1. Introduction to Android

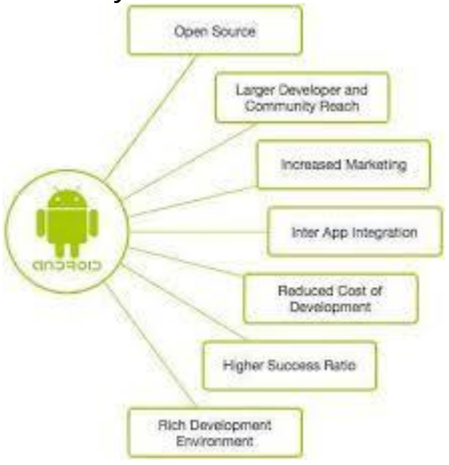
1.1. IntroductiontoMobileApplicationDevelopment

Mobile OS are keep on introducing from Palm OS in 1996 to Windows pocket PC in 2000 then to Blackberry OS and Android. One of the most popular used mobile OS now days are ANDROID. There are around 2.0 lack+ games, application and widgets available on the market for Android users. Android is a powerful Operating System supporting a large number of applications in Smart Phones. These applications make life more comfortable and advanced for the users. Android is a Linux-based operating system designed for mobile devices such as smartphones and tablet computers. The Android is a light weight OS and can easily embedded into different hardware devices like networking equipment, smart TV systems including set top boxes and built in systems and various devices like as house hold appliances and wrist watches.

1.2.What is Android

It is a platform that provides tools and technologies which can used to develop and build mobile.
Applications. The Android platform is open and free software stack that includes an operating system, middleware services and also key applications for use on mobile devices.The Android environment uses the Linux operating system at its core.
Middleware Services makes it easier for software developers to perform communication and input/output, so that developers can concentrate on the specific purpose of their application without need of concentrating on input output resource coding of device. Android provides a middleware layer including libraries that provide services such as data storage, screen display, multimedia, and web browsing. Because the middleware libraries are compiled to machine language, services execute quickly.
Android also provides an application framework that developers integrate into their applications for application development.

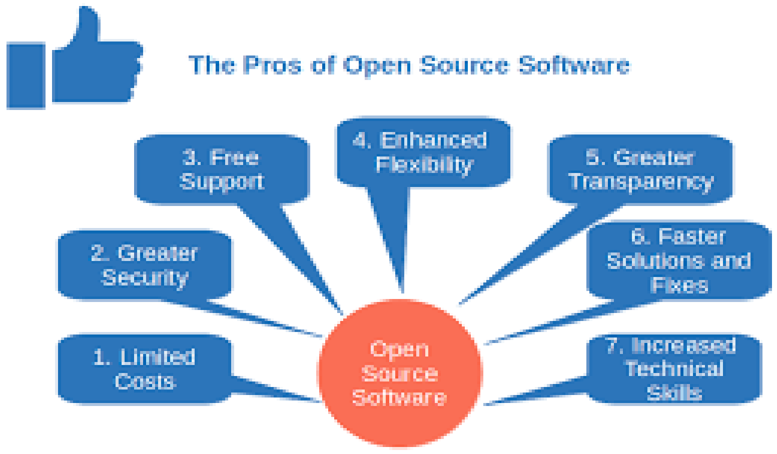
1.3. Why Android?


Open Source
Larger Developer and Community Reach
Increased Marketing
Inter App Integration
Reduced Cost of Development
Higher Success Ratio
Rich Development Environment
android

1.4. Features

1) Open source framework, Android OS is open source and it is part of the Open Handset Alliance, most of the leading handset manufacturers in the world have Android phones.
2) Uses of tools are very simple.
3) Availability of Apps, majority of the apps in Google Play are free as compared to the paid apps on iPhone. 4) Inbuilt support for the Flash.
5) Social networking integration like with Twitter, Facebook, ..etc
6)Integrated Applications & Features, Eg: Android allows an option to share, after taking a photo with the Camera.
7) Free to customizes, we can customize widgets as our wish with new properties.
8) Better Notification System (comprises emails, updates from various widgets.)
9) Updated user interface design
10) It has a better App Market, to easily upload and download.
11) Different Resolutions for Different Screen Sizes.
12) System wide copy and paste functionalities.
13) Multi touch interfacing
14) Java Support so can develop Robust applications.
15) Multi language support
16) Multitask Support, i.e Android phone can run different applications simultaneously, which obviously makes it easier to multitask and improves the overall functionality of the phone.

What is Open Source Project ?



Key features of Android

1. Storage: SQLite, a lightweight relational database, is used for data storage purposes.
2. Connectivity: Android supports connectivity technologies including GSM/CDMA/ Blue tooth, Wi-Fi, 3G, 4G,..etc.
3. Messaging : SMS and MMS are available forms of messaging
4. Multiple language support : Android supports multiple languages
5. Web browser : The web browser available in Android is based on the open-source WebKit layout engine
6. Java support : Java classes are compiled into Dalvik executables and run on Dalvik, a specialized virtual
7. machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU
8. Media support : Android supports the following audio/video/still media formats, like WAV, JPEG, PNG, GIF,BMP, 3GP or MP4 container
9. Streaming media support : HTTP Live Streaming is supported by RealPlayer and others for Android
10. Additional hardware support : Android can use video/still cameras, touchscreens, GPS, accelerometers,gyroscopes, barometers, magnetometers, gaming Controllers, sensors.
11. Multi-touch : Android has native support for multi-touch
12. Bluetooth : Supports sending files, accessing the phone book, voice dialing and sending contacts between phones.
13. Video calling : Skype 2.1 offers video calling in Android 2.3, including front camera supporting
14. Multitasking : Multitasking of applications
15. Tethering : Android supports tethering, which allows a phone to be used as a wireless/wired Wi-Fi
16. External storage : Most Android devices include MicroSD slot and can read MicroSD cards formatted with FAT32, Ext3 or Ext4 file system.

1.5. History & Versions

The code names of android ranges from A to M currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat and Lollipop, Marshmallow, nougat, oreo, pie,Quince Tart,Red Velvet Cake,Snow Cone,Tiramisu Upside Down Cake & Vanilla Ice Cream. Let's understand the android history in a sequence.

What is API level?

version	SDK / API level	Version code	Codename	Cumulative usage	Year
Android 15 DEV	Level 35	VANILLA_ICE_CREAM	Vanilla Ice Cream 2	—	*TBD*
Android 14	Level 34	UPSIDE_DOWN_CAKE	Upside Down Cake 2	12.6%	2023
	▪ **targetSdk** will need to be 34+ for new apps and app updates by August 31, 2024.				
Android 13	Level 33	TIRAMISU	Tiramisu 2	41.4%	2022
	▪ **targetSdk** must be 33+ for new apps and app updates since August 31, 2023.				
Android 12	Level 32 **Android 12L**	S_V2	Snow Cone 2	58.5%	
	Level 31 **Android 12**	S			2021

Android 11	Level 30	R	Red Velvet Cake 2	75.0%	2020
Android 10	Level 29	Q	Quince Tart 2	84.0%	2019
Android 9	Level 28	P	Pie	90.0%	2018
Android 8	Level 27 Android 8.1	O_MR1	Oreo	92.1%	2017
	Level 26 Android 8.0	O		95.1%	
Android 7	Level 25 Android 7.1	N_MR1	Nougat	95.6%	2016
	Level 24 Android 7.0	N		97.0%	
Android 6	Level 23	M	Marshmallow	98.4%	2015
Android 5	Level 22 Android 5.1	LOLLIPOP_MR1	Lollipop	99.2%	
	Level 21 Android 5.0	LOLLIPOP, L		99.5%	2014

	▪ Jetpack Compose requires a **minSdk** of 21 or higher. ▪ Google Play services v23.30.99+ (August 2023) drops support for API levels below 21.				
Android 4	Level 20 Android 4.4W 3	KITKAT_WATCH	KitKat	99.8%	
	Level 19 Android 4.4	KITKAT			2013
	▪ Jetpack/AndroidX libraries require a **minSdk** of 19 or higher since October 2023. ▪ Google Play services v21.33.56+ (July 2021) drops support for API levels below 19.				
	Level 18 Android 4.3	JELLY_BEAN_MR2	Jelly Bean	99.8%	
	Level 17 Android 4.2	JELLY_BEAN_MR1		99.8%	2012
	Level 16 Android 4.1	JELLY_BEAN		99.8%	
	▪ Google Play services v14.8.39+ (December 2018) drops support for API levels below 16.				
	Level 15 Androi	ICE_CREAM_SANDWICH_MR1		99.9%	2011

	d 4.0.3 – 4.0.4		Ice Cream Sandwich		
	Level 14 Android 4.0.1 – 4.0.2	ICE_CREAM_SANDWICH			
	▪ Earlier Jetpack/AndroidX libraries required a **minSdk** of 14 or higher.				
Android 3	Level 13 Android 3.2	HONEYCOMB_MR2	Honeycomb	*No data*	
	Level 12 Android 3.1	HONEYCOMB_MR1			
	Level 11 Android 3.0	HONEYCOMB			
Android 2	Level 10 Android 2.3.3 – 2.3.7	GINGERBREAD_MR1	Gingerbread		
	Level 9 Android 2.3.0 – 2.3.2	GINGERBREAD			2010
	Level 8 Android 2.2	FROYO	Froyo		
	Level 7 Android 2.1	ECLAIR_MR1	Eclair		

	Level 6 **Android 2.0.1**	ECLAIR_0_1			2009
	Level 5 **Android 2.0**	ECLAIR			
Android 1	Level 4 **Android 1.6**	DONUT	Donut		
	Level 3 **Android 1.5**	CUPCAKE	Cupcake		
	Level 2 **Android 1.1**	BASE_1_1	Petit Four		
	Level 1 **Android 1.0**	BASE	*None*		2008

VERSIONS & VERSION WISE FEATURES

Cupcake
This is first version as operating system and released on April 30, 2009.
Features

- In this version onscreen soft keyboard introduced with auto-rotation option
- Supports both land and porttaint modes

- Dictionary with custom words
- Copy and Paste feature added in the web browser.
- Introduced on screen widgets like analog clock, calendar,music player,and search option etc.

Donut
This version released on September 15, 2009.
Fearures:

- Introduced quick search
- Virtual Private Network settings, user can add his/her vpn settings in control panel.
- Voice search and Search box were added.
- Battery usage indicator introduced like for which application how battry in consuming
- Faster OS boot times and fast web browsing experience.
- Google play store updates like catogiries in google play like games,apps
- Getsures for storing, loading,volume controls.

Éclair
This version released on October 26, 2009.
Features

- This version Bluetooth 2.1
- Typing speed is Improved with virtual keyboard, with dictionary option
- Introduced adding events in calendar also invite some one for your event as well.
- From this version onwards only we can build dynamic applications.
- Suppoting zoom option in browser
- no Adobe flash media support.

Froyo
This version is realsed on Released on May 20, 2010.
Features

- It is Support Adobe Flash
- Multi touch event getures are introduced.
- From this version on words we can set mode like night/day mode so that we can save some battery life.
- Device policy manager feature in updated like password & security etc.

- Data backup application is also introduced.
- Crash reporting to developer paly stotre account
- Hotspot also introduced.
- Improved Application launcher with better browser
- No internet calling.

Gingerbread
This version is released on December 6, 2010.
Features:

- User Interface is updated with high efficiency and speed
- Internet calling is introduced
- One touch word selection and copy/paste.
- Supports third party keyboards as well
- 3d motion sensors are updated
- Data encoding and decoding introduced
- Not supports multi-core processors.

Honeycomb
This version released on February 22, 2011.
Features:

- Its Supporing multi-core processors
- It is suitable for tablets with high reach user interface
- Action bar also introduced.
- http live streaming introdiced

IceCreamSandwich
This version released on November 14, 2011.
Features

- Home Screen folder and favorites grouping
- Introduced Virtual button in the UI.
- User can adjust the sizes of widgets
- With swipe user can clear browsers tabs, notifications, older tasks
- Screen shot sharing
- Android beam app for data sharing.

JellyBean
This version released on June 27, 2012.

- Autocomplete option for dial pad
- Smoother user interface is updated

- Multi language support is introduced
- Location accuracy updated .

Kitkat:
This version released on Sep 30, 2013.
Features

- Battery management option introduced.
- “OK google” voice commonds improved.
- Cloud storage optionintroduced.
- Cloud printer introduced.
- Multi tasking introduced like Responds faster, touch is more accurate so that multiple things can also be done at a time like browsing the web, listening to music,

Lollipop
This version released on Oct 15, 2014.
Features

- From this version android is 64-bit system architecture
- From this version apps can support tv, watches and cars
- More security is introduced.
- You can create your rules in your device like settings quick menu customization.

MarshMallow:
This version released on May 28, 2015.
Features

- Design and visual changes
- “Google now tap on” this the most and important feature in android easy to search option in google
- “Ok Google” option on home screen.
- File explorer option for moving files from one location to other .
- Batter indicator on status bar
- Security: App level permissions

Nougat
This version released on June 30, 2016.
Features

- Battery Power Optimization:

From android 7.0 onwords battery life is improved, when mobile is ideal battery is aslo used in very very rare percentage.

- Better Notification Control:

some applications are regularly sending notifactions to users like shoppingkart,banking for those notifications performance increased. Easy to integrate using firebase notification.

- Data Saver

From this version onwards device wont consume more data for background process. So our data could be safe.

- Arrange Quick Settings Menu :

Here we can customize quick settings menu which are using regularly.

- Enable Power Notifications:

If any unraed notifcations it wont consume data upto device unlock.

- Copy-Paste in a Jiffy in Split Screen :

Her we can split our screen into 2 parts of multi tasking that could be in either horizontal or vertical.

- Change the Display Size:

Display sizes like font iamge sizes we can adjust.

- App Shortcuts :

We can create shortcuts for apps on main screen.

- Pin Apps to Share Menu:

Here its very easy to share app and content others.

- Enhanced File Explorer:

It should displays the explore of all files into based on format like size, space etc.

- Mono Audio Playback :

This is the good news for who are single music with single jack it can support without disturbance.

- Emergency Info :

Even if phone is in locked also here can work emergency features like Road block on GPS, battery low etc.

- View Shortcuts with a Physical Keyboard :

This are the key board shortcuts for tab users who are using physical keyboards..

- Bonus Feature: Night Mode: brightness of the screen based on the day & night automatically adjusted.

26 Cool New Features & Changes in Android 8.0 Oreo

The latest version of Android is Oreo introduced in Aug 22 2017.
1. Redesigned Settings Menu
completely the setting menu is changed in android 8.0, that completely we can customize that as per or requirement.
2. Revamped File Manager
File manager is completely updated with different folder like based on the content and all.
3. Battery-Saving Background Restrictions
Based on battery percentage we can restrict some background apps like whatsup,facebook, maps,GPS etc.
4. Snooze Individual Notifications
If we are busy we can snooze indivisual notifications to next 15 minutes. After 15 mintues automatically notification can trigger as new notifaction.
5. Fingerprint Scanner Gestures
Up to know this feature we have for locking and unlocking for device only that only few manufactires are introduced but now this we can get for each and every app as well.
6. Turn on Wi-Fi Automatically

Based on the mobile location automatically wifi turn and turn off facility introduced, so automatically battery life is saved.
7. More Granular Storage Controls
By selecting the file it self we can find the how much space that file is occupying and also OS is also occupying less memory compare to previous versions.
8. Picture-in-Picture Mode
picture in picture mode is nothing but while am watching one videos if am searching other video then older video should come to right of bottom the with playing only so without disturbance we can check other.
9. Autofill Framework
Some our regularly activities like entring address and credit/devbit card information those fileds automatically fills based on the previous data so that here we can save the time .
10. Rescue Party
When a fatal bug comes through and your device starts crashing, the os will trying to automatically fix the issue,that is Rescue Party, basically it fixes when your phone repeatedly reboots, or when a app continuously crashes.
11. Notification Channels
Normally when we no .of notifications latest one should top but here we can gave priority so that always that notification on the top.
12. Adaptive Icons
Up to now in android we have only styles of icon like Square,circle,oval are available but from this version on words we can customize icons.
13. Hi-Fi Bluetooth Codecs
Bluetooth has always lower audio quality when compared to a set of wired headphones. So from this version google implroved Bluetooth range with high frequency.
14 Tab Navigation
Up to only we have option to acces single tab in browsers but from 8.0 onwards we can manage multiple tabs in browsers like Laptops/Desktops.
15. App Drawer Improvements
This is easy navigation to installed app once we enable this feature if you touch any where on the screen it should open navigation from left side with all apps as listview, so easily we can access those.
16. Wider Color Gamut in Apps
Upto 7.0 when we want to use any colors of Image, backbround & text color only we have option RGB but from 8.0 onwards Render wide color gamut content with using below code (extra palet option to select color).
android:colorMode="wideColorGamut"

17. Wi-Fi Aware
In lower versions if any open networks available simply it will display only give some notification open network available but from android 8.0 on words automatically it can connect.
18. Notification Badges on the Home Screen
This is for unread notifications it will display badgets for unread notifications same as ios (icon it self will display count of unread notification) this we can enabled or disabled from “Apps & Notifications” menu in Settings.
19. No More ‘Unknown Sources’ Setting
In previous versions there is option in settings i.e “Unknown Resources” that time if we check unkonown resources its accept any type of APK. But from now it won’t accept APK from unknown resources only it can accept from official stores only.
20. Battery Percentage Indicator Tweaks
Previously, we have small overlay to your battery indicator that shows the current battery percentage. But now, in this version we have option to display the percentage next to the battery indicator in your status bar.
21. Battery & Connection Indicators in Quick Settings
This is just a minor change, but when you pull down your notification panel there is option to access your Quick Settings now you will see a set of network and battery indicators on the top of the screen, next to the Settings
22. Powered by Android’ on Boot Screen
In bottom of the screen while booting the device its showing one message i.e. powered by Android its previously already available in pixel & nexus.
23. Package Installer Progress Bar
When we installing any application in previous versions we used progress Dialog but now progress dialog as deprecated so instead of the now using progress Bar with animation.
24. Smart Text Selection
This feature isvery helpful to user when user copying some text from message or some other location previously all text should be same but from 8.0 if it contains URL that portion highlighted as a URL and other properties like Bold, italic, color those should effect as it is when we pasted in some other location.
25. Overlay Sticky Notification
Overlay is nothing but hightlighthing the necessary content. In Orieo google added this feature for notifications but for this Android 8.0 require permission from the user.

26. New Emojis
Previolsy in android only we have rounded traditional emojis but nowGoogle launched blob shaped smiles with gradient properties. Using emojis we can express our fellings in lesser time while chatting with others.

BEST 13 FEATURES OF ANDROID 9.0 PIE

Gesture Navigation
In Previous versions Android has used a standard three-button navigation bar at the bottom of the screen for ages. In Pie, you can drop the standard Back, Home, and Recent buttons in favor of a new gesture-based navigation system.
If want to try this feature, click on Settings > System > Gestures > Swipe up on Home button. Your navigation bar will change right away.
Adaptive Battery and Brightness
Adaptive Battery is an expansion of the Doze feature introduced in Android 6 Marshmallow. Doze put apps that you weren't using into a "deep sleep" to prevent them from wasting battery. Now, Adaptive Battery goes further by learning about the apps and services you use most often, then adjusting what you don't use as much to use less battery.
App Actions
Google's launcher already recognizes the apps you're most likely to use based on the time of day. Now, App Actions let you quickly start tasks by predicting what you want to do.
For example, you might see a shortcut to start Google Maps navigation to work in the morning. At work, you might see an App Action to chat with your coworker on Hangouts. And when you plug in headphones, you'll see an App Action for your most recent playlist.
Slices
Similar to App Actions, Slices let you jump right to certain actions in apps. For instance, Google says that if you search for uber on your phone, you'll see a shortcut to hail a ride to work, complete with price and ETA.
Improved Security Features

The first is that this version of Android “restricts access to mic, camera, and all SensorManager sensors from apps that are idle” according to Google.
Second, a new lockdown mode fortifies your phone in emergency situations. After enabling this at Settings > Security & location > Lock screen preferences > Show lockdown option, you can tap Lockdown on the Power menu. This instantly locks your phone, disables fingerprint unlocking and Smart Lock, and hides notifications on your lock screen
Digital Wellbeing
The dashboard shows you how many notifications apps send you, how much time you spend in apps, and how often you check your phone. You can also set daily time limits to keep yourself from wasting hours in time-sinking apps
New Accessibility Menu
A new menu in Android Pie makes it simple to access common functions for users who need assistance.
Enable this menu at Settings > Accessibility > Accessibility Menu. Turn on the Use service slider and confirm the prompt, and you’ll see a new icon to the right of the navigation bar. Tap this anytime to bring up a large menu with shortcuts to Volume, Recent apps, Quick Settings, and more.
New Screenshot Shortcut
The default Power + Volume Down button combination for screenshots is a little awkward. Thus, in Android Pie, you can take a shortcut from the Power menu anytime.
What’s more, you can also tap the Edit command in the notification that appears to make adjustments to your shot right away.
Easier Screen Rotation
Android automatically switches your screen orientation based on how it’s situated. You can lock the orientation to portrait or landscape, but this turns into a pain if you need to switch often.
In Pie, if you have Auto-Rotate turned off, you’ll see a new icon on the right side of the navigation bar when you rotate your device to landscape. Tap it to lock in landscape orientation, and it will stay even if you turn back to portrait. Just tap the icon again to rotate back to portrait.
Volume and Sound Improvements
When you press a Volume button, you’ll notice the slider now appears on the right side instead of the top. What’s more, pressing volume buttons now changes the Media volume instead of the Ringer volume like before. This simple volume tweak makes it easier to avoid opening a YouTube video and accidentally playing it at full blast.

Tap the Note icon to mute or unmute media audio. You can tap the icon above this to toggle your Calls volume between Ring, Vibrate, and Mute. You'll need to select the Gear icon to open the Sound menu and make detailed adjustments.
Selectable Dark Mode
Android Oreo included a dark mode, but the system automatically decided whether to enable it based on your wallpaper. Now you can choose for yourself at Settings > System > Display > Advanced > Device theme
Easier Text Selection
If you copy and paste a lot, you'll love a small change in Pie. Now when you long-press to select text and grab the handles, a little magnifier lets you see exactly what you're selecting.
More Notification Information
If you want to see which apps are sending distracting notifications, head to Settings > Apps & notifications > Notifications. In the Recently sent section, you can see which apps have pinged you recently. Tap See all from last 7 days to view more info.
Changing Most recent to Most frequent lets you find the worst offenders. Android will also suggest that you disable notifications from apps you swipe away frequently. Don't forget about the notification channels introduced in Oreo, either.

ANDROID 10 FEATURES

A dark mode
Dark mode can be activated by a quick setting or when you activate the battery saver option. Android 10 features, having been requested for a long time now. But the Mountain View company is also working with third-party developers to implement dark modes in their apps.
Smart Reply for all messaging apps
Smart Reply is one of the better Google features out there, predicting what you're going to say in response to a message. It's currently available for Google apps, but it's now coming to all messaging apps in Android 10.
This means you can now get suggested responses in the likes of WhatsApp and Facebook Messenger — a handy way to save time when a short response will do. These suggestions are all made using on-device machine learning, purportedly maintaining your privacy as the relevant information isn't sent to Google's servers.
A better sharing menu

Many Android enthusiasts have criticized the platform's sharing functionality for being slow and unintuitive. Fortunately, Google has overhauled this menu in Android 10.
The new sharing menu is meant to be much faster than the legacy menu, but it's also supposed to do a better job of recommending contacts and apps for sharing.
Focus Mode
One of the more prominent user-facing Android 10 features is the so-called Focus Mode, an extension of the Digital Wellbeing suite. As the name suggests, this mode will help you focus by graying out apps you deem distracting and hiding their notifications.
Digital Wellbeing is also getting another feature thanks to integrated parental controls. Google has already offered parental tools via the Family Link app, but out-of-the-box support is welcome nonetheless.
Quicker access to settings
It's already super easy to toggle Wi-Fi, Bluetooth, and other connectivity options, but Google is making this process a little easier when you're in apps. Enter the settings panel.
This new popup window can be summoned by apps in certain situations. Google gives the example of launching a browser when in airplane mode. The browser can now tell users to activate Wi-Fi, then automatically summon the settings panel.
A standard depth format
Android 10 also brings a new depth format, dubbed (surprise) the Dynamic Depth Format, and it opens the door for depth-editing in loads of third-party apps.
Starting in Android 10 "apps can request a Dynamic Depth image which consists of a JPEG, XMP metadata related to depth related elements, and a depth and confidence map embedded in the same file on devices that advertise support," reads an excerpt of the Android Developers Blog. Google confirmed that Facebook is one such app making use of depth data on the Pixel 4.
Google also confirmed that the new format will let third-party apps tweak depth data to create "specialized blurs and bokeh options." Hopefully, third-party developers embrace this new, Google-pushed standard.
An improved Files app
Google's previous Files app was a no-frills affair, and it didn't even have a shortcut in the app drawer. Fortunately, the Android 10 Files app is a step above the previous version.
Not only does the new app have a shortcut, but it also offers a revised UI, a universal search bar at the top, and quick access to other apps.
Overhauled permissions

Android 10 also has plenty of privacy-related tweaks, with roles being one of the biggest additions in this regard. With roles, the platform can now automatically grant specific permissions to an app based on its use-case. So a text messaging app would automatically gain the ability to send/receive texts, as well as access to your contacts.
Another major change is a tweak affecting location permissions. Now, users have the option to either grant location access in general to an app, or only allowing access when the app is actively being used.
Wi-Fi sharing via QR codes
Xiaomi and Huawei smartphones have allowed users to share Wi-Fi credentials via QR codes for a long time now. So we're glad to see Google adopt this trend with the new Android update.
The feature is easy to use, as you tap on your Wi-Fi connection, hit the share button, then authenticate with your phone's password or a fingerprint. From here, you should see a QR code, and your friend can scan this code to gain access. Again, it's nothing new for third-party brands, but we're happy Google is catching up in this regard.
Gestures, app drawers play nicely
The early Android 10 previews introduced more comprehensive gesture navigation, such as sliding your finger inwards from the screen edge to go back. Unfortunately, this gesture didn't work well in apps with navigation drawers/menus. This meant users wanting to see these overflow menus might accidentally activate the back gesture instead.
Fortunately, Android 10 has a so-called "peek" feature to solve the issue. Now, users wanting to see an app's navigation drawer/overflow menu need to swipe in and hold for a second. Once you see the menu "peek" out (see image above), you can continue the swipe gesture to fully reveal it.
Google Assistant 'handles'
The switch to gesture navigation means that you can no longer activate the Google Assistant via the home button. You can however activate the assistant via an inward swipe from the lower corners of your screen on Android 10.
This activation method isn't obvious though, so Google has added visual cues in the form of Google Assistant "handles" in the corners (seen above). Hopefully this subtle addition eases confusion about Assistant's activation in Android 10.
Hearing aids get streaming support
Google is also bringing another nifty accessibility feature to Android 10, allowing users to stream music via Bluetooth to their hearing aids.

According to *Engadget*, the feature uses the new Audio Streaming for Hearing Aids protocol (ASHA) to stream music, calls, and other audio from a Pixel phone.
The feature is reportedly compatible with the Pixel 3 and Pixel 3a series for now, but Google is open-sourcing the platform. So hopefully we quickly see more phones and hearing aids offering this truly useful function.
Security updates via Play Store
Another major initiative coming to Android 10 is Project Mainline, an effort by Google to deliver some security updates via the Play Store. This significantly reduces the waiting time for security updates in theory, as you don't have to wait for your operator to approve the update. Furthermore, it shouldn't require a lengthy installation process, working in a similar manner to app updates.
The only real downside to this solution is that some security vulnerabilities can't be patched in this fashion, necessitating a traditional security update anyway. But it's definitely a major step in the right direction.
Qualcomm has also announced that phones equipped with its 2020 chipsets will let you update graphics drivers via the Play Store. It's believed that this is due to Project Mainline as well. This is definitely one of the coolest Android 10 features if you have a supported phone.
Wet, overheating USB warnings
Android 10's release has also seen *XDA* uncover evidence of new USB-related functionality in the update. More specifically, the operating system will now display a warning when your USB port is wet. Any connected accessories will also be disabled until you either manually enable them or until the phone detects that the port is no longer wet. Furthermore, the latest Android update will also issue a warning if your USB port is overheating. You'll still need to unplug your phone from the charger or USB cable of course, but the warning is welcomed nonetheless.

ANDROID 11 FEATURES

Built-in screen recording
Android 11 brings a feature some custom Android forks already have – built-in screen recording. This lets you use easily-accessible controls within the phone to create a video of what's on your screen, similar to screenshots.
You can even record the sound of your phone, or use the mic to narrate what you're doing, so this should be a great feature for creating short-

form video like TikToks, or for making tutorial videos to help less tech-savvy relatives make the most of their smartphone.
Related to this, the Android 11 sharing functions have seen wider rollout to apps, so you can easily copy and paste images, text and videos from one app and send them straight to another.
New conversations tab
When you swipe down from the top in Android 10, you bring down your notification bar – well, in android 11, that's split into two sections, consisting of your notifications, and your 'conversations'.
These conversations are basically just chat notifications from chat apps like WhatsApp, Facebook Messenger and Twitter, so you know they're probably more worth paying attention to than the other notifications.
This way you can easily see if people need you, and reply straight away too. You can also now pin conversations to 'Bubbles' – you might have seen these for Facebook Messenger, where a floating icon appears above other apps to show you that you have a message.
Now you can get messages from more apps to appear as bubbles, and you can pin them there too so the chat is always easily-accessible.
A related improvement is that Gboard, the default Android keyboard, will now auto-fill forms with relevant information, similar to what Chrome currently does, which should save time if you're frequently filling out your information.
Smart home and media controls
The new Android 11 update brings loads of changes for people who use loads of smart home devices.
From one easily-accessible menu (accessed by long-pressing the power button) you can control all the IoT (Internet of Things) devices you have connected to your phone, as well as NFC bank cards.
Talking of home, there's a new Bedtime Mode in the phone which you can set to run during the night, which turns on Do Not Disturb and makes the phone screen black-and-white to protect your vision.
There are new media controls too. It's now easier to make music play from other devices connected to your phone, like Bluetooth speakers or other gadgets.
Plus, Android Auto now works wirelessly on your car, saving you from having to fiddle with wires to get it working. And on the topic of transport, now when you turn on Airplane mode, any headphones connected via Bluetooth will stay connected, so you won't have to reconnect them.
Improved accessibility
Google has improved its Voice Access mode in Android 11, although there isn't too much information on what's changed.

Apparently, the hands-free mode is now 'faster and easier to use' – a bigger change is that the mode now works offline, so you don't need to always be connected to use it. These changes should hopefully make Android 11 a lot more accessible, letting people stay connected regardless of disability.
A cool new mode is the braille keyboard, so you can write braille messages without needing to buy separate software.
Finally, the Lookout app now lets you scan documents and food labels, so people with limited vision can still 'read' documents and see what's in their food
Improved prediction tools
Android 11 will seemingly reduce the work you need to do on your phone, by predicting your habits and patterns.
One such example of this is smart folders, so you can let Android 11 automatically sort your apps into folders of similar apps, like games or productivity tools.
App suggestions is also tweaked to suggest apps based on your routine – for example, if you always log onto your Fitbit app first thing in the morning to examine your sleep habits, the phone will now automatically pop that app into the Home screen in the morning so it's easily accessible.
Finally, apparently the Smart Reply feature already usable in Android phones has received some tweaks. This mode suggests some automatic responses when you receive a message, letting you reply with one tap (if any of the responses are appropriate) but it's not clear what's new here.
Security and privacy
Android 11 also brings some changes to app permissions. Now, you can grant an app permission to, say, your camera or location, on a one-off basis, instead of the existing options of 'all the time' and 'only while using the app'.
In addition, your phone will automatically revoke permissions for apps if you haven't used them in a while, so an app won't track your location if you forget you have it downloaded, for example.
Other useful features
There are a few new Android 11 features which don't fit into the above categories.
One of these is better 5G detection, so if you're using a 5G phone Android 11 apps will recognize this more easily, and run faster as a result.
On the topic of improved innovative software, Android 11 will now detect hinge angles on foldable phones, so apps can better run when your foldable phone isn't flat.

Picture-in-picture mode is a useful way of watching videos while you do other things, and now it’s easier to change the size of the window so you can better fit whatever else you’re doing.
There’s a new ‘Nearby Share’ mode which lets you easily send information and data to nearby devices like your tablet, Chromebook or computer using Chrome, so you can easily send a document from your phone to your PC, for example.
Finally, you can now see older notifications in settings, in case you accidentally swiped one away or ignored it for too long.
Android 11 Easter egg
The last part of our Android 11 guide will explore the Easter egg – new versions of Android always have one of these, and this one was discovered by users almost as soon as the beta rolled out.
This Easter egg is very similar to the one in Android 7, in that it gives you a pet cat to look after. If that sounds weird, it is, but it’s pretty fun too. If you want to find out more for yourself, follow our guide below on how to enable the Android 11 Easter Egg.
Firstly, head into the Settings app, then click About Phone, then Android Version, then repeatedly click where it says ‘Android Version: 11’. If you do this right, the Android 11 logo should pop up, which is a green dial. Turn this dial clockwise from its default position to 11. This won’t work straight away, you need to turn it around a good few times – if you can’t go beyond 10, turn the dial back to 1, and then turn it back to 11 again. Some users report this working after three tries, but if it still doesn’t work then keep trying. When it works, a tiny cat will pop up on the bottom of the screen.
Now you can head over to the smart home control menu detailed above – if you skipped over that section, you can find the menu by long-pressing the power button on the side of your phone.
Here you can find options to interact with the cat, but if you’ve already mapped controls to this menu, the options won’t be visible – instead, try to add new controls, then click ‘See Other Apps’ to find the cat tools.
You can re-name your cat, and over time more cats might turn up too.

