**Activity:**

An activity is the entry point of an app for interacting with the user.

It represents a single screen with a user interface.

For example, an email app might have one activity that shows a list of new emails.

another activity to compose an email, and another activity for reading emails.

Although the activities work together to form a better user experience in the email app

An activity goes through a number of states over the course of its lifetime

a series of callbacks is used to handle transitions between states.

the callbacks are as follows:

* onCreate()
  + You must implement this callback, which fires when the system creates your activity.
  + You should initialize the essential components of your activity inside this callback
  + For example, your app should create views and bind data to lists here.
  + Most importantly, this is where you must call setContentView() to define the layout for the activity'suser interface.
  + When onCreate() finishes, the next callback is always onStart().
* onStart()
* As onCreate() exits, the activity enters the Started state, and the activity becomes visible to the user.
* This callback contains what amounts to the activity’s final preparations for coming to the foregroundand becoming interactive.
* When onStart() finishes, the next callback is always onResume().
* onResume()
* At this point, the activity is at the top of the activity stack
* It captures all user input.
* The onPause() callback always follows onResume().
* onPause()
* The system calls onPause() when the activity loses focus and enters a Paused state. - This state occurs when, for example, the user taps the Back or Recents button.
* When the system calls onPause() for your activity, it technically means your activity is still partially visible, but most often is an indication that the user is leaving the activity, and the activity will soon enter the Stopped or Resumed state.
* Once onPause() finishes executing the next callback is either onStop() or onResume(), depending on what happens after the activity enters the Paused state.
* onStop()
* The system calls onStop() when the activity is no longer visible to the user.
* This may happen because the activity is being destroyed or a new activity is starting and is covering the stopped activity.
* In all of these cases, the stopped activity is no longer visible at all.
* The next callback that the system calls is either onRestart(), if the activity is coming back to interact with the user, or by onDestroy() if this activity is completely terminating.
* onRestart()
* The system invokes this callback when an activity in the Stopped state is about to restart. onRestart() restores the state of the activity from the time that it was stopped.
* This callback is always followed by onStart().
* onDestroy()
* The system invokes this callback before an activity is destroyed.
* This callback is the final one that the activity receives.
* onDestroy() is usually implemented to ensure that all of an activity’s resources are released when the activity, or the process containing it, is destroyed.

Logcat:

The Logcat window in Android Studio helps you debug your app by displaying logs from your device in real time—for example, messages that you added to your app with the Log class, messages from services that run on Android, or system messages, such as when a garbage collection occurs. When an app throws an exception, Logcat shows a message followed by the associated stack trace containing links to the line of code. Each log has a date, timestamp, process and thread ID, tag, package name, priority, and message associated with it. Different tags have a unique colour that helps identify the type of log. Each log entry has a priority of FATAL, ERROR, WARNING, INFO, DEBUG, or VERBOSE. we can Query logs using key-value search for example we can use

* tag: Matches against the tag field of the log entry.
* package: Matches against the package name of the logging app.
* message: Matches against the message part of the log entry.
* level: Matches the specified or higher severe log level–for example, DEBUG.

Log:

* API for sending log output.
* Generally, we should use the Log.v(), Log.d(), Log.i(), Log.w(), and Log.e() methods to write logs.
* we can then view the logs in logcat.
* Log.d : Send a DEBUG log message.
* Log.e : Send an ERROR log message.
* Log.i : Send a INFO log message.
* Log.v : Send a VERBOSE log message.
* Log.w : Send a WARN log message.

View, View Group and Layout:

A View is defined as the user interface which is used to create interactive UI components such as TextView, ImageView, EditText, RadioButton, etc. It is also responsible for event handling A ViewGroup act as a base class for that hold other Views or ViewGroups and is used to define the properties. These view groups are generally called layouts. Types of Android Layout:

* Linear Layout: It is a ViewGroup subclass, used to provide child View elements one by one in a particular direction either horizontally or vertically based on the orientation property. The default orientation is horizontal.
* Relative Layout: It is a ViewGroup subclass, used to specify the position of child View elements relative to each other like (A to the right of B) or relative to the parent (fix to the top of the parent).
* Constraint Layout: It is a ViewGroup subclass, used to specify the position of layout constraints for every child View relative to other views present. A ConstraintLayout is similar to a RelativeLayout, but having more power. used mostlty with drag and drop UI generation.
* Frame Layout: It is a ViewGroup subclass, used to specify the position of View elements it contains on the top of each other to display only a single View inside the FrameLayout.
* Table Layout: TableLayout is a ViewGroup subclass, used to display the child View elements in rows and columns.

Pixel in Android:

In Android, dp (density-independent pixels), sp (scale-independent pixels), and px (pixels), are units of measurement used to define dimensions in the layout and design of Android applications. Each of these units serves a specific purpose, particularly related to screen density and text size.

**dp** (Density-independent Pixels):

dp is a unit of measurement that is density-independent.

It is recommended to specify dimensions in layouts to ensure consistent visual size across different screen densities.

1 dp is equivalent to one physical pixel on a medium-density screen (160 dpi).

The actual size of a dp is adjusted based on the screen’s density.

**sp** (Scale-independent Pixels):

sp is similar to dp, but it is specifically used for defining text size.

It takes into account the user’s preferred text size setting in the device’s system settings.

1 sp is the same as 1 dp, but it also considers the user’s font size preference.

**px**(Pixels):

px is the most basic unit and represents a single pixel on the screen.

It is not recommended to specify dimensions in layouts due to its lack of density independence.

**Text View:**

It is a view used to display the textual data on screen We can set the text dynamically on this view from our program.

<TextView

android:textSize="20dp"

android:text="Phase-2 Hinjewadi"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

**Edit Text:** It is a view used to take the input from user.

<EditText

android:hint="Enter email"

android:inputType="textEmailAddress"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

**Button:**

It is a view that creates a clickable button on screen. It provides the onclick event that we can handle.

<Button

android:id="@+id/btnSave"

android:text="Save"

android:onClick="save"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

Button btnSave = findViewById(R.id.btnSave);

btnSave.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Log.e(tag,"Save Button is Clicked");

}

});

// OR

public void save(View view) {

Log.e(tag, "save() Button is Clicked");

}

**Checkbox:**

It is a view that creates a checkbox

We can select multiple checkbox at same time

to add multpile ckeckbox add multiple checkBox Views

The view selection can be monitored by the method isSelected().

<CheckBox

android:id="@+id/checkboxC"

android:text="C"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<CheckBox

android:id="@+id/checkboxCPP"

android:text="CPP"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

**Radio Button:**

It is a view that creates a radioButton We can select only one radioButton at a time RadioButtons should be created inside a radioGroup to have only 1 selection The view selection can be monitored by the method isSelected().

