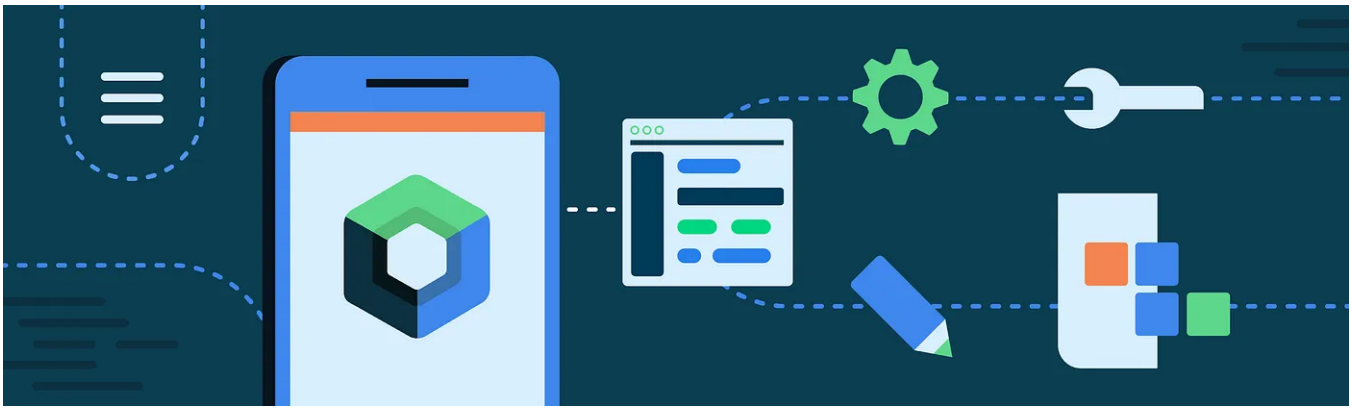


Modern Android Development in 2023

Jorge Luis Castro Medina · Follow
13 min read · Jan 23

Hello everyone 👋🏻, I'd like to share with you how to build Android apps with the latest trends for the year 2023.

Disclaimer

This is an article from my opinion and professional experience, taking into account different opinions among the Android developer community, also constantly reviewing the guides provided by Google for Android.

I have to make it clear that there are very interesting tools, patterns and architectures that I may not mention but it does not mean that they cannot be other interesting alternatives to develop Android applications.

What is android?

Android is an open-source operating system based on the Linux kernel and developed by Google. It's used in a wide variety of devices, including smartphones, tablets, TVs, and smartwatches.

Currently, Android is the operating system most used in the world for mobile devices; according to a report by **statcounter** with a sample of the last 12 months, Android has a market share of **71.96%.**

Next, I will mention a list of tools, libraries, architectures, guides, and other utilities that I consider important to building modern applications on Android.

Kotlin ❤️

Image by https://talently.tech/

Kotlin is a programming language developed by JetBrains. Recommended by Google who officially announced it in May 2017 (see publication here). It is a modern programming language that has compatibility with Java and can run on the JVM, which has made its adoption in the development of Android applications very fast.

Whether you are new to Android or not, you should consider Kotlin as your first choice, don't swim against the tide 🚤 😎, Google announced this approach at Google I/O 2019. With Kotlin, you will be able to use all the features of a modern language, including the power of Coroutines and the use of modern libraries developed for the Android ecosystem.

Official kotlin documentation here

Jetpack Compose 😍

Image by blogger.googleusercontent.com

> *Jetpack Compose is Android's recommended modern toolkit for building native UI. It simplifies and accelerates UI development on Android.*
>
> *– Jetpack Compose documentation*

Jetpack Compose is part of the Android Jetpack library and uses the Kotlin programming language to easily create a native user interface. Also, it integrates with other Android Jetpack libraries, such as LiveData and ViewModel, to make it easier to build reactive and maintainable Android applications.

Some key features of Jetpack Compose include:

1. Declarative UI.
2. Customizable widgets.
3. Easy integration with existing code.
4. Live preview.
5. Improved performance.

Resources:

- Official documentation.
- Compose to Kotlin Compatibility Map
- Jetpack Compose Roadmap
- Course.

Jetpack Compose documentation

Android Jetpack

Image by Florina Muntenescu on medium

> *Jetpack is a suite of libraries to help developers follow best practices, reduce boilerplate code, and write code that works consistently across Android versions and devices so that developers can focus on the code they care about.*
>
> _ *Android Jetpack documentation*

Some of its most common tools are:

