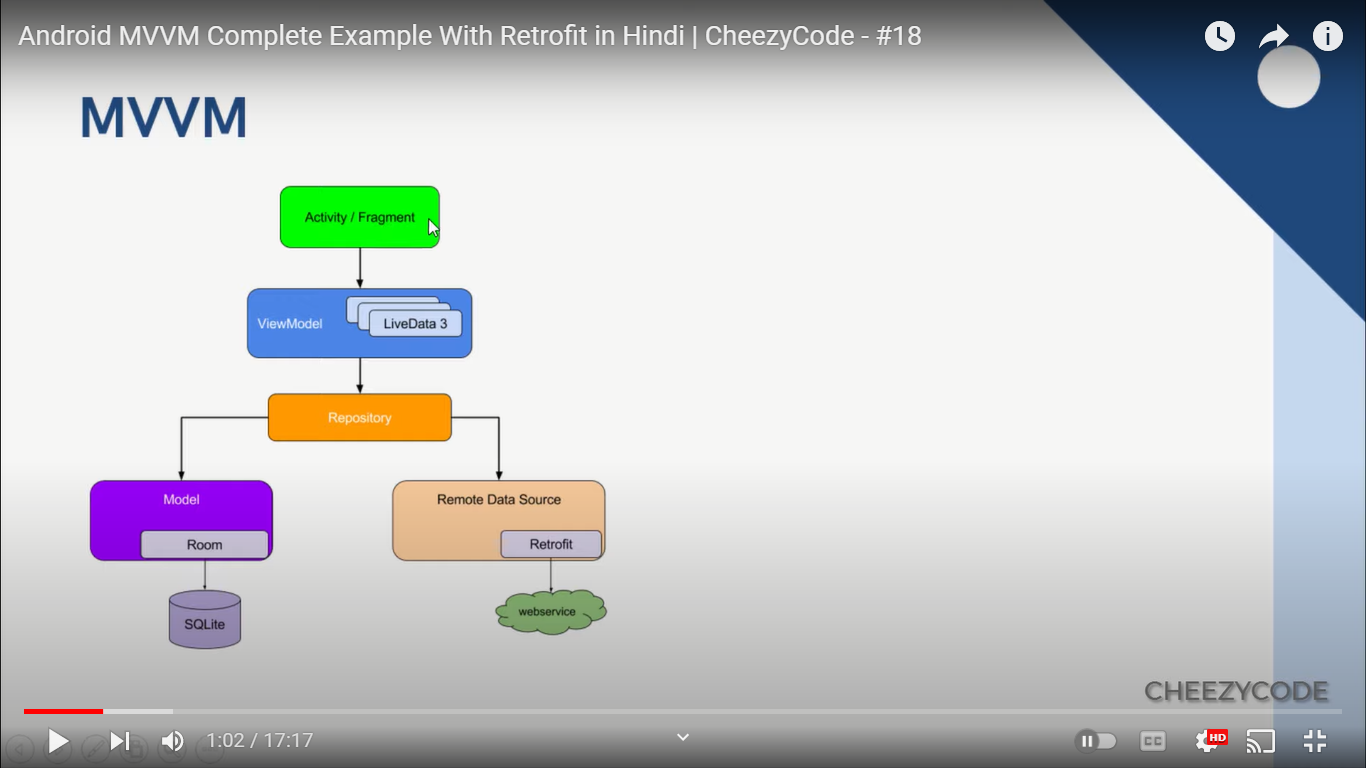
**Notes For Interview**

**MVVM**

MVVM with Retrofit example :

https://www.howtodoandroid.com/mvvm-retrofit-recyclerview-kotlin/?unapproved=2712&moderation-hash=ce16e7d96d0622b1d40853bffe40e05c&comment=2712#comment-2712



Interview Questions for Fresher to Experienced :

<https://www.interviewbit.com/android-interview-questions/>

Projects with code :

<https://www.interviewbit.com/blog/android-projects/>

Skills of Android Developer :

<https://www.interviewbit.com/blog/android-developer-skills/>

Salary of Android Developer :

<https://www.interviewbit.com/blog/android-developer-salary-in-india/>

**What is MVVM :**

### Model-View-ViewModel (MVVM) is an architecture pattern in Android that separates an application's business logic from its graphical user interface (GUI).

### **Advantages of MVVM :**

* Your code is further decoupled (Code is separated in layers or modules- the biggest advantage.)
* The project is even easier to maintain.
* Your code is even more easily testable.
* New features can be added more quickly.

### **Disadvantages of MVVM :**

* It has a slightly steep learning curve. How all the layers work together may take some time to understand.
* It adds a lot of extra classes, so it’s not ideal for low-complexity projects.

### **Recommended app architecture**

**> This section demonstrates how to structure your app following recommended best practices**.

Considering the common architectural principles mentioned in the previous section, each application should have at least two layers:

* The UI layer that displays application data on the screen.
* The data layer that contains the business logic of your app and exposes application data.

You can add an additional layer called the domain layer to simplify and reuse the interactions between the UI and data layers.

**Modern App Architecture**

This Modern App Architecture encourages using the following techniques, among others:

* A reactive and layered architecture.
* Unidirectional Data Flow (UDF) in all layers of the app.
* A UI layer with state holders to manage the complexity of the UI.
* Coroutines and flows.
* Dependency injection best practices.