<RadioGroup

android:orientation="horizontal"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<RadioButton

android:id="@+id/radioMale"

android:text="Male"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<RadioButton

android:id="@+id/radioFemale"

android:text="Female"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

</RadioGroup>

**Spinner:**

It is a view that creates a Dropdown Spinners provide a quick way to select one value from a set. - In the default state, a spinner shows its currently selected value. Tapping the spinner displays a menu showing all other values the user can select. To add a spinner to your layout.

<Spinner

android:id="@+id/spinnerCourses"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

The choices you provide for the spinner can come from any source, but you must provide them through a SpinnerAdapter, such as an ArrayAdapter.

List<String> courses;

spinnerCourse = findViewById(R.id.spinnerCourse);

courses = new ArrayList<>();

courses.add("PG-DAC");

courses.add("PG-DMC");

courses.add("PG-DBDA");

ArrayAdapter arrayAdapter = new ArrayAdapter(this,

android.R.layout.simple\_list\_item\_1,courses);

spinnerCourse.setAdapter(arrayAdapter);

When the user selects an item from the spinner's menu, the Spinner object receives an on-item selected event. To define the selection event handler for a spinner, implement the AdapterView.OnItemSelectedListener interface and the corresponding onItemSelected() callback method.

spinnerCourse.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {

@Override

public void onItemSelected(AdapterView<?> parent, View view int position, long id)

{

Log.e("SpinnerActivity",courses.get(position));

}

@Override

public void onNothingSelected(AdapterView<?> parent) {}

});

}

**Array Adapter:**

It provides views for an Adapter View Returns a view for each object in a collection of data objects you provide, and can be used with list-based user interface widgets such as ListView or Spinner. By default, the array adapter creates a view by calling Object.toString() on each data object in the collection you provide, and places the result in a TextView.

**Context:**

This is an abstract class whose implementation is provided by the Android system. It provides global information about an application environment. It is used for application-level operations such as launching activities, broadcasting and receiving intents, etc. It allows us to access files and other resources such as pictures, Activities, Fragments and Services. The Context Class provides access to several resources and services that are needed to build an Android application.

There are two types of contexts:

* Application:
* getApplicationContext() will give the application context
* It is related with the life cycle of application
* Application context is tied to the application itself and remains alive as long as the application is running.
* This means that it can be used across multiple activities, and is useful for accessing global resources and classes that are not tied to any specific activity, such as shared preferences or database helpers.
* Activity:
* getContext() will give the Activity context
* It is related with the life cycle of activity

* An activity context should be used when you need access to resources or classes that are tied to a specific activity, such as views or resources that are specific to that activity.

Both "Activity" and "Application" are subclasses of the "Context"class, which provides access to application-specific resources and classes.

**Toast:**

* A toast provides simple feedback about an operation in a small popup.
* It only fills the amount of space required for the message and the current activity remains visible and interactive.
* Toasts automatically disappear after a timeout.
* If your app targets Android 12 (API level 31) or higher, its toast is limited to two lines of text and shows the application icon next to the text.
* Be aware that the line length of this text varies by screen size, so it's good to make the text as short as possible.
* To Instantiate a Toast object, Use the makeText() method, which takes the following parameters:

1. The activity Context.

2. The text that should appear to the user.

3. The duration that the toast should remain on the screen.

* The makeText() method returns a properly initialized Toast object.
* To display the toast, call the show() method, as demonstrated in the following example:

Toast.makeText(this, "Hello from toast", Toast.LENGTH\_SHORT).show();

* If your app is in the foreground, consider using a snackbar instead of using a toast.
* Snackbars include user-actionable options, which can provide a better app experience.

**SnackBar:**

* It is similar to toast but with some advanced options.
* The snackbar component serves as a brief notification that appears at the bottom of the screen.
* It provides feedback about an operation or action without interrupting the user experience.
* Snackbars disappear after a few seconds.
* The user can also dismiss them with an action, such as tapping a button.
* Consider these three use cases where you might use a snack bar:

1. Action Confirmation: After a user deletes an email or message, a snackbar appears to confirm the action and offer an "Undo" option.

2. Network Status: When the app loses its internet connection, a snackbar pops up to note that it is now offline.

3. Data Submission: Upon successfully submitting a form or updating settings, a snackbar notes that the change has saved successfully.

**ImageView:**

It is a view that can hold the images

It Displays image resources, for example Bitmap or Drawable resources.

<ImageView

android:src="@drawable/my\_image"

android:layout\_width="150dp"

android:layout\_height="150dp"/>

**ScrollView:**

* A view group that allows the view hierarchy placed within it to be scrolled.
* Scroll view may have only one direct child placed within it.
* To add multiple views within the scroll view, make the direct child you add a view group, for example
* LinearLayout, and place additional views within that LinearLayout.
* Scroll view supports vertical scrolling only. For horizontal scrolling, use HorizontalScrollView instead.
* Never add a RecyclerView or ListView to a scroll view, doing so results in poor user interface
* performance and a poor user experience.

**Intent:**

* It is a messaging object that is used to communicate between different components of android application
* It is mostly used to

1. start an activity

2. start a service

3. deliver a broadcast.

* There are 2 types of intents:

1. Implicit Intent:

* It is used to start the component where we don't know the class name of the activity or service we want to start.
* Here general action is specified to start the components
* For example, if you want to show the user a location on a map, you can use an implicit intent to request that map application from your app to show a specified location.
* When you use an implicit intent, the Android system finds the appropriate component to start by comparing the contents of the intent to the intent filters declared in the manifest file of other apps on the device.
* If the intent matches an intent filter, the system starts that component and delivers it the Intent object.
* If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.
* To start the component of android application using Implicit Intent we need to have the information about the intent filter of that application
* Intent filter is given for the component inside the manifest file
* Intent filter consists of 3 elements

1. Action:

It determines the action for an intent

It should be the string value of an action

It should not be the classname string constant

1. data:

It determines what type of data can be passed to this component

eg -> URI(scheme,host,port,path) and MIME type.

