IT332: Mobile Application Development

Lecture # 08 : Android Permissions - I

Muhammad Imran

Outline

- Preface
- Frequently-Asked Questions About Permissions
- Characteristics of Permissions
- Permission Group
- Alternatives to Declaring Permissions
- Best Practices
- Workflow for Using Permissions
- Permission Enforcement with Custom App Permission

Preface

- In the late 1990's, a wave of viruses spread through the Internet, delivered via email, using contact information scrapped from Microsoft Outlook.
- A virus emailed copies of itself to each of the Outlook contacts that had an email address.
- This was possible because Outlook did not protect data from programs using the Outlook API
- Nowadays, many applications secure data by requiring explicit rights grant for other programs to access
- Those rights could be granted on a case-by-case basis or all at once at install time.

Preface

- The purpose of a permission is to protect the privacy of an Android user.
- Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet).
- Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request.

Frequently-Asked Questions About Permissions

- Since Android M 6.0, permission system has changed a fair bit.
- Let's answer some common questions about permissions to help get us started.

What Is a Permission?

- A permission is a way for Android (or, sometimes, a third-party app) to require an app developer to notify the user about something that the app will do that might raise concerns with the user.
- Only if an app holds a certain permission, can the app do certain things that are defended by that permission.

What Is a Permission?

- Android builds upon system security features by allowing users to grant or deny different app permissions.
- These permissions protect access to the following:
  - Restricted data, such as system state and a user's contact information.
  - Restricted actions, such as connecting to a paired device and recording audio.

What Is a Permission?

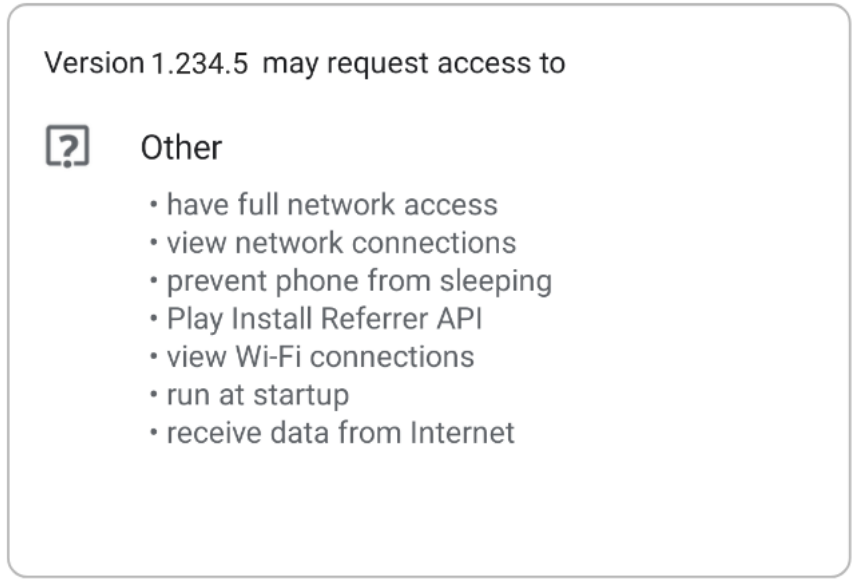
- In order for your app to access restricted data and perform restricted actions, you must declare the appropriate permissions.
- Some permission, known as install-time permissions, are automatically granted when your app is installed.
- Other permissions, known as runtime permissions, require your app to go a step further and request the permission at runtime.

Types of Permissions

- Android categorizes permissions into different types
  - Install-time permissions,
  - Runtime permissions, and
  - Special permissions.
- Each permission's type indicates the scope of restricted data that your app can access, and the scope of restricted actions that your app can perform, when the system grants your app that permission.

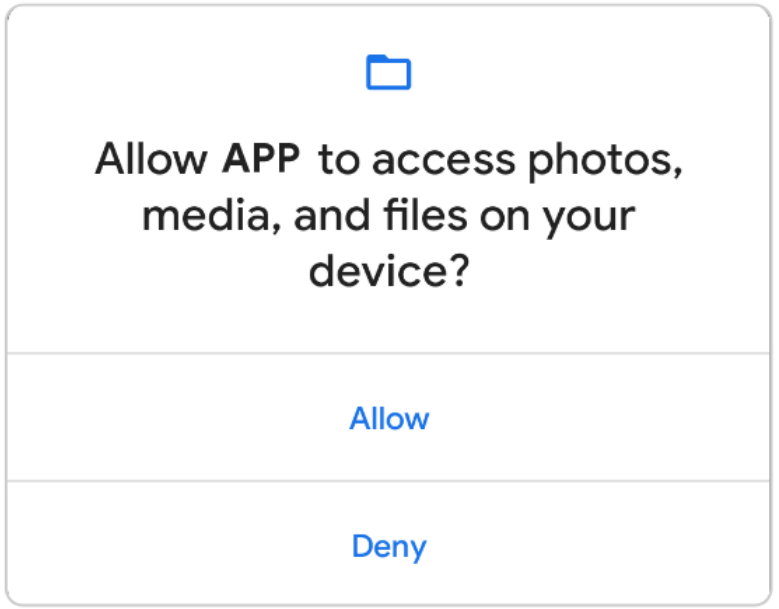
Install-time Permissions

- Install-time permissions give your app limited access to restricted data, and they allow your app to perform restricted actions that minimally affect the system or other apps.
- When you declare install-time permissions in your app, the system automatically grants your app the permissions when the user installs your app.
- Android includes several sub-types of install-time permissions, including normal permissions and signature permissions.



Runtime Permissions

- Runtime permissions, also known as dangerous permissions, give your app additional access to restricted data, and they allow your app to perform restricted actions that more substantially affect the system and other apps.
- Therefore, you need to request runtime permissions in your app before you can access the restricted data or perform restricted actions.



Special Permissions

- Special permissions correspond to particular app operations.
- Only the platform and Original Equipment Manufacturers (OEMs) can define special permissions.
- Additionally, the platform and OEMs usually define special permissions when they want to protect access to particularly powerful actions, such as drawing over other apps.

