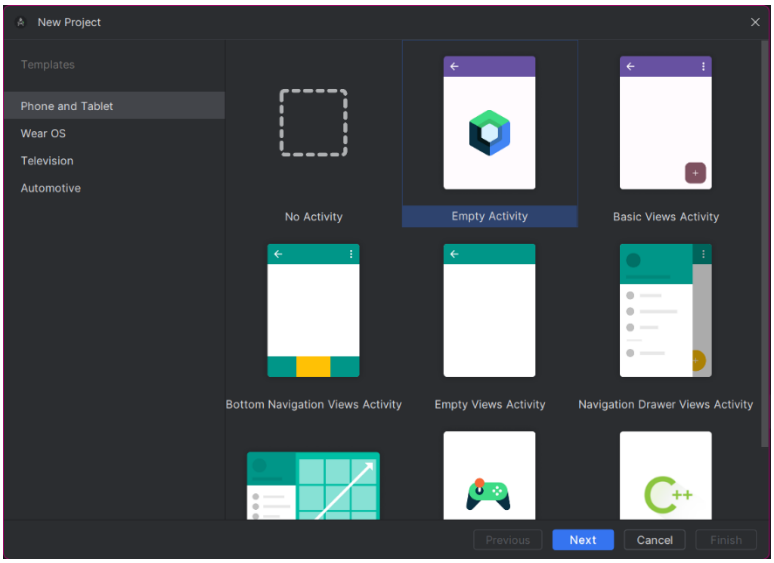
CameraX – Tutorial

Android / Kotlin

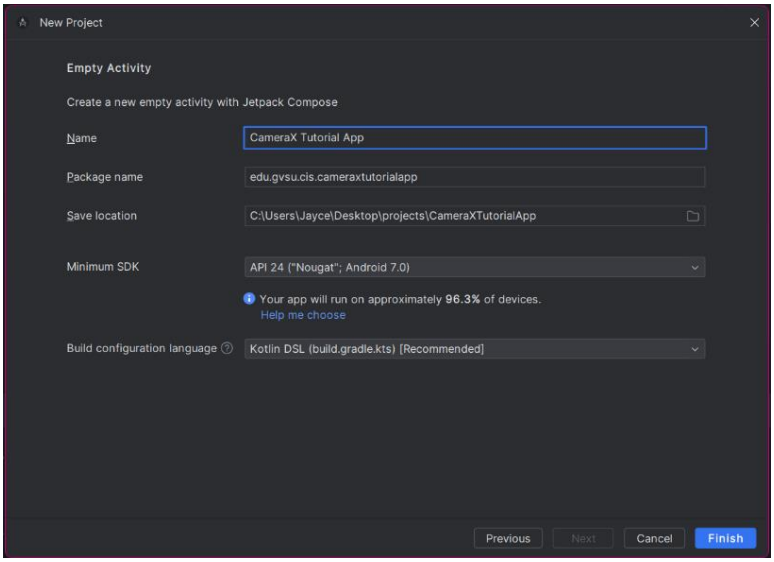
Jaycee Waycaster & Houston Moore
CIS 357 01

Overview

CameraX is a Jetpack Library for Android built to help make camera app development easier. It's a fairly easy to use library, and in this tutorial we'll be showing you how to use the library to build a basic camera app!
This tutorial was made with the latest Android Studio Beta version available at time of writing this (Android Studio Jellyfish | 2023.3.1 Beta 1) however most instructions will apply to recent previous versions. We will be using API 24 ("Nougat"; Android 7.0) for the minimum SDK.



Start by creating a new project with an empty activity



Then fill out the information for your project, we're using API 24 for our tutorial app. (For future reference, API 21 is the minimum required SDK for CameraX)

Now we need to set up out build.gradle file for Camera X

```
dependencies {
    implementation("androidx.camera:camera-core:1.2.2")
    implementation("androidx.camera:camera-camera2:1.2.2")
    implementation("androidx.camera:camera-lifecycle:1.2.2")
    implementation("androidx.camera:camera-video:1.2.2")

    implementation("androidx.camera:camera-view:1.2.2")
    implementation("androidx.camera:camera-extensions:1.2.2")
```

