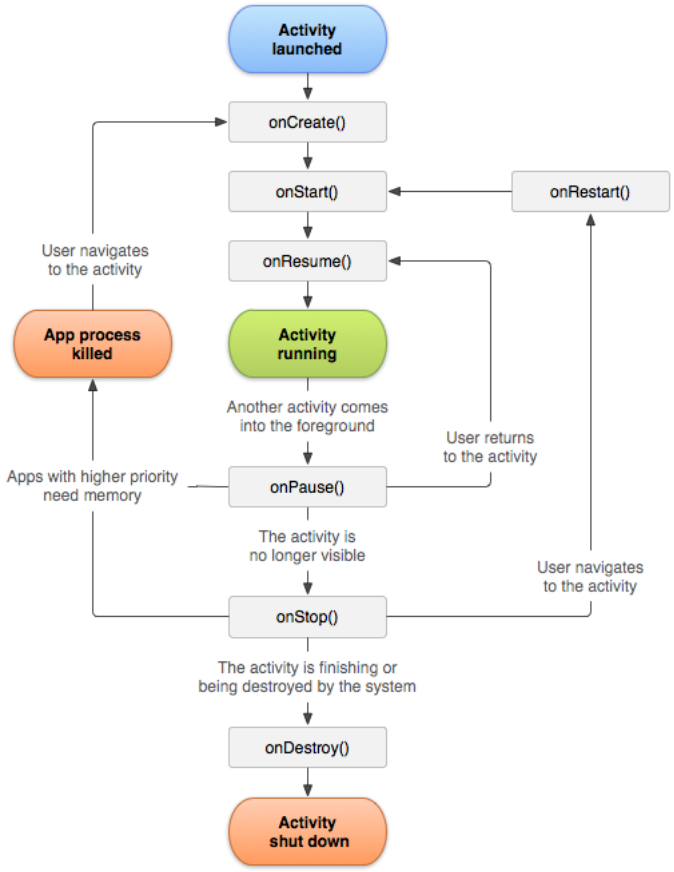
Manifest (AndroidManifest.xml): describes essential information about your app to the Android build tools, the Android operating system, and Google Play. Contains the declaration of components of the app, permissions and HW and SW features that the app requires. Included in the APK generated in the build process.

Resources: additional files and static content that your code uses; facilita a adequação da app a conf do dispositivo.

Activity: serves as the entry point for an app's interaction with the user; execution host; provides the window in which the app draws its UI; declared on the manifest file; In the mobile-app experience, the user journey often begins non-deterministically;



Intent: a messaging object you can use to request an action from another app component; There are three fundamental use-cases for intents: Starting an activity, starting a service, or delivering a broadcast. **Explicit intents** specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name. Activity sabe "ir buscar" valor do intent, após reconfigs. **Implicit intents** do not specify the app component to start, but rather declare a general action to perform; Requires at least one intent filter in the manifest file of the target application, so it shows itself competent to the package manager to respond to the action.

An **intent filter** is a declaration in the manifest file (`<intent-filter>`) that associates an app component with an intent. The intent filter specifies the type of intent that the component can respond to, and the component that can handle the intent. Attributes: `action`, `category` and `data`.

Parcelable: Interface for classes whose instances can be written to and restored from a `Parcel`. A `Parcel` is a container for a message (data and object references) that can be sent through an `IBinder`; The `@Parcelize` annotation is used to automatically generate the Parcelable implementation for a class, for the supported types: primitive types, objects and enums, String, CharSequence, Exception, …, all Serializable and Parcelable.

A **task** is a collection of activities that users interact with when trying to do something in your app. These activities are arranged in a stack — the **back stack** — in the order in which each activity is opened. A new activity is pushed onto the back stack when it is started; **Back button**: Android 11 (or lower) - activity is popped off the stack and destroyed; Android 12 (or higher) - activity is moved to background.

A **foreground** task is the task that the user is currently interacting with; A **background** task is a task that is not currently visible to the user.

For the process to be determined as important by the runtime, to keep executing, it must either have a visible activity or have a job being executed in the `WorkManager`.

