Android CameraX in Jetpack Compose

Camera API Update History

- Camera (pre 2014)
- Camera2 (since 2014)
  - “lower-level” interface
  - more control, but require more manual configuration
- CameraX (built on-top of Camera2)
  - “higher-level” interface, easier to use
  - designed to work with Jetpack Compose

Setup: build.gradle.kts

```
dependencies {
    val camerax_version = "1.6.0"
    implementation("androidx.camera:camera-core:${camerax_version}")
    implementation("androidx.camera:camera-camera2:${camerax_version}")
    implementation("androidx.camera:camera-lifecycle:${camerax_version}")
    implementation("androidx.camera:camera-view:${camerax_version}")
    implementation("androidx.camera:camera-extensions:${camerax_version}")
    implementation("androidx.camera:camera-compose:${camerax_version}")

    implementation("com.google.accompanist:accompanist-permissions:0.37.3")
}
```

Setup: AndroidManifest.xml

```
<manifest ....>
    <uses-feature
        android:name="android.hardware.camera"
        android:required="true" />
    <uses-permission android:name="android.permission.CAMERA" />
    <application ...>
    </application>
</manifest>
```

Example Use Cases

- Preview
- Image Capture (saved to a file)

Step 1: Build All The Use Cases

```
class AppViewModel(val app: Application): AndroidViewModel(app) {
    // Graphics Rendering surface (needed by the UI)
    private val _surfaceReq = MutableStateFlow<SurfaceRequest?>(null)
    val surfaceReq: StateFlow<SurfaceRequest?> = _surfaceReq.asStateFlow()

    // Build the Preview use case
    private val cameraPreview = Preview.Builder().build().apply {
        setSurfaceProvider {
            _surfaceReq.update { it }
        }
    }
    // Build the Image capture use case
    private val imageCapture = ImageCapture.Builder()
       .setIoExecutor(Executors.newSingleThreadExecutor())
       .build()
}
```

Step 2: Bind the Use Cases

```
class AppViewModel(val app: Application): AndroidViewModel(app) {

   suspend fun bindToCamera(lifecycleOwner: LifecycleOwner) {
       val processCameraProvider = ProcessCameraProvider.awaitInstance(app.applicationContext)
       processCameraProvider.bindToLifecycle(lifecycleOwner, DEFAULT_FRONT_CAMERA,
           cameraPreview, imageCapture) // These two use cases are shown in Step 1
       try {
           awaitCancellation()
       } finally {
           processCameraProvider.unbindAll()
       }
    }
}
```

Step 3: Initialize CameraX from the UI

```
@Composable
fun CameraScreen(....., viewModel = AppViewModel) {
    val lifecycleOwner: LifecycleOwner = LocalLifecycleOwner.current,
    val surfaceRequest by viewModel.surfaceReq.collectAsStateWithLifecycle()

    LaunchedEffect(lifecycleOwner) {
        viewModel.bindToCamera(lifecycleOwner)
    }

    if (surfaceRequest != null) {
        // This builtin composable shows the camera preview
        CameraXViewfinder(surfaceRequest!!)   // Provided by CameraX
    }
}
```

Photo Capture

```
class AppViewModel(val app: Application): AndroidViewModel(app) {
   fun takePhoto() {
       // Save the photo (JPG) to the application folder
       val fileDir = app.applicationContext.filesDir
       val newPhotoFile = File(fileDir, “myphoto.jpg”)
       val outputOption = ImageCapture.OutputFileOptions
           .Builder(newPhotoFile)
           .build()
       viewModelScope.launch(Dispatchers.IO) {
           // Use the device orientation as the photo orientation
           imageCapture.targetRotation = cameraPreview.targetRotation
           imageCapture.takePicture(outputOption)
       }
    }
}
```