Lint tool
and
App permissions best practices

GROUP2:

Drashty Majumdar

Revanth Thantepudi

Shaunak Pandit

Sowmya Chirravuri

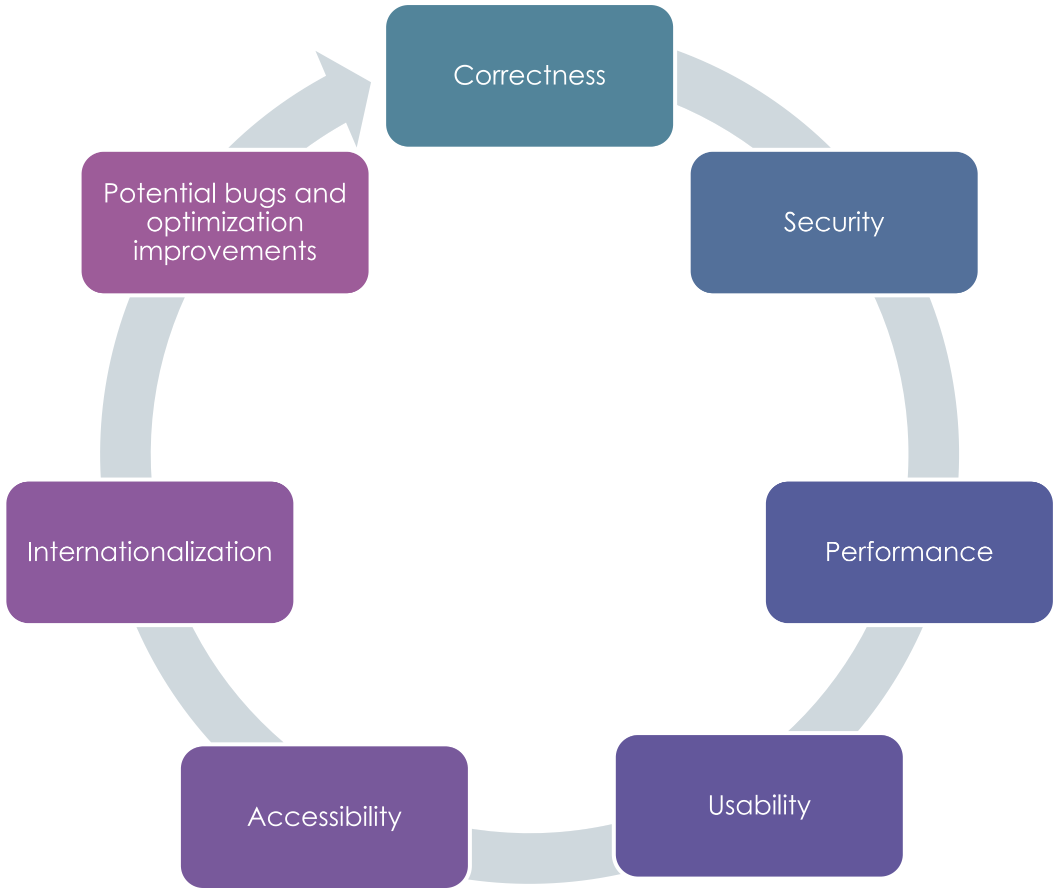
Tharun Poduru

Contents

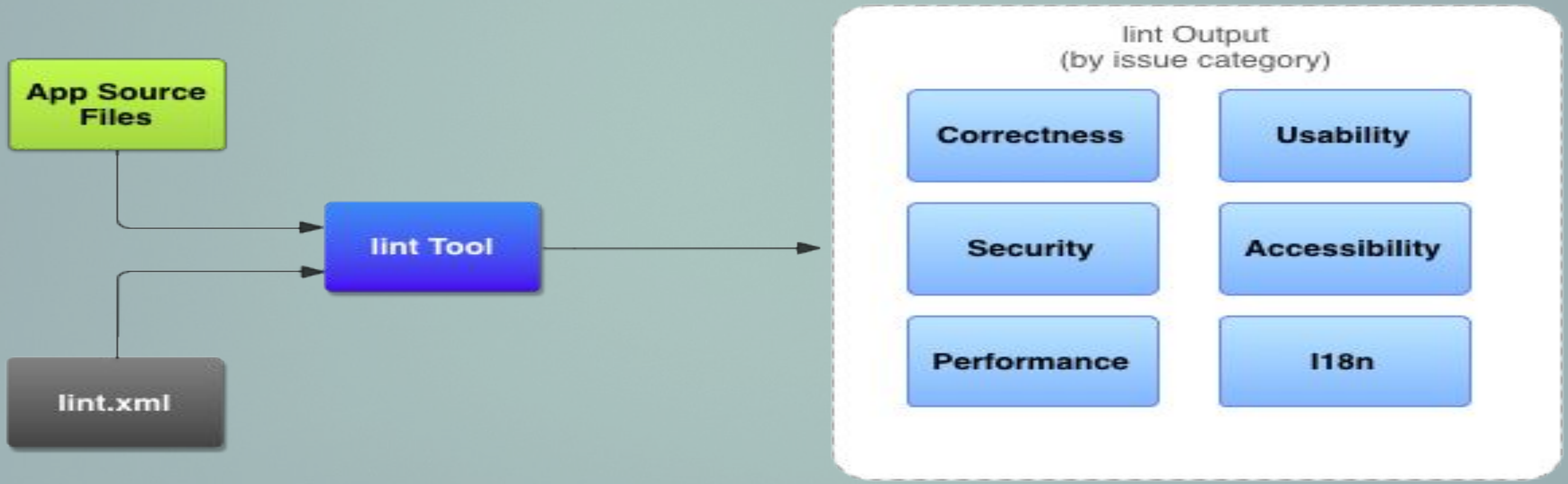
Overview

- Android studio provides a code scanning tool called lint.
- It helps to identify and correct errors with the structural quality of your code without having to execute the app.
- Each problem which is detected by the tool will be displayed with a description message and a severity level.
- It is possible to make improvements to our code depending on the severity of the problem.

Lint tool checks



Workflow with the lint tool



Lint.xml

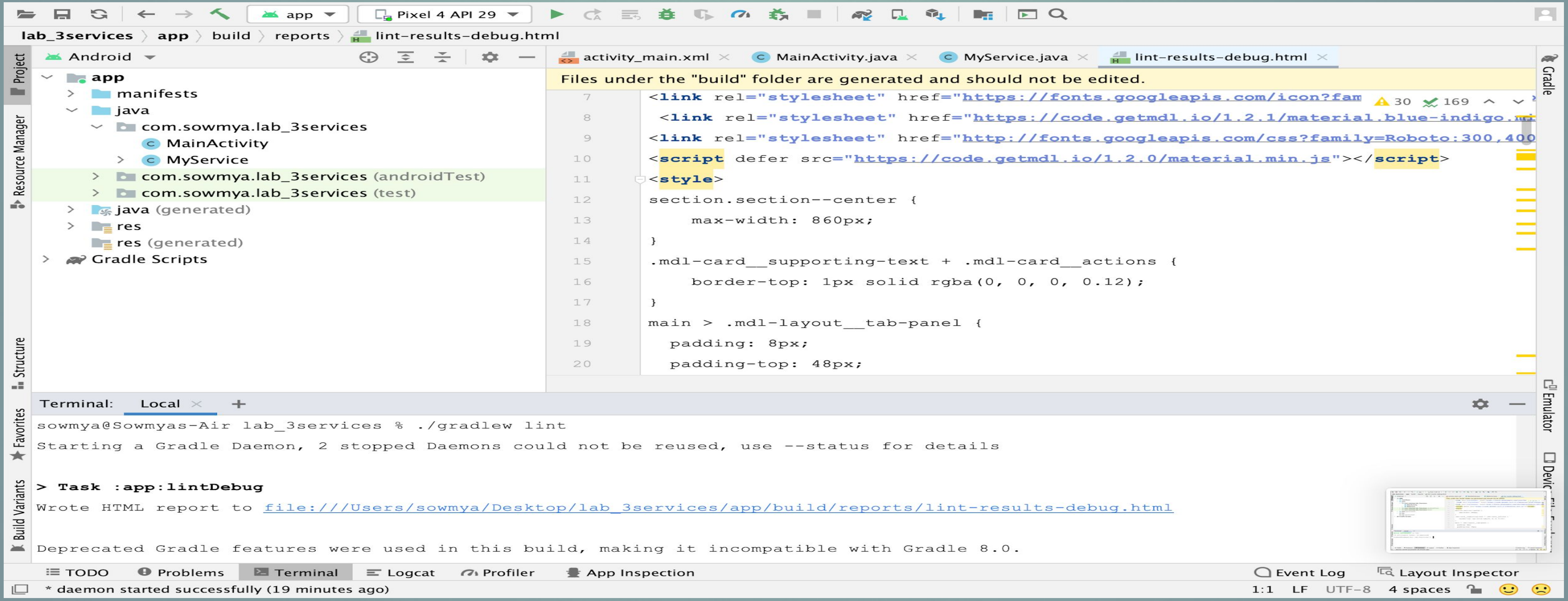
- A configuration file that you can use to specify any lint checks that you want to exclude and to customize problems severity levels.

