



30-Day Android Challenge

From Basics to Advanced

(Expert-Level Deconstruction 📱)

Day 19

Runtime Permissions

Mastering Runtime Permissions – Beyond the Basics,
Towards Trust & Seamless UX! 🌓



Esmaeeil Moradi

@esmaeeil-moradi700

Day 19



Mastering Runtime Permissions – Beyond the Basics, Towards Trust & Seamless UX! 🛡️

Hey Android Architects!

Yesterday, we took a deep dive into the advanced intricacies of Dependency Injection with Hilt and Dagger, and I hear it landed perfectly! That's the caliber of knowledge we're pushing for. Today, we're tackling another critical, often misunderstood, aspect of Android development: Runtime Permissions.

Permissions are more than just a checkbox; they are the gatekeepers of user privacy and trust. Get it wrong, and you not only risk app crashes but, more importantly, user uninstalls, bad reviews, and a shattered reputation. This isn't just about getting access; it's about asking correctly and building lasting trust.

We've come a long way from the "all-or-nothing" days before Marshmallow. Modern Android demands a nuanced, user-centric approach. My Day 19 deep-dive will be your definitive guide to navigating this complex landscape, focusing on best practices, critical edge cases, and the latest APIs.



Esmaeel Moradi

@esmaeel-moradi700

Understanding the Android Permission Model: The Foundation of Trust



Before we write a single line of code, let's solidify our understanding of Android's permission philosophy:

- **Normal vs. Dangerous Permissions:** Normal Permissions (e.g., INTERNET, VIBRATE) are automatically granted at install time because they pose minimal risk to user privacy.
- **Dangerous Permissions** (e.g., CAMERA, ACCESS_FINE_LOCATION) access sensitive user data or system resources. These must be explicitly granted by the user at runtime. This is where our focus lies.
- **Runtime Permissions:** Introduced in Android 6.0 (API 23) Marshmallow. They give users granular control, allowing them to grant or deny permissions at any time, even after app installation.
- **Permission Groups:** Permissions are organized into groups (e.g., LOCATION group includes ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION). If a user grants one permission in a group, all other permissions in that same group are automatically granted. This is a crucial detail for simplifying requests.
- **Manifest Declaration is Still Key:** Remember, even for runtime permissions, you must declare them in your *AndroidManifest.xml* first. This tells the system your app intends to use these capabilities.



Esmaeel Moradi

@esmaeel-moradi700

The Classic Runtime Permission Request Flow (For Understanding)

While modern Android has better APIs, understanding the traditional callback-based flow is fundamental to appreciating the improvements.

1. **Check if Permission is Granted:** Use

ContextCompat.checkSelfPermission(context, permission) to see if you already have the permission.

2. **shouldShowRequestPermissionRationale(): The Crucial Conversation Starter:** *This method returns true if the user previously denied the permission, and you should show a rationale. It's not for the very first time you ask. It's your opportunity to explain why your app needs this permission in a user-friendly way. If you don't explain, users are more likely to deny again, or even worse, select "Never ask again."*

3. **Request the Permission:** Call

requestPermissions(activity, arrayOf(permission), REQUEST_CODE). This pops up the system permission dialog.

4. **Handle the Result:** Override

onRequestPermissionsResult(requestCode, permissions, grantResults) in your Activity or Fragment. Check the grantResults array to see if the permission was granted or denied.



Esmaeel Moradi

@esmaeel-moradi700

```
// Example: Traditional Single Permission Request (Activity)
import android.Manifest
import android.content.pm.PackageManager
import android.os.Bundle
import android.widget.Button
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.ContextCompat

class LegacyPermissionActivity : AppCompatActivity() {

    private val CAMERA_PERMISSION_CODE = 101

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_legacy_permission)

        findViewById<Button>(R.id.btn_request_camera).setOnClickListener {
            checkCameraPermission()
        }
    }

    private fun checkCameraPermission() {
        when {
            // 1. Permission already granted
            ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) ==
                PackageManager.PERMISSION_GRANTED -> {
                Toast.makeText(
                    this,
                    "Camera permission already granted!",
                    Toast.LENGTH_SHORT
                ).show()
                openCamera()
            }
            // 2. Should show rationale
            // (user previously denied, but not permanently)
            shouldShowRequestPermissionRationale(Manifest.permission.CAMERA) -> {
                showRationaleDialog(
                    "Camera access is needed to take photos for your profile.",
                    Manifest.permission.CAMERA
                )
            }
        }
    }
}
```



