플러터로 시작하는 네이티브

Permission 1편 - Android & iOS

Permission in Flutter

Flutter에서의 플랫폼 권한 처리 Flutter 애플리케이션에서 플랫폼 권한을 관리하는 것은 앱이 사용자의 디바이스 기능(예: 카메라, GPS, 연락처 등)에 접근할 때 중요합니다. 권한 관리는 사용자의 프라이버시를 보호하고, 앱의 사용성을 높이는 데 필수적인 요소입니다.

Flutter 권한 처리의 중요성

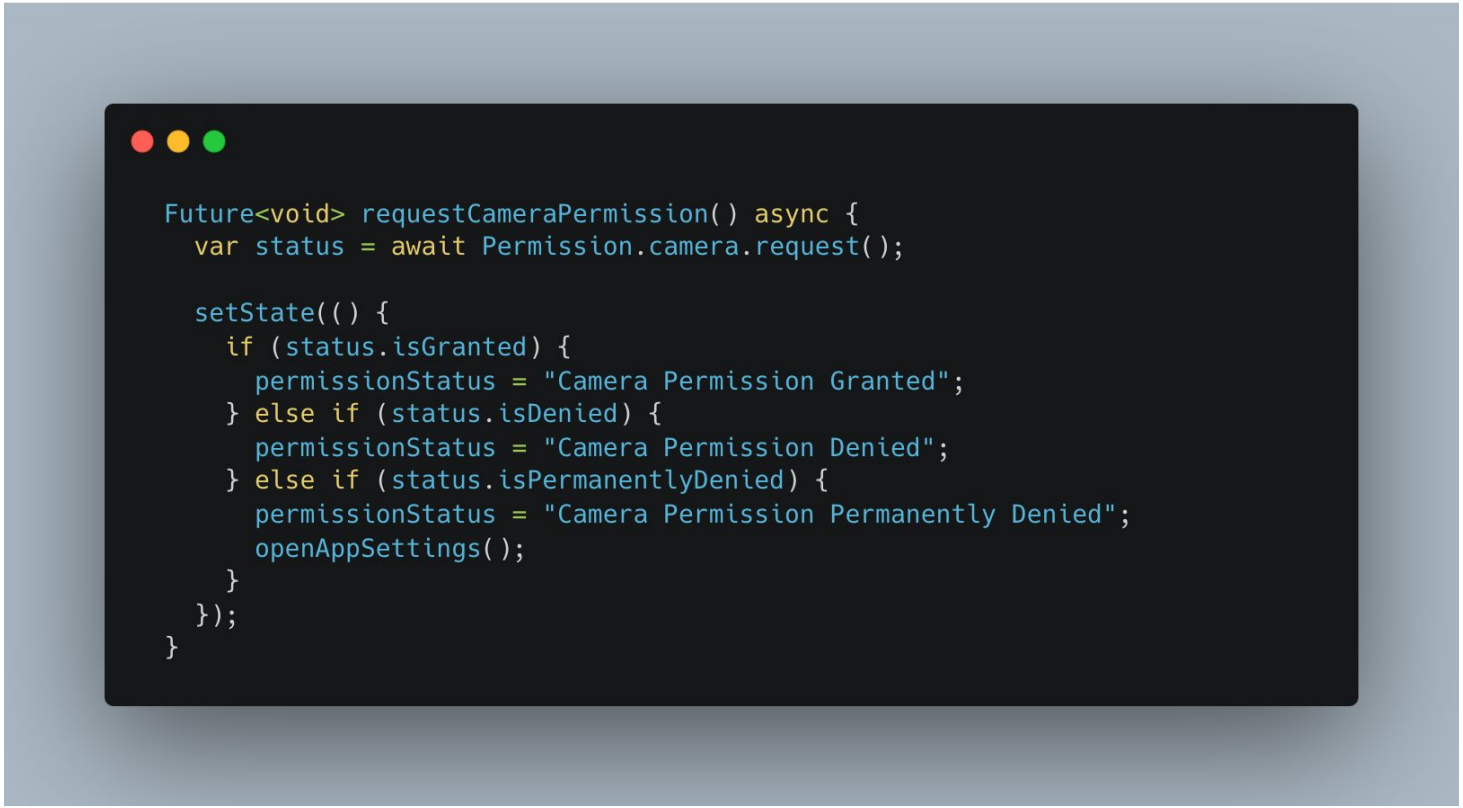
- **사용자 동의**: 사용자가 명시적으로 기능 사용을 허가해야 함
- **법적 요구사항**: 많은 국가에서 앱의 권한 사용에 대해 엄격한 법적 규제 존재
- **사용자 경험**: 적절한 권한 요청은 사용자 경험을 개선하고 불필요한 방해를 줄임

