**Week 11: CameraX Services in Android**

**Part 1: Create and configure project**

1. In Android Studio, create a new project, and select Empty Views Activity when prompted:

1. Next, name the app "CameraXApp". Set the minimum API level to 21 (**which is the minimum required for CameraX**).
2. Open the build.gradle file for the CameraXApp.app module, and add the CameraX dependencies:

def camerax\_version = "1.3.0"  
implementation "androidx.camera:camera-core:$**{**camerax\_version**}**"  
implementation "androidx.camera:camera-camera2:$**{**camerax\_version**}**"  
implementation "androidx.camera:camera-lifecycle:$**{**camerax\_version**}**"  
implementation "androidx.camera:camera-video:$**{**camerax\_version**}**"  
implementation "androidx.camera:camera-view:$**{**camerax\_version**}**"  
implementation "androidx.camera:camera-extensions:$**{**camerax\_version**}**"

1. CameraX needs some methods that are part of Java 8, so we need to set our compile options accordingly. At the end of the android block, right after buildTypes, add the following:

compileOptions **{** sourceCompatibility JavaVersion.*VERSION\_1\_8* targetCompatibility JavaVersion.*VERSION\_1\_8***}**

1. Please sync the project upon new configurations and dependencies added.

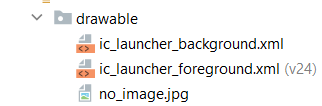
1. Add the following permission in the AndroidManifest file.

<uses-feature android:name="android.hardware.camera.any" />  
<uses-permission android:name="android.permission.CAMERA" />  
<uses-permission android:name="android.permission.RECORD\_AUDIO" />  
<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"  
 android:maxSdkVersion="28" />

1. Uses viewBinding.

buildFeatures **{** viewBinding = true  
**}**

1. Copy the **no\_image.jpg** file to the drawable folder.

****

1. Add the following text into the strings.xml resource file.

<string name="camera">Start Camera</string>  
<string name="photo">Take Photo</string>  
<string name="cancel">Cancel</string>  
<string name="back">Back</string>

**Part 2: MainActivity Layout**

1. Open the activity\_main layout file at res/layout/activity\_main.xml, and replace it with the following code:

<ImageView  
 android:id="@+id/imageView"  
 android:layout\_width="315dp"  
 android:layout\_height="345dp"  
 android:layout\_marginTop="32dp"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:srcCompat="@drawable/no\_image" />  
  
<Button  
 android:id="@+id/button\_camera"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginBottom="32dp"  
 android:text="@string/camera"  
 android:textAppearance="@style/TextAppearance.AppCompat.Display1"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent" />

1. Apply **ActivityMainBinding** class to the MainActivity class.

1. Declare the following variables in the MainActivity.

var binding: ActivityMainBinding? = null  
var startForResult: ActivityResultLauncher<Intent>? = null  
private val TAG = "CameraXMain"

1. Implement the **startForResult** for returning result from the second activity. Place the following code right after the setContentView statement.

startForResult = registerForActivityResult<Intent, ActivityResult>(  
 ActivityResultContracts.StartActivityForResult()  
) **{** result: ActivityResult **->**

if (result.*resultCode* == *RESULT\_OK*) {

//TODO CODE: Last part

}

**}**

1. Create a second activity call **CameraActivity**.

1. Implement the OnClickListener for the button\_camera. Set the intent to the second activity. For example:

val intent = Intent(this, CameraActivity::class.*java*)

startForResult!!.launch(intent)

**Part 3: Camera Activity and Layout**

1. Open the activity\_camera layout file at res/layout/activity\_camera.xml, and replace it with the following code:

<androidx.camera.view.PreviewView  
 android:id="@+id/viewFinder"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintHorizontal\_bias="0.0"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintVertical\_bias="1.0"></androidx.camera.view.PreviewView>  
  
<Button  
 android:id="@+id/button\_takephoto"  
 android:layout\_width="110dp"  
 android:layout\_height="110dp"  
 android:layout\_marginStart="16dp"  
 android:layout\_marginBottom="16dp"  
 android:elevation="2dp"  
 android:text="@string/photo"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent" />  
  
<Button  
 android:id="@+id/button\_back"  
 android:layout\_width="110dp"  
 android:layout\_height="110dp"  
 android:layout\_marginEnd="16dp"  
 android:layout\_marginBottom="16dp"  
 android:text="@string/back"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintTop\_toTopOf="@+id/button\_takephoto" />