Esmaeel Moradi
@esmaeel-moradi700



```
// 3. Request permission for the first time or after permanent denial
else -> {
    requestPermissions(
        arrayOf(Manifest.permission.CAMERA),
        CAMERA_PERMISSION_CODE
    )
}
}
}

private fun showRationaleDialog(message: String, permission: String) {
    AlertDialog.Builder(this)
        .setTitle("Permission Needed")
        .setMessage(message)
        .setPositiveButton("OK") { _, _ ->
            requestPermissions(arrayOf(permission), CAMERA_PERMISSION_CODE)
        }
        .setNegativeButton("Cancel") { dialog, _ ->
            dialog.dismiss()
            Toast.makeText(
                this,
                "Camera permission denied.",
                Toast.LENGTH_SHORT
            ).show()
        }
        .create()
        .show()
}

override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    // Call super!
    if (requestCode == CAMERA_PERMISSION_CODE) {
        if (grantResults.isNotEmpty() &&
            grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            Toast.makeText(
                this,
                "Camera permission granted!",
                Toast.LENGTH_SHORT
            ).show()
            openCamera()
        } else {
            Toast.makeText(
                this,
                "Camera permission denied.",
                Toast.LENGTH_SHORT
            ).show()
        }
    }
}

private fun openCamera() { /* ... Camera logic ... */ }
```

Handling Multiple Permissions



You can request several permissions at once. The system dialog will show all requested permissions.

```
// Example: Requesting Multiple Permissions (Traditional)
import android.Manifest

class MultiplePermissionActivity : AppCompatActivity() {

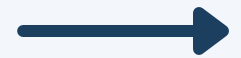
    private val MULTIPLE_PERMISSIONS_CODE = 102
    private val REQUIRED_PERMISSIONS = arrayOf(
        Manifest.permission.CAMERA,
        Manifest.permission.READ_EXTERNAL_STORAGE,
        Manifest.permission.ACCESS_FINE_LOCATION
    )

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_multiple_permission)

        findViewById<Button>(R.id.btn_request_multiple).setOnClickListener {
            checkAndRequestMultiplePermissions()
        }
    }
}
```



Esmaeel Moradi
@esmaeeil-moradi700



```
private fun checkAndRequestMultiplePermissions() {
    val permissionsToRequest = REQUIRED_PERMISSIONS.filter {
        ContextCompat.checkSelfPermission(this, it) !=
            PackageManager.PERMISSION_GRANTED
    }.toArray()

    if (permissionsToRequest.isNotEmpty()) {
        // You'd typically show a single rationale for all if needed
        // For simplicity, we just request them here.
        requestPermissions(permissionsToRequest, MULTIPLE_PERMISSIONS_CODE)
    } else {
        Toast.makeText(
            this,
            "All permissions already granted!",
            Toast.LENGTH_SHORT
        ).show()
        startAppFeatures()
    }
}

override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    if (requestCode == MULTIPLE_PERMISSIONS_CODE) {
        val deniedPermissions = permissions.filterIndexed { index, _ ->
            grantResults[index] != PackageManager.PERMISSION_GRANTED
        }

        if (deniedPermissions.isEmpty()) {
            Toast.makeText(
                this,
                "All requested permissions granted!",
                Toast.LENGTH_SHORT
            ).show()
            startAppFeatures()
        } else {
            Toast.makeText(
                this,
                "Some permissions were denied: $deniedPermissions",
                Toast.LENGTH_LONG
            ).show()
            // Handle partial or full denial.
            // Potentially show a rationale or guide to settings.
        }
    }
}

private fun startAppFeatures() { /* ... */ }
```



Esmaeel Moradi
@esmaeeil-moradi700

Best Practices and User Experience (UX): Beyond the Code



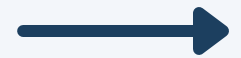
Mastering permissions isn't just about syntax; it's about empathetic design.