1. category:

It determines the intent action belong to which type of category

It should be the string value of an category

It should not be the classname string constant

public void btnCall(View view){

Intent intent = new Intent(Intent.ACTION\_DIAL,

Uri.parse("tel:8983049388"));

startActivity(intent);

}

public void btnBrowse(View view){

Intent intent = new

Intent(Intent.ACTION\_VIEW,Uri.parse("https://sunbeaminfo.com/placements"));

startActivity(intent);

}

public void btnSendMessage(View view){

Intent intent = new Intent(Intent.ACTION\_SEND);

intent.putExtra(Intent.EXTRA\_TEXT, "Hello");

intent.setType("text/plain");

startActivity(intent);

}

1. Explicit Intent:

* It is used specify which component of which application will satisfy the intent, by specifying a full ComponentName
* It is used to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you might start a new activity within your app in response to a user action, or start a service to download a file in the background.
* To start the component of android application using Explicit Intent we need to have the information about the Fully Qualified Classname of that component from android application.

public void btnSecond(View view){

Intent intent = new Intent(this, SecondActivity.class);

startActivity(intent);

}

//OR

public void btnSecond(View view){

Intent intent = new Intent(this, SecondActivity.class);

intent.putExtra("k\_name","v\_sunbeam");

startActivity(intent);

}

// in second activity get the data if passed through intent

Intent intent = getIntent();

String name = intent.getStringExtra("k\_name");

Log.e(tag,name);

**Starting the Activity for Result:**

* Starting another activity, whether it is one within your app or from another app, doesn't need to be a one-way operation.
* You can also start an activity and receive a result back.
* For example, your app can start a camera app and receive the captured photo as a result. Or you might start the Contacts app for the user to select a contact, and then receive the contact details as a result.
* The Activity Result APIs provide components for registering for a result, launching the activity that produces the result, and handling the result once it is dispatched by the system.
* When in a ComponentActivity or a Fragment, the Activity Result APIs provide a registerForActivityResult() API for registering the result callback. registerForActivityResult() takes an ActivityResultContract and an ActivityResultCallback and returns an ActivityResultLauncher, which you use to launch the other activity.

ActivityResultLauncher activityResultLauncher = registerForActivityResult(new

ActivityResultContracts.StartActivityForResult(), new

ActivityResultCallback<ActivityResult>() {

@Override

public void onActivityResult(ActivityResult result) {

}

});

public void getStudentData(View v){

Intent intent = new Intent(this, InputStudentDetails.class);

activityResultLauncher.launch(intent);

}

**APP Bar (Action Bar):**

* The app bar, also known as the action bar, is one of the most important design elements in your app's
* activities, because it provides a visual structure and interactive elements that are familiar to users.
* It is a view which is present at top of the activity
* The key functions of the app bar are as follows:

1. Dedicated space for giving your app an identity and indicating the user's location in the app.
2. Predictable access to important actions, such as search.
3. Support for navigation and view switching, using tabs or menus.

* In its most basic form, the action bar displays the title for the activity on one side and an overflow menu on the other.
* If you want the action bar on all activities then change the themes.xml file.
* Select the theme as Theme.Material3.DayNight into themes.xml file
* If you don’t want the action bar on specific activities and then hide it using below method.

getSupportActionBar().hide();

* If you want to add the title to action bar then:

getSupportActionBar().setTitle("Details");

* To provide the back action button on action bar add the parent attribute in manifest file for that corresponding activity

<activity

android:parentActivityName=".MainActivity"

android:name=".SecondActivity"

android:exported="false" />

* As we don’t require the action bar over here we can hide it using the hide().

**ToolBar:**

* It is similar to the ActionBar but is more cutomizable
* Action Bar is part of Activity where as the tool bar is a View Group
* A Toolbar is a generalization of action bars for use within application layouts. While an action bar is traditionally part of an Activity's opaque window decor controlled by the framework, a - Toolbar may be placed at any arbitrary level of nesting within a view hierarchy.
* An application may choose to designate a Toolbar as the action bar for an Activity using the setSupportActionBar() method.

<androidx.appcompat.widget.Toolbar

android:id="@+id/toolbBar"

android:background="@android:color/darker\_gray"

app:title="Register"

app:titleTextColor="@color/white"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

Toolbar toolbar = findViewById(R.id.toolBar);

setSupportActionBar(toolbar);

**Menus:**

* Menus are a common user interface component
* There are 3 types of menus:

1. Options menu:

* The options menu is the primary collection of menu items for an activity.
* It's where you place actions such as "Search," "Compose email," and "Settings".

1. Popup menu

* A popup menu displays a vertical list of items that's attached to the view that invokes the menu.
* we can display such menus on long click on the list view item.

1. Context menu

* A context menu is a floating menu that appears when the user performs a touch & hold on an element.
* It provides actions that affect the selected content or context frame.
* You can provide a context menu for any view, but they are most often used for items in a RecylerView or other view collections in which the user can perform direct actions on each item.

// Options Menu

@Override

public boolean onCreateOptionsMenu(Menu menu)

{

//menu.add("Add");

//menu.add("Settings");

getMenuInflater().inflate(R.menu.mainmenu,menu);

return super.onCreateOptionsMenu(menu);

}

@Override

public boolean onOptionsItemSelected(@NonNull MenuItem item) {

if(item.getTitle().equals("Add"))

Toast.makeText(this, "Add", Toast.LENGTH\_SHORT).show();

if(item.getTitle().equals("Settings"))

Toast.makeText(this, "Settings", Toast.LENGTH\_SHORT).show();

return super.onOptionsItemSelected(item);

}

// Popup Menu

public void popupMenu(View view){

PopupMenu popupMenu = new PopupMenu(this,view);

popupMenu.getMenu().add("Edit");

popupMenu.getMenu().add("Delete");

popupMenu.show();

popupMenu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {

@Override

public boolean onMenuItemClick(MenuItem item) {

if(item.getTitle().equals("Edit"))

Toast.makeText(MainActivity.this, "Edit",

Toast.LENGTH\_SHORT).show();

if(item.getTitle().equals("Delete"))

Toast.makeText(MainActivity.this, "Delete",

Toast.LENGTH\_SHORT).show();

return false;

}

});

}

//Context Menu

@Override

public void onCreateContextMenu(ContextMenu menu, View v,

ContextMenu.ContextMenuInfo menuInfo) {

super.onCreateContextMenu(menu, v, menuInfo);

menu.add("Edit");

menu.add("Delete");

}

@Override

public boolean onContextItemSelected(@NonNull MenuItem item) {

if(item.getTitle().equals("Edit"))

Toast.makeText(MainActivity.this, "Edit", Toast.LENGTH\_SHORT).show();

if(item.getTitle().equals("Delete"))

Toast.makeText(MainActivity.this, "Delete", Toast.LENGTH\_SHORT).show();

return super.onContextItemSelected(item);

}

<menu xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto">

<item

app:showAsAction="always"

android:icon="@drawable/add"

android:title="Add"/>

<item

app:showAsAction="collapseActionView"

android:title="Settings"/>

</menu>

**ListView:**

* Displays a vertically-scrollable collection of views
* each view is positioned immediately below the previous view in the list.
* It does not know the details, such as type and contents, of the views it contains.
* Instead list view requests views on demand from the Adapter as needed, such as to display new views as the user scrolls up or down.
* For more advanced features and performance it is recommended to use RecyclerView.