1. Declare the following variables and constant values in the CameraActivity class:

**typealias LumaListener = (luma: Double) -> Unit**  
class CameraActivity : AppCompatActivity() {  
  
 **private lateinit var binding: ActivityCameraBinding  
 private var imageCapture: ImageCapture? = null  
 private lateinit var cameraExecutor: ExecutorService  
 private var photoURL = ""**

}

1. In the onCreate method, add in the following code:

override fun onCreate(savedInstanceState: Bundle?) **{**  
 super.onCreate(savedInstanceState)  
  
 binding = ActivityCameraBinding.inflate(*layoutInflater*)  
 val view: View = binding.*root* setContentView(view)  
  
 *// Request camera permissions* if (allPermissionsGranted()) {  
 startCamera()  
 } else {  
 requestPermissions()  
 }  
  
 *// Set up the listeners for take photo and video capture buttons* binding.buttonTakephoto.setOnClickListener {takePhoto()  
 }

cameraExecutor = Executors.newSingleThreadExecutor()

**}**

1. Add the following stub methods in the CameraActivity class:

private fun takePhoto() {}

private fun startCamera() {}

private fun requestPermissions() {}

private fun allPermissionsGranted() = REQUIRED\_PERMISSIONS.*all* {ContextCompat.checkSelfPermission(*baseContext*, **it**) == PackageManager.*PERMISSION\_GRANTED*}override fun onDestroy() {  
 super.onDestroy()  
 cameraExecutor.shutdown()  
}

companion object {  
 private const val TAG = "CameraXApp"  
 private const val FILENAME\_FORMAT = "yyyy-MM-dd-HH-mm-ss-SSS"  
  
 private val REQUIRED\_PERMISSIONS =  
 *mutableListOf* (  
 android.Manifest.permission.*CAMERA*,  
 android.Manifest.permission.*RECORD\_AUDIO* ).*apply* **{** if (Build.VERSION.*SDK\_INT* <= Build.VERSION\_CODES.*P*) {  
 add(android.Manifest.permission.*WRITE\_EXTERNAL\_STORAGE*)  
 }  
 **}**.*toTypedArray*()  
}

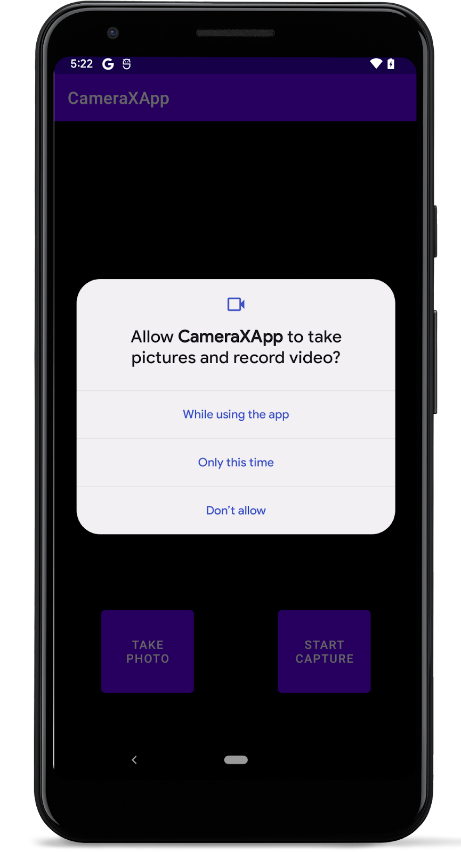
1. Add in the following code in the **CameraActivity** class under variable declaration section.

private val activityResultLauncher =  
 registerForActivityResult(  
 ActivityResultContracts.RequestMultiplePermissions())  
 **{** permissions **->** *// Handle Permission granted/rejected* var permissionGranted = true  
 permissions.entries.*forEach* **{** if (**it**.key in REQUIRED\_PERMISSIONS && **it**.value == false)  
 permissionGranted = false  
 **}** if (!permissionGranted) {  
 Toast.makeText(*baseContext*,  
 "Permission request denied",  
 Toast.*LENGTH\_SHORT*).show()  
 } else {  
 startCamera()  
 }  
 **}**

1. Add the following functionalities in the **requestPermission** method.

private fun requestPermissions() {  
 activityResultLauncher.launch(REQUIRED\_PERMISSIONS)  
}