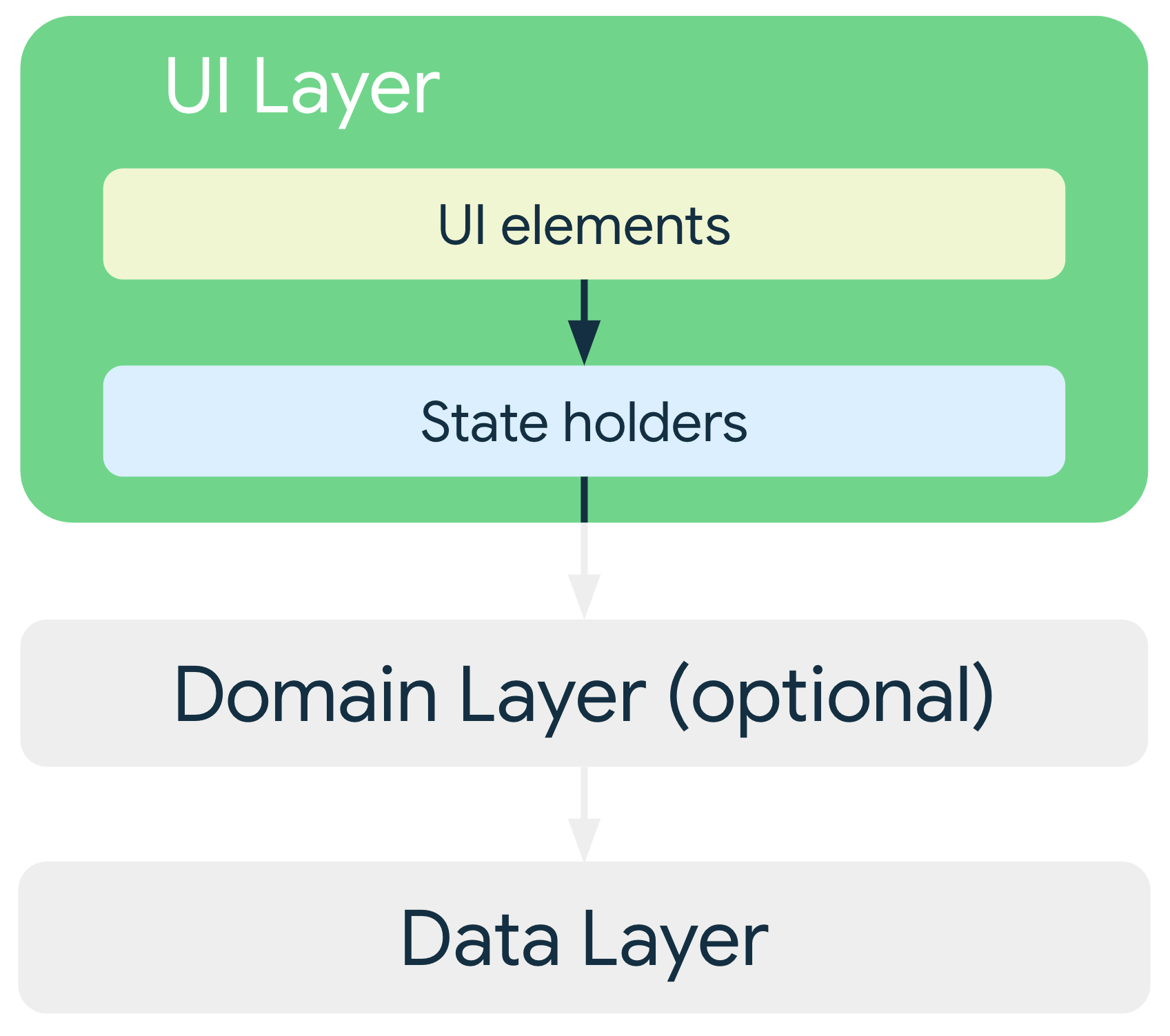
For more information, see the following sections, the other Architecture pages in the table of contents, and the [recommendations page](https://developer.android.com/topic/architecture/recommendations) that contains a summary of the most important best practices.

### **UI layer**

The role of the UI layer (or presentation layer) is to display the application data on the screen. Whenever the data changes, either due to user interaction (such as pressing a button) or external input (such as a network response), the UI should update to reflect the changes.

The UI layer is made up of two things:

* UI elements that render the data on the screen. You build these elements using Views or [Jetpack Compose](https://developer.android.com/jetpack/compose) functions.
* State holders (such as [ViewModel](https://developer.android.com/topic/libraries/architecture/viewmodel) classes) that hold data, expose it to the UI, and handle logic.



To learn more about this layer, see the [UI layer page](https://developer.android.com/jetpack/guide/ui-layer).

### **Data layer**

The data layer of an app contains the business logic. The business logic is what gives value to your app—it's made of rules that determine how your app creates, stores, and changes data.

The data layer is made of repositories that each can contain zero to many data sources. You should create a repository class for each different type of data you handle in your app. For example, you might create a MoviesRepository class for data related to movies, or a PaymentsRepository class for data related to payments.

Repository classes are responsible for the following tasks:

* Exposing data to the rest of the app.
* Centralizing changes to the data.
* Resolving conflicts between multiple data sources.
* Abstracting sources of data from the rest of the app.
* Containing business logic.

Each data source class should have the responsibility of working with only one source of data, which can be a file, a network source, or a local database. Data source classes are the bridge between the application and the system for data operations.

To learn more about this layer, see the [data layer page](https://developer.android.com/jetpack/guide/data-layer).

### **Domain layer**

The domain layer is an optional layer that sits between the UI and data layers.

The domain layer is responsible for encapsulating complex business logic, or simple business logic that is reused by multiple ViewModels. This layer is optional because not all apps will have these requirements. You should use it only when needed—for example, to handle complexity or favor reusability.

Classes in this layer are commonly called use cases or interactors. Each use case should have responsibility over a single functionality. For example, your app could have a GetTimeZoneUseCase class if multiple ViewModels rely on time zones to display the proper message on the screen.

To learn more about this layer, see the [domain layer page](https://developer.android.com/jetpack/guide/domain-layer).

**What is ViewModel :**

It acts as a link between the Model and the View. It’s responsible for transforming the data from the Model. It provides data streams to the View. It also uses hooks or callbacks to update the View. It’ll ask for the data from the Model.

ViewModel is a class that is responsible for preparing and managing the data for an [Activity](https://developer.android.com/reference/android/app/Activity.html) or a [Fragment](https://developer.android.com/reference/android/support/v4/app/Fragment). It also handles the communication of the Activity / Fragment with the rest of the application (e.g. calling the business logic classes).

A ViewModel is always created in association with a scope (an fragment or an activity) and will be retained as long as the scope is alive. E.g. if it is an Activity, until it is finished.

In other words, this means that a ViewModel will not be destroyed if its owner is destroyed for a configuration change (e.g. rotation). The new instance of the owner will just re-connected to the existing ViewModel.

The purpose of the ViewModel is to acquire and keep the information that is necessary for an Activity or a Fragment. The Activity or the Fragment should be able to observe changes in the ViewModel. ViewModels usually expose this information via [LiveData](https://developer.android.com/reference/android/arch/lifecycle/LiveData) or Android Data Binding. You can also use any observability construct from you favorite framework.

ViewModel's only responsibility is to manage the data for the UI. It should never access your view hierarchy or hold a reference back to the Activity or the Fragment.

**What is Live Data in kotlin ?**

- It is an observable data holder class. Whenever any changes in data, it update us about the changes.  
- It is life cycle aware – means the change in data is only available to those components (activity, fragments etc) which are in active state.  
- MutableLiveData – We can make changes in data.

- If an activity or fragment is recreated due to a configuration change, like device rotation, it immediately receives the latest available data.

**What is repository ?**

- A repository class isolates data sources, such as a Room database and web services, from the rest of the app. The repository class provides a clean API for data access to the rest of the app. Using repositories is a recommended best practice for code separation and architecture.

You should make one repository for each model (data).

**MVVM, MVC, MVP**

**Source**

- <https://www.youtube.com/watch?v=97BRLkicQd0> – By Cheezy code.

**Need of Architecture :**

- Scalability (we can easily add new features) & Maintainability (we can easily do changes in existing features) of an application.

- We can easily do unit testing.

- If the project is large and complex, we can use MVVM or MVP, if project is small and simple we can use MVC.

**> MVC**

- This is a default architecture of our android project.(when we create project, it will be created in this architecture).

- Controller is tightly coupled with View(it can directly exchange data).