- ViewModel
- Room
- DataStore
- WorkManager
- Navigation
- CameraX
- Compose

Material Design

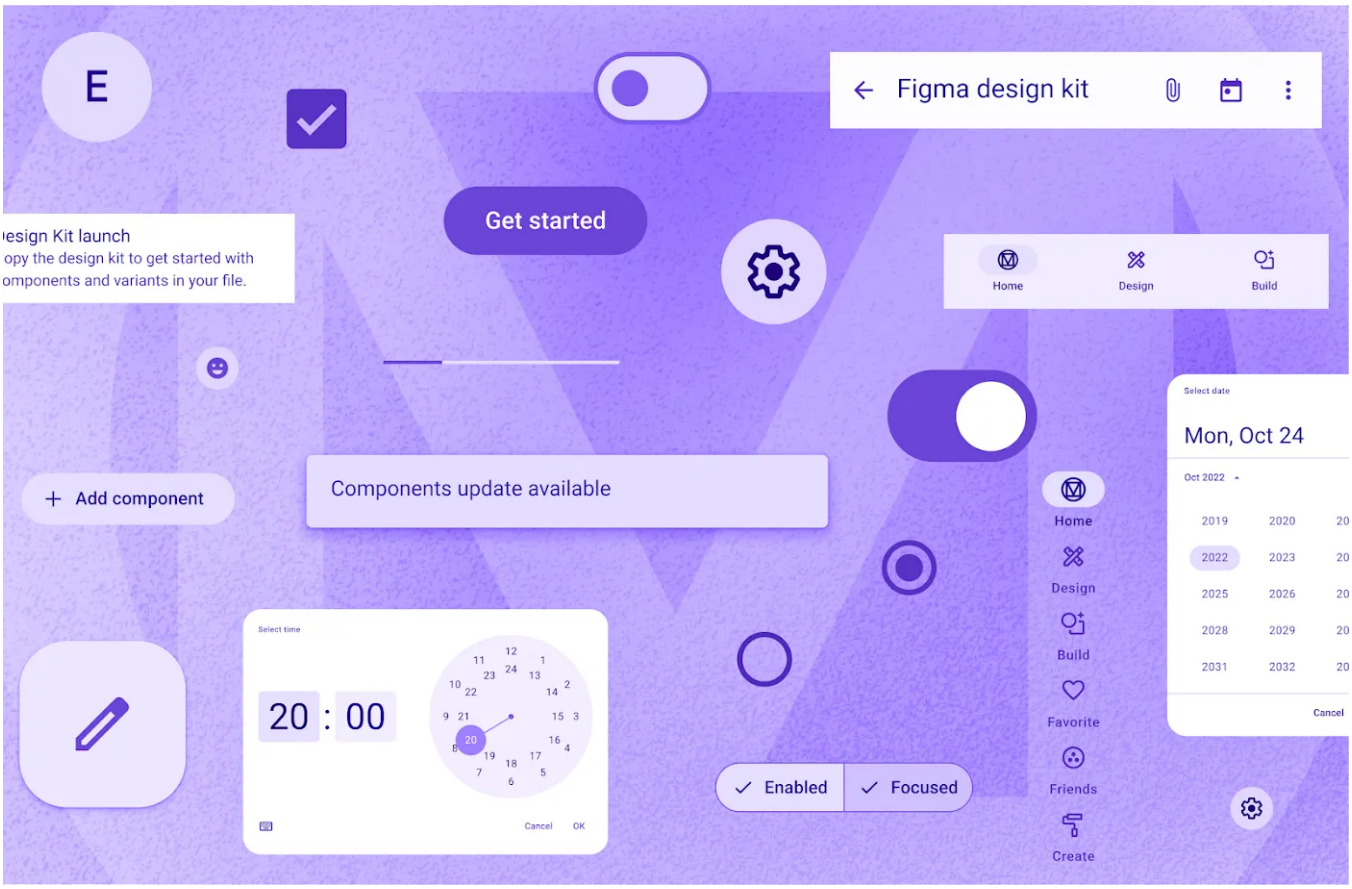


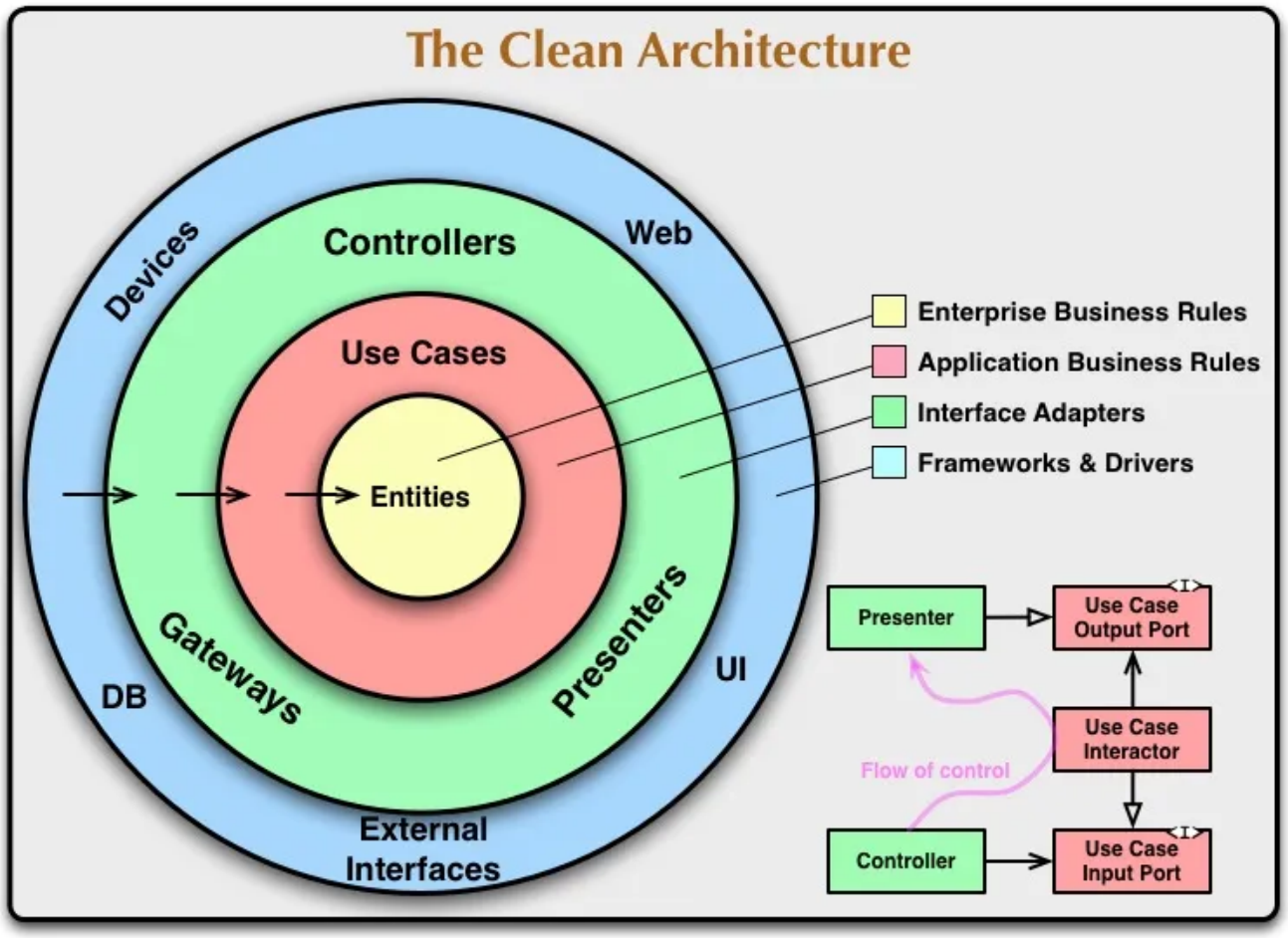
Illustration by Reed Hollett

> *Material Design is an adaptable system of guidelines, components, and tools that support the best practices of user interface design. Backed by open-source code, Material Design streamlines collaboration between designers and developers, and helps teams quickly build beautiful products.*
>
> *_ Material Design Site*

Material Design is backed by designers and developers from Google and will allow us to have a guide to work on our UI/UX for Android, Flutter and Web.

Currently, the last version for Material Design is 3, you can see more here

Clean Architecture



The concept of **"Clean Architecture"** was introduced by Robert C. Martin. It is based on the separation of responsibilities through the division of software into layers.

Characteristics

1. Independent of Frameworks.
2. Testable.
3. Independent of UI.
4. Independent of Database.
5. Independent of any external agency.

The Dependency Rule

The dependency rule is described very well by the author in his piece, The Clean Code Blog

> *The overriding rule that makes this architecture work is* The Dependency Rule. *This rule says that* source code dependencies *can only point* inwards. *Nothing in an inner circle*

> *can know anything at all about something in an outer circle. In particular, the name of something declared in an outer circle must not be mentioned by the code in the an inner circle. That includes, functions, classes. variables, or any other named software entity.*

The Clean Code Blog

Clean Architecture in Android

- **Presentation**: Activities, Fragments, View Models, others view components.
- **Domain**: Use Cases, Entities, Repositories, others domain components.
- **Data**: Repository implementations, Mappers, DTO's, etc.

Architecture Patterns for Presentation Layer

An architecture pattern is a higher-level strategy that aims to help design a software architecture and is characterized by being a solution within a reusable framework for common architectural problems. Architectural patterns are similar to design patterns, but they are larger in scale and address more global issues such as the overall structure of the system, the relationships between components, and the way that data is managed.

Within the Presentation layer, we have some architecture patterns, of which I would like to highlight the following:

- **MVVM**
- **MVI**

I do not want to go into explaining each one because on the internet you find too much information about this. 😅