Permission in Flutter

인기 있는 Flutter 권한 패키지

1. **permission_handler**
   - 설명: 다양한 플랫폼(안드로이드, iOS 등)에서 권한을 요청하고 관리할 수 있게 해주는 종합적인 패키지
   - 사용 예: `Permission.camera.request()`를 통해 카메라 권한 요청
2. **location**
   - 설명: 위치 서비스 권한을 관리하며, 사용자의 현재 위치를 추적
   - 사용 예: `Location().requestPermission()`을 통해 위치 권한 요청

Permission in Flutter



```dart
Future<void> requestCameraPermission() async {
  var status = await Permission.camera.request();

  setState(() {
    if (status.isGranted) {
      permissionStatus = "Camera Permission Granted";
    } else if (status.isDenied) {
      permissionStatus = "Camera Permission Denied";
    } else if (status.isPermanentlyDenied) {
      permissionStatus = "Camera Permission Permanently Denied";
      openAppSettings();
    }
  });
}
```

Permission in Flutter

For apps that need the following functionality please complete the following in your app's `AndroidManifest.xml`

- To schedule notifications the following changes are needed
  - Specify the appropriate permissions between the `<manifest>` tags.
    - `<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>` : this is required so the plugin can known when the device is rebooted. This is required so that the plugin can reschedule notifications upon a reboot
    - If the app requires scheduling notifications with exact timings (aka exact alarms), there are two options since Android 14 brought about behavioural changes (see here for more details)
      - specify `<uses-permission android:name="android.permission.SCHEDULE_EXACT_ALARM" />` and call the `requestExactAlarmsPermission()` exposed by the `AndroidFlutterNotificationsPlugin` class so that the user can grant the permission via the app or
      - specify `<uses-permission android:name="android.permission.USE_EXACT_ALARM" />`. Users will not be prompted to grant permission, however as per the official Android documentation on the `USE_EXACT_ALARM` permission (refer to here and here), this requires the app to target Android 13 (API level 33) or higher and could be subject to approval and auditing by the app store(s) used to publish theapp
  - Specify the following between the `<application>` tags so that the plugin can actually show the scheduled notification(s)