**View -------------------- Controller ---------------------- Model**

**(Layout files- XML) (Activity/Fragment) (Data class, POJO etc.)**

**> MVP**

**View -------------------- Presenter ---------------------- Model**

**(Layout files- XML, (Simple class with (Data class, POJO etc.)**

**Activity/Fragment) (presentation logic)**

- Presenter is not tightly coupled with View (presenter will access view with **Interfaces**).

**> MVVM**

- Google recomments MVVM, whether the app is small or large.

**View -------------------- View Model ---------------------- Repository ---------------API or Database**

**(Layout files- XML, (Simple class with**

**Activity/Fragment) (presentation logic)**

- All four components are not tightly coupled. (communication between View & ViewModel is done through **Observables**).

**Coroutines**

<https://flexiple.com/android/using-kotlin-coroutine-builders-in-android/> - can refer.

<https://www.youtube.com/watch?v=Mi5DEm45XlM&list=PLRKyZvuMYSIN-P6oJDEu3zGLl5UQNvx9y> – Series By Cheezy Code- can refer.

<https://www.youtube.com/watch?v=ShNhJ3wMpvQ&list=PLQkwcJG4YTCQcFEPuYGuv54nYai_lwil_> - Series By Phillip Lackner- For Practical examples.

**What is Coroutines :**

Coroutines are lightweight threads. By lightweight, it means that creating coroutines doesn’t allocate new threads. Instead, they use predefined thread pools and smart scheduling for the purpose of which task to execute next and which tasks later.

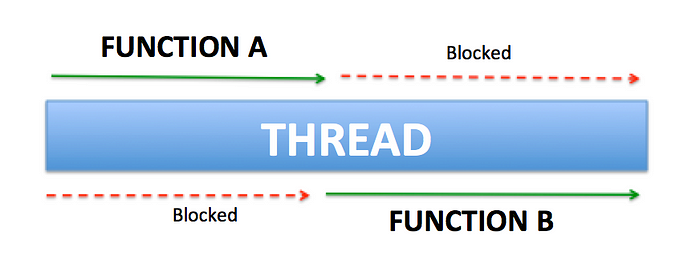
**What is Suspend Function :**

suspend function is a function that could be started, paused and resume, (and pause and resume…. if wanted repeatedly) and then end.

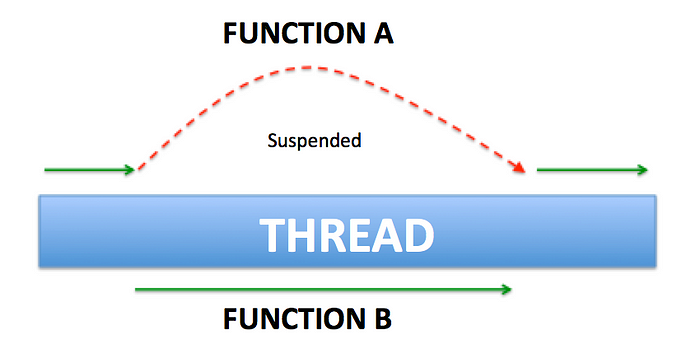
We can use suspend function to execute long-running operations and wait for them to complete without blocking main thread.

**Illustrating in Diagram**

From the cue above, I could imagine the difference between *Blocking* and *Suspending* as below



BLOCKING: Function A has to be completed before Function B continues. The thread is locked for Function A to complete its execution.



SUSPENDING: Function A, while has started, could be suspended, and let Function B execute, then only resume later. The thread is not locked by Function A.

**Difference between coroutines and threads :**

* Coroutines are executed within a thread. We can start many coroutines in single thread.
* Coroutines are suspendable.
* They can switch their context. (Can switch to another thread).
* Coroutines are light-weight, fast and works asynchronously.
* Threads are **blocking**. If you are performing a task as simple as delaying the execution for a second (Sleep), Thread would be blocked and can not be used for any other operation.
* Threads are **not lifecycle aware**. They do not have any knowledge of Lifecycle components (Activity, Fragment, ViewModel). A thread will be running even if UI component is destroyed which requires us to handle clean up and memory leaks.

**Scopes in Coroutines :**

<https://www.geeksforgeeks.org/scopes-in-kotlin-coroutines/>

Scope in Kotlin’s coroutines can be defined as the restrictions within which the [Kotlin coroutines](https://www.geeksforgeeks.org/kotlin-coroutines-on-android/) are being executed. Scopes help to predict the lifecycle of the coroutines. There are basically**3 scopes** in Kotlin coroutines:

1. Global Scope
2. LifeCycle Scope
3. ViewModel Scope
4. **Global Scope :**

When Coroutines are launched within the global scope, they live long as the application does. If the coroutines finish it’s a job, it will be destroyed and will not keep alive until the application dies, but let’s imagine a situation when the coroutines has some work or instruction left to do, and suddenly we end the application, then the coroutines will also die, as the maximum lifetime of the coroutine is equal to the lifetime of the application. Coroutines launched in the global scope will be launched in a separate thread.

**Problem with Global Scope** - The main problem with the coroutines launched in the global scope is that when the activity in which coroutines is launched dies, the coroutines will not die with the activity, since the lifetime of coroutines is decided on the basis of application lifetime, not the activity lifetime. **Since the coroutine is using the resources of the activity in which it is launched, and now since that activity has died, the resources will not be garbage collected as a coroutine is still referring to that resources. This problem can lead to a memory leak**. So using global scope all the time is not always a good idea.

1. **Lifecycle Scope :**

When we use the lifecycle scope, all the coroutines launched within the activity also dies when the activity dies.

1. **ViewModel Scope :**

coroutine in this scope will live as long the view model is alive. ViewModel is a class that manages and stores the UI-related data by following the principles of the [lifecycle system in android.](https://www.geeksforgeeks.org/activity-lifecycle-in-android-with-demo-app/)

**Coroutine context (Dispatchers) :**

<https://www.geeksforgeeks.org/dispatchers-in-kotlin-coroutines/>

Dispatchers help [coroutines](https://www.geeksforgeeks.org/kotlin-coroutines-on-android/) in deciding the thread on which the work has to be done. Dispatchers are passed as the arguments to the **GlobalScope** by mentioning which type of dispatchers we can use depending on the work that we want the coroutine to do.

### Types of Dispatchers

There are majorly 4 types of Dispatchers.

1. **Main  Dispatcher**
2. **IO Dispatcher**
3. **Default Dispatcher**
4. **Unconfined Dispatcher**

#### **Main Dispatcher:**

It starts the coroutine in the main thread. It is mostly used when we need to perform the UI operations within the coroutine, as UI can only be changed from the main thread(also called the UI thread).

1. **IO Dispatcher :**

It starts the coroutine in the IO thread, it is used to perform all the data operations such as networking, reading, or writing from the database, reading, or writing to the files eg: Fetching data from the database is an IO operation, which is done on the IO thread.