When Will I Need a Permission?

- Most permissions that you will deal with, come from Android itself.
- If you are trying out some code and you crash with a `SecurityException` the description of the exception may tell you that you need to hold a certain permission— that means you need to add the corresponding <uses-permission> element to your manifest.
- Third-party code, including Google's own Play Services SDK, may define their own custom permissions.
- Ideally, you find out that you need to request a permission through documentation, and otherwise you find out through crashing during testing.

What Are Some Common Permissions, and What Do They Defend?

- There are dozens upon dozens of permissions in Android. Like:
  - INTERNET, if your application wishes to access the Internet through any means from your own process, using anything from raw Java sockets through the WebView widget
  - WRITE_EXTERNAL_STORAGE, for writing data to external storage
  - ACCESS_COARSE_LOCATION and ACCESS_FINE_LOCATION, for determining where the device is
  - READ_CONTACTS, to get at personally-identifying information of arbitrary contacts that the user has in their Contacts app
- Full name of the permission has `android.permission.` as a prefix (e.g.,`android.permission.INTERNET`)

How Do I Request a Permission?

- Put a `<uses-permission>` element in your manifest, with an `android:name` attribute identifying the permission that you are interested in.

```
<?xml version="1.0"?>
<manifest package="com.commonsware.android.fileseditor"
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:versionCode="1"
  android:versionName="1.0">

  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

- This is sufficient for most permissions and most devices.
- Permissions considered to be dangerous need special attention on Android 6.0+

When Is the User Informed About These Permissions?

- It depends on the permission, the version of Android the user is using and from where the user is installing the app

i. Installing Through SDK Tools

- Apps installed using Android Studio will not be prompted for permissions.
- Android 6.0+ and dangerous permissions change this up a bit

ii. Installing from the Play Store, Android 5.1 and Older

- The user will be presented with a roster of permission groups that contain permissions that you are requesting and that are considered to be dangerous.

When Is the User Informed About These Permissions?

- It depends on the permission, the version of Android the user is using and from where the user is installing the app