ANDROID 12 FEATURES

Wallpaper-based Dynamic Theming
The ‘monet’ dynamic theming engine is one of the highlights of Android 12 and what makes Material You so special. The engine will pull the colors from the wallpaper applied and use them throughout the UI and system accents for theming purposes. What makes the experience even

better is that the colors are also applied to apps that have been updated to support the theming engine.
The dynamic theming engine in Android 12 will make sure you never get bored of the UI. A simple wallpaper change will be enough to give everything a fresh coat of paint.
Game Dashboard
A Pixel-exclusive Android 12 feature for now, Game Dashboard aims to improve your gaming experience by overlaying some important tools and information, like the ability to record your gameplay, streaming your gaming session on YouTube, or displaying a live FPS counter so that you can get an idea of how your phone is performing.
You can also select from one of the three different gaming profiles — Performance, Standard, and Battery Saver — depending on your requirements. However, this particular feature will only work with games that have been updated to take advantage of the new APIs in Android 12
One-handed mode
Taking a cue from third-party Android skins, Google has added a native one-handed mode in Android 12. As the name indicates, the feature makes it easy to use your phone with one hand — a boon for devices with large displays and gargantuan size like the Pixel 6 Pro.
On your device, head over to Settings > System> Gestures. Tap on One-Handed mode and enable the toggle. Now, you can enable one-handed mode by simply swiping down from the bottom edge of the display. Do note that this feature only works when you are using gesture navigation on your device.
Quick Tap
Taking a cue from the Back Tap gesture in iOS 14, Google has introduced a new Quick Tap feature in Android 12. With a simple double-tap on the back of your phone, you can launch an app of your choice, control media playback, take a screenshot, show recent apps, and trigger Google Assistant.
The Quick Tap gesture is tucked under Settings > System > Gestures in Android 12. Now, depending on your preference, select an action that you want to assign to Quick Tap.
Direct share in the Recents overview menu
The quick image sharing feature from the Recents overview menu has been further improved in Android 12. Now when you simply drag-and-drop images from Instagram, a web page in Chrome, or other similar apps, a list of your recommended contacts will show up at the bottom for quick and easy sharing. This makes the entire process of sharing content from the Recents overview more streamlined and convenient.
Privacy Dashboard

Google is once again stepping up its efforts in the privacy department with the Privacy dashboard, which will give you an overview of which apps have accessed your location data, camera, microphone, and other such permissions over the last 24 hours. You get a complete breakdown, right down to the minute when an app uses a particular permission.
If you are paranoid about the installed apps on your device silently tracking you or listening to your conversations, you can use the Privacy dashboard to figure out if that's true or not. Privacy dashboard can be accessed from Settings > Privacy > Privacy dashboard.
Dim the screen
If you find your phone's screen brightness a bit too bright even at the lowest setting possible, there's a new Extra Dim option in Android 12 that further dims the screen. This is great for anyone who tends to stare at their phone's display in a dark room, as the extra-low brightness will help reduce the strain on their eyes.
You can find the option tucked under Settings > Accessibility > Extra dim. Alternatively, there's a Quick Settings tile for it as well.
Conversation widget
Google gave widgets a major revamp in Android 12 and added a handy Conversation widget to go along with it. The widget will help you access your most frequent conversations in just a tap, irrespective of the app you use.
Below is how you can add a Conversation widget to your home screen:

1. Long-press on an empty area of your home screen. From the menu that pops up, select Widgets .
2. Scroll down and select Conversation, followed by the Conversation widget.
3. You can resize the widget as per your liking. Now, select a recent conversation that you'd like to show in the widget. You can then quickly access the conversation by simply tapping on it.

Scrolling screenshots
This is a feature that Google has taken ages to bring to its flavor of Android. Gone are the days when just taking a screenshot would suffice — users now frequently need to screenshot long lists. With scrolling screenshots, you'll no longer have to take multiple screenshots to capture a long list. Instead, take a screenshot and from the toolbar tip that pops up at the bottom, tap Capture more. Remember that this option will only show up in lists or where it is actually possible to take a long screenshot.

One advantage of Google's implementation is that it lets you decide the starting and ending of the screenshot, so you can capture exactly the part you want.

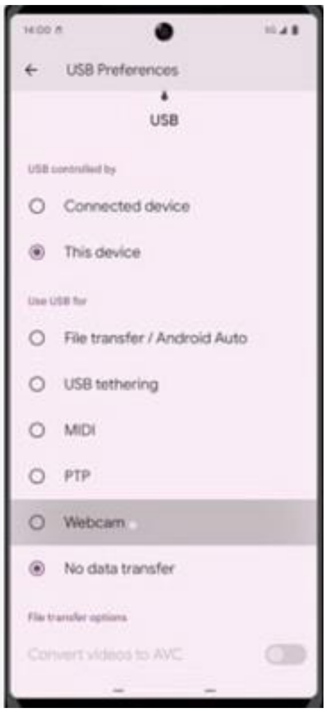
ANDROID 13 FEATURES

Android 13 Released on January 2022, Android 13 has not been released yet.
However, Android versions typically come with various improvements, new features, and enhancements to performance and security. Some common areas of improvement in Android updates include:

1. User Interface (UI) Changes: Android updates often bring changes to the user interface, introducing new design elements, animations, and visual improvements.
2. Performance Enhancements: Each new Android version aims to improve overall system performance, making devices run more smoothly and efficiently.
3. Security Updates: Android updates address security vulnerabilities and enhance the overall security of the operating system. This includes improvements to the security framework, encryption, and protection against malware.
4. Battery Optimization: Android updates may include optimizations to improve battery life and efficiency, allowing devices to last longer on a single charge.
5. New Features and APIs: New Android versions introduce additional features and APIs (Application Programming Interfaces) that developers can use to enhance their apps.
6. Privacy Controls: Android updates often include new privacy features and controls to give users more control over their data and app permissions.
7. Connectivity Improvements: Updates may bring enhancements to connectivity features, including improvements to Wi-Fi, Bluetooth, and mobile data.
8. System-Wide Dark Mode: Recent Android versions have included a system-wide dark mode, providing users with a darker color scheme for improved visibility and reduced eye strain in low-light conditions.

ANDROID 14 FEATURES

Android 14 Released on Nov 2023.
Features:
Native Webcam Support



USB Preferences

USB

USB controlled by

- ○ Connected device
- ◉ This device

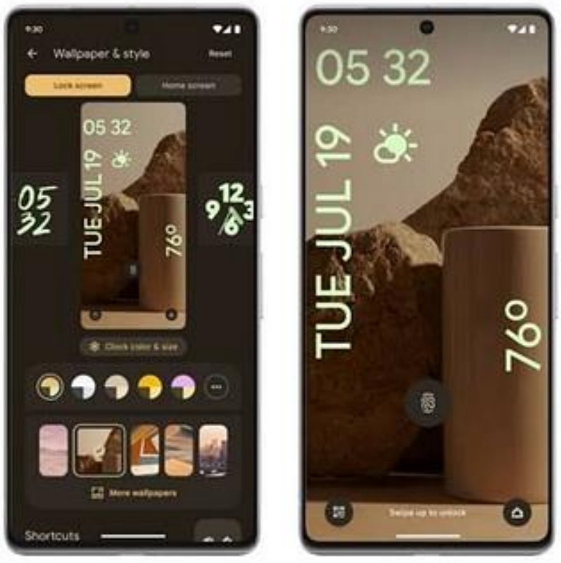
Use USB for

- ○ File transfer / Android Auto
- ○ USB tethering
- ○ MIDI
- ○ PTP
- ○ Webcam
- ◉ No data transfer

File transfer options

Convert videos to AVC

In general, laptop webcams aren't very clear — they usually offer just okay quality, like 720p or even less. We've all faced this problem. There may be a cool solution in Android 14 for those looking for a better webcam. With this solution, you can use your Android phone as a 1080p camera for your computer.
Consider this scenario: you plug your phone into your computer, and a menu appears. In that menu, you could choose "webcam," and your phone's camera would become your computer's camera. Currently, this feature isn't part of the system, not even hidden away in a secret setting. If Google gives it the green light, it might appear in Android 14. You might be able to get a fancier camera setup down the road without spending more money.
Enhance lock screen



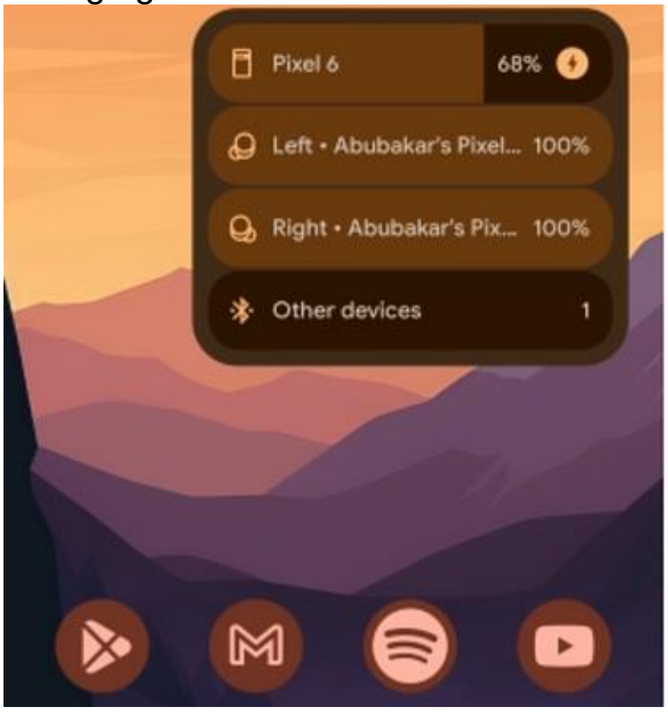
With Android 14, you can customize your phone in a variety of ways. One big thing how your lock screen appears. Your clock can look different and even what apps appear at the bottom corners can be customized.

If all goes well for Google, these changes will be included in the official Android 14 release next month. Just a heads-up, the clock styles they showed at I/O weren't great. We hope the final versions will be more fun and less boring.

Satellite connectivity

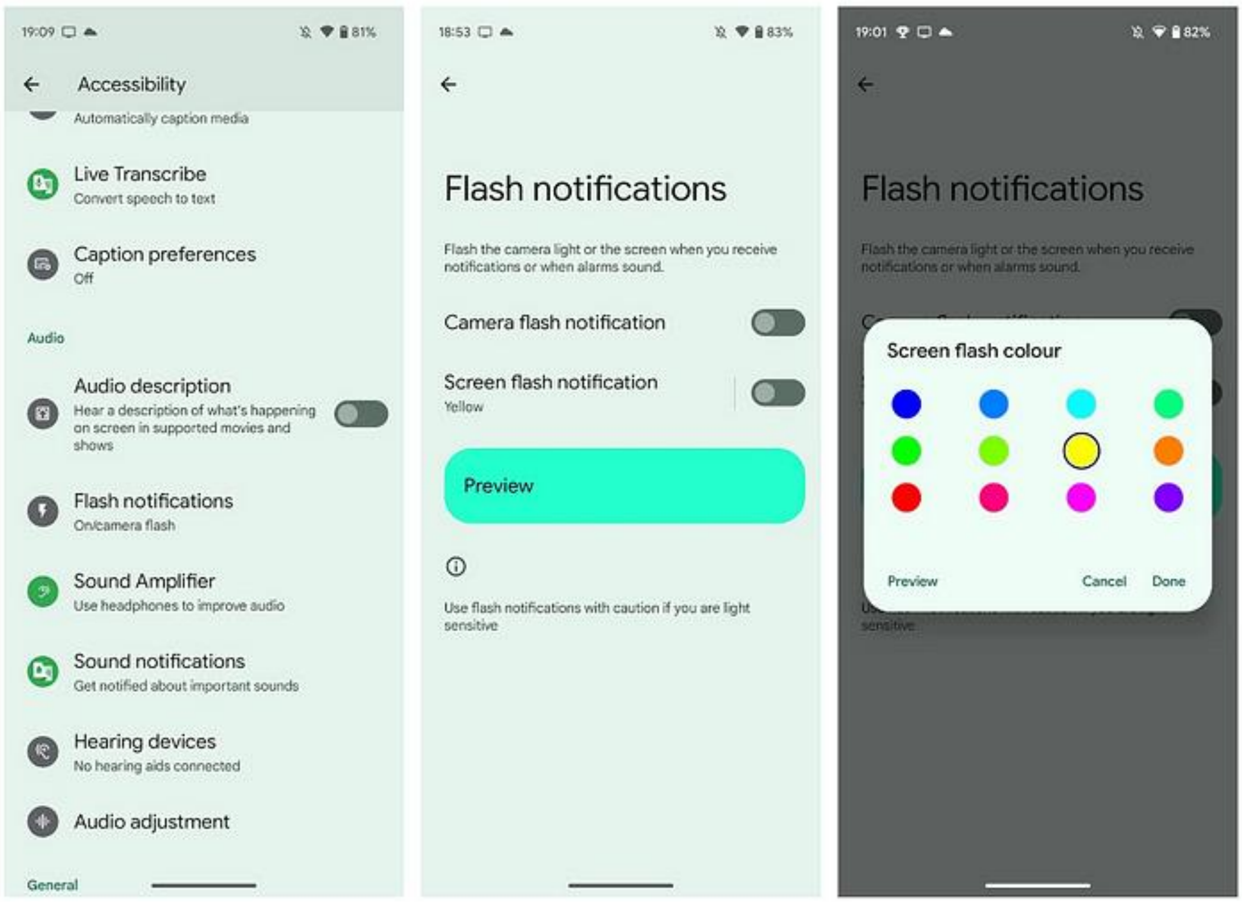
9:41
Emergency SOS
End
CONNECTED
Turn Left to Stay Connected to Satellite
Show Messages

Your phone might soon be able to connect to satellites with Android 14! As a result, you can send important messages even without cellular or WiFi service. Apple did this with their iPhone 14 series last year. According to a tweet from TeamPixel, Android 14 will support satellite connections for sending messages. Additionally, Google Pixel and Samsung Galaxy phone owners may be the first to test it. For this satellite thing to work, your phone needs special hardware. It depends on other Android phone makers whether they add this later.
Imagine yourself in a place where there is no cell signal or WiFi. Texts can still be sent if your phone supports satellite technology. There's something magical about it when you're out in the wilderness!
Charging Pill


Pixel 6
68%
Left • Abubakar's Pixel... 100%
Right • Abubakar's Pix... 100%
Other devices
1

With the latest update for Android 14 Beta, a new feature is added. A pretty animated pill will appear on your screen when you connect your device to charge. This discovery was made by Mishaal, and interestingly, the Android Beta Twitter account accidentally spilled the beans on it.
Tapping the pill has no special function, but it's a visually appealing addition to Material You.
When you charge your device, it's like a little bit of fun, adding a touch of excitement.
Drag drop feature(Gesture support enhance)

There was a cool trick in iOS 15 that let you move text and pictures around. Guess what? The Android 14 operating system will be able to do the same thing. Actually, it’s already there if you’re using the Android 14 Beta 3 version, and it works really well.
Here’s the deal: pick some text, hold it down, and move it. Then, use your other hand to switch to another app where you want to put the text, and let go. There it is, the text appears like magic. As soon as Android 14 is released, you’ll be able to do the same thing with pictures as well Cool, right?
Flash notifications



19:09 81%

Accessibility

Automatically caption media

Live Transcribe
Convert speech to text

Caption preferences
Off

Audio

Audio description
Hear a description of what's happening on screen in supported movies and shows

Flash notifications
On/camera flash

Sound Amplifier
Use headphones to improve audio

Sound notifications
Get notified about important sounds

Hearing devices
No hearing aids connected

Audio adjustment

General

18:53 83%

Flash notifications

Flash the camera light or the screen when you receive notifications or when alarms sound.

Camera flash notification

Screen flash notification
Yellow

Preview

Use flash notifications with caution if you are light sensitive

19:01 82%

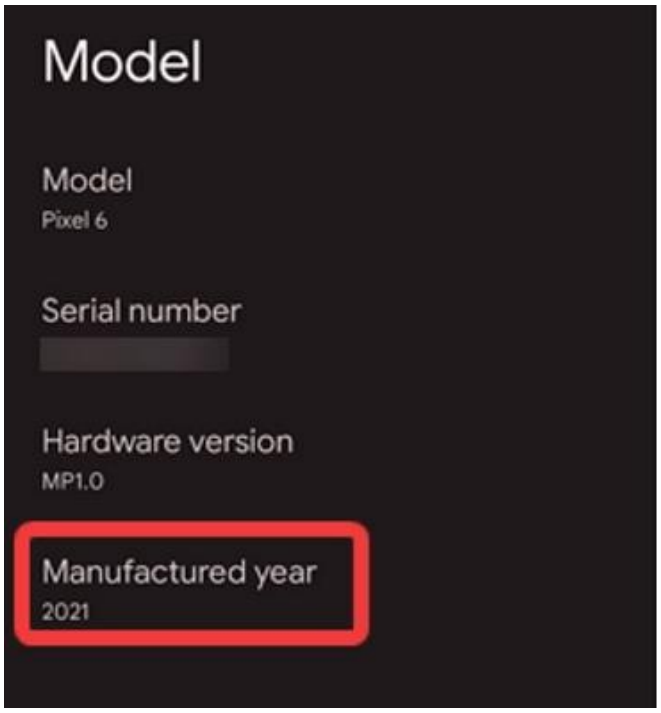
Flash notifications

Flash the camera light or the screen when you receive notifications or when alarms sound.

Screen flash colour

Preview Cancel Done

I really like the Notification Flashes feature. That's great for people who frequently check their phones in noisy places where hearing notifications is difficult, or for those with hearing challenges.
To get it working, just head to your phone's Settings, then go to Display, and find Flash notifications. From there, you can flip the switch for Camera Flash and/or Screen Flash.
There's an extra cool thing you can do with Screen Flashes. You don't have to stick with one color. Nope, you can choose from a variety of colors. Moreover, you can see how each color will look before making a decision. Check out how the color flashes, then decide if it's right for you.
Trying on different outfits before you pick the right one is like trying on clothes before you buy them!
Device Manufacturing Date



Model

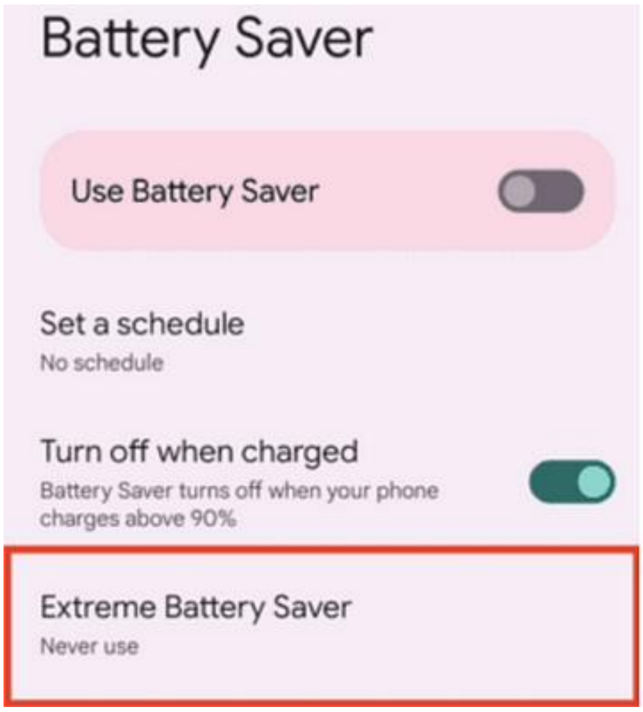
Model
Pixel 6

Serial number
[redacted]

Hardware version
MP1.0

Manufactured year
2021

In Android 14, by this feature allows you to easily find out the year your phone was produced. All you need to do is navigate to Settings > About Phone > Model. Once you're there, you'll spot the Manufactured year displayed alongside details like Hardware version, Serial number, and model.
Now, you might not think this information is super crucial, but it can be surprisingly handy, especially when you're considering purchasing a pre-owned or refurbished device that doesn't come with its original retail packaging. Knowing the year of manufacture can give you a better idea of the phone's age and possibly its condition.
So, while it may not be something you use every day, it's a small yet useful addition for those times when you're exploring options for buying a second-hand device.
Extreme Battery Saver option



Battery Saver

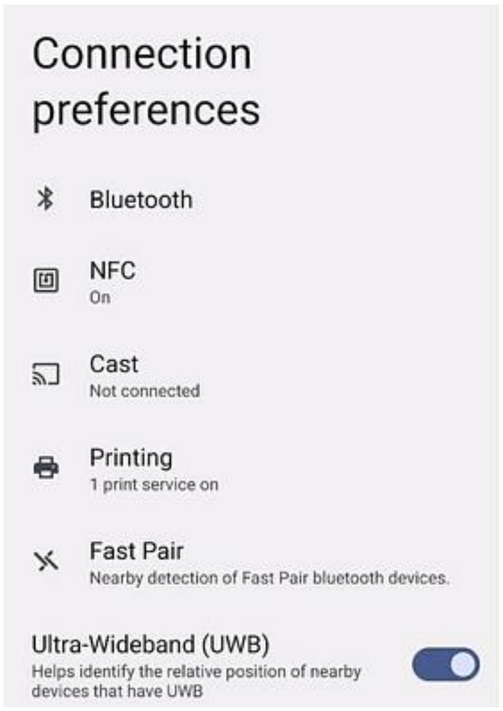
Use Battery Saver

Set a schedule
No schedule

Turn off when charged
Battery Saver turns off when your phone charges above 90%

Extreme Battery Saver
Never use

In the Android 14 Beta 3 version, they've introduced a clever little feature that could potentially stretch your Android phone's battery life when you've got the Battery Saver mode turned on. Activating the Battery Saver is easy — just head over to Settings, tap on Battery, then locate and flip the switch for Battery Saver.
As soon as you enable this mode, the first thing you'll notice is that your phone switches to Dark mode.
This switch helps conserve battery power. But that's not all! The feature also takes some extra steps to manage your phone's energy consumption. It might slow down or even pause a few apps that are running in the background, which in turn reduces the drain on your battery.
It's like giving your phone a little extra boost to keep it running longer when you're in a pinch. So, remember, when you're looking to extend your battery life on Android 14 Beta 3, turning on Battery Saver could be a smart move.
Fast Pair option



Connection preferences

Bluetooth

NFC
On

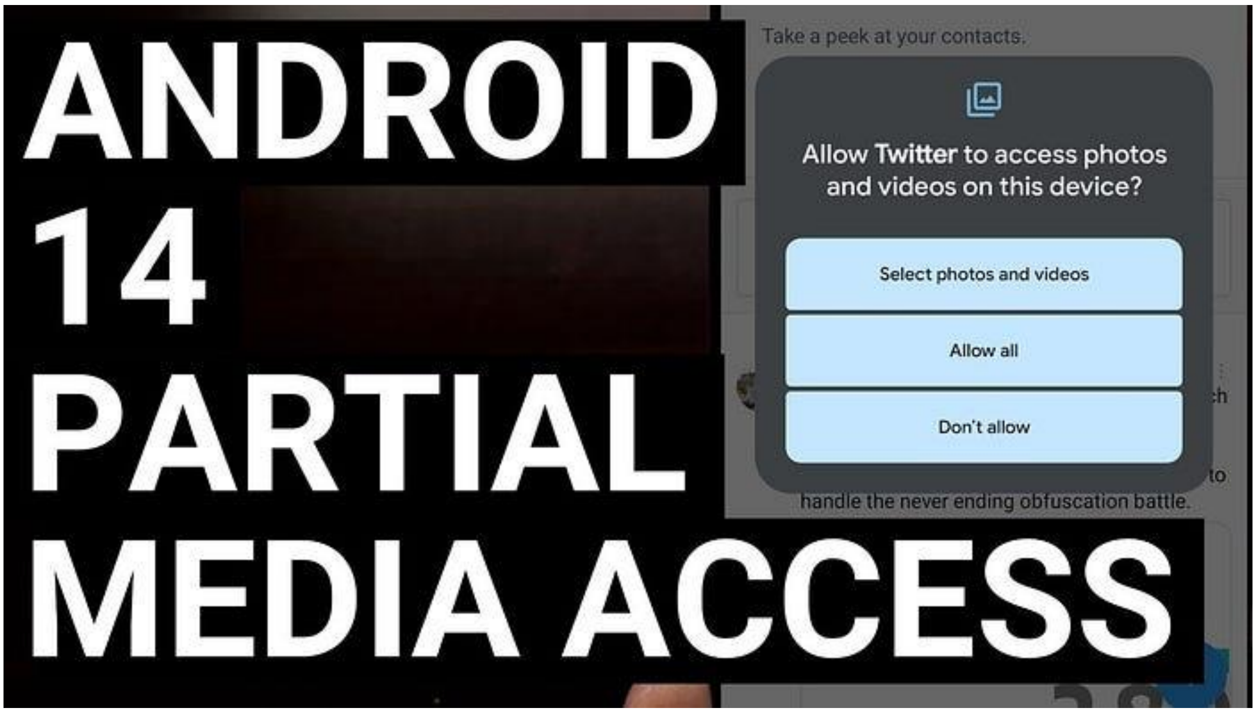
Cast
Not connected

Printing
1 print service on

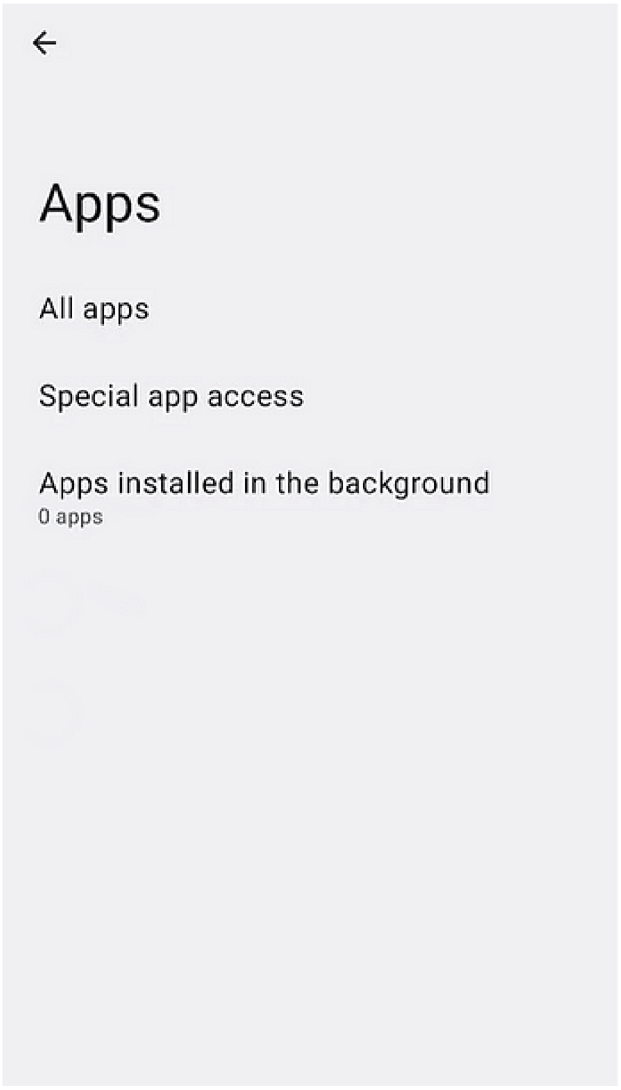
Fast Pair
Nearby detection of Fast Pair bluetooth devices.

Ultra-Wideband (UWB)
Helps identify the relative position of nearby devices that have UWB

In Android 14, there’s this really cool thing called “Fast Pair.” It’s like a special button hidden in the Connection settings. What’s it do? Well, it makes connecting your Bluetooth stuff (like headphones or speakers) to your Android phone super easy. No more dealing with tricky steps to make them talk to each other.
With Fast Pair, you just tap a bit, and boom! Your Bluetooth stuff is friends with your Android. Imagine it like a magical shortcut for making gadgets work together.
So, when you want to use your cool wireless stuff, you don’t need to be a tech wizard. It’s all about making things faster and simpler, so you can enjoy your devices without the hassle. That’s Android 14 being awesome for you!
Grant partial access to photos and videos


ANDROID 14 PARTIAL MEDIA ACCESS
Take a peek at your contacts.
Allow Twitter to access photos and videos on this device?
Select photos and videos
Allow all
Don't allow
handle the never ending obfuscation battle.

basically, this feature allows you to set additional filters for specific apps that request gallery access. Many social networking apps currently ask for permission to access our galleries, giving them the ability to retrieve all our mobile device images.
In Android 14, you can establish extra filters between these apps. For instance, let’s say I’ve organized multiple albums or folders within my gallery, each containing different types of images, such as social life photos and personal documents. Prior to Android 14, there was no option to exclusively share only my social life images with these apps. However, in Android 14, I can now apply this filter specifically for these applications. This way, I can safeguard my other images from being accessed by these apps. It’s a welcomed privacy enhancement that grants users more control over their image sharing.
Background Install Control (bloatware remover)

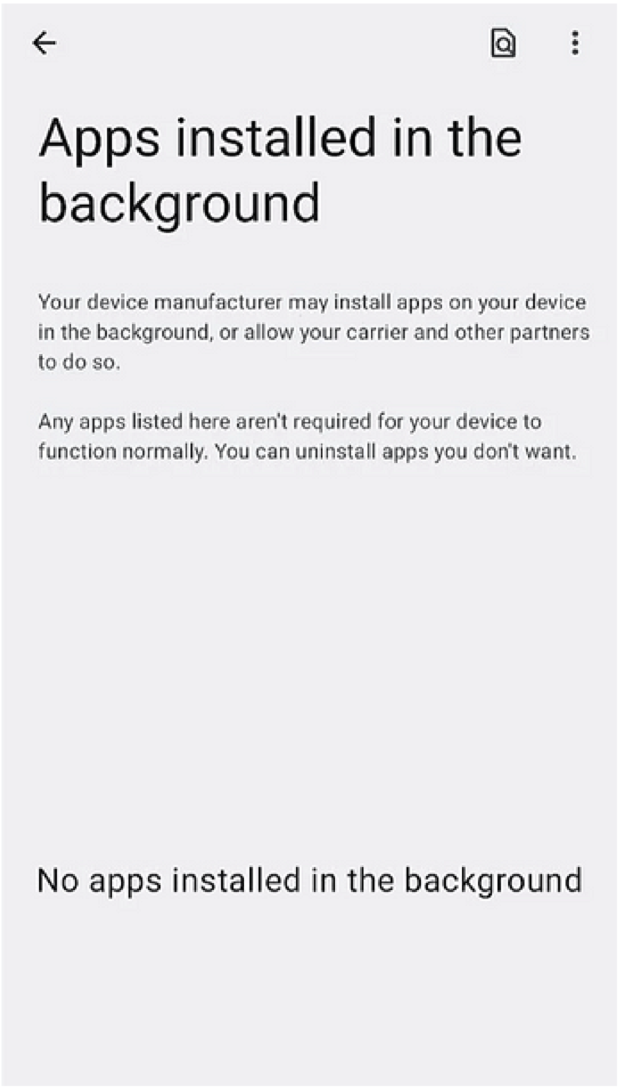


Apps

All apps

Special app access

Apps installed in the background
0 apps



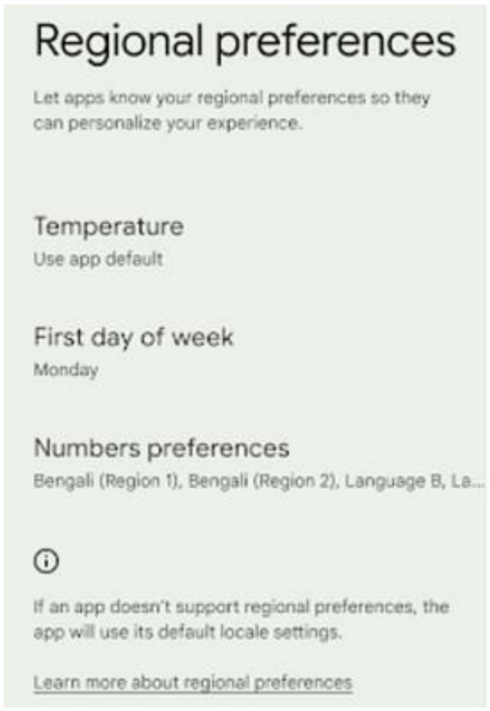
Apps installed in the background

Your device manufacturer may install apps on your device in the background, or allow your carrier and other partners to do so.

Any apps listed here aren't required for your device to function normally. You can uninstall apps you don't want.

No apps installed in the background

By this option you able to know and delete bloatware they can run on a background without your knowledge. basically by this feature you can track apps that are installed in the background. Any apps that are silently downloaded in the background are listed in this menu.
You can then look through the menu to delete apps you don’t want.it would be a lot easier to find and get rid of any bloatware weighing down your device.
Regional preference on calendar and number


Regional preferences

Let apps know your regional preferences so they can personalize your experience.

Temperature
Use app default

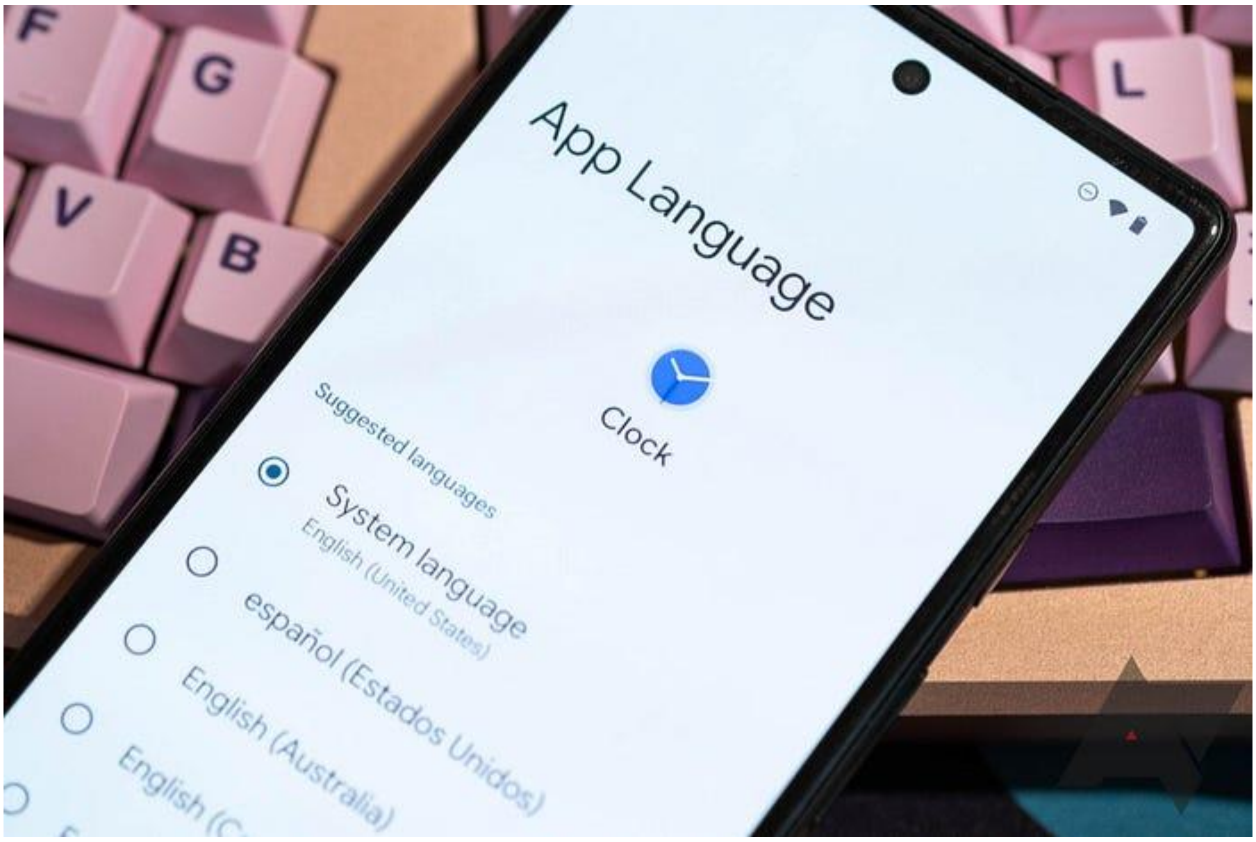
First day of week
Monday

Numbers preferences
Bengali (Region 1), Bengali (Region 2), Language B, La...