1. **Default Dispatcher**

It starts the coroutine in the Default Thread. We should choose this when we are planning to do Complex and long-running calculations, which can block the main thread and freeze the UI eg: Suppose we need to do the 10,000 calculations and we are doing all these calculations on the UI thread ie main thread, and if we wait for the result or 10,000 calculations, till that time our main thread would be blocked, and our UI will be frozen, leading to poor user experience. So in this case we need to use the Default Thread. The default dispatcher that is used when coroutines are launched in GlobalScope is represented by Dispatchers. Default and **uses a shared background pool of threads**, so launch(Dispatchers.Default) { … } uses the same dispatcher as GlobalScope.launch { … }.

If no dispatcher nor any other ContinuationInterceptor is specified in their context, Default dispatcher is used.

1. **Unconfined Dispatcher :**

As the name suggests unconfined dispatcher is not confined to any specific thread. It executes the initial continuation of a coroutine in the current call-frame and lets the coroutine resume in whatever thread that is used by the corresponding suspending function, without mandating any specific threading policy.

**Coroutine Builders :**

[**https://flexiple.com/android/using-kotlin-coroutine-builders-in-android**](https://flexiple.com/android/using-kotlin-coroutine-builders-in-android)

As the name suggests, coroutine builders are a way of creating coroutines. Since they are not suspending themselves, they can be called from non-suspending code or any other piece of code. They act as a link between the suspending and non-suspending parts of our code.

In the following sections, we will be discussing the following three Kotlin coroutine builders:

1. runBlocking

2. launch

3. async

## 1) runBlocking

As the name suggests, runBlocking is a coroutine builder that blocks the current thread until all tasks of the coroutine it creates, finish. So, why do we need runBlocking when we are clearly focusing to avoid blocking the main thread?

We typically runBlocking to run tests on suspending functions. While running tests, we want to make sure not to finish the test while we are doing heavy work in test suspend functions.

## 2) launch

launch is essentially a Kotlin coroutine builder that is “fire and forget”. This means that launch creates a new coroutine that won’t return any result to the caller. It also allows to start a coroutine in the background.

3) async

We now move on to the best part of Kotlin Coroutines builders – async.

async is a coroutine builder which returns some value to the caller. async can be used to perform an asynchronous task which returns a value. This value in Kotlin terms is a Deferred<T> value.

We can simply use async to create a coroutine doing some heavy work like so:

**async-await**

await() is a suspending function that is called upon the async builder to fetch the value of the Deferred object that is returned. The coroutine started by async will be suspended until the result is ready. When the result is ready, it is returned and the coroutine resumes.

Now, coming to the best part about using Kotlin Coroutines with async. We can use async coroutine builder to call multiple pieces of asynchronous code in parallel. We can then combine the results from both to display a combined result. This leads to lower latency and in extension, a superior user experience.

However, with async and await, we save considerable amount of time. We make asynchronous calls to both functions in parallel and wait for each function to return a value before showing the result.

**Job in Coroutines :**

A Job in Kotlin coroutines represents a cancellable piece of work. Every coroutine has an associated job that you can use to manage its lifecycle. When you create a coroutine using a coroutine builder like launch or async, it returns a Job instance. You can then use this Job to cancel the coroutine, await its completion, or even combine it with other jobs to create a hierarchy.

To know more explanation (Join(), Cancel()), see the following link :

<https://www.geeksforgeeks.org/jobs-waiting-cancellation-in-kotlin-coroutines/>

**Failure response handling with coroutines in retrofit**:

*lifecycleScope*.*launch*(Dispatchers.IO) **{**

val result = retrofitInterface.getFollowing(authToken)

if (result.*isSuccessful*) {

Log.e("COR", result.body().*toString*())

} else {

val errorObj = JSONObject(result.errorBody()!!.charStream().*readText*()).getString("errorMessage")

// Above line is to get the error message from result.errorBody().

// Here "errorMessage" is key of response in postman. It may differ according to response.

Log.e("COR-Error", errorObj)

}

}

**General**

**Null Safety in Kotlin :**

* What exactly Kotlin says is ‘Nulls in Kotlin don’t exist until you say’, that is no variable by default is null.
* We just differentiate between Null and Non-Null object types in Kotlin and have to explicitly handle it as we used to do with a variable in Java.
* Kotlin by default forces to initialize every object with a value and that should not be null.  
  var showButton: Button = null //Compile Time Error
* However, we can make a nullable type object, by explicitly informing Kotlin that the object can be null.  
  var showButton: Button? = null // No Compile Time Error
* There might be cases when you know that your object cannot be null, then we use Not Null Operator !! to assert that object is not null for sure and hence avoiding the null check. (I suggest to use this rarely since if you are using this operator you are actually doing the same things as we used to do in Java)
* The not-null assertion operator (!!) converts any value to a non-null type and throws an exception if the value is null. We can write x!!, and this will return a non-null value of x or throw an NPE if x is null.

**High level functions :**

- <https://www.geeksforgeeks.org/kotlin-higher-order-functions/>

High level functions :

* Can accept functions as parameters.
* Can return a function
* Or can do both

**Lambdas :**

* It is just a function with no name.
* (function) { s (parameter) -> println(s) (Body) }
* The full syntactic form of lambda expressions is as follows:
* val sum: (Int, Int) -> Int = { x: Int, y: Int -> x + y }
* If you leave all the optional annotations out, what's left looks like this:
* val sum = { x: Int, y: Int -> x + y }
* call- println(sum(4,5))
* <https://www.youtube.com/watch?v=_GogUOLqc64>

**Parcelable & Serializable in Kotlin :**

* In some cases, we have to pass complex POJO class objects (e.x. Student, Employee, Vehicle etc.) to another activity. In this scenario, we have to do some extra stuff with the POJO class that makes object to be easily transferrable from one activity to another. Parcelable and Serializable interfaces used for this. See the link below for more explanation.
* <https://medium.com/@shibuprasad008/serializable-vs-parcelable-in-android-773ed4330401>

**Start-O-Preneur for Interview :**

- TestArticleFragment.kt and TestNewsPagerAdapter.kt is made for Article view. Both are accessible from ProductivityFragment.kt. Sometimes app crashes on click of any item in Productivity tab.

- SearchFragment is activated from bottom\_nav.xml.

**Types of Links in Android**

There are 3 types of Links in Android :

<https://developer.android.com/training/app-links>

1. Deep links
2. Web links
3. App links

**Deep links :**

[**https://developer.android.com/guide/navigation/design/deep-link#:~:text=In%20Android%2C%20a%20deep%20link,deep%20links%3A%20explicit%20and%20implicit**](https://developer.android.com/guide/navigation/design/deep-link#:~:text=In%20Android%2C%20a%20deep%20link,deep%20links%3A%20explicit%20and%20implicit)

- In Android, a deep link is a link that takes you directly to a specific destination within an app.

- The Navigation component lets you create two different types of deep links: explicit and implicit.