<ListView

android:id="@+id/listView"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

**Recycler View:**

* RecyclerView makes it easy to efficiently display large sets of data.
* You supply the data and define how each item looks, and the RecyclerView library dynamically creates the elements when they're needed.
* As the name implies, RecyclerView recycles those individual elements.
* When an item scrolls off the screen, RecyclerView doesn't destroy its view. Instead, RecyclerView reuses the view for new items that have scrolled onscreen.
* RecyclerView improves performance and your app's responsiveness, and it reduces power consumption.
* It is a ViewGroup which is used to display the data in the form of List or Grid
* It is more efficient and customizable than the ListView
* RecyclerView is the ViewGroup that contains the views corresponding to your data.
* It's a view itself, so you add RecyclerView to your layout the way you would add any other UI element.

Implementation of Recylcer View, adapter and view holder:

**activity\_main.xml**

<androidx.recyclerview.widget.RecyclerView

android:id="@+id/recyclerView"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"/>

**CountryAdapter.java**

public class CountryAdapter extends

RecyclerView.Adapter<CountryAdapter.MyViewHolder> {

Context context;

List<String> countries;

public CountryAdapter(Context context, List<String> countries) {

this.context = context;

this.countries = countries;

}

@NonNull

@Override

public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

View view =

LayoutInflater.from(context).inflate(R.layout.countries\_list,null);

return new MyViewHolder(view);

}

@Override

public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {

holder.textCountry.setText(countries.get(position));

}

@Override

public int getItemCount() {

return countries.size();

}

class MyViewHolder extends RecyclerView.ViewHolder{

TextView textCountry;

public MyViewHolder(@NonNull View itemView) {

super(itemView);

textCountry = itemView.findViewById(R.id.textCountry);

}

}

}

**MainActivity:**

**// Inside MainActivity**

RecyclerView recyclerView;

List<String> countries = new ArrayList<>();

recyclerView = findViewById(R.id.recyclerView);

countries.add("India");

countries.add("USA");

countries.add("UK");

countries.add("China");

countries.add("Japan");

CountryAdapter countryAdapter = new CountryAdapter(this,countries);

recyclerView.setAdapter(countryAdapter);

recyclerView.setLayoutManager(new LinearLayoutManager(this));

// OR

recyclerView.setLayoutManager(new GridLayoutManager(this,1));

* The ViewHolder is a wrapper around a View that contains the layout for an individual item in the list.
* The Adapter creates ViewHolder objects as needed and also sets the data for those views.
* The process of associating views to their data is called binding.
* When you define your adapter, you override three key methods:

1. onCreateViewHolder()

* RecyclerView calls this method whenever it needs to create a new ViewHolder.
* The method creates and initializes the ViewHolder and its associated View, but does not fill in the view's contents—the ViewHolder has not yet been bound to specific data.

1. onBindViewHolder()

* RecyclerView calls this method to associate a ViewHolder with data.
* The method fetches the appropriate data and uses the data to fill in the view holder's layout.
* For example, if the RecyclerView displays a list of names, the method might find the appropriate name in the list and fill in the view holder's TextView widget.

1. getItemCount()

* RecyclerView calls this method to get the size of the dataset.
* For example, in an address book app, this might be the total number of addresses.
* RecyclerView uses this to determine when there are no more items that can be displayed.

LayoutManager:

* The items in your RecyclerView are arranged by a LayoutManager class.
* The RecyclerView library provides three layout managers, which handle the most common layout situations

1. LinearLayoutManager arranges the items in a one-dimensional list.
2. GridLayoutManager arranges the items in a two-dimensional grid

* If the grid is arranged vertically, GridLayoutManager tries to make all the elements in each row have the same width and height, but different rows can have different heights.
* If the grid is arranged horizontally, GridLayoutManager tries to make all the elements in each column have the same width and height, but different columns can have different widths.

1. StaggeredGridLayoutManager is similar to GridLayoutManager, but it does not require that items in a row have the same height (for vertical grids) or items in the same column have the same width (for horizontal grids). The result is that the items in a row or column can end up offset from each other.

// GridLayoutManager ->

recyclerView.setLayoutManager(new

GridLayoutManager(this,1,LinearLayoutManager.VERTICAL,false));

recyclerView.setLayoutManager(new

GridLayoutManager(this,1,LinearLayoutManager.HORIZONTAL,false));

recyclerView.setLayoutManager(new

GridLayoutManager(this,5,LinearLayoutManager.VERTICAL,false));

recyclerView.setLayoutManager(new

GridLayoutManager(this,5,LinearLayoutManager.HORIZONTAL,false));

**Click Animation:**

* The recyclerview do not show any animation effect when clicked.
* If we want the ripple/hover effect on click of the view inside recyclerview then add the below attributes inside the view that needs to be animated when clicked.

<TextView

android:background="?android:attr/selectableItemBackground"

android:clickable="true"

android:text="TextView that animate when clicked"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

**Fragments:**

* A Fragment represents a reusable portion of your app's UI.
* It defines and manages its own layout, has its own lifecycle, and can handle its own input events.
* Fragments can't live on their own. They must be hosted by an activity or another fragment.
* Create a new Blank Fragment and name it as FirstFragment
* To load the fragment into mainActivity add the below code in activity\_main.xml file.

<androidx.fragment.app.FragmentContainerView

android:id="@+id/fragment\_container\_view"

android:name="com.sunbeaminfo.app1.FirstFragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"/>

* Fragment lifecycle is managed by its callback methods which are as below:

1. onCreate

2. onCreateView

3. onViewCreated

4. onViewStateRestored

5. onStart

6. onResume

7. onPause

8. onStop

9. onDestroy

**ViewPager2:**

It is used to display Fragments in a swipe able format. viewpager requires an custom adapter which will provide the information about the fragments.

**Android Storage:**

Android provides several options to save your apps data:

1. App specific Storage: Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage.
2. Shared Storage: Store files that your app intends to share with other apps, including media, documents, and other files.
3. Preferences (Shared Preferences): Store private, primitive data in key-value pairs.
4. Database (Sqlite Database): Store structured data in a private database using the Room persistence library.
5. **App Specific Storage:**

* In many cases, your app creates files that other apps don't need to access, or shouldn't access.
* The system provides the following locations for storing such app-specific files:

1. Internal storage directories:

* These directories include both a dedicated location for storing persistent files,and another location for storing cache data.
* The system prevents other apps from accessing these locations, and on Android 10 (API level 29) and higher, these locations are encrypted.
* These characteristics make these locations a good place to store sensitive data that only your app itself can access.

1. External storage directories:

* These directories include both a dedicated location for storing persistent files, and another location for storing cache data.
* Although it's possible for another app to access these directories if that app has the proper permissions, the files stored in these directories are meant for use only by your app.
* If you specifically intend to create files that other apps should be able to access, your app should store these files in the shared storage part of external storage instead.
* To Access and store files we can use the File API to access and store files.