The following dependencies need to be added to your build.gradle file

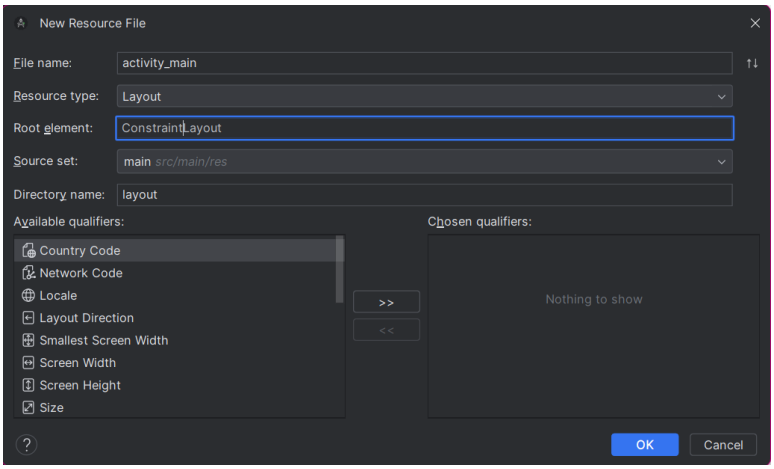
```
    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = "1.8"
    }
}
```

CameraX requires some methods from Java 8 as well, so you need to add these compile options in the same build.gradle file

```
android {
    namespace = "edu.gvsu.cis.came
    compileSdk = 34

    buildFeatures{
        viewBinding = true
    }
```

Finally, you need to enable view binding under the buildFeatures

Now, let's create the layout file for our main activity.



Make a new layout file, we recommend a constraint layout but you can use any layout you like.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    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">

    <androidx.camera.view.PreviewView
        android:id="@+id/viewFinder"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <Button
        android:id="@+id/image_capture_button"
        android:layout_width="110dp"
        android:layout_height="110dp"
        android:layout_marginBottom="50dp"
        android:layout_marginEnd="50dp"
        android:elevation="2dp"
        android:text="@string/take_photo"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintEnd_toStartOf="@id/vertical_centerline" />

    <Button
        android:id="@+id/video_capture_button"
        android:layout_width="110dp"
        android:layout_height="110dp"
        android:layout_marginBottom="50dp"
        android:layout_marginStart="50dp"
        android:elevation="2dp"
        android:text="@string/start_capture"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toEndOf="@id/vertical_centerline" />

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/vertical_centerline"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintGuide_percent=".50" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

The simplest way to get up and running with your layout file is to use the basic layout provided in the "Getting Started with CameraX" guide created by the Android Developers available in the resources at the end of this tutorial.

```xml
<resources>
    <string name="app_name">CameraXApp</string>
    <string name="take_photo">Take Photo</string>
    <string name="start_capture">Start Capture</string>
    <string name="stop_capture">Stop Capture</string>
</resources>
```

Then update the res/values/strings file with the following values

Now let's set up our main activity file

```
import android.Manifest
import android.content.ContentValues
import android.content.pm.PackageManager
import android.os.Build
import android.os.Bundle
import android.provider.MediaStore
import androidx.appcompat.app.AppCompatActivity
import androidx.camera.core.ImageCapture
import androidx.camera.video.Recorder
import androidx.camera.video.Recording
import androidx.camera.video.VideoCapture
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import com.android.example.cameraxapp.databinding.ActivityMainBinding
import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors
import android.widget.Toast
import androidx.activity.result.contract.ActivityResultContracts
import androidx.camera.lifecycle.ProcessCameraProvider
import androidx.camera.core.Preview
import androidx.camera.core.CameraSelector
import android.util.Log
import androidx.camera.core.ImageAnalysis
import androidx.camera.core.ImageCaptureException
import androidx.camera.core.ImageProxy
import androidx.camera.video.FallbackStrategy
import androidx.camera.video.MediaStoreOutputOptions
import androidx.camera.video.Quality
import androidx.camera.video.QualitySelector
import androidx.camera.video.VideoRecordEvent
import androidx.core.content.PermissionChecker
import java.nio.ByteBuffer
import java.text.SimpleDateFormat
import java.util.Locale
```

You'll need these imports

```
class MainActivity : AppCompatActivity() {
    private lateinit var viewBinding: ActivityMainBinding

    private var imageCapture: ImageCapture? = null

    private lateinit var cameraExecutor: ExecutorService

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

        // Request camera permissions
        if (allPermissionsGranted()) {
            startCamera()
        } else {
            requestPermissions()
        }

        // Set up the listeners for take photo and video capture buttons
        viewBinding.imageCaptureButton.setOnClickListener { takePhoto() }
        cameraExecutor = Executors.newSingleThreadExecutor()
    }

    private fun takePhoto() {}

    private fun startCamera() {}

    private fun requestPermissions() {}
```

Now we'll create theses variables and placeholder functions.