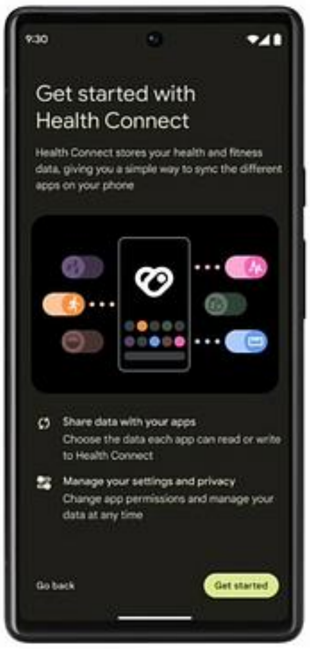
If an app doesn't support regional preferences, the app will use its default locale settings.

Learn more about regional preferences

Basically, it allows users to configure system-wide preferences for temperature units, calendar formats, first days of the week, and number systems.
As a result of this setting, apps will no longer need to ask for or make assumptions about your preferences. In Android 14, Google is expanding this concept to even more areas.
The new "regional preferences" feature in Android 14 allows users to select their preferred units for temperature, calendar, first day of the week, and number system for each location. When you toggle a hidden developer option in Settings > System > Languages & input, you will find it under "Regional preferences".
Different Application languages preferences



Using this option, you can run an app according to your language preference. Globally, many users want to use specific apps in their preferred language without changing the system's language.
In Android 14, you have more application options, so you can set different apps to different languages according to your preference. Android 13 already supported this to some extent, but in Android 14, you have more application options to select from.
Enhances Health paring apps

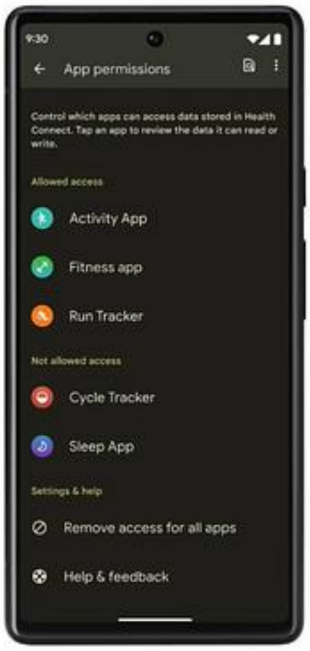

9:30

Get started with Health Connect

Health Connect stores your health and fitness data, giving you a simple way to sync the different apps on your phone

Share data with your apps
Choose the data each app can read or write to Health Connect

Manage your settings and privacy
Change app permissions and manage your data at any time


9:30

App permissions

Control which apps can access data stored in Health Connect. Tap an app to review the data it can read or write.

Allowed access
- Activity App
- Fitness app
- Run Tracker

Not allowed access
- Cycle Tracker
- Sleep App

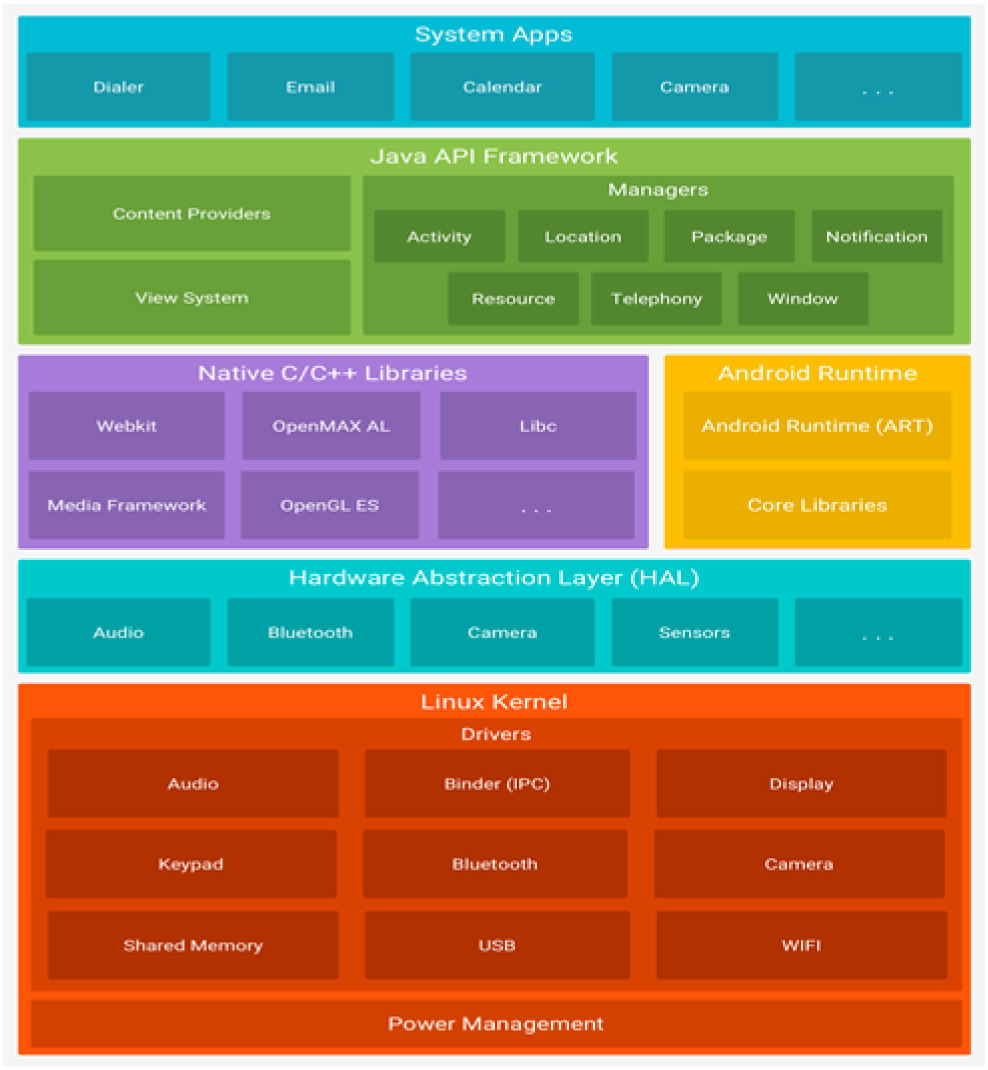
Settings & help
- Remove access for all apps
- Help & feedback

With Health Connect, you can get all the important health and fitness information from different apps in one convenient location. It's like a central organizer for your health information. If you have a Samsung health app, a Fitbit app, or a Peloton fitness device, the Health Connect app can pull data from all of them.
Your app store currently offers the Health Connect app for download. This allows you to easily access and manage your health data from different sources. However, there's exciting news! Google has just announced that starting with the release of Android 14 beta 2, Health Connect will be included as a built-in feature of the Android operating system.
This means that when you get a new device or update your existing Android device to Android 14 or later, you'll automatically have the Health Connect app pre-installed. You won't need it.
Furthermore, the built-in integration of Health Connect means that you'll also receive updates seamlessly. These updates will happen automatically as part of the regular Android system updates, ensuring that you always have the latest features and improvements for managing your health data without any extra effort on your part.
This streamlined experience makes it easier for users to stay on top of their health and fitness information, all while enjoying the benefits of a hassle-free setup and continuous updates.
I believe these 14 features are unique and good, so please let me know if you have any others apart from these 14 features. I've gathered information from a number of websites and researched. Please leave a

comment if you notice anything that does not seem right or if you have any thoughts to share. It’s really important for me to hear your thoughts!

ARCHITECTURE

Architectute is nothing but physical structural plan or block diagram of some view.
Android is Linux based software stack , it is open source for lot of devices. For developers they are giving full acces to Frame work API acces to develop core level applications, because of that only manufactrers are customizing their own features.
The following diagram is shows rhe components of android.



Android architecture has total 6 layers.

SYSTEM APPS

The top most layer is system APPS. Normally normal devices will come with Contacts, Messages, Browsers, calenders etc.. so here user can install any number of applications which they are using, some third party applications may be we can set as default apps like browers, Messengars and keyboards
These all very helpful to for because if developer is developing his own applications he can those functionalities for example if we developemnet any application related to SMS just developer invoke the services then next portion of the service like sending sms & sms status will be done android system only

JAVA API FRAMEWORK

The Android OS complete feature set is avaible for devlopers in the form of APIs those APIs are written by Java language. This are very helpful to developers to simplify their app development, even developer can use cor part of architecture alos to build apps like components and services which are included in API framework. The following are few of the APIs

- View System:

View System is for building of user interface in android like textview, button, grid, list, image,vidoes etc. here using XML/Java we can build UI is very very rich, simple & attractive.

- Content Providers:

Content providers is nothing but we can access the data from other applications like while sending the message we are accessing contacts data & in now days all social media applicatiosn we are accessing the contacts, gallery etc.
Managers:
These are also API but non coding portion providing layouts, images, strings, colors, default styles etc.

- Resource

Resource manager provides acces to resources for our developemnet such as Strings, Images, nimations, graphics, colors, vidoes etc.

- Notification

Notification Manger will help enable the custom notification in status bar.

- Activity

Activity Manager as very very important API in android it is only controlling the navigation of each screen in our apps using lifecycle methods.

- Location

This Api could help to acces the geo location,navigation using Services of Android Architecture.

- Package

Each package contains multiple java classes and it can control all the functionalities of those classes.

- Telephony

It can communicate with networks like mobile data, wifi, hotspot & sharing data etc.

- Window

It can handle the multiple windows in our device. For example symultantously while listing the songs chatting with some one so its handling two two different windows at a time like same some windows cache memory & browser windows etc.

1. Native C/C++ Libraries

Many of the android componnts and servicew are build from native code those android runtime & Hardware abstraction layers because native libraries wriiten in C & C++.

- Webkit

It can support all type of browsers.

- Open Max AL

It should create run time environment.

- Libc

Lot of componets are developed by C & C++ so it always requied support of Libc.

- Meida framework

Android apps can support all type of media like jpg, png, mp3, mp4 etc.

- Open GLES

It can support 2D & 3D graphics

1. Android Run Time
2. Android Runtime (ART)

From android 5.0 and above version of android with its own process and its own instance of Android Runtime. ART is designed for low memory device by excuting the DEX (byte code) files to run on virtual devices,and bytecode is designed special optimized for low memory devices.
ART features

- Optimized garbage collection (GC)
- Just In Time complier (JIT)
- Ahead of Time (AOT)
- Better debugging support
- Supporting Crash reports

Note:
Previosly Lower version (less than 5.0) is run in Dalvik virtual machine, now as well as it can run in ART also, then it can run DVM as well, but vicevera is not possible.

- Core Librariries

Android application also includes a set of core libraries that provide most of the runtime functionality of the Java programming language, and it includes android 8.0 features as well

1. Android Abstraction Layer(HAL)

The android hardware abstraction layer provides User interfaces that expose hardware capabilities of device with higher version of Java. It can lot of hard ware components, suac as Audio, Bluetooth, Camera, sensors. When access device hardware through a framework

API Android system loads the library module for that hardware component.

1. Linux Kernel

Linux kernel is hardware layer. For example, the ART relies on the Linux kernel for functionalities such as threading and low-level memory management. Linux kernel allows Android to take advantage of key features like develop hardware drivers for a well-known kernel such as Audio, Keypad, Shraed memory, Usb, Bluetooth, Binder(IPC)Display, Camera, Wifi, Power Management etc.

•

INSTALLATION OF ANDROID STUDIO

For Android development with Kotlin you need download Android Studio latest from the Android Studio page.

Android Studio provides a complete IDE, including an advanced code editor and app templates. It also contains tools for development, debugging, testing, and performance that make it faster and easier to develop apps. You can use Android Studio to test your apps with a large range of preconfigured emulators, or on your own mobile device. You can also build production apps and publish apps on the Google Play store.

Android Studio is available for computers running Windows or Linux, and for Macs running macOS. The OpenJDK (Java Development Kit) is bundled with Android Studio.

The installation is similar for all platforms. Any differences are noted below.

1. Navigate to the Android Studio download page and follow the instructions to download and install Android Studio.
2. Accept the default configurations for all steps, and ensure that all components are selected for installation.
3. After the install is complete, the setup wizard downloads and installs additional components, including the Android SDK. Be patient, because this process might take some time, depending on your internet speed.
4. When the installation completes, Android Studio starts, and you are ready to create your first project.

Creating Project in Android Studio

- Open Android Studio: Launch the Android Studio IDE on your computer.
- Start a New Project: On the welcome screen, click on "Start a new Android Studio project," or if you have an open project, you can go to File > New > New Project.
- Choose a Template: Android Studio offers several templates for different types of apps. Select the one that best fits your project needs. Common choices include "Empty views Activity" ". Click "Next" once you've chosen.
- Configure Your Project: Here, you'll set up the details for your project, including the name, save location, language (Java or Kotlin), minimum SDK version, etc. Make sure to give your project a meaningful name and select a suitable package name. Then, click "Finish."
- Gradle Build: Android Studio will take a moment to set up your project and download any necessary dependencies. This process

might take some time depending on your internet speed and the complexity of your project.

- Project Structure: Once the Gradle build is complete, you'll see your project structure in the left sidebar. This includes folders like "app," "res," and "Gradle Scripts." The "app" folder contains your app's code and resources.
- Run Your App: Before you start coding, it's a good idea to run your app to make sure everything is set up correctly. You can do this by clicking the green play button in the toolbar or by going to Run > Run 'app'.
- Choose a Device: Android Studio will prompt you to select a device to run your app on. You can choose a physical device connected to your computer or an emulator. If you don't have any devices set up, you can create a new virtual device by clicking "Create New Virtual Device."
- Wait for Build: Android Studio will build your project and deploy it to the selected device or emulator. This process might take some time, especially the first time you run your app.
- View Your App: Once the build is complete, you should see your app running on the selected device or emulator. Congratulations, you've successfully created and run your first Android Studio project!

ANDROID PROJECT STRUCTURE

The project structure in Android Studio organizes your app's source code, resources, configuration files, and dependencies in a hierarchical manner. Here's a breakdown of the typical structure:

- app:
    - manifests: Contains the AndroidManifest.xml file, which describes essential information about your app to the Android system, such as activities, permissions, and services.
    - java: Contains the Java (or Kotlin) source code for your app. This is where you write your activities, fragments, services, and other components.
    - res: Contains all the resources used by your app, such as layouts, drawables, strings, and values.
        - drawable: Contains images and other drawable resources.

        - layout: Contains XML files defining the layout structure of your app's UI.
        - values: Contains XML files defining various values, such as strings, colors, dimensions, and styles.
    - assets: Contains raw asset files that your app uses. These files are included in the APK without any processing.
    - libs: Contains third-party libraries (JAR files) that your app depends on.
    - build.gradle: This file configures the build settings for your app module, such as dependencies and build types.
- Gradle Scripts:
    - build.gradle (Project): This file configures the build settings for the entire project, such as the Gradle version and repositories.
    - build.gradle (Module): This file configures the build settings for the app module, such as dependencies, packaging options, and signing configurations.
- External Libraries: Contains all the third-party libraries that your app depends on. You'll see these libraries listed here after you've added them to your project.
- Gradle: Contains Gradle wrapper files necessary for building your project using Gradle.
- Android Dependencies: Contains the libraries and dependencies that your app uses. These are typically added via Gradle scripts and are automatically managed by Android Studio.
- Build: Contains build-related files and directories generated during the build process, such as intermediates, outputs, and logs.
- .idea: Contains configuration files for Android Studio, such as project settings, run configurations, and code styles. This directory is specific to Android Studio and is typically hidden from view.
- Gradle: Contains files related to the Gradle build system, such as wrapper files and caches.

OVERVIEW OF XML

XML stands for Extensible Mark-up Language.XML is a very popular format and commonly used for sharing data on the internet. This chapter explains how to parse the XML file and extract necessary information from it.
HTML is a forgiving language. It tolerates a host of sins, from imprecise markup to altogether missing elements, and can still generate a web

page in the browser. XML, on the other hand, is basically a tyrant. Violate even the most trivial rule, and the browser or your application will crash. Some people find comfort in the uncompromising nature of XML, because it won't work unless you build it correctly. It's great to get instant feedback when you do something wrong!

ADVANTAGES OF XML

The main features or advantages of XML are given below.
1) XML separates data from HTML: If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.
With XML, data can be stored in separate XML files. This way you can focus on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.
With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.
2) XML simplifies data sharing: In the real world, computer systems and databases contain data in incompatible formats.
XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.
This makes it much easier to create data that can be shared by different applications.
3) XML simplifies data transport: One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.
Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.
4) XML simplifies Platform change
Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.
XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.
5) XML increases data availability: Different applications can access your data, not only in HTML pages, but also from XML data sources.
With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.
6) XML can be used to create new internet languages : A lot of new Internet languages are created with XML.

Here are some examples: XHTML, WSDL for describing available web services, WAP and WML as markup languages for handheld devices, RSS languages for news feeds, RDF and OWL for describing resources and ontology, SMIL for describing multimedia for the web

RULES

There are nine basic rules for building good XML:

1. All XML must have a root element.
2. All tags must be closed.
3. All tags must be properly nested.
4. Tag names have strict limits.
5. Tag names are case sensitive.
6. Tag names cannot contain spaces.
7. Attribute values must appear within quotes (“”).
8. White space is preserved.
9. HTML tags should be avoided (optional).

XML that follows these rules is said to be “well formed.” But don’t confuse well-formed XML with valid XML!
Now let’s look at the rules with some examples.
Rule 1: All XML Must Have a Root Element
A root element is simply a set of tags that contains your XML content.

```
<root>
  <author>Ernest Hemingway</author>
  <author>John Steinbeck</author>
  <author>James Joyce</author>
</root>
```

Rule 2: All Tags Must Be Closed
When a tag is declared (opened), it must also be closed. Any unclosed tags will break the code. Even tags that don’t need to be closed in HTML must be closed in XML or XHTML. To open a tag, type the name of the element between less-than (<) and greater-than (>) characters, like this opening tag:

```
<author>
```

To close a tag, repeat the opening tag exactly, but insert a slash in front of the tag name, like this closing tag:

```
</author>
```

Even empty tags, such as <hr> and <br>, must be closed.
Wrong:

```
<author>Ernest Hemingway
<p>Roses are Red
<hr>
```

Right:
<author>Ernest Hemingway</author>
<p>Roses are Red</p>
<hr></>
<hr />
Rule 3: All Tags Must Be Properly Nested
When you insert (nest) one tag within another, pay attention to the order in which you open each tag, and then close the tags in the reverse order. If you open element A and then element B, you must first close B before closing A. Even HTML tags that usually will work without a strict structure must follow the stricter XML rules when they're used within an XML file.
Wrong:
<A><B>Text</A></B>
<b><i>Text</b></i>
Right:
<A><B>Text</B></A>
<b><i>Text</i></b>
Rule 4: Tag Names Have Strict Limits
Tag names can't start with the letters *xml*, a number, or punctuation, except for the underscore character (_).
The letters *XML* are used in various commands and can't start your tag name. Numbers and punctuation also aren't allowed in the beginning of the tag name.
Wrong:
<01_author>
<"author">
Right:
<author>
<_author>
Rule 5: Tag Names Are Case Sensitive
Uppercase and lowercase matter in XML. Opening and closing tags must match exactly. For example, <ROOT>, <Root>, and <root> are three different tags.
Wrong:
<author>Hemingway</AUTHOR>
<Author>Hemingway</aUTHOR>
Right:
<author>Hemingway</author>
<AUTHOR>Hemingway</AUTHOR>
Rule 6: Tag Names Cannot Contain Spaces
Spaces in tag names can cause all sorts of problems with data-intensive

applications, so they're prohibited in XML.Rule 7: Attribute Values Must Appear Within Quotes
Attribute values modify a tag or help identify the type of information being tagged. If you're a web designer, you may be used to the flexibility of HTML, in which some attributes don't require quotes. In XML, all attribute values must appear within quotes. For example:
<chapter number="1″>
<artist title="author" nationality="USA">
Rule 8: White Space Is Preserved
If you're in the habit of adding extra spaces and hard returns in your HTML code, watch out! Such spacing is honored by XML and can play havoc with your applications. Use extra spacing judiciously.

LAYOUTS

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. You can declare a layout in two ways:
1. Declare UI elements in XML. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
2. Instantiate layout elements at runtime. Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.
The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application's UI. For example, you could declare your application's default layouts in XML, including the screen elements that will appear in them and their properties. You could then add code in your application that would modify the state of the screen objects, including those declared in XML, at run time.
In Android we have different types of Layouts are there. Will discuss each layout in detail

RELATIVE LAYOUT

In Android, RelativeLayout let you position your component base on the

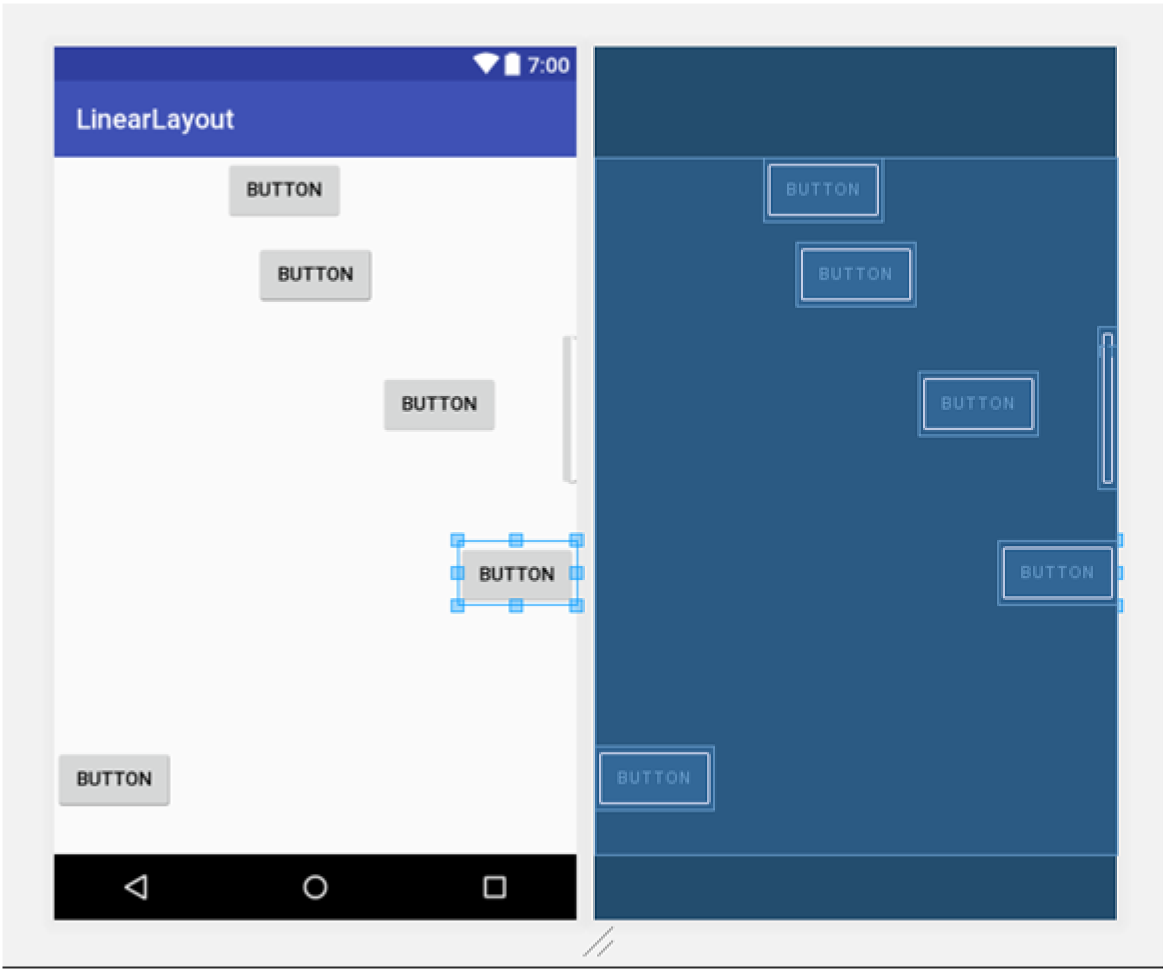
nearby (relative or sibling) component's position. It's the most flexible layout, that allow you to position your component to display in anywhere you want (if you know how to “relative” it).

In RelativeLayout, you can use “above, below, left and right” to arrange the component position, for example, display a “button1” below “button2”, or display “button3” on right of the “button1”.

Note: The RelativeLayout is very flexible, but hard to master it. Suggest you use Studio IDE to drag the component, then view study the Studio generated XML layout code to understand how to code “relative” components. Example code & properties of Relative Layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:layout_alignParentLeft="true"
      android:layout_alignParentStart="true"
      android:layout_marginBottom="31dp"
      android:text="Button" />
   <Button
      android:id="@+id/button2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:layout_alignParentLeft="true"
      android:layout_alignParentStart="true"
      android:layout_marginBottom="31dp"
      android:text="Button" />
   <Button
      android:id="@+id/button3"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_marginLeft="37dp"
      android:layout_marginStart="37dp"
      android:layout_toEndOf="@+id/button"
```

```
        android:layout_toRightOf="@+id/button"
        android:text="Button" />
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/button3"
        android:layout_marginLeft="157dp"
        android:layout_marginStart="157dp"
        android:layout_marginTop="77dp"
        android:layout_toEndOf="@+id/button3"
        android:layout_toRightOf="@+id/button3"
        android:text="Button" />
    <Button
        android:id="@+id/button5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/button3"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="14dp"
        android:text="Button" />
    <Button
        android:id="@+id/button6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignEnd="@+id/button5"
        android:layout_alignRight="@+id/button5"
        android:layout_alignTop="@+id/button5"
        android:text="Button" />
    <Button
        android:id="@+id/button7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:text="Button"
        android:layout_marginRight="57dp"
        android:layout_marginEnd="57dp"
        android:layout_alignTop="@+id/button4"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />
    <Button
        android:id="@+id/button8"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/button4"
        android:layout_marginTop="38dp"
        android:text="Button" />
</RelativeLayout>
```



Properties of Relative Layout

android:id: This is the ID which uniquely identifies the layout

android:gravity: This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc

android:ignoreGravity: This indicates what view should not be affected by gravity.

android:layout_above: The bottom edge of this view above the given anchor view ID and must be a reference to another resource, in the form “@[+][package:]type:name”

android:layout_alignBottom: Makes the bottom edge of this view match the bottom edge of the given anchor view ID and must be a reference to another resource, in the form “@[+][package:]type:name”.

android:layout_alignLeft: Makes the left edge of this view match the left edge of the given anchor view ID and must be a reference to another resource, in the form “@[+][package:]type:name”.

android:layout_alignParentBottom: If true, makes the bottom edge of this view match the bottom edge of the parent. Must be a boolean value, either “true” or “false”.

android:layout_alignParentEnd : If true, makes the end edge of this view match the end edge of the parent. Must be a boolean value, either “true” or “false”.

android:layout_alignParentLeft : If true, makes the left edge of this view match the left edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_alignParentRight: If true, makes the right edge of this view match the right edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_alignParentStart: If true, makes the start edge of this view match the start edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_alignParentTop : If true, makes the top edge of this view match the top edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_alignRight: Makes the right edge of this view match the right edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

android:layout_alignStart: Makes the start edge of this view match the start edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

android:layout_alignTop: Makes the top edge of this view match the top edge of the given anchor view ID and must be a reference to another resource, in the form “@[+][package:]type:name”.

android:layout_below: The top edge of this view below the given anchor view ID and must be a reference to another resource, in the form “@[+][package:]type:name”.

android:layout_centerHorizontal: If true, centers this child horizontally within its parent. Must be a boolean value, either “true” or “false”.

android:layout_centerInParent: If true, centers this child horizontally and vertically within its parent. Must be a boolean value, either “true” or “false”.

android:layout_centerVertical: If true, centers this child vertically within its parent. Must be a boolean value, either “true” or “false”.

android:layout_toEndOf: Positions the start edge of this view to the end of the given anchor view ID and must be a reference to another resource, in the form “@[+][package:]type:name”.

android:layout_toLeftOf: Positions the right edge of this view to the left of the given anchor view ID and must be a reference to another resource, in the form “@[+][package:]type:name”.

android:layout_toRightOf: Positions the left edge of this view to the right of the given anchor view ID and must be a reference to another resource, in the form “@[+][package:]type:name”.

android:layout_toStartOf: Positions the end edge of this view to the start of the given anchor view ID and must be a reference to another resource, in the form “@[+][package:]type:name”.

LINEAR LAYOUT

android:id: This is the ID which uniquely identifies the layout.
android:baselineAligned: This must be a boolean value, either “true” or “false” and prevents the layout from aligning its children’s baselines.
android:baselineAlignedChildIndex : When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align
android:divider : This is drawable to use as a vertical divider between buttons. You use a color value, in the form of “#rgb”, “#argb”, “#rrggbb”, or “#aarrggbb”.
android:gravity : This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:orientation: This specifies the direction of arrangement and you will use “horizontal” for a row, “vertical” for a column. The default is horizontal.
android:weightSum: Sum up of child weight
Example Code with weight Property

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:background="#0707f1"
android:weightSum="10">
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="3"
    android:background="#f9f003"
    android:orientation="horizontal">
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
   android:layout_marginRight="20dp"
    android:layout_weight="1"/>
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_weight="1"/>
    <Button
        android:id="@+id/button21"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        android:layout_weight="1"/>
</LinearLayout>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="3"
    android:background="#94f50b"
    android:orientation="horizontal">


<Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_weight="1"/>
<Button
    android:id="@+id/button4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    android:layout_weight="1"/>
</LinearLayout>
<LinearLayout
    android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:background="#33111b"
        android:orientation="horizontal">
    <Button
        android:id="@+id/button5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        android:layout_gravity="center"
        android:layout_weight="1"/>
    </LinearLayout>
</LinearLayout>
```

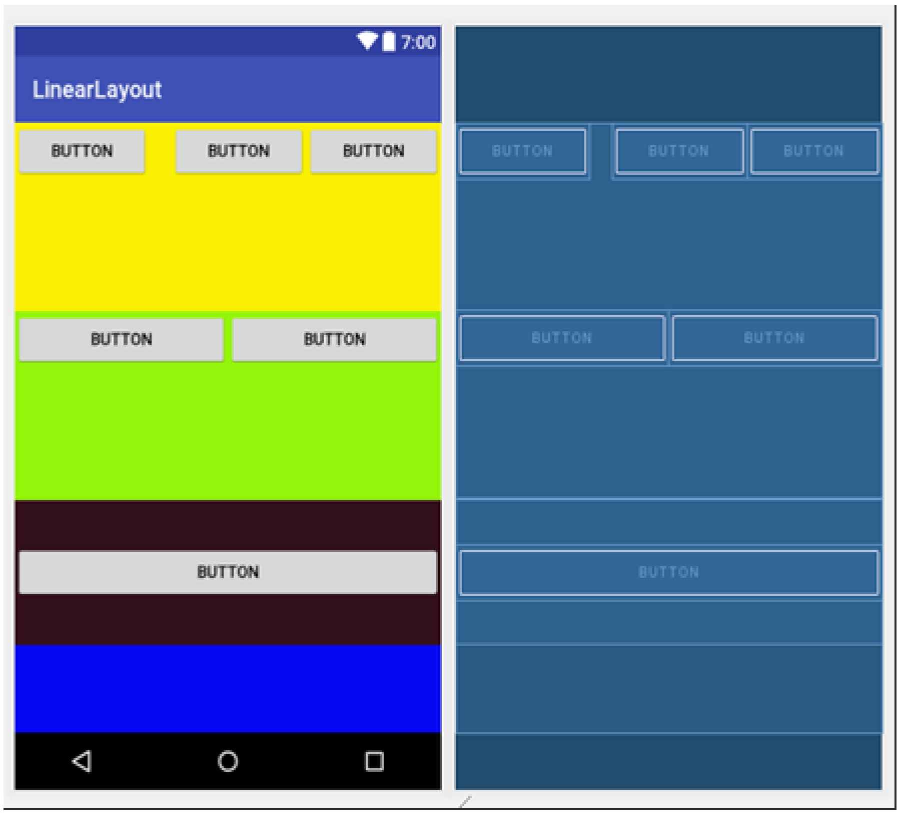


TABLE LAYOUT

TableLayout positions its children into rows and columns. TableLayout containers do not display border lines for their rows, columns, or cells. The table will have as many columns as the row with the most cells. A table can leave cells empty. Cells can span multiple columns, as they can in HTML. You can span columns by using the span field in the TableRow.LayoutParams class.
Note: Cells cannot span multiple rows.

TableRow objects are the child views of a TableLayout (each TableRow defines a single row in the table). Each row has zero or more cells, each of which is defined by any kind of other View. So, the cells of a row may be composed of a variety of View objects, like ImageView or TextView objects. A cell may also be a ViewGroup object (for example, you can nest another TableLayout as a cell).