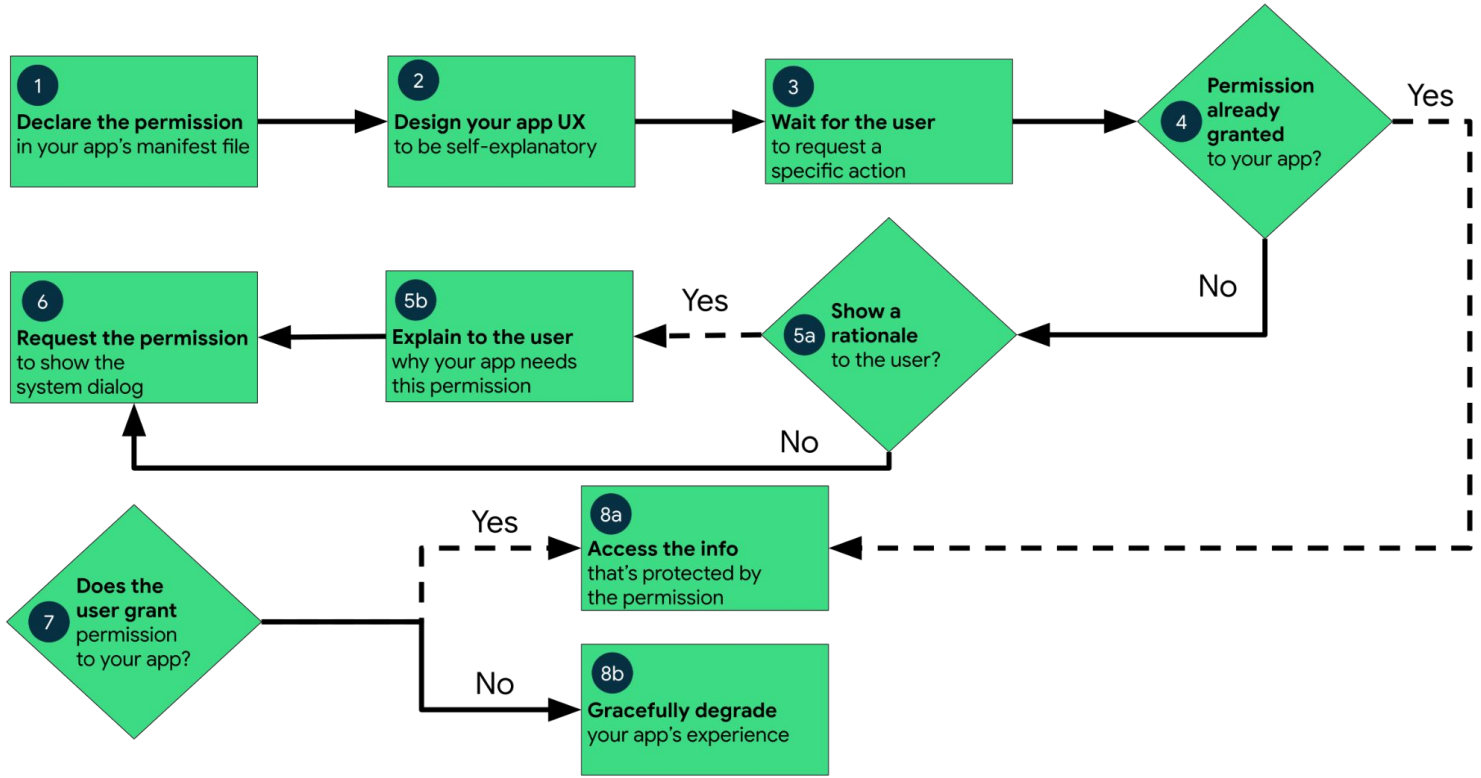
```
<receiver android:exported="false" android:name="com.dexterous.flutterlocalnotifications
<receiver android:exported="false" android:name="com.dexterous.flutterlocalnotifications
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <action android:name="android.intent.action.MY_PACKAGE_REPLACED"/>
        <action android:name="android.intent.action.QUICKBOOT_POWERON" />
        <action android:name="com.htc.intent.action.QUICKBOOT_POWERON"/>
    </intent-filter>
</receiver>
```

Android에서의 Permission

Android에서의 Permission - History

안드로이드 버전	출시 연도	주요 권한 변경사항
Android 1.0	2008	초기 권한 시스템 도입: 설치 시 모든 권한을 사용자에게 요청
Android 6.0 Marshmallow	2015	런타임 권한 도입: 앱 실행 중 권한 요청 가능 권한 그룹화
Android 8.0 Oreo	2017	백그라운드 실행 제한: 백그라운드 앱의 자원 사용 제한으로 권한 관리 강화
Android 9.0 Pie	2018	프라이버시 강화: 백그라운드에서의 마이크, 카메라 사용 제한
Android 10	2019	위치 정보 권한 강화: 포그라운드에서만 위치 정보 접근 허용 권한 자동 재설정: 장기간 미사용 앱의 권한 자동 재설정
Android 11	2020	한 번만 권한 부여: 일부 권한에 대해 일회성 허용 선택 가능 권한 자동 재설정 확장
Android 12	2021	세밀한 권한 관리: 권한 세분화 및 사용자 권한 관리 강화 투명한 권한 사용 기록 제공
Android 13	2022	세분화된 미디어 파일 접근 권한: 사진, 동영상, 오디오 파일에 대한 접근 권한 세분화