In addition, you can also see the **guide to app architecture**

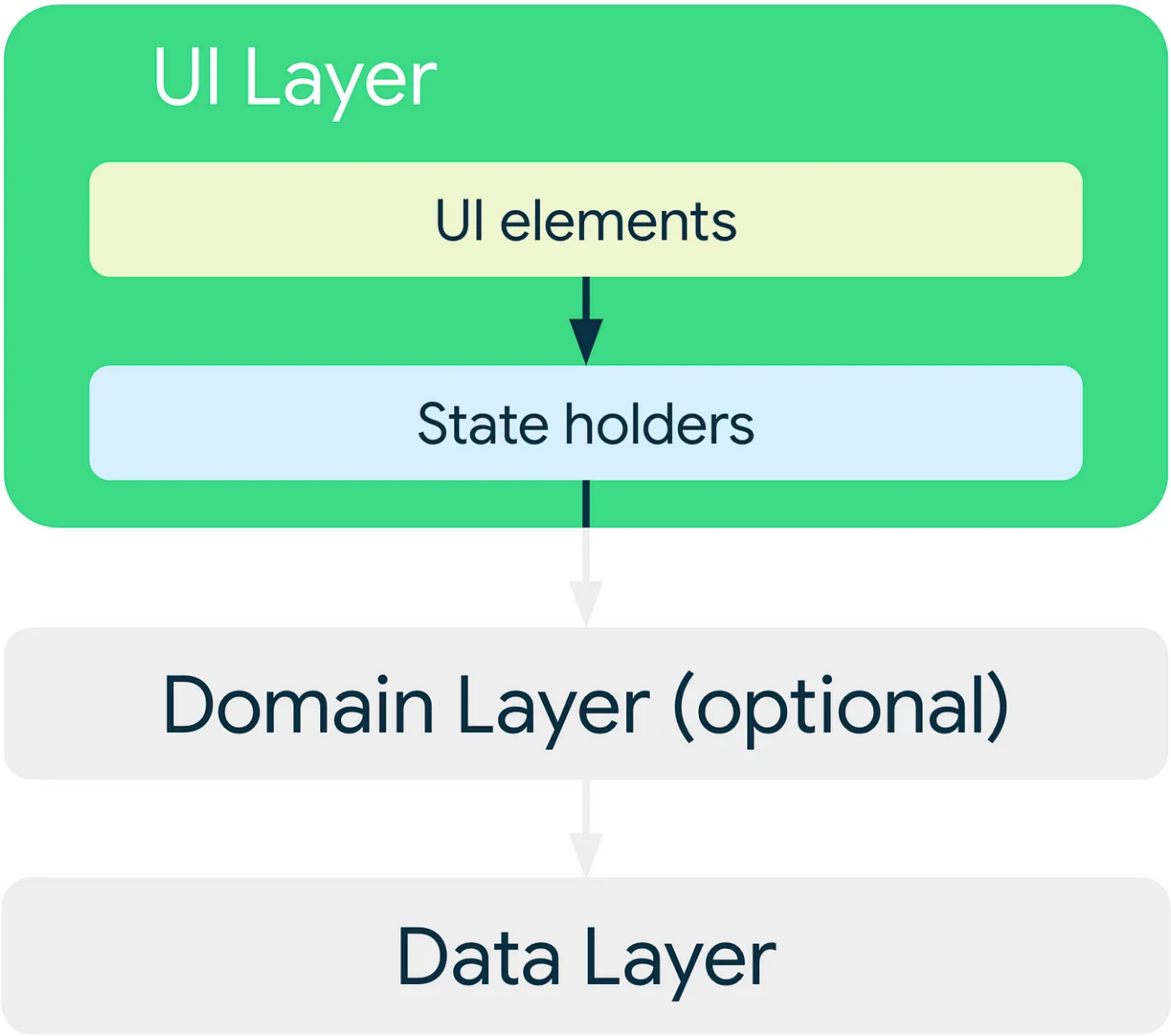


Image by developer.android.com

Dependency Injection

Dependency injection is a software design pattern that allows a client to obtain its dependencies from an external source rather than creating them itself. It is a technique for achieving Inversion of Control (IoC) between objects and their dependencies.

- Hilt ❤️
- Dagger
- Koin

Modularization

Modularization is a software design technique that allows you to divide an application into independent modules, each with its own functionality and responsibility.

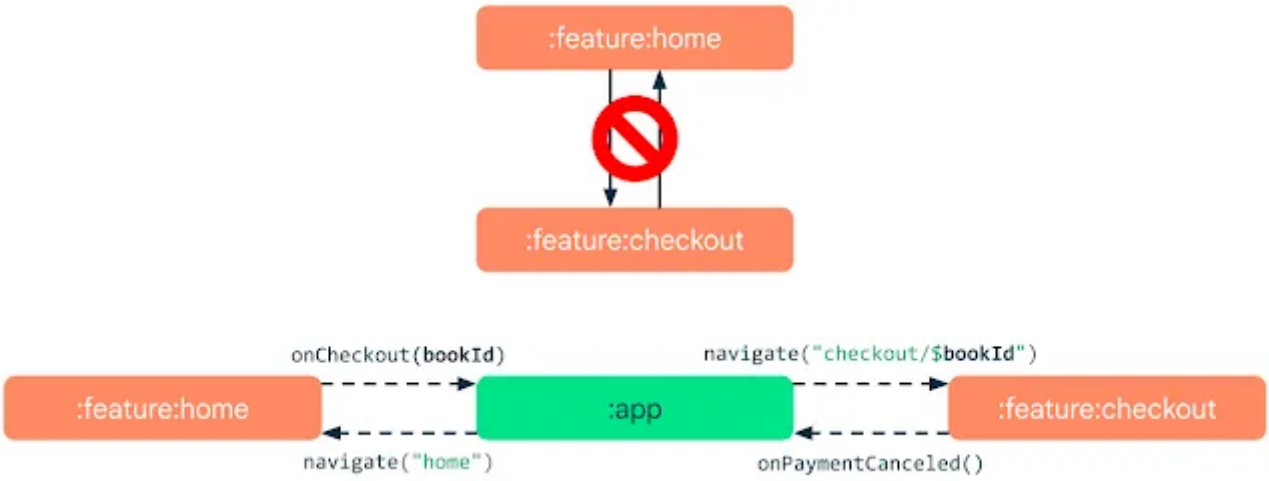


Image by developer.android.com

Benefits of modularization

Reusability: By having independent modules, they can be reused in different parts of the application or even in other applications.

Strict visibility control: Modules enable you to easily control what you expose to other parts of your codebase.

Customizable delivery: Play Feature Delivery uses the advanced capabilities of app bundles, allowing you to deliver certain features of your app conditionally or on demand.

Scalability: By having independent modules, functionalities can be added or removed without affecting other parts of the application.

Ease of maintenance: By dividing the application into independent modules, each with its own functionality and responsibility, it is easier to understand and maintain the code.

Ease of testing: By having independent modules, they can be tested in isolation, which makes it easy to detect and fix errors.

Architecture improvement: Modularizing helps to improve the architecture of the application, allowing a better organization and structure of the code.

Improve collaboration: By having independent modules, developers can work on different parts of the application simultaneously and without interference.

Build Time: Some Gradle functionalities such as incremental build, build cache or parallel build, can leverage modularity to improve build performance.

See more in the official documentation.

Network

- OkHttp
- Retrofit

Serialization

In this section I would like to mention two important tools in my opinion: **Moshi** widely used in conjunction with Retrofit and **Kotlin Serialization**, the bet of the Kotlin team at Jetbrain.

- Moshi
- Kotlin Serialization

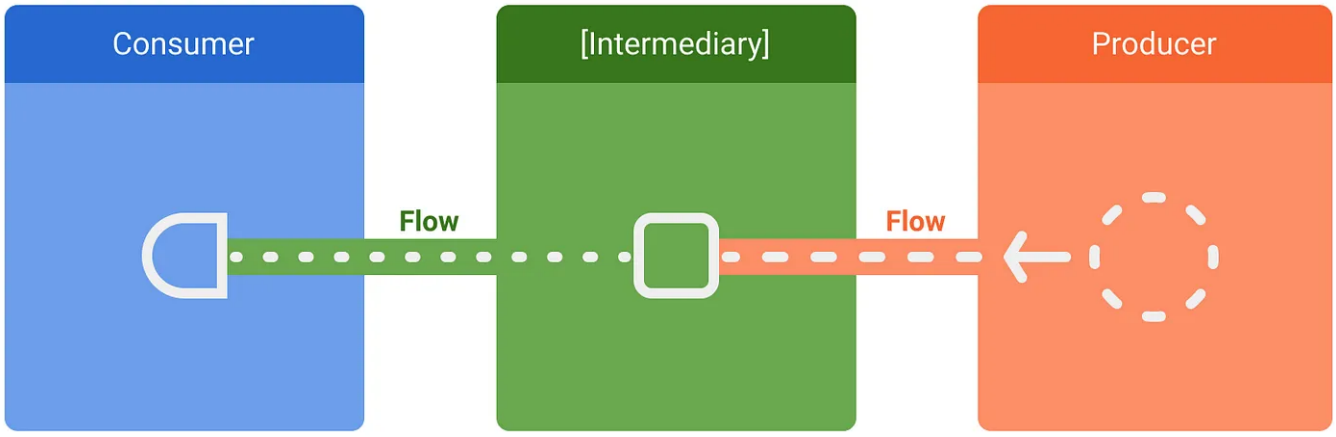
Moshi and **Kotlin Serialization** are two serialization/deserialization libraries for Kotlin and Java that allow you to convert objects to JSON or another serialization format and vice versa. Both provide a user-friendly interface optimized for use in mobile and desktop applications. Moshi primarily focuses on JSON serialization, while Kotlin Serialization has support for various serialization formats, including JSON.