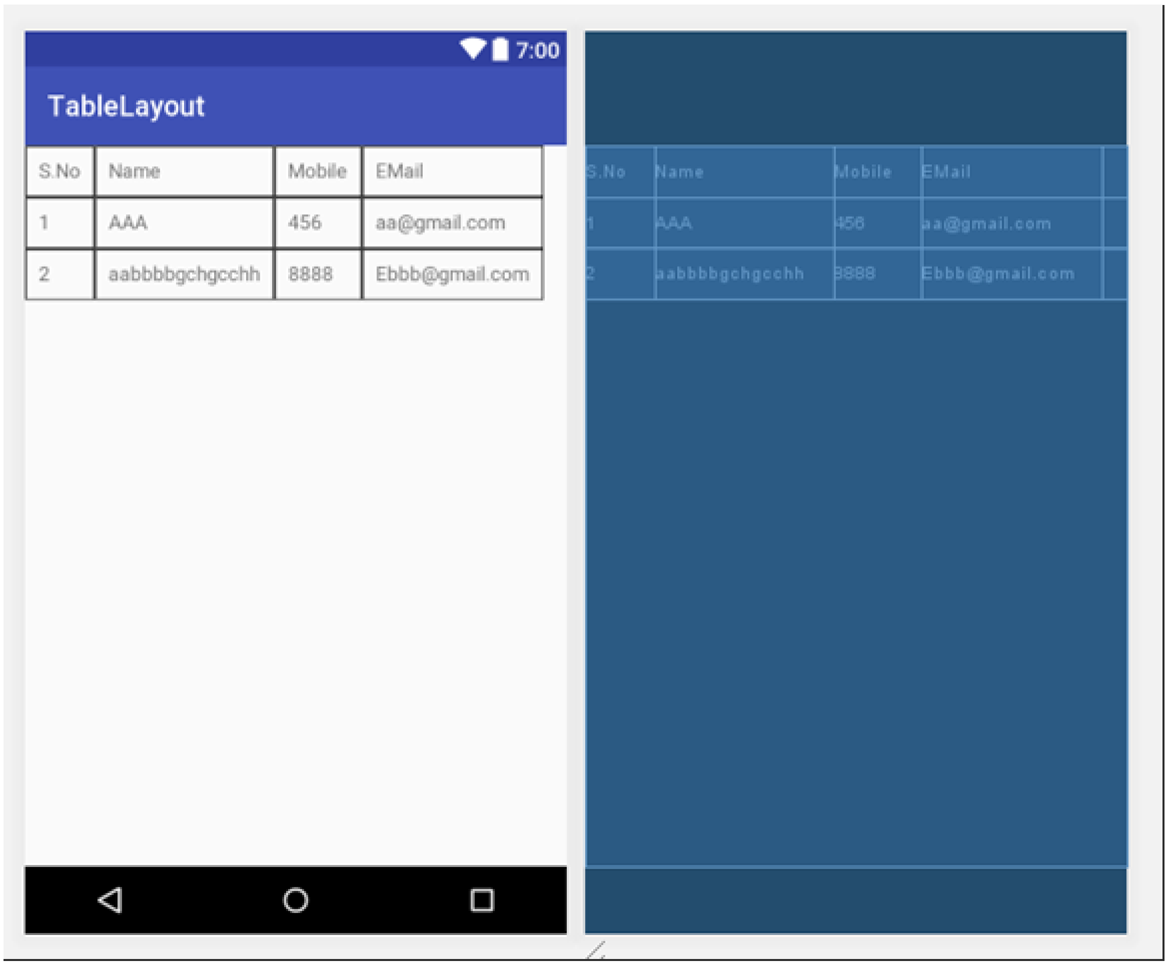
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent"   >
<TableLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content">
   <TableRow
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@drawable/border"
      android:text="S.No"
      android:padding="10dp"/>
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@drawable/border"
      android:text="Name"
      android:padding="10dp"/>
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@drawable/border"
      android:text="Mobile"
      android:padding="10dp"/>
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@drawable/border"
      android:text="EMail"
      android:padding="10dp"/>
</TableRow>
   <TableRow
      android:layout_width="wrap_content"
      android:layout_height="wrap_content">
      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="1"
         android:padding="10dp"
         android:background="@drawable/border"/>
```

```
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="AAA"
            android:padding="10dp"
            android:background="@drawable/border"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="456"
            android:padding="10dp"
            android:background="@drawable/border"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="aa@gmail.com"
            android:padding="10dp"
            android:background="@drawable/border"/>


    </TableRow>
    <TableRow
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="2"
            android:padding="10dp"
            android:background="@drawable/border"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="aabbbbgchgcchh"
            android:padding="10dp"
            android:background="@drawable/border"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="8888"
            android:padding="10dp"
            android:background="@drawable/border"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Ebbb@gmail.com"
            android:padding="10dp"
            android:background="@drawable/border"/>
    </TableRow>
</TableLayout>
</LinearLayout>
```

For Table layout if you want borders need to implement below code and apply as a background for each widget

In drawabale folder create one xml with some name with below code

```
<?xml version="1.0" encoding="utf-8"?>
<shape
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:shape="rectangle">
   <stroke android:color="#000"
      android:width="1dp"></stroke>
</shape>
```



S.No	Name	Mobile	EMail
1	AAA	456	aa@gmail.com
2	aabbbbgchgcchh	8888	Ebbb@gmail.com

FRAME LAYOUT

FrameLayout is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other. You can, however, add multiple children to a FrameLayout

and control their position within the FrameLayout by assigning gravity to each child, using the android:layout_gravity attribute.
Child views are drawn in a stack, with the most recently added child on top. The size of the FrameLayout is the size of its largest child (plus padding), visible or not (if the FrameLayout's parent permits). Views that are GONE are used for sizing only if setConsiderGoneChildrenWhenMeasuring() is set to true.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/frame"
    android:orientation="vertical" android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:measureAllChildren="true"  >
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone"
        android:src="@drawable/ic_launcher"/>
</FrameLayout>
```

android:id: This is the ID which uniquely identifies the layout.
android:foreground: This defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:foregroundGravity: Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:measureAllChildren: Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false.

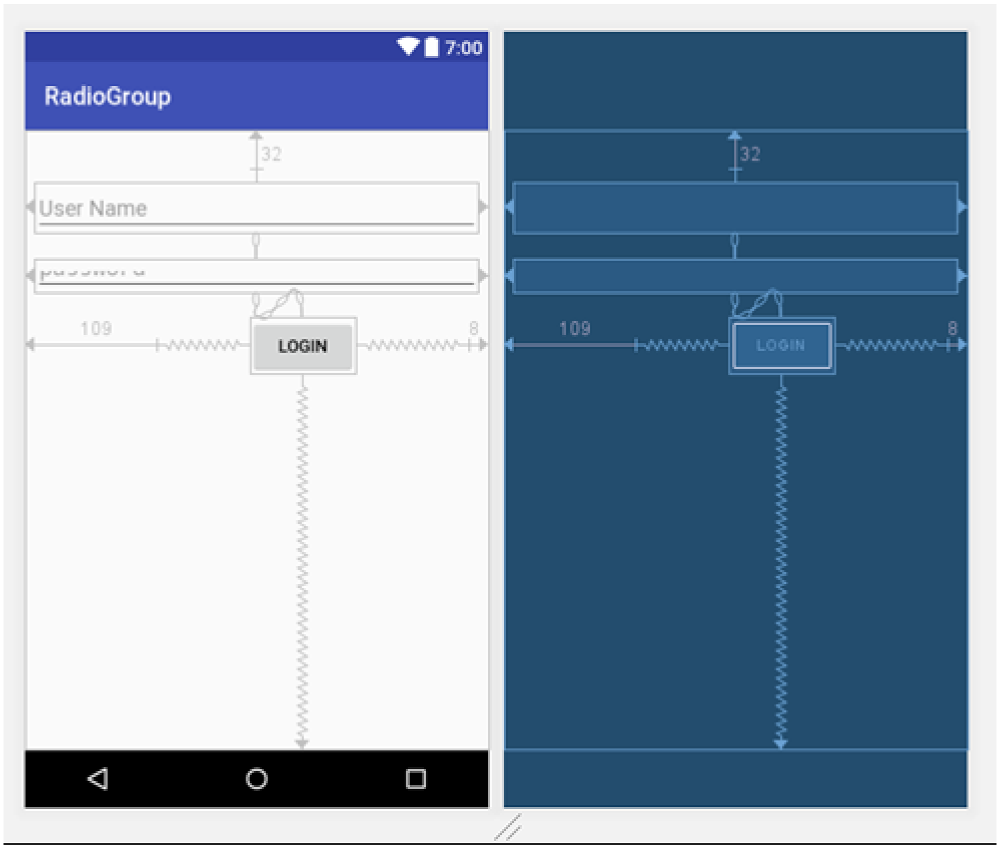
CONSTRAINT LAYOUT

The ConstraintLayout is a powerful new class, imagine a RelativeLayout on steroids – yea, that's the ConstraintLayout. It allows us to lay out child views using 'constraints' to define position based relationships between different views found in our layout. The aim of the ConstraintLayout is to help reduce the number of nested views, which will improve the performance of our layout files. The layout class also makes it easier for us to define layouts than when using a RelativeLayout as we can now anchor any side of a view with any side of another, rather than having to place a whole view to any side of another

```
<?xml version="1.0" encoding="utf-8"?>
```

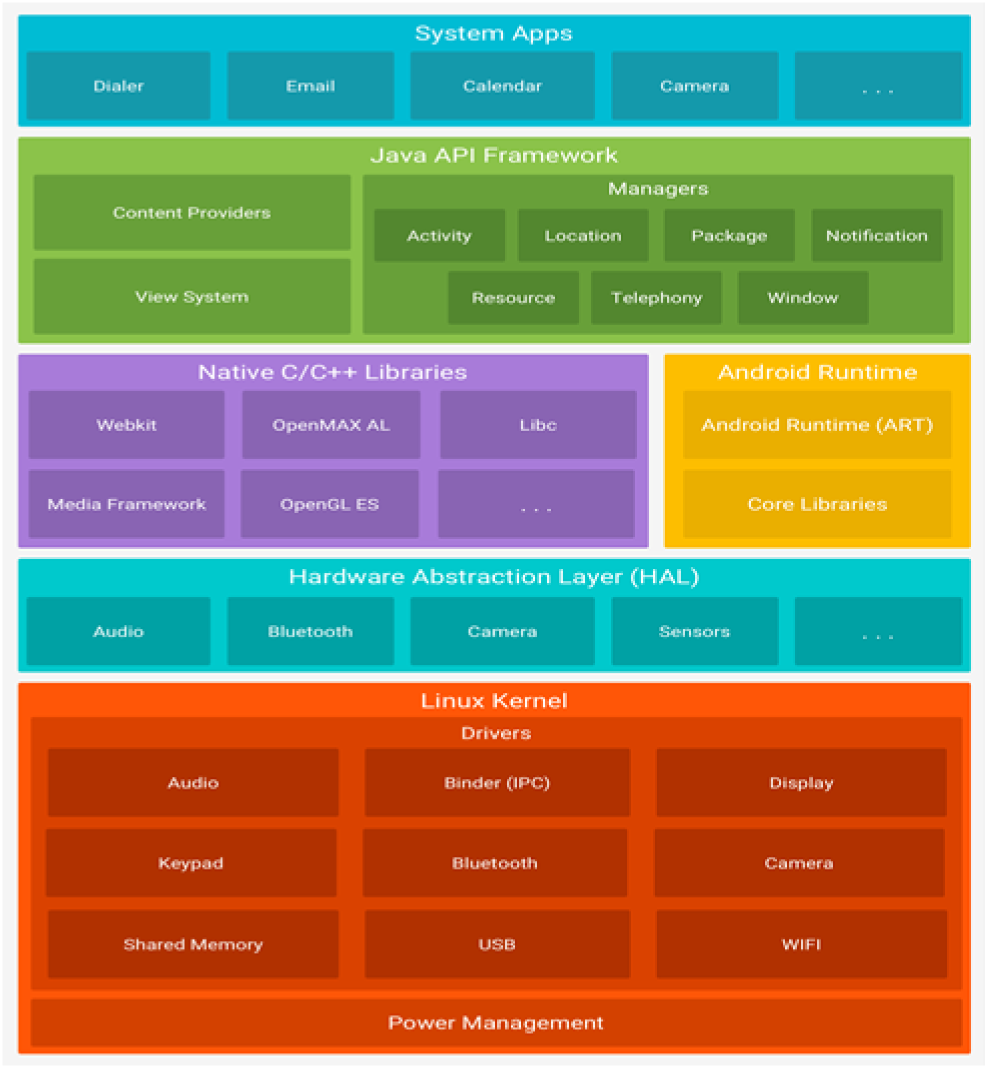
```
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="androiindians.radiogroup.Main2Activity">
   <EditText
      android:id="@+id/editText"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:ems="10"
      android:inputType="textPersonName"
      android:hint="User Name"
      app:layout_constraintStart_toStartOf="parent"
      android:layout_marginStart="8dp"
      app:layout_constraintEnd_toEndOf="parent"
      android:layout_marginEnd="8dp"
      app:layout_constraintTop_toTopOf="parent"
      android:layout_marginTop="32dp"
      android:layout_marginBottom="8dp"
      app:layout_constraintBottom_toTopOf="@+id/editText2" />
   <EditText
      android:id="@+id/editText2"
      android:layout_width="match_parent"
      android:layout_height="29dp"
      android:ems="10"
      android:hint="password"
      android:inputType="textPassword"
      tools:layout_editor_absoluteX="0dp"
      tools:layout_editor_absoluteY="91dp"
      app:layout_constraintStart_toStartOf="parent"
      android:layout_marginStart="8dp"
      app:layout_constraintEnd_toEndOf="parent"
      android:layout_marginEnd="8dp"
      android:layout_marginTop="16dp"
      app:layout_constraintTop_toBottomOf="@+id/editText"
      android:layout_marginBottom="16dp"
      app:layout_constraintBottom_toTopOf="@+id/button" />
   <Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Login"
      tools:layout_editor_absoluteY="143dp"
      app:layout_constraintStart_toStartOf="parent"
      android:layout_marginStart="109dp"
      android:layout_marginLeft="109dp"
      app:layout_constraintEnd_toEndOf="parent"
      android:layout_marginEnd="8dp"
      android:layout_marginRight="8dp"
      app:layout_constraintHorizontal_bias="0.43"
      app:layout_constraintBottom_toBottomOf="parent"
      android:layout_marginBottom="320dp"
```

```
        android:layout_marginTop="32dp"
        app:layout_constraintTop_toBottomOf="@+id/editText2" />
</android.support.constraint.ConstraintLayout>
```



ARCHITECTURE OF ANDROID

Architectute is nothing but physical structural plan or block diagram of some view.
Android is Linux based software stack , it is open source for lot of devices. For developers they are giving full acces to Frame work API acces to develop core level applications, because of that only manufactrers are customizing their own features.
The following diagram is shows rhe components of android.



Android architecture has total 6 layers.

SYSTEM APPS

The top most layer is system APPS. Normally normal devices will come with Contacts, Messages, Browsers, calenders etc.. so here user can install any number of applications which they are using, some third party applications may be we can set as default apps like browers, Messengars and keyboards
These all very helpful to for because if developer is developing his own applications he can those functionalities for example if we developemnet any application related to SMS just developer invoke the services then next portion of the service like sending sms & sms status will be done android system only

JAVA API FRAMEWORK

The Android OS complete feature set is avaible for devlopers in the form of APIs those APIs are written by Java language. This are very helpful to developers to simplify their app development, even developer can use cor part of architecture alos to build apps like components and services which are included in API framework. The following are few of the APIs

- View System:

View System is for building of user interface in android like textview, button, grid, list, image,vidoes etc. here using XML/Java we can build UI is very very rich, simple & attractive.

- Content Providers:

Content providers is nothing but we can access the data from other applications like while sending the message we are accessing contacts data & in now days all social media applicatiosn we are accessing the contacts, gallery etc.
Managers:
These are also API but non coding portion providing layouts, images, strings, colors, default styles etc.

- Resource

Resource manager provides acces to resources for our developemnet such as Strings, Images, nimations, graphics, colors, vidoes etc.

- Notification

Notification Manger will help enable the custom notification in status bar.

- Activity

Activity Manager as very very important API in android it is only controlling the navigation of each screen in our apps using lifecycle methods.

- Location

This Api could help to acces the geo location,navigation using Services of Android Architecture.

- Package

Each package contains multiple java classes and it can control all the functionalities of those classes.

- Telephony

It can communicate with networks like mobile data, wifi, hotspot & sharing data etc.

- Window

It can handle the multiple windows in our device. For example symultantously while listing the songs chatting with some one so its handling two two different windows at a time like same some windows cache memory & browser windows etc.

1. Native C/C++ Libraries

Many of the android componnts and servicew are build from native code those android runtime & Hardware abstraction layers because native libraries wriiten in C & C++.

- Webkit

It can support all type of browsers.

- Open Max AL

It should create run time environment.

- Libc

Lot of componets are developed by C & C++ so it always requied support of Libc.

- Meida framework

Android apps can support all type of media like jpg, png, mp3, mp4 etc.

- Open GLES

It can support 2D & 3D graphics

1. Android Run Time
2. Android Runtime (ART)

From android 5.0 and above version of android with its own process and its own instance of Android Runtime. ART is designed for low memory device by excuting the DEX (byte code) files to run on virtual

devices,and bytecode is designed special optimized for low memory devices.
ART features

- Optimized garbage collection (GC)
- Just In Time complier (JIT)
- Ahead of Time (AOT)
- Better debugging support
- Supporting Crash reports

Note:
Previosly Lower version (less than 5.0) is run in Dalvik virtual machine, now as well as it can run in ART also, then it can run DVM as well, but vicevera is not possible.

- Core Librariries

Android application also includes a set of core libraries that provide most of the runtime functionality of the Java programming language, and it includes android 8.0 features as well

1. Android Abstraction Layer(HAL)

The android hardware abstraction layer provides User interfaces that expose hardware capabilities of device with higher version of Java. It can lot of hard ware components, suac as Audio, Bluetooth, Camera, sensors. When access device hardware through a framework API Android system loads the library module for that hardware component.

1. Linux Kernel

Linux kernel is hardware layer. For example, the ART relies on the Linux kernel for functionalities such as threading and low-level memory management. Linux kernel allows Android to take advantage of key features like develop hardware drivers for a well-known kernel such as Audio, Keypad, Shraed memory, Usb, Bluetooth, Binder(IPC)Display, Camera, Wifi, Power Management etc.

COMMON UI COMPONENTS AND EVENTS

TEXTVIEW & BUTTON

TextView:

- A TextView is a user interface element used to display text to the user.
- It can be used to display static text, such as labels, headers, descriptions, etc.
- TextViews can be customized with various attributes like text size, text color, font style, alignment, etc., to match the design requirements of the app.
- They are commonly used in combination with other UI elements to provide context or instructions to the user.
- TextViews can also be used dynamically to display data fetched from databases or APIs

Button:

- A Button is a user interface element used to perform actions or trigger events when clicked.
- It typically displays a labeled text or an icon, indicating the action it performs when clicked.
- Buttons can be customized with various attributes like text, background color, text color, padding, etc.
- They are often used to implement user interactions like submitting a form, navigating to another screen, or triggering some functionality.
- Buttons can also be styled to match the overall design language of the app, providing a consistent user experience.

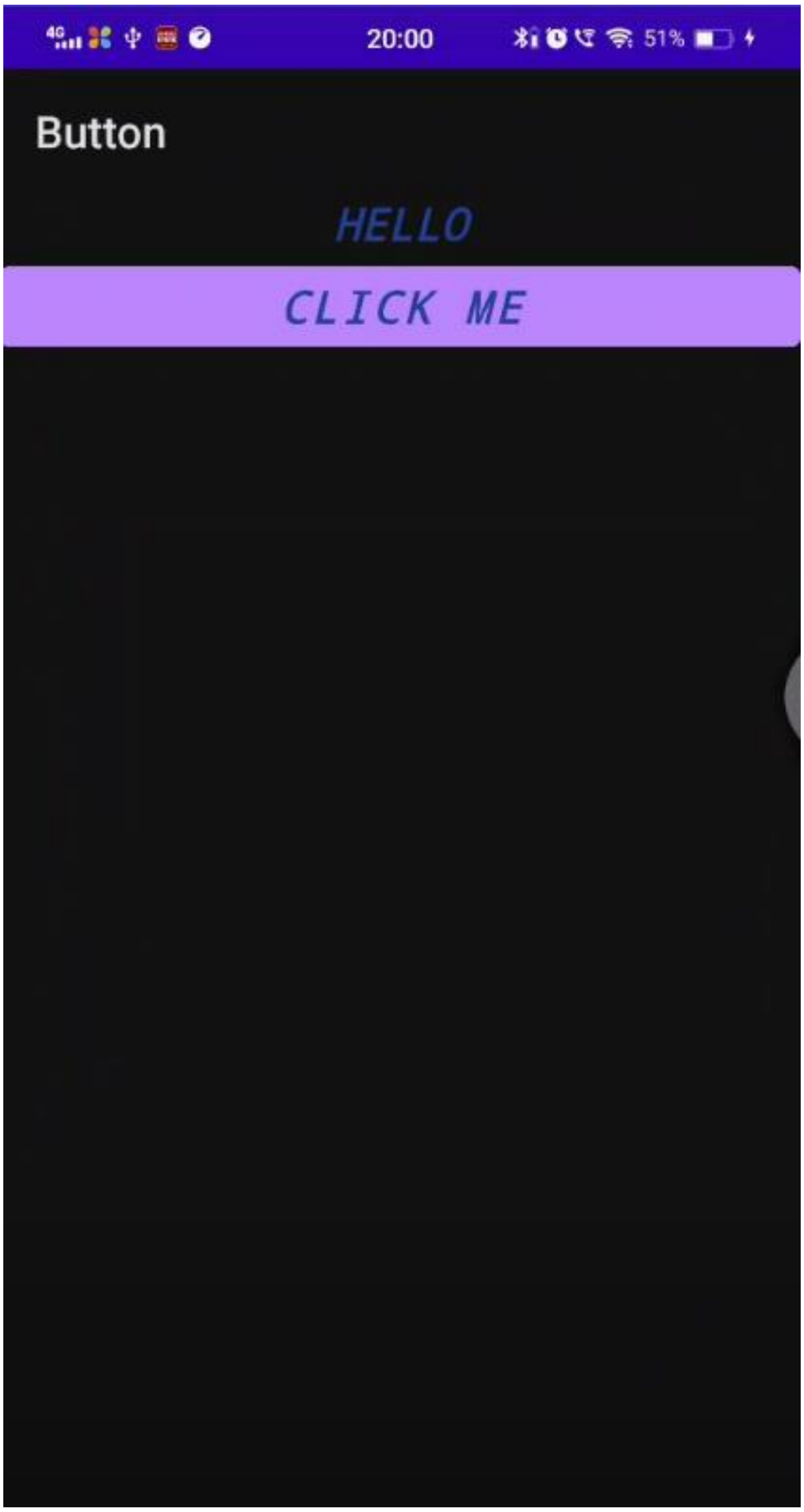
activity_main.xml

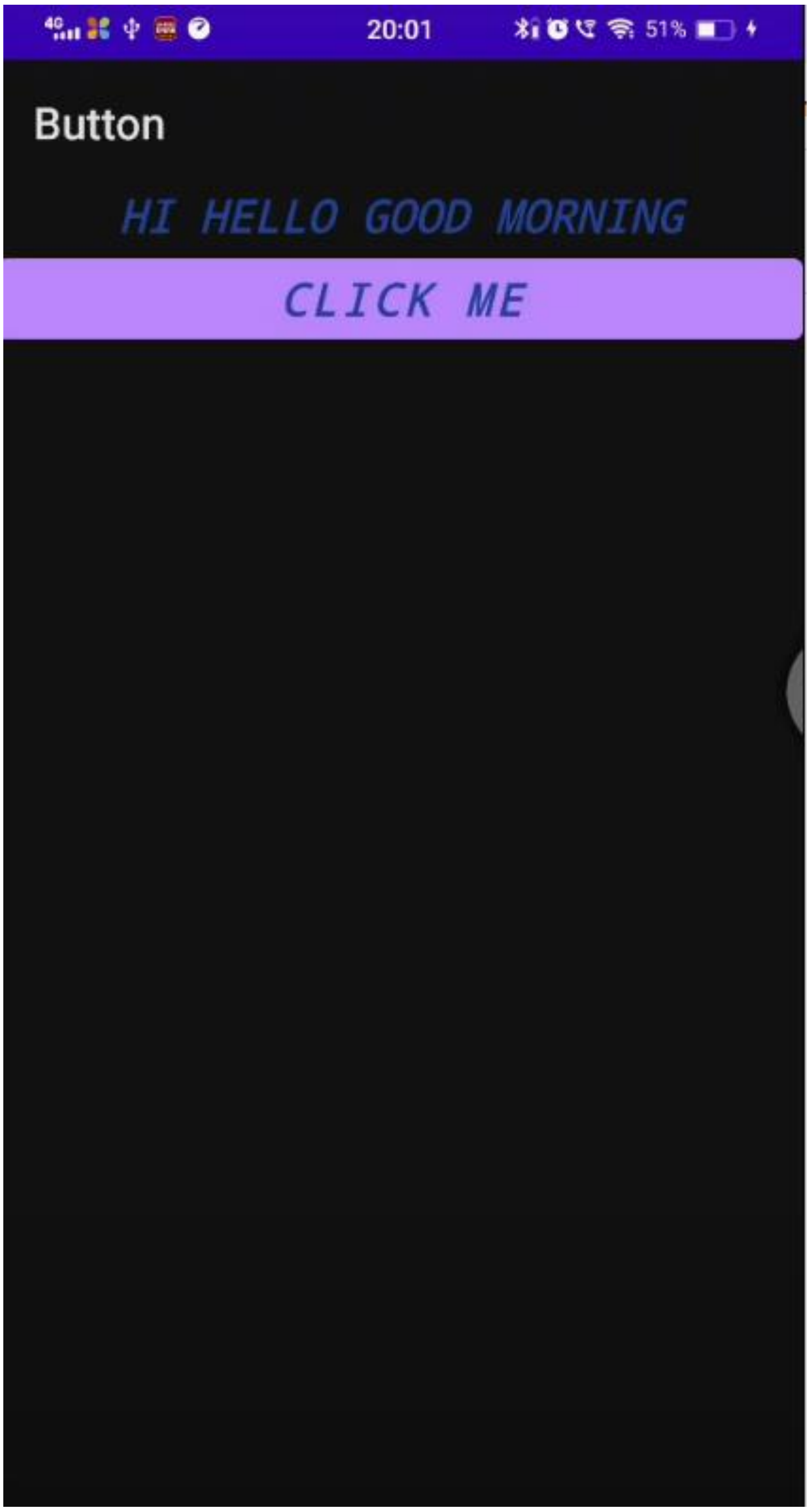
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="androindian.MainActivity">


    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="hello"
        android:textSize="20sp"
        android:textColor="#234098"
        android:textAllCaps="true"
        android:textStyle="bold|italic"
        android:id="@+id/tv"
        android:typeface="monospace"
        android:gravity="center"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Click Me"
        android:textSize="20sp"
        android:textColor="#234098"
        android:textAllCaps="true"
        android:textStyle="bold|italic"
        android:id="@+id/bt"
        android:typeface="monospace"
        android:gravity="center"/>
</LinearLayout>
```

MainActivity.kt

```
package com.androinidan.button


import androidx.appcompat.app.AppCompatActivity
import android.widget.TextView
import android.os.Bundle
import android.view.View
import android.widget.Button
import com.androinidan.button.R


class MainActivity : AppCompatActivity() {
    var textView: TextView? = null
    var button: Button? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        textView = findViewById<View>(R.id.tv) as TextView
        button = findViewById<View>(R.id.bt) as Button
        button!!.setOnClickListener {
            textView!!.text = "Hi Hello Good morning"
        }
    }
}
```

output


20:00
51%
Button
HELLO
CLICK ME



CHECKBOX

a CheckBox is a user interface element that allows users to toggle between two states - checked and unchecked. It is typically used when users need to select one or more options from a list of choices.
Here are some key points about CheckBoxes in Android:

- Appearance:
    - A CheckBox typically consists of a square or rectangular box followed by a label (text).
    - When checked, a checkmark or a tick appears inside the box to indicate the selection.
    - When unchecked, the box remains empty.
- Usage:
    - CheckBoxes are commonly used in forms, settings screens, and lists where users need to make multiple selections.
    - They are useful for allowing users to specify preferences, select items from a list, or indicate agreement to terms and conditions.
- Attributes:
    - CheckBoxes can be customized using various attributes to control their appearance and behavior.
    - Common attributes include text (label), checked state, text color, text size, padding, background, etc.
    - You can also specify an optional listener to be notified when the CheckBox state changes, allowing you to perform actions based on user interactions.
- Interactivity:
    - CheckBoxes respond to user input by toggling between the checked and unchecked states when clicked.
    - They can be configured to be either checked or unchecked by default.
    - When used in groups (such as within a CheckBox group), multiple CheckBoxes can be selected simultaneously.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">


    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#FF5722"
        android:textSize="30sp"
        android:textStyle="bold"
        android:textAllCaps="true"
        android:padding="10dp"
        android:layout_margin="10dp"
        android:gravity="center"
        android:layout_gravity="center"
        android:id="@+id/tv1"
        />


    <CheckBox
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Android"
        android:textColor="#FF5722"
        android:textSize="30sp"
        android:textStyle="bold"
        android:textAllCaps="true"
        android:padding="10dp"
        android:layout_margin="10dp"
        android:gravity="center"
        android:layout_gravity="center"
        android:id="@+id/check"
        ></CheckBox>



    </LinearLayout>
```

MainActivity.kt

```
package com.androinidan.checkboxkotlin


import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.CheckBox
import android.widget.CompoundButton
import android.widget.Switch
import android.widget.TextView
import android.widget.ToggleButton


class MainActivity : AppCompatActivity() {


    var check: CheckBox?=null
    var tv: TextView?=null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


        tv = findViewById(R.id.tv1)


        check = findViewById(R.id.check)




        check!!.setOnClickListener{
            if(check!!.isChecked)
                tv!!.text=("Checked")
            else
                tv!!.text=("UnChecked")


        }

    }


}
```

TOGGLE BUTTON

A ToggleButton in Android is a UI element that represents a two-state button, similar to a physical toggle switch. It allows users to toggle between two states, typically "on" and "off" or "checked" and "unchecked". ToggleButtons are useful for enabling or disabling a feature, activating or deactivating a setting, or switching between two modes.
Here are some key points about ToggleButtons in Android:

- Appearance:
    - A ToggleButton typically consists of a rectangular button with two states: pressed (activated) and unpressed (deactivated).
    - The button's appearance changes to indicate its current state, often with a different background color or visual indicator.

    - ToggleButtons usually display text to describe each state, such as "On" and "Off", "Yes" and "No", or "Enabled" and "Disabled".
- Usage:
    - ToggleButtons are commonly used in settings screens, control panels, and other parts of the user interface where users need to quickly switch between two states.
    - They provide a clear and intuitive way for users to control settings or enable/disable features with a single tap.
- Attributes:
    - ToggleButtons can be customized using various attributes to control their appearance, text, and behavior.
    - Common attributes include text for each state, text color, text size, background, padding, etc.
    - You can also specify an optional listener to be notified when the ToggleButton state changes, allowing you to perform actions based on user interactions.
- Interactivity:
    - ToggleButtons respond to user input by toggling between the two states when clicked.
    - The button's state automatically switches between "on" and "off" each time it's clicked.
    - You can programmatically set the initial state of the ToggleButton using the setChecked() method.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#FF5722"
        android:textSize="30sp"
        android:textStyle="bold"
        android:textAllCaps="true"
        android:padding="10dp"
        android:layout_margin="10dp"
        android:gravity="center"
        android:layout_gravity="center"
        android:id="@+id/tv1"
        />


    <ToggleButton
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="#FF5722"
        android:textSize="30sp"
        android:textStyle="bold"
        android:textAllCaps="true"
        android:padding="10dp"
        android:layout_margin="10dp"
        android:gravity="center"
        android:layout_gravity="center"
        android:id="@+id/toggle"></ToggleButton>



</LinearLayout>
```

MainActivity.kt

```
package com.androinidan.checkboxkotlin


import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.CheckBox
import android.widget.CompoundButton
import android.widget.Switch
import android.widget.TextView
import android.widget.ToggleButton


class MainActivity : AppCompatActivity() {


    var tv: TextView?=null
    var toogle: ToggleButton?=null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


        tv = findViewById(R.id.tv1)


        toogle=findViewById(R.id.toggle)
        toogle!!.setOnClickListener{
            if(toogle!!.isChecked)
                tv!!.text=("TON")
            else
                tv!!.text=("TOFF")
        }


    }


}
```

SWITCH

a Switch is a UI element that allows users to toggle between two states - typically "on" and "off". It's similar to a ToggleButton but has a more modern and compact design, resembling a physical switch.
Here are some key points about Switches in Android:

- Appearance:
  - A Switch consists of a small rectangular track and a thumb (also called a handle or knob) that slides horizontally along the track.
  - The thumb's position indicates the current state of the Switch: to the left for "off" and to the right for "on".
  - The track's color or appearance often changes to indicate the active and inactive states.
- Usage:
  - Switches are commonly used in settings screens, control panels, and other parts of the user interface where users need to quickly switch between two states.
  - They provide a clear and intuitive way for users to enable or disable a feature, activate or deactivate a setting, or switch between two modes.
- Attributes:
  - Switches can be customized using various attributes to control their appearance and behavior.

- Common attributes include track and thumb colors, track and thumb drawable resources, text for each state, padding, etc.
- You can also specify an optional listener to be notified when the Switch state changes, allowing you to perform actions based on user interactions.
- Interactivity:
    - Switches respond to user input by sliding the thumb horizontally along the track to toggle between the two states.
    - The Switch's state automatically changes from "off" to "on" or vice versa each time the user interacts with it.
    - You can programmatically set the initial state of the Switch using the setChecked() method.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#FF5722"
        android:textSize="30sp"
        android:textStyle="bold"
        android:textAllCaps="true"
        android:padding="10dp"
        android:layout_margin="10dp"
        android:gravity="center"
        android:layout_gravity="center"
        android:id="@+id/tv1"
        />



    <Switch
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Switch"
        android:textColor="#FF5722"
        android:textSize="30sp"
        android:textStyle="bold"
        android:textAllCaps="true"
        android:padding="10dp"
        android:layout_margin="10dp"
        android:gravity="center"
        android:layout_gravity="center"
        android:id="@+id/switch1"></Switch>


</LinearLayout>
```

MainActivity.kt

```
package com.androinidan.checkboxkotlin


import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.CheckBox
import android.widget.CompoundButton
import android.widget.Switch
import android.widget.TextView
import android.widget.ToggleButton


class MainActivity : AppCompatActivity() {


    var tv: TextView?=null
    var swit: Switch?=null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


        tv = findViewById(R.id.tv1)


        swit=findViewById(R.id.switch1)


        swit!!.setOnClickListener{
            if(swit!!.isChecked)
                tv!!.text=("SON")
            else
                tv!!.text=("SOFF")
        }
    }


}
```

RADIO GROUP AND RADIO BUTTON

In Android, a RadioGroup is a layout manager used to group multiple RadioButton elements together. RadioButtons are UI elements that allow users to select one option from a set of mutually exclusive options. Here's an explanation of each:

- RadioGroup:
    - A RadioGroup is a layout container used to hold multiple RadioButton elements.
    - It ensures that only one RadioButton within the group can be selected at a time, enforcing mutual exclusivity.
    - When a RadioButton is selected within a RadioGroup, any previously selected RadioButton within the same group is automatically deselected.
    - RadioButtons within the same RadioGroup are typically aligned vertically or horizontally, depending on the layout configuration.
    - RadioGroups are commonly used in forms, questionnaires, and settings screens where users need to make a single selection from a list of options.
- RadioButton:
    - A RadioButton is a UI element that represents a single choice in a set of mutually exclusive options.
    - It typically consists of a circular button followed by a label (text).
    - When selected, a dot or circle appears inside the button to indicate the selection.
    - RadioButton elements are placed within a RadioGroup to ensure mutual exclusivity.
    - RadioButtons are commonly used in conjunction with RadioGroups to allow users to select one option from a list of choices.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#FF5722"
        android:textSize="30sp"
        android:textStyle="bold"
        android:textAllCaps="true"
        android:padding="10dp"
        android:layout_margin="10dp"
        android:gravity="center"
        android:layout_gravity="center"
        android:id="@+id/tv1"
        />



    <RadioGroup
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:id="@+id/rg">
        <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/r1"
            android:text="Male"></RadioButton>
        <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/r2"
            android:text="Female"></RadioButton>
    </RadioGroup>


</LinearLayout>
```

MainActivity.kt

```
package com.androinidan.checkboxkotlin


import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.CheckBox
import android.widget.CompoundButton
import android.widget.RadioButton
import android.widget.RadioGroup
import android.widget.Switch
import android.widget.TextView
import android.widget.ToggleButton


class MainActivity : AppCompatActivity() {


     var radiogroup: RadioGroup?=null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


        tv = findViewById(R.id.tv1)



        radiogroup=findViewById(R.id.rg)
        r1=findViewById(R.id.r1)


        radiogroup.setOnCheckedChangeListener{ group, checkedId ->
            if(checkedId==R.id.r1)
                tv!!.text=("MALE")
            else
                tv!!.text=("FEMALE")
        }


        }
    }
```

SEEKBAR

A SeekBar in Android is a UI element that allows users to select a value within a specified range by sliding a thumb along a horizontal or vertical track. SeekBars are commonly used for tasks such as adjusting volume, seeking through media playback, or setting progress indicators.
Here are some key points about SeekBars in Android:

- Appearance:
    - A SeekBar consists of a track and a thumb (also known as a slider or handle).

  - The track represents the range of values that the SeekBar can take, typically displayed as a horizontal or vertical line.
  - The thumb is a draggable icon that can be moved along the track to select a value.
  - The position of the thumb indicates the current selected value within the range.
- Usage:
  - SeekBars are commonly used in media players, audio/video recording apps, settings screens, and other parts of the user interface where users need to adjust a value within a range.
  - They provide a visual and interactive way for users to control settings, adjust parameters, or seek through content.
- Attributes:
  - SeekBars can be customized using various attributes to control their appearance and behavior.
  - Common attributes include track and thumb colors, track height, thumb size, padding, and the range of values that the SeekBar can take.
  - You can also specify an optional listener to be notified when the SeekBar value changes, allowing you to perform actions based on user interactions.
- Interactivity:
  - Users interact with SeekBars by dragging the thumb along the track to select a value.
  - As the thumb is moved, the selected value is updated accordingly.
  - You can programmatically set the initial value of the SeekBar using the setProgress() method.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity" >


   <SeekBar
      android:id="@+id/seekBar1"
      android:layout_width="300dp"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentTop="true"
      android:layout_marginTop="26dp"
      android:max="10"/>
```

```
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/seekBar1"
        android:layout_marginLeft="29dp"
        android:layout_marginTop="14dp" />


</RelativeLayout>
```

MainActivity.kt

```
package com.androidian.seekbar


import android.os.Bundle
import android.view.View
import android.widget.SeekBar
import android.widget.SeekBar.OnSeekBarChangeListener
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity


class MainActivity : AppCompatActivity() {
    private var seekBar: SeekBar? = null
    private var textView: TextView? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        seekBar = findViewById<View>(R.id.seekBar1) as SeekBar
        textView = findViewById<View>(R.id.textView1) as TextView
        // Initialize the textview with '0'
        textView!!.text = seekBar!!.progress.toString() + "/" + seekBar!!.max
        seekBar!!.setOnSeekBarChangeListener(
            object : OnSeekBarChangeListener {
                var progress = 0
                override fun onProgressChanged(
                    seekBar: SeekBar,
                    progresValue: Int, fromUser: Boolean
                ) {
                    progress = progresValue
                }


                override fun onStartTrackingTouch(seekBar: SeekBar) {
                    // Do something here,
                    //if you want to do anything at the start of
                    // touching the seekbar
                }
                override fun onStopTrackingTouch(seekBar: SeekBar) {
                    // Display the value in textview
                    textView!!.text = progress.toString() + "/" + seekBar.max
                }
            })
    }
}
```

RATING BAR

A RatingBar in Android is a UI element that allows users to provide a rating or feedback by selecting a number of stars from a predefined range. It is commonly used in apps where users are asked to rate products, services, or content.
Here are some key points about RatingBars in Android:

- Appearance:
    - A RatingBar consists of a row of stars or other symbols representing the rating scale.
    - Each star can be filled (selected) or empty (unselected), indicating the user's rating.
    - The number of stars displayed represents the rating scale's range, typically from 0 to 5.
    - Depending on the configuration, users can either tap on individual stars or swipe across the RatingBar to select their rating.
- Usage:
    - RatingBars are commonly used in review systems, feedback forms, product ratings, and other parts of the user interface where users need to provide a numerical rating.
    - They provide a simple and intuitive way for users to express their opinions or preferences by selecting a number of stars.
- Attributes:
    - RatingBars can be customized using various attributes to control their appearance and behavior.
    - Common attributes include the number of stars, star size, star spacing, star color (selected and unselected), and the initial rating value.