Permission in Android



Permission in Android

Permission Related

Permission과 관련된 키워드는?

AndroidManifest.XML , Permission Level , ContextCompat, ActivityCompat, SecurityException, shared_prefs, packages.xml, Permission Group, signature, systemOrSignature, Special permissions, Permission groups, Permission request UI,

Permission in Android

`AndroidManifest.xml`

`AndroidManifest.xml`은 안드로이드 애플리케이션의 핵심 설정 파일로, 앱에 대한 필수 정보를 안드로이드 운영 체제에 제공합니다. 이 파일은 프로젝트의 `app/manifests` 디렉토리에 위치합니다.

요소	설명	예시
패키지 이름	앱의 고유 식별자	`<manifest package="com.example.myapp">`
액티비티	사용자 인터페이스 화면	`<activity android:name=".MainActivity">`
서비스	백그라운드에서 실행되는 컴포넌트	`<service android:name=".MyService">`
브로드캐스트 리시버	시스템 이벤트 수신 컴포넌트	`<receiver android:name=".MyReceiver">`
콘텐트 프로바이더	데이터 공유 컴포넌트	`<provider android:name=".MyContentProvider">`
권한	앱이 필요로 하는 권한	`<uses-permission android:name="android.permission.INTERNET">`
하드웨어/소프트웨어 기능	앱이 사용하는 기기 기능	`<uses-feature android:name="android.hardware.camera">`
앱 버전	앱의 버전 정보	`<manifest android:versionCode="1" android:versionName="1.0">`
SDK 버전	지원하는 안드로이드 버전	`<uses-sdk android:minSdkVersion="21" android:targetSdkVersion="30">`
앱 테마 및 아이콘	앱의 시각적 스타일	`<application android:icon="@mipmap/ic_launcher" android:theme="@style/AppTheme">`
백업 설정	앱 데이터 백업 허용 여부	`<application android:allowBackup="true">`
화면 방향	앱의 화면 방향 설정	`<activity android:screenOrientation="portrait">`
인텐트 필터	컴포넌트가 응답할 수 있는 인텐트	`<intent-filter><action android:name="android.intent.action.MAIN" /></intent-filter>`
메타데이터	추가 정보 제공	`<meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version">`

Permission in Android

Permission Level

퍼미션 레벨은 권한이 필요로 하는 보안 수준을 의미합니다. 안드로이드에서는 퍼미션을 크게 세 가지 레벨로 분류합니다:

- **Normal**: 사용자에게 중요한 보안 위험을 초래하지 않는 권한 (예: 인터넷 사용 권한).
- **Dangerous**: 사용자 데이터나 디바이스 기능에 영향을 미칠 수 있는 권한 (예: 연락처 접근 권한).
- **Signature**: 같은 개발자가 서명한 애플리케이션끼리만 사용할 수 있는 권한.

Permission in Android

ContextCompat

- Context 관련 작업을 처리

주요 기능:

- 리소스 접근
- 권한 체크
- 색상 리소스 가져오기
- 파일 관련 작업

```
int color = ContextCompat.getColor(context, R.color.my_color);
boolean hasPermission = ContextCompat.checkSelfPermission(context, Manifest.permission.CAMERA) ==
PackageManager.PERMISSION_GRANTED;
```

Permission in Android

ContextCompat

`ContextCompat` 클래스는 안드로이드에서 하위 호환성을 제공하기 위해 사용되는 유틸리티 클래스입니다. 주로 API 레벨에 따른 호환성을 쉽게 관리할 수 있도록 도와줍니다. 퍼미션과 관련해서는 다음과 같은 메소드를 제공합니다:

이 메소드를 통해 특정 권한이 부여되었는지 여부를 확인할 수 있습니다.