First, you'll need to create a lateinit var for viewbinding to the main layout file.

Then, a variable to hold the captured image initialized to null.

Next, a lateinit var for the camera executor

Then in the onCreate function, we'll be initializing viewbinding and creating an if-else block which either starts the camera or runs the function to request permissions

Next, create your click listener for the capture button and put the camera executor in its own thread.

And now, make the necessary placeholder functions like so.

```
    private fun allPermissionsGranted() = REQUIRED_PERMISSIONS.all { it: String
        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 (
                Manifest.permission.CAMERA,
                Manifest.permission.RECORD_AUDIO
            ).apply { this: MutableList<String>
                if (Build.VERSION.SDK_INT <= Build.VERSION_CODES.P) {
                    add(Manifest.permission.WRITE_EXTERNAL_STORAGE)
                }
            }.toTypedArray()
    }
}
```

Next, you'll make a function which checks that the necessary permissions are granted to your app.

Then, make sure you are shutting down the camera executor thread you created earlier!

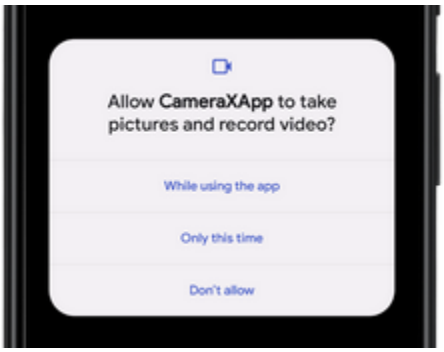
Finally, you'll need to make a companion object that describes the metadata attached to each photo, the filename structure, and a mutable list of the required permissions and android version.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-feature android:name="android.hardware.camera.any" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
        android:maxSdkVersion="28" />

    <application
        android:allowBackup="true"
```

Next, you'll need to add these permissions into your AndroidManifest.xml file

```
private val activityResultLauncher =
    registerForActivityResult(
        ActivityResultContracts.RequestMultiplePermissions())
    { permissions ->
        // Handle Permission granted/rejected
        var permissionGranted = true
        permissions.entries.forEach { it: Map.Entry<String, Boolean>
            if (it.key in REQUIRED_PERMISSIONS && it.value == false)
                permissionGranted = false
        }
        if (!permissionGranted) {
            Toast.makeText(baseContext,
                text: "Permission request denied",
                Toast.LENGTH_SHORT).show()
        } else {
            startCamera()
        }
    }
```

Then, make a launcher for requesting each required permission, that starts the camera if all checks are passed, creating this popup.



Now, let's start filling in those placeholder functions, starting with the "startCamera()" function.

```
private fun startCamera() {
    val cameraProviderFuture = ProcessCameraProvider.getInstance( context: this)

    cameraProviderFuture.addListener({
        // Used to bind the lifecycle of cameras to the lifecycle owner
        val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

        // Preview
        val preview = Preview.Builder() Preview.Builder
            .build() Preview
            .also { it: Preview
                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(
                 lifecycleOwner: this, cameraSelector, preview)

        } catch(exc: Exception) {
            Log.e(TAG,  msg: "Use case binding failed", exc)
        }

    }, ContextCompat.getMainExecutor( context: this))
}
```

Fill in the startCamera function like so,

```
private fun takePhoto() {
    // 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())
    val contentValues = ContentValues().apply { this: ContentValues
        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")
        }
    }