iii. Installing from the Play Store, Android 6.0+

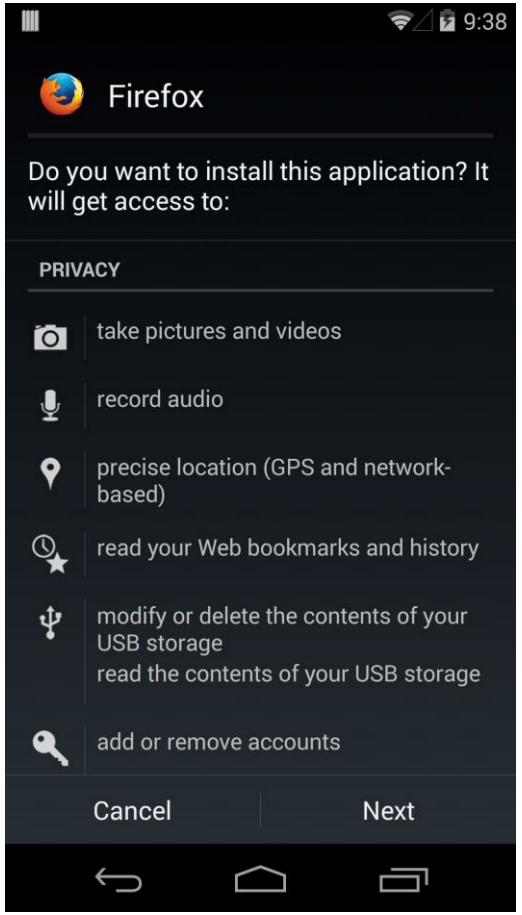
- Depends upon the value of targetSdkVersion for your app.
- If your targetSdkVersion is 22 or lower, the user sees the list of permission groups which contain permissions that you are requesting and that are considered to be dangerous.
- If your targetSdkVersion is 23 or higher, the user is not prompted about permissions at install time. Instead these prompts will occur when the user runs your app and when you ask the user for the permissions

When Is the User Informed About These Permissions?

- It depends on the permission, the version of Android the user is using and from where the user is installing the app

iv. Installing by Other Means, Android 5.1 and Older

- If you install an app by any means (e.g., downloading from a Web site), you will be prompted with a list of all requested permissions
- This prompt will not appear until you have begun the installation process.
- Before then, the device cannot examine the manifest inside the APK file to find the permissions.

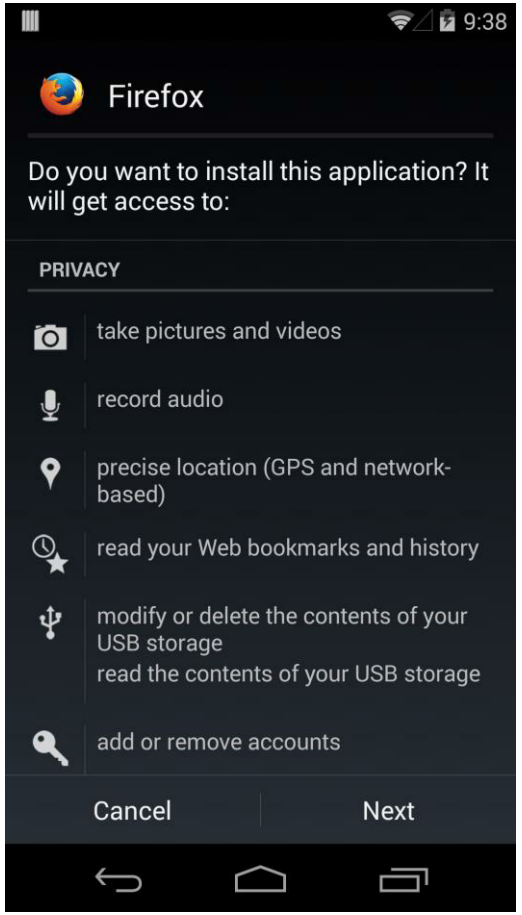


When Is the User Informed About These Permissions?

- It depends on the permission, the version of Android the user is using and from where the user is installing the app

v. Installing by Other Means, Android 6.0+

- You will not be prompted about any permissions at install time



Characteristics of Permissions

- Several bits of information make up a permission, and some of those affect app developers or users.
  - Name
    - android:name attribute of the <uses-permission> identify a permission
    - Android framework-defined permissions will begin with android.permission.
    - Permissions from libraries or third-party apps will have some other prefix
  - Protection Level
    - The definition of a permission, in the framework defines “protection level”.
    - This describes how the permission itself should be validated.
    - The two common protection levels are normal and dangerous.

Normal Permission

- Normal permissions cover areas where there's very little risk to the user's privacy.
- The system automatically grants the app that permission at install time
- The system doesn't prompt the user to grant normal permissions, and users cannot revoke these permissions.
- The classic example is the `INTERNET` permission.
- Permission to set the time zone is a normal permission.
- Users can see normal permissions, in other places:
  - on the Play Store when clicking on a “Permissions” link
  - in Settings for an app
  - using third-party tools

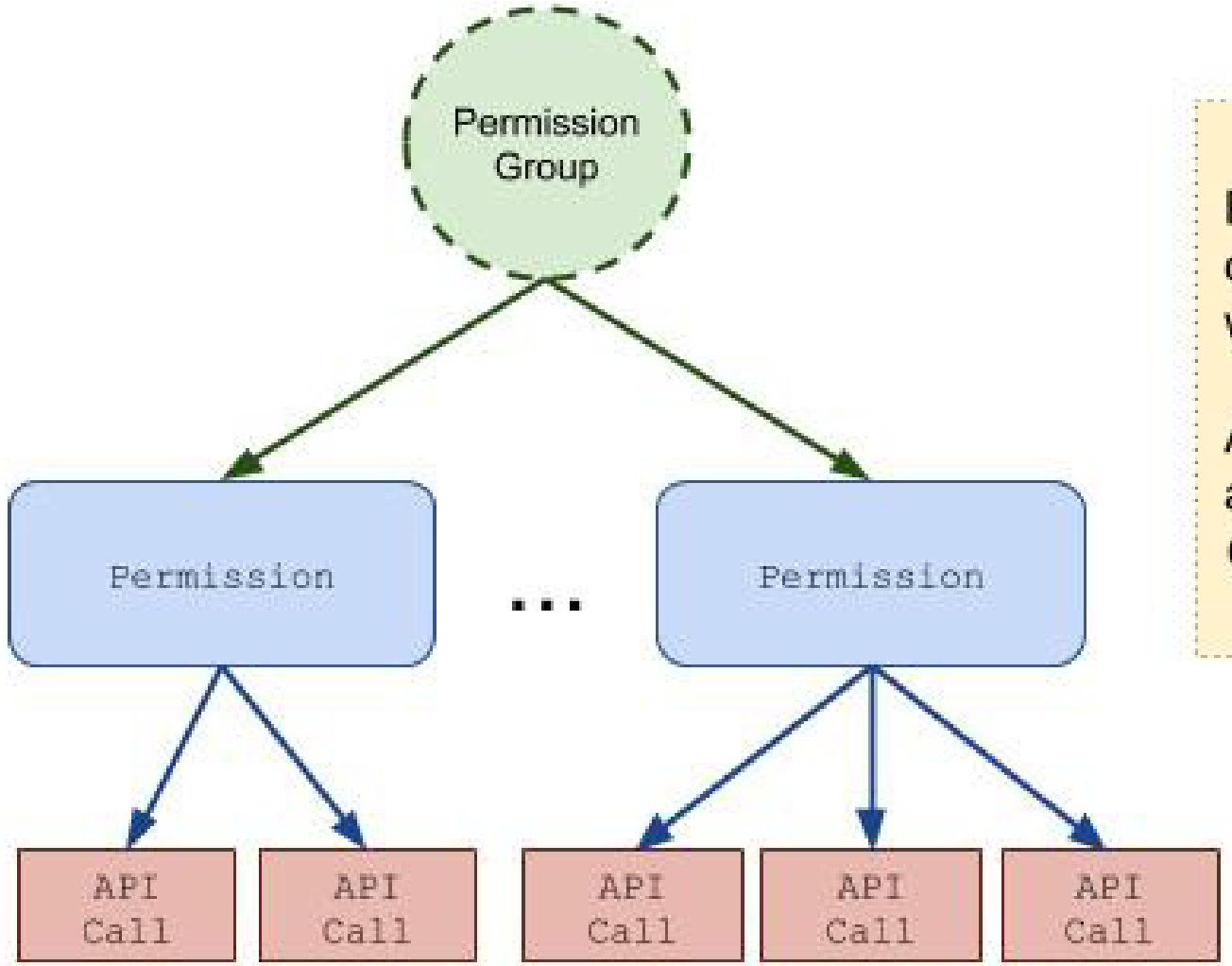