Permission in Android

ActivityCompat

- Activity 관련 작업을 처리

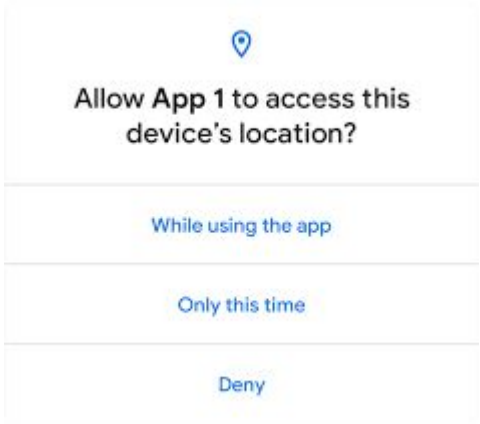
주요 기능:

- 런타임 권한 요청
- 결과를 반환하는 액티비티 시작
- 액티비티 재생성

```
ActivityCompat.requestPermissions(activity, new String[]{Manifest.permission.PERMISSION_NAME}, REQUEST_CODE);
```

Permission in Android

Runtime Permission



RuntimePermission은 안드로이드 6.0 (Marshmallow, API 레벨 23)부터 도입된 권한 관리 방식.

- 애플리케이션이 실행 중에 필요한 권한을 사용자에게 요청하고, 사용자가 이를 승인하거나 거부할 수 있도록 한다.

Permission in Android

SecurityException

SecurityException은 보안 관련 오류가 발생했을 때 던져지는 예외입니다. 주로 애플리케이션이 허가되지 않은 작업을 시도했을 때 발생합니다. 예를 들어, 명시되지 않은 퍼미션을 요구할 때 발생할 수 있습니다.

packages.xml

packages.xml 파일은 안드로이드 시스템 내부의 /data/system 디렉토리에 위치하며, 디바이스에 설치된 모든 패키지에 대한 정보를 저장하는 XML 파일입니다. 각 패키지의 퍼미션 정보와 관련된 내용도 포함되어 있습니다.

Permission Group

퍼미션 그룹은 기능적으로 유사한 퍼미션을 그룹화한 것입니다. 예를 들어, CONTACTS 그룹에는 연락처 읽기와 쓰기 퍼미션이 포함됩니다. 퍼미션 그룹을 통해 사용자에게 퍼미션을 일괄적으로 묻는 것이 가능합니다.

Permission in Android

signature

서명 퍼미션은 같은 서명을 가진 애플리케이션들 사이에서만 사용될 수 있는 퍼미션을 의미합니다. 같은 개발자가 만든 애플리케이션들끼리 데이터를 공유하거나 특정 기능을 사용할 수 있도록 합니다.