Executing lint tool in two ways

Command Line

Android Studio(manually)

Using command line



Files under the "build" folder are generated and should not be edited.

```
<link rel="stylesheet" href="https://fonts.googleapis.com/icon?fam
 <link rel="stylesheet" href="https://code.getmdl.io/1.2.1/material.blue-indigo.m
<link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Roboto:300,400
<script defer src="https://code.getmdl.io/1.2.0/material.min.js"></script>
<style>
section.section--center {
    max-width: 860px;
}
.mdl-card__supporting-text + .mdl-card__actions {
    border-top: 1px solid rgba(0, 0, 0, 0.12);
}
main > .mdl-layout__tab-panel {
  padding: 8px;
  padding-top: 48px;
```

```
sowmya@Sowmyas-Air lab_3services % ./gradlew lint
Starting a Gradle Daemon, 2 stopped Daemons could not be reused, use --status for details

> Task :app:lintDebug
Wrote HTML report to file:///Users/sowmya/Desktop/lab_3services/app/build/reports/lint-results-debug.html

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.
```

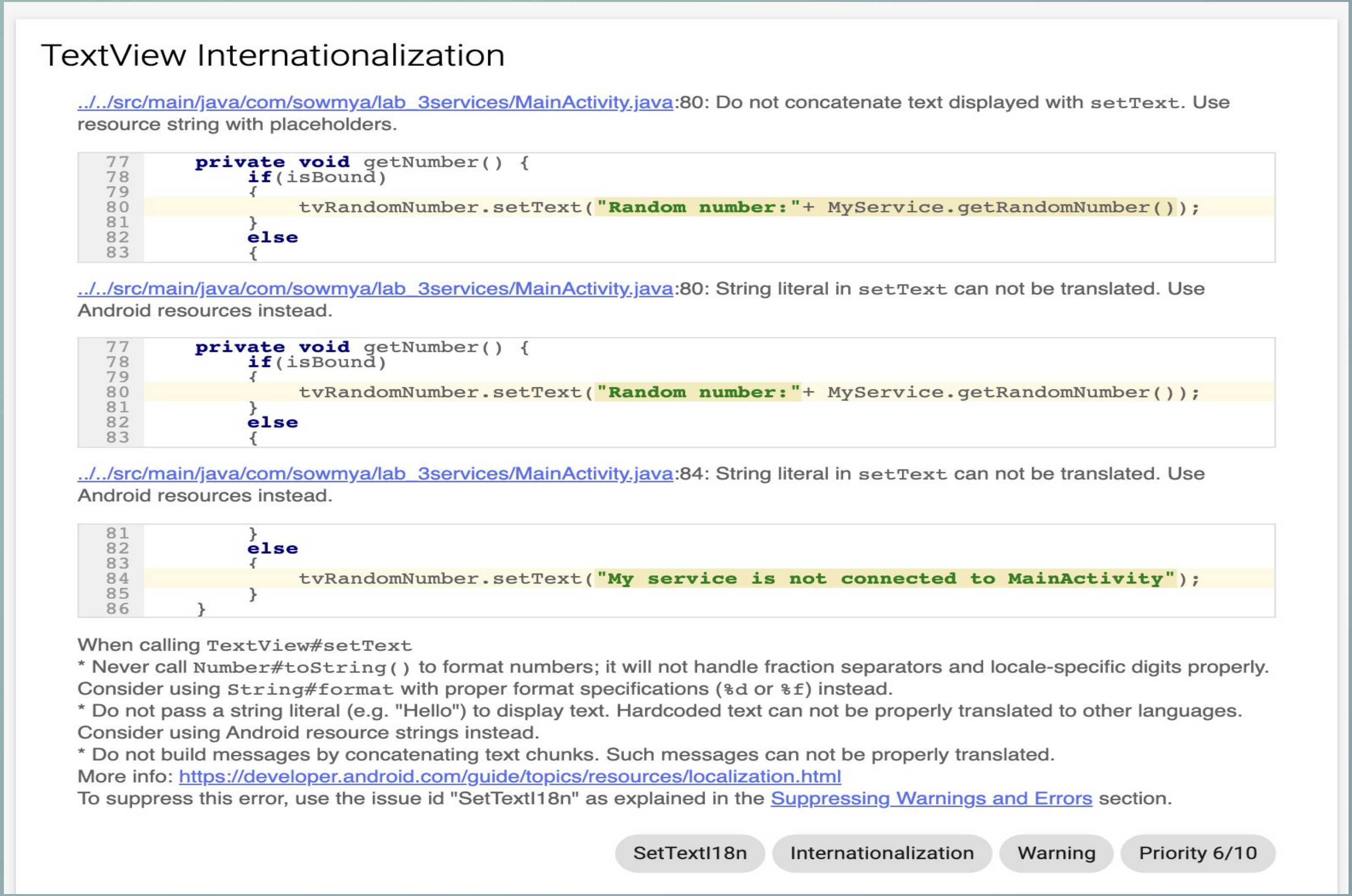
* daemon started successfully (19 minutes ago)

Results from command line

These command line should be used

On Linux or Mac: ./gradlew lint

On Windows :gradlew lint



TextView Internationalization

../../src/main/java/com/sowmya/lab_3services/MainActivity.java:80: Do not concatenate text displayed with `setText`. Use resource string with placeholders.

```
    private void getNumber() {
        if(isBound)
        {
            tvRandomNumber.setText("Random number:"+ MyService.getRandomNumber());
        }
        else
        {
```

../../src/main/java/com/sowmya/lab_3services/MainActivity.java:80: String literal in `setText` can not be translated. Use Android resources instead.

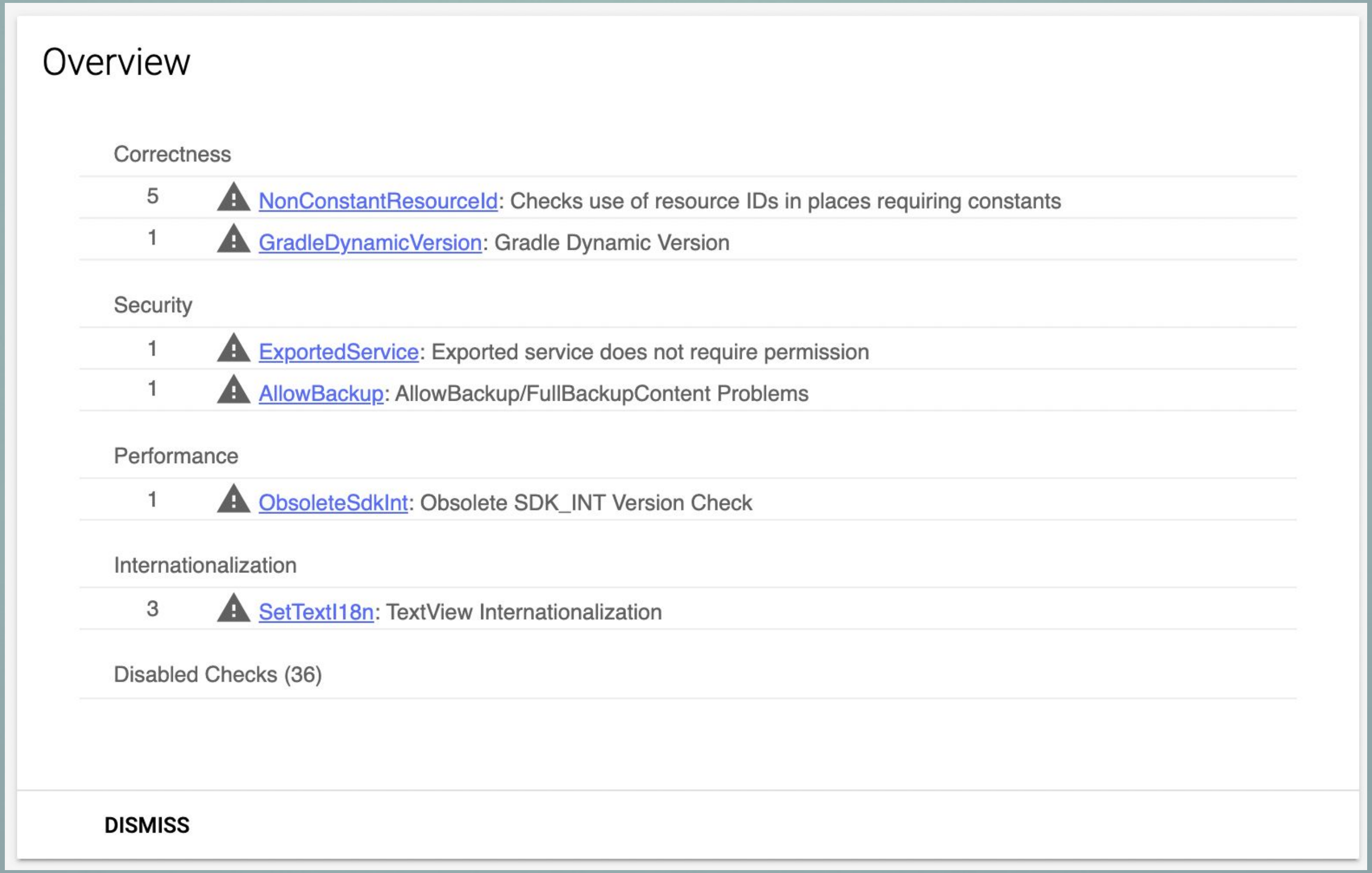
```
    private void getNumber() {
        if(isBound)
        {
            tvRandomNumber.setText("Random number:"+ MyService.getRandomNumber());
        }
        else
        {
```

../../src/main/java/com/sowmya/lab_3services/MainActivity.java:84: String literal in `setText` can not be translated. Use Android resources instead.

```
        }
        else
        {
            tvRandomNumber.setText("My service is not connected to MainActivity");
        }
    }
```

When calling `TextView#setText`
* Never call `Number#toString()` to format numbers; it will not handle fraction separators and locale-specific digits properly. Consider using `String#format` with proper format specifications (`%d` or `%f`) instead.
* Do not pass a string literal (e.g. "Hello") to display text. Hardcoded text can not be properly translated to other languages. Consider using Android resource strings instead.
* Do not build messages by concatenating text chunks. Such messages can not be properly translated.
More info: https://developer.android.com/guide/topics/resources/localization.html
To suppress this error, use the issue id "SetTextI18n" as explained in the Suppressing Warnings and Errors section.