Dangerous Permission

- A dangerous permission is one that the definers of the permission (e.g., Google) wants to ensure that the user is aware of and has agreed to.
- A dangerous permission must be explicitly granted the permission by user at runtime
- Classically, this meant that the user would be prompted for this permission at install time.
- On old versions of Android and the Play Store, dangerous permissions would be listed before normal permissions.
- With Android 6.0+, while dangerous permissions are not displayed at install time (for apps with a targetSdkVersion of 23 or higher),
- They will be displayed to the user while the app is running, before the app tries doing something that requires one of those permissions.

Permission Group

- Permissions are collected into permission groups.
- Permissions are organized into groups related to a device's capabilities or features.
- Under this system, permission requests are handled at the group level and a single permission group corresponds to several permission declarations in the app manifest.
- For example, the SMS group includes both the `READ_SMS` and the `RECEIVE_SMS` declarations
- This enables the user to make more meaningful and informed choices, without being overwhelmed by complex and technical permission requests.

Permission Group



Related API Calls and access to object properties are associated with a specific permission request.

And related permission requests are rolled up into a *Permissions Group*.

Permission Group & System Behavior

- When device is running Android 6.0+ & app's targetSdkVersion is 23 or higher and app requests a dangerous permission
  - If the app doesn't currently have any permissions in the permission group, the system shows the permission request dialog and describe the permission group that the app wants access to.
  - The dialog doesn't describe the specific permission within that group.
  - For example, if an app requests the READ_CONTACTS permission, the system dialog just says the app needs access to the device's contacts. If the user grants approval, the system gives the app just the permission it requested.

Permission Group & System Behavior

- When device is running Android 6.0+ & app's targetSdkVersion is 23 or higher and app requests a dangerous permission
  - If the app has already been granted another dangerous permission in the same permission group, the system immediately grants the permission without any interaction with the user.
