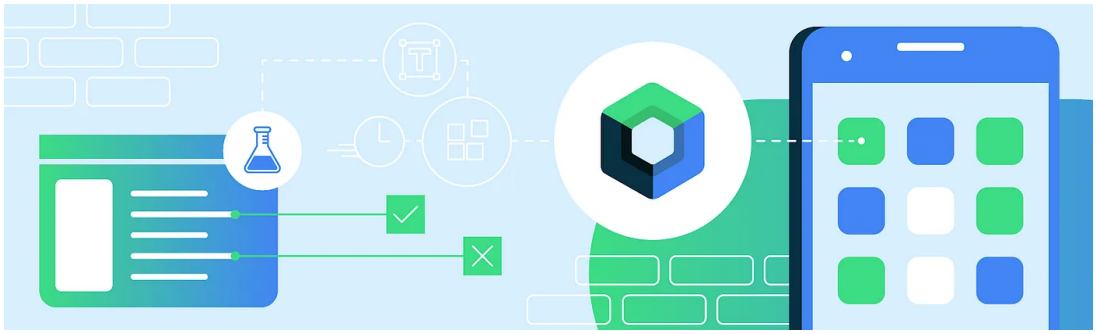


Modern Android Development in 2024

Jorge Luis Castro Medina · Follow
18 min read · Feb 12, 2024

Hello everyone 👋🏻, we are kicking off a new year, and I don't want to miss this chance to share with you an article titled `Modern Android Development` in its 2024 version. 🚀

If you missed my previous article, you can take a look at Modern Android Development in 2023 and review the changes for this year

Disclaimer

📝 This article reflects my personal opinions and professional insights, considering diverse viewpoints within the Android developer community. Additionally, I regularly review the guidelines presented by Google for Android.

🧯 It is crucial to emphasize that while I may not explicitly mention certain compelling tools, patterns, and architectures, this omission does not negate their potential as valuable alternatives for developing Android applications.

Use Kotlin everywhere ❤️

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

Kotlin is a multipurpose language that we can use not only for developing Android applications, although much of its popularity has been due to the latter, as we can see in the following graph.

KotlinConf '23

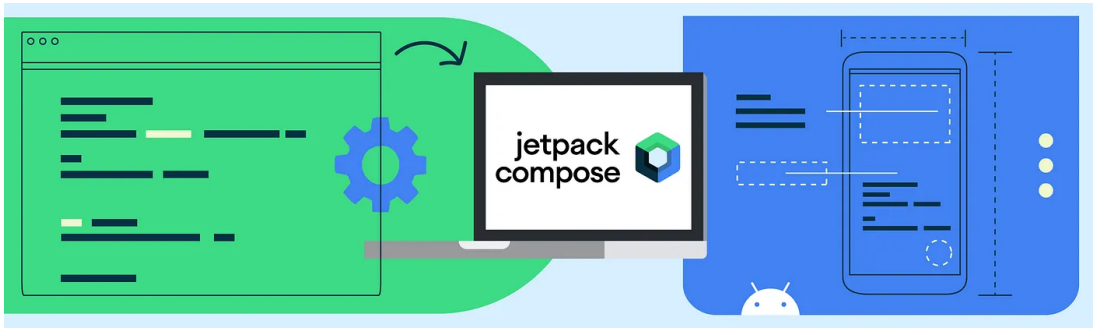
Kotlin 2.0 is here

Another important thing to highlight is the release of `Kotlin 2.0`, which is just around the corner. As of the article's date, it's in its version 2.0.0-beta3

The new K2 compiler is also another addition in Kotlin 2.0, which will bring significant performance improvements, accelerate the development of new language features, unify all platforms supported by Kotlin, and provide a better architecture for multiplatform projects.

Check out the recap of KotlinConf '23 where you can find more information.

Compose 🚀



> *Jetpack Compose is Android's recommended modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs.*
>
> — *Jetpack Compose documentacion*

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:

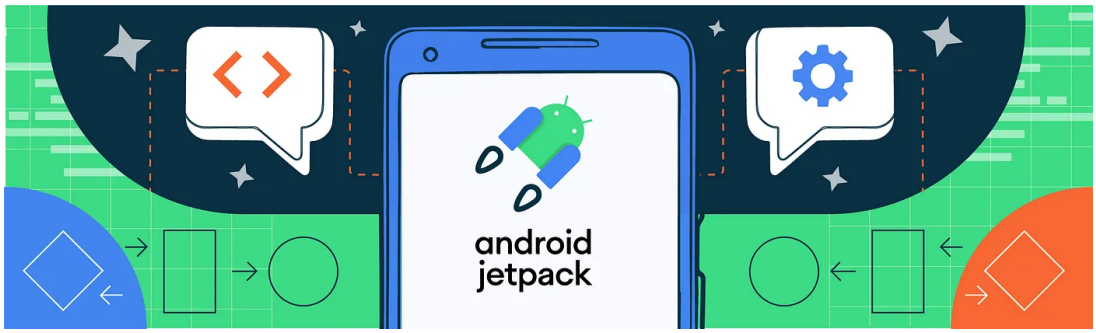
1. Declarative UI.
2. Customizable widgets.