- There are 2 types of Deep links :

1) Implicit Deep Link :

An [implicit deep link](https://developer.android.com/training/app-links/deep-linking) refers to a specific destination **(static- destination**) in an app. When the deep link is invoked—for example, when a user clicks a link—Android can then open your app to the corresponding destination.

2) Explicit Deep Link :

An [explicit deep link](https://developer.android.com/training/app-links/deep-linking) is a single instance of a deep link that uses a [PendingIntent](https://developer.android.com/reference/android/app/PendingIntent) to take users to a specific location(**destination is dynamic**) within your app. You might surface an explicit deep link as part of a notification or an app widget, for example.

**Push Notifications using FCM API- 17.2.24**

**- Resources used :**

- <https://www.youtube.com/watch?v=wqUi6tzN76c>

- Yorder-Customer app for sending notifications to a perticular user.

- Paw Garage app for sending notifications to topic.

**Steps :**

- First connect your project to firebase and add firebase-messaging sdk as per below link:

<https://firebase.google.com/docs/cloud-messaging/android/first-message>

- We need server key from Firebase console -> Project Settings -> Cloud Messaging -> Copy Server Key.

- We need to pass this Server Key as Authorization Header in FCM Api call as “key=SERVER\_KEY”

- Then see the link- <https://firebase.google.com/docs/cloud-messaging/android/client>

- Do necessary changes in Manifest.

- Request runtime permission for POST\_NOTIFICATIONS for Android 13(API level 33 or higher) and above devices. Below android 13, no permission needed.

- If we need to send notifications to **a perticular user or device**, we nedd to make a class which extends

FirebaseMessagingService() class. And add the <service/> in Manifest as described in docs. In this class

We should override onNewToken() and onMessageReceived() methods. A new token is generated in onNewToken() method, whenever we first install the app. We should save this token shared pref. When user sign in, we should add this token to firestore in user table. When user sign out, we should remove this token from firestore. When we need to send notification, first we need to get token from database for perticular user based on userId and then send notification to this token. See Yorder-Customer app, SellersFragment.kt for reference.

- If we need to send notifications **to topic**, we need to subscribeToTopic in Login or VerifyOtp fragment and unsubscribeToTopic when user logout. We can send the notification as per AddAdmissionFragment.kt in Paw Garage.

- If the app is in background, the notification is handledby the device’s system tray. A user tap on a notification opens the app launcher by default.

- **Messages with both notification and data payload, when received in the background**. In this case, the notification is delivered to the device’s system tray, and the data payload is delivered in the extras of the intent of your launcher Activity.

- If the app is in foreground, notification is handled by onMessageReceived method in FirebaseMessagingService() class.

**Authentication using Firebase**

**General Steps :**

- First add firebase to your android project as per below link :

<https://firebase.google.com/docs/android/setup>

- Then add firebase authentication to your app ad per following link :

<https://firebase.google.com/docs/auth/android/start>

**Phone number (OTP) authentication :**

Firebase link - <https://firebase.google.com/docs/auth/android/phone-auth>

Video Link - <https://www.youtube.com/watch?v=zKa14ULHGBQ>

App – Paw Garage App.

- After doing above two general steps, Enable Phone-Number sign in for your firebase project by :

Click on your project on console -> Authentication -> Sign-in Method -> Select Phone.

- SHA-1 & SHA-256 keys are required for phone anthentication. Obtain SHA-1 & SHA-256 keys from android studio and paste it to firebase project at below place :

Go to your project on console -> Project Settings -> Add fingerprints -> Paste SHA-1 & SHA-256 keys here-> Save.

- Now download google-services.json file and paste it in android studio project.

- Now in Login screen we can present the UI to enter the phone number. And Send OTP button.

- In onClick of Send OTP button, we can write the code mentioned in “3. Send verification code to user’s phone” in above Firebase Link.

- In onVerificationCompleted() method, we can make method,

signInWithPhoneAuthCredential(credential). After successful sign in in this method, we can navigate user to HomeFragment.

- In onCodeSent() method, we can navigate the user to VerifyOTPScreen with verificationId, token and phoneNumber.

- In onClick of verifyOTP button, we will get the entered OTP by user from otpView. We will make credential variable from verificationId and OTP. And we will call signInWithPhoneAuthCredential(credential).

- We will implement OTPListener interface in this screen. onOTPComplete() of this interface will fire when we complete typing of OTP. We will make credential variable from verificationId and OTP. And we will call signInWithPhoneAuthCredential(credential).

- OnClick of ResendOTP, we can repeat the code mentioned in “3. Send verification code to user’s phone” in above Firebase Link. And do same as related steps and sign in.

**Email & Password authentication – Login & Register :**

Firebase link - <https://firebase.google.com/docs/auth/android/password-auth>

Video Link - <https://www.youtube.com/watch?v=gaykE36N7PY>

App – Start-O-Preneur, Yorder-Customer.

- After doing above two general steps, Enable Email/Password sign in for your firebase project by :

Click on your project on console -> Authentication -> Sign-in Method -> Email/password.

- We can Register the user with following method :

auth.createUserWithEmailAndPassword(email, password) { }

- After successful registration, we can store the user details in firestore database. And navigate user to HomeScreen.

- We can Login the user with following method :

auth.signInWithEmailAndPassword(email, password) { }

- After successful login, we can navigate user to HomeScreen.

**Forgot Password (Reset Password) authentication :**

Source link – <https://firebase.google.com/docs/auth/android/manage-users>

<https://itnext.io/android-firebase-authentication-email-and-password-login-b06980cf864a>

App – Start-O-Preneur, Yorder-Customer

- After doing above two general steps, we can send password reset email as per below :

Firebase.auth.sendPasswordResetEmail(email)

**Update Password, Update Email, Delete Account in Firebase authentication :**

See the methods in below link :

- <https://firebase.google.com/docs/auth/android/manage-users>

- <https://itnext.io/android-firebase-authentication-email-and-password-login-b06980cf864a>

**Authentication using Google SignIn :**

Firebase link - <https://firebase.google.com/docs/auth/android/google-signin>

Video Link - <https://www.youtube.com/watch?v=_318sOlkJBQ>

App – Start-O-Preneur.

- After doing above two general steps, also add the dependency for the Google Play services library and specify its version as below :  
    implementation("com.google.android.gms:play-services-auth:21.0.0")

- Enable Google sign in for your firebase project by :

Click on your project on console -> Authentication -> Sign-in Method -> Google.

- SHA-1 & SHA-256 keys are required for google anthentication. Obtain SHA-1 & SHA-256 keys from android studio and paste it to firebase project at below place :

Go to your project on console -> Project Settings -> Add fingerprints -> Paste SHA-1 & SHA-256 keys here-> Save.

- Now download google-services.json file and paste it in android studio project.

- I have not followed the code, which is mentioned in above firebase link.

- I have followed the code mentioned in Video Link.

- See LoginFragment.kt in Start-O-Preneur for the code.

- In addOnCompleteListener of in mAuth.signInWithCredential(credential) { } method, we can add the user details in firestore database and navigate the user to HomeFragment.