  - For example, if an app had previously requested and been granted the READ_CONTACTS permission, and it then requests WRITE_CONTACTS, the system immediately grants that permission without showing the permissions dialog to the user.

Permission Group & System Behavior

- If the device is running Android 5.1 (API level 22) or lower, or the app's targetSdkVersion is 22 or lower
  - The system asks the user to grant the permissions at install time.
  - Once again, the system just tells the user what permission groups the app needs, not the individual permissions.
  - For example, when an app requests READ_CONTACTS the install dialog lists the Contacts group. When the user accepts, only the READ_CONTACTS permission is granted to the app.

Evaluate whether your app needs to declare permissions

- Before you declare permissions in your app, consider whether you need to do so.
- Every time the user tries an app feature that requires a runtime permission, your app has to interrupt the user's work with a permission request.
- If the user doesn't understand why your app requests a particular permission, they could deny the permission or even uninstall your app.
- If another installed app is able to perform the functionality on your app's behalf, you should delegate the task to it using an intent.
- In doing so, you don't need to declare the necessary permissions because the other app declares the permission instead.

Alternatives to Declaring Permissions

Several use cases exist that your app can fulfill without declaring the need for any permissions.

❖Show Nearby Places

- If an app wants user's approximate location to show location-aware information, such as nearby restaurants, you may require a rough estimate of a device's location, to within about 1.25 miles (2 km). In these situations, you can declare the ACCESS_COARSE_LOCATION permission.
- It's better to not declare the permission, and instead ask the user to enter an address or a postal code.
- If you require a more precise estimate of a device's location, it's OK to declare the ACCESS_FINE_LOCATION permission.

Alternatives to Declaring Permissions

Several use cases exist that your app can fulfill without declaring the need for any permissions.

❖Take a Photo

- Users might take pictures in your app, using the pre-installed system camera app.
- In this situation, don't declare the CAMERA permission.
- Instead, invoke the ACTION_IMAGE_CAPTURE intent action.

Alternatives to Declaring Permissions

Several use cases exist that your app can fulfill without declaring the need for any permissions.

- Record a Video
  - Users might record videos in your app, using the pre-installed system camera app.