3. Easy integration with existing code (old view system).
4. Live preview.
5. Improved performance.

Resources:

- Jetpack Compose documentation
- Compose to Kotlin Compatibility Map
- Jetpack Compose Roadmap
- Course.
- API Guidelines for `@Composable` components in Jetpack Compose

Android Jetpack ⚙️



> 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
- Media3
- Glance

Material You / Material Design 🥰

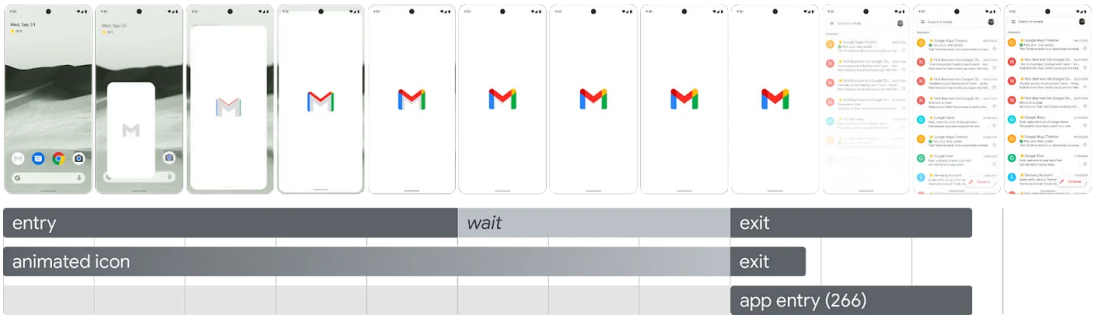
Material You, a new customization feature introduced in Android 12 and implemented within Material Design 3, empowers users to tailor the visual appearance of their operating system to align with their personal preferences. This innovative addition complements Material Design, an adaptable system of guidelines, components, and tools crafted to uphold the highest standards of user interface design. Supported by open-source code, Material Design fosters seamless collaboration between designers and developers, enabling teams to efficiently create stunning products.

Currently, the last version for Material Design is 3, you can see more here. Furthermore, you can take advantage of Material Theme Builder as assistance in defining your application's theme.

Codelab

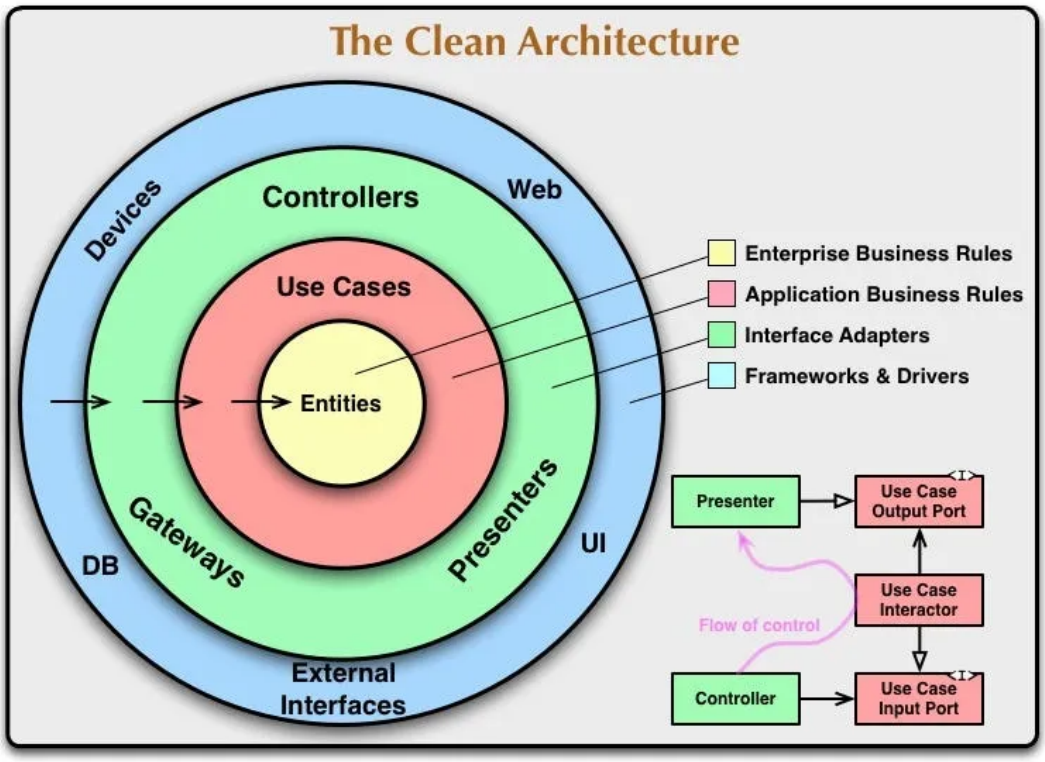
Theming in Compose with Material 3

Splash screens API



The SplashScreen API in Android is essential to ensure that applications display correctly on Android 12 and later versions. Not updating can impact the application launch experience. It is crucial to quickly adopt this API for a consistent user experience on the latest versions of the operating system.

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, Composables, Fragments, View Models and 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**