**Implement Google Map in Android Studio**

Video Link followed for below steps - <https://www.youtube.com/watch?v=_gpreGNtNCM>

Can Refer (little confusing) - <https://developers.google.com/maps/documentation/android-sdk/start>

**Steps :**

- To enable the Maps in android, we need API\_KEY.

- Go to console.cloud.google.com -> Create New Project -> Enter billing details, if needed -> Give Project Name -> Location- No Organization -> Create Project.

- Select the project -> Go to APIs Overview -> Credentials (on left column) -> Create Credentials (on top)

-> API Key -> Copy your API key (You can see API key on Credentials menu also).

- Once you have your API key (it starts with "AIza"), define a new property in your  
project's local.properties file (e.g. MAPS\_API\_KEY=Aiza...), and replace the  
"YOUR\_API\_KEY" string in Manifest file with "${MAPS\_API\_KEY}" as below :

- In Manifest in <application/> tag as below :

<meta-data  
 android:name="com.google.android.geo.API\_KEY"  
 android:value="${MAPS\_API\_KEY}" />

- Add the following three permissions in Manifest :

- INTERNET

- ACCESS\_FINE\_LOCATION

- ACCESS\_COURSE\_LOCATION

- Add following dependency in app level gragle file :

implementation 'com.google.android.gms:play-services-maps:18.2.0'

- We can skip adding Secrets Gradle Plugin dependency and its code.

- Now to add Maps Activity in android studio project,

Right click on package -> New -> Google -> Google Maps Views Activity.

- This will add the necessary code in MapsActivity.kt file and its XML file.

- We can use the fragment also instead of MapsActivity. For that we can copy paste above code from activity to fragment and its XML.

- Now we can run the app.

**How to get current location :**

- Source - <https://stackoverflow.com/questions/71137555/getcurrentlocation-method-in-kotlin>

- App - SellersFragment.kt in Yorder-Customer, LocationFragment.kt in Paw Garage.

fusedLocationClient.getCurrentLocation(Priority.PRIORITY\_HIGH\_ACCURACY, object : CancellationToken() {

override fun onCanceledRequested(listener: OnTokenCanceledListener) = CancellationTokenSource().token

override fun isCancellationRequested() = false

})

.addOnSuccessListener {

if (it == null)

Toast.makeText(this, "Cannot get location.", Toast.LENGTH\_SHORT).show()

else {

val lat = it.latitude

val lon = it.longitude

}

}

**How to get address from latitude & longitude :**

- App - LocationFragment.kt in Paw Garage.

private fun getAddressFromCoordinates() {  
 try {  
 val geoCoder = Geocoder(requireContext(), Locale.getDefault())  
 val addressList: List<Address>? =  
 geoCoder.getFromLocation(mLatitude, mLongitude, 1)  
  
 if (!addressList.isNullOrEmpty()) {  
 val address = addressList.first()  
  
 if (address.hasLatitude() && address.hasLongitude()) {  
  
 Log.e("FLOW", "In getAddressFromCoordinates, Lat- $mLatitude, Lng- $mLongitude")  
 Log.e("NST", address.toString())  
  
 locationText =  
 address.getAddressLine(0)  
 addMarkerAt(LatLng(address.latitude, address.longitude), address.getAddressLine(0))  
 getNearbyPlaces()  
  
 Log.e("FLOW", "Address(getAddressFromCoordinates)- $locationText")  
 }  
 } else {  
 Log.e("FLOW", "AddressList is Null or Empty")  
 Toast.makeText(requireContext(), "No address found.", Toast.LENGTH\_SHORT).show()  
 }  
 } catch (e: Exception) {  
 Toast.makeText(AppDelegate.applicationContext(), e.message, Toast.LENGTH\_SHORT).show()  
 Log.e("ERROR", e.toString())  
 }  
 }

**How to get Nearby Places in google map (LocationFragment.kt- Paw Garage) :**

https://stackoverflow.com/questions/63770890/search-nearby-places-in-android-studio-using-kotlin

https://developers.google.com/maps/documentation/places/web-service/search-nearby

- We also need API\_KEY(May be from PlacesAPI) in above.

**How to get searched places(location from name) on google map (LocationFragment.kt- Paw Garage) :**

https://www.geeksforgeeks.org/how-to-add-searchview-in-google-maps-in-android/

- We also need API\_KEY(from Maps SDK for Android) in above.

**Publish App on Play Store**

**Sources**

- Create signed apk - <https://www.youtube.com/watch?v=92sKMUmv_wo>

- <https://www.youtube.com/watch?v=Ormjb-BX1sw> (Publish App)

- New policy(20 testers) explaination - <https://www.youtube.com/watch?v=bzhmgkuR0dU>

- <https://www.youtube.com/watch?v=L9sU_aGeQEY> (Publish app As per new 20 testers rule).

- https://copperdigital.com/blog/a-step-by-step-guide-to-upload-app-to-google-play-store/

**New Rules**

- Devwlopers, who has created new personal developer accounts after 13, Nov-23, the app should be tested by min 20 testers for 14 days before publishing it to play store.

**How to crete signed apk :**

Purpose – It is a digital signature of our app, that is used when we update our app to identify our app.

- Open your project in Android Studio -> Build -> Generate signed Bundle/APK -> Click on APK

-> Next -> Key store path -> Click on Create New or Choose Exixting(in case of app update)

-> Select any path on your PC -> Enter any Password and Confirm Password -> Keep Alias as it is

-> Enter password and confirm password fo this also. -> Validity(years) – you can enter 200 years

Also -> Fill other general details like name etc and click Ok. -> Click Next -> Select Release

-> Tick in all Signature Versions -> Click Finish. -> You can see ‘locate’ at bottom-right at studio,

Click on it -> Open ‘release’ folder and you can find app-release.apk.

**Steps**

# Search in google- google play console- and open it.

# First we need to create developer account. One time charge is 25$ for 25 years.

# After log in with your developer account, play store screen will open.

# In **All Apps** (on top-left) screen, click a button- Create App (top-right).

# Then fill the **App Details** like App name, Default Language, App or Game, Free or Paid (In case of Paid you have to create your Merchant account, in which the money will come).

# Then fill the **Declarations** details.

# Then click on **Create App** button at bottom-right.

# In **Set up your app**, you have to fill following details :

**> Set Privacy Policy** – Paste here privacy policy url.

**>** **App Access** – If parts of yout app are resricted by login credentials etc. or not.

**> Ads** – Contains ads ot not.

**> Content Rating**- Email, Category, Violence, Sexuality(any sexual material), Language (offensive language), Controlled Substance (illegal drugs), Miscellaneous. And press submit at bottom-left.

**> Target audience** – Generally mark 13 yrs and above. If below we have to submit family policy.

**> News App** – Tick Yes or No.

**> Data Safety** – Is your app collects user data and share with third party ?,

**> Government App** – Yes or No.

**>** **Select an app category and provide contact details** :

- App category(Educational, Social media etc), Email(mandatory), Phone number, Website.