**2. Shared Storage:**

* Use shared storage for user data that can or should be accessible to other apps and saved even if the user uninstalls your app.
* Android provides APIs for storing and accessing the following types of shareable data:

1. Media content:

* The system provides standard public directories for these kinds of files, so the user has a common location for all their photos, another common location for all their music and audio files, and so on.
* Your app can access this content using the platform's MediaStore API.

1. Documents and other files:

* The system has a special directory for containing other file types, such as PDF documents and books.
* Your app can access these files using the platform's Storage Access Framework.

1. Datasets:

* On Android 11 (API level 30) and higher, the system caches large datasets that multiple apps might use.
* These datasets can support use cases like machine learning and media playback.
* Apps can access these shared datasets using the BlobStoreManager API.

**3. Sqlite Database:**

* Saving data to a database is ideal for repeating or structured data, such as contact information.
* It is a mini version of actual sql database for android.
* It is used to store the structured data inside it.
* The SQLiteOpenHelper class contains a useful set of APIs for managing your database.
* When you use this class to obtain references to your database, the system performs the potentially long-running operations of creating and updating the database only when needed and not during app startup.
* All you need to do is call getWritableDatabase() or getReadableDatabase().
* To use SQLiteOpenHelper, create a subclass that overrides the onCreate() and onUpgrade() callback methods.

public class DbHelper extends SQLiteOpenHelper {

// ctor that creates database with the given name and version

public DbHelper(@Nullable Context context) {

super(context, "dmc\_db", null, 1);

}

// This creates the required tables in the database.

// gets called only once

@Override

public void onCreate(SQLiteDatabase db) {

Log.e("Sqlite","oncreate called");

String sql = "CREATE TABLE employee(empid INTEGER PRIMARY KEY

AUTOINCREMENT,name TEXT,sal REAL)";

db.execSQL(sql);

}

// This method gets called if new tables are added or dropped.

// gets called only when version is changed

@Override

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {}

// method to insert the employee in database

public void insertEmployee(Employee employee){

SQLiteDatabase db = getWritableDatabase();

ContentValues values = new ContentValues();

values.put("name",employee.getName());

values.put("sal",employee.getSal());

db.insert("employee",null,values);

db.close();

}

// method to get all the employees from database

public List<Employee> getEmployees() {

List<Employee> employeeList = new ArrayList()

SQLiteDatabase db = getReadableDatabase();

Cursor cursor = db.query("employee",null,null,null,null,null,null);

while(cursor.moveToNext()){

Employee e = new Employee();

e.setEmpid(cursor.getInt(0));

e.setName(cursor.getString(1));

e.setSal(cursor.getDouble(2));

employeeList.add(e);

}

db.close();

return employeeList;

}

// method to edit the employee from database

public void editEmployee(int empid,Employee employee) {

SQLiteDatabase db = getWritableDatabase();

ContentValues values = new ContentValues();

values.put("sal",employee.getSal());

db.update("employee",values,"empid=?",new String[]{empid+""});

db.close();

}

// method to delete the employee from database

public void deleteEmployee(int empid) {

SQLiteDatabase db = getWritableDatabase();

db.delete("employee","empid=?",new String[]{empid+""});

db.close();

}

}

**Room Persistance Library**

The Room persistence library provides an abstraction layer over SQLite to allow fluent database access

while harnessing the full power of SQLite. In particular, Room provides the following benefits:

1. Compile-time verification of SQL queries.

2. Convenience annotations that minimize repetitive and error-prone boilerplate code.

3. Streamlined database migration paths.

Because of these considerations, it is highly recommend that you use Room instead of using the SQLite

APIs directly.

To implement the crud operations using Room Persistance Library, add the dependencies inside your

gradle

def room\_version = "2.7.1"

implementation "androidx.room:room-runtime:$room\_version"

annotationProcessor "androidx.room:room-compiler:$room\_version"

val room\_version = "2.7.1"

implementation("androidx.room:room-runtime:$room\_version")

annotationProcessor("androidx.room:room-compiler:$room\_version")

There are three major components in Room:

1. The Database class

that holds the database and serves as the main access point for the underlying connection to

your app's persisted data.

2. Data Entities

SUNBEAM INFOTECH

Prepared By : Rohan Paramane

4 / 6

Day08\_Help.MD

Sunbeam Infotech

2025-05-20

that represent tables in your app's database.

3. Data Access Objects (DAOs)

that provide methods that your app can use to query, update, insert, and delete data in the

database.

Steps for the implementation

1. Create and entity class Person with the annotaions that will represent the fields of table in database

@Entity(tableName = "person")

public class Person {

@PrimaryKey(autoGenerate = true)

private int id;

@ColumnInfo(name = "pname")

private String name;

@ColumnInfo

private int age;

}

2. Cretae the package db and inside it add the PersonDao interface which will have methods along with

annotaion for perfoming the CRUD operations.

@Dao

public interface PersonDao {

@Insert

void insertPerson(Person person);

@Query("SELECT \* FROM person")

List<Person> getData();

@Query("SELECT \* FROM person WHERE age>:age")

List<Person> getSpecificPersons(int age);

@Query("DELETE FROM person WHERE id=:id")

void deletePerson(int id);

@Query("UPDATE person set name=:name where id=:id")

void updatePerson(int id,String name);

}

3. Create an abstract class PersonDatabase which represent the database, name, and the version. use

proper annotations. It will be used tp return the Dao objects

here the class name should be kept as AppDatabase as it is an applicatio level database. however for

better understanding and done for single person class we are usnig the name as PersonDatabase

SUNBEAM INFOTECH

Prepared By : Rohan Paramane

5 / 6

Day08\_Help.MD

Sunbeam Infotech

2025-05-20

@Database(entities = {Person.class},version = 1)

public abstract class PersonDatabase extends RoomDatabase {

private static PersonDatabase personDatabase = null;

public abstract PersonDao personDao();

//method that returns the object of PersonDatabase

public static PersonDatabase getInstance(Context context){

if(personDatabase == null){

personDatabase =

Room.databaseBuilder(context,PersonDatabase.class,"person\_db")

. allowMainThreadQueries()

. build();

}

return personDatabase;

}

}

4. use this class to call the methods declared in the PersonDao interface.

5. When any changes is done in the db and need to be reflected inside the recycler view it might not

refresh the recyclerview with the method notifyDatasetChanged() as we are creating multiple objects of

ArrayList, so for this we need to use setter for that collection use it inside the adapter.

// In Adapter

public void setPersonList(List<Person> personList) {

this.personList = personList;

notifyDataSetChanged();

}

**Content Provider:**

* A content provider manages access to a central repository of data.
* A provider is part of an Android application, which often provides its own UI for working with the data.
* However, content providers are primarily used by other applications, which access the provider using a provider client object.
* Together, providers and provider clients offer a consistent, standard interface to data that also handles interprocess communication and secure data access.
* Typically, you work with content providers in one of two scenarios: implementing code to access an existing content provider in another application or creating a new content provider in your application to share data with other applications.