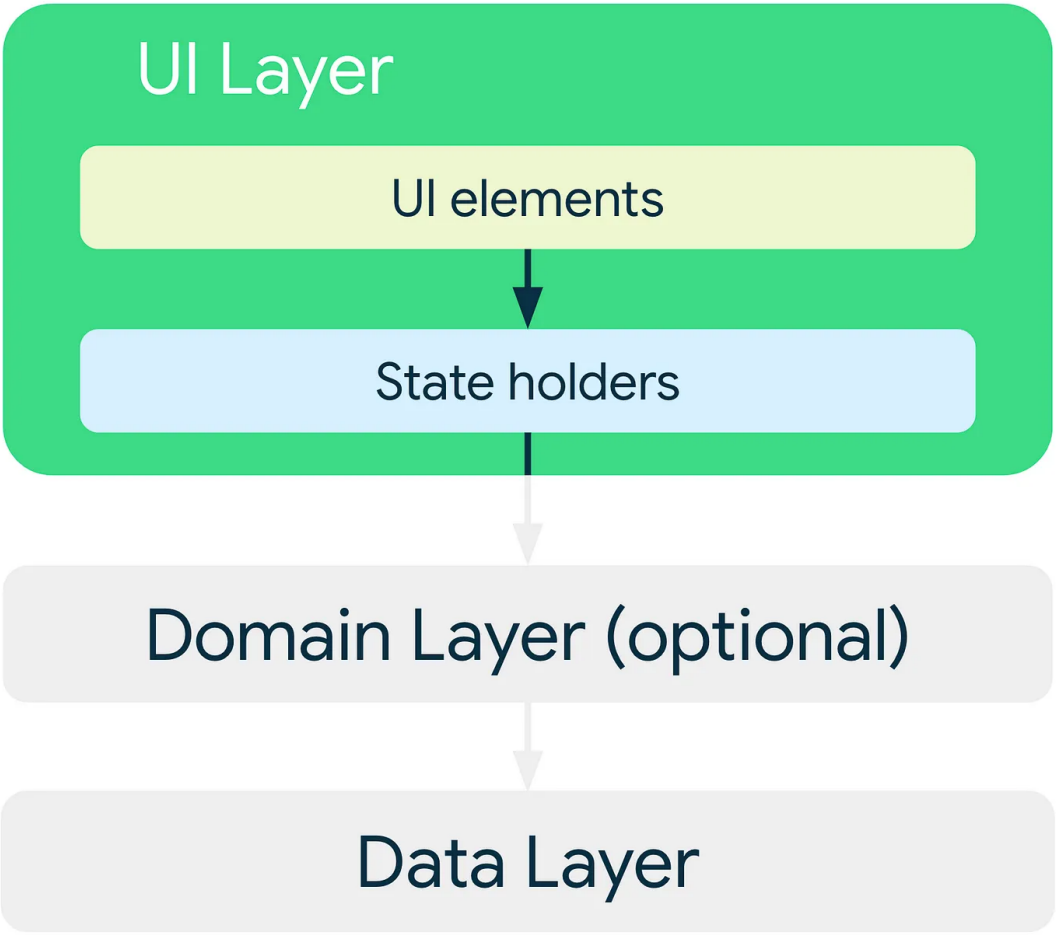


Image by developer.android.com

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.

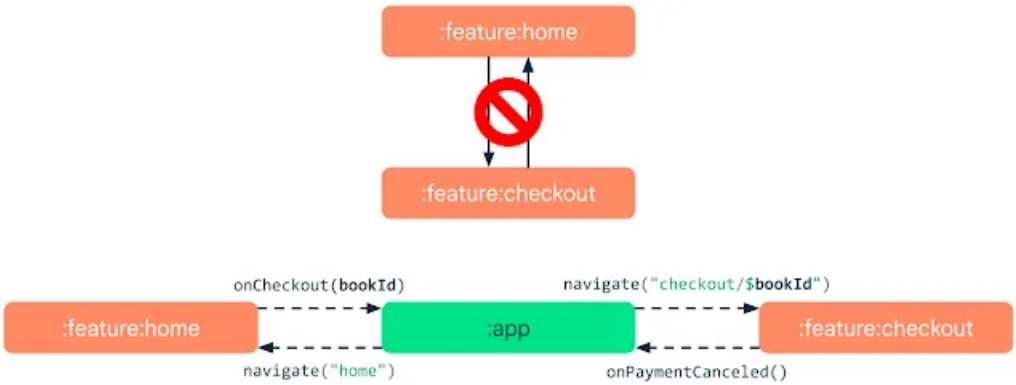


Image by developer.android.com

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
- Ktor

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 Kotlin team's bet at JetBrains.

- 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.

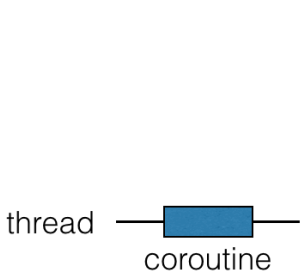
Image Loading

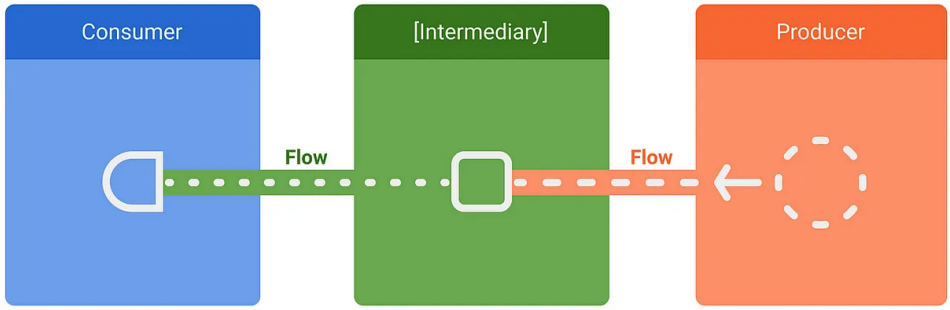
> To load an image from the internet, there are several third-party libraries available to help you handle the process. Image loading libraries do a lot of the heavy lifting for you; they handle both caching (so you don't download

> 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 it comes to reactive programming and asynchronous processes, Kotlin Coroutines stand out with their Suspension Functions and Flow. However, it is crucial to acknowledge the value of RxJava in Android application development. Despite the growing adoption of Coroutines and Flow, RxJava remains a robust and popular choice in multiple projects.