  - In this situation, don't declare the CAMERA permission.
  - Instead, invoke the ACTION_VIDEO_CAPTURE intent action

Alternatives to Declaring Permissions

Several use cases exist that your app can fulfill without declaring the need for any permissions.

❖Open Another App's Media or Documents

- Your app might show content that another app created, such as photos, videos, or text files.
- In this situation, don't declare the READ_EXTERNAL_STORAGE permission.
- Also, don't directly access shared or external storage devices using methods like getExternalStorageDirectory(), which are deprecated as of Android 10 (API level 29).
- Instead, use the device's media store to open media files, and the Storage Access Framework to open documents and other files.

Alternatives to Declaring Permissions

Several use cases exist that your app can fulfill without declaring the need for any permissions.

❖Filter Phone Calls

- To minimize unnecessary interruptions for the user, your app might filter phone calls for spam.
- To support this functionality, don't declare the READ_PHONE_STATE permission.
- Instead, use the CallScreeningService API.

Best Practices

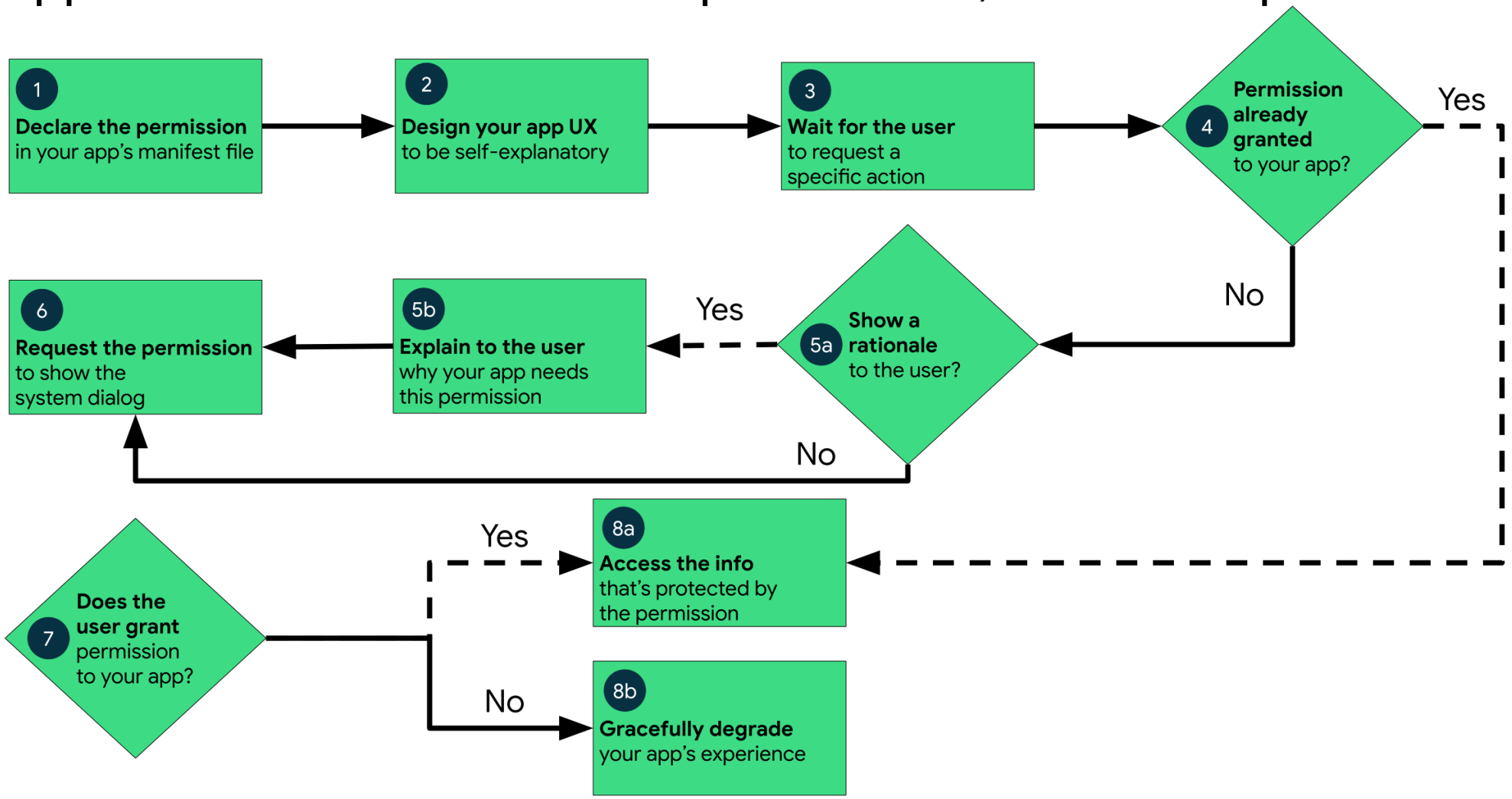
- To use permissions effectively in your app, follow these best practices:
- Request a minimal number of permissions in your app.
  - When the user requests a particular action in your app, your app should request only the permissions that it needs to complete that action.
- Associate runtime permissions with specific tasks and actions in your app.
  - Request permissions as late into the flow of your app's use cases as possible.
- For example, if your app allows users to send audio messages to others, wait until the user has navigated to the messaging screen and has pressed the Send audio message button. After the user presses the button, your app can then request access to the microphone.

Workflow for Using Permissions

- Before you declare permissions in your app, think for alternatives to declaring permissions
  - many use cases, such as taking photos, pausing media playback, and displaying relevant ads, without needing to declare any permissions.

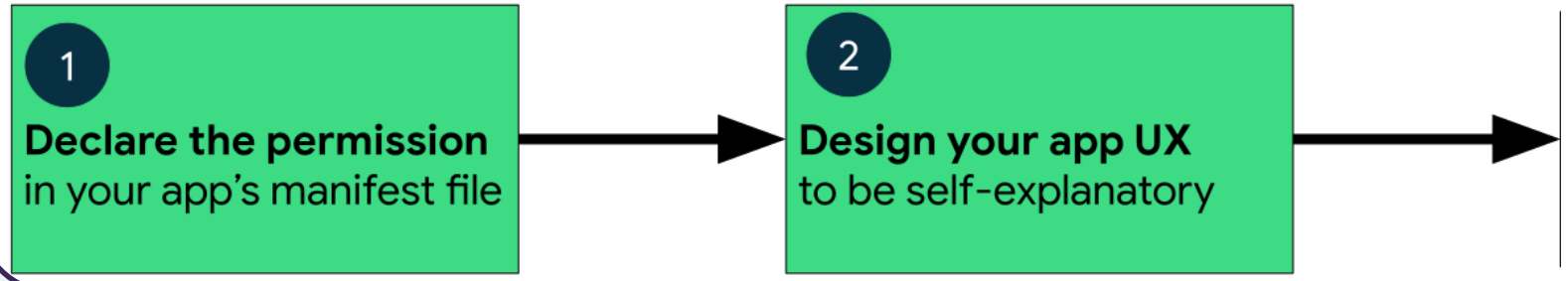
Workflow for Using Permissions

- If your app needs to declare and use permissions, follow this process:



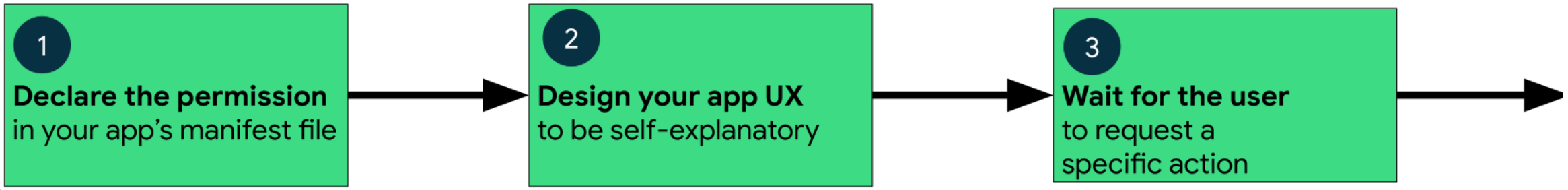
Workflow for Using Permissions

1. Declare the Permission
   - In your app's manifest file, declare the permissions that your app might need to request.