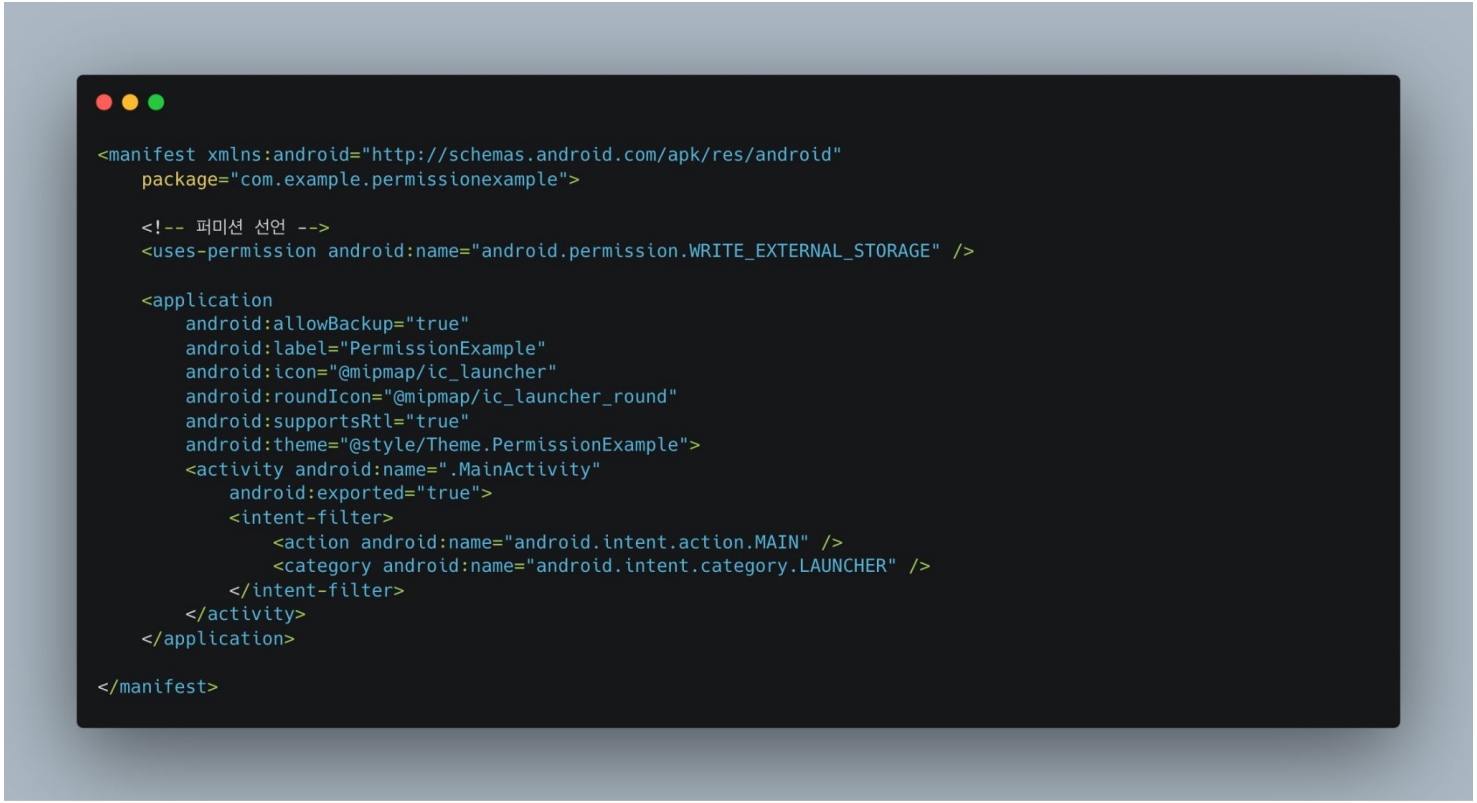
systemOrSignature

이 퍼미션 레벨은 시스템 애플리케이션 또는 같은 서명을 가진 애플리케이션에서만 사용할 수 있는 퍼미션을 의미합니다. 높은 수준의 보안이 필요한 기능을 보호하기 위해 사용됩니다.

Special permissions

특수 퍼미션은 일반 퍼미션보다 더 높은 보안이 필요한 권한을 의미합니다. 예를 들어, `SYSTEM_ALERT_WINDOW`나 `WRITE_SETTINGS` 등이 이에 해당합니다. 사용자는 설정 메뉴에서 직접 이 권한을 부여해야 합니다.