A execução de uma tarefa por via de um **Foreground Service** lançado a partir de uma activity exige q este seja definido no manifest. Não é igual a uma asyncTaks. A execução de uma tarefa por via de uma AsyncTask a partir de uma activity não garante que a execução da tarefa seja concluída se essa activity deixar de estar visível. A palavra foreground na designação de Foreground Service advém de a execução do serviço implicar a afixação de informação ao utilizador.

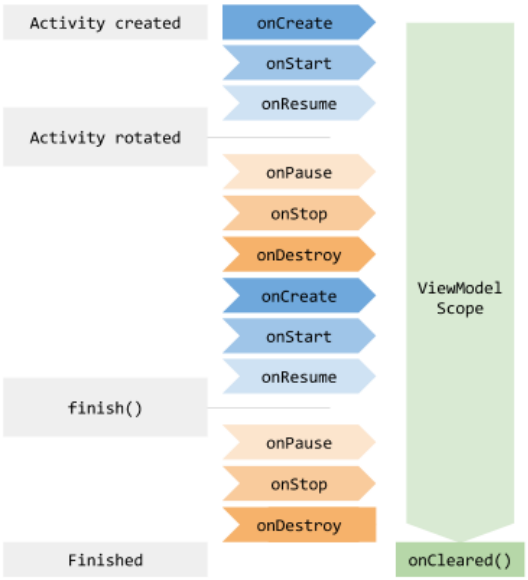
Background work is work that runs outside of the main thread. Can be persistent (remains scheduled through app restarts and device reboots) – use `WorkManager`, or impersistent (canceled when the app is stopped or device rebooted). Immediate impersistent work (sync work) should be done using coroutines. Long running or deferred wok should be done using `WorkManager`, being persistent. doWork é chamado pelo menos uma vez numa background thread. A necessidade de controlar o consumo de energia levou à introdução do WorkManager. O doWork dá relevância ao processo hospedeiro. Allows for work to be done regardless of whether the app is running, for example execution of a periodic work.

Flow: a stream of asynchronous events that you can observe; can be used to represent a stream of events that occur over time; The `StateFlow` class is a state-holder flow that emits the current and new state updates to its collectors. The `SharedFlow` class is a hot flow that shares emitted values among all its collectors; it's a highly-configurable generalization of the `StateFlow`. The `collectAsState()` extension function is used to collect the latest value of a flow as a `StateFlow` instance. `MutableFlow` tem `collect` e `emit`. O callback do collect corre na main thread.

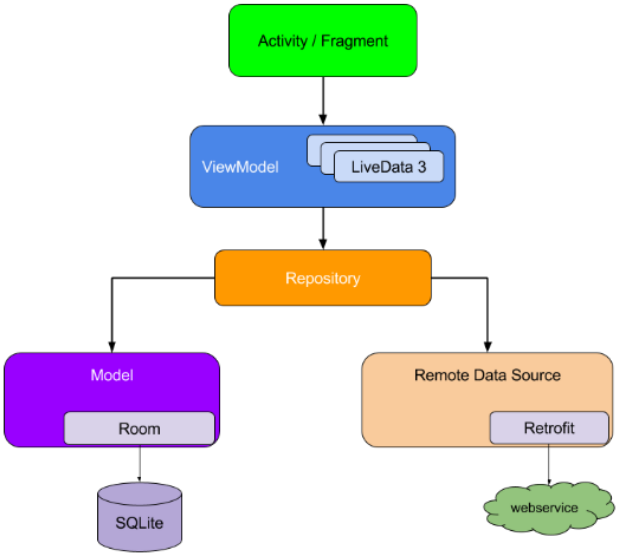
Lifecycle-aware components perform actions in response to a change in the lifecycle status of another component, such as activities and fragments. These components help you produce better-organized, and often lighter-weight code, that is easier to maintain.

ViewModel: Alternative execution host; designed to store and manage UI-related data in a lifecycle conscious way; allows data to survive configuration changes such as screen rotations; persist UI state; access to business logic. Só funciona corretamente se for recebido através do provider viewModels() ou assim. Segundo Android Jetpack, deve ser especifico a activity.