For new projects always choose `Kotlin Coroutines` ❤️. Explore some Kotlin Coroutines Concepts

Local Store

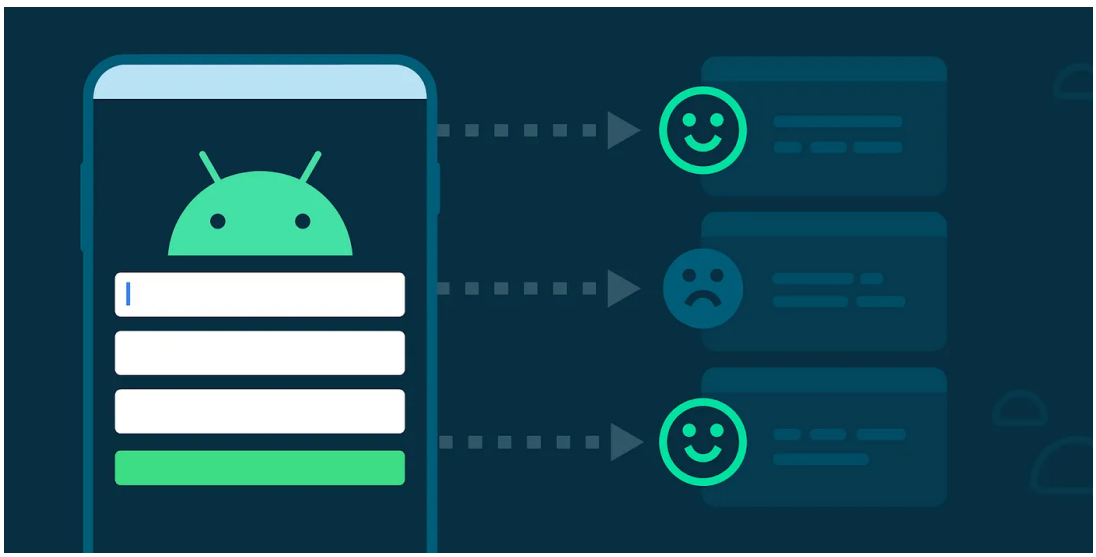
An important point when building mobile applications is having the ability to persist data locally, such as some session data or cache data, among others. It is important to choose the right storage option based on the needs of your application. We could store unstructured data like key-value or structured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

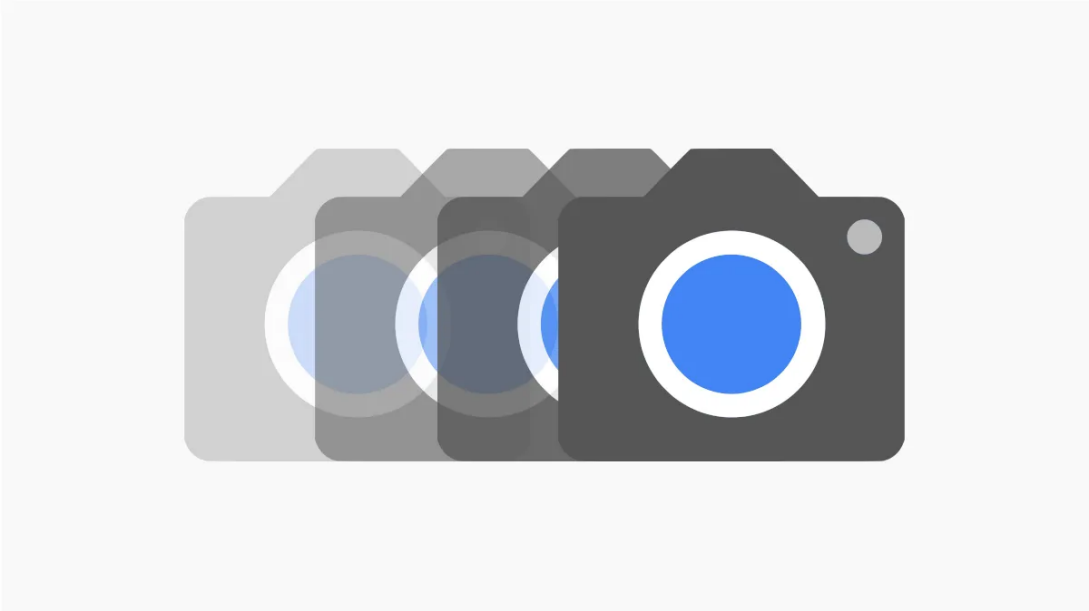


Testing in software development is essential to ensure product quality. It detects errors, validates requirements, and ensures customer satisfaction. Below are some of the most commonly used tools for this purpose:

- JUnit 5
- Mockk
- Kotest
- Kluent
- Turbine
- Espresso
- Robolectric
- Maestro

Testing section of the Tools documentation

Screenshot testing 📸



Screenshot testing in Android involves automatically capturing screenshots of various UI elements in an application and comparing them against baseline images to detect any unintended visual changes. It helps ensure consistent UI appearance across different versions and configurations of the app, catching visual regressions early in the development process.

- Paparazzi
- Roborazzi

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. Additionally, you can also disable certain tasks or customize R8's behavior through `ProGuard` rules files.