Permission in Android



```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.permissionexample">

    <!-- 퍼미션 선언 -->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application
        android:allowBackup="true"
        android:label="PermissionExample"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.PermissionExample">
        <activity android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Permission in Android

```
class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            PermissionExampleTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    PermissionRequestScreen()
                }
            }
        }
    }
}
```

Permission in Android

```
@Composable
fun PermissionRequestScreen() {
    var permissionGranted by remember { mutableStateOf(false) }
    var permissionDenied by remember { mutableStateOf(false) }

    val requestPermissionLauncher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.RequestPermission()
    ) { isGranted: Boolean ->
        if (isGranted) {
            permissionGranted = true
            permissionDenied = false
        } else {
            permissionGranted = false
            permissionDenied = true
        }
    }

    Column(modifier = Modifier.padding(16.dp)) {
        if (permissionGranted) {
            Text(text = "Permission already granted")
        } else {
            Text(text = "Permission not granted")
        }

        if (permissionDenied) {
            Text(text = "Permission denied")
        }

        Button(onClick = {
            if (ContextCompat.checkSelfPermission(
                    context,
                    Manifest.permission.WRITE_EXTERNAL_STORAGE
                ) != PackageManager.PERMISSION_GRANTED
            ) {
                requestPermissionLauncher.launch(Manifest.permission.WRITE_EXTERNAL_STORAGE)
            } else {
                permissionGranted = true
            }
        }) {
            Text(text = "Request Permission")
        }
    }
}
```

Permission in Android

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.app1">

    <!-- 퍼미션 선언 -->
    <permission
        android:name="com.example.app1.permission.MY_SIGNATURE_PERMISSION"
        android:protectionLevel="signature" />

    <uses-permission android:name="com.example.app1.permission.MY_SIGNATURE_PERMISSION" />

    <application
        android:allowBackup="true"
        android:label="App1"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.App1">
        <activity android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Permission in Android

ActivityCompat & ActivityResultContract

1. 목적:
    - 액티비티 결과 처리의 표준화
    - 권한 요청, 이미지 선택 등 공통 작업의 간소화
2. 주요 구성 요소:
    - Contract: 입력과 출력 타입을 정의
    - Launcher: 결과를 받아올 작업 실행
3. 장점:
    - 타입 안전성 보장
    - 라이프사이클 인식
    - 코드 모듈화 및 재사용성 향상
    - 구성 변경 시 자동 상태 유지

```kotlin
class MainActivity : AppCompatActivity() {
    private val CAMERA_PERMISSION_CODE = 100

    fun checkCameraPermission() {
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) !=
PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.CAMERA), CAMERA_PERMISSION_CODE)
        } else {
            // 카메라 권한이 있는 경우의 로직
        }
    }

    override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String>, grantResults: IntArray) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        if (requestCode == CAMERA_PERMISSION_CODE) {
            if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                // 권한이 승인된 경우의 로직
            } else {
                // 권한이 거부된 경우의 로직
            }
        }
    }
}
```

```kotlin
class MainActivity : AppCompatActivity() {
    private val requestCameraPermission = registerForActivityResult(ActivityResultContracts.RequestPermission()) {
isGranted ->
        if (isGranted) {
            // 권한이 승인된 경우의 로직
        } else {
            // 권한이 거부된 경우의 로직
        }
    }