Detailed Results of Code



Overview

Correctness

5	⚠ NonConstantResourceId: Checks use of resource IDs in places requiring constants
1	⚠ GradleDynamicVersion: Gradle Dynamic Version

Security

1	⚠ ExportedService: Exported service does not require permission
1	⚠ AllowBackup: AllowBackup/FullBackupContent Problems

Performance

1	⚠ ObsoleteSdkInt: Obsolete SDK_INT Version Check

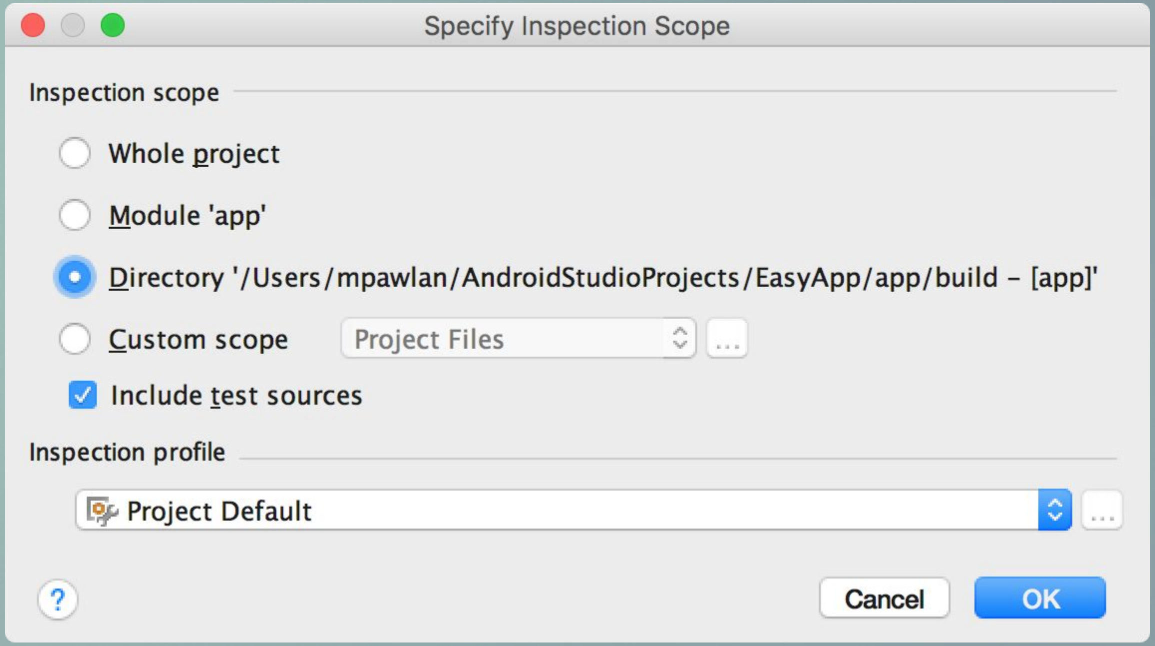
Internationalization

3	⚠ SetTextI18n: TextView Internationalization

Disabled Checks (36)

Set the inspection scope and profile

- In the Android view, open your project and select the project, a folder, or a file that you want to analyze.
- From the menu bar, select Analyze > Inspect Code.
- In the Specify Inspection Scope dialog, review the settings.


Specify Inspection Scope

Inspection scope

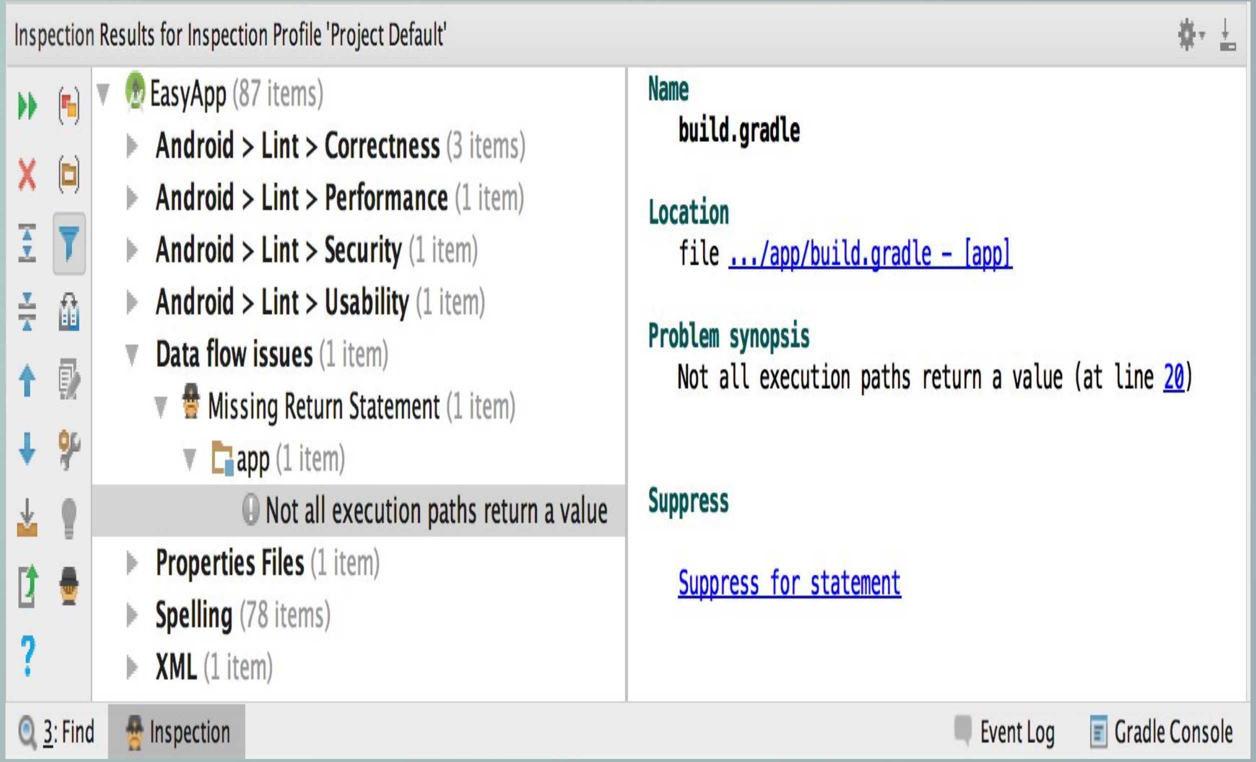
( ) Whole project
( ) Module 'app'
(●) Directory '/Users/mpawlan/AndroidStudioProjects/EasyApp/app/build – [app]'
( ) Custom scope Project Files
[✓] Include test sources

Inspection profile

Project Default

Set the inspection scope and profile

- Under Inspection profile, keep the default profile (Project Default).
- Click OK to run the inspection. it shows lint and other IDE inspection results from the Inspect Code run.
- In the left pane tree view, view the inspection results by expanding and selecting error categories, types, and issues. The right pane displays the inspection report for the selected error category, type, or issue and provides the name and location of the error. Where applicable, the inspection report displays other information such as a problem synopsis to help you correct the problem.
- In the left pane tree view, right-click a category, type, or issue to display the context menu. Depending on the context, you can do all or some of the following: jump to source, exclude and include selected items, suppress problems, edit settings, manage inspection alerts, and rerun an inspection.