- **Contextual Requests: NEVER** ask for all permissions at app launch unless your app literally cannot function without them (which is rare). Request permissions exactly when the user is about to use a feature that requires them (e.g., ask for location permission only when the user taps "Show nearby restaurants"). This makes the request relevant and clear.
- **Clear and Empathetic Rationale:** This is where most developers fail. When *shouldShowRequestPermissionRationale()* returns *true*, you must show a custom UI (dialog, Snackbar, or dedicated screen) explaining why your app needs the permission. Use clear, concise, and non-technical language. Frame it from the user's perspective: "We need camera access to let you upload a profile picture." This is a **conversation**, not just a notification.
- **"Never Ask Again" / Denied Permanently:** This is the nightmare scenario. If *shouldShowRequestPermissionRationale()* returns *false* after the user has previously denied the permission (and often checked "Don't ask again"), it means the user has permanently denied it.
- **Guiding Users to Settings:** When a permission is permanently denied, the only way to get it back is for the user to manually enable it from app settings. Provide a clear button or prompt that takes them directly to your app's settings screen:



Esmaeel Moradi

@esmaeel-moradi700



```
// Guiding users to app settings
import android.content.Intent
import android.net.Uri
import android.provider.Settings
import androidx.appcompat.app.AppCompatActivity

fun openAppSettings(activity: AppCompatActivity) {
    val intent = Intent(Settings.ACTION_APPLICATION_DETAILS_SETTINGS).apply {
        data = Uri.fromParts("package", activity.packageName, null)
    }
    activity.startActivity(intent)
}
```

- **Graceful Degradation:** Always design your app to work, even if a user denies a permission. If camera access is denied, can they still use the app by uploading a photo from the gallery? Or can they manually type in location data? Inform the user about limited functionality without being accusatory.
- **Avoid Over-asking:** Only request permissions that are absolutely essential for your app's core functionality. Every unnecessary permission request erodes user trust.



Esmaeel Moradi
@esmaeel-moradi700

Permissions in Android 10+ (API 29+): Critical Changes



Android is constantly evolving its privacy model. Staying updated is non-negotiable.

- **Scoped Storage (Android 10+):** This was a monumental shift. Android significantly restricted broad *READ_EXTERNAL_STORAGE* and *WRITE_EXTERNAL_STORAGE* access.

1. **Recommendation:** Move away from these broad permissions for media and general file access.
2. **Alternatives:** Utilize *MediaStore* APIs for media files (images, videos, audio) and the *Storage Access Framework (SAF)* for general document and file access. Do not request broad storage permissions unless you are a file manager or a specialized app that genuinely needs it. Hacking around Scoped Storage often leads to app rejections.



Esmaeel Moradi
@esmaeel-moradi700



- **Background Location (Android 10+):** The highly sensitive `ACCESS_BACKGROUND_LOCATION` permission.
 1. **Crucial:** Only request this if your app absolutely needs location access when in the background (e.g., a real-time tracking app).
 2. **Flow:** You can only ask for `ACCESS_BACKGROUND_LOCATION` after `ACCESS_FINE_LOCATION` or `ACCESS_COARSE_LOCATION` has been granted. The system will prompt the user again.
 3. **UX is paramount here.** Be extremely transparent about why background location is needed.
- **Foreground Service Type Permissions (Android 10+):** If your app uses a foreground service that requires certain permissions (e.g., camera, location, microphone), you must declare `android:foregroundServiceType` in your `AndroidManifest.xml` alongside the permission. This clarifies your app's intent to the system and the user.