  - You can also specify an optional listener to be notified when the user's rating changes, allowing you to perform actions based on user interactions.
- Interactivity:
  - Users interact with RatingBars by tapping on individual stars to select their rating.
  - Depending on the configuration, users may also be able to swipe across the RatingBar to select their rating.
  - The selected rating is represented by the number of filled stars, while the unselected stars remain empty.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Radio Button Example"
        android:textAllCaps="true"
        android:textStyle="italic"
        android:textSize="20sp"
        android:textColor="#309"
        android:padding="5dp"
        android:layout_margin="5dp"
        android:id="@+id/tv1"/>


    <RatingBar
        android:id="@+id/ratingBar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:stepSize="1"/>


</LinearLayout>
```

MainActivity.kt

```
package com.androinidan.radiobutton


import android.annotation.SuppressLint
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.ImageButton
import android.widget.ImageView
import android.widget.RadioGroup
import android.widget.RatingBar
import android.widget.TextView
import android.widget.Toast


class MainActivity : AppCompatActivity() {


    var ratingbar: RatingBar?=null
    @SuppressLint("WrongConstant")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


        ratingbar = findViewById(R.id.ratingBar)


        ratingbar!!.setOnRatingBarChangeListener { ratingBar, fl, b ->
            Toast.makeText(applicationContext,"you seleted"+fl,Toast.LENGTH_LONG).show()
        }
    }
}
```

IMAGE BUTTON

An ImageButton in Android is a UI element that displays a clickable image. It's essentially a button with an image instead of text. ImageButton allows users to perform actions or trigger events when the image is clicked.
Here are some key points about ImageButtons in Android:

- Appearance:
  - An ImageButton displays an image as its content instead of text.
  - The image can be any drawable resource, such as PNG, JPG, SVG, etc.
  - ImageButton supports both fixed and adjustable sizes, depending on the layout configuration.

  - The appearance of the ImageButton can be customized using various attributes, including padding, background, scale type, etc.
- Usage:
  - ImageButtons are commonly used to provide a visual representation of actions or functions in the user interface.
  - They are useful for tasks such as navigating to another screen, submitting a form, playing media, toggling settings, etc.
  - ImageButtons can be placed anywhere in the layout and can serve as standalone buttons or as part of a larger UI component.
- Attributes:
  - ImageButtons can be customized using various attributes to control their appearance and behavior.
  - Common attributes include the image source (src), image scale type, background color, padding, etc.
  - You can also specify an optional listener to be notified when the ImageButton is clicked, allowing you to perform actions based on user interactions.
- Interactivity:
  - ImageButtons respond to user input by triggering an action or event when clicked.
  - When the user taps on the ImageButton, the specified action is performed, such as navigating to another activity, executing a function, or displaying a message.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity"
   android:orientation="vertical">


   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Radio Button Example"
      android:textAllCaps="true"
      android:textStyle="italic"
      android:textSize="20sp"
      android:textColor="#309"
```

```
        android:padding="5dp"
        android:layout_margin="5dp"
        android:id="@+id/tv1"/>


    <ImageButton
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/ib"
        android:src="@mipmap/download"></ImageButton>
</LinearLayout>
```

MainActivity.kt

```
package com.androinidan.radiobutton


import android.annotation.SuppressLint
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.ImageButton
import android.widget.ImageView
import android.widget.RadioGroup
import android.widget.RatingBar
import android.widget.TextView
import android.widget.Toast


class MainActivity : AppCompatActivity() {


    var ib: ImageButton?=null

    @SuppressLint("WrongConstant")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


        ib=findViewById(R.id.ib)


        ib!!.setOnClickListener{
            Toast.makeText(applicationContext,"You seletcd imagebutton",5000).show()
        }


    }
}
```

IMAGEVIEW

An ImageView in Android is a UI element used to display images within an app's user interface. It's one of the most commonly used widgets for showing images, whether they are loaded from resources within the app or fetched dynamically from external sources like the internet.
Here are some key points about ImageViews in Android:

- Appearance:
  - An ImageView displays a single image, which can be loaded from various sources such as drawables, files, URIs, or network URLs.
  - The image can be displayed using various scale types, such as center-crop, fit-center, center-inside, etc., to control how it fits within the ImageView's boundaries.
  - ImageView supports both fixed and adjustable sizes, depending on the layout configuration.
  - The appearance of the ImageView can be customized using various attributes, including padding, background, scale type, etc.
- Usage:
  - ImageViews are used to show images within the app's UI, such as photos, icons, logos, thumbnails, etc.
  - They are versatile and can be used in a wide range of scenarios, including image galleries, profile pictures, product listings, and more.
  - ImageViews can be placed anywhere in the layout, including within ViewGroup containers like LinearLayout, RelativeLayout, ConstraintLayout, etc.
- Attributes:
  - ImageViews can be customized using various attributes to control their appearance and behavior.
  - Common attributes include the image source (src), scale type, background color, padding, visibility, etc.
  - You can load images into an ImageView programmatically using methods like setImageResource(), setImageDrawable(), setImageBitmap(), or by using libraries like Picasso or Glide for more advanced image loading and caching capabilities.

- Interactivity:
  - ImageViews are typically static and don't respond to user input by default.
  - However, you can make them interactive by combining them with other UI elements, such as Buttons or ClickListeners, to perform actions when clicked.
  - For example, you can wrap an ImageView inside a Button or use it as a background for a Button to create a clickable image button.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Radio Button Example"
        android:textAllCaps="true"
        android:textStyle="italic"
        android:textSize="20sp"
        android:textColor="#309"
        android:padding="5dp"
        android:layout_margin="5dp"
        android:id="@+id/tv1"/>



    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/iv"
        android:src="@mipmap/download"></ImageView>



</LinearLayout>
```

MainActivity.kt

```
package com.androinidan.radiobutton


import android.annotation.SuppressLint
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.ImageButton
import android.widget.ImageView
import android.widget.RadioGroup
import android.widget.RatingBar
import android.widget.TextView
import android.widget.Toast


class MainActivity : AppCompatActivity() {


    var iv: ImageView?=null

    @SuppressLint("WrongConstant")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


        iv=findViewById(R.id.iv)




        iv!!.setOnClickListener{
            Toast.makeText(applicationContext,"You seletd
image",Toast.LENGTH_LONG).show()
        }


    }
}
```

TEXT FIELDS

EditText is a user interface element used for capturing user input in the form of text. It allows users to enter and edit text interactively on the screen. EditTexts are commonly used in forms, input fields, search bars, chat interfaces, and other parts of the user interface where textual input is required.

Here are some key points about EditTexts in Kotlin:

- Appearance:
  - An EditText appears as a rectangular input field where users can type text.
  - The appearance of the EditText can be customized using various attributes such as text size, text color, background color, hint text, etc.
  - EditTexts can be single-line or multi-line, depending on the input requirements.
- Usage:
  - EditTexts are used to collect textual input from users, such as names, email addresses, passwords, comments, etc.
  - They support a wide range of input types, including text, numbers, dates, email addresses, phone numbers, and more.
  - EditTexts can be placed anywhere in the layout, including within ViewGroup containers like LinearLayout, RelativeLayout, ConstraintLayout, etc.
- Attributes:
  - EditTexts can be customized using various attributes to control their appearance and behavior.
  - Common attributes include text size, text color, hint text (placeholder text displayed when the EditText is empty), input type (text, number, email, password, etc.), background color, padding, etc.
- Interactivity:
  - EditTexts respond to user input by allowing users to type text using the on-screen keyboard.
  - They provide features such as auto-capitalization, auto-correction, spell-checking, and suggestions (depending on the device's keyboard settings).
  - You can programmatically retrieve the text entered into an EditText using the `text` property or listen for changes using a TextWatcher.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical"

    >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Example Form"
        android:textColor="#E91E63"
        android:textSize="20sp"
        android:textStyle="italic"
        android:textAllCaps="true"
        android:layout_margin="5dp"
        android:padding="5dp"

         />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        >

    <EditText
        android:id="@+id/editTextTextPersonName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        android:hint="textPersonName"
        android:textColor="#E91E63"
        android:textSize="20sp"
        android:textStyle="italic"
        android:textAllCaps="true"
        android:layout_margin="5dp"
        android:padding="5dp"
        android:textColorHint="#3F51B5"/>

    <EditText
        android:id="@+id/editTextTextPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPassword"
        android:hint="textPassword"
        android:textColor="#E91E63"
        android:textSize="20sp"
        android:textStyle="italic"
        android:textAllCaps="true"
        android:layout_margin="5dp"
        android:padding="5dp"
        android:textColorHint="#3F51B5"/>

    <EditText
        android:id="@+id/editTextNumberPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="numberPassword"
        android:hint="numberPassword"
        android:textColor="#E91E63"
        android:textSize="20sp"
        android:textStyle="italic"
        android:textAllCaps="true"
        android:layout_margin="5dp"
        android:padding="5dp"
        android:textColorHint="#3F51B5"/>

    <EditText
        android:id="@+id/editTextTextEmailAddress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textEmailAddress"
        android:hint="textEmailAddress"
        android:textColor="#E91E63"
        android:textSize="20sp"
        android:textStyle="italic"
        android:textAllCaps="true"
        android:layout_margin="5dp"
        android:padding="5dp"
        android:textColorHint="#3F51B5"/>

    <EditText
        android:id="@+id/editTextTextMultiLine"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:gravity="start|top"
        android:inputType="textMultiLine"
        android:textColor="#E91E63"
        android:textSize="20sp"
        android:textStyle="italic"
        android:textAllCaps="true"
        android:layout_margin="5dp"
        android:padding="5dp"
        android:textColorHint="#3F51B5"
        android:hint="textMultiLine"
        android:minLines="10"/>


    <EditText
        android:id="@+id/editTextTextPostalAddress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPostalAddress"
        android:hint="textPostalAddress"
        android:textColor="#E91E63"
        android:textSize="20sp"
        android:textStyle="italic"
        android:textAllCaps="true"
        android:layout_margin="5dp"
        android:padding="5dp"
        android:textColorHint="#3F51B5"/>


    <EditText
        android:id="@+id/editTextTime"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="time"
        android:hint="time"
        android:textColor="#E91E63"
        android:textSize="20sp"
        android:textStyle="italic"
        android:textAllCaps="true"
        android:layout_margin="5dp"
        android:padding="5dp"
        android:textColorHint="#3F51B5"/>


    <EditText
        android:id="@+id/editTextNumber"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="number"
        android:hint="number"
        android:textColor="#E91E63"
        android:textSize="20sp"
        android:textStyle="italic"
        android:textAllCaps="true"
        android:layout_margin="5dp"
        android:padding="5dp"
        android:textColorHint="#3F51B5"/>
    <EditText
        android:id="@+id/editTextNumberSigned"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="numberSigned"
        android:hint="numberSigned"
        android:textColor="#E91E63"
        android:textSize="20sp"
        android:textStyle="italic"
        android:textAllCaps="true"
        android:layout_margin="5dp"
        android:padding="5dp"
        android:textColorHint="#3F51B5"/>

    <EditText
        android:id="@+id/editTextNumberDecimal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="numberDecimal"
        android:hint="numberDecimal"
        android:textColor="#E91E63"
        android:textSize="20sp"
        android:textStyle="italic"
        android:textAllCaps="true"
        android:layout_margin="5dp"
        android:padding="5dp"
        android:textColorHint="#3F51B5"/>

        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/bt"
            android:text="Submit"></Button>
    </LinearLayout>
    </ScrollView>

</LinearLayout>
```

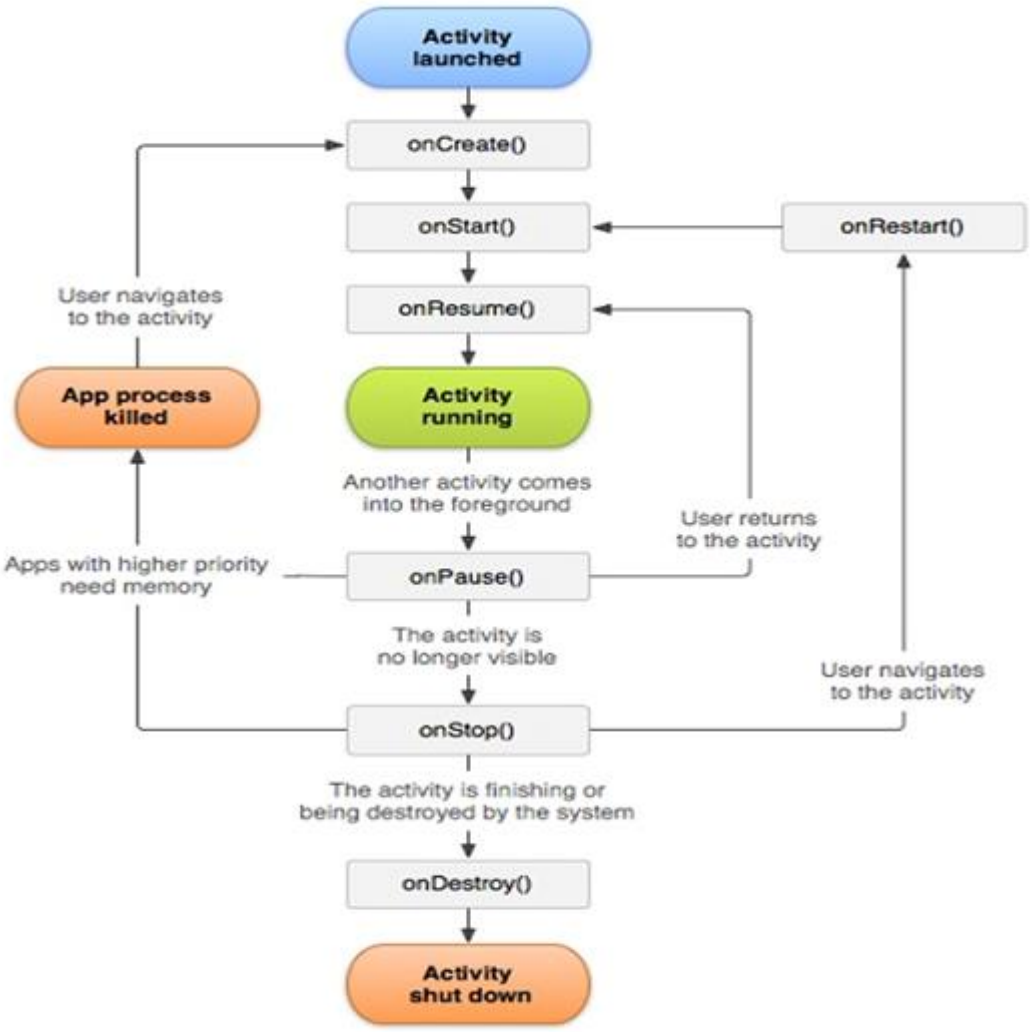
MainActivity.kt

```
package com.androinidan.form

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.Toast

class MainActivity : AppCompatActivity() {

    var button: Button?=null
    var e1: EditText?=null
    var e2: EditText?=null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        button=findViewById(R.id.bt)
        e1=findViewById(R.id.editTextTextPersonName)
        e2=findViewById(R.id.editTextTextPassword)

        button!!.setOnClickListener {

            var s1: String= e1!!.text.toString()
            var s2: String=e2!!.text.toString()
            Toast.makeText(applicationContext,s1+s2,Toast.LENGTH_LONG).show()

        }
    }
}
```

COMPONENTS OF ANDROID

ACTIVITY LIFE CYCLE

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View). While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with windows Floating set) or embedded inside of another activity (using ActivityGroup). There are two methods almost all subclasses of Activity will implement:


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

onPause() is where you deal with the user leaving your activity. Most importantly, any changes made by the user should at this point be committed (usually to the ContentProvider holding the data).
To be of use with Context.startActivity(), all activity classes must have a corresponding <activity> declaration in their package's AndroidManifest.xml
Activities in the system are managed as an activity stack. When a new activity is started, it is placed on the top of the stack and becomes the running activity — the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits.
An activity has essentially four states:
If an activity is in the foreground of the screen (at the top of the stack), it is active or running.
If an activity has lost focus but is still visible (that is, a new non-full-sized or transparent activity has focus on top of your activity), it is paused. A paused activity is completely alive (it maintains all state and member

information and remains attached to the window manager), but can be killed by the system in extreme low memory situations.
If an activity is completely obscured by another activity, it is stopped. It still retains all state and member information, however, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere.
If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.
The following diagram shows the important state paths of an Activity. The square rectangles represent callback methods you can implement to perform operations when the Activity moves between states. The colored ovals are major states the Activity can be in.
There are three key loops you may be interested in monitoring within your activity:
The entire lifetime of an activity happens between the first call to onCreate(Bundle) through to a single final call to onDestroy(). An activity will do all setup of “global” state in onCreate(), and release all remaining resources in onDestroy(). For example, if it has a thread running in the background to download data from the network, it may create that thread in onCreate() and then stop the thread in onDestroy().
The visible lifetime of an activity happens between a call to onStart() until a corresponding call to onStop(). During this time the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods you can maintain resources that are needed to show the activity to the user. For example, you can register a BroadcastReceiver in onStart() to monitor for changes that impact your UI, and unregister it in onStop() when the user no longer sees what you are displaying. The onStart() and onStop() methods can be called multiple times, as the activity becomes visible and hidden to the user.
The foreground lifetime of an activity happens between a call to onResume() until a corresponding call to onPause(). During this time the activity is in front of all other activities and interacting with the user. An activity can frequently go between the resumed and paused states — for example when the device goes to sleep, when an activity result is delivered, when a new intent is delivered — so the code in these methods should be fairly lightweight.
The entire lifecycle of an activity is defined by the following Activity methods. All of these are hooks that you can override to do appropriate work when the activity changes state. All activities will implement

onCreate(Bundle) to do their initial setup; many will also implement onPause() to commit changes to data and otherwise prepare to stop interacting with the user. You should always call up to your superclass when implementing these methods.

```
public class Activity extends ApplicationContext {
        protected void onCreate(Bundle savedInstanceState);
        protected void onStart();
        protected void onRestart();
        protected void onResume();
        protected void onPause();
        protected void onStop();
        protected void onDestroy();
 }
```

Example Program

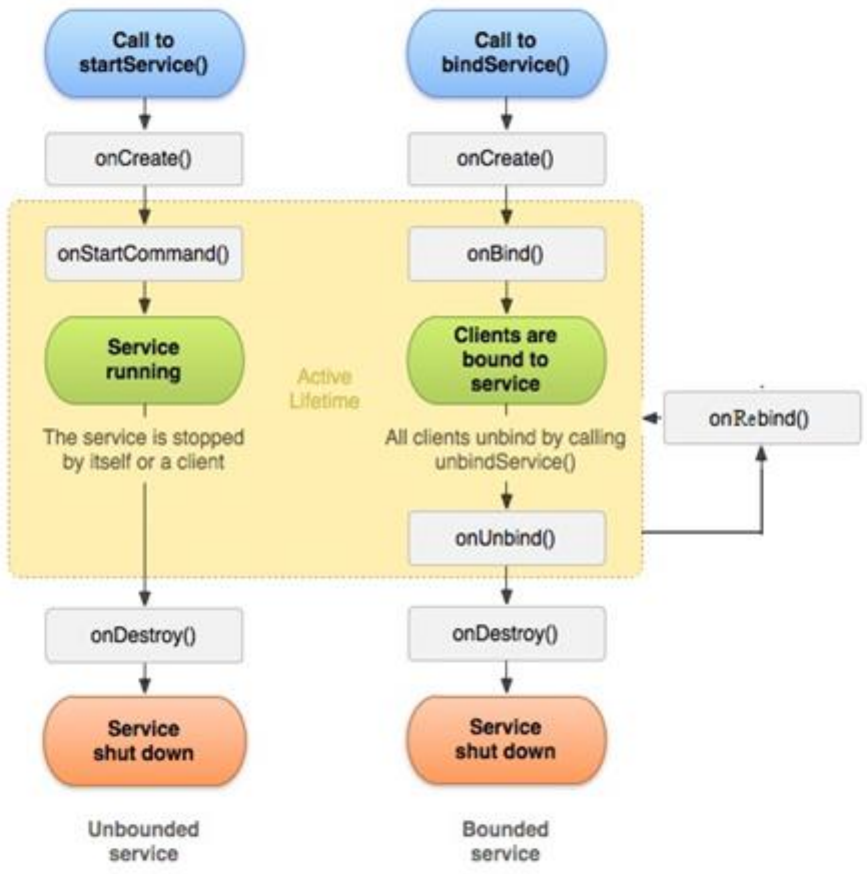
```
package androiindians.lifecycle;


import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Toast;


public class MainActivity extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast.makeText(MainActivity.this,
                "Oncreate",Toast.LENGTH_LONG).show();
        Log.i("result","Create");


    }
    protected void onStop(){
        super.onStop();
        Toast.makeText(MainActivity.this,
                "OnStop",Toast.LENGTH_LONG).show();
        Log.i("result","OnStop");
    }
    protected void onStart(){
        super.onStart();
        Toast.makeText(MainActivity.this,
                "onStart",Toast.LENGTH_LONG).show();
        Log.i("result","onStart");
    }
    protected void onRestart(){
        super.onRestart();
        Toast.makeText(MainActivity.this,
                "onRestart",Toast.LENGTH_LONG).show();
        Log.i("result","onRestart");
    }
    protected void onResume(){
        super.onResume();
        Toast.makeText(MainActivity.this,
                "OnResume",Toast.LENGTH_LONG).show();
        Log.i("result","OnResume");
    }
    protected void onPause(){
        super.onPause();
        Toast.makeText(MainActivity.this,
                "OnPause",Toast.LENGTH_LONG).show();
        Log.i("result","OnPause");
    }


}
```

SERVICES

A Service is an app component that can handle long-running operations in the background, and it does not provide a user interface. Another application component can start a service, and it continues to run in the background even if the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC).

For example, a service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

These are the three different types of services:



Scheduled

A service is scheduled when an API such as the JobScheduler, introduced in Android 5.0 (API level 21), launches the service. You can use the JobScheduler by registering jobs and specifying their requirements for network and timing. The system then gracefully schedules the jobs for execution at the appropriate times. The JobScheduler provides many methods to define service-execution conditions.

Note: If your app targets Android 5.0 (API level 21), Google recommends that you use the JobScheduler to execute background services. For more information about using this class, see the JobScheduler reference documentation.

Started

A service is started when an application component (such as an activity) calls startService(). After it's started, a service can run in the background indefinitely, even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller.

For example, it can download or upload a file over the network. When the operation is complete, the service should stop itself.

Bound

A service is bound when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Regardless of whether your application is started, bound, or both, any application component can use the service (even from a separate application) in the same way that any component can use an activity—by starting it with an Intent. However, you can declare the service as private in the manifest file and block access from other applications. This is discussed more in the section about Declaring the service in the manifest.

Caution: A service runs in the main thread of its hosting process; the service does not create its own thread and does not run in a separate process unless you specify otherwise. If your service is going to perform any CPU-intensive work or blocking operations, such as MP3 playback or networking, you should create a new thread within the service to complete that work. By using a separate thread, you can reduce the risk of Application Not Responding (ANR) errors, and the application's main thread can remain dedicated to user interaction with your activities.

Activity_mai.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:tools="http://schemas.android.com/tools"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Services Example"
         />


    <Button
        android:id="@+id/start"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start Servie" />


    <Button
        android:id="@+id/stop"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Stop Services" />


</LinearLayout>
</layout>
```

MainActivity.kt

```
package com.androindian.services


import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.databinding.DataBindingUtil
import com.androindian.services.databinding.ActivityMainBinding


class MainActivity : AppCompatActivity() {
    var bining: ActivityMainBinding?=null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        bining=DataBindingUtil.setContentView(this,R.layout.activity_main)
        bining?.start?.setOnClickListener {
            var intent= Intent(this,Myservices::class.java)
            startService(intent)


        }


        bining?.stop?.setOnClickListener {
            var intent= Intent(this,Myservices::class.java)
            stopService(intent)


        }
    }
}
```

Myservices.kt

```
package com.androindian.services


import android.app.Service
import android.content.Intent
import android.os.IBinder
import android.widget.Toast


class Myservices : Service() {


    override fun onBind(intent: Intent): IBinder {
        TODO("Return the communication channel to the service.")
        Toast.makeText(this,"onBind",Toast.LENGTH_LONG).show()
    }


    override fun onStart(intent: Intent?, startId: Int) {
        super.onStart(intent, startId)
        Toast.makeText(this,"onStart",Toast.LENGTH_LONG).show()
    }


    override fun onCreate() {
        super.onCreate()
        Toast.makeText(this,"onCreate",Toast.LENGTH_LONG).show()
    }


    override fun onDestroy() {
        super.onDestroy()
        Toast.makeText(this,"onDestroy",Toast.LENGTH_LONG).show()
    }


    override fun onRebind(intent: Intent?) {
        super.onRebind(intent)
        Toast.makeText(this,"onRebind",Toast.LENGTH_LONG).show()
    }
}
```

BROADCAST RECEIVERS

a BroadcastReceiver is a component that allows an app to listen for and respond to system-wide broadcast messages or events. These events could originate from the system itself, such as when the device boots up, when the battery level changes, when the network connectivity state changes, etc. Additionally, apps can send custom broadcasts to communicate within the app or between different apps.
Here are some key points about BroadcastReceivers in Android:

- Purpose:
  - BroadcastReceivers allow apps to receive and respond to broadcast messages sent by the system or other apps.
  - They provide a way for apps to be notified of important system events or changes and to perform actions in response.
- Types of Broadcasts:
  - There are two main types of broadcasts in Android: system broadcasts and custom broadcasts.
  - System broadcasts are sent by the system to notify apps of system events, such as the device booting up, the battery level changing, the network connectivity state changing, etc.
  - Custom broadcasts are broadcasts sent by apps to communicate within the app or between different apps.
- Registration:
  - BroadcastReceivers can be registered either statically in the AndroidManifest.xml file or dynamically at runtime using code.
  - Static registration in the manifest is used for system-wide broadcasts or broadcasts that need to be received even when the app is not running.

    - Dynamic registration using code is used for broadcasts that are relevant only when the app is running.
- Intent Filters:
    - BroadcastReceivers use intent filters to specify the types of broadcasts they are interested in receiving.
    - Intent filters define the action, category, and data URI of the broadcasts that the BroadcastReceiver should respond to.
    - A BroadcastReceiver can have multiple intent filters to listen for different types of broadcasts.
- Handling Broadcasts:
    - When a broadcast that matches the BroadcastReceiver's intent filter is received, the onReceive() method of the BroadcastReceiver is called.
    - The onReceive() method is where you implement the logic to handle the broadcast and perform any necessary actions or operations.

Activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
    <EditText android:id="@+id/extraIntent"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/sendMessage" />
    <Button
        android:id="@+id/btnStartBroadcast"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/extraIntent"
        android:onClick="broadcastCustomIntent"
        android:text="@string/myBroadcastIntent" />
</RelativeLayout>
```

MainActivity.Java

```
package androidindians.broadcastreciver


import android.content.Intent
import android.os.Bundle
import android.support.v7.app.AppCompatActivity
import android.view.View
import android.widget.EditText
```

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }


    fun broadcastCustomIntent(view: View) {
        val intent = Intent("MyCustomIntent")
        val et = findViewById<EditText>(R.id.extraIntent)
        // add data to the Intent
        intent.putExtra("message", et.text.toString())
        intent.action = "androidindians.broadcastreciver.A_CUSTOM_INTENT"
        sendBroadcast(intent)
    }
}
```

MyBroadcastReceiver.java

```
package androidindians.broadcastreciver


import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.widget.Toast


class MyBroadcastReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        // Extract data included in the Intent
        val intentData: CharSequence? = intent.getCharSequenceExtra("message")
        Toast.makeText(context, "Raj received the Intent's message: $intentData",
Toast.LENGTH_LONG).show()
    }
}


}
```

Manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="androiindians.broadcastreciver">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name="MyBroadcastReceiver">
            <intent-filter>
                <action android:name="androiindians.broadcastreciver.A_CUSTOM_INTENT">
                </action>
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

Strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">BroadcastReceiversTest</string>
    <string name="action_settings">Settings</string>
    <string name="sendMessage">Write a message to broadcast!</string>
    <string name="myBroadcastIntent">Broadcast an Intent now...</string>
</resources>
```

CONTENT PROVIDER

A Content Provider in Android is a component that manages access to a central repository of data. It acts as an intermediary between different applications, allowing them to share and access data in a consistent and secure manner. Content Providers are commonly used to manage structured data stored in databases, files, or other data sources.
Here's an explanation of Content Providers along with an example code in Kotlin:

- Purpose:
  - Content Providers facilitate data sharing between different applications in Android.
  - They provide a standardized interface for querying, inserting, updating, and deleting data.
  - Content Providers ensure data integrity and security by enforcing access permissions and data encapsulation.
- Components:
  - URI (Uniform Resource Identifier): Content Providers use URIs to identify data resources. URIs typically have the

format content://authority/path, where "authority" identifies the Content Provider and "path" specifies the data resource.

- Contract: Content Providers define a contract that specifies the data schema, column names, MIME types, and URIs that can be accessed by other applications.
- Data Operations: Content Providers support CRUD (Create, Read, Update, Delete) operations for manipulating data. Other applications can perform these operations using ContentResolver objects.