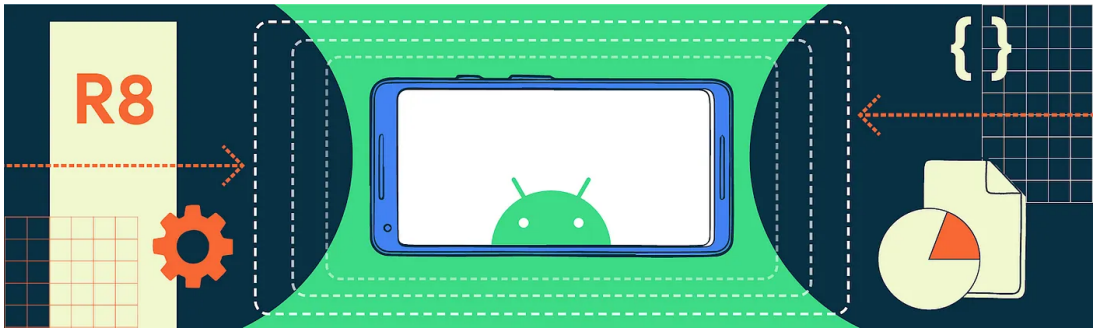


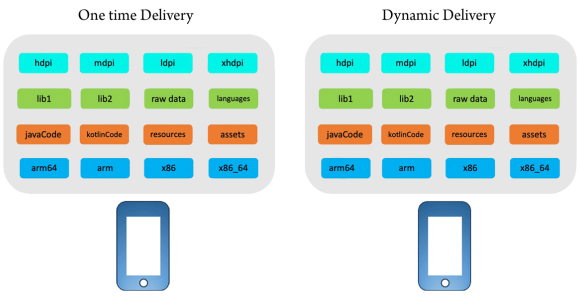
Image by androidtopics.dipien.com

- Code shrinking
- Resource shrinking
- Obfuscation
- Optimization

Third-party tools

- DexGuard

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*

Adaptive layouts

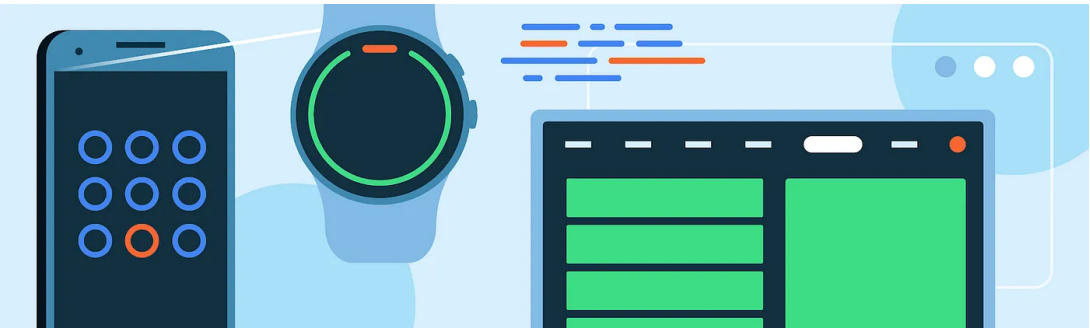


Image by android-developers.googleblog.com

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

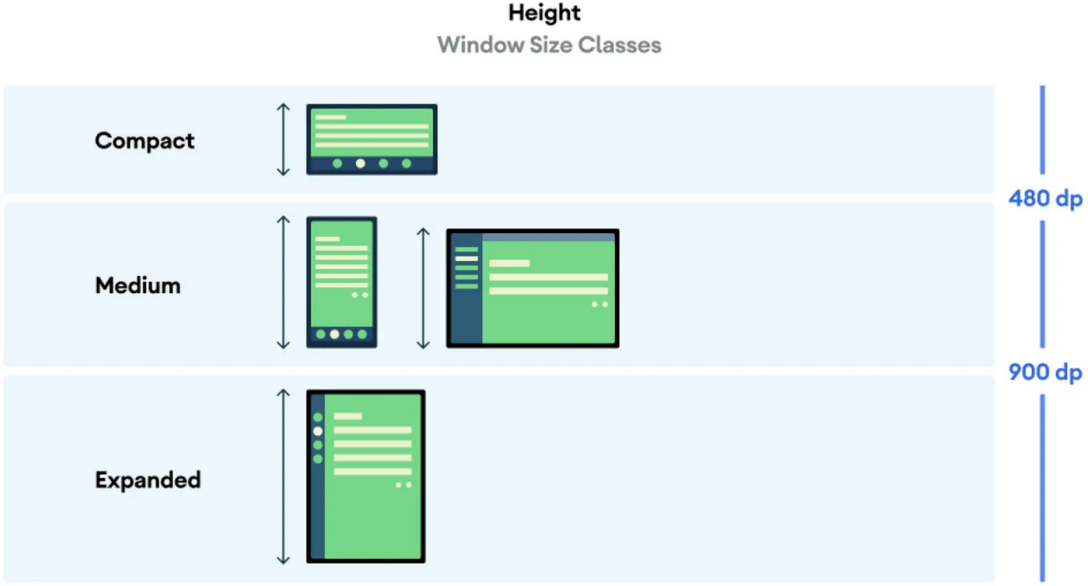
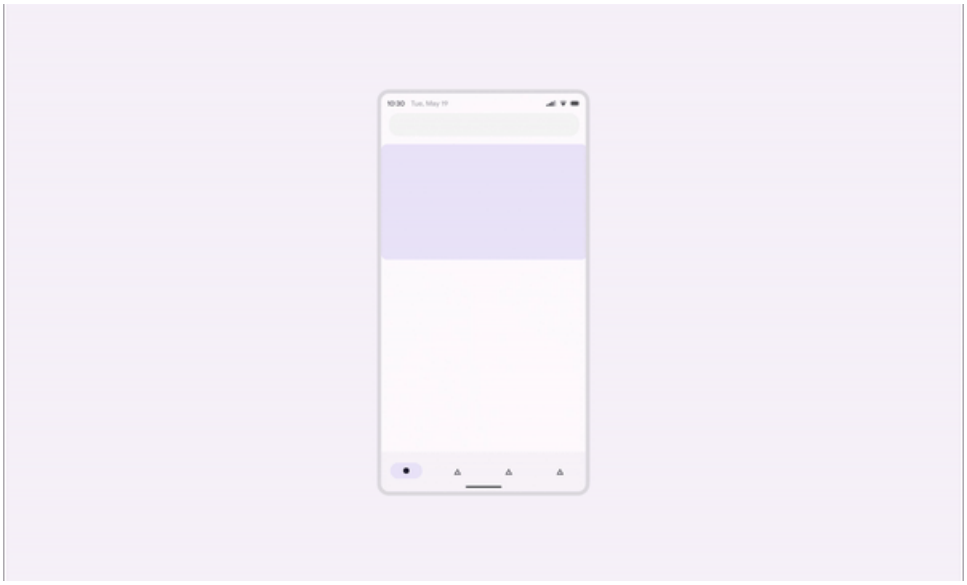


Image by developer.android.com

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.