Esmaeel Moradi

@esmaeel-moradi700

Embracing the Modern Way: Activity Result API (Recommended)



The traditional `startActivityForResult()` and `onRequestPermissionsResult()` callback system was clunky, prone to boilerplate, and difficult to manage, especially in Fragments. The Activity Result API is the modern, cleaner, and recommended approach.

It eliminates the need for managing request codes and overriding callbacks, making your code significantly cleaner and more robust.

- **Introducing ActivityResultLauncher:** This is your new entry point. You register a launcher, and then simply call `launch()` on it.

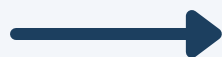


Esmaeel Moradi
@esmaeeil-moradi700

```
// Example: Single Permission with ActivityResultLauncher (Modern)
import android.Manifest
import android.content.pm.PackageManager
import android.os.Bundle
import android.widget.Button
import android.widget.Toast
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.ContextCompat

class ModernPermissionActivity : AppCompatActivity() {

    // 1. Register the launcher for a single permission
    private val requestCameraPermissionLauncher =
        registerForActivityResult(ActivityResultContracts.RequestPermission()) {
            isGranted: Boolean -> // Break here to avoid horizontal scroll
            if (isGranted) {
                Toast.makeText(
                    this,
                    "Camera permission granted!",
                    Toast.LENGTH_SHORT
                ).show()
                openCamera()
            } else {
                // Permission denied. Handle it!
                // Check if we should show rationale again,
                // or if it's permanently denied
                if (shouldShowRequestPermissionRationale(
                    Manifest.permission.CAMERA
                )) {
                    if (shouldShowRequestPermissionRationale(
                        Manifest.permission.CAMERA
                    )) {
                        showRationaleDialog(
                            "Camera access is needed to take photos for your profile.",
                            Manifest.permission.CAMERA
                        ) {
                            requestCameraPermissionLauncher.launch(
                                Manifest.permission.CAMERA
                            )
                        }
                    }
                } else {
                    // Permanently denied (User checked "Don't ask again")
                    Toast.makeText(
                        this,
                        "Camera permission denied permanently. Go to settings.",
                        Toast.LENGTH_LONG
                    ).show()
                    openAppSettings(this) // Guide to settings
                }
            }
        }
}
```



```

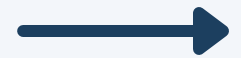
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_modern_permission)

    findViewById<Button>(R.id.btn_request_camera_modern).setOnClickListener {
        when {
            // 1. Permission already granted
            ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.CAMERA
            ) == PackageManager.PERMISSION_GRANTED -> {
                Toast.makeText(
                    this,
                    "Camera permission already granted!",
                    Toast.LENGTH_SHORT
                ).show()
                openCamera()
            }
            // 2. Should show rationale
            shouldShowRequestPermissionRationale(
                Manifest.permission.CAMERA
            ) -> {
                showRationaleDialog(
                    "Camera access is needed to take photos for your profile.",
                    Manifest.permission.CAMERA
                ) {
                    requestCameraPermissionLauncher.launch(
                        Manifest.permission.CAMERA
                    )
                }
            }
            // 3. Request permission (first time or after permanent denial
            // check above)
            else -> {
                requestCameraPermissionLauncher.launch(
                    Manifest.permission.CAMERA
                )
            }
        }
    }
}

```



Esmaeel Moradi
@esmaeel-moradi700



```
private fun showRationaleDialog(  
    message: String,  
    permission: String,  
    onPositive: () -> Unit  
) {  
    AlertDialog.Builder(this)  
        .setTitle("Permission Needed")  
        .setMessage(message)  
        .setPositiveButton("OK") { _, _ -> onPositive() }  
        .setNegativeButton("Cancel") { dialog, _ -> dialog.dismiss() }  
        .create()  
        .show()  
}  
  
// Reuse the openAppSettings from above  
// fun openAppSettings(activity: AppCompatActivity) { ... }  
  
private fun openCamera() { /* ... Camera logic ... */ }  
}
```