Image Loading

> *To load an image from the internet, there are several third-party libraries available to help you handle the process. Image loading libraries do a lot of the heavy lifting for you; they handle both caching (so you don't download the image multiple times) and networking logic to download the image and display it on screen.*
>
> *_ Official Android Documentation*

- Coil
- Glide

Reactivity / Thread Management



When we talk about reactive programming and asynchronous processes, our first option is `Kotlin Coroutines` ; thanks to the `Suspension Functions` and `Flow` we can cover all these needs. However, I believe that in this section it is worth highlighting the importance of `RxJava` even within the development of Android applications. For those of us who have been working with Android for a couple of years, we know that RxJava is a very powerful tool with a very large set of functions for working with data streams. I still consider RxJava to be an interesting alternative to consider today.

- Kotlin Coroutines: [suspend functions / Flow Api] (Nice to have ❤️)
- RxJava

Local Store

An important point when building mobile applications is having the ability to persist data locally, such as some session data or cache data, among others. It is important to choose the right storage option based on the needs of your application. We could store unstructured data like key-value or structured data like a database. Keep in

mind that this point does not mention all the types of local storage that we have available (such as file storage), only the tools that allow us to save data.

Feature	SharedPreferences	Preferences DataStore	Proto DataStore
Async API	✅ (only for reading changed values, via listener)	✅ (via `Flow`)	✅ (via `Flow`)
Synchronous API	✅ (but not safe to call on UI thread)	❌	❌
Safe to call on UI thread	❌*	✅ (work is moved to `Dispatchers.IO` under the hood)	✅ (work is moved to `Dispatchers.IO` under the hood)
Can signal errors	❌	✅	✅
Safe from runtime exceptions	❌**	✅	✅
Has a transactional API with strong consistency guarantees	❌	✅	✅
Handles data migration	❌	✅ (from SharedPreferences)	✅ (from SharedPreferences)
Type safety	❌	❌	✅ with Protocol Buffers

Suggestions:

- ~~SharedPreferences~~
- DataStore
- EncryptedSharedPreferences

Testing

- JUnit 5

- Mockk
- Espresso
- Robolectric

R8 optimizations

R8 is the default compiler that converts your project's Java bytecode into the DEX format that runs on the Android platform. It is a tool that helps us to obfuscate and reduce the code of our application by shortening the names of the classes and their properties, eliminating unused code and resources within the project. To see more, check the Android documentation about Shrink, obfuscate, and optimize your app.

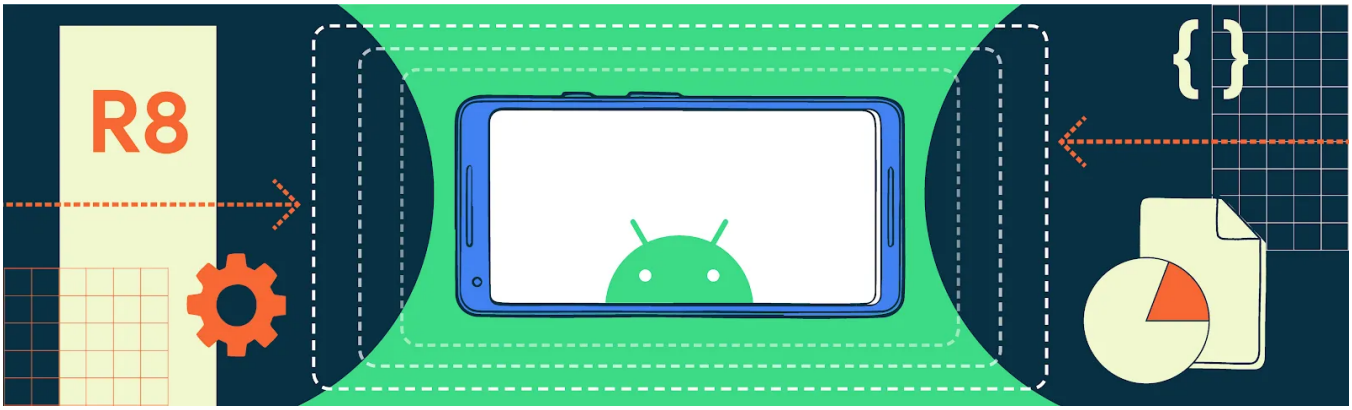


Image by androidtopics.dipien.com

- Code shrinking
- Resource shrinking
- Obfuscation
- Optimization

Play Feature Delivery

> *Google Play's app serving model, called Dynamic Delivery, uses Android App Bundles to generate and serve optimized APKs for each user's device configuration, so users download only the code and resources they need to run your app.*
>
> *_ Android Documentation*