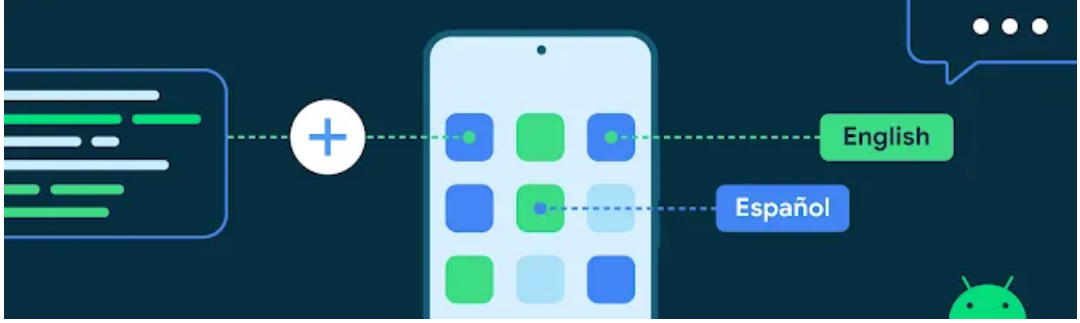


Other related resources

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

Form-Factor Training

Localization 🌎



Localization involves adapting a product to meet the needs of diverse audiences in different regions. This includes translating text, adjusting formats, and considering cultural aspects. Its advantages include access to global markets, enhanced user experience, increased customer satisfaction, competitiveness in the global market, and compliance with local regulations.

Note: *BCP 47 is a standard used by Android for internationalization*

References

- Support different languages and cultures
- Localize your app
- Per-app language preferences
- Per-App Language Preferences — Part 1
- Per-App Language Preferences — Part 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

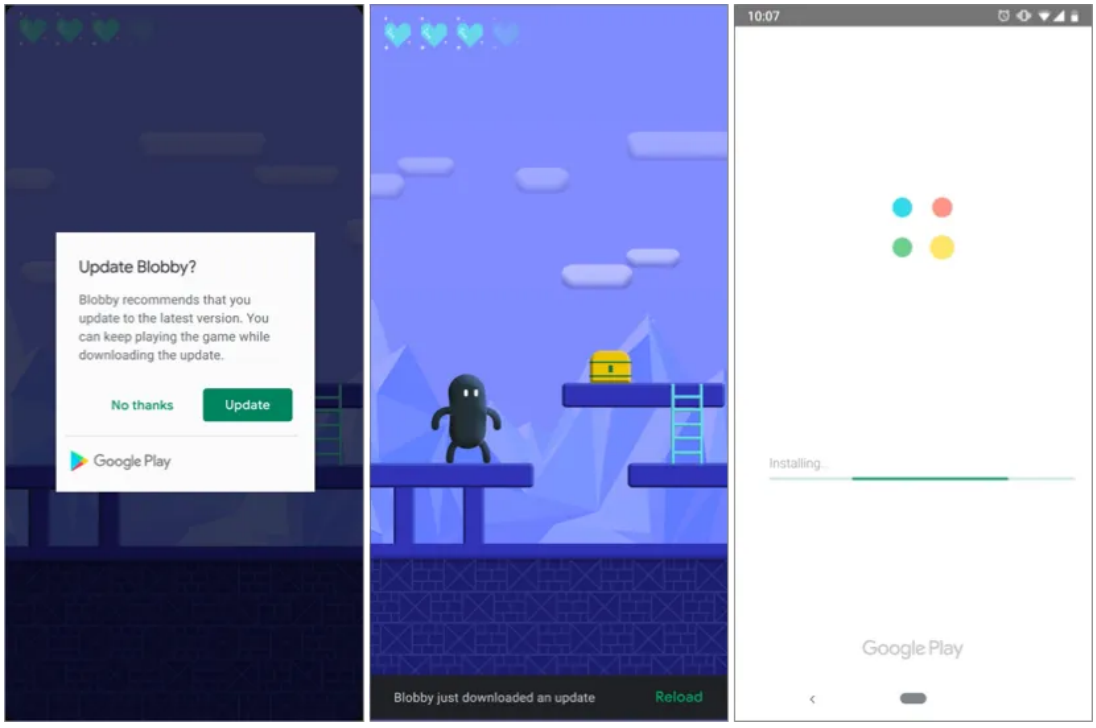
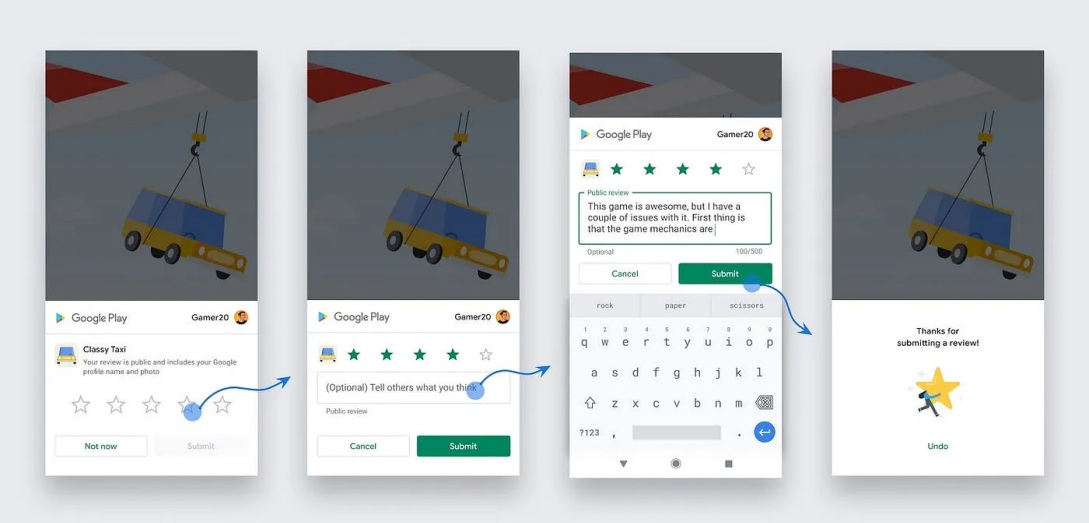


Image by developer.android.com

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



Observability 👀

Image taken from Elastic Blog

In an increasingly competitive app ecosystem, achieving a good user experience begins with ensuring that the app is bug-free. One of the best ways to ensure that the app is bug-free is to immediately detect issues as they arise and know how to start addressing them. Use Android Vitals to identify the areas of your app that have the most crashes and issues with responsiveness. Then, utilize custom crash reports in Firebase Crashlytics to get more details about the root causes in order to troubleshoot the issues effectively.

