de la Definición de compatibilidad de Android, se especifican los formatos de medios que un dispositivo debe admitir para ser compatible con Android 8.1.

7.1 Recomendaciones sobre codificación de video

Formatos y códecs de video:

En la siguiente tabla, se enumeran los perfiles y parámetros de codificación de video del marco de trabajo de medios de Android recomendados para la reproducción utilizando el códec de perfil básico de H.264.

Las mismas recomendaciones se aplican al códec de perfil principal, que solo está disponible en Android 6.0 y versiones posteriores.

	SD (baja calidad)	SD (alta calidad)	HD 720p
Resolución de video	176 x 144 px	480 x 360 px	1280 x 720 px
Velocidad de fotogramas del video	12 fps	30 fps	30 fps
Tasa de bits del video	56 Kbps	500 Kbps	2 Mbps
Códec de audio	AAC-LC	AAC-LC	AAC-LC
Canales de audio	Mono	Estéreo	Estéreo
Tasa de bits del audio	24 Kbps	128 Kbps	196 Kbps

En la siguiente tabla, se enumeran los perfiles y parámetros de codificación de video del marco de trabajo de medios de Android recomendados para la reproducción con el códec de medios VP8.

	Baja calidad	Calidad media	HD 720p	HD 1080p
Resolución de vídeo	320 x 180 px	640 x 360 px	1280 x 720 px	1920 x 1080 px
Velocidad de fotogramas del vídeo	30 fps	30 fps	30 fps	30 fps – 60 fps
Tasa de bits del vídeo	800 Kbps	2-4 Mbps	4 Mbps	+10 Mbps

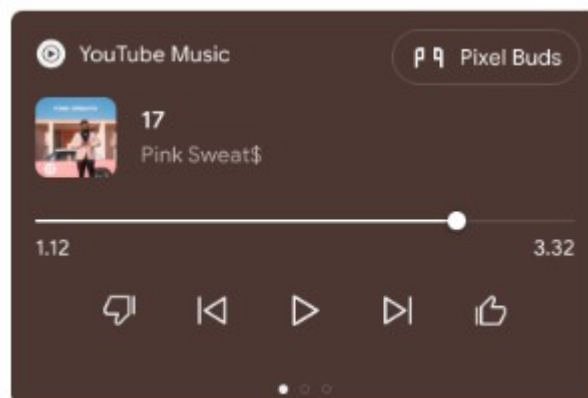
8. Controles multimedia

Android 11 actualiza cómo se muestran los controles de medios, haciendo uso de las API `MediaSession` y **MediaRouter2** para representar los controles y la información de salida de audio.

Los controles de medios en Android 11 se encuentran cerca de la Configuración rápida. Las sesiones de varias aplicaciones se organizan en un carrusel deslizable. El carrusel enumera las sesiones en este orden:

- Transmisiones que se reproducen localmente en el teléfono
- Transmisiones remotas, como las detectadas en dispositivos externos o sesiones de transmisión
- Sesiones reanudables anteriores, en el orden en que se reprodujeron por última vez

Los usuarios pueden reiniciar sesiones anteriores desde el carrusel sin tener que iniciar la aplicación. Cuando comienza la reproducción, el usuario interactúa con los controles multimedia de la forma habitual.



8.1 Apoyo a la reanudación de la reproducción

Para usar esta función, debe habilitar la reanudación de medios en la configuración de Opciones de desarrollador.

Para que su aplicación de jugador aparezca en el área de configuración de configuración rápida, debe crear una **MediaStyleNotification** con un **MediaSessiontoken** válido.

Para mostrar el icono de la marca del reproductor multimedia, utilice **NotificationBuilder.setSmallIcon()**.

Para admitir la reanudación de la reproducción, las aplicaciones deben implementar un **MediaBrowserService** y **MediaSession**.

La función de reanudación de la reproducción se puede desactivar mediante la aplicación Configuración, en las opciones de **Sonido > Medios** . El usuario también puede acceder a Configuración tocando el ícono de ajustes que aparece después de deslizar el carrusel expandido.

8.2 Implementación MediaBrowserService

Una vez que se inicia el dispositivo, el sistema busca las cinco aplicaciones multimedia usadas más recientemente y proporciona controles que se pueden usar para reiniciar la reproducción desde cada aplicación.

El sistema intenta contactarlo MediaBrowserService con una conexión desde SystemUI. Su aplicación debe permitir este tipo de conexiones; de lo contrario, no podrá admitir la reanudación de la reproducción.

Las conexiones de SystemUI se pueden identificar y verificar mediante el nombre `com.android.systemui` y la firma del paquete. El SystemUI está firmado con la firma de la plataforma. Puede encontrar un ejemplo de cómo verificar la firma de la plataforma en la aplicación UAMP .

Para admitir la reanudación de la reproducción, MediaBrowserService debe implementar estos comportamientos:

- `onGetRoot()` debe devolver una raíz no nula rápidamente. Otra lógica compleja debe manejarse en `onLoadChildren()`
- Cuando `onLoadChildren()` se llama al ID de medios raíz, el resultado debe contener un elemento secundario `FLAG_PLAYABLE` .
- MediaBrowserService debe devolver el elemento multimedia reproducido más recientemente cuando reciben una consulta `EXTRA_RECENT` . El valor devuelto debe ser un elemento multimedia real en lugar de una función genérica.
- MediaBrowserService debe proporcionar una `MediaDescription` adecuada con un título y un subtítulo que no estén vacíos . También debe establecer un URI de icono o un mapa de bits de icono .

BIBLIOGRAFÍA

- i. Android. Programación Multimedia y de dispositivos móviles. Garceta.
- ii. Programación Multimedia y Dispositivos Móviles. Editorial Síntesis.
- iii. Android App Development. For Dummies.
- iv. Beginning Android Programming with Android Studio. Wrox.
- v. Java Programming for Android Developers. For Dummies.
- vi. <https://academiaandroid.com/>
- vii. <https://developer.android.com/>
- viii. <https://www.redhat.com>
- ix. <https://desarrolloweb.com>
- x. <https://developer.android.com/training/camerax>
- xi. <https://developer.android.com/training/camerax/architecture>
- xii. <https://developer.android.com/training/camerax/configuration>
- xiii. <https://developer.android.com/training/camerax/preview>
- xiv. <https://developer.android.com/training/camerax/take-photo>
- xv. <https://developer.android.com/training/camerax/analyze>