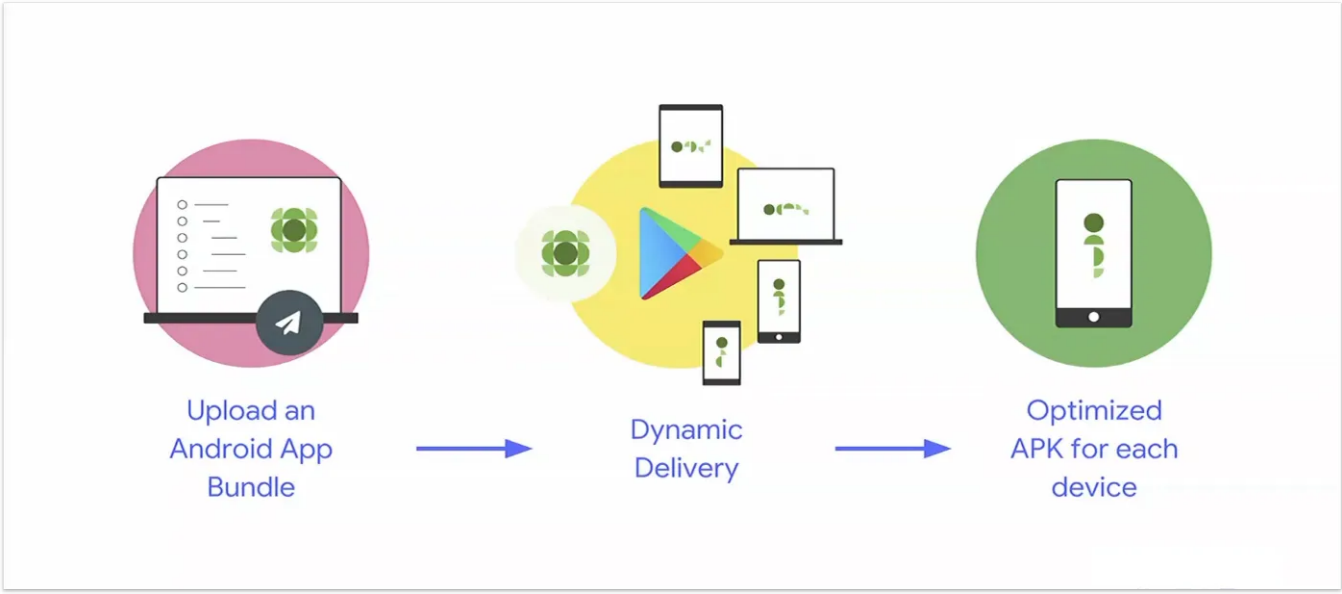


Image by miro.medium.com

Adaptive layouts

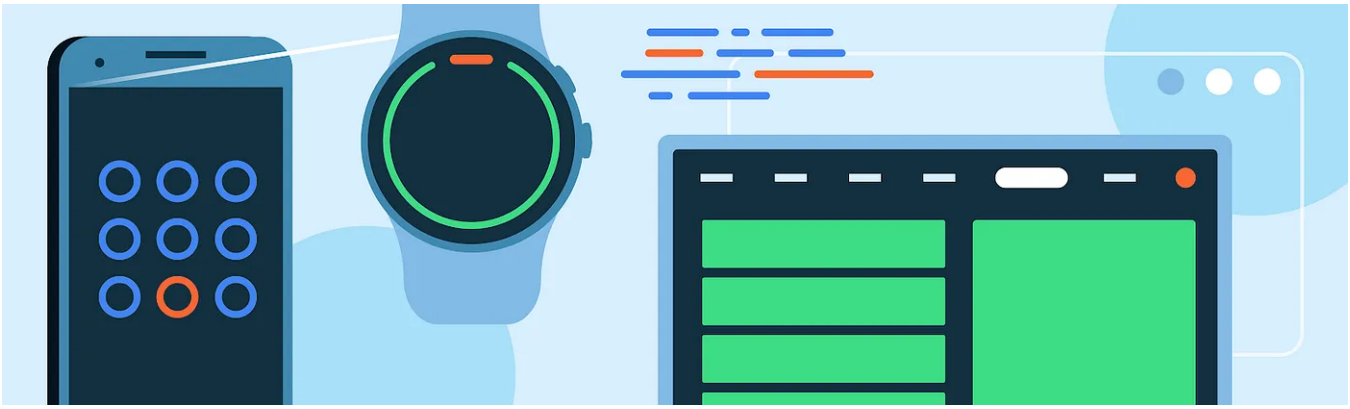


Image by android-developers.googleblog.com

With the growth in the use of mobile devices with different form factors, we need to have tools that allow us to work with our Android applications adapted to different types of screens. That is why Android provides us with **Window Size Classes**, which, in a simple way, are three large groups of screen formats that mark critical points for us to develop our designs. With this we avoid the complexity of thinking about many screen designs to reduce our possibilities to 3 groups which are: **Compat**, **Medium** and **Expanded.**

Windows Size Classes

Image by developer.android.com

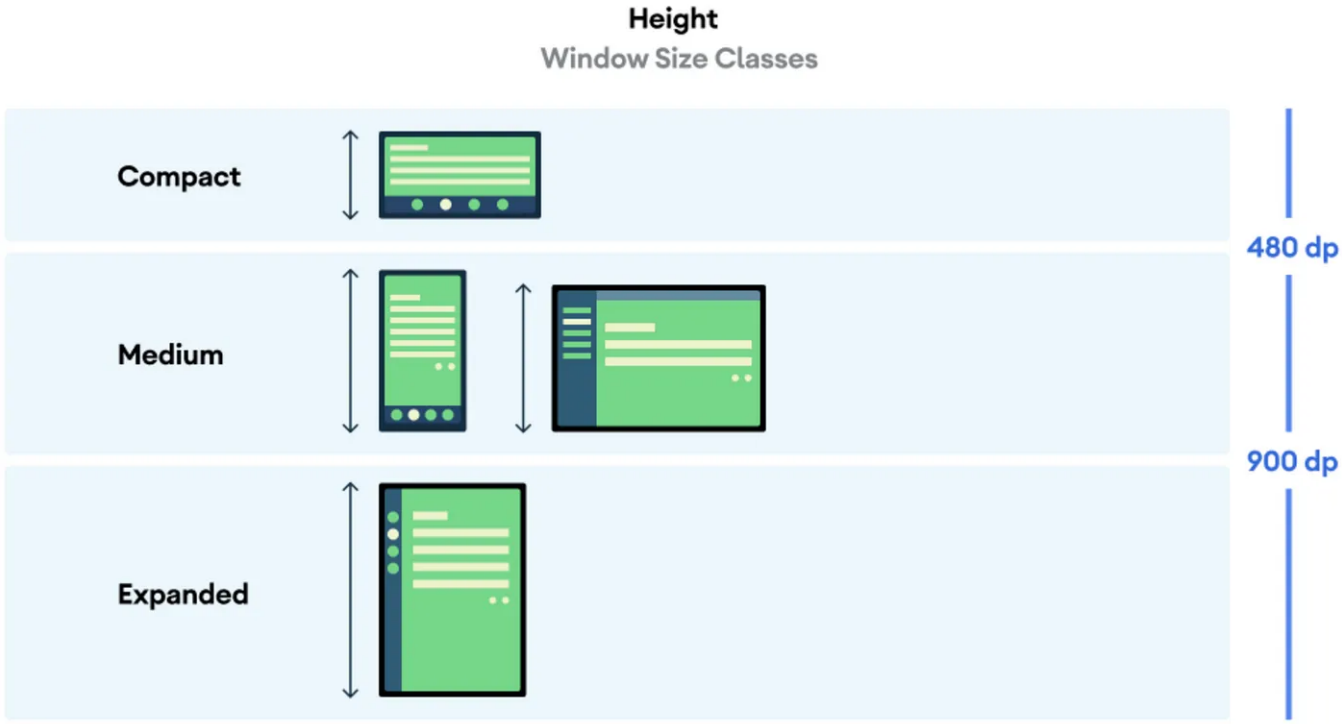
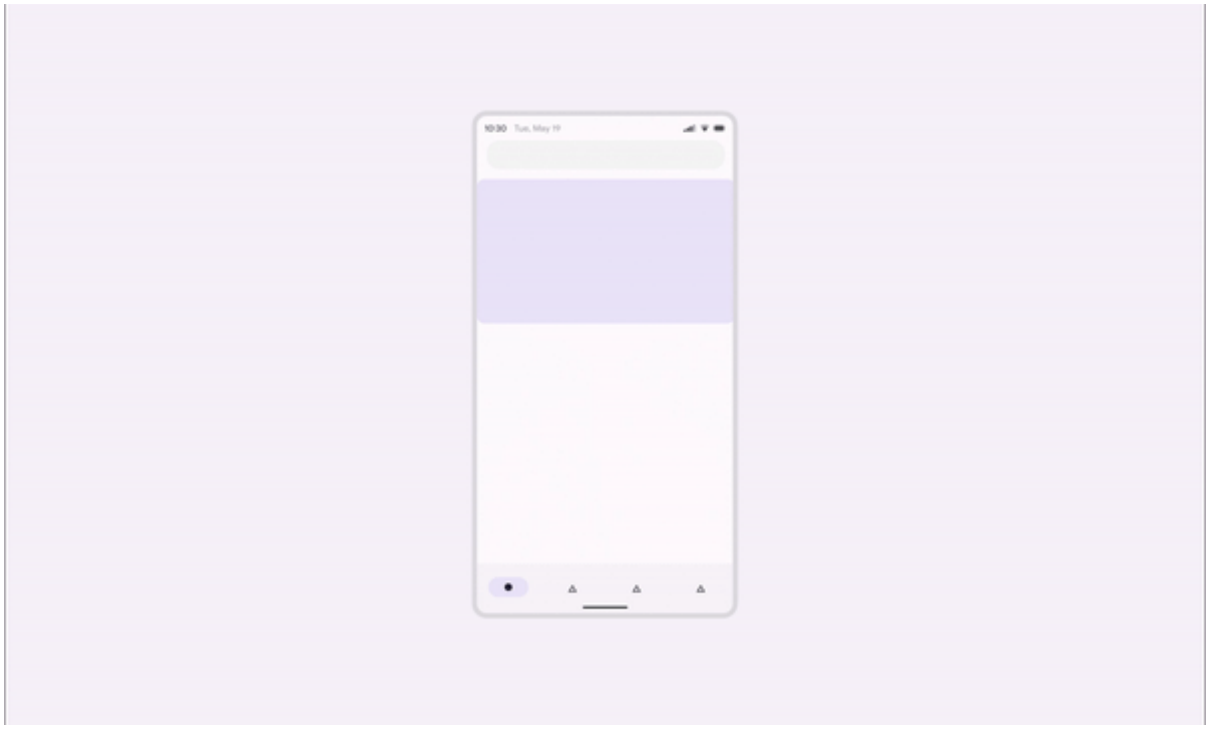


Image by developer.android.com

Support different screen sizes

Another important resource that we have are the **Canonical Layouts**, which are predefined screen designs that can be used for most scenarios in our Android applications and also show us a guide on how to adapt them to large screens.