ViewModel objects receive a **SavedStateHandle** object through its constructor. This object is a key-value map that lets you write and retrieve objects to and from the saved state. These values persist after the process is killed by the system and remain available through the same object. Saved state is tied to your task stack. A introdução do SavedStateHandle ajuda a lidar com o controlo automático de recursos de memória.



Application: base class for maintaining global application state; provides application-level resources with the `Context` interface; described in the manifest file. provides application-level resources such as the `Context` object; It is not required.



Repository is a class that abstracts access to multiple data sources. The Repository is not part of the Architecture Components libraries, but is a suggested best practice for code separation and architecture.

Retrofit is a REST Client for Java and Android.

Room is a persistence library that provides an abstraction layer over SQLite to allow for more robust database access while harnessing the full power of SQLite. H as three major components: database class (main access point for the underlying connection), data entities (represent tables) and DAO (provide methods that your app can use to query, update, insert,…). Métodos podem ser sync ou async, dependendo do tipo de retorno. DAO creates the implementation of the methods in build-time.

SharedPreferences: object points to a file containing key-value pairs and provides simple methods to read and write them. Sobrevive a reboot, terminação do host process, e pode ser realizado na UI thread.

	ViewModel	Saved instance state	Persistent storage
Storage location	in memory	in memory	on disk or network
Survives configuration change	Yes	Yes	Yes
Survives system-initiated process death	No	Yes	Yes
Survives user complete activity dismissal/onFinish()	No	No	Yes
Data limitations	complex objects are fine, but space is limited by available memory	only for primitive types and simple, small objects such as String	only limited by disk space or cost / time of retrieval from the network resource
Read/write time	quick (memory access only)	slow (requires serialization/deserialization and disk access)	slow (requires disk access or network transaction)

`onSaveInstanceState(state: Bundle)` -> You can store values inside the state, using for example, `state.setString("a", "1")`. You can restore these values inside `onCreate(savedInstanceState: Bundle?)` with `savedInstanceState.getString("a")`. You can also store Parcelables with `putParcelable()`. Does not survive reboot.

OkHttp is a library to make asynchronous requests, uses `OkHttpClient`. You can create a call with `httpClient.newCall(Request)`. Use `execute()` on a `Call` to execute it synchronously and `enqueue()` to do it asynchronously. Use `suspendCoroutine` to suspend the coroutine, waiting for the asynchronous call to resume (in the callback passed to enqueue, put `continuation.resume`). **Cache** realiza-se com objeto `Cache`, onde lhe é passado o local onde são criados os ficheiros (usar `getCacheDir` para receber do sistema qual a diretoria atribuída) e o tamanho máximo que a cache pode ocupar.

GSON is a library to serialize and deserialize JSON content. Use `@SerializedName("actualJsonProp")` in a property of the destination class (to serialize to an object) to map its name appropriately.

In **Jetpack Compose**, `MutableState` stores a value. If its value is passed as a parameter to a composable, when it changes, a recomposition is triggered on the composable. If we intend to create a mutableState inside of a composable, we need to use `remember`; this allows for the mutableState to maintain across multiple recompositions of the composable where it is declared. While remember helps you retain state across recompositions, the state is not retained across configuration changes. For this, you must use `rememberSaveable`. `rememberSaveable` automatically saves any value that can be saved in a `Bundle`. For other values, you can pass in a custom saver object.

Automatic tests in Android: JUnit, Mocck for mocks. Instrumented tests: createAndroidComposeRule<>(), rule.onNodeWithTag(""): performClick, assertDoesNotExist, assertExists, assertIsEnabled, assertIsNotEnabled.

Cloud Firestore is a NoSQL, document-oriented database. You store data in documents, which are organized into collections. There are three ways to retrieve data stored in Cloud Firestore: Call a method to get the data once, Set a listener to receive data-change events or Bulk-load Firestore snapshot data from an external source via data bundles. You can listen to a document with the `addSnapshotListener()` method. An initial call using the callback you provide creates a document snapshot immediately with the current contents of the single document. Then, each time the contents change, another call updates the document snapshot. `ListenerRegistration` usada para cancelar a subscrição a notificações de atualizações. Pode na main.