Example Code:

- Let's create a simple Content Provider in Kotlin that manages a list of books. We'll define a contract class that specifies the data schema, a Content Provider class that implements the CRUD operations, and use a SQLite database to store the data.

```
// BookContract.kt
object BookContract {
    const val CONTENT_AUTHORITY = "com.example.bookprovider"
    const val PATH_BOOKS = "books"
    val CONTENT_URI: Uri = Uri.parse("content://$CONTENT_AUTHORITY/$PATH_BOOKS")

    object BookEntry {
        const val TABLE_NAME = "books"
        const val COLUMN_ID = "_id"
        const val COLUMN_TITLE = "title"
        const val COLUMN_AUTHOR = "author"
    }
}



// BookProvider.kt
class BookProvider : ContentProvider() {
    private lateinit var dbHelper: BookDbHelper



    override fun onCreate(): Boolean {
        dbHelper = BookDbHelper(context!!)
        return true
    }



    override fun insert(uri: Uri, values: ContentValues?): Uri? {
        val db = dbHelper.writableDatabase
        val id = db.insert(BookContract.BookEntry.TABLE_NAME, null, values)
        return if (id > 0) {
            val newUri = ContentUris.withAppendedId(BookContract.CONTENT_URI, id)
            context?.contentResolver?.notifyChange(newUri, null)
            newUri
        } else {
            throw SQLException("Failed to insert row into $uri")
        }
    }
    // Implement other ContentProvider methods (query, update, delete) similarly


    override fun getType(uri: Uri): String? {
        return
"vnd.android.cursor.item/$BookContract.CONTENT_AUTHORITY.$BookContract.BookEntry.TABLE_
NAME"
    }
}


// BookDbHelper.kt
class BookDbHelper(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null,
DATABASE_VERSION) {
    override fun onCreate(db: SQLiteDatabase) {
        val SQL_CREATE_BOOKS_TABLE = "CREATE TABLE
${BookContract.BookEntry.TABLE_NAME} (" +
                "${BookContract.BookEntry.COLUMN_ID} INTEGER PRIMARY KEY AUTOINCREMENT, "
+
                "${BookContract.BookEntry.COLUMN_TITLE} TEXT NOT NULL, " +
                "${BookContract.BookEntry.COLUMN_AUTHOR} TEXT NOT NULL);"
        db.execSQL(SQL_CREATE_BOOKS_TABLE)
    }


    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        // Upgrade database schema if needed
    }


    companion object {
        private const val DATABASE_NAME = "books.db"
        private const val DATABASE_VERSION = 1
    }
}
```

In this example:

- BookContract defines the contract for the Content Provider, including the content URI and column names.
- BookProvider implements the Content Provider class, providing methods for CRUD operations.
- BookDbHelper is a helper class for managing the SQLite database used by the Content Provider.

INTENTS

an Intent is a messaging object that facilitates communication between components of an application, as well as between different applications. Intents are used for various purposes, including starting activities, services, and broadcasts, as well as passing data between components. Here's an explanation of Intents in Android:

- Purpose:
  - Intents are used to initiate various actions within an Android application or between different applications.
  - They provide a flexible and loosely coupled way for components to communicate with each other.
  - Intents enable developers to create dynamic and interactive user experiences by launching activities, starting services, broadcasting messages, and passing data.
- Types:
  - Explicit Intents: Explicit Intents are used to start a specific component within the same application by specifying the target component's class name or package name.
  - Implicit Intents: Implicit Intents are used to trigger an action based on the intent's action string, such as opening a web page, sending an email, making a phone call, or sharing content with other apps. Implicit Intents allow the system to find the best matching component to handle the requested action.
- Components:
  - Activities: Intents are commonly used to start activities, which represent individual screens in an Android application. By using Intents, developers can navigate between different activities within the same app or launch activities from other apps.
  - Services: Intents can also be used to start background services, which perform long-running operations in the background, such as downloading files, playing music, or processing data.

    - Broadcast Receivers: Intents are used to broadcast messages to interested components using the publish-subscribe pattern. Broadcast Receivers can listen for specific broadcast messages and respond accordingly.
- Data Passing:
    - Intents can carry additional data in the form of key-value pairs called Extras. Extras can be primitive data types, strings, arrays, or custom Parcelable or Serializable objects.
    - Data can be passed between components using Intent extras, allowing components to share information and coordinate their behavior.

Explicit Intent Example

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity"
   android:orientation="vertical">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="We are in First Page"
       />

   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/name"/>
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/mobile"/>
```

```
  <Button
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="Move to Second"
     android:id="@+id/bt1"
     />

</LinearLayout>
</layout>


MainActivity.kt
package com.example.exintent

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.databinding.DataBindingUtil
import com.example.exintent.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
  var binding: ActivityMainBinding?=null
  override fun onCreate(savedInstanceState: Bundle?) {
     super.onCreate(savedInstanceState)
     binding=
DataBindingUtil.setContentView(this,R.layout.activity_main)


     binding?.bt1?.setOnClickListener{
        var intent= Intent(this@MainActivity,Second::class.java)
       // intent.putExtra("key","values")
        intent.putExtra("name",binding?.name?.text.toString())
        intent.putExtra("mobile",binding?.mobile?.text.toString())

        startActivity(intent)
     }
  }
}
Activity_second.kt


<?xml version="1.0" encoding="utf-8"?>
<layout>
```

```
  <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      tools:context=".Second"
      android:orientation="vertical">

      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="We are in Second Page"
         />
      <Button
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:text="Move to Third"
         android:id="@+id/bt2"
         />

  </LinearLayout>
</layout>
```

SecondActivity.kt

```
package com.example.exintent

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.Toast
import androidx.databinding.DataBindingUtil
import com.example.exintent.databinding.ActivitySecondBinding

class Second : AppCompatActivity() {
  var binding: ActivitySecondBinding?=null
  override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
```

```
        binding=
DataBindingUtil.setContentView(this,R.layout.activity_second)

        var dataintent=intent
        var s1=dataintent.getStringExtra("name")
        var s2=dataintent.getStringExtra("mobile")
        Toast.makeText(this@Second,""+s1+s2,Toast.LENGTH_LONG).sh
ow()


        binding?.bt2?.setOnClickListener {
            var intent= Intent(this@Second,Third::class.java)
            intent.putExtra("data",s1+s2)
            startActivity(intent)
          //  finish()
        }
    }
}
```

Activity_third.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".Third">

</androidx.constraintlayout.widget.ConstraintLayout>
```

ThirdActivity.kt

```
package com.example.exintent

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast

class Third : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_third)

        var dataintent=intent
        var s1=dataintent.getStringExtra("data")

        Toast.makeText(this@Third,""+s1, Toast.LENGTH_LONG).show()

    }

    override fun onBackPressed() {
        super.onBackPressed()
        var intent= Intent(this@Third,MainActivity::class.java)
        startActivity(intent)
    }
}
```

Androidmanifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Exintent"
        tools:targetApi="31">
        <activity
            android:name=".Third"
            android:exported="false" />
        <activity
            android:name=".Second"
            android:exported="false" />
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Implicit Intent example

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
    <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity"
        android:orientation="vertical">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Implicit Intent  eaxmple!"
            />

        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Camera "
            android:id="@+id/camera"/>
        <Button
            android:layout_width="match_parent"
```

```
          android:layout_height="wrap_content"
          android:text="Browser "
          android:id="@+id/browser"/>
      <Button
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="Gallary "
          android:id="@+id/gallary"/>

  </LinearLayout>
</layout>
```

MainActivity.kt

```
package com.example.implicit


import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.provider.MediaStore
import androidx.databinding.DataBindingUtil
import com.example.implicit.databinding.ActivityMainBinding
import android.content.Intent
import android.net.Uri

class MainActivity : AppCompatActivity() {

  var binding: ActivityMainBinding?=null
  override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      binding=
DataBindingUtil.setContentView(this,R.layout.activity_main)

      binding?.camera?.setOnClickListener {

          var intent= Intent(MediaStore.ACTION_IMAGE_CAPTURE)
          startActivity(intent)
      }
      binding?.browser?.setOnClickListener {

          var intent= Intent(android.content.Intent.ACTION_VIEW)
```

```
            intent.data=Uri.parse("https://androindian.com/registration-using-
retrofit-raw-data/")
            startActivity(intent)
        }

        binding?.gallary?.setOnClickListener {

            var intent= Intent(Intent.ACTION_PICK)
           // intent.setType("image/*")
            intent.setType("video/*")
            //  intent.setType("doc/*")

            startActivity(intent)
        }
    }

}
```

Advanced User Interface

In Android development, an Adapter is a bridge between a UI component (such as a ListView, RecyclerView, Spinner, etc.) and the underlying data source. Adapters are used to populate UI components with data, providing a way to dynamically display and manage large datasets efficiently.
Here's an explanation of Adapters in Android:

- **Purpose:**
    - **Adapters are used to bind data to UI components, allowing them to display information from a data source.**
    - **They abstract away the complexity of managing data presentation, handling data updates, and recycling views to optimize memory usage and performance.**
- **Types of Adapters:**
    - **ArrayAdapter: ArrayAdapter is a simple adapter that works with arrays or Lists of data. It converts each**

ITEM IN THE DATA SOURCE INTO A VIEW THAT CAN BE DISPLAYED WITHIN A UI COMPONENT.

- CursorAdapter: CursorAdapter is used with data from a database query represented as a Cursor object. It allows efficient retrieval and binding of data from the database to UI components.
- BaseAdapter: BaseAdapter is a generic adapter class that provides a foundation for custom adapters. It requires implementing certain methods to manage data binding and view recycling.
- RecyclerView.Adapter: RecyclerView.Adapter is used with RecyclerView, a more flexible and efficient replacement for ListView. It provides similar functionality to BaseAdapter but with additional features like support for different view types and animations.

- Components:
  - getView() (or onCreateViewHolder() and onBindViewHolder() in RecyclerView.Adapter): This method is responsible for creating and populating views for individual items in the data source. It inflates a layout file to create a view, binds data to the view, and returns it to be displayed in the UI component.
  - getItem() (or similar methods): These methods retrieve individual items from the data source based on their position.
  - getItemCount(): This method returns the total number of items in the data source.
  - notifyDataSetChanged(): This method notifies the adapter that the underlying dataset has changed, prompting it to refresh the UI components accordingly.
- Usage:
  - Adapters are typically used in conjunction with UI components like ListView, RecyclerView, Spinner, etc., to display data in a structured format.

- THEY ENABLE DEVELOPERS TO DECOUPLE DATA MANAGEMENT FROM UI RENDERING, MAKING IT EASIER TO UPDATE AND MAINTAIN THE UI DYNAMICALLY.

LISTVIEW EXAMPLE

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity"
   android:orientation="vertical">

<ListView
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:id="@+id/list"/>


</LinearLayout>
</layout>
```

MainActivity.kt

```
package com.example.list

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.ArrayAdapter
import android.widget.Toast
import androidx.databinding.DataBindingUtil
import com.example.list.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
 var binding: ActivityMainBinding?=null
var Course=arrayOf("Android","Java","CoreJava","Adv Java",
"PHP","Python", "Core Java","Adv Java","Android","Java",
  "CoreJava","Adv Java", "PHP","Python", "Core Java","Adv Java",
  "Android","Java","CoreJava","Adv Java", "PHP","Python", "Core
Java","Adv Java")
  override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      binding=
DataBindingUtil.setContentView(this,R.layout.activity_main)

      // predefined layout & data from kotlin
     // var courseadapter=
ArrayAdapter(this@MainActivity,android.R.layout.simple_list_item_1,
Course)
      //binding?.list?.adapter=courseadapter

      // predefined layout & data from Res
      //var xyz=resources.getStringArray(R.array.Cities)
     var courseadapter=
ArrayAdapter(this@MainActivity,android.R.layout.simple_list_item_1,
resources.getStringArray(R.array.xyz))
      binding?.list?.adapter=courseadapter

      // custom layout & data from kotlin

      //var courseadapter=
ArrayAdapter(this@MainActivity,R.layout.custom, Course)
      //binding?.list?.adapter=courseadapter

      binding?.list?.setOnItemClickListener { parent, view, position, id ->
         Toast.makeText(this,""+position,Toast.LENGTH_LONG).show()
      }
  }
}
```

Custom.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:textStyle="bold"
  android:textAllCaps="true"
  android:textColor="#FF5722"
  android:textSize="20sp"
  android:layout_margin="10dp"
  android:padding="10dp">

</TextView>
```

String.xml

```
<resources>
  <string name="app_name">List</string>

  <string-array name="xyz">
     <item>abc</item>
     <item>abc</item>
     <item>abc</item>
     <item>abc</item>
     <item>abc</item>
     <item>abc</item>
     <item>abc</item>
     <item>abc</item>
     <item>abc</item>
     <item>abc</item>
     <item>abc</item>

  </string-array>

</resources>
```

GRIDVIEW EXAMPLE

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity">
  <GridView
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:id="@+id/grid"
     android:layout_margin="10dp"
     android:numColumns="auto_fit"/>



</RelativeLayout>
</layout>
```

MainActivity.kt

```
package com.example.grid

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.AdapterView
import android.widget.AdapterView.OnItemSelectedListener
import android.widget.ArrayAdapter
import android.widget.Toast
import androidx.databinding.DataBindingUtil
import com.example.grid.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
  var binding: ActivityMainBinding?=null
  var course=arrayOf("Android", "Java","CoreJava","Core Java","
advjava","adv Java",
     "python","androidjava","javaandroid")
  override fun onCreate(savedInstanceState: Bundle?) {
     super.onCreate(savedInstanceState)
     binding=
DataBindingUtil.setContentView(this,R.layout.activity_main)

     var gridadaprer= ArrayAdapter(this,
android.R.layout.simple_list_item_1, course)
     binding?.grid?.adapter=gridadaprer

     binding?.grid?.setOnItemClickListener { parent, view, position, id ->
```

```
        Toast.makeText(this,""+id,Toast.LENGTH_LONG).show()
      }


        }
}
```

SPINNER EXAMPLE

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">


   <Spinner
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:id="@+id/spin"
       android:layout_below="@id/grid"
       android:layout_margin="10dp"/>


</RelativeLayout>
</layout>
```

MainActivity.kt

```
package com.example.grid

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.AdapterView
import android.widget.AdapterView.OnItemSelectedListener
import android.widget.ArrayAdapter
import android.widget.Toast
import androidx.databinding.DataBindingUtil
import com.example.grid.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
   var binding: ActivityMainBinding?=null
   var course=arrayOf("Android", "Java","CoreJava","Core Java","
advjava","adv Java",
       "python","androidjava","javaandroid")
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       binding=
DataBindingUtil.setContentView(this,R.layout.activity_main)




       var spinadp= ArrayAdapter(this,
android.R.layout.simple_list_item_1, course)
       binding?.spin?.adapter=spinadp

       binding?.spin?.setOnItemSelectedListener(object:
OnItemSelectedListener{
           override fun onItemSelected(
               parent: AdapterView<*>?,
               view: View?,
               position: Int,
               id: Long
           ) {
               Toast.makeText(this@MainActivity,""+id,Toast.LENGTH_LON
G).show()
           }

           override fun onNothingSelected(parent: AdapterView<*>?) {

           }
       })

     }
}
```

AUTOCOMPLETETEXTVIEW EXAMPLE

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">



   <AutoCompleteTextView
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:id="@+id/auto"
       android:layout_below="@id/spin"
       android:completionThreshold="1"
       android:layout_margin="10dp"/>

</RelativeLayout>
</layout>
```

MainActivity.kt

```
package com.example.grid

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.AdapterView
import android.widget.AdapterView.OnItemSelectedListener
import android.widget.ArrayAdapter
import android.widget.Toast
import androidx.databinding.DataBindingUtil
import com.example.grid.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
   var binding: ActivityMainBinding?=null
   var course=arrayOf("Android", "Java","CoreJava","Core Java","
advjava","adv Java",
       "python","androidjava","javaandroid")
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       binding=
DataBindingUtil.setContentView(this,R.layout.activity_main)



       var autoadapter= ArrayAdapter(this,
android.R.layout.simple_list_item_1, course)
       binding?.auto?.setAdapter(autoadapter)
   }
}
```

RecyclerView

RecyclerView is a powerful and flexible UI component introduced in Android's support library to efficiently display large datasets in a scrollable list or grid format. It's an improved version of the older ListView and GridView components, providing better performance, more flexibility, and easier customization.
Here's an explanation of RecyclerView in Android:

- Purpose:
    - RecyclerView is designed to efficiently display large datasets in a scrollable list or grid format.
    - It efficiently manages the memory usage by recycling views as they move off-screen, reducing the need for creating and destroying views dynamically.
    - RecyclerView provides a flexible architecture for implementing various layout managers, item decorations, and item animations.
- Components:

  - LayoutManager: RecyclerView uses layout managers to arrange items in a list, grid, or custom layout. Android provides several built-in layout managers like LinearLayoutManager (for vertical or horizontal lists), GridLayoutManager (for grids), and StaggeredGridLayoutManager (for staggered grids).
  - Adapter: RecyclerView.Adapter is responsible for providing data to the RecyclerView and creating views to represent individual items in the dataset. It works similarly to adapters in ListView and GridView but with additional methods for managing view types and item animations.
  - ViewHolder: RecyclerView.ViewHolder represents individual views (or view holders) for items in the dataset. It holds references to the views within each item layout, allowing efficient recycling of views as they scroll off-screen.
- Features:
  - View Recycling: RecyclerView recycles views as they scroll off-screen and reuses them to display new items, reducing memory usage and improving performance.
  - Item Animations: RecyclerView supports built-in item animations for adding, removing, and changing items in the dataset, providing a smoother user experience.
  - Item Decorations: RecyclerView allows you to add custom decorations (such as dividers, margins, or decorators) to individual items or the entire list/grid.
  - Item Touch Helpers: RecyclerView provides built-in support for handling touch events, gestures, and drag-and-drop interactions on individual items.
- Usage:
  - To use RecyclerView in your app, you need to add the RecyclerView dependency to your project's build.gradle file.
  - Next, you create a layout file for the item view that represents individual items in the dataset.
  - Then, you define a custom adapter by extending RecyclerView.Adapter and implement methods to manage data binding and view creation.
  - Finally, you set up the RecyclerView in your activity or fragment, configure it with a layout manager and adapter, and provide the dataset to be displayed.

Build.gradle

```
plugins {
   id("com.android.application")
   id("org.jetbrains.kotlin.android")
}

android {
   namespace = "com.example.recycler"
   compileSdk = 34

   defaultConfig {
       applicationId = "com.example.recycler"
       minSdk = 24
       targetSdk = 34
       versionCode = 1
       versionName = "1.0"

       testInstrumentationRunner =
"androidx.test.runner.AndroidJUnitRunner"
   }

   buildTypes {
       release {
           isMinifyEnabled = false
           proguardFiles(
               getDefaultProguardFile("proguard-android-optimize.txt"),
               "proguard-rules.pro"
           )
       }
   }
   compileOptions {
       sourceCompatibility = JavaVersion.VERSION_1_8
       targetCompatibility = JavaVersion.VERSION_1_8
   }
   kotlinOptions {
       jvmTarget = "1.8"
   }
   dataBinding{
       enable=true
   }
}
```

```
dependencies {

   implementation("androidx.core:core-ktx:1.12.0")
   implementation("androidx.appcompat:appcompat:1.6.1")
   implementation("com.google.android.material:material:1.11.0")
   implementation("androidx.constraintlayout:constraintlayout:2.1.4")
   testImplementation("junit:junit:4.13.2")
   androidTestImplementation("androidx.test.ext:junit:1.1.5")
   androidTestImplementation("androidx.test.espresso:espresso-
core:3.5.1")
}
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
   <RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
       xmlns:app="http://schemas.android.com/apk/res-auto"
       xmlns:tools="http://schemas.android.com/tools"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       tools:context=".MainActivity">

       <TextView
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="Recycler view example"
           android:id="@+id/tv1"
           />
       <androidx.recyclerview.widget.RecyclerView
           android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:id="@+id/rec"
           android:layout_below="@id/tv1"
           android:layout_margin="10dp"/>

   </RelativeLayout>
</layout>
```

MainActivity.kt

```
package com.example.recycler
```

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.databinding.DataBindingUtil
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.recycler.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
   var binding: ActivityMainBinding?=null
   var Names=arrayOf("AAA","BBB","CCC")
   var Mobile=arrayOf("5253","87567","7565")
   var
Emails=arrayOf("aaa@gmail.com","bbb@yamil.com","ccc@hotmail.com
")
   var
Profile=arrayOf(R.mipmap.ic_launcher,R.mipmap.ic_launcher_round,
R.mipmap.ic_launcher)
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       binding=
DataBindingUtil.setContentView(this,R.layout.activity_main)

       var linearlayoutmanager= LinearLayoutManager(this@MainActivity,
          LinearLayoutManager.VERTICAL, false)
       binding?.rec?.layoutManager=linearlayoutmanager

       var customAdapter=CustomAdapter(this@MainActivity, Names,
Mobile, Emails, Profile)
       binding?.rec?.adapter=customAdapter
   }
}
```

CustomAdapter.kt

```
package com.example.recycler

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageView
import android.widget.LinearLayout
import android.widget.TextView
import android.widget.Toast
import androidx.recyclerview.widget.RecyclerView

class CustomAdapter(mainActivity: MainActivity, names:
Array<String>, mobile: Array<String>, emails: Array<String>,
                 profile: Array<Int>) :
RecyclerView.Adapter<CustomAdapter.MyViewHolder>() {

         var adpnames: Array<String>
   var adpmobile: Array<String>
   var adpemail: Array<String>
   var adpprofile: Array<Int>
   var adpContext: Context

   init {
       adpnames=names
       adpemail=emails
       adpmobile=mobile
       adpprofile=profile
       adpContext=mainActivity
   }



   override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
CustomAdapter.MyViewHolder {
       var
view=LayoutInflater.from(parent.context).inflate(R.layout.custom,parent,f
alse)
       return MyViewHolder(view)
   }

   override fun onBindViewHolder(holder: CustomAdapter.MyViewHolder,
position: Int) {

       holder.name?.text=adpnames[holder.adapterPosition]
       holder.email?.text=adpemail[holder.adapterPosition]
       holder.mobile?.text=adpmobile[holder.adapterPosition]
       holder.profile?.setImageResource(adpprofile[holder.adapterPosition]
)
```

```
        holder?.profile?.setOnClickListener{
            Toast.makeText(adpContext,"image"+position,Toast.LENGTH_L
ONG).show()
        }

        holder?.linearLayout?.setOnClickListener{
            Toast.makeText(adpContext,"adp"+position,Toast.LENGTH_LON
G).show()
        }
    }

    override fun getItemCount(): Int {
        return adpnames.size
    }

    inner class MyViewHolder (itemview:
View):RecyclerView.ViewHolder(itemview){
        var name:TextView?=null
        var mobile:TextView?=null
        var email:TextView?=null
        var profile:ImageView?=null
        var linearLayout:LinearLayout?=null

        init {
            name=itemview.findViewById(R.id.name)
            mobile=itemview.findViewById(R.id.mobile)
            email=itemview.findViewById(R.id.email)
            profile=itemview.findViewById(R.id.iv)
            linearLayout=itemview.findViewById(R.id.linear)

        }


    }

}
```

Custom.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/iv"
        android:layout_margin="5dp"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/linear"
        android:orientation="vertical">
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/name"
            android:layout_margin="5dp"/>
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/mobile"
            android:layout_margin="5dp"/>
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/email"
            android:layout_margin="5dp"/>
    </LinearLayout>

</LinearLayout>
```

TELEPHONY MANAGER

IMEI

In Android, accessing the IMEI (International Mobile Equipment Identity) number directly is restricted due to privacy and security concerns. However, you can retrieve the IMEI number using the TelephonyManager class, but you need to request the READ_PHONE_STATE permission in your app's manifest file to access it.
Here's how you can retrieve the IMEI number using Kotlin:

- Add the necessary permission to your AndroidManifest.xml:
-

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

Use the TelephonyManager class to retrieve the IMEI number

```
import android.Manifest
import android.content.Context
import android.content.pm.PackageManager
import android.os.Build
import android.os.Bundle
import android.telephony.TelephonyManager
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat

class MainActivity : AppCompatActivity() {

    private val REQUEST_READ_PHONE_STATE = 101

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            if (ContextCompat.checkSelfPermission(
                    this,
                    Manifest.permission.READ_PHONE_STATE
                ) != PackageManager.PERMISSION_GRANTED
            ) {
                ActivityCompat.requestPermissions(
                    this,
                    arrayOf(Manifest.permission.READ_PHONE_STATE),
                    REQUEST_READ_PHONE_STATE
                )
            } else {
                getIMEINumber()
            }
        } else {
            getIMEINumber()
        }
    }

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults)

        if (requestCode == REQUEST_READ_PHONE_STATE) {
            if (grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                getIMEINumber()
            } else {
                Toast.makeText(
                    this,
                    "Permission denied. IMEI cannot be retrieved.",
                    Toast.LENGTH_SHORT
                ).show()
            }
        }
    }

    private fun getIMEINumber() {
        val telephonyManager =
            getSystemService(Context.TELEPHONY_SERVICE) as
TelephonyManager
        val imei = telephonyManager.imei

        if (imei != null) {
            Toast.makeText(this, "IMEI number: $imei",
Toast.LENGTH_LONG).show()
```

```
        } else {
            Toast.makeText(this, "IMEI number not available",
Toast.LENGTH_LONG).show()
        }
    }
}
```

PHONE STATUS

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
  >

   <androidx.cardview.widget.CardView
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_margin="16dp">

       <LinearLayout
           android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:layout_margin="16dp"
           android:orientation="vertical">
   <Button
       android:id="@+id/button1"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Ringer"
       android:layout_margin="16dp"/>

   <Button
       android:id="@+id/button2"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Vibrate "
        android:layout_margin="16dp"/>

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Silent "

        android:layout_margin="16dp"/>

    </androidx.cardview.widget.CardView>
</LinearLayout>
</LinearLayout>
</layout>
```

MainActivity.kt

```
package com.example.phonestatus

import android.content.Context
import android.media.AudioManager
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.databinding.DataBindingUtil
import com.example.phonestatus.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
    var binding: ActivityMainBinding?=null
    var audioManager: AudioManager?=null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding=
DataBindingUtil.setContentView(this,R.layout.activity_main)


        audioManager=getSystemService(Context.AUDIO_SERVICE) as
AudioManager?
        binding?.button1?.setOnClickListener{
            audioManager?.ringerMode=AudioManager.RINGER_MODE_NO
RMAL
        }
        binding?.button1?.setOnClickListener{
            audioManager?.ringerMode=AudioManager.RINGER_MODE_VI
BRATE
        }
        binding?.button3?.setOnClickListener{
            audioManager?.ringerMode=AudioManager.RINGER_MODE_SIL
ENT
        }
    }
}
```

Bluetooth

Add Bluetooth Permissions:

- First, you need to add the necessary permissions to your AndroidManifest.xml file.

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

Check Bluetooth Availability:

- Before using Bluetooth, it's a good idea to check if it's available on the device.

```
private fun isBluetoothAvailable(): Boolean {
    val bluetoothAdapter = BluetoothAdapter.getDefaultAdapter()
    return bluetoothAdapter != null
}
```

Enable Bluetooth:

- You may need to enable Bluetooth if it's not already enabled.

```
private fun enableBluetooth() {
    val bluetoothAdapter = BluetoothAdapter.getDefaultAdapter()
    if (!bluetoothAdapter.isEnabled) {
        val enableBtIntent = Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE)
        startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT)
    }
}
```

Listen for Bluetooth Events:

- You can register a BroadcastReceiver to listen for Bluetooth-related events.

```
private val bluetoothReceiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        when (intent?.action) {
            BluetoothAdapter.ACTION_STATE_CHANGED -> {
                val state = intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,
BluetoothAdapter.ERROR)
                when (state) {
                    BluetoothAdapter.STATE_ON -> {
                        // Bluetooth is enabled
                    }
                    BluetoothAdapter.STATE_OFF -> {
                        // Bluetooth is disabled
                    }
                }
            }
            BluetoothDevice.ACTION_FOUND -> {
                // A new Bluetooth device is discovered
                val device: BluetoothDevice? =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE)
                device?.let {
                    // Do something with the discovered device
                }
            }
        }
    }
}
```

Register and Unregister BroadcastReceiver:

- Don't forget to register and unregister the BroadcastReceiver.

```
override fun onResume() {
    super.onResume()
    val filter = IntentFilter().apply {
        addAction(BluetoothAdapter.ACTION_STATE_CHANGED)
        addAction(BluetoothDevice.ACTION_FOUND)
    }
    registerReceiver(bluetoothReceiver, filter)
}


override fun onPause() {
    super.onPause()
    unregisterReceiver(bluetoothReceiver)
}
```

DIALOGS

dialogs are small UI components that overlay the current activity to prompt the user for information or to confirm an action. There are several types of dialogs available in Android, such as AlertDialog, CustomDialog, DatePickerDialog, TimePickerDialog, etc.

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:tools="http://schemas.android.com/tools"
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto">
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity"
  android:orientation="vertical">

  <TextView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Dialogs Example"
     />
  <Button
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="Predefined"
     android:id="@+id/preferined"/>
  <Button
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="custom"
     android:id="@+id/custom"/>
  <Button
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="progress circle"
     android:id="@+id/progress1"/>
  <Button
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="Progress horizontal"
```

```
        android:id="@+id/progress2"/>

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"

        android:id="@+id/showdate"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Pick date "
        android:id="@+id/pickdate"/>

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"

        android:id="@+id/showtime"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Pick date "
        android:id="@+id/picktime"/>

    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
</layout>
```

MainActivity.kt

```
package com.example.dialogs

import android.app.AlertDialog
import android.app.DatePickerDialog
import android.app.Dialog
import android.app.ProgressDialog
import android.app.TimePickerDialog
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.TimePicker
import android.widget.Toast
import androidx.databinding.DataBindingUtil
import com.example.dialogs.databinding.ActivityMainBinding
import java.util.Calendar

class MainActivity : AppCompatActivity() {
   var binding: ActivityMainBinding?=null
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       binding=
DataBindingUtil.setContentView(this,R.layout.activity_main)

       binding?.preferined?.setOnClickListener{
           var predefined= AlertDialog.Builder(this)
           predefined.setTitle("Are you Sure")
           predefined.setMessage("Want to Close")

           predefined.setPositiveButton("Yes"){ dialogInterface,which->
               Toast.makeText(this@MainActivity,"Yes",Toast.LENGTH_LON
G).show()
           }
           predefined.setNegativeButton("NO"){ dialogInterface,which->
               Toast.makeText(this@MainActivity,"No",Toast.LENGTH_LON
G).show()
           }
           predefined.setNeutralButton("Cancel"){ dialogInterface,which->
               Toast.makeText(this@MainActivity,"Cancel",Toast.LENGTH_L
ONG).show()
           }
           predefined.show()
       }


       binding?.custom?.setOnClickListener{
           var dialog= Dialog(this)
           dialog.setCancelable(false)
           dialog.setContentView(R.layout.custom)
           var bt: Button?=null
```

```
            bt=dialog?.findViewById(R.id.button)

            bt?.setOnClickListener{
               dialog.dismiss()
               Toast.makeText(this@MainActivity,"Button",Toast.LENGTH_L
ONG).show()
            }
            dialog.show()
         }

         binding?.progress1?.setOnClickListener {
            var progressDialog=ProgressDialog(this@MainActivity)
            progressDialog.setTitle("Please wait")
            progressDialog.setMessage("Data laoding")
            progressDialog.show()
         }

         binding?.progress2?.setOnClickListener {
            var progressDialog=ProgressDialog(this@MainActivity)
            progressDialog.setTitle("Please wait")
            progressDialog.setMessage("Data laoding")
            progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZ
ONTAL)
            progressDialog.show()
         }

         binding?.pickdate?.setOnClickListener{

            var caldender= Calendar.getInstance()
            var myear=caldender.get(Calendar.YEAR)
            var mmonth=caldender.get(Calendar.MONTH)
            var mday=caldender.get(Calendar.DAY_OF_MONTH)

            var datePickerDialog= DatePickerDialog(this@MainActivity,
               DatePickerDialog.OnDateSetListener{view
,year,monthOfYear,dayOfMonth->
                  binding?.showdate?.setText(""+ dayOfMonth + "-" +
monthOfYear+ "-" + year)
               },myear,mmonth,mday)

            datePickerDialog.getDatePicker().setMaxDate(System.currentTim
eMillis());
```

```
          //
datePickerDialog.getDatePicker().setMinDate(System.currentTimeMillis(
));

           datePickerDialog.show()
       }

       binding?.picktime?.setOnClickListener{

           var caldender=Calendar.getInstance()
           var mhour=caldender.get(Calendar.HOUR)
           var mmin=caldender.get(Calendar.MINUTE)
           var msec=caldender.get(Calendar.SECOND)

            var timePickerDialog=TimePickerDialog(this@MainActivity,
object: TimePickerDialog.OnTimeSetListener{

                override fun onTimeSet(view: TimePicker?, hourOfDay: Int,
minute: Int) {
                    binding?.showtime?.setText(""+ hourOfDay + ":" + minute)
                }

            },mhour,mmin,false)

           timePickerDialog.show()

       }
   }
}
```

WEB SERVICES

Web services are software systems designed to support interoperable machine-to-machine interaction over a network. They allow different systems and applications to communicate with each other using standard protocols and data formats. Web services are widely used for

integrating disparate systems, enabling data exchange, and building distributed applications.
Here's an introduction to web services:

- Purpose:
    - Web services enable communication between different software applications running on different platforms and frameworks.
    - They provide a standardized way for systems to interact and exchange data over a network, regardless of the programming language, operating system, or platform they are built on.
    - Web services facilitate the integration of heterogeneous systems and the development of distributed and modular applications.
- Key Characteristics:
    - Interoperability: Web services use standard protocols and data formats such as XML, JSON, SOAP, and REST to ensure interoperability between different systems and platforms.
    - Loose Coupling: Web services promote loose coupling between systems by decoupling the implementation details of the service from its consumers. This allows for easier maintenance, scalability, and evolution of the system.
    - Platform Independence: Web services are platform-independent, meaning they can be accessed and consumed by any application regardless of the platform or technology stack it is built on.
    - Scalability: Web services can scale horizontally by adding more instances of the service to handle increasing demand and workload.
- **TYPES OF WEB SERVICES:**
    - SOAP (Simple Object Access Protocol): SOAP is a protocol for exchanging structured information in the implementation of web services. It uses XML for message format and typically runs over HTTP or SMTP.
    - REST (Representational State Transfer): REST is an architectural style for designing networked applications. It relies on stateless communication and uses standard HTTP methods (GET, POST, PUT, DELETE) for CRUD operations.

    - XML-RPC: XML-RPC is a simple protocol that uses XML to encode messages and HTTP as the transport mechanism. It allows remote procedure calls over the internet.
- Components of Web Services:
    - Service Provider: The service provider is responsible for implementing and exposing the web service.
    - Service Consumer: The service consumer is the application or system that consumes or accesses the web service.
    - Service Description: The service description provides information about the web service, including its operations, input/output parameters, data types, etc. It is typically defined using standards like WSDL (Web Services Description Language) for SOAP-based services or OpenAPI (formerly known as Swagger) for RESTful services.
    - Service Registry: The service registry is a directory or repository that maintains metadata about available web services, allowing service consumers to discover and access them.