   - By declaring install-time permissions, your app can access the restricted data and perform the restricted actions associated with these install-time permissions after the user installs your app.
   - If your app declares any runtime permissions, continue to the next step.



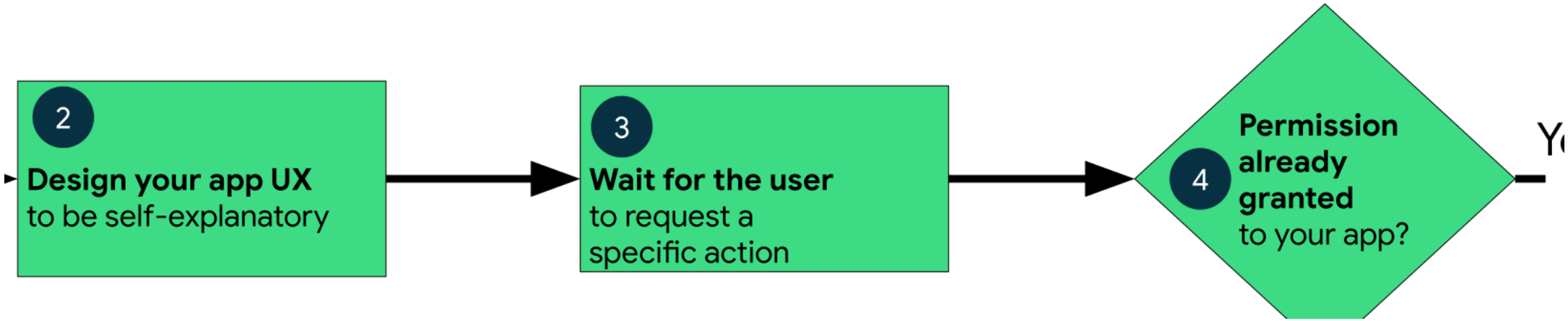
Workflow for Using Permissions

2. Design your app's UX
   - Design your app's UX so that specific actions in your app are associated with specific runtime permissions.
   - Users should know which actions might require them to grant permission for your app to access private user data.



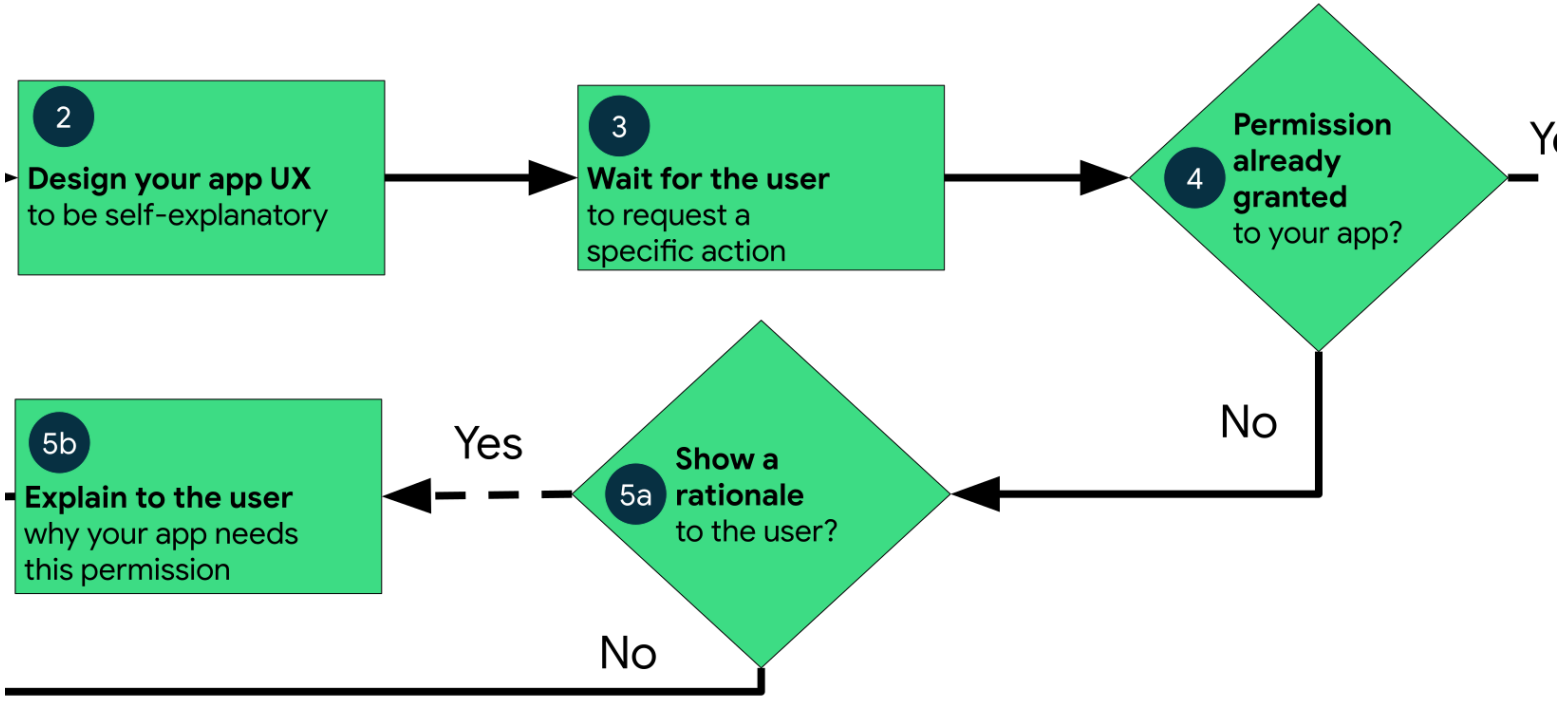
Workflow for Using Permissions

3. Wait for the user
   - Wait for the user to invoke the task or action in your app that requires access to specific private user data.
   - At that time, your app can request the runtime permission that's required for accessing that data.



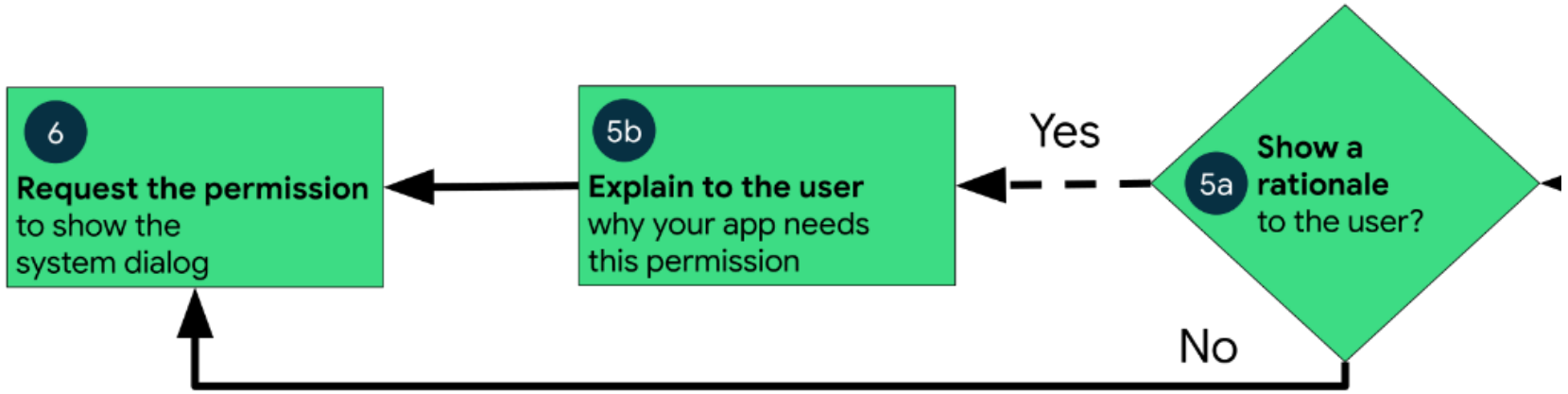
Workflow for Using Permissions

4. Permission already granted?
   - Check whether the user has already granted the runtime permission that your app requires.
   - If so, your app can access the private user data. If not, continue to the next step.



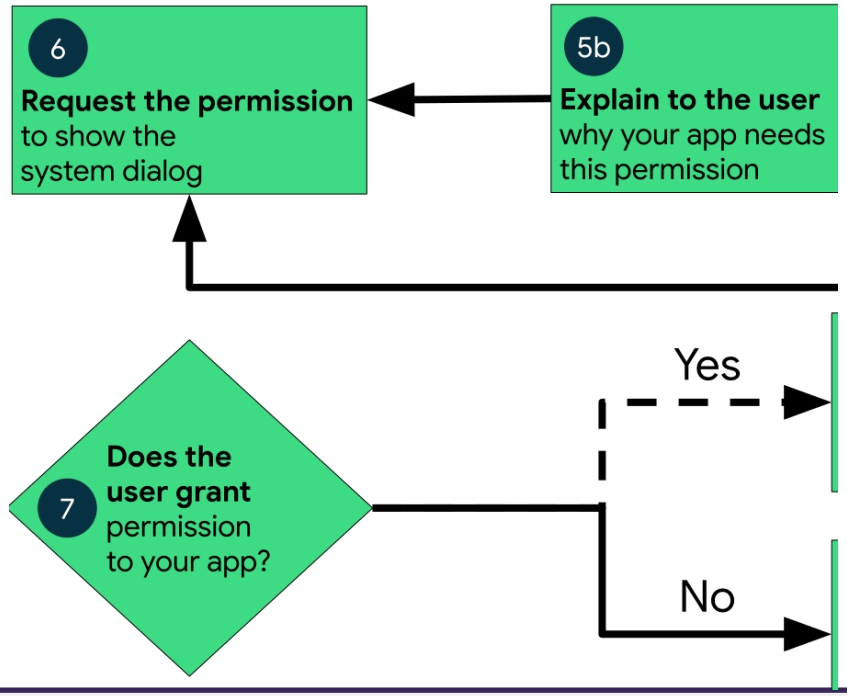
Workflow for Using Permissions

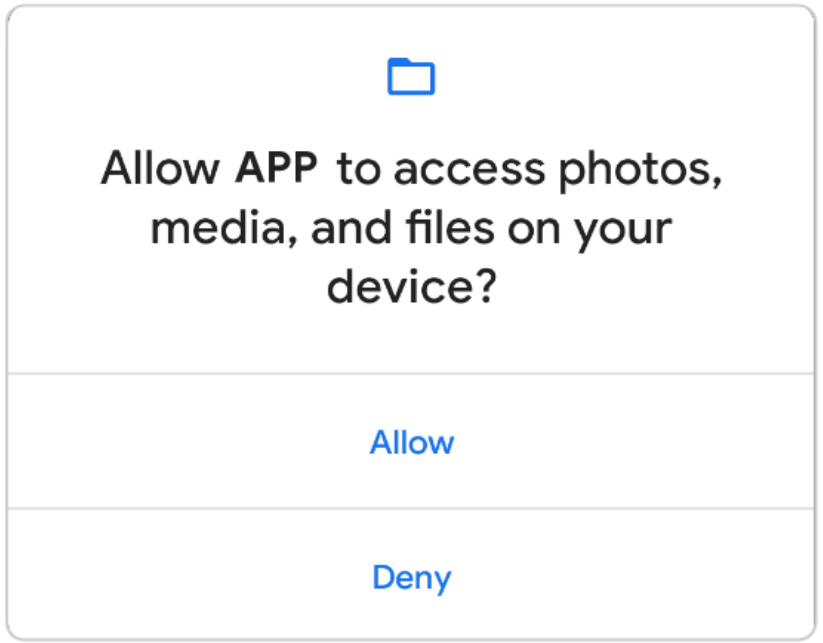
5. Check whether your app should show a rationale?
   - Rationale explains why your app needs a particular runtime permission.
   - If app shouldn't show a rationale, continue to the next step directly
   - If app should show a rationale, present it to the user in a UI element.
   - After the user acknowledges the rationale, continue to the next step.



Workflow for Using Permissions

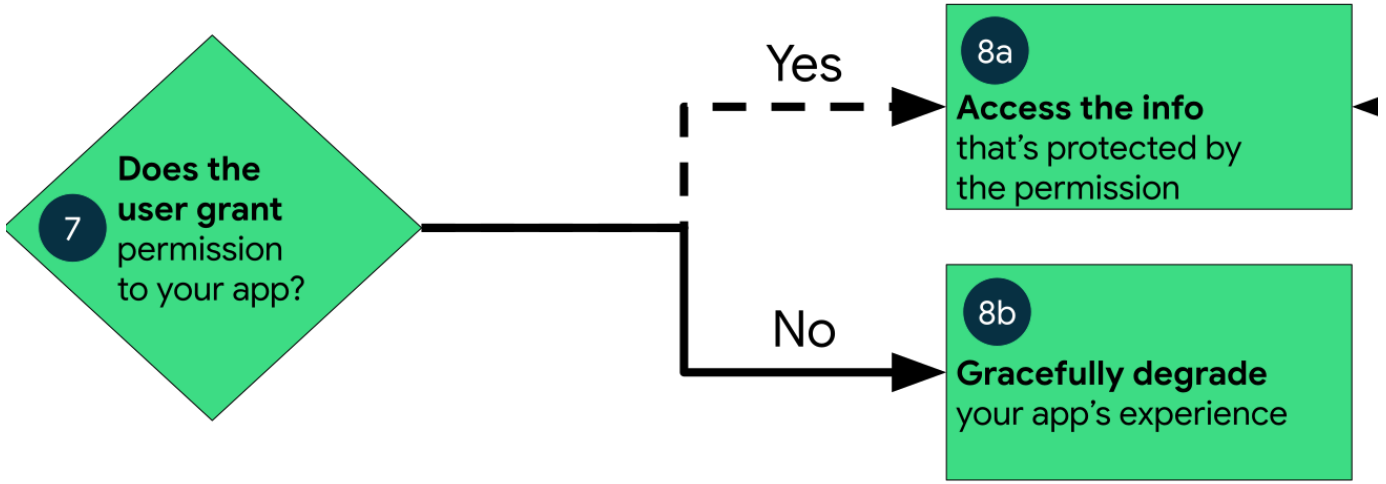
6. Request the runtime permission
   - Request the runtime permission that your app requires in order to access the private user data.
   - The system displays a runtime permission prompt





Workflow for Using Permissions

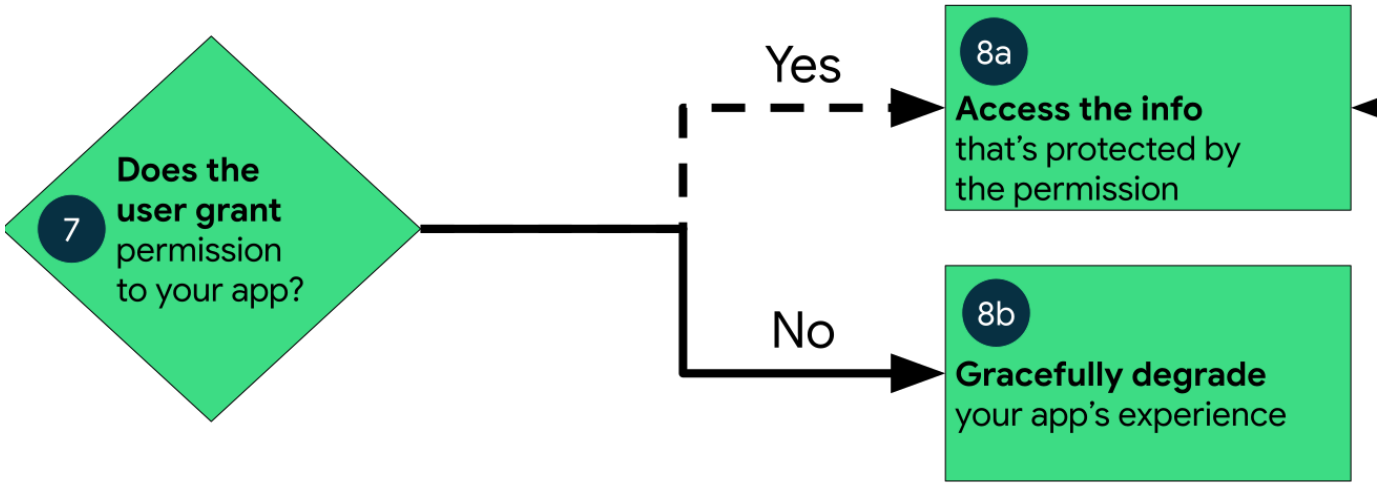
7. Does the user Grant?
   - Check the user's response, whether they chose to grant or deny the runtime permission



Workflow for Using Permissions

8. Handle Response

- If the user granted the permission to your app, you can access the private user data.
- If the user denied the permission instead, gracefully degrade your app experience so that it provides functionality to the user, even without the information that's protected by that permission.



Define a Custom App Permission

- By defining custom permissions, an app can share its resources and capabilities with other apps