Inspection Results for Inspection Profile 'Project Default'

```
▼ EasyApp (87 items)
  ▸ Android > Lint > Correctness (3 items)
  ▸ Android > Lint > Performance (1 item)
  ▸ Android > Lint > Security (1 item)
  ▸ Android > Lint > Usability (1 item)
  ▼ Data flow issues (1 item)
    ▼ Missing Return Statement (1 item)
      ▼ app (1 item)
          Not all execution paths return a value
  ▸ Properties Files (1 item)
  ▸ Spelling (78 items)
  ▸ XML (1 item)
```

Name
build.gradle

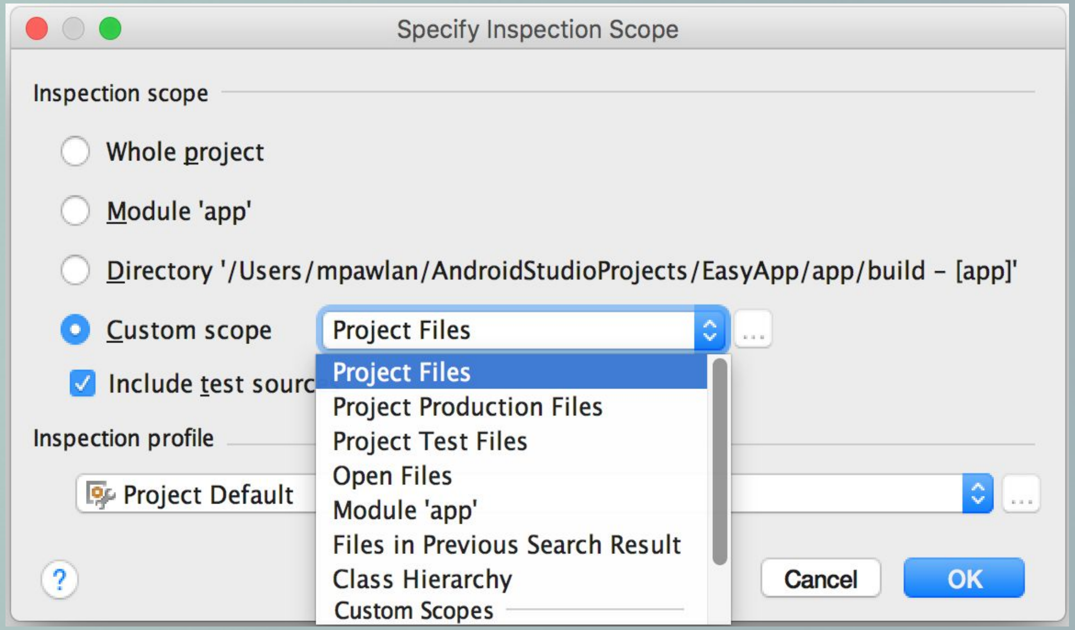
Location
file .../app/build.gradle - [app]

Problem synopsis
Not all execution paths return a value (at line 20)

Suppress
Suppress for statement

Use a custom scope

- In the Specify Inspection Scope dialog, click Custom scope.
- Click the Custom scope drop-down list to display your options.
- Click OK


Specify Inspection Scope

Inspection scope

Whole project

Module 'app'

Directory '/Users/mpawlan/AndroidStudioProjects/EasyApp/app/build – [app]'

Custom scope: Project Files

Include test sourc

Inspection profile

Project Default

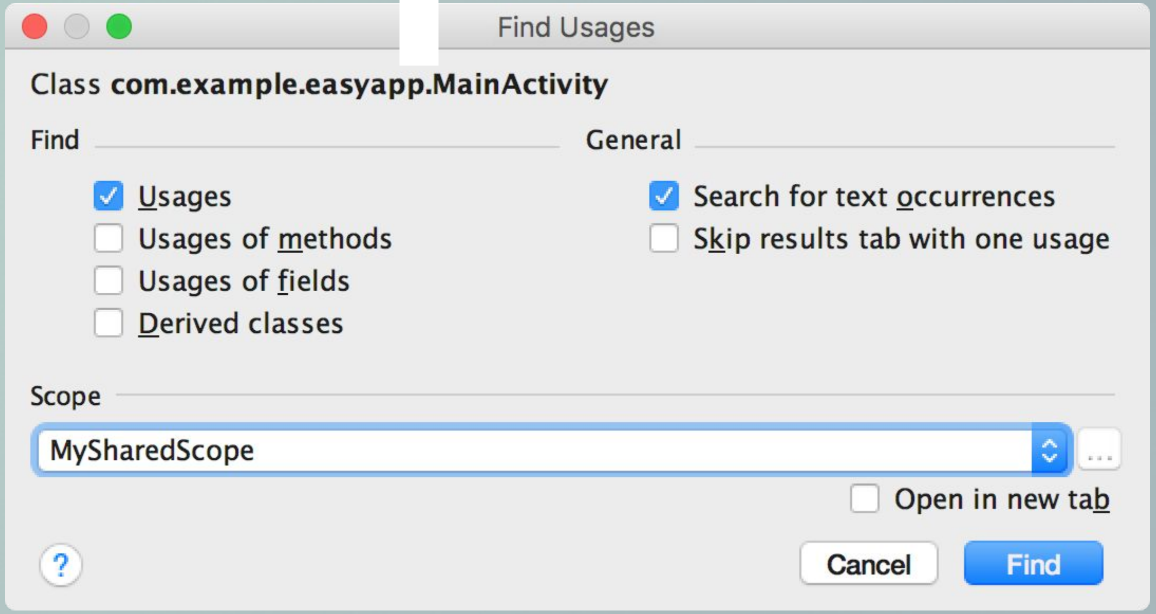
- **Project Files:** All of the files in the current project.
- **Project Production Files:** Only the production files in the current project.
- **Project Test Files:** Only the test files in the current project.
- **Open Files:** Only the files you have open in the current project.
- **Module your-module:** Only the files in the corresponding module folder in your current project.
- **Current File:** Only the current file in your current project. Appears when you have a file or folder selected.
- **Class Hierarchy:** When you select this one and click **OK**, a dialog appears with all of the classes in the current project. Use the **Search by Name** field in the dialog to filter and select the classes to inspect. If you do not filter the classes list, code inspection inspects all of the classes.

- Click add + to define a new scope.

- Both the local and shared scopes are used within the project for the Inspect Code feature. A Shared scope can also be used with other project features that have a scope field. For example, when you click Edit Settings to change the settings for Find Usages, the resulting dialog has a Scope field where you can select a shared scope.

- Give the scope a name and click OK.The right pane of the Scopes dialog populates with options that enable you to define the custom scope.

- From the drop-down list, select Project.A list of available projects appears.


Find Usages

Class **com.example.easyapp.MainActivity**

Find
- [✓] Usages
- [ ] Usages of methods
- [ ] Usages of fields
- [ ] Derived classes

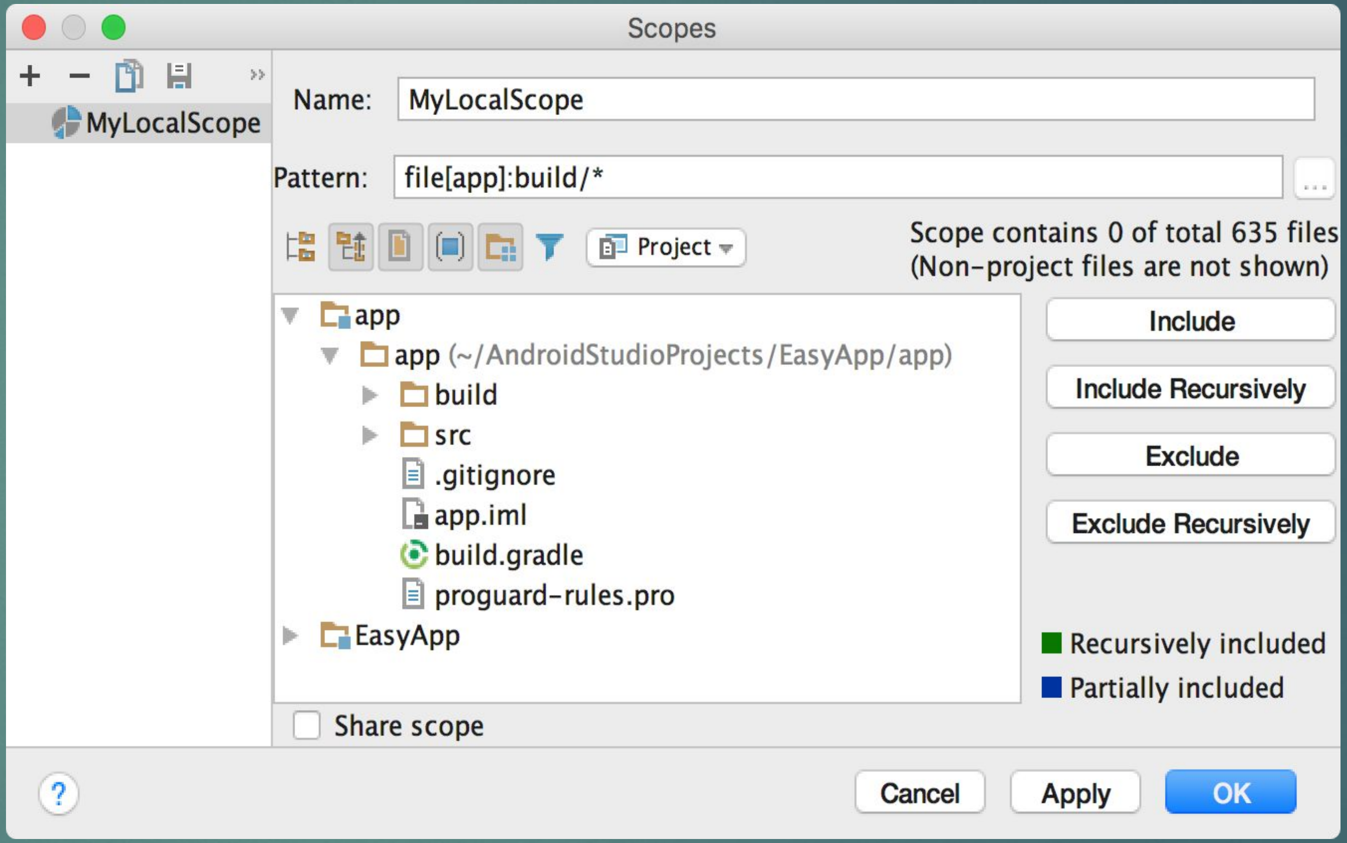
General
- [✓] Search for text occurrences
- [ ] Skip results tab with one usage

Scope: MySharedScope

- [ ] Open in new tab

Expand the project folders, select what you want to add to the custom scope, and click one of the buttons on the right.