**>** **Set up your store listing :**

- App name, Short description, Full description,

- **Graphics :**

- App Icon (Transparent PNG or JPEG, 512px X 512px, Up to 1 MB)

- Feature Graphic (PNG or JPEG, 1024px X 500px, Up to 1 MB)

- Video (Optional) – youtube video link of your app

- Phone screenshots (2-8 nos, PNG or JPEG, 16:9 or 9:16 aspect ratio, Each side between 320px and 3840 px, Up to 8 MB per screenshot).

- 7 inch tablet screenshots (Up to 8 nos, PNG or JPEG, 16:9 or 9:16 aspect ratio, Each side between 320px and 3840 px, Up to 8 MB per screenshot) – Not mandatory, we can upload phone screenshots also here.

- 10 inch tablet screenshots (Up to 8 nos, PNG or JPEG, 16:9 or 9:16 aspect ratio, Each side between 320px and 3840 px, Up to 8 MB per screenshot) ) – Not mandatory, we can upload phone screenshots also here.

# **Closed Testing (New Requirement)** :

> We have to upload our app bundle (signed apk) and provide 20 testers email ids.

> Publish the app for review.

> After necessary days for review, the app will be published for closed testing.

> Share the app link with testers :

- Go to playstore and select the app.

- Go to Close Testing from left menu and go to Testers from top menu.

- There are 2 links. You can copy from Join on Android link and share with your testers by whatsapp or email. By clicking on this link, tester is redirected to play store and can download the app.

- Testers should download the app and remain downloaded for at least 14 days.

- After 14 days you can click on Apply For Production button in Production menu.

- Some questions you have to answer and click on Apply button. This usualy takes up to 7 days. After successful apply for production, your app is ready to publish the final release. And follow the below steps.

**>** **Publish your app on google play :**

- In Close Testing menu(left), select Promote Release and select Production.

- Select Countries/Regions.

- Release details -> - Release name(version) – e.g. 1(1.0)

- Release notes – e.g. Initial release or Bug fixes etc.

- Then click on Review Release button at bottom-right.

- Then click on Start rollout to Production button at bottom-right.

- Now your app will be in review. It will generally take up to 7 days. And if fullfils all requirements of playstore, it will be live.

**Git, GitHub, SourceTree, BitBucket**

**Sources :**

- Create repository, Commit, Pull, Push, Revert commit, Create & Delete branch, Merge branch, Create Pull request - <https://www.youtube.com/watch?v=MaqVvXv6zrU>

For Theory - <https://www.javatpoint.com/git>

**What is Git :**

- **Git** is an **open-source distributed version control system software**. It tracks changes to files. It is developed to co-ordinate the work among the developers.

**Meaning of Distributed :**   
One of Git's great features is that it is distributed. Distributed means that instead of switching the project to another machine, we can create a "clone" of the entire repository. Also, instead of just having one central repository that you send changes to, every user has their own repository that contains the entire commit history of the project. We do not need to connect to the remote repository; the change is just stored on our local repository. If necessary, we can push these changes to a remote repository.

**Benefits of Git :**

A version control application allows us to **keep track** of all the changes that we make in the files of our project. Every time we make changes in files of an existing project, we can push those changes to a repository. Other developers are allowed to pull your changes from the repository and continue to work with the updates that you added to the project files.

**What is GitHub :**

- GitHub is a Git repository hosting service. GitHub also facilitates with many of its features, such as access control and collaboration. It provides a Web-based graphical interface.

**Difference between Clone and Fork the repository :**

- Clone – Creates copy of your repository to your local drive.

- Fork – Creates copy of repository on remote server (GitHub.com).

**Difference between Pull and Fetch in GitHub desktop :**

Fetch gets the latest updates from origin but doesn't update your local working copy with the changes. After you click Fetch origin, the button changes to Pull Origin. Click Pull Origin to update your local working copy with the fetched updates.

**What is GitHub desktop :**

GitHub Desktop is a Graphical User Interface (software) to create and manage a Git repository without using the command line.

**What is Source tree and BitBucket :**

Bitbucket is a web-based hosting service that allows users to host private or public repositories, while SourceTree is a client application and GUI for managing and interacting with Git and Mercurial repositories

**New features in Android 14**

Source - <https://beebom.com/best-android-14-features/>

1. Notification flashes.

2. Lockscreen Customizations

One of the major features of Android 14 is the new Lock screen customization options. As the name suggests, Android now has a bunch of lock screen clocks each with its own smooth animation to enhance the look of your home screen. You can change the way your clock looks on the lock screen and add app shortcuts that show up on the lower corners of your lock screen.

3. Magic Compose :

- Magic Compose in the Messages app can help you write your text messages in different moods.

4. Separate ring and notification volumes :

5. Drag and Drop Text and Images to Different Apps :

6. Using your Android device as a Webcam :

**New features in Android 15**

Source - <https://beebom.com/android-15-features/>

- Android 15 is an upcoming major release of the Android mobile operating system. With the release of the first developer preview in February 2024, Google expects the platform to reach beta stage in April 2024 with a final release expected in Q3 2024.

1. Screen recording for a single app :

- If user use this feature, the unwanted notifications etc during recording will not show.

2. Notification Cooldown :

-  It lowers the volume of successive notifications that come from the same app. It’s designed to prevent users from being overwhelmed by too many notifications.

3. New In-app camera controls :

- User can adjust the brightness, and flash light intensity during photo capture.

4. Universal toggle for keyboard vibration control :

5. One click bluetooth popup dialog with available devices :

- Previously we have to long press on bluetooth icon, now it is one click.

**Paging3 with MVVM & Retrofit :**

**Sources :**

https://howtodoandroid.com/pagination-with-paging-3-android/ **- As per this, done in Starto-Preneur**.

https://www.youtube.com/playlist?list=PLRKyZvuMYSIPci119n2gt\_kq1GU-PAYRk

**About Flow & Live Data**

Source - <https://medium.com/paycell-tech-team/understanding-the-differences-between-android-livedata-and-flow-92a89913d42b#:~:text=When%20it%20comes%20to%20handling,a%20part%20of%20Kotlin%20Coroutines>

- When it comes to handling asynchronous data streams in Android app development, two popular options are LiveData and Flow. LiveData has been a core part of the Android Architecture Components, while Flow was introduced as a part of Kotlin Coroutines. Both LiveData and Flow offer similar functionalities but with some key differences. In this article, we will explore the characteristics, use cases, and comparison of LiveData and Flow to help you understand when and where to use each of them in your Android projects.

# **Overview of LiveData**

LiveData is a data holder class that is lifecycle-aware, meaning it respects the lifecycle of Android components such as activities and fragments. LiveData provides observable data that can be observed by multiple observers, usually UI components, and automatically updates them when the data changes. LiveData simplifies the management of UI components, as it automatically handles subscription and unsubscription based on the lifecycle state.