The architecture of web services defines the structure, components, and interactions involved in designing and implementing distributed software systems that communicate over a network using standard protocols and data formats. There are several architectural styles for web services, with SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) being the most commonly used. Here's an overview of the architecture of both SOAP and REST web services:

- SOAP (Simple Object Access Protocol):
  SOAP is a protocol for exchanging structured information in the implementation of web services. It defines a standard messaging framework that allows systems to exchange XML-based messages over a network. The architecture of SOAP web services typically includes the following components:
    - Service Provider: The service provider implements the SOAP web service by exposing a set of operations (methods) that can be invoked remotely by service consumers. The service provider publishes a WSDL (Web Services Description Language) document describing the service interface.
    - Service Consumer: The service consumer is the application or system that consumes or invokes the operations exposed by the SOAP web service. The service consumer typically

generates client-side proxy classes based on the WSDL document to interact with the service.
    - SOAP Message: SOAP messages are XML-based documents that contain the information exchanged between the service provider and service consumer. A SOAP message consists of a header section (optional) and a body section containing the payload data.
    - Transport Protocol: SOAP messages are typically transported over HTTP or SMTP (Simple Mail Transfer Protocol). The choice of transport protocol depends on the requirements and constraints of the application.
    - Service Description (WSDL): The WSDL document describes the interface of the SOAP web service, including the operations it supports, input/output parameters, data types, and message formats. WSDL provides a standardized way for service consumers to discover and understand the capabilities of the service.
    - SOAP Binding: SOAP bindings define the rules and conventions for exchanging SOAP messages over a specific transport protocol. Common SOAP bindings include SOAP over HTTP, SOAP over SMTP, and SOAP over JMS (Java Message Service).
- REST (Representational State Transfer):
  REST is an architectural style for designing networked applications based on the principles of simplicity, scalability, and statelessness. RESTful web services use standard HTTP methods (GET, POST, PUT, DELETE) to perform CRUD (Create, Read, Update, Delete) operations on resources identified by URIs (Uniform Resource Identifiers). The architecture of RESTful web services typically includes the following components:
    - Resource: Resources are the key abstractions in RESTful architecture, representing entities or data objects that can be accessed and manipulated by clients. Resources are identified by unique URIs and can have multiple representations (e.g., JSON, XML, HTML).
    - HTTP Methods: RESTful web services use standard HTTP methods to perform operations on resources. GET is used to retrieve resource representations, POST is used to create new resources, PUT is used to update existing resources, and DELETE is used to delete resources.
    - Representation: Resource representations are the data formats used to represent resource state, such as JSON

(JavaScript Object Notation) or XML (eXtensible Markup Language). Clients can request different representations of a resource using content negotiation.

- Uniform Interface: RESTful web services adhere to the principles of a uniform interface, which include using standard HTTP methods, resource identification through URIs, self-descriptive messages, and hypermedia as the engine of application state (HATEOAS).
- Statelessness: RESTful web services are stateless, meaning each request from the client contains all the information necessary for the server to fulfill the request. The server does not store any client state between requests, improving scalability and reliability.
- Hypermedia Controls (HATEOAS): HATEOAS is a principle of RESTful architecture that enables clients to navigate the application state dynamically by following hypermedia links embedded in resource representations. Hypermedia controls provide a self-descriptive and discoverable API, reducing the coupling between clients and servers.

THE COMPONENTS OF WEB SERVICES

The components of web services are the essential building blocks that collectively enable the creation, deployment, and consumption of distributed software systems over a network. These components work together to facilitate communication, data exchange, and interoperability between different systems and applications. Here are the key components of web services:

- Service Provider:
    - The service provider is the entity responsible for implementing and exposing the web service to potential consumers.
    - It develops the functionality and logic of the service, including defining the service interface, implementing service operations, and hosting the service on a server.
    - The service provider publishes metadata about the service, such as its interface definition and endpoint location, to allow consumers to discover and access the service.
- Service Consumer:

    - The service consumer is the entity that consumes or interacts with the web service to access its functionality and data.
    - It may be an application, system, or component that initiates requests to the service provider to perform specific operations or retrieve information.
    - The service consumer typically generates client-side code or proxies based on the service metadata to communicate with the service provider.
- Service Interface:
    - The service interface defines the contract or agreement between the service provider and consumer regarding how they communicate and interact with each other.
    - It specifies the operations supported by the service, including their input parameters, output results, and any errors or exceptions that may occur.
    - The service interface is typically described using a formal language or notation, such as WSDL (Web Services Description Language) for SOAP-based services or OpenAPI (formerly known as Swagger) for RESTful services.
- Service Description:
    - The service description provides metadata or documentation about the web service, including its capabilities, interface, protocols, and data formats.
    - It allows service consumers to discover, understand, and interact with the service without prior knowledge of its implementation details.
    - Service descriptions are typically represented using standardized formats, such as WSDL for SOAP services or OpenAPI for RESTful services, and are published by the service provider for consumption by potential clients.
- Service Endpoint:
    - The service endpoint is the network address or location where the web service is hosted and can be accessed by service consumers.
    - It consists of a Uniform Resource Identifier (URI) or URL (Uniform Resource Locator) that uniquely identifies the service and specifies the communication protocol (e.g., HTTP, HTTPS).
    - Service endpoints allow clients to connect to the service provider over the network and invoke its operations to perform desired tasks or retrieve information.

- Data Format:
    - Data format refers to the encoding and representation of data exchanged between the service provider and consumer during communication.
    - Web services support various data formats, such as XML (eXtensible Markup Language), JSON (JavaScript Object Notation), and SOAP (Simple Object Access Protocol), depending on the protocol and messaging format used.
    - The choice of data format affects factors such as interoperability, performance, and complexity of data processing and serialization/deserialization.
- ## WEB SERVICES ADVANTAGES

Web services offer numerous advantages that make them a popular choice for building distributed systems and enabling interoperable communication between disparate applications and platforms. Here are some key advantages of web services:

- Interoperability:
    - Web services facilitate interoperable communication between different systems and platforms, regardless of the programming language, operating system, or hardware infrastructure used.
    - They use standard protocols and data formats such as HTTP, XML, JSON, SOAP, and REST, enabling seamless integration and data exchange between heterogeneous environments.
- Platform Independence:
    - Web services are platform-independent, allowing them to be accessed and consumed by any application or system regardless of the underlying technology stack.
    - They enable organizations to build distributed systems that can run on diverse platforms and infrastructures, including desktops, mobile devices, cloud environments, and IoT (Internet of Things) devices.
- Loose Coupling:
    - Web services promote loose coupling between systems by decoupling the implementation details of the service provider from its consumers.
    - They allow systems to interact and exchange data without relying on specific implementation details, enabling easier

maintenance, scalability, and evolution of distributed applications.

- Scalability:
    - Web services can scale horizontally by adding more instances of the service to handle increasing demand and workload.
    - They support distributed architectures and cloud deployments, allowing organizations to scale their systems dynamically to accommodate growing user bases and workloads.
- Reusability:
    - Web services promote reusability of components and services by exposing modular and encapsulated functionality that can be accessed and reused by multiple applications.
    - They enable organizations to leverage existing services and infrastructure to build new applications and functionalities more efficiently.
- Ease of Integration:
    - Web services simplify the integration of disparate systems and applications by providing standardized communication protocols and interfaces.
    - They enable seamless data exchange and interaction between internal systems, external partners, and third-party services, streamlining business processes and workflows.
- Security:
    - Web services support various security mechanisms, such as SSL/TLS encryption, message signing, and authentication, to ensure the confidentiality, integrity, and authenticity of data exchanged over the network.
    - They enable organizations to implement robust security policies and controls to protect sensitive information and mitigate security risks.
- Cost-Effectiveness:
    - Web services reduce development and maintenance costs by promoting code reuse, interoperability, and scalability.
    - They enable organizations to leverage existing infrastructure and resources more efficiently, avoiding the need for costly and time-consuming custom integrations and proprietary solutions.

JSON

JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is widely used as a format for transmitting data between a server and a client in web applications, APIs, and various other systems. JSON is language-independent and supported by most programming languages and frameworks.
Here's an introduction to JSON:

- Data Format:
  - JSON is a text-based data format that consists of key-value pairs and arrays, organized in a hierarchical structure.
  - It is based on two fundamental data structures: objects (collection of key-value pairs enclosed in curly braces {}) and arrays (ordered list of values enclosed in square brackets []).
  - JSON supports basic data types such as strings, numbers, booleans, null, objects, and arrays. It also allows nesting of objects and arrays to represent complex data structures.
- Example:

```
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "zipCode": "12345"
  },
  "interests": ["reading", "coding", "traveling"]
}
```

- Key Characteristics:
  - Human Readable: JSON is designed to be easy for humans to read and write, making it a popular choice for configuration files, data serialization, and API responses.
  - Lightweight: JSON has a simple and compact syntax, which makes it lightweight and efficient for transmitting data over the network.
  - Language-Independent: JSON is language-independent and can be parsed and generated by most programming

    languages and frameworks. It is not tied to any specific programming language or platform.
    - Widely Supported: JSON is supported by a wide range of tools, libraries, and frameworks across different programming languages and platforms. It has become a de facto standard for data interchange on the web.
    - Schemaless: JSON is schemaless, meaning there are no predefined schemas or data types enforced by the format itself. This allows for flexibility and dynamic handling of data structures.
- Usage:
    - JSON is commonly used in web development for transmitting data between a client and a server in AJAX requests, RESTful APIs, and web services.
    - It is used for storing configuration data, application state, and user preferences in web applications and mobile apps.
    - JSON is also used for serializing and deserializing data in programming languages for data storage, inter-process communication, and data exchange between different systems and components.

Overall, JSON is a versatile and widely adopted data format that offers simplicity, readability, and interoperability, making it a popular choice for transmitting and storing structured data in modern software development. It provides a lightweight and flexible solution for exchanging data between systems and applications in a variety of contexts.

JSON ARCHITECTURE

JSON (JavaScript Object Notation) itself does not have a specific architecture like web services or other software systems. Instead, it is a lightweight data interchange format that is used to represent and exchange structured data between systems. However, the use of JSON within software architectures can be described in terms of how it fits into the overall design and communication patterns of a system. Here's how JSON typically fits into various software architectures:

- Client-Server Architecture:

  - In a client-server architecture, JSON is often used as the data format for communication between the client and the server.
  - The client sends JSON-formatted requests to the server to perform actions or retrieve data, and the server responds with JSON-formatted responses containing the requested data or status information.
  - JSON is commonly used in AJAX requests, RESTful APIs, and other web service architectures where data needs to be transmitted between the client and the server over HTTP.
- Microservices Architecture:
  - In a microservices architecture, JSON is often used as the data format for communication between microservices.
  - Each microservice may expose APIs that accept and return JSON-formatted data, allowing them to communicate with each other in a language-independent and platform-independent manner.
  - JSON is well-suited for microservices architectures due to its simplicity, flexibility, and ease of parsing and generating data in different programming languages.
- Message Queue Architecture:
  - In a message queue architecture, JSON can be used as the message format for passing messages between different components or systems.
  - Messages sent between producers and consumers in the message queue may be serialized and deserialized to JSON format, allowing them to be easily processed and understood by different systems.
  - JSON provides a lightweight and human-readable format for representing message data, making it suitable for message-oriented architectures.
- Data Storage Architecture:
  - In data storage architectures, JSON is often used as a data format for storing structured data in databases or NoSQL data stores.
  - JSON documents can be stored as records or documents in databases such as MongoDB, Couchbase, or Firebase, allowing for flexible and schemaless data storage.
  - JSON's hierarchical structure and support for nested objects and arrays make it well-suited for representing complex data structures in data storage architectures.

Overall, JSON is a versatile data format that can be used in various software architectures to facilitate communication, data exchange, and data storage between different components and systems. Its simplicity, flexibility, and interoperability make it a popular choice for representing structured data in modern software development.

TYPES OF WEBSERVICES

In the context of web development and APIs, there are several types of requests that clients can make to servers to interact with resources. These requests are typically categorized based on the HTTP method used for making the request. The most common HTTP methods used for making requests are:

- GET:
  - The GET method is used to request data from a specified resource.
  - It is used to retrieve data or information from the server without modifying it.
  - GET requests should only be used for idempotent operations, meaning that they should not have any side effects on the server.
  - Example: Retrieving a list of products from an e-commerce website.
- POST:
  - The POST method is used to submit data to be processed to a specified resource.
  - It is commonly used for creating new resources or submitting form data to the server.
  - POST requests can have side effects on the server, such as creating or updating data.
  - Example: Submitting a new order to an e-commerce website.
- PUT:
  - The PUT method is used to update data on a specified resource.
  - It is commonly used for updating existing resources with new data.
  - PUT requests are idempotent, meaning that multiple identical requests should have the same effect as a single request.
  - Example: Updating the details of a user profile.
- DELETE:

  - The DELETE method is used to delete a specified resource from the server.
  - It is commonly used for deleting existing resources from the server.
  - DELETE requests are idempotent, meaning that multiple identical requests should have the same effect as a single request.
  - Example: Deleting a customer record from a database.
- PATCH:
  - The PATCH method is used to apply partial modifications to a resource.
  - It is commonly used for updating parts of an existing resource without affecting the rest of the resource.
  - PATCH requests are not necessarily idempotent, meaning that multiple identical requests may have different effects.
  - Example: Updating the status of an order without modifying other details.
- HEAD:
  - The HEAD method is similar to the GET method but does not return a message body in the response.
  - It is commonly used to retrieve metadata about a resource, such as headers and status codes, without transferring the entire content.
  - HEAD requests are typically used for checking the availability and properties of a resource.
  - Example: Checking if a file exists on a server without downloading its contents.

These are the most commonly used HTTP methods for making requests to web servers. Each method has its specific purpose and usage, and understanding their differences is essential for building RESTful APIs and interacting with web resources effectively.

RETROFIT

Retrofit is a type-safe HTTP client for Android and Java that simplifies the process of consuming RESTful web services. It is a widely used library for making network requests and handling API responses in Android apps. Retrofit is developed by Square, the same company behind other popular libraries such as OkHttp.

Here's an overview of Retrofit and its key features:

- Declarative API:
    - Retrofit provides a high-level, declarative API for defining HTTP requests and handling responses.
    - It allows developers to define interfaces that represent RESTful API endpoints, with methods corresponding to different HTTP operations (GET, POST, PUT, DELETE, etc.) and annotations for specifying request parameters, headers, and response types.
- Type-Safe Requests:
    - Retrofit generates implementation code for the defined interfaces at compile time, providing type safety and ensuring that requests and responses are properly handled.
    - Developers can define Java interfaces with method signatures that match the expected request and response types, making the code more readable and maintainable.
- Integration with OkHttp:
    - Retrofit seamlessly integrates with OkHttp, a powerful HTTP client library for Java and Android, for handling network communication.
    - OkHttp provides features such as connection pooling, request/response caching, and interceptors, which Retrofit leverages to provide efficient and reliable network communication.
- Serialization and Deserialization:
    - Retrofit supports automatic serialization of request bodies and deserialization of response bodies using converter libraries such as Gson, Moshi, Jackson, and XML.
    - Developers can configure Retrofit to use their preferred serialization/deserialization library based on the data format (JSON, XML) used by the API.
- Asynchronous and Synchronous Execution:
    - Retrofit supports both synchronous and asynchronous execution of HTTP requests.
    - Asynchronous requests are executed on background threads, allowing the main UI thread to remain responsive, while synchronous requests are executed on the calling thread, potentially blocking it until the request completes.
- Error Handling:

    - Retrofit provides built-in support for handling HTTP errors and network exceptions, such as timeouts, connection errors, and server errors.
    - Developers can define custom error handling logic, such as retrying requests, logging errors, or displaying error messages to the user.
- Easy Integration with Android:
    - Retrofit is designed to work seamlessly with Android apps and integrates well with other Android libraries and frameworks.
    - It supports features such as RxJava for reactive programming, LiveData for observing data changes, and Coroutines for asynchronous programming.

Sample Project Using Retrofit

API:

API:
Registration
URL: https://androindian.com/test/Register.php
Request:
{“username”:“Raj”,“password”:“1234”,“email”:“Rajtest2356178@admin}
Response:
{“key”:”You are registered successfully.”,”status”:”success”}
{“key”:”Email already Exists”,”status”:”failed”}

API:
URL: https://androindian.com/test/Register.php
Request:
{“username”:“Raj@gmail.com”,“password”:“1234”}
Response:
{“key”:”Login successfully.”,”status”:”success”}
{“key”:”invalid credentials”,”status”:”failed”}

Build .gradle

plugins {

```
  id("com.android.application")
  id("org.jetbrains.kotlin.android")
}

android {
  namespace = "com.example.project8pm"
  compileSdk = 34

  defaultConfig {
     applicationId = "com.example.project8pm"
     minSdk = 24
     targetSdk = 34
     versionCode = 1
     versionName = "1.0"

     testInstrumentationRunner =
"androidx.test.runner.AndroidJUnitRunner"
  }

  buildTypes {
     release {
        isMinifyEnabled = false
        proguardFiles(
           getDefaultProguardFile("proguard-android-optimize.txt"),
           "proguard-rules.pro"
        )
     }
  }
  compileOptions {
     sourceCompatibility = JavaVersion.VERSION_1_8
     targetCompatibility = JavaVersion.VERSION_1_8
  }
  kotlinOptions {
     jvmTarget = "1.8"
  }
  dataBinding{
     enable=true
  }
}

dependencies {
  implementation("androidx.core:core-ktx:1.12.0")
  implementation("androidx.appcompat:appcompat:1.6.1")
  implementation("com.google.android.material:material:1.11.0")
  implementation("androidx.constraintlayout:constraintlayout:2.1.4")
  testImplementation("junit:junit:4.13.2")
  androidTestImplementation("androidx.test.ext:junit:1.1.5")
  androidTestImplementation("androidx.test.espresso:espresso-
core:3.5.1")

  implementation ("com.squareup.retrofit2:converter-gson:2.4.0")
  implementation ("com.squareup.retrofit2:retrofit:2.4.0")
  implementation ("com.google.code.gson:gson:2.8.9")
  implementation ("com.squareup.picasso:picasso:2.5.2")

}
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
<androidx.cardview.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_margin="5dp">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Register Here"
        android:layout_gravity="center" />
    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/username"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="UserName"
        android:layout_margin="5dp"
        style="@style/Widget.MaterialComponents.TextInputLayout.Outli
nedBox">

        <com.google.android.material.textfield.TextInputEditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            />

    </com.google.android.material.textfield.TextInputLayout>
    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/email"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="email"
        android:layout_margin="5dp"
        style="@style/Widget.MaterialComponents.TextInputLayout.Outli
nedBox">

        <com.google.android.material.textfield.TextInputEditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            />

    </com.google.android.material.textfield.TextInputLayout>
    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/password"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Password"
        android:layout_margin="5dp"
        style="@style/Widget.MaterialComponents.TextInputLayout.Outli
nedBox">

        <com.google.android.material.textfield.TextInputEditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            />

    </com.google.android.material.textfield.TextInputLayout>
    <Button
        android:id="@+id/login"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Login"
        android:padding="5dp"
        android:layout_margin="5dp"
        android:layout_gravity="center"
        />

</LinearLayout>
</androidx.cardview.widget.CardView>

</RelativeLayout>
</layout>
```

MainActivity.kt

```
package com.example.project8pm

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast
import androidx.databinding.DataBindingUtil
import com.example.project8pm.databinding.ActivityMainBinding
import com.google.gson.JsonParser
import org.json.JSONObject
import retrofit2.Call
import retrofit2.Response
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class MainActivity : AppCompatActivity() {

    var binding: ActivityMainBinding?=null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
       binding= DataBindingUtil.
setContentView(this,R.layout.activity_main)
```

```
    binding?.login?.setOnClickListener {

        var uname=binding?.username?.editText?.text.toString().trim()
        var uemail=binding?.email?.editText?.text.toString().trim()
        var upass=binding?.password?.editText?.text.toString().trim()

        var jsonObject=JSONObject()
        jsonObject.put("username",uname)
        jsonObject.put("email",uemail)
        jsonObject.put("password",upass)

        var retrofit=Retrofit.Builder().
                baseUrl("https://androindian.com/test/").
                addConverterFactory(GsonConverterFactory.create()).build()

        var retroInterface=retrofit.create(SampleInterface::class.java)

        var requestjson=
JsonParser().parse(jsonObject.toString()).asJsonObject

        var regresponsecall=retroInterface.createUser(requestjson)

        regresponsecall?.enqueue(object :
retrofit2.Callback<RegisterResponse>{
            override fun onResponse(
                call: Call<RegisterResponse>,
                response: Response<RegisterResponse>
            ) {
                var res=response.body()?.status

                if(res.equals("success")){
                    var res1=response.body()?.key
                    Toast.makeText(this@MainActivity,res1,Toast.LENGTH_L
ONG).show()
                    var intent= Intent(this@MainActivity,Login::class.java)
                    startActivity(intent)
                }else{
                    var res1=response.body()?.key
                    Toast.makeText(this@MainActivity,res1,Toast.LENGTH_L
ONG).show()
                }
            }
            override fun onFailure(call: Call<RegisterResponse>, t: 
Throwable) {
                Toast.makeText(this@MainActivity,""+t,Toast.LENGTH_LO
NG).show()
            }
        })


      }
  }
}
```

SampleInterface.kt

```
package com.example.project8pm

import com.google.gson.JsonObject
import retrofit2.Call
import retrofit2.http.Body
import retrofit2.http.GET
import retrofit2.http.Headers
import retrofit2.http.POST

public interface SampleInterface {

  @Headers("Content-Type:application/json")
  @POST("Register_raw.php")
  fun createUser(@Body jsonObject: JsonObject): 
Call<RegisterResponse>

  @Headers("Content-Type:application/json")
  @POST("Login_raw.php")
  fun loginUser(@Body jsonObject: JsonObject): Call<LoginResponse>

  @Headers("Content-Type:application/json")
  @GET("users?page=2")
  fun loadData(): Call<ListResponse>
}
```

RegisterResponse.kt

```
package com.example.project8pm

import com.google.gson.annotations.Expose
import com.google.gson.annotations.SerializedName

class RegisterResponse {
  @SerializedName("key")
  @Expose
  var key: String? = null

  @SerializedName("status")
  @Expose
  var status: String? = null
}
```

Activity_login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
  <RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      tools:context=".Login">
      <androidx.cardview.widget.CardView
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_margin="10dp"
          android:layout_centerHorizontal="true"
          android:layout_centerVertical="true">
          <LinearLayout
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:orientation="vertical"
              android:layout_margin="5dp">
              <TextView
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:text="Login Here"
                  android:layout_gravity="center" />



              <com.google.android.material.textfield.TextInputLayout
                  android:id="@+id/email"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="email"
            android:layout_margin="5dp"
            style="@style/Widget.MaterialComponents.TextInputLayout.
OutlinedBox">

            <com.google.android.material.textfield.TextInputEditText
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                />

        </com.google.android.material.textfield.TextInputLayout>
        <com.google.android.material.textfield.TextInputLayout
            android:id="@+id/password"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Password"
            android:layout_margin="5dp"
            style="@style/Widget.MaterialComponents.TextInputLayout.
OutlinedBox">

            <com.google.android.material.textfield.TextInputEditText
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                />

        </com.google.android.material.textfield.TextInputLayout>
        <Button
            android:id="@+id/login"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Login"
            android:padding="5dp"
            android:layout_margin="5dp"
            android:layout_gravity="center"
            />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/newuser"
```

```
                android:text="New User Register Here"
                android:textSize="25sp"/>

            </LinearLayout>
        </androidx.cardview.widget.CardView>

    </RelativeLayout>
</layout>
```

Login.kt

```
package com.example.project8pm

import android.content.Intent
import android.content.SharedPreferences
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast
import androidx.databinding.DataBindingUtil
import com.example.project8pm.databinding.ActivityLoginBinding
import com.google.gson.JsonParser
import org.json.JSONObject
import retrofit2.Call
import retrofit2.Response
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class Login : AppCompatActivity() {
    var binding: ActivityLoginBinding?=null
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       binding=
DataBindingUtil.setContentView(this,R.layout.activity_login)

       binding?.newuser?.setOnClickListener {
           var intent= Intent(this@Login,MainActivity::class.java)
           startActivity(intent)
       }

       binding?.login?.setOnClickListener {

           var uemail=binding?.email?.editText?.text.toString().trim()
           var upass=binding?.password?.editText?.text.toString().trim()
```

```
        var jsonObject= JSONObject()

        jsonObject.put("username",uemail)
        jsonObject.put("password",upass)

        var retrofit= Retrofit.Builder().
        baseUrl("https://androindian.com/test/").
        addConverterFactory(GsonConverterFactory.create()).build()

        var retroInterface=retrofit.create(SampleInterface::class.java)

        var requestjson=
JsonParser().parse(jsonObject.toString()).asJsonObject

        var logsponsecall=retroInterface.loginUser(requestjson)

        logsponsecall?.enqueue(object :
retrofit2.Callback<LoginResponse>{
            override fun onResponse(
                call: Call<LoginResponse>,
                response: Response<LoginResponse>
            ) {
                var res=response.body()?.status

                if(res.equals("success")){
                    var res1=response.body()?.key
                    Toast.makeText(this@Login,res1,
Toast.LENGTH_LONG).show()
                    var intent=Intent(this@Login,DashBoard::class.java)
                    startActivity(intent)

                    var
sharedpredpreferences=getSharedPreferences("Login",
MODE_PRIVATE)
                    var editor=sharedpredpreferences.edit()
                   // editor.put("key","vales")
                    editor.putString("email",uemail)
                    editor.putString("password",upass)
                    editor.commit()
                }else{
                    var res1=response.body()?.key
                    Toast.makeText(this@Login,res1,
Toast.LENGTH_LONG).show()
                }
            }

            override fun onFailure(call: Call<LoginResponse>, t:
Throwable) {
                Toast.makeText(this@Login,""+t,
Toast.LENGTH_LONG).show()
            }
        })


    }
  }
}
```

LoginResponse.kt

```
package com.example.project8pm

import com.google.gson.annotations.Expose
import com.google.gson.annotations.SerializedName

class LoginResponse {
  @SerializedName("key")
  @Expose
  var key: String? = null

  @SerializedName("status")
  @Expose
  var status: String? = null
}
```

Activity_dashboard.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".DashBoard"
  android:orientation="vertical">
```

```
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/logout"
        android:layout_margin="10dp"
        android:text="Logout"/>

    <androidx.recyclerview.widget.RecyclerView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:id="@+id/rec"/>

</LinearLayout>
</layout>
```

Dashboard.kt

```
package com.example.project8pm

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast
import androidx.databinding.DataBindingUtil
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.project8pm.databinding.ActivityDashBoardBinding
import retrofit2.Call
import retrofit2.Response
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class DashBoard : AppCompatActivity() {
    var connectionChecking: ConnectionChecking?=null
    var binding:ActivityDashBoardBinding?=null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding=DataBindingUtil.setContentView(this,R.layout.activity_dash
_board)

        var sharedpredpreferences=getSharedPreferences("Login",
MODE_PRIVATE)
```

```
        var s1=sharedpredpreferences.getString("email",null)
        var s2=sharedpredpreferences.getString("password",null)

        Toast.makeText(this@DashBoard,s1+s2,Toast.LENGTH_LONG).sh
ow()

        binding?.logout?.setOnClickListener {
            var editor=sharedpredpreferences.edit()
            editor.clear()
            editor.apply()
        }
        connectionChecking=ConnectionChecking()

        if(connectionChecking?.isConnectingToInternet(this@DashBoard)
== true) {

            var retrofit = Retrofit.Builder().baseUrl("https://reqres.in/api/")
                .addConverterFactory(GsonConverterFactory.create()).build()

            var retroInterface = retrofit.create(SampleInterface::class.java)

            var listCall = retroInterface.loadData()
            listCall?.enqueue(object : retrofit2.Callback<ListResponse> {
                override fun onResponse(
                    call: Call<ListResponse>,
                    response: Response<ListResponse>
                ) {
                    var res = response.body()?.total

                    if (res !== 0) {
                        var linearLayoutManager =
                            LinearLayoutManager(this@DashBoard,
LinearLayoutManager.VERTICAL, false)
                        binding?.rec?.layoutManager = linearLayoutManager
                        var customAdapter = CustomAdapter(
                            this@DashBoard,
                            response.body()?.data as List<ListInsideData>
                        )
                        binding?.rec?.adapter = customAdapter

                    } else {
```

```
                    Toast.makeText(this@DashBoard, "No data found",
Toast.LENGTH_LONG).show()
                }
            }

            override fun onFailure(call: Call<ListResponse>, t: Throwable) {
                Toast.makeText(this@DashBoard, "" + t,
Toast.LENGTH_LONG).show()
            }
        })
    }
    else{
        Toast.makeText(this@DashBoard,"Please check
internrt",Toast.LENGTH_LONG).show()
    }
  }
}
```

ListResponse.kt

```
package com.example.project8pm

import com.google.gson.annotations.Expose
import com.google.gson.annotations.SerializedName

class ListResponse {
  @SerializedName("page")
  @Expose
  var page: Int? = null

  @SerializedName("per_page")
  @Expose
  var perPage: Int? = null

  @SerializedName("total")
  @Expose
  var total: Int? = null

  @SerializedName("total_pages")
  @Expose
  var totalPages: Int? = null

  @SerializedName("data")
  @Expose
```

```
    var data: List<ListInsideData>? = null

    @SerializedName("support")
    @Expose
    var support: Support? = null
}
```

ListInsideData.kt

```
package com.example.project8pm

import com.google.gson.annotations.Expose
import com.google.gson.annotations.SerializedName

class ListInsideData {
    @SerializedName("id")
    @Expose
    var id: Int? = null

    @SerializedName("email")
    @Expose
    var email: String? = null

    @SerializedName("first_name")
    @Expose
    var firstName: String? = null

    @SerializedName("last_name")
    @Expose
    var lastName: String? = null

    @SerializedName("avatar")
    @Expose
    var avatar: String? = null
}
```

Support.kt

```
package com.example.project8pm

import com.google.gson.annotations.Expose
import com.google.gson.annotations.SerializedName

class Support {
    @SerializedName("url")
    @Expose
    var url: String? = null

    @SerializedName("text")
    @Expose
    var text: String? = null
}
CustomAdapter.kt
package com.example.project8pm

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageView
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.squareup.picasso.Picasso


class CustomAdapter(dashBoard: DashBoard, listInsideData:
List<ListInsideData>) :
RecyclerView.Adapter<CustomAdapter.MyViewHolder>() {

    var adpContext: Context
    var adpdata:List<ListInsideData>
    init {
        adpdata=listInsideData
        adpContext=dashBoard
    }
    class MyViewHolder(itemView:
View):RecyclerView.ViewHolder(itemView) {

        var adpimage: ImageView?=null
        var adpfname:TextView?=null
        var adplname: TextView?=null
        var adpid:TextView?=null
        var adpemail:TextView?=null

        init {
            adpfname=itemView.findViewById(R.id.firstname)
            adpemail=itemView.findViewById(R.id.email)
            adpid=itemView.findViewById(R.id.uid)
            adplname=itemView.findViewById(R.id.lastname)
            adpimage=itemView.findViewById(R.id.profile)
        }

    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
CustomAdapter.MyViewHolder {
        var
view=LayoutInflater.from(parent.context).inflate(R.layout.custom,parent,f
alse)
        return MyViewHolder(view)
    }

    override fun onBindViewHolder(holder: CustomAdapter.MyViewHolder,
position: Int) {
        holder.adpfname?.text=adpdata.get(holder.adapterPosition).firstNa
me
        holder.adpid?.text=adpdata.get(holder.adapterPosition).id.toString()
        holder.adplname?.text=adpdata.get(holder.adapterPosition).lastNa
me
        holder.adpemail?.text=adpdata.get(holder.adapterPosition).email

        Picasso.with(adpContext).load(adpdata.get(holder.adapterPosition).
avatar).into(holder.adpimage)

    }

    override fun getItemCount(): Int {
        return adpdata.size
    }

}
```

Custom.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <androidx.cardview.widget.CardView
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dp">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:layout_margin="5dp">
            <ImageView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:id="@+id/profile"/>
            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical">
                <TextView
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:id="@+id/uid"/>
                <TextView
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:id="@+id/firstname"/>
                <TextView
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:id="@+id/lastname"/>
                <TextView
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:id="@+id/email"/>
            </LinearLayout>
        </LinearLayout>
    </androidx.cardview.widget.CardView>
</LinearLayout>
```

Activity_splash.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Splash">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/iv"
        android:src="@mipmap/ic_launcher"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"/>

</RelativeLayout>
</layout>
```

Splash.kt

```
package com.example.project8pm

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.os.Handler
import android.view.animation.Animation
import android.view.animation.AnimationUtils
import androidx.databinding.DataBindingUtil
import com.example.project8pm.databinding.ActivitySplashBinding


class Splash : AppCompatActivity() {
    var binding: ActivitySplashBinding?=null
     var animation: Animation?=null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding=DataBindingUtil.setContentView(this,R.layout.activity_splas
h)

        animation=AnimationUtils.loadAnimation(this@Splash,R.anim.rotate
)

        binding?.iv?.startAnimation(animation)
        var sharedpredpreferences=getSharedPreferences("Login",
MODE_PRIVATE)
```

```
        var s1=sharedpredpreferences.getString("email",null)
        var s2=sharedpredpreferences.getString("password",null)

        Handler().postDelayed(Runnable {
            if(s1.isNullOrBlank()) {
                var intent = Intent(this@Splash, Login::class.java)
                startActivity(intent)
            }else{

                var intent = Intent(this@Splash, DashBoard::class.java)
                startActivity(intent)
            }
        },10000)


    }
}
```

ConnectionChecking.kt

```
package com.example.project8pm

import android.content.Context
import android.net.ConnectivityManager
import android.net.NetworkInfo

class ConnectionChecking {

    fun isConnectingToInternet(context: Context):Boolean{

        var
connectivity=context.getSystemService(Context.CONNECTIVITY_SERV
ICE) as ConnectivityManager

        if(connectivity!=null){
            var info=connectivity.allNetworkInfo
            if(info!=null)
                for (i in info.indices)
                    if(info[i].state== NetworkInfo.State.CONNECTED){
                        return true
                    }
        }
        return false
```

```
   }
}
AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools">

   <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
   <uses-permission android:name="android.permission.INTERNET" />

   <application
       android:allowBackup="true"
       android:dataExtractionRules="@xml/data_extraction_rules"
       android:fullBackupContent="@xml/backup_rules"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:roundIcon="@mipmap/ic_launcher_round"
       android:supportsRtl="true"
       android:theme="@style/Theme.Project8PM"
       tools:targetApi="31">
       <activity
          android:name=".Login"
          android:exported="false" />
       <activity
          android:name=".DashBoard"
          android:exported="false" />
       <activity
          android:name=".MainActivity"
          android:exported="false" />
       <activity
          android:name=".Splash"
          android:exported="true">
          <intent-filter>
             <action android:name="android.intent.action.MAIN" />

             <category android:name="android.intent.category.LAUNCHER"
/>
          </intent-filter>
       </activity>
   </application>
```

```
</manifest>
```

VOLLEY

Volley is an HTTP library developed by Google that simplifies network requests in Android applications. It provides a straightforward API for making network requests, handling responses, and performing image loading. Volley is designed to be fast, efficient, and easy to use, making it a popular choice for networking tasks in Android apps.
Here are some key features of Volley:

- Ease of Use:
  - Volley provides a simple and intuitive API for making network requests, requiring minimal boilerplate code.
  - It offers a RequestQueue class for managing and dispatching network requests, making it easy to send multiple requests simultaneously.
- Asynchronous Requests:
  - Volley performs network requests asynchronously by default, allowing the main UI thread to remain responsive.
  - It automatically manages threading and concurrency, handling network operations on background threads and delivering results on the main thread.
- Request Prioritization and Cancellation:
  - Volley allows developers to prioritize network requests based on their importance or urgency.
  - It supports request cancellation, allowing developers to cancel ongoing requests if they are no longer needed or if the user navigates away from the current screen.
- Automatic Retries and Backoff:
  - Volley includes built-in support for automatic retries and backoff strategies for handling transient network errors.
  - It automatically retries failed requests with exponential backoff, reducing the likelihood of server overload and improving the reliability of network operations.
- Request Caching:
  - Volley supports request caching, allowing responses to be cached on the device to improve performance and reduce network usage.
  - It provides a flexible caching mechanism that can be customized based on the requirements of the application.