Other related resources

- 3 things to know about Form Factors at Google I/O 2022
- Playlist: Form Factors at Google I/O

Form-Factor Training

Form Factors at Google I/O 2022 (2)

Performance

Image by android-developers.googleblog.com

While we develop applications for Android, we must ensure that the user experience is better, not only at the beginning of the application but also throughout its execution. For this reason, it is important to have tools that allow us to carry out a preventive analysis and constant monitoring of cases that may affect the performance of the application, so here is a list of tools that will help you with this purpose:

- Benchmark
- Baseline Profiles
- App Startup
- Firebase Performance Monitoring
- JankStats library

In-App Updates

> *When your users keep your app up to date on their devices, they can try new features, as well as benefit from performance improvements and bug fixes. Although some users enable background updates when their device is connected to an unmetered connection, other users might need to be reminded to install updates. In-app updates is a Google Play Core libraries feature that prompts active users to update your app.*
>
> *The in-app updates feature is supported on devices running Android 5.0 (API level 21) or higher. Additionally, in-app updates are only supported for Android mobile devices, Android tablets, and Chrome OS devices.*
>
> _ *In-App Updates documentation*

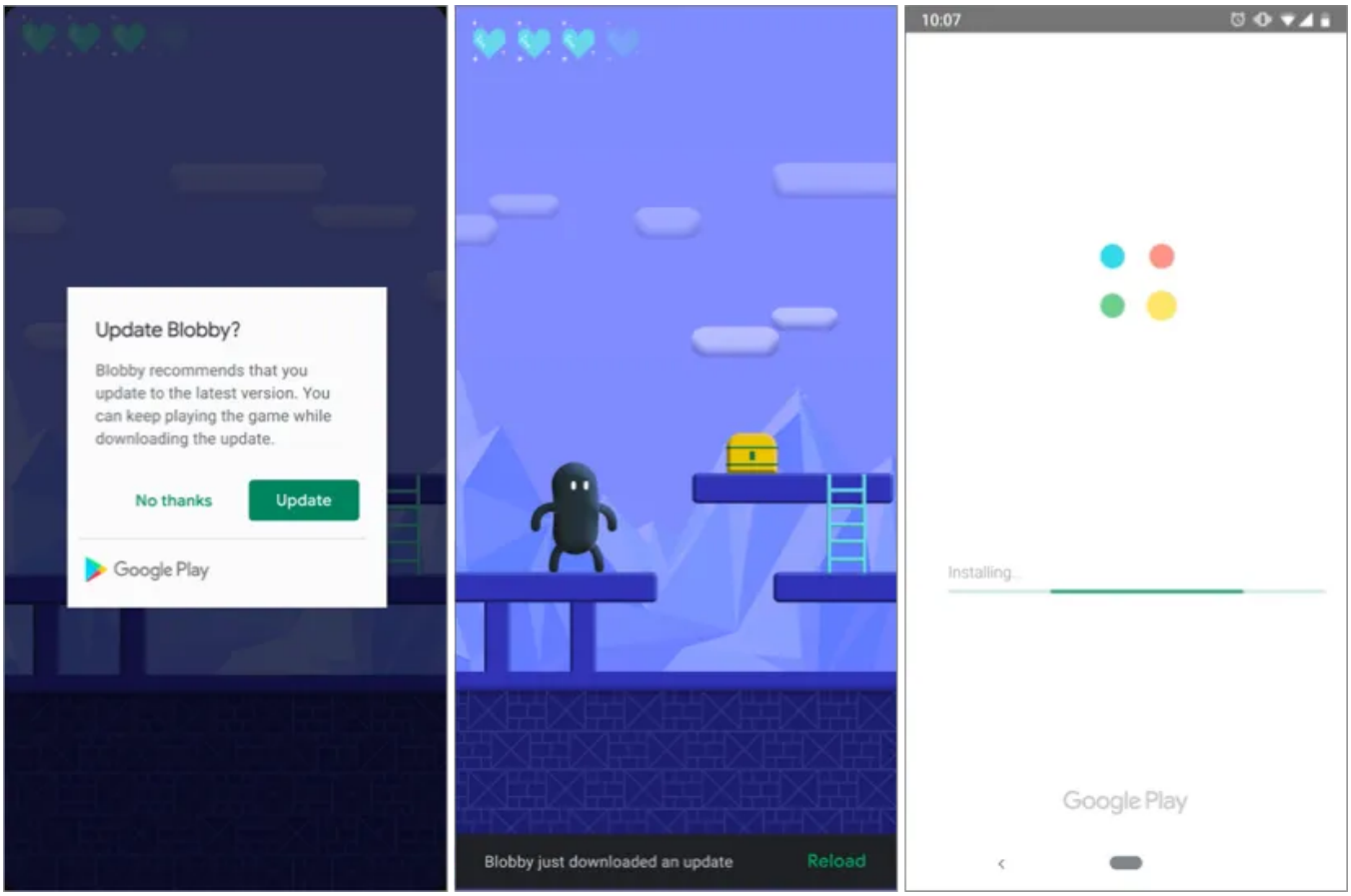
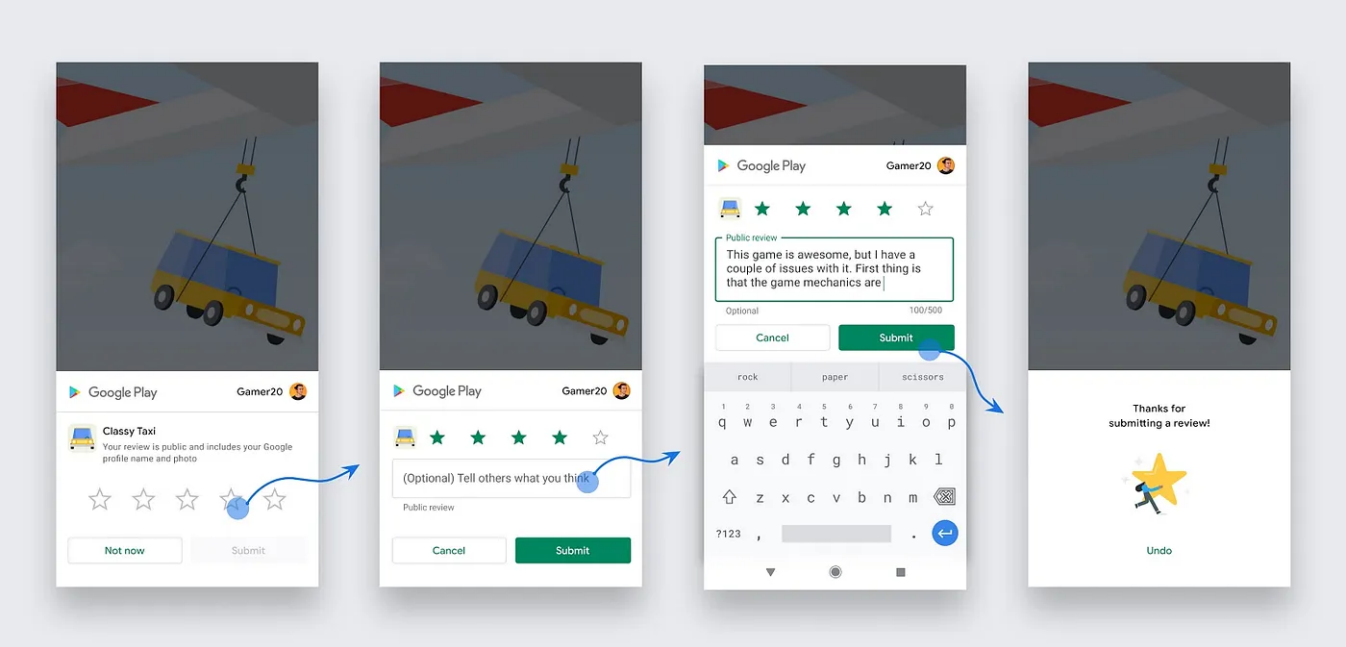