Click **OK**. The custom scope appears at the bottom of the drop-down list.


Scopes

MyLocalScope

Name: MyLocalScope

Pattern: file[app]:build/*

Scope contains 0 of total 635 files
(Non-project files are not shown)

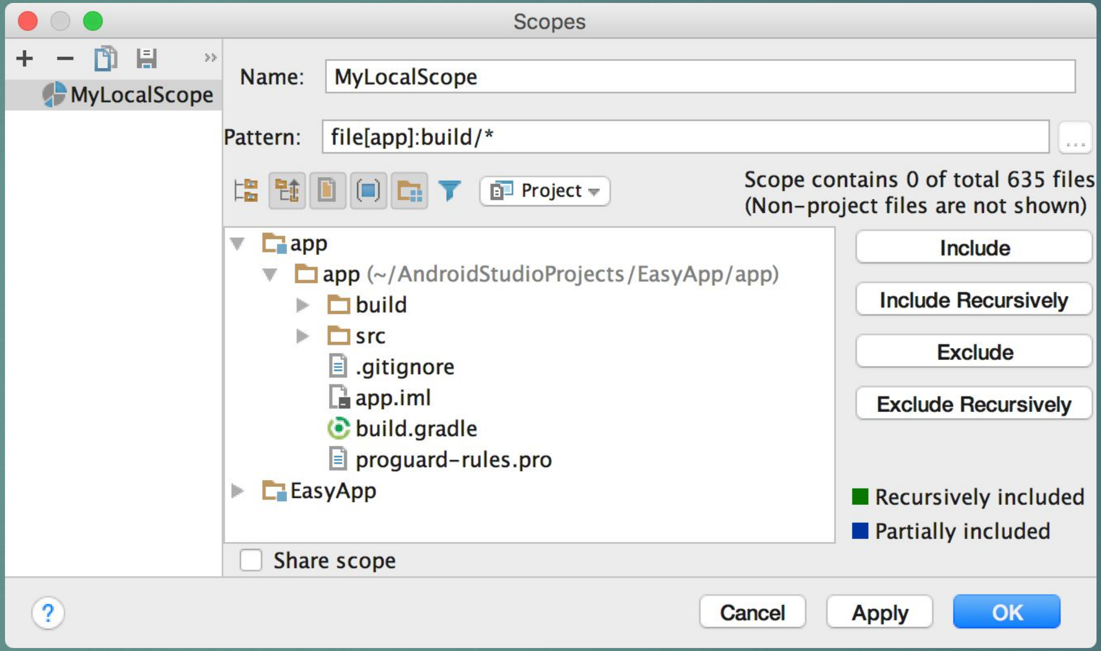
```
app
  app (~/AndroidStudioProjects/EasyApp/app)
    build
    src
    .gitignore
    app.iml
    build.gradle
    proguard-rules.pro
EasyApp
```

1. **Include**: Include this folder and its files, but do not include any of its subfolders.

2. **Include Recursively**: Include this folder and all of its files and subfolders and their files.

3. **Exclude**: Exclude this folder and its files, but do not exclude any of its subfolders.

4. **Exclude Recursively**: Exclude this folder and all of its files and subfolders and their files.


Scopes

MyLocalScope

Name: MyLocalScope

Pattern: file[app]:build/*

Scope contains 0 of total 635 files
(Non-project files are not shown)

```
▼ app
  ▼ app (~/AndroidStudioProjects/EasyApp/app)
    ▶ build
    ▶ src
      .gitignore
      app.iml
      build.gradle
      proguard-rules.pro
▶ EasyApp
```

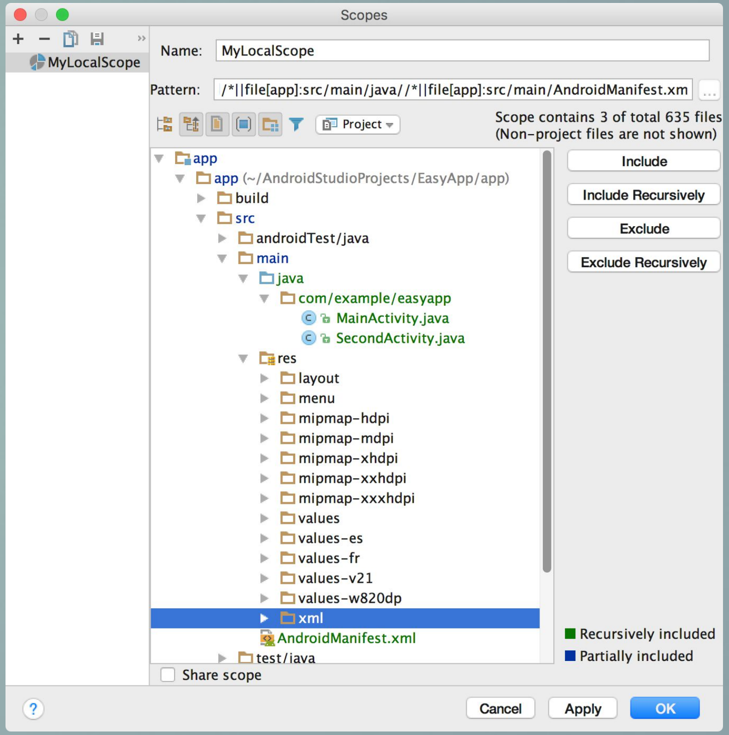
🟩 Recursively included
🟦 Partially included

[ ] Share scope

- Main folder is included, and that the java folder is included recursively. The blue indicates partially included folders and the green indicates recursively included folders and files.

- If you select the java folder and click Exclude Recursively, the green highlighting goes away on the java folder and all folders and files under it.

- If you instead select the green highlighted MainActivity.java file and click Exclude, MainActivity.java is no longer green highlighted but everything else under the java folder is green highlighted.


Scopes

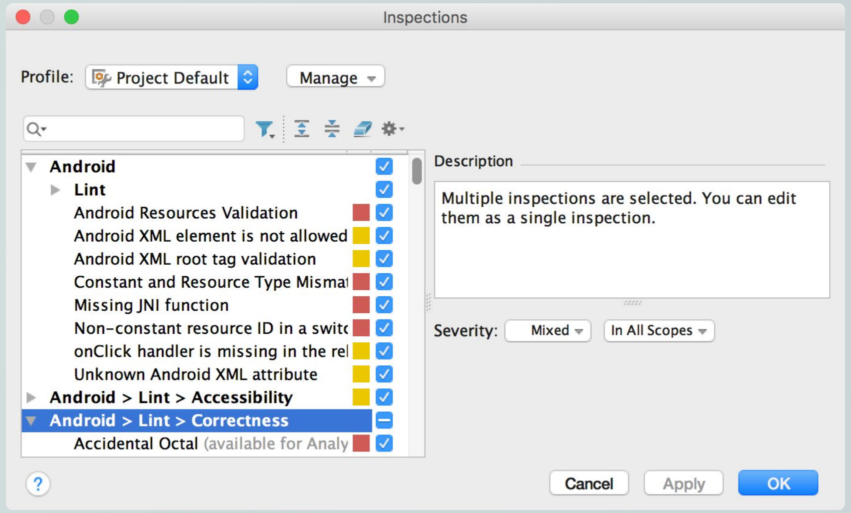
Name: MyLocalScope

Pattern: /*||file[app]:src/main/java//*||file[app]:src/main/AndroidManifest.xm

Scope contains 3 of total 635 files
(Non-project files are not shown)

Review and edit inspection profiles

- Select Analyze > Inspect Code.
- In the Specify Scope dialog under Inspection Profile, click More.
- Select the Profile drop-down list to toggle between Default (Android Studio) and Project Default (the active project) inspections.
- In the Inspections dialog in the left pane, select a top-level profile category, or expand a group and select a specific profile. When you select a profile category, you can edit all of the inspections in that category as a single inspection.
- Select the Manage drop-down list to copy, rename, add descriptions to, export, and import inspections.
- When you're done, click OK.


Inspections

Profile: Project Default Manage

- Android ✓
  - Lint ✓
  - Android Resources Validation 🟥 ✓
  - Android XML element is not allowed 🟨 ✓
  - Android XML root tag validation 🟨 ✓
  - Constant and Resource Type Mismat 🟥 ✓
  - Missing JNI function 🟥 ✓
  - Non-constant resource ID in a switc 🟥 ✓
  - onClick handler is missing in the rel 🟥 ✓
  - Unknown Android XML attribute 🟨 ✓
- Android > Lint > Accessibility 🟨 ✓
- Android > Lint > Correctness 🟥 −
  - Accidental Octal (available for Analy 🟥 ✓

Description

Multiple inspections are selected. You can edit them as a single inspection.