    fun checkCameraPermission() {
        requestCameraPermission.launch(Manifest.permission.CAMERA)
    }
}
```

Permission in Android

Conclusion

iOS에서의 Permission

권한 타입	설명	NS 키
위치 서비스	앱이 내비게이션 또는 위치 추적 등을 위해 사용자의 위치 데이터에 접근	`NSLocationWhenInUseUsageDescription`
카메라	앱이 사진을 찍거나 비디오를 녹화하기 위해 디바이스의 카메라를 사용	`NSCameraUsageDescription`
마이크	앱이 오디오를 녹음하거나 마이크를 사용할 수 있게 합니다 (예: 통화 또는 음성 녹음 시).	`NSMicrophoneUsageDescription`
사진	앱이 사진이나 비디오를 저장하거나 업로드하기 위해 사진 라이브러리에 접근할 수 있게 합니다.	`NSPhotoLibraryUsageDescription`
연락처	앱이 연락처 목록에 접근하여 연락처를 읽거나 수정할 수 있게 합니다.	`NSContactsUsageDescription`
달력	앱이 달력에 접근하여 이벤트를 읽거나 생성 또는 수정할 수 있게 합니다.	`NSCalendarsUsageDescription`
리마인더	앱이 리마인더를 읽거나 수정할 수 있게 합니다.	`NSRemindersUsageDescription`
건강	앱이 건강 앱에 저장된 건강 및 피트니스 데이터에 접근할 수 있게 합니다.	`NSHealthShareUsageDescription`

iOS에서의 Permission

권한 요청의 기본 흐름

1. **권한 선언**: 앱의 `Info.plist` 파일에 해당 권한의 사용 이유를 명시
2. **권한 요청**: 앱이 처음으로 해당 기능을 사용할 때 사용자에게 권한을 요청
3. **권한 처리**: 사용자가 권한 요청을 수락하거나 거부했을 때의 결과를 처리

iOS에서의 Permission

```
<key>NSCameraUsageDescription</key>
<string>이 앱은 사진을 찍기 위해 카메라 접근 권한이 필요합니다.</string>
<key>NSMicrophoneUsageDescription</key>
<string>이 앱은 음성 녹음을 위해 마이크 접근 권한이 필요합니다.</string>
<key>NSLocationWhenInUseUsageDescription</key>
<string>이 앱은 위치 정보를 사용하여 사용자 맞춤 서비스를 제공합니다.</string>
```

iOS에서의 Permission

```swift
import CoreLocation

class ViewController: UIViewController, CLLocationManagerDelegate {
    let locationManager = CLLocationManager()

    override func viewDidLoad() {
        super.viewDidLoad()

        locationManager.delegate = self
        locationManager.requestWhenInUseAuthorization()
    }

    func locationManager(_ manager: CLLocationManager, didChangeAuthorization status: CLAuthorizationStatus) {
        switch status {
        case .authorizedWhenInUse, .authorizedAlways:
            print("Location access granted")
        case .denied, .restricted:
            print("Location access denied")
        default:
            break
        }
    }
}
```

iOS에서의 Permission

특성	iOS	Android
권한 요청 시점	기능 첫 사용 시	개발자가 선택한 시점
권한 그룹화	관련 권한 그룹화	개별 권한 관리
요청 횟수	각 권한당 1회	여러 번 가능 (사용자 설정에 따라)
설정 변경	설정 앱에서만 가능	설정 앱 또는 앱 내에서 가능
런타임 권한 도입	iOS 6 이후 모든 버전	Android 6.0 (API 23) 이후
거부 후 재요청	불가능 (설정에서 변경)	가능 (앱에서 재요청)
백그라운드 권한	일부 권한에 대해 옵션 제공	별도 요청 필요 (Android 10+)
권한 선언	Info.plist에 사용 이유 추가	AndroidManifest.xml에 선언
권한 API	일관되고 단순함	버전에 따라 다른 API 사용

iOS에서의 Permission

안드로이드:

- 안드로이드는 통합된 권한 API를 사용합니다.
- 주로 `ActivityCompat.requestPermissions()` 메서드를 사용하여 여러 권한을 한 번에 요청할 수 있습니다.
- 권한 체크와 요청을 위해 `ContextCompat.checkSelfPermission()`과 `shouldShowRequestPermissionRationale()` 등의 메서드를 사용합니다.

iOS:

- iOS는 각 기능이나 서비스에 대해 별도의 클래스와 메서드를 사용하여 권한을 요청합니다.
- 예를 들어:
  - 위치 권한: `CLLocationManager`
  - 카메라 권한: `AVCaptureDevice.requestAccess(for:completionHandler:)`
  - 사진 라이브러리 접근: `PHPhotoLibrary.requestAuthorization(_:)`
  - 알림: `UNUserNotificationCenter.requestAuthorization(options:completionHandler:)`

Permission 제한

Permission 제한