Image by developer.android.com

In-App Reviews

> *The Google Play In-App Review API lets you prompt users to submit Play Store ratings and reviews without the inconvenience of leaving your app or game.*
>
> *Generally, the in-app review flow can be triggered at any time throughout the user journey of your app. During the flow, the user has the ability to rate your app using the 1 to 5 star system and to add an optional comment. Once submitted, the review is sent to the Play Store and eventually displayed.*
>
> *To protect user privacy and avoid API misuse, there are strict guidelines that your app should follow about when to request in-app reviews and the design of the review prompt.*
>
> _ *In-App Reviews documentation*



Accessibility

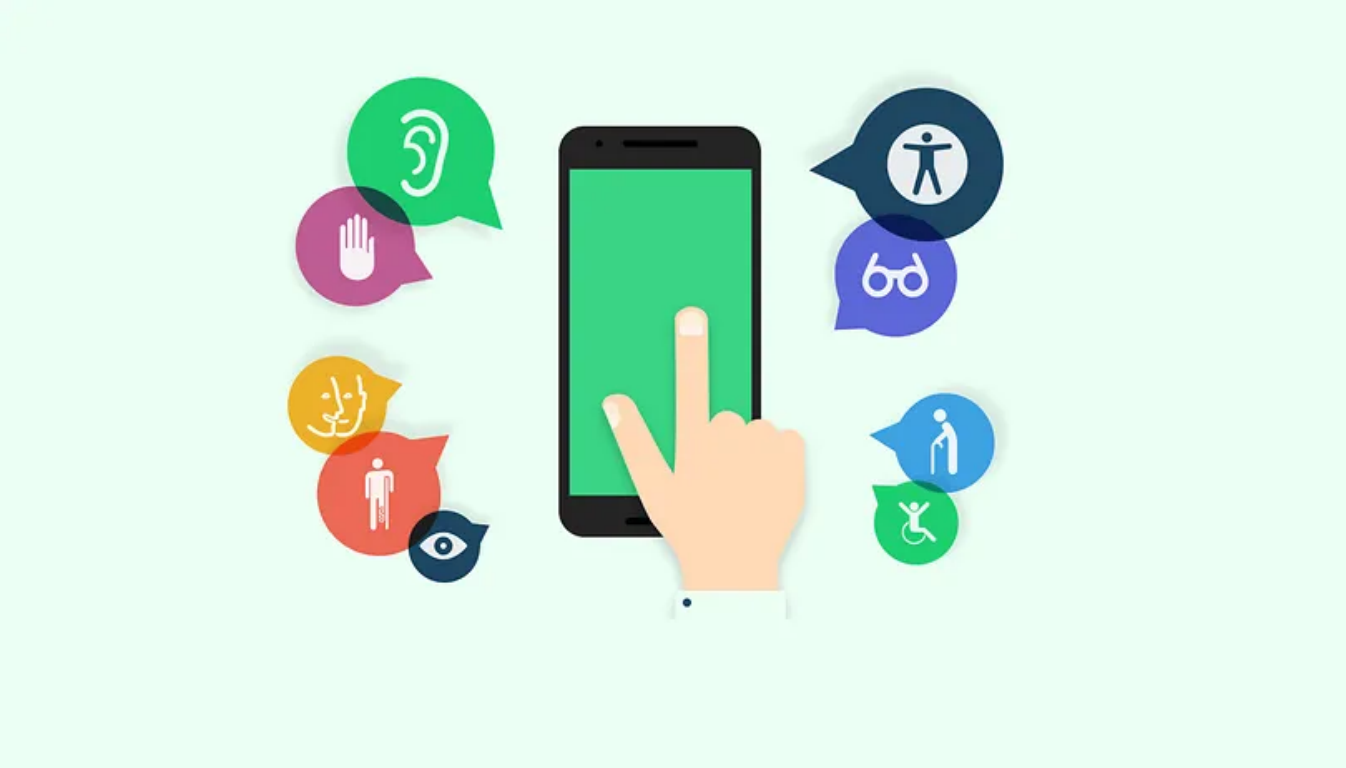


Image by fscl01.fonpit.de

Accessibility is an important feature in the design and construction of software that provides the ability for people with accessibility needs to use the application, in addition to improving their user experience. Some disabilities that this concept aims to improve are: people with vision problems, color blindness, hearing problems, dexterity problems, and cognitive disabilities, among others.

Considerations:

- Increase text visibility (Color contrast, Resizable Text)
- Use large, simple controls
- Describe each UI element

Check Accessibility — Android doc.

Security

Image by android.com

Security is one, if not the most important aspect, that we must take into account when developing applications that protect the integrity of the device, the security of the data, and the trust of the user, which is why I list below a series of tips that will help you with this purpose.