Esmaeel Moradi
@esmaeel-moradi700



```
// Example: Multiple Permissions with ActivityResultLauncher (Modern)
import android.Manifest
import android.content.pm.PackageManager
import android.os.Bundle
import android.widget.Button
import android.widget.Toast
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.ContextCompat

class ModernMultiplePermissionActivity : AppCompatActivity() {

    private val REQUIRED_PERMISSIONS = arrayOf(
        Manifest.permission.CAMERA,
        Manifest.permission.READ_EXTERNAL_STORAGE,
        Manifest.permission.ACCESS_FINE_LOCATION
    )

    // 1. Register the launcher for multiple permissions
    private val requestMultiplePermissionsLauncher =
        registerForActivityResult(
            ActivityResultContracts.RequestMultiplePermissions()
        ) { permissionsMap -> // Break here for screenshot friendly
            val granted = permissionsMap.all {
                it.value == true // Check if ALL are granted
            }

            if (granted) {
                Toast.makeText(
                    this,
                    "All requested permissions granted!",
                    Toast.LENGTH_SHORT
                ).show()
                startAppFeatures()
            } else {
                val deniedPermissions = permissionsMap
                    .filter { !it.value } // Filter to get denied ones
                    .keys.toSet() // Get the set of denied permission names

                Toast.makeText(
                    this,
                    "Some permissions denied: $deniedPermissions",
                    Toast.LENGTH_LONG
                ).show()

                // Handle partial or full denial.
                // You might iterate through deniedPermissions
                // and show specific rationales
                // or guide to settings if a critical one is permanently denied.
                handleDeniedPermissions(deniedPermissions)
            }
        }
}
```



Esmaeel Moradi
@esmaeeil-moradi700

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_modern_multiple_permission)

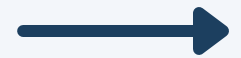
    findViewById<Button>(R.id.btn_request_multiple_modern)
        .setOnClickListener {
            // Filter permissions to request only those not yet granted
            val permissionsToRequest = REQUIRED_PERMISSIONS.filter {
                ContextCompat.checkSelfPermission(this, it) !=
                    PackageManager.PERMISSION_GRANTED
            }.toTypedArray()

            if (permissionsToRequest.isNotEmpty()) {
                requestMultiplePermissionsLauncher.launch(permissionsToRequest)
            } else {
                Toast.makeText(
                    this,
                    "All permissions already granted!",
                    Toast.LENGTH_SHORT
                ).show()
                startAppFeatures()
            }
        }
    }

private fun handleDeniedPermissions(denied: Set<String>) {
    denied.forEach { permission ->
        if (shouldShowRequestPermissionRationale(permission)) {
            // Show rationale for this specific permission
            // For simplicity, we just toast here. In real app, show a dialog.
            Toast.makeText(
                this,
                "Rationale needed for $permission",
                Toast.LENGTH_SHORT
            ).show()
        } else {
            // Permanently denied
            Toast.makeText(
                this,
                "$permission permanently denied. Guide to settings.",
                Toast.LENGTH_SHORT
            ).show()
            openAppSettings(this)
            // Guide to settings for critical permissions
        }
    }
}

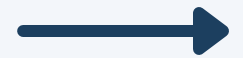
// Reuse openAppSettings(activity: AppCompatActivity) here too.
// private fun startAppFeatures() { ... }

```



Esmaeel Moradi
 @esmaeel-moradi700

Conclusion: Building Trust Through Responsible Permission Handling



Mastering runtime permissions is more than just technical proficiency; it's a testament to your commitment to user privacy and a robust user experience. By implementing permissions thoughtfully – requesting contextually, providing clear rationales, handling "never ask again" gracefully, and embracing modern APIs like *ActivityResultLauncher* – you're not just writing code; you're building trust.

Robust and user-friendly permission handling is a hallmark of professional Android development. Stay updated with Android's evolving privacy landscape and continuously refine your permission flows. Your users (and your app's ratings!) will thank you.



Esmaeel Moradi

@esmaeel-moradi700

Looking forward to your
insights and experiences!
Let's elevate Android
development together.



Esmaeel Moradi

@esmaeel-moradi700