- Image Loading:
  - Volley includes a specialized module for loading images from network URLs and caching them in memory and/or disk.
  - It provides efficient handling of image requests, including automatic resizing, caching, and memory management.
- Customizable Retry Policies and Timeouts:
  - Volley allows developers to customize retry policies and timeouts for individual requests or for the entire RequestQueue.
  - It provides fine-grained control over network behavior, allowing developers to optimize performance and reliability based on specific requirements.
- Integration with Android:
  - Volley is designed to integrate seamlessly with Android applications and follows Android best practices and conventions.
  - It supports features such as asynchronous loading of images into ImageViews, request prioritization, and automatic request cancellation in response to activity lifecycle events.

Overall, Volley is a powerful and versatile library for handling network requests in Android applications. Its simplicity, efficiency, and rich feature set make it a popular choice for developers building networking-enabled apps.

Example program
Build.gardle

```
plugins {
  id("com.android.application")
  id("org.jetbrains.kotlin.android")
}

android {
  namespace = "com.example.volleyexreg"
  compileSdk = 34

  defaultConfig {
    applicationId = "com.example.volleyexreg"
    minSdk = 24
    targetSdk = 34
    versionCode = 1
    versionName = "1.0"
```

```
        testInstrumentationRunner =
"androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = "1.8"
    }
    dataBinding{
        enable=true
    }
}

dependencies {

    implementation("androidx.core:core-ktx:1.12.0")
    implementation("androidx.appcompat:appcompat:1.6.1")
    implementation("com.google.android.material:material:1.11.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-
core:3.5.1")
    implementation ("com.android.volley:volley:1.2.1")
}
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
  <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
      tools:context="com.androindian.retrofit.MainActivity">

      <EditText
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:hint="Name"
         android:id="@+id/name"/>

      <EditText
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:hint="Email"
         android:id="@+id/email"/> <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Pass"
      android:id="@+id/Pass"/>

      <Button
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:text="Reg"
         android:id="@+id/Reg"/>


  </LinearLayout>
</layout>
```

MainActivity.kt

```
package com.example.volleyexreg

import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.databinding.DataBindingUtil
import com.android.volley.Request
import com.android.volley.RequestQueue
import com.android.volley.toolbox.JsonObjectRequest
import com.android.volley.toolbox.Volley

import com.example.volleyexreg.databinding.ActivityMainBinding
import org.json.JSONException
import org.json.JSONObject

class MainActivity : AppCompatActivity() {

   var binding: ActivityMainBinding?=null

   var url="https://androindian.com/test/Register_raw.php"

   var requestQueue: RequestQueue? = null

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       binding=
DataBindingUtil.setContentView(this,R.layout.activity_main)
       requestQueue= Volley.newRequestQueue(this@MainActivity)

       binding?.Reg?.setOnClickListener{
           var jsonObject= JSONObject()

          // jsonObject.put("key","Values")
           jsonObject.put("username",binding?.name?.text.toString().trim())
           jsonObject.put("password",binding?.Pass?.text.toString().trim())
           jsonObject.put("email",binding?.email?.text.toString().trim())

           var jsonObjectRequest=
JsonObjectRequest(Request.Method.POST,
               url,jsonObject, { response->

                   var res=response.getString("status")

                   if(res.equals("failed", ignoreCase=true)){
                       var res1=response.getString("key")
```

```
                    Toast.makeText(this@MainActivity,res1,
Toast.LENGTH_LONG).show()
                }else{
                    var res1=response.getString("key")
                    Toast.makeText(this@MainActivity,res1,
Toast.LENGTH_LONG).show()

                }


            }){
            error->
            Toast.makeText(this@MainActivity,""+error.toString(),
Toast.LENGTH_LONG).show()
        }
        requestQueue?.add(jsonObjectRequest)
    }
  }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools">

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

  <application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:usesCleartextTraffic="true"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.Volleyexreg"
    tools:targetApi="31">
    <activity
      android:name=".MainActivity"
      android:exported="true">
      <intent-filter>
```

```
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER"
/>
        </intent-filter>
    </activity>
</application>

</manifest>
```

FRAGMENTS

Fragments are a fundamental building block of Android user interfaces that represent a portion of a user interface or behavior within an Activity. They were introduced in Android 3.0 (API level 11) as a way to create more modular and flexible UI designs, especially for larger screen sizes such as tablets.
Here's an overview of fragments and their key features:

- Modular UI Components:
  - Fragments represent reusable portions of a UI that can be combined and reconfigured within an Activity.
  - They allow developers to break down the user interface into smaller, self-contained components, making it easier to manage and maintain complex UIs.
- Lifecycle:
  - Fragments have their own lifecycle, similar to Activities, with methods such as onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy().
  - They are tightly coupled with the lifecycle of their hosting Activity and can receive lifecycle callbacks from the Activity.
- UI Components:
  - Fragments can contain their own layout and UI components, including views, buttons, text fields, etc.
  - They can inflate layouts using XML resources or programmatically create UI elements.
- Reusability:
  - Fragments are designed to be reusable across multiple Activities and layouts.

    - They can be included in different layouts or hosted by different Activities, allowing developers to create modular and flexible UI designs.
- Communication:
    - Fragments can communicate with their hosting Activity and other fragments within the same Activity.
    - They can send data or events to the hosting Activity using interfaces or callbacks, allowing for interaction between different UI components.
- Back Stack:
    - Fragments can be added to a back stack, allowing users to navigate back to previous fragments using the back button.
    - The back stack is managed by the FragmentManager, which handles transactions and manages the lifecycle of fragments.
- Dynamic UI Updates:
    - Fragments can be dynamically added, removed, or replaced at runtime, allowing for dynamic UI updates and transitions.
    - Developers can use FragmentTransactions to perform these operations and animate the transitions between fragments.
- Support for Different Screen Sizes:
    - Fragments are particularly useful for creating adaptive UIs that can adjust to different screen sizes and orientations.
    - They allow developers to create layouts optimized for both phones and tablets by reusing fragments in different configurations.

Overall, fragments are a powerful and flexible tool for building Android user interfaces. They provide a modular and reusable way to create UI components, facilitate communication between UI elements, and support dynamic UI updates for a variety of screen sizes and configurations.

Fragment example

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fragment Example"
         />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Load Fragment 1"
        android:id="@+id/bt1"
        android:layout_margin="10dp"
        />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Load Fragment 2"
        android:id="@+id/bt2"
        android:layout_margin="10dp"
        />
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="10dp"
        android:id="@+id/frame"
        android:background="#098"/>
</LinearLayout>
</layout>
```

MainActivity.kt

```
package com.example.fragmentex

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.databinding.DataBindingUtil
import com.example.fragmentex.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
  var binding: ActivityMainBinding?=null
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding=
DataBindingUtil.setContentView(this,R.layout.activity_main)

    binding?.bt1?.setOnClickListener{
      var fragmentManager=supportFragmentManager
      var fragmentTransaction=fragmentManager.beginTransaction()
      var firstfrag=FirstFrag()

      fragmentTransaction.replace(R.id.frame,firstfrag)
      fragmentTransaction.commit()
    }

    binding?.bt2?.setOnClickListener{
      var fragmentManager=supportFragmentManager
      var fragmentTransaction=fragmentManager.beginTransaction()
      var secfrag=SecFrag()

      fragmentTransaction.replace(R.id.frame,secfrag)
      fragmentTransaction.commit()
    }
  }
}
```

Fragment_first.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".FirstFrag">

  <!-- TODO: Update blank fragment layout -->
  <TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:text="@string/hello_blank_fragment" />

</FrameLayout>
```

FirstFrag.kt

```
package com.example.fragmentex

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup

// TODO: Rename parameter arguments, choose names that match
// the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
private const val ARG_PARAM1 = "param1"
private const val ARG_PARAM2 = "param2"

/**
* A simple [Fragment] subclass.
* Use the [FirstFrag.newInstance] factory method to
* create an instance of this fragment.
*/
class FirstFrag : Fragment() {
  // TODO: Rename and change types of parameters
  private var param1: String? = null
  private var param2: String? = null

  override fun onCreate(savedInstanceState: Bundle?) {
     super.onCreate(savedInstanceState)
     arguments?.let {
        param1 = it.getString(ARG_PARAM1)
        param2 = it.getString(ARG_PARAM2)
     }
  }

  override fun onCreateView(
     inflater: LayoutInflater, container: ViewGroup?,
     savedInstanceState: Bundle?
  ): View? {
     // Inflate the layout for this fragment
     return inflater.inflate(R.layout.fragment_first, container, false)
  }

  companion object {
     /**
      * Use this factory method to create a new instance of
```

```
         * this fragment using the provided parameters.
         *
         * @param param1 Parameter 1.
         * @param param2 Parameter 2.
         * @return A new instance of fragment FirstFrag.
         */
        // TODO: Rename and change types and number of parameters
        @JvmStatic
        fun newInstance(param1: String, param2: String) =
            FirstFrag().apply {
                arguments = Bundle().apply {
                    putString(ARG_PARAM1, param1)
                    putString(ARG_PARAM2, param2)
                }
            }
    }
}
Fragment_second.xml
<?xml version="1.0" encoding="utf-8"?>
<layout>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecFrag"
    android:orientation="vertical">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="First fragment" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/bt3"
        android:text="Move to Activity"/>

</LinearLayout>
</layout>
```

Secondfrag.kt

```
package com.example.fragmentex

import android.content.Intent
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import androidx.databinding.DataBindingUtil
import com.example.fragmentex.databinding.FragmentSecBinding


class SecFrag : Fragment() {

  // var bt: Button?=null
  var binding: FragmentSecBinding?=null
   override fun onCreateView(
       inflater: LayoutInflater, container: ViewGroup?,
       savedInstanceState: Bundle?
   ): View? {
       binding= DataBindingUtil.inflate(inflater,
           R.layout.fragment_sec, container, false)
  //var view= inflater.inflate(R.layout.fragment_sec, container, false)

       //bt=view.findViewById(R.id.bt3)

      binding?.bt3?.setOnClickListener{
           var intent= Intent(context,HomePage::class.java)
           startActivity(intent)
       }

       //return view
       return binding?.root
   }


}
```

HomePage.kt

```
package com.example.fragmentex
```

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class HomePage : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_home_page)
   }
}
```

Activity_home.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".HomePage">

</androidx.constraintlayout.widget.ConstraintLayout>
```

Androidmanifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools">

   <application
      android:allowBackup="true"
      android:dataExtractionRules="@xml/data_extraction_rules"
      android:fullBackupContent="@xml/backup_rules"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:roundIcon="@mipmap/ic_launcher_round"
      android:supportsRtl="true"
      android:theme="@style/Theme.Fragmentex"
      tools:targetApi="31">
      <activity
         android:name=".HomePage"
         android:exported="false" />
      <activity
         android:name=".MainActivity"
         android:exported="true">
         <intent-filter>
```

```
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

SQLite Databases

SQLite is a popular choice for developers due to its simplicity, reliability, and efficiency. Here are some key aspects:

- Serverless Architecture: Unlike traditional database management systems like MySQL or PostgreSQL, SQLite does not require a separate server process. It operates directly on the database file. This makes it easy to set up and use since there's no need to configure or manage a database server.
- Zero Configuration: SQLite doesn't require any configuration or administration. You can start using it by simply including the SQLite library in your application and accessing the database file.
- Self-Contained: The entire SQLite database is contained in a single disk file, making it easy to distribute and manage. This file can be easily copied or moved around without any special tools or procedures.
- Transactional: SQLite supports ACID (Atomicity, Consistency, Isolation, Durability) transactions, ensuring data integrity and reliability. This means you can execute multiple operations as part of a single transaction, and SQLite will ensure that either all operations succeed or none of them do.
- Cross-Platform: SQLite is cross-platform and runs on various operating systems including Windows, macOS, Linux, and mobile operating systems like Android and iOS. This makes it suitable for a wide range of applications and devices.
- Embeddable: SQLite is designed to be embedded directly into applications. It's a lightweight library that can be linked statically or dynamically, allowing developers to incorporate a full-featured SQL database engine into their applications with minimal overhead.
- High Performance: Despite its small size and simplicity, SQLite is highly efficient and performs well in most scenarios. It's optimized

for read-heavy workloads and can handle large datasets with ease.

Example program

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity"
   android:orientation="vertical">

   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Sqlite Example"
        />
   <EditText
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:hint="Name"
       android:id="@+id/name"
       />
   <EditText
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:hint="location"
       android:id="@+id/location"
       />
   <EditText
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:hint="designantion"
       android:id="@+id/designation"
```

```
        />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Save"
        android:id="@+id/save"
        />
</LinearLayout>
</layout>
MainActivity.kt
package com.example.sqliteexa

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast
import androidx.databinding.DataBindingUtil
import com.example.sqliteexa.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
    var binding: ActivityMainBinding?=null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding=
DataBindingUtil.setContentView(this,R.layout.activity_main)

        binding?.save?.setOnClickListener {

            var username=binding?.name?.text.toString().trim()
            var location=binding?.location?.text.toString().trim()
            var designation=binding?.designation?.text.toString().trim()

            var dbHandler=DbHandler(this@MainActivity)

            dbHandler.insertUserDetails(username,location,designation)

            Toast.makeText(this@MainActivity," Detals
saved",Toast.LENGTH_LONG).show()

            var intent= Intent(this@MainActivity,DetailActivity::class.java)
            startActivity(intent)
```

```
      }
   }
}
```

Dbhandler.kt

```
package com.example.sqliteexa

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class DbHandler(context: Context) :
SQLiteOpenHelper(context,DB_NAME,null, DB_VERSION){

   companion object{
       private const val  DB_NAME="userdb"
       private  const val DB_VERSION=1
       private const val Table_Users="userdetails"
       private const val KEY_ID="id"
       private const val KEY_NAME="name"
       private const val KEY_LOC="location"
       private const val KEY_DES="designation"

   }
   override fun onCreate(db: SQLiteDatabase?) {
     //  var CREATE_TABLE= ("CREATE TABLE " + Table_Users + "( " +
KEY_ID + "INTEGER PRIMARY KEY AUTOINCREMENT, " +
       //        KEY_NAME + "TEXT, " + KEY_LOC + "TEXT, "+ KEY_DES +
"TEXT" + ")")
       //db?.execSQL(CREATE_TABLE)



       val CREATE_TABLE = ("CREATE TABLE " + Table_Users + "("
               + KEY_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"
+ KEY_NAME + " TEXT,"
               + KEY_LOC + " TEXT,"
               + KEY_DES + " TEXT" + ")")
       db?.execSQL(CREATE_TABLE)

   }
```

```
  override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int,
newVersion: Int) {
      db?.execSQL("DROP TABLE IF EXISTS " + Table_Users)

   //   db.execSQL("DROP TABLE IF EXISTS " + TABLE_Users
      onCreate(db)
  }

  fun insertUserDetails(name: String?, location:
String?,designation:String?){
      var db=this.writableDatabase
      var contentValues=ContentValues()
      contentValues.put(KEY_NAME,name)
      contentValues.put(KEY_LOC,location)
      contentValues.put(KEY_DES,designation)

      var newRowId=db.insert(Table_Users,null,contentValues)

  }

  @SuppressLint("Range")
  fun GetUser(): ArrayList<HashMap<String,String>>{
      var db=this.writableDatabase
      var userlist=ArrayList<HashMap<String,String>>()
      var query="SELECT  name, location, designation FROM "+
Table_Users
      var cursor=db.rawQuery(query,null)
      while (cursor.moveToNext()){
          var user=HashMap<String,String>()
          user["name"]=cursor.getString(cursor.getColumnIndex(KEY_NAM
E))
          user["location"]=cursor.getString(cursor.getColumnIndex(KEY_L
OC))
          user["designation"]=cursor.getString(cursor.getColumnIndex(KEY
_DES))
          userlist.add(user)
      }
      return userlist
  }

  fun deleteUser(userid: Int){
      var db=this.writableDatabase
      db.delete(Table_Users, KEY_ID + " =?", arrayOf(userid.toString()))
      db.close()
  }

  fun updateudetails(location: String?, desingnation: String?, id:Int):
          Int{
      var db=this.writableDatabase
      var contentValues=ContentValues()

      contentValues.put(KEY_LOC,location)
      contentValues.put(KEY_DES,desingnation)

      return db.update(Table_Users,contentValues,"$KEY_ID=?",
arrayOf(id.toString()))


  }

  @SuppressLint("Range")
  fun GetUserByUserid(userid: Int):
ArrayList<HashMap<String,String>>{
      var db=this.writableDatabase
      var userlist=ArrayList<HashMap<String,String>>()
      var query="SELECT  name, location, designation FROM "+
Table_Users
      var cursor=db.query(Table_Users, arrayOf(KEY_NAME, KEY_LOC,
KEY_DES),
          KEY_ID + "=?",
          arrayOf(userid.toString()),
          null,
          null,
          null,
          null
      )
      if (cursor.moveToNext()){
          var user=HashMap<String,String>()
          user["name"]=cursor.getString(cursor.getColumnIndex(KEY_NAM
E))
          user["location"]=cursor.getString(cursor.getColumnIndex(KEY_L
OC))
```

```
            user["designation"]=cursor.getString(cursor.getColumnIndex(KEY
_DES))
            userlist.add(user)
        }
        return userlist
    }
}
```

Activity_detail.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".DetailActivity"
   android:orientation="vertical">

   <TextView
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:text="Saved Data"/>

   <ListView
       android:layout_width="match_parent"
       android:layout_height="wrap_content"

       android:id="@+id/list"/>

</LinearLayout>
</layout>
```

DetailActivity.kt

```
package com.example.sqliteexa

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.ListAdapter
import android.widget.SimpleAdapter
import androidx.databinding.DataBindingUtil
import com.example.sqliteexa.databinding.ActivityDetailBinding
```

```
class DetailActivity : AppCompatActivity() {
   var binding: ActivityDetailBinding?=null
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      binding=
DataBindingUtil.setContentView(this,R.layout.activity_detail)

      var dbHandler=DbHandler(this@DetailActivity)

      var userlist=dbHandler.GetUser()

      var simpleadapater :ListAdapter=
SimpleAdapter(this@DetailActivity,
         userlist,
         R.layout.custom,
         arrayOf("name","designation","location"),
         intArrayOf(R.id.name,R.id.designation,R.id.location)

      )
      binding?.list?.adapter=simpleadapater
   }
}
```

Custom,xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">

   <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/name"/>

   <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/location"/>
   <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
```

```
    android:id="@+id/designation"/>

</LinearLayout>
```

ROOMDATABASE

Room is an Android Library that provides an abstraction layer over SQLite to allow for more robust database access while harnessing the full power of SQLite. It simplifies database interactions and provides compile-time checks, better performance, and increased readability of code compared to using SQLite directly.
Here are some key features of Room:

- SQLite Abstraction: Room provides a high-level abstraction layer over SQLite, allowing developers to work with a more intuitive API rather than directly dealing with raw SQL queries.
- Compile-time Checks: Room performs compile-time checks of SQLite queries, which helps catch errors early in the development process. This includes syntax errors and type mismatches in queries.
- Annotation-Based: Room uses annotations to define the database schema and map Java/Kotlin objects to database tables. This reduces boilerplate code and makes database operations more concise and readable.
- Entity-Relationship Mapping: Room supports defining entity classes to represent database tables and defining relationships between entities. This allows for easy navigation of object relationships within the database.
- LiveData and RxJava Support: Room seamlessly integrates with LiveData and RxJava, allowing for reactive updates to database queries and easy integration with Android's lifecycle-aware components.
- Database Migrations: Room simplifies the process of database migrations by providing built-in support for schema migrations. Developers can define migration strategies to update the database schema as needed without losing data.
- Testing Support: Room provides utilities for testing database interactions, including in-memory database instances and transaction annotations for easy rollback of test data.

Overall, Room simplifies the process of working with SQLite databases on Android by providing a higher-level abstraction, compile-time checks, and integration with modern Android architecture components. It's widely used by Android developers to build robust and efficient database-driven applications.

Example of room database

Build.gradle

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}


android {
    compileSdk 31


    defaultConfig {
        applicationId "com.example.roomexample"
        minSdk 21
        targetSdk 31
        versionCode 1
        versionName "1.0"


        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }


    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }


    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }


    kotlinOptions {
        jvmTarget = '1.8'
    }
}


dependencies {
    implementation 'androidx.appcompat:appcompat:1.4.0'
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'


    // Room components
    implementation 'androidx.room:room-runtime:2.4.0'
    implementation 'androidx.room:room-ktx:2.4.0'
    kapt 'androidx.room:room-compiler:2.4.0'


    // Coroutine support
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.5.2'


    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

MainActivity.kt

```
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.room.Room
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch


class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


        val userDao = MyAppDatabase.getInstance(applicationContext).userDao()


        // Inserting a user
        GlobalScope.launch(Dispatchers.IO) {
            userDao.insertUser(User(name = "John", age = 30))
        }
        // Retrieving all users
        GlobalScope.launch(Dispatchers.IO) {
            val users = userDao.getAllUsers()
            // Do something with the users
        }
    }
}
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Room Example"
        android:textSize="24sp"
        android:layout_centerInParent="true"/>


</RelativeLayout>
```

user.kt

```
import androidx.room.Entity
import androidx.room.PrimaryKey


@Entity(tableName = "users")
data class User(
    @PrimaryKey(autoGenerate = true)
    val id: Long = 0,
    val name: String,
    val age: Int
)
```

MyappDatabse.kt

```
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase


@Database(entities = [User::class], version = 1)
abstract class MyAppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao


    companion object {
        @Volatile
        private var INSTANCE: MyAppDatabase? = null


        fun getInstance(context: Context): MyAppDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    MyAppDatabase::class.java,
                    "myapp_database"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

ANDROID MATERIAL DESIGN

Android Material Design is a design language developed by Google in 2014 to provide a unified system for designing user interfaces across different platforms and devices. It emphasizes principles such as bold colors, responsive animations, and depth effects to create visually appealing and intuitive user experiences. Here are some key aspects of Android Material Design:

- Material:
    - Material is the metaphor used in Material Design. It's inspired by the physical world and its textures, with elements like paper and ink. Material surfaces reimagine the mediums of paper and ink.
- Components:
    - Material Design provides a comprehensive set of UI components such as buttons, cards, text fields, dialogs, and

more. These components are designed to be consistent, flexible, and easy to use across different devices and screen sizes.

- Layouts:
    - Material Design offers flexible layout patterns such as the CoordinatorLayout, AppBarLayout, and BottomAppBar, which help in creating dynamic and responsive UIs. These layouts are optimized for touch interactions and support various scrolling behaviors.
- Typography:
    - Typography plays a crucial role in Material Design. It emphasizes readability and hierarchy, with guidelines for font styles, sizes, and spacing. The use of typography helps in creating clear and visually appealing interfaces.
- Color:
    - Material Design encourages the use of vibrant colors and bold contrasts to create visually striking designs. It provides a color palette with predefined primary and accent colors, along with guidelines for color usage and accessibility.
- Motion:
    - Motion is an essential aspect of Material Design, as it adds depth and context to user interactions. Guidelines for motion include principles such as meaningful transitions, responsive animations, and smooth scrolling effects.
- Iconography:
    - Material Design includes a set of consistent and scalable icons that can be used across different platforms and resolutions. These icons follow specific design guidelines to ensure clarity and recognition.
- Accessibility:
    - Accessibility is a core principle of Material Design. It provides guidelines for designing inclusive and accessible user interfaces, including recommendations for text contrast, focus indicators, and screen reader compatibility.
- Adaptive Design:
    - Material Design promotes adaptive design principles, allowing UIs to adapt seamlessly to different screen sizes, orientations, and input methods. This ensures a consistent and intuitive user experience across various devices and platforms.

Overall, Android Material Design provides a comprehensive set of guidelines, components, and tools for designing modern and visually appealing user interfaces that are intuitive, accessible, and responsive across different devices and screen sizes. It has become the standard design language for Android app development and is widely adopted by developers and designers alike.

HOW TO UPLOAD APPS IN PLAYSTORE

Uploading an app to the Google Play Store involves several steps. Here's a detailed guide on how to do it:

1. PREREQUISITES:

- Google Play Developer Account: You need a Google Play Developer account to publish apps on the Play Store. You can sign up for an account on the Google Play Console.
- App Package (APK): Your app must be packaged as an APK file. This file contains the compiled code and resources of your app.
- App Icons, Screenshots, and Graphics: Prepare high-quality icons, screenshots, feature graphics, and promotional images for your app.

2. PREPARE YOUR APP FOR RELEASE:

- Optimize and Test Your App: Make sure your app is optimized for performance, stability, and user experience. Test it on various devices and screen sizes to ensure compatibility.
- Set App Permissions: Define the necessary permissions for your app in the AndroidManifest.xml file.
- Generate Signed APK: Sign your app with a digital certificate to ensure its authenticity. You can generate a signed APK using Android Studio or the command line.

3. CREATE A NEW APP LISTING ON GOOGLE PLAY CONSOLE:

- Log in to Google Play Console: Go to the Google Play Console and sign in with your developer account.

- Click on "Create app": Provide the required details such as the default language, app title, and package name. Click "Create."

4. Fill in the App Details:

- Store Listing: Fill in the app details including the app title, short and full descriptions, and other metadata such as category, content rating, and contact details.
- Upload Images and Multimedia: Upload high-quality app icons, screenshots, feature graphics, and promotional videos.
- Provide Pricing and Distribution Settings: Set the pricing, distribution countries, and distribution options for your app.

5. Upload Your App Bundle or APK:

- Prepare Your Release: Go to the "App releases" section and click "Manage."
- Upload Your APK: Upload your signed APK or Android App Bundle (AAB) file. If you're using an AAB, Google Play will generate APKs optimized for different device configurations.
- Fill in Release Details: Provide release notes, select a release track (e.g., production, beta, alpha), and set the rollout percentage if necessary.

6. Review and Publish:

- Review Your App Listing: Review all the details and content of your app listing to ensure accuracy and compliance with Google Play policies.
- Submit for Review: Once you're satisfied, click "Review" or "Submit for review" to submit your app for review by Google Play.
- Wait for Approval: Google Play will review your app to ensure it meets their policies and guidelines. This process may take a few hours to several days.
- App Goes Live: Once your app is approved, it will be published on the Google Play Store and will be available for users to download and install.

7. Manage Your App Listing:

- Monitor Performance: Use the Google Play Console to monitor your app's performance, user feedback, and reviews.

- Update Your App: Regularly update your app with bug fixes, new features, and improvements based on user feedback and analytics.

That's a comprehensive overview of the process for uploading an app to the Google Play Store. Make sure to follow Google's guidelines and best practices throughout the process to ensure a smooth publishing experience.

ADMOBACCOUNT

To add ads to your Android app, you typically integrate a third-party ad network SDK into your app and then use their APIs to display ads. Here's a general guide on how to do it using Google AdMob, one of the most popular ad networks for Android apps:

1. SET UP GOOGLE ADMOB:

- Create an AdMob Account: Sign up for an account on the AdMob website.
- Create an Ad Unit: Once logged in, create an ad unit for your app. Choose the ad format (e.g., banner, interstitial, rewarded) and specify the details.

2. INTEGRATE THE ADMOB SDK INTO YOUR APP:

- Add the AdMob SDK Dependency: Add the AdMob SDK dependency to your app's build.gradle file.

```
implementation 'com.google.android.gms:play-services-ads:20.7.0'
```

Initialize AdMob: Initialize the AdMob SDK in your app's Application class or in the onCreate() method of your main activity.

```
class MyApp : Application() {
    override fun onCreate() {
        super.onCreate()
        MobileAds.initialize(this)
    }
}
```

ADD AD VIEWS TO YOUR LAYOUTS:

- Banner Ads: Add a AdView element to your layout XML file where you want the banner ad to appear.

```
<com.google.android.gms.ads.AdView
    android:id="@+id/adView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:adSize="BANNER"
    app:adUnitId="@string/banner_ad_unit_id"/>
```

- Interstitial Ads: Interstitial ads are full-screen ads that cover the interface of an app. You need to load and display them programmatically at appropriate times (e.g., between game levels).
- Rewarded Ads: Rewarded ads are ads that users can choose to watch in exchange for in-app rewards. You also need to load and display them programmatically.

4. LOAD AND DISPLAY ADS IN YOUR ACTIVITY/FRAGMENT:

- Banner Ads: Load and display banner ads in your activity or fragment.

```
val adView: AdView = findViewById(R.id.adView)
val adRequest = AdRequest.Builder().build()
adView.loadAd(adRequest)
```

Interstitial Ads: Load and display interstitial ads when appropriate (e.g., after a game level).

```
val interstitialAd = InterstitialAd(this)
interstitialAd.adUnitId = getString(R.string.interstitial_ad_unit_id)
interstitialAd.loadAd(AdRequest.Builder().build())
// Display the ad when ready
if (interstitialAd.isLoaded) {
    interstitialAd.show()
}
```

Rewarded Ads: Load and display rewarded ads, and listen for events such as user watching the ad.

```
val rewardedAd = RewardedAd(this,
getString(R.string.rewarded_ad_unit_id))
val adCallback = object : RewardedAdCallback() {
    override fun onRewardedAdClosed() {
        // Ad closed, proceed with reward if applicable
    }
}
rewardedAd.fullScreenContentCallback = adCallback
rewardedAd.loadAd(AdRequest.Builder().build())
// Display the ad when ready
rewardedAd.show(this, adCallback)
```

HANDLE AD EVENTS AND REWARDS:

- Implement listeners/callbacks for ad events such as ad loaded, ad closed, ad clicked, etc.
- Handle reward distribution if you're using rewarded ads.

6. TEST YOUR ADS:

- Use test ad units during development to avoid serving real ads to your test users.
- Test various ad formats and placements to ensure they display correctly and do not interfere with the user experience.

7. PUBLISH YOUR APP:

- Once you've integrated ads into your app and tested them thoroughly, publish your app on the Google Play Store.

That's a general overview of how to add ads to your Android app using Google AdMob. Remember to follow AdMob's policies and guidelines to ensure compliance and maximize ad revenue while providing a good user experience.

GOOGLE MAPS

To display maps in your Android app using the Google Maps API, you need to obtain an API key from the Google Cloud Console. Here's how you can generate a Google Maps API key:

1. Create a Google Cloud Project:

- Go to the Google Cloud Console and sign in with your Google account.
- Create a new project by clicking on the "Select a project" dropdown at the top of the page, then clicking on "New Project". Follow the prompts to create a new project.

2. Enable the Google Maps API:

- Once your project is created, navigate to the "APIs & Services" > "Library" section in the left sidebar.
- Search for "Google Maps Android API" and click on it.
- Click the "Enable" button to enable the API for your project.

3. Create an API Key:

- After enabling the Google Maps Android API, navigate to the "APIs & Services" > "Credentials" section in the left sidebar.
- Click on the "Create credentials" dropdown and select "API key".
- A new API key will be generated for you. Copy this API key.

4. Restrict Your API Key (Optional but Recommended):

- To improve security and manage usage, it's recommended to restrict your API key. You can restrict it based on:
  - Android apps: By providing the package name and SHA-1 fingerprint of your app's signing certificate.
  - HTTP referrers: By specifying the URLs of the websites that are allowed to use the API key.
- You can set these restrictions by clicking on the API key you just created, then selecting "Restrict key" and following the instructions.

5. Add the API Key to Your Android Project:

- Open your Android project in Android Studio.
- Navigate to your AndroidManifest.xml file and add the following <meta-data> element inside the <application> element, replacing "YOUR_API_KEY" with your actual API key:

```
<application>
   <!-- Other application elements -->
```

```
    <meta-data
        android:name="com.google.android.geo.API_KEY"
        android:value="YOUR_API_KEY" />
</application>
```

Ensure Required Permissions:

- Ensure that your app has the necessary permissions to access the internet and access the device's location if your app requires it.

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

7. Testing:

- Run your app on a physical device or emulator that has Google Play services installed.
- If everything is set up correctly, you should be able to display maps in your app using the Google Maps API.

That's it! You've successfully generated a Google Maps API key and integrated it into your Android app. Now you can use the key to display maps and utilize other features provided by the Google Maps API.

Generating SHA1 key

To generate an SHA-1 key for your app's signing certificate in Android Studio, follow these steps:

1. Open Your Project in Android Studio:

- Open your Android project in Android Studio if it's not already open.

2. Navigate to Gradle Tab:

- On the right side of the Android Studio window, you'll find a tab called "Gradle" (usually located near the top-right corner). Click on it to open the Gradle view.

3. Expand Your App:

- In the Gradle view, expand your project's folder, then expand the "Tasks" folder, and finally expand the "android" folder.

4. Double-click on "signingReport":

- Inside the "android" folder, you'll find a task named "signingReport". Double-click on it.

5. View SHA-1 Key:

- After double-clicking on "signingReport", Android Studio will run the task and display the SHA-1 key for your app's signing certificate in the "Run" tab at the bottom of the window.

6. Locate SHA-1 Key:

- Look for the line that starts with "SHA1:". The SHA-1 key will be displayed next to it.

Example:

ruby

```
Variant: debug
Config: debug
Store: /path/to/your/debug.keystore
Alias: androiddebugkey
MD5: A1:B2:C3:D4:E5:F6:G7:H8:I9:J0:K1:L2:M3:N4:O5:P6
SHA1: A1B2C3D4E5F6G7H8I9J0K1L2M3N4O5P6
SHA-256: A1B2C3D4E5F6G7H8I9J0K1L2M3N4O5P6
Valid until: Wednesday, December 31, 2049
```

- Make sure you are using the correct keystore file (debug or release) depending on whether you are building your app for development or production.
- The SHA-1 key is required for configuring API keys for services like Google Maps, Firebase, etc.

That's it! You've successfully generated the SHA-1 key for your app's signing certificate in Android Studio. You can now use this key to configure API access for various services.