Tools

- Google Analytics
- Android Vitals
- Firebase Crashlytics
- Firebase Performance Monitoring

Accessibility

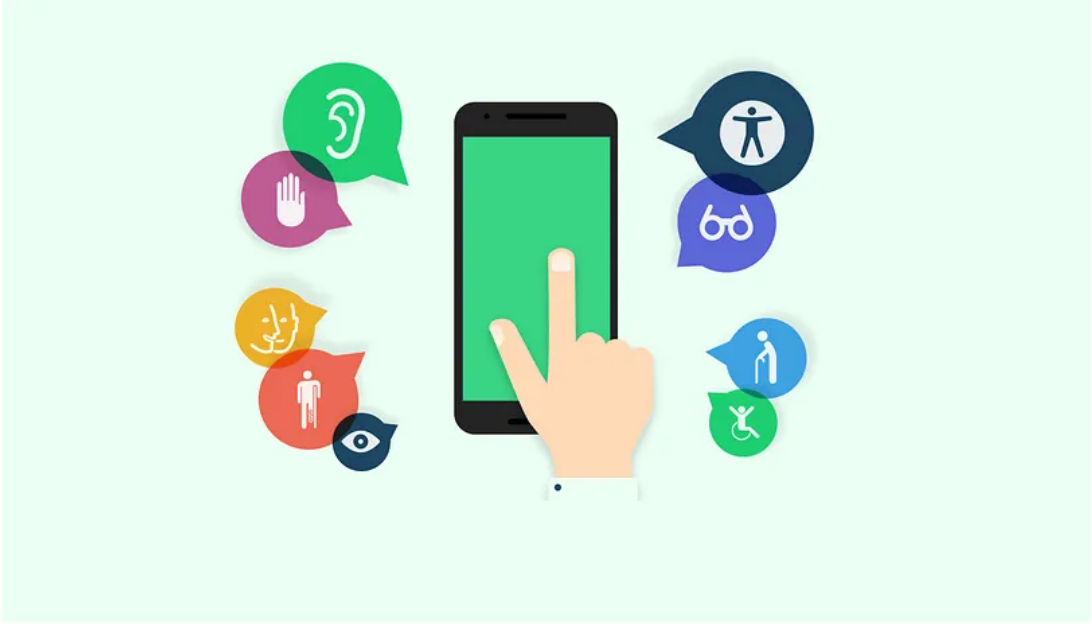


Image by fscl01.fonpit.de

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.

- Sign in your user with Credential Manager: Credential Manager is a Jetpack API that supports multiple sign-in methods, such as username and password, passkeys, and federated sign-in solutions (such as Sign-in with Google) in a single API, thus simplifying the integration for developers.
- 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.
- Implement SSL Pinning: Use SSL Pinning to further secure communications between your application and remote servers. This helps prevent man-in-the-middle attacks and ensures that

communication only occurs with trusted servers possessing a specific SSL certificate.