Severity: Mixed In All Scopes

Cancel Apply OK

App permissions best practices

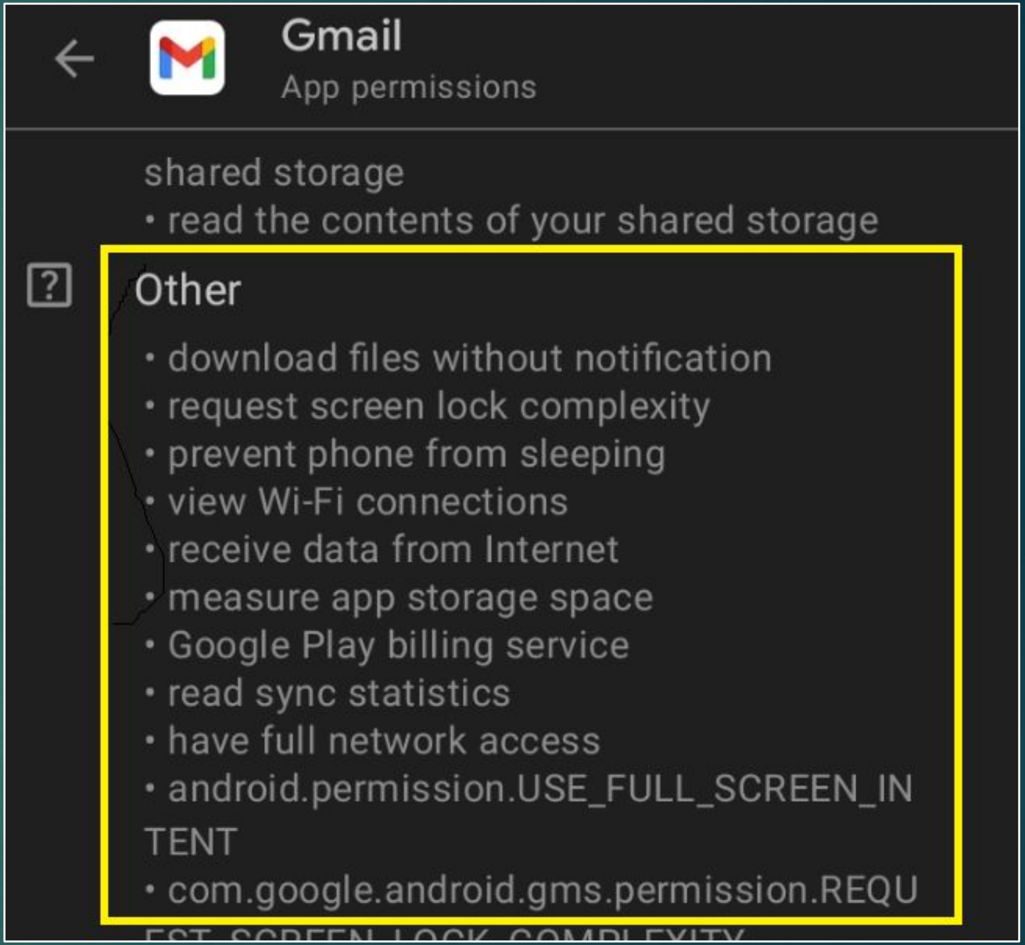
Goals

- Control
- Data minimization
- Transparency

Major types of permissions

1) __Install-time__:

- Allow LIMITED access to sensitive information
- Allow restricted actions that MINIMALLY affect the system or other apps
- System automatically grants the permissions upon installation
- Example: Access to internet



shared storage
• read the contents of your shared storage

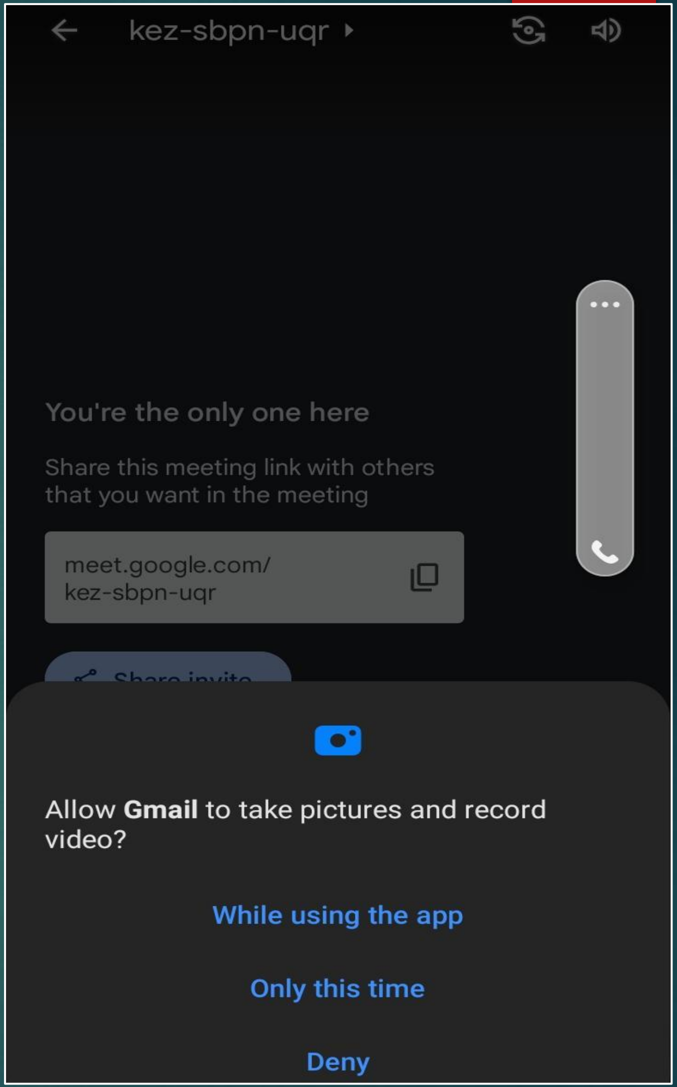
Other
• download files without notification
• request screen lock complexity
• prevent phone from sleeping
• view Wi-Fi connections
• receive data from Internet
• measure app storage space
• Google Play billing service
• read sync statistics
• have full network access
• android.permission.USE_FULL_SCREEN_INTENT
• com.google.android.gms.permission.REQU

Install-time permissions of Gmail app

Major types of permissions

2) <u>Runtime:</u>

- Allow ADDITIONAL access to sensitive information.
- Allow restricted actions that SUBSTANTIALLY affect the system or other apps.
- System asks when situation arises.
- Introduced in Android M. "Only this time" option introduced in Android 11
- Example: Access to file system



You're the only one here

Share this meeting link with others that you want in the meeting

meet.google.com/
kez-sbpn-uqr

Allow **Gmail** to take pictures and record video?

While using the app

Only this time

Deny

Runtime permission in Google Meet while starting a meeting

Pros of using lint tools

- The lint tool checks for structural code problems that could affect the quality and performance of your Android application.

- Applications that use lint tools are more maintainable and scalable

Best Practices

- ✔ **Request a minimal number of permissions:**

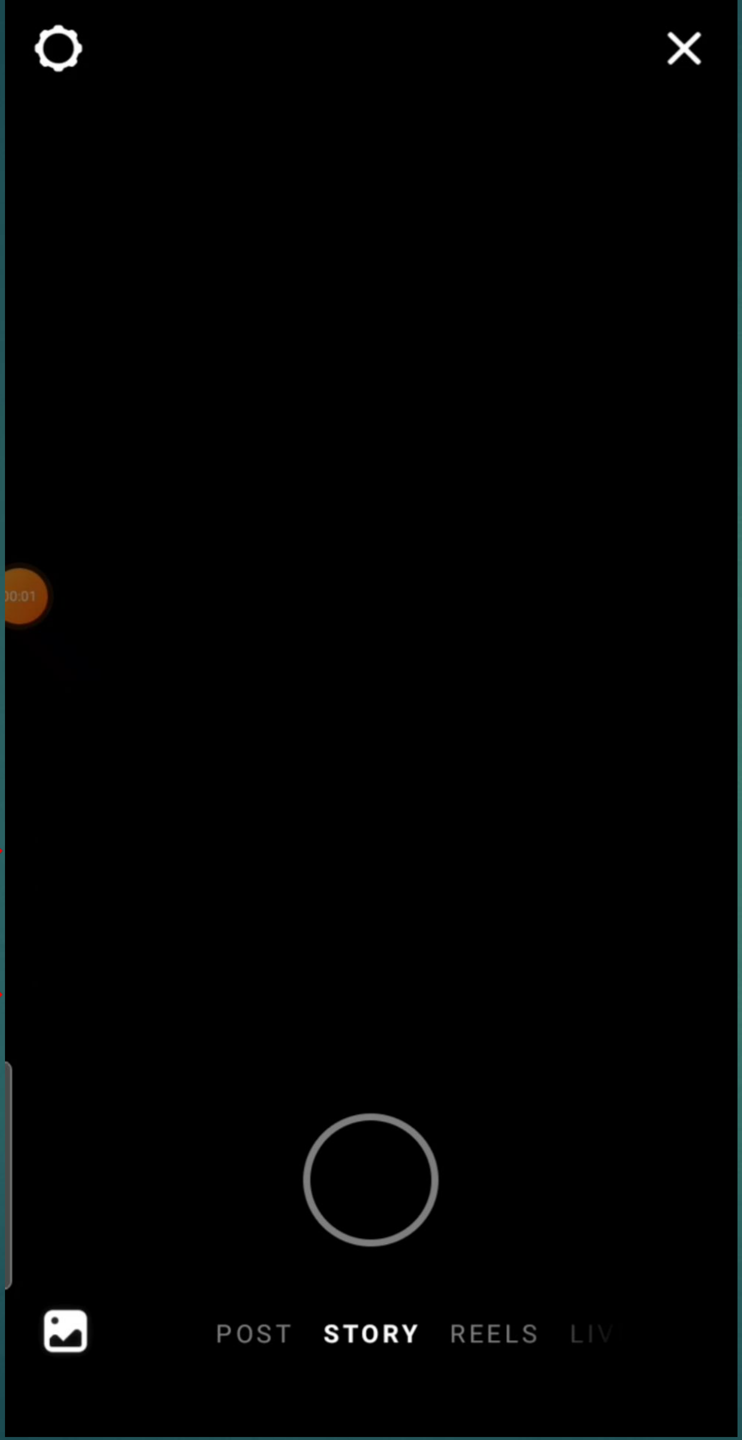
Action	**Do** ✓	**Don't** ✗
Show nearby places	Ask for address/postal code OR declare the `ACCESS_COARSE_LOCATION` permission	declare the `ACCESS_FINE_LOCATION` permission
Take a photo/video	Invoke `ACTION_IMAGE_CAPTURE` intent action that can be sent to have the camera app capture an image/video and return it	Declare `CAMERA` permission
Open a document	Invoke `ACTION_OPEN_DOCUMENT` intent action to select a specific document or file to open.	Declare `READ_EXTERNAL_STORAGE` permission