- Encrypt sensitive data and files: Use EncryptedSharedPreferences and EncryptedFile.
- Apply signature-based permissions:
- Use **signature-based permissions** when sharing data between apps you have control over.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">
    <permission android:name="my_custom_permission_name"
                android:protectionLevel="signature" />
```

- Do not put keys, tokens, or sensitive data required for your application's configuration directly inside files or classes that are inside the project repository. Use **local.properties** instead.

Version Catalogs

Gradle provides a standard way to centrally manage project dependencies called the version catalog; it was experimentally introduced in version 7.0 and officially released in version 7.4.

Advantage:

- For each catalog, Gradle generates *type-safe accessors* so that you can easily add dependencies with autocompletion in the IDE.
- Each catalog is visible to all projects of a build. It is a central place to declare a version of a dependency and to make sure that a change to that version applies to every subproject.
- Catalogs can declare dependency bundles, which are "groups of dependencies" that are commonly used together.
- Catalogs can separate the group and name of a dependency from its actual version and use version references instead, making it possible to share a version declaration between multiple dependencies.

see more

Logger

A logger is a software tool used to register information about the execution of a program; important events, errors debug messages and other information that may be useful in diagnosing problems or understanding how a program is working. Loggers can be configured to write messages to different locations, such as a log file, to the console, to a database, or by sending the messages to a logging server.

- Timber

Linter

Image taken from https://miro.medium.com/

Linter is a programming tool that is used to analyze the program source code to find potential problems or bugs in the code. These issues can be syntactic, inappropriate code style, lack of documentation, security issues, and so on, and they can have an impact on the quality and maintainability of the code.

- Android Lint
- Detekt
- Ktlint

Follow me in

- **Twitter:** @devjcastro
- **Linkedin:** dev-jorgecastro

Android App Development | AndroidDev | Android | Jetpack Compose

Follow

Written by Jorge Luis Castro Medina

516 Followers

I'm a Software Engineer passionate about mobile technologies, and I like everything related to software design and architecture

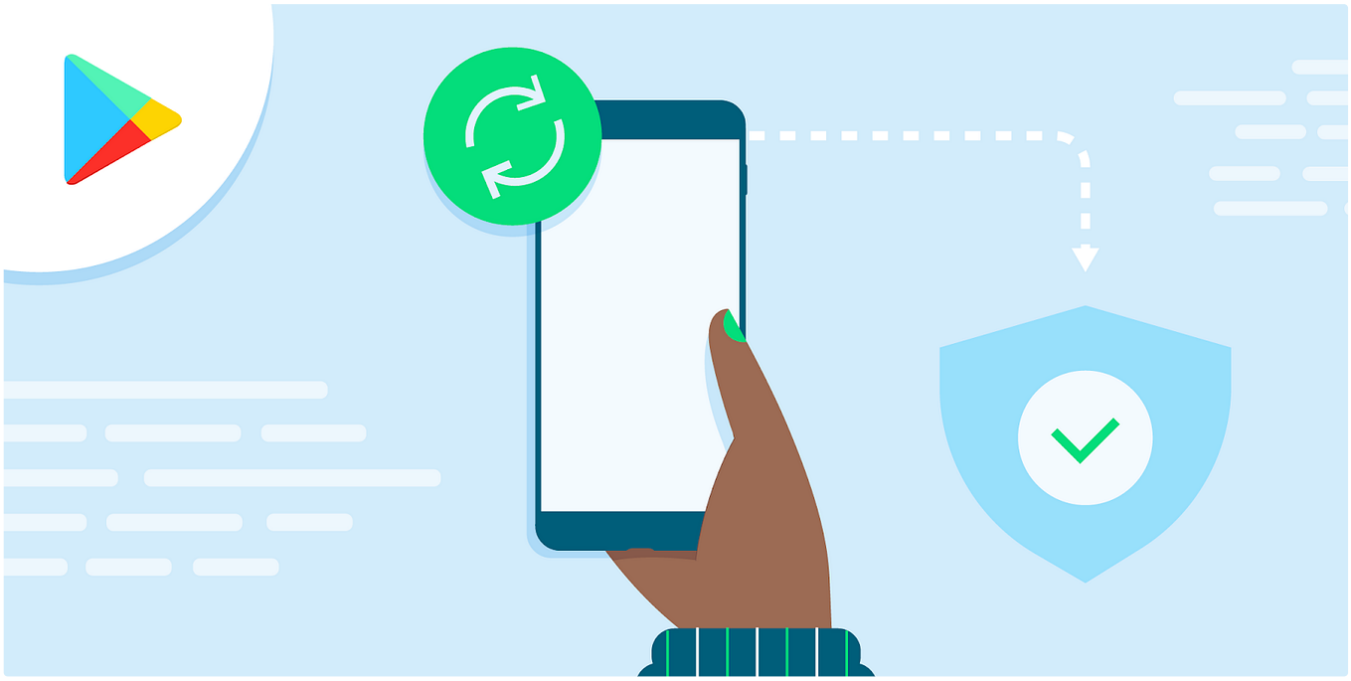
More from Jorge Luis Castro Medina

Jorge Luis Castro Medina

Kotlin Multiplatform — This is the way.

Revolutionizing App Development: Unleashing the Power of Kotlin Multiplatform

5 min read · Jul 24



Jorge Luis Castro Medina

Update your Android app signing key from a new Keystore with Play App Signing.

Jorge Luis Castro Medina

Add blur effect to Image in Jetpack Compose

Blur is a visual effect that is often used in graphics software to reduce the clarity or sharpness of an image or video.

3 min read · Jan 9

Jorge Luis Castro Medina

How to deploy your Android App to the Internal track in the Play Store Console using Fastlane

A step-by-step guide on how to use Fastlane to deploy your Android app to the Internal track in the Play Store Console

6 min read · Jun 16

See all from Jorge Luis Castro Medina

Recommended from Medium

Yahya Bin Naveed

How I Learned to Develop Android Apps With Jetpack Compose (2023)

IN ONLY 3 MONTHS

10 min read · Apr 13

365 3

Benoit Ruiz in Better Programming

Advice From a Software Engineer With 8 Years of Experience

Practical tips for those who want to advance in their careers

22 min read · Mar 21

10.1K 201

Lists

Medium Publications Accepting Story Submissions

154 stories · 770 saves

Staff Picks

465 stories · 318 saves

Duggu

50 best practices for Kotlin programming

I am currently crafting an article outlining the most effective Kotlin programming practices based on my knowledge and experience.

4 min read · Sep 20

31 2

Elye in Mobile App Development Publication

The onBackPressed Is Now Deprecated in Android 13 and Might Need Major Changes in Existing Apps

Migrate to Android 13's “Predictive Back” Before It’s Too Late

· 15 min read · Sep 3


```
: class BackgroundService extends Service


erride
ublic int onStartCommand(Intent intent, int flags, int start
 return Service.START_STICKY;


 @Nullable
)verride
ublic IBinder onBind(Intent intent) {
   return null;
```

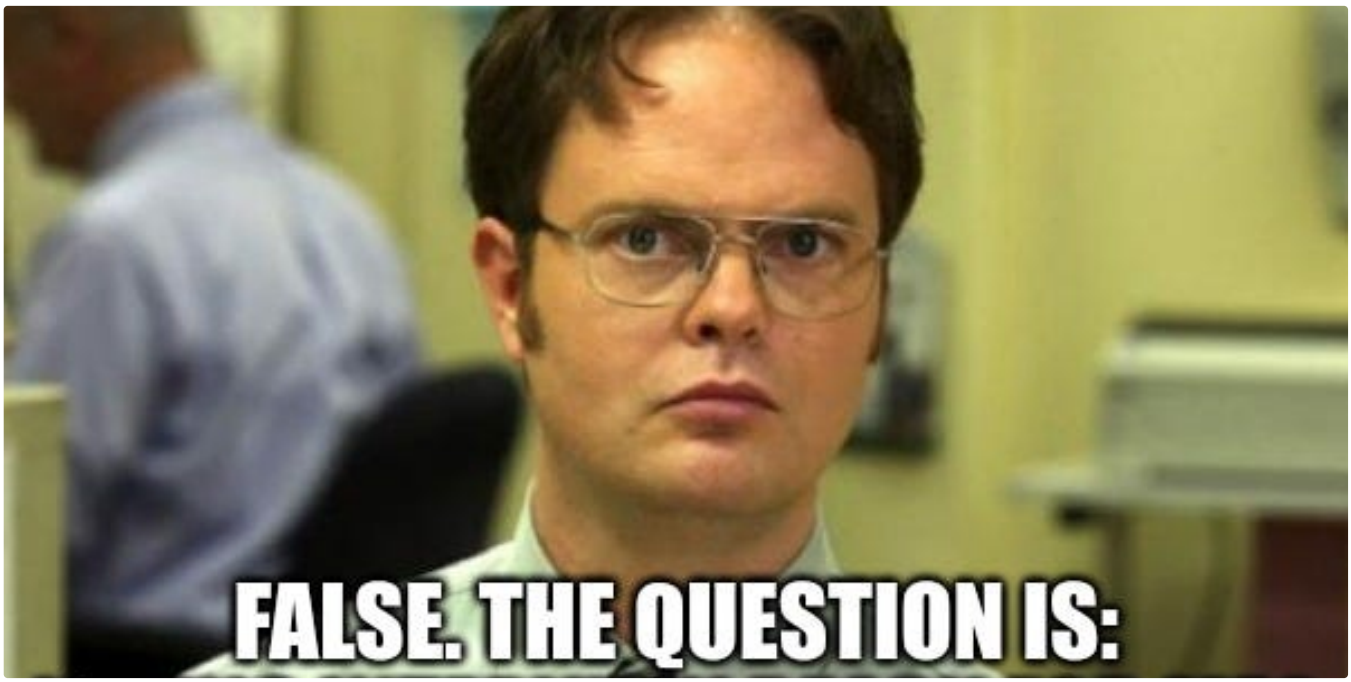
Shubh

Background service running forever.

Background service running forever in android

2 min read · May 5

114 2



Alessandro Faranda Gancio

UPDATED: Common Android Developer Interview Questions and Answers

As an experienced Android developer, I have been through numerous job interviews, both as a candidate and an interviewer. While every...

· 12 min read · May 11

140

See more recommendations