`res/xml/network_security_config.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <pin-set expiration="2018-01-01">
            <pin digest="SHA-256">ReplaceWithYourPin</pin>
            <!-- backup pin -->
            <pin digest="SHA-256">ReplaceWithYourPin</pin>
        </pin-set>
    </domain-config>
</network-security-config>
```

- Implementa Runtime Application Self Protection (RASP): It is a security technique that protects applications at runtime against attacks and vulnerabilities. RASP works by monitoring the application's behavior and detecting suspicious activities that could indicate an attack. some of the advantages that `RASP` provides us:

  - Code Obfuscation.
  - Root Detection.
  - Tampering/App Hook Detection.
  - Prevention of reverse engineering attacks.
  - Anti-Debugging Techniques.
  - Virtual Environment Detection.
  - Runtime Analysis of App Behavior.

Look at this article for more information: Runtime Application Self Protection techniques(RASP) in Android Apps. Also some Security guidelines by Android

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

Secrets Gradle Plugin

Google strongly recommends that you not check an API key into your version control system. Instead, you should store it in a local `secrets.properties` file, which is located in the root directory of your project but excluded from version control, and then use the Secrets Gradle Plugin for Android to read the API key.

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.

- Klogging
- Timber

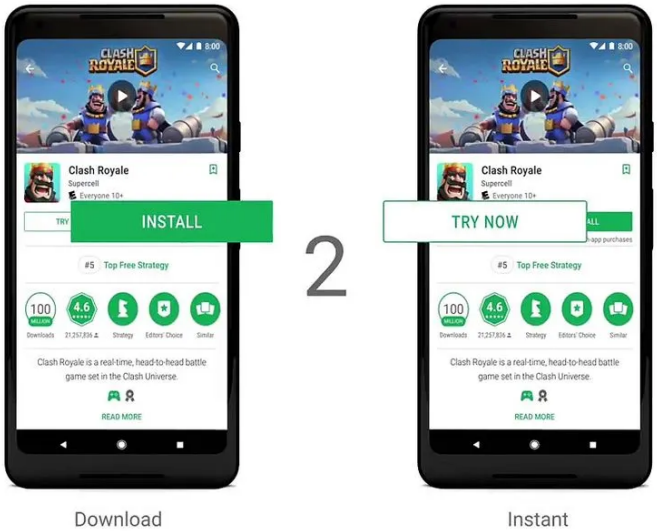
Linter / Static Code Analyzer

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 ⏫
- Konsist

Google Play Instant

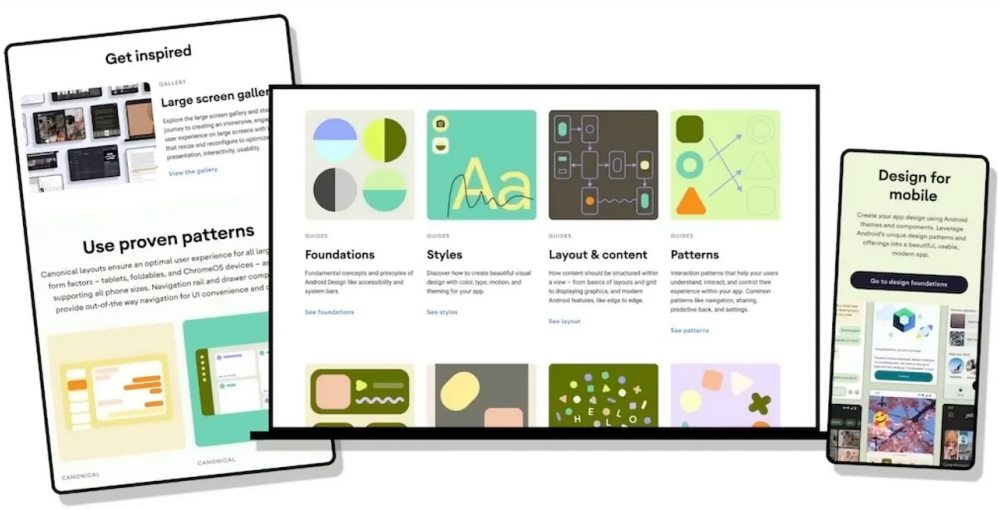


Download Instant

Google Play Instant enables native apps and games to launch on devices running Android 5.0 (API level 21) or higher without being installed. You can build these types of experiences, called *instant apps* and *instant games*, using Android Studio. By allowing users to run an instant app or instant game, known as providing an *instant experience*, you improve your app or game's discovery, which helps drive more active users or installations.

— Overview of Google Play Instant

New Design Hub



The Android team provides a new design hub to help create beautiful, modern Android apps, a centralized place to understand design for Android

AI

`Gemini` and `PalM 2`, two state-of-the-art artificial intelligence (AI) models developed by Google, are poised to transform the landscape of Android application development. These models offer a range of benefits that will drive efficiency, user experience, and innovation in applications.

- Gemini
- PalM 2

AI Coding Assistant Tools

Studio Bot

> *Studio Bot is your coding companion for Android development. It's a conversational experience in Android Studio that helps you be more productive by answering Android development queries. It's powered by artificial intelligence and can understand natural language, so you can ask development questions in plain English. Studio Bot can help Android developers generate code, find relevant resources, learn best practices, and save time.*
>
> — *Studio Bot*

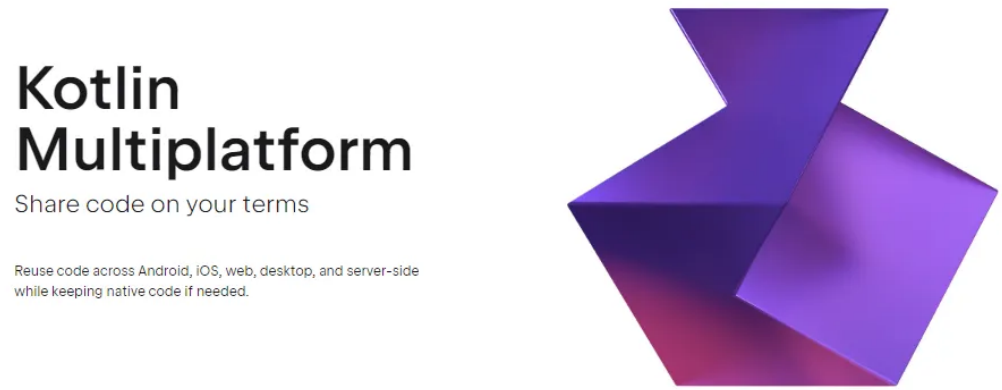
Github Copilot

GitHub Copilot is an AI pair programmer. You can use GitHub Copilot to get suggestions for whole lines or entire functions right inside your editor.

Amazon CodeWhisperer

It is an Amazon service that generates code recommendations based on the context of your current code. It helps you write more efficient and secure code, and discover new APIs and tools.

Kotlin Multiplatform 🖥️📱⌚🥰🚀



Finally, and equally significant, the standout revelation of the year is Kotlin Multiplatform 🎉. It emerges as a formidable contender in the realm of

cross-platform application development. While our primary focus may lie in Android app development, Kotlin Multiplatform offers us the flexibility to craft a fully native Android application leveraging the KMP framework. This strategic move not only future-proofs our projects but also equips us with the necessary infrastructure for a seamless transition to a multiplatform environment, should we choose to pursue it. 🚀

If you're keen on delving deeper into Kotlin Multiplatform, I'd like to share a couple of articles I wrote a few months ago. In these pieces, I explore the current state of this technology and its implications for modern software development

- Kotlin Multiplatform — This is the way
- The State of Kotlin Multiplatform — Webinar 2023

Sources

- What's new in Android — IO/23
- Kotlin Multiplatform

If you like my content and want to support my work, you can give me a cup of coffee ☕ 🥰

Follow me in

- **Twitter:** @devjcastro
- **Linkedin**: devjcastro

Android App Development | AndroidDev | Android | Jetpack Compose | Kotlin Multiplatform