Best Practices

Action	Do ✓	Don't ✗
Identify the device	Use `InstanceID` library or system functions like `randomUUID()`	Access the IMEI of the device
Implement an additional or advanced feature	Make permission optional using `android:required` tag	

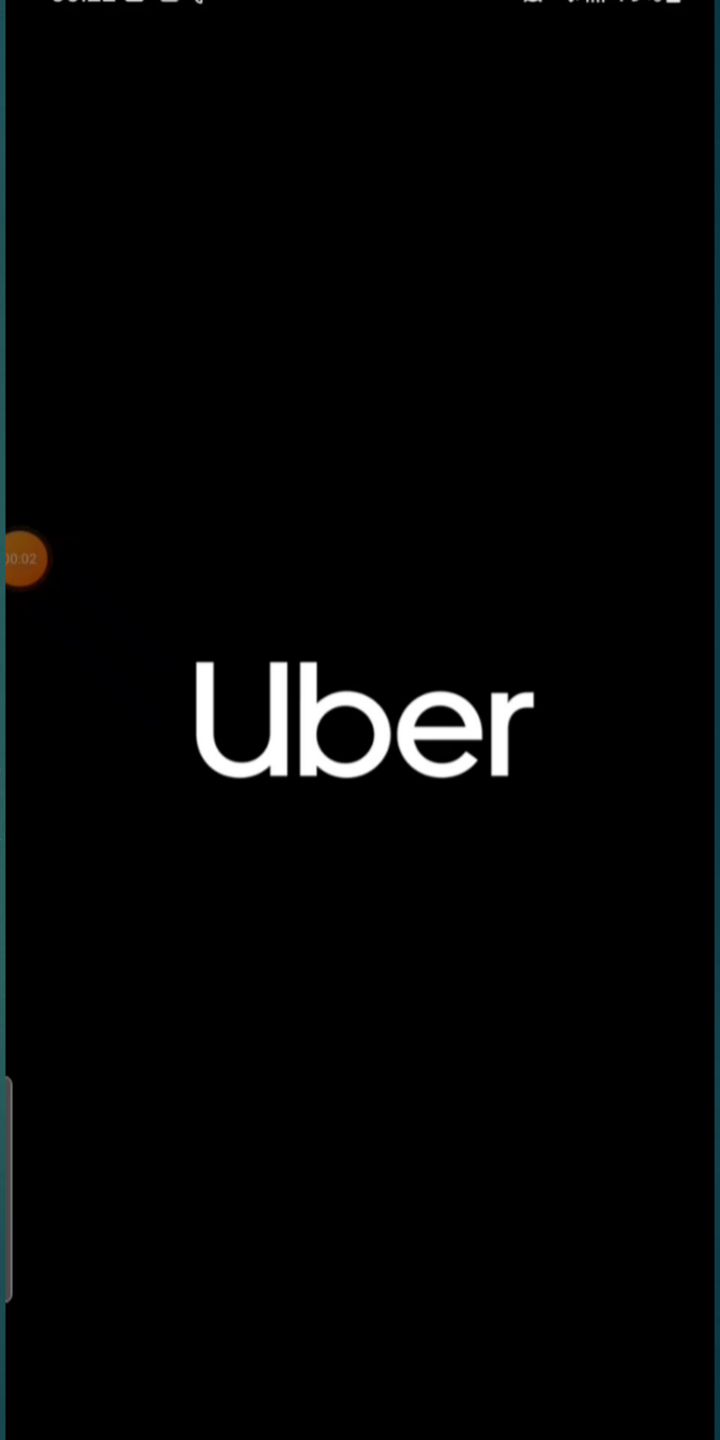
Instagram example

✗



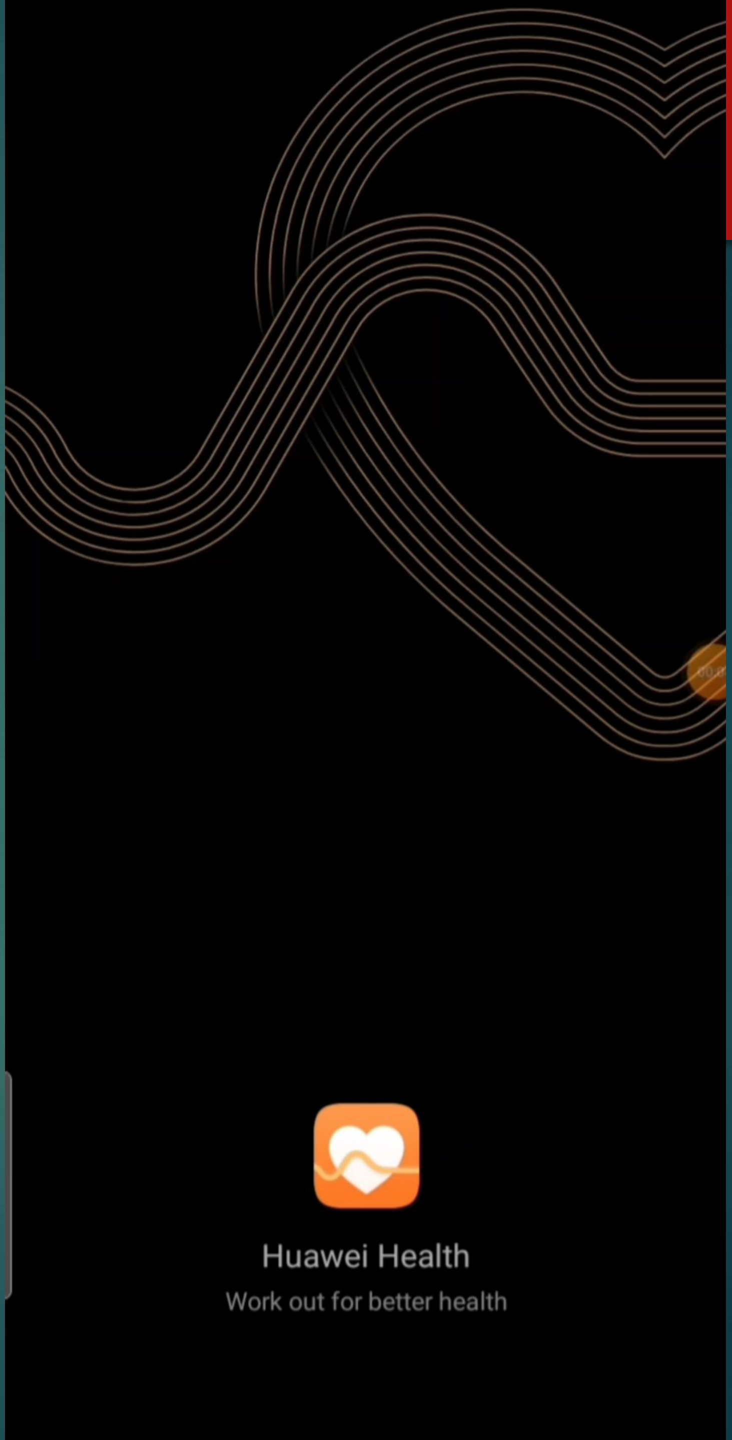
Uber example

✓



Best Practices

- ✔ **Associate runtime permissions with specific actions**
  - In case of permissions that substantially affect the privacy or system, wait for the event that requires them to be asked.
  - For example: Record audio, Access precise location, access camera.