**Content Resolver:**

* When you want to access data in a content provider, you use the ContentResolver object in your application's Context to communicate with the provider as a client.
* The ContentResolver object communicates with the provider object, an instance of a class that implements ContentProvider.
* The provider object receives data requests from clients, performs the requested action, and returns the results.
* This object has methods that call identically named methods in the provider object, an instance of one of the concrete subclasses of ContentProvider.
* The ContentResolver methods provide the basic "CRUD" (create, retrieve, update, and delete) functions of persistent storage.

**Content URI's:**

* A content URI is a URI that identifies data in a provider
* Content URIs include the symbolic name of the entire provider—its authority and a name that points to a table—a path
* When you call a client method to access a table in a provider, the content URI for the table is one of the arguments.
* The ContentProvider uses the path part of the content URI to choose the table to access.
* A provider usually has a path for each table it exposes.

content://user\_dictionary/words

* The content:// string is the scheme, which is always present and identifies this as a content URI.
* The user\_dictionary string is the provider's authority.
* The words string is the table's path.

A content provider presents data to external applications as one or more tables that are similar to the tables found in a relational database. A row represents an instance of some type of data the provider collects, and each column in the row represents an individual piece of data collected for an instance.

A content provider coordinates access to the data storage layer in your application for a number of different APIs and components. As illustrated in figure 1, these include the following:

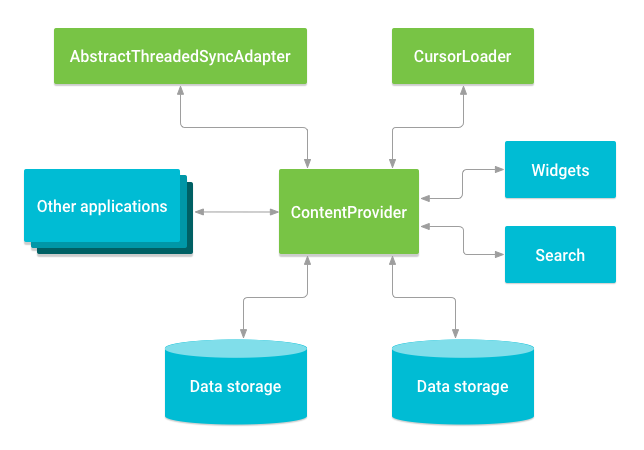
Sharing access to your application data with other applications

Sending data to a widget

Returning custom search suggestions for your application through the search framework using SearchRecentSuggestionsProvider

Synchronizing application data with your server using an implementation of AbstractThreadedSyncAdapter

Loading data in your UI using a CursorLoader



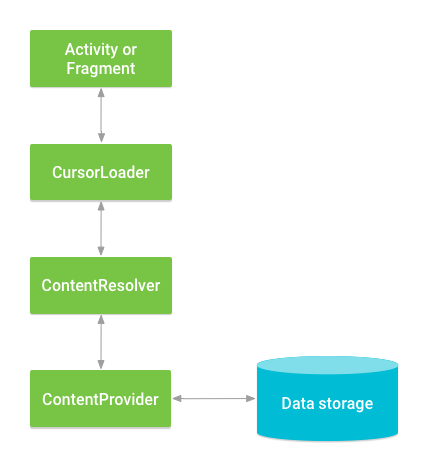
**Access a provider:**

When you want to access data in a content provider, you use the [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver) object in your application's [Context](https://developer.android.com/reference/android/content/Context) to communicate with the provider as a client. The ContentResolver object communicates with the provider object, an instance of a class that implements [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider).

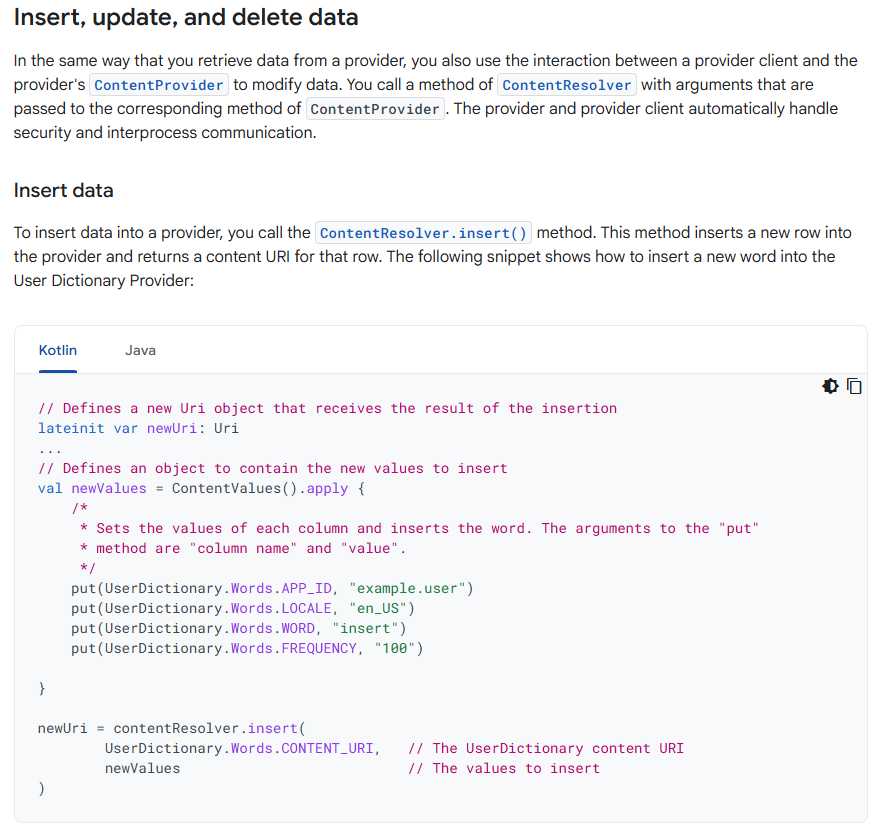
The provider object receives data requests from clients, performs the requested action, and returns the results. This object has methods that call identically named methods in the provider object, an instance of one of the concrete subclasses of ContentProvider. The ContentResolver methods provide the basic "CRUD" (create, retrieve, update, and delete) functions of persistent storage.