1. Run the app. It should now ask permission to use the camera.



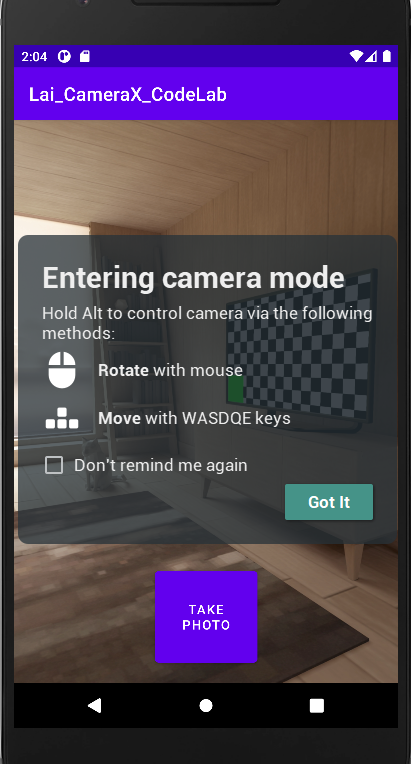
**Part 3: Implement Preview Use Case**

In a camera application, the viewfinder is used to let the user preview the photo they will be taking. We will implement a viewfinder using the CameraX Preview class.

1. Write the **startCamera** method as follows:

private fun startCamera() {  
 val cameraProviderFuture = ProcessCameraProvider.getInstance(this)  
  
 cameraProviderFuture.addListener(**{** *// Used to bind the lifecycle of cameras to the lifecycle owner* val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()  
  
 *// Preview* val preview = Preview.Builder()  
 .build()  
 .*also* **{  
 it**.setSurfaceProvider(viewBinding.viewFinder.surfaceProvider)  
 **}** *// Select back camera as a default* val cameraSelector = CameraSelector.*DEFAULT\_BACK\_CAMERA* try {  
 *// Unbind use cases before rebinding* cameraProvider.unbindAll()  
  
 *// Bind use cases to camera* cameraProvider.bindToLifecycle(  
 this, cameraSelector, preview)  
  
 } catch(exc: Exception) {  
 Log.e(TAG, "Use case binding failed", exc)  
 }  
  
 **}**, ContextCompat.getMainExecutor(this))  
}

1. Run the app.

**Part 4: Implement Image Capture Use Case**

1. Write the following code in the **takePhoto** method. This will create the filename for the image using the timestamp value.

*// Get a stable reference of the modifiable image capture use case*val imageCapture = imageCapture ?: return  
  
*// Create time stamped name and MediaStore entry.*val name = SimpleDateFormat(FILENAME\_FORMAT, Locale.*US*)  
 .format(System.currentTimeMillis())

1. Next, create a MediaStore content value to hold the image. Use a timestamp so the display name in MediaStore will be unique.

val contentValues = ContentValues().*apply* **{** put(MediaStore.MediaColumns.*DISPLAY\_NAME*, name)  
 put(MediaStore.MediaColumns.*MIME\_TYPE*, "image/jpeg")  
 if(Build.VERSION.*SDK\_INT* > Build.VERSION\_CODES.*P*) {  
 put(MediaStore.Images.Media.*RELATIVE\_PATH*, "Pictures/CameraX-Image")  
 }  
**}**

1. Create an [OutputFileOptions](https://developer.android.com/reference/kotlin/androidx/camera/core/ImageCapture.OutputFileOptions?authuser=1" \t "_blank) object. This object is where we can specify things about how we want our output to be. We want the output saved in the MediaStore so other apps could display it, so add our MediaStore entry.

*// Create output options object which contains file + metadata*val outputOptions = ImageCapture.OutputFileOptions  
 .Builder(*contentResolver*,  
 MediaStore.Images.Media.*EXTERNAL\_CONTENT\_URI*,  
 contentValues)  
 .build()

1. Call takePicture() on the imageCapture object. Pass in outputOptions, the executor, and a callback for when the image is saved. You'll fill out the callback next.

imageCapture.takePicture(  
 outputOptions,  
 ContextCompat.getMainExecutor(this),  
 object : ImageCapture.OnImageSavedCallback {

override fun onError(exc: ImageCaptureException) {  
 Log.e(TAG, "Photo capture failed: ${exc.message}", exc)  
 }  
  
 override fun onImageSaved(output: ImageCapture.OutputFileResults){  
 val msg = "Photo capture succeeded: ${output.*savedUri*}"  
photoURL = output.*savedUri*.*toString*()  
 Toast.makeText(*baseContext*, msg, Toast.*LENGTH\_SHORT*).show()  
 Log.d(TAG, msg)  
 }  
 }  
)