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

    // Set up image capture listener, which is triggered after photo has
    // been taken
    imageCapture.takePicture(
        outputOptions,
        ContextCompat.getMainExecutor( context: this),
        object : ImageCapture.OnImageSavedCallback {
            override fun onError(exc: ImageCaptureException) {
                Log.e(TAG,  msg: "Photo capture failed: ${exc.message}", exc)
            }
```

Next, lets populate the takePhoto() function.

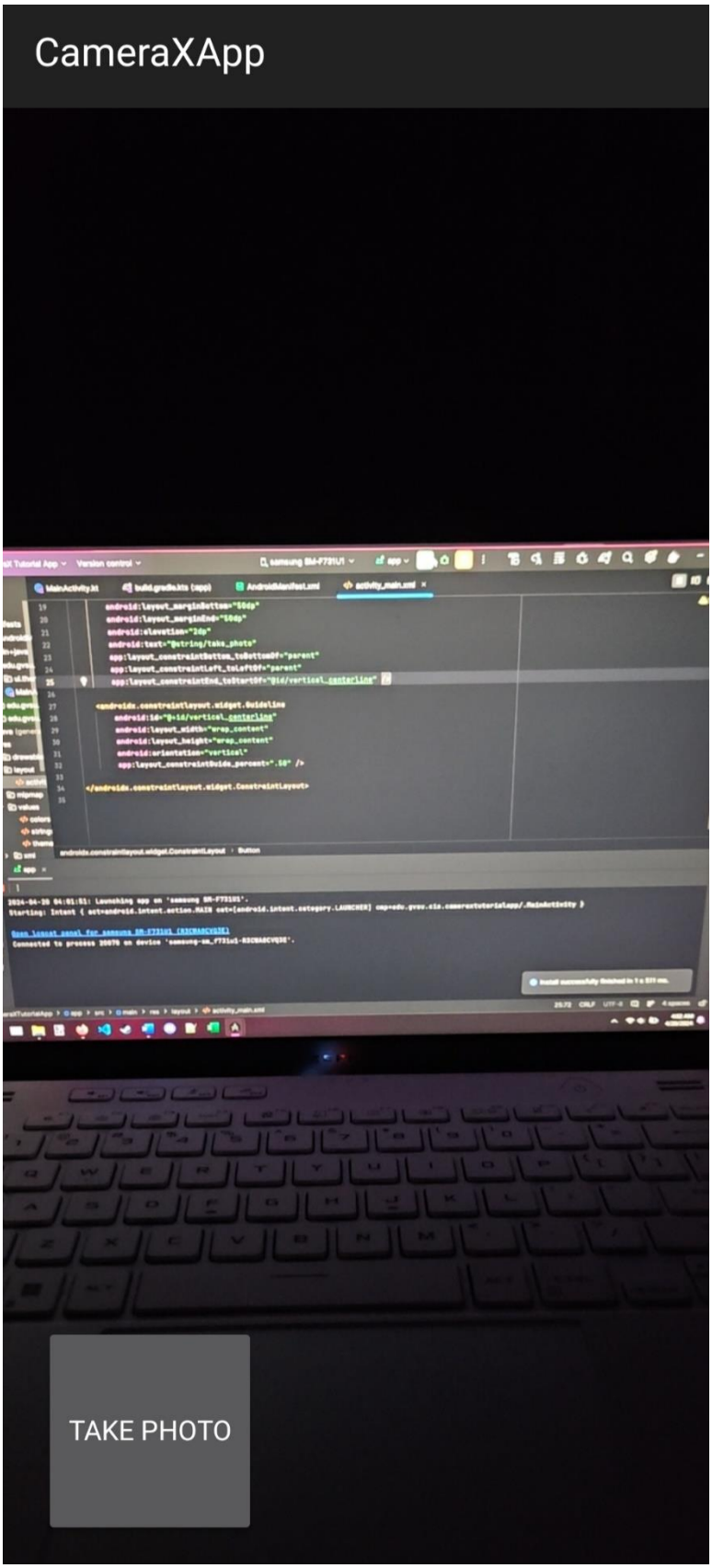
```
            override fun
                    onImageSaved(output: ImageCapture.OutputFileResults){
                val msg = "Photo capture succeeded: ${output.savedUri}"
                Toast.makeText(baseContext, msg, Toast.LENGTH_SHORT).show()
                Log.d(TAG, msg)
            }
        }
    )
}
```

```
    // Preview
    val preview = Preview.Builder() Preview.Builder
        .build() Preview
        .also { it: Preview
            it.setSurfaceProvider(viewBinding.viewFinder.surfaceProvider)
        }

    imageCapture = ImageCapture.Builder()
        .build()

    // Select back camera as a default
    val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA

    try {
        // Unbind use cases before rebinding
        cameraProvider.unbindAll()
```

Then, you'll need to add this highlighted line to the startCamera() function, to enable image capture.

```
private fun requestPermissions() {
    activityResultLauncher.launch(REQUIRED_PERMISSIONS)
}
```

Finally, we need to fill in the requestPermissions() function like so.



And Voila! Our CameraX app is working! This library is extremely useful, and far more powerful than our humble beginner app may seem. You can continue to implement video, sound recording, galleries, analyzation, and much more.

Resources Used

[1] CameraX Overview – Android Developers
https://developer.android.com/media/camera/camerax

[2] Getting Started with CameraX – Android Developers
https://developer.android.com/codelabs/camerax-getting-started