# **Overview of Flow**

Flow is a new asynchronous stream processing API introduced with Kotlin Coroutines. It allows developers to emit multiple values asynchronously and provides powerful operators for transforming and combining these values. Flow is designed to work seamlessly with coroutines, providing a convenient way to handle asynchronous operations in a structured and sequential manner.

**Download Manager**

**Source**

- <https://developer.android.com/reference/android/app/DownloadManager>

- <https://www.youtube.com/watch?v=4t8EevQSYK4>

- <https://www.youtube.com/watch?v=-JKFWsPcoqs>

- The download manager is a system service that handles long-running HTTP downloads. Clients may request that a URI be downloaded to a particular destination file. The download manager will conduct the download in the background, taking care of HTTP interactions and retrying downloads after failures or across connectivity changes and system reboots.

- Apps that request downloads through this API should register (not compulsary) a broadcast receiver for [ACTION\_NOTIFICATION\_CLICKED](https://developer.android.com/reference/android/app/DownloadManager#ACTION_NOTIFICATION_CLICKED) to appropriately handle when the user clicks on a running download in a notification or from the downloads UI.

- Note that the application must have the [Manifest.permission.INTERNET](https://developer.android.com/reference/android/Manifest.permission#INTERNET) permission to use this class.

**View Pager2**

**Source**

<https://developer.android.com/develop/ui/views/animations/screen-slide-2>

<https://www.youtube.com/watch?v=N8ySWQtDD24>

- **Tab Layout with ViewPager2 in Kotlin**

- <https://medium.com/busoft/how-to-use-viewpager2-with-tablayout-in-android-eaf5b810ef7c>

- https://www.youtube.com/watch?v=SpNdUI4jqOA

- ViewPager2 objects have built-in swipe gestures to transition through pages, and they display screen slide animations by default, so you don't need to create your own animation. ViewPager2 uses[**FragmentStateAdapter**](https://developer.android.com/reference/kotlin/androidx/viewpager2/adapter/FragmentStateAdapter) objects as a supply for new pages to display, so the FragmentStateAdapter uses the fragment class that you created.

**Bottom Seets**

**Sources**

- <https://m3.material.io/components/bottom-sheets/overview>

- <https://www.simplifiedcoding.net/bottom-sheet-android/>

- Bottom sheets show secondary content anchored to the bottom of the screen.

## Types of Bottom Sheet

### #1 Persistent Bottom Sheet

As clear from the name, it is persistent (will not hide) at the bottom of the screen. A user can view the full Bottom Sheet by dragging the sheet up vertically. The Bottom Sheet is slightly elevated, and it can display more options or app content to the user. For example, the picture that we saw above is an example of a persistent bottom sheet. **The screen behind bottomsheet is accessible.**

### #2 Modal Bottom Sheets

Again as the name suggests, these sheets behave like Modals or dialogues**. It shadows the activity or fragment when activated.** And if we tap outside the Bottom Sheet, it is dismissed just like a modal. A user can also slide up and slide down to activate and deactivate the Bottom Sheet, respectively.

**Parallax Effect**

**Sources**

**Collapsing toolbar with Parallax effect ( Best tutorial with different mode effects ) :**

<https://www.youtube.com/watch?v=6UmHGn076To>

Can also read for Parallax effect :

<https://en.proft.me/2017/04/23/how-create-parallax-scrolling-android/>

<https://blog.mindorks.com/parallax-effect-in-android>

**Work Manager**

**Sources**

- <https://developer.android.com/topic/libraries/architecture/workmanager>

- <https://www.youtube.com/watch?v=sPPQ8lTsdhM&t=15s> by Cheezy Code.

- <https://www.youtube.com/watch?v=prVmo-IMmys> – Can refer

> [WorkManager](https://developer.android.com/reference/androidx/work/WorkManager) is the recommended solution for persistent work. Work is persistent when it remains scheduled through app restarts and system reboots. Because most background processing is best accomplished through persistent work, WorkManager is the primary recommended API for background processing.

**> Types of persistent work :**

WorkManager handles three types of persistent work:

* **Immediate**: Tasks that must begin immediately and complete soon. May be expedited.
* **Long Running**: Tasks which might run for longer, potentially longer than 10 minutes.
* **Deferrable**: Scheduled tasks that start at a later time and can run periodically.

**> Benefits of Work Manager :**

**- Work constraints -**  For example, run only when the device is on an unmetered network, when the device is idle, or when it has sufficient battery.

- Robust scheduling

- Flexible retry policy

- Work chaining

**Architecture Components in Android**

- <https://www.geeksforgeeks.org/jetpack-architecture-components-in-android/>

- <https://www.kodeco.com/books/advanced-android-app-architecture/v1.0/chapters/4-android-architecture-components>

- To remember – Rawalpindi - RaWVaLLPiNDi (Room, WorkManager, ViewModel, LiveData, Lifecycle, Paging, Navigation, DataBinding).

**Difference between Canary, Beta, RC and Stable Releases in Android Studio**

**Source** - <https://medium.com/android-news/what-is-the-difference-between-canary-beta-rc-and-stable-releases-in-the-android-studio-bbbb77e7c3cf>

So now let’s focus on the different releases. To explain that I will use Android Studio 3.0 as an example. This version of IDE was released in the following releases:

## 1.) Canary release

Android Studio 3.0 began its journey as the canary release, because just like I have mentioned before canary releases’ intention is to showcase the new features. Although these builds are tested, they are still very unstable. 3.0 version of the Android Studio had 9 canary releases. Every canary release almost always provides some bug fixes or sometimes some new features. Documentation of the last canary release can be accessed [here](https://androidstudio.googleblog.com/2017/08/android-studio-30-canary-9.html).

## 2.) Beta release

IDE went through its canary releases and showed the new features to the world. It had gotten some bug fixes and then the next step was the beta release. Beta release is usually more stable and usable, but it is still totally normal to experience some bugs. Android Studio 3.0 had 7 beta releases and it was slowly being polished to the final product. If you are interested in what the last beta release introduced, you can access the documentation [here](https://androidstudio.googleblog.com/2017/10/android-studio-30-beta-7-is-now.html).

## 3.) RC release

RC stands for release candidate and it is also known as “going silver.” It is the last step before the stable release. At this stage Android Studio 3.0 was ready for use on a production site, but it had still not been marked as a stable version just in case if more bugs appeared. Android Studio 3.0 had 2 RC versions and only general bug fixes were introduced. The last RC release’s documentation can be found [here](https://androidstudio.googleblog.com/2017/10/android-studio-30-rc-2-now-available.html).

## 4.) Stable release

Stable release is, as you might have guessed it, also known as “going gold.” This is the release most people are going to use, because hopefully it doesn’t have annoying bugs, it is performant and reliable. Link to the documentation is [here](https://androidstudio.googleblog.com/2017/10/android-studio-30-now-available.html).

**Android Studio Version Release Updates :**

- <https://androidstudio.googleblog.com/>