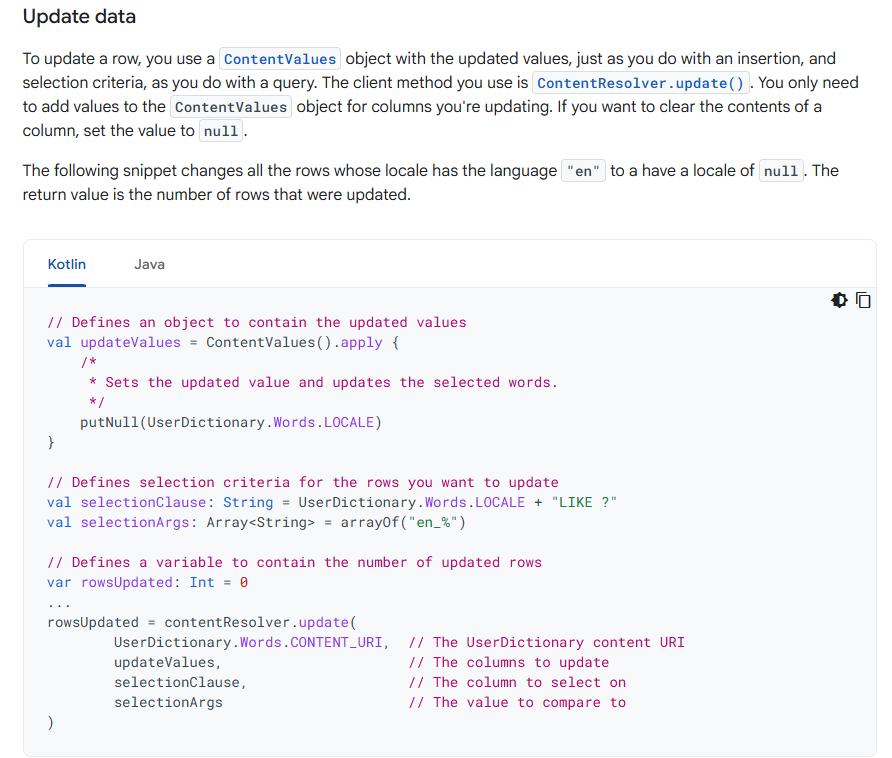
A common pattern for accessing a ContentProvider from your UI uses a [CursorLoader](https://developer.android.com/reference/android/content/CursorLoader) to run an asynchronous query in the background. The [Activity](https://developer.android.com/reference/android/app/Activity) or [Fragment](https://developer.android.com/reference/android/app/Fragment) in your UI calls a CursorLoader to the query, which in turn gets the ContentProvider using the ContentResolver.

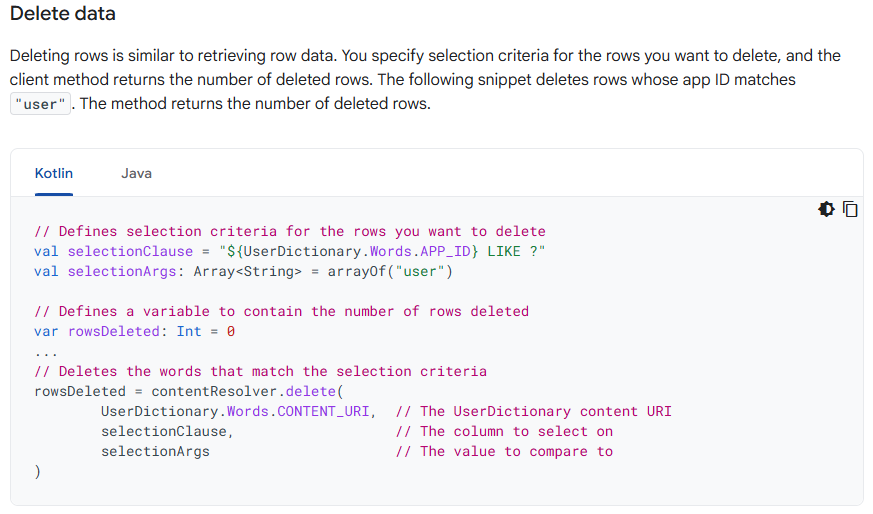
This lets the UI continue to be available to the user while the query is running. This pattern involves the interaction of a number of different objects, as well as the underlying storage mechanism, as illustrated in figure 2.

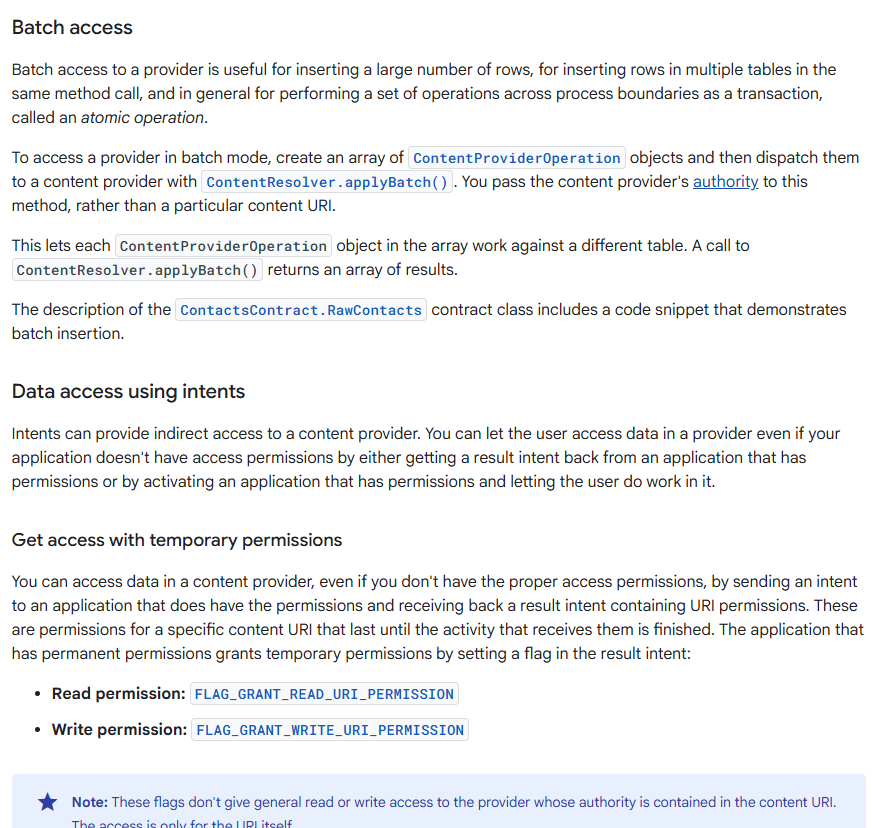
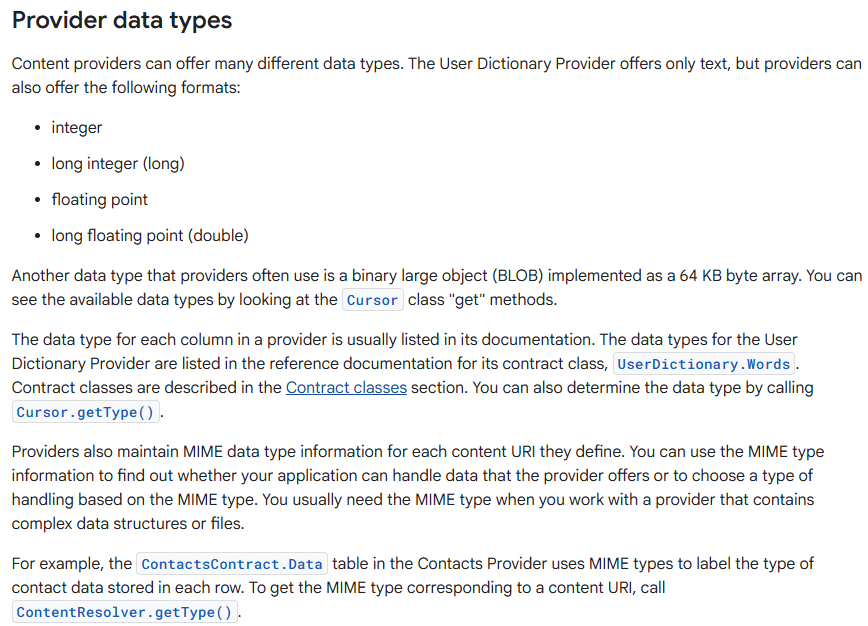