- Apps can expose their functionality to other apps by defining permissions which those other apps can request.

Permission Enforcement

- Permissions aren't only for requesting system functionality.
- Services provided by apps can enforce custom permissions to restrict who can use them.
- Permissions applied using the android:permission attribute to the <activity> tag in the manifest restrict who can start that Activity.
- The permission is checked during startActivity() and startActivityForResult().
- If the caller doesn't have the required permission, then SecurityException is thrown from the call.
- Service permission enforcement is also possible
- Broadcast permission enforcement can also be implemented

Define a Custom App Permission

- To enforce your own permissions, you must first declare them in your AndroidManifest.xml using one or more <permission> elements.

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

    <permission
      android:name="com.example.myapp.permission.DEADLY_ACTIVITY"
      android:label="@string/permlab_deadlyActivity"
      android:description="@string/permdesc_deadlyActivity"
      android:permissionGroup="android.permission-group.COST_MONEY"
      android:protectionLevel="dangerous" />
    ...
</manifest>
```

Define a Custom App Permission

- The protectionLevel attribute is required, telling the system how the user is to be informed of apps requiring the permission
- The android:permissionGroup attribute is optional, and only used to help the system display permissions to the user.
  - In most cases you should set this to a standard system group (listed in android.Manifest.permission_group) (you can define a group yourself)
- You need to supply both a label and description for the permission.
  - The label should be short, to describe functionality the permission is protecting
  - The description should be a couple of sentences describing what the permission allows a holder to do.

Define a Custom App Permission

- Here is an example of a label and description for the CALL_PHONE permission:

```
<string name="permlab_callPhone">directly call phone numbers</string>
<string name="permdesc_callPhone">Allows the app to call
    phone numbers without your intervention. Malicious apps may
    cause unexpected calls on your phone bill. Note that this does not
    allow the app to call emergency numbers.</string>
```

Recommended Readings

- Page # 579 to 587, Chapter: Requesting Permissions from The Busy Coder's Guide to Android Development, Final Version by Mark L. Murphy, 2019

- User Guide: https://developer.android.com/guide/topics/permissions/overview