1. Go to the startCamera() method and copy this code under the code for preview.

imageCapture = ImageCapture.Builder()  
 .build()

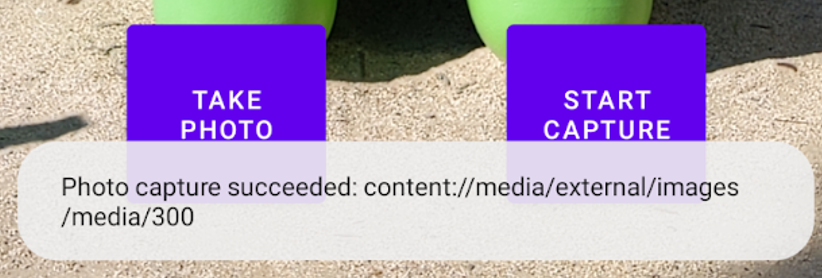
1. Update the call to bindToLifecycle() in the try block to include the new use case:

cameraProvider.bindToLifecycle(  
 this, cameraSelector, preview, **imageCapture**)

1. Implement the back button to return the captured photo URL.

binding.buttonBack.setOnClickListener **{** v **->** *//Sent the picture back to the MainActivity* val intent = Intent()  
 intent.putExtra("URL", photoURL)  
 setResult(*RESULT\_OK*, intent) *//Return ok code with data* finish()  
**}**

1. Rerun the app and press **Take Photo**. Analyse the message from the Logcat panel.



**Part 5 : Implement Image Analysis Use Case**

A great way to make our camera app more interesting is using the [ImageAnalysis](https://developer.android.com/reference/kotlin/androidx/camera/core/ImageAnalysis?authuser=1" \t "_blank) feature. It allows us to define a custom class that implements the ImageAnalysis.Analyzer interface, and which will be called with incoming camera frames.

1. Include the following class inside the CameraActivity class:

private class LuminosityAnalyzer(private val listener: LumaListener) : ImageAnalysis.Analyzer {  
  
 private fun ByteBuffer.toByteArray(): ByteArray {  
 rewind() *// Rewind the buffer to zero* val data = ByteArray(remaining())  
 get(data) *// Copy the buffer into a byte array* return data *// Return the byte array* }  
  
 override fun analyze(image: ImageProxy) {  
  
 val buffer = image.*planes*[0].*buffer* val data = buffer.*toByteArray*()  
 val pixels = data.*map* **{ it**.toInt() and 0xFF **}** val luma = pixels.*average*()  
  
 listener(luma)  
  
 image.close()  
 }  
}

The analyzer logs the average luminosity of the image. To create an analyzer, we override the analyze function in a class that implements the ImageAnalysis.Analyzer interface.

1. In the startCamera() method, add this code under the imageCapture code:

val imageAnalyzer = ImageAnalysis.Builder()  
 .build()  
 .*also* **{  
 it**.setAnalyzer(cameraExecutor, LuminosityAnalyzer **{** luma **->** Log.d(TAG, "Average luminosity: $luma")  
 **}**)  
**}**

1. Update the bindToLifecycle() call on the cameraProvider to include the imageAnalyzer

cameraProvider.bindToLifecycle(  
 this, cameraSelector, preview, imageCapture, **imageAnalyzer**)

1. Run the app now! It will produce a message similar to this in logcat approximately every second.

**D/CameraXApp: Average luminosity: ...**

**Part 6 : Returning the photo URL**

1. Back to the code in Part 2 section 4 in **MainActivity** class:

startForResult = registerForActivityResult<Intent, ActivityResult>(  
 ActivityResultContracts.StartActivityForResult()  
) **{** result: ActivityResult **->**

if (result.*resultCode* == *RESULT\_OK*) {

//**TODO CODE**: Last part

}

**}**

1. Replace the following code under the **TODO CODE** section.

val intent = result.*data*val file\_uri = intent!!.getStringExtra("URL")  
Log.d(TAG, file\_uri!!)  
try {  
 val imageUri = Uri.parse(file\_uri)  
  
val source = ImageDecoder.createSource(this.*contentResolver*, imageUri)  
 val bitmap:Bitmap = ImageDecoder.decodeBitmap(source)  
 binding!!.imageView.setImageBitmap(bitmap)  
  
 Log.d(TAG, imageUri.toString())  
} catch (e: Exception) {  
 Log.d(  
 TAG,"Uri Error:" + e.*localizedMessage* )  
}

1. Run the app now! Take a photo and press the back button.