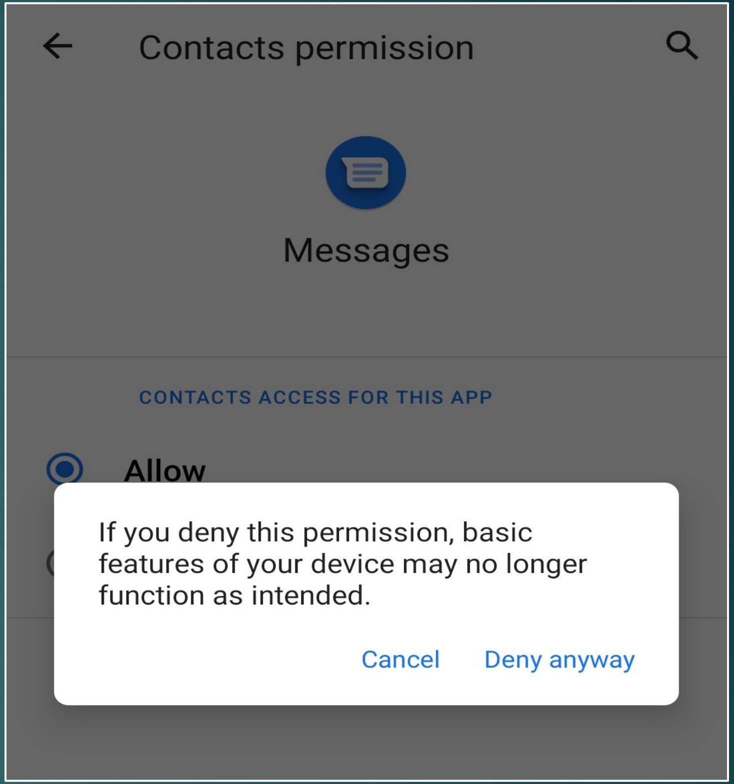
Best Practices

- ✔ **Consider your app's dependencies**
  - Select libraries or SDKs that use permissions as fewer as possible to perform a task

Best Practices

✔ **Be transparent:**

- Explain why your app needs the permission
- If ContextCompat.checkSelfPermission() method returns PERMISSION_DENIED, call shouldShowRequestPermissionRationale(). If this method returns true, show an educational UI to the user.


Contacts permission

Messages

CONTACTS ACCESS FOR THIS APP

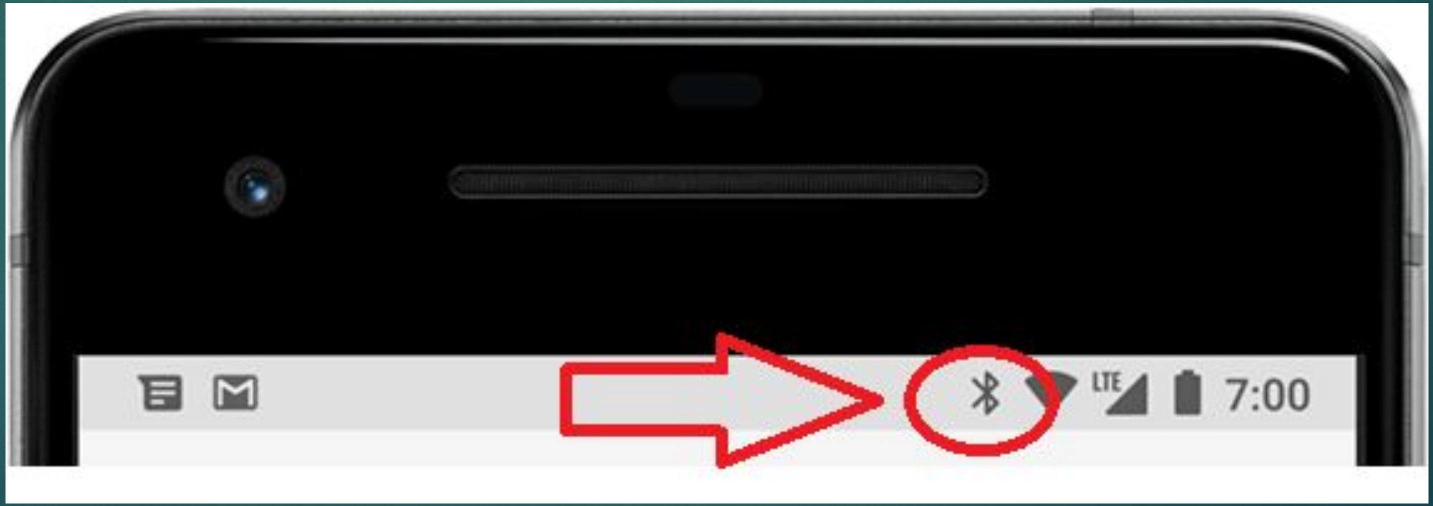
Allow

If you deny this permission, basic features of your device may no longer function as intended.

Dialog after CONTACTS permission is DENIED in Messages app

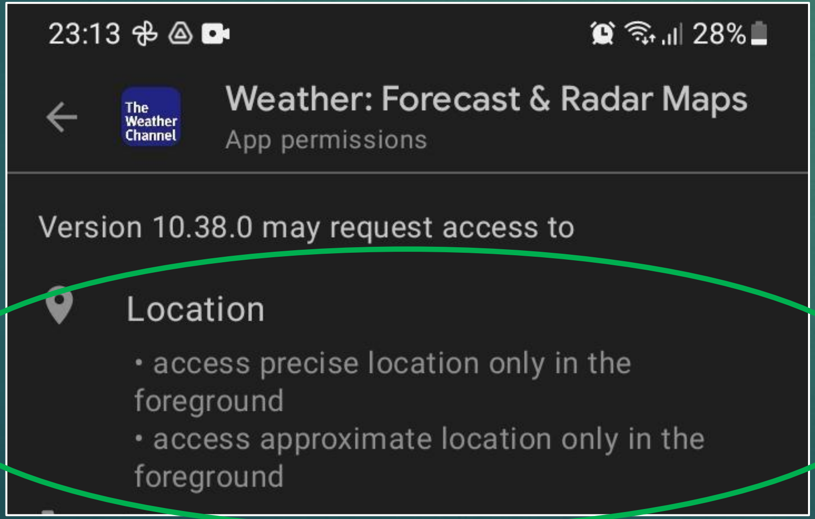
Best Practices

- ✔ **Make system accesses explicit**
  - Indicate on the screen while sensitive data or hardware, such as the camera or microphone, is being accessed.



Best Practices

- ✔ **Limit background access to location**
  - If location access is required only when user is directly interacting with app, restrict background access to location to the apps.

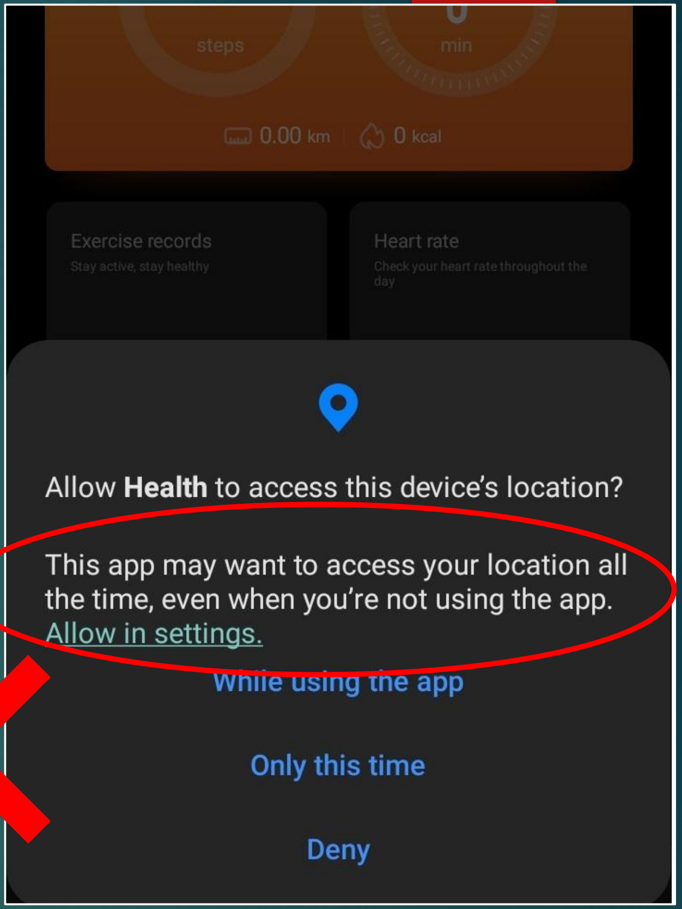


23:13 28%

Weather: Forecast & Radar Maps
App permissions

Version 10.38.0 may request access to

Location
• access precise location only in the foreground
• access approximate location only in the foreground



steps min
0.00 km 0 kcal
Exercise records
Stay active, stay healthy
Heart rate
Check your heart rate throughout the day

Allow **Health** to access this device's location?

This app may want to access your location all the time, even when you're not using the app. Allow in settings.

While using the app

Only this time

Deny

Best Practices

- ✔ **Test for both permissions models**

- Prior to Android 6.0, the apps declared all the permissions in the manifest file. Runtime permissions were introduced later.

- Tips:
  1. Identify current permissions and the related code paths
  2. Test user flows across permission-protected services and data.
  3. Test with various combinations of granted or revoked permissions and make sure the app can handle all permission configurations gracefully.

Sources

- https://developer.android.com/
- https://lifehacker.com/
- https://www.androidcentral.com/