****

****

****

****

****

**SharedPreferences:**

* It is used to store primitive data in keyValue pairs
* If you have a relatively small collection of key-values that you'd like to save, you can use the SharedPreferences APIs.
* A SharedPreferences object points to a file containing key-value pairs and provides simple methods to read and write them.
* Each SharedPreferences file is managed by the framework and can be private or shared.
* To Get a handle to shared preferences we can create a new shared preference file or access an existing one by calling the method getSharedPreferences().
* we can call this from any Context in our app.
* When naming your shared preference files, you should use a name that's uniquely identifiable to your
* app.
* A good way to do this is prefix the file name with your application ID.
* For example: "com.example.myapp.PREFERENCE\_FILE\_KEY"
* To write to a shared preferences file, create a SharedPreferences.Editor by calling edit() on your SharedPreferences.
* Pass the keys and values you want to write with methods such as: putInt() andputString().
* Then call apply () or commit () to save the changes.
* To retrieve values from a shared preferences file, call methods such as getInt() and getString(), providing the key for the value you want, and optionally a default value to return if the key isn't present.

// To add the data in Shared preference

getSharedPreferences("com.sunbeaminfo.app1.PERSON\_SP",MODE\_PRIVATE).edit().putBool

ean("login\_status",true).apply();

// To get the data from Shared preference

boolean status =

getSharedPreferences("com.sunbeaminfo.app1.PERSON\_SP",MODE\_PRIVATE).getBoolean("lo

gin\_status",false);

**Animations:**

There are two types of animation:

1. Property Animation

* An animation defined in XML that modifies properties of the target object, such as background color or alpha value, over a set amount of time.
* Here we use the res/animator dir to keep our animator xml file

1. Tween Animation

* An animation defined in XML that performs transitions on a graphic such as rotating, fading, moving, and stretching.
* Here we use the res/anim dir to keep our animator xml file

Tween Animation:

An animation defined in XML that performs transitions on a graphic such as rotating, fading, moving, and stretching. The file must have a single root element: either an <alpha>, <scale>, <translate>, <rotate>, or <set> element that holds a group (or groups) of other animation elements (including nested elements).

1. alpha

A fade-in or fade-out animation. Represents an AlphaAnimation.

Attributes:

android:fromAlpha

Float. Starting opacity offset, where 0.0 is transparent and 1.0 is opaque.

android:toAlpha

Float. Ending opacity offset, where 0.0 is transparent and 1.0 is opaque.

1. rotate

It rotates the view in clockwise direction pivoted at the position(0,0) of x and y from the top left cornerof Image view.

to keep the position(0,0) at the center of the view pivot the x and y values to 50%

Attributes:

android:fromDegrees

Float. Starting angular position, in degrees.

android:toDegrees

Float. Ending angular position, in degrees.

android:pivotX

Float or percentage. The X coordinate of the center of rotation. Expressed either in pixels relative to the object's left edge, such as "5"; in percentage relative to the object's left edge, such as "5%"; or in percentage relative to the parent container's left edge, such as "5%p".

android:pivotY

Float or percentage. The Y coordinate of the center of rotation. Expressed either in pixels relative to the object's top edge, such as "5"; in percentage relative to the object's top edge, such as "5%"; or in percentage relative to the parent container's top edge, such as "5%p".

1. scale

It expands the view in both x and y direction.

It starts the position(0,0) of x and y from the top left corner of Image view

to keep the position(0,0) at the center of the view pivot the x and y values to 50%

Attributes:

android:fromXScale

Float. Starting X size offset, where 1.0 is no change.

android:toXScale

Float. Ending X size offset, where 1.0 is no change.

android:fromYScale

Float. Starting Y size offset, where 1.0 is no change.

android:toYScale

Float. Ending Y size offset, where 1.0 is no change.

android:pivotX

Float. The X coordinate to remain fixed when the object is scaled.

android:pivotY

Float. The Y coordinate to remain fixed when the object is scaled.

1. translate

A vertical and/or horizontal motion.

Represents a TranslateAnimation.

Supports the following attributes in any of these three formats:

Values from -100 to 100 ending with "%", indicating a percentage relative to itself.

Values from -100 to 100 ending in "%p", indicating a percentage relative to its parent.

A float value with no suffix, indicating an absolute value.

Attributes:

android:fromXDelta

Float or percentage. Starting X offset.

android:toXDelta

Float or percentage. Ending X offset.

android:fromYDelta

Float or percentage. Starting Y offset.

android:toYDelta

Float or percentage. Ending Y offset.

we can give the values for the attributes in pixels relative to the normal position, such as "5".

In percentage relative to the element width, such as "5%".

Also, in percentage relative to the parent width, such as "5%p".

For animation the onclick attribute of button may not work so in this case provide the id for button and set the onclick listener.

MultiThreading:

* When we execute a code, os cretaes a processs and the code starts the execution inside this process.
* If in the same process we want to execute some tasks parallely then we need to create sub processes under it which are called as threads.
* Threads are also called as light weight process which executes under the main process.
* When the user launches your app, Android creates a new process along with an execution thread.
* This main thread, also known as the UI thread, is responsible for everything that happens onscreen.
* Threads persist throughout the lifetime of the activities that create them.
* Threads continue to execute, uninterrupted, regardless of the creation or destruction of activities, although they will be terminated when the application terminates.

Spalsh Activity:

* A splash screen is mostly the first screen of the app when it is opened.
* It is a constant screen that appears for a specific amount of time and generally shows for the first time when the app is launched.
* Splash screen is used to display some basic introductory information such as the company logo, content, etc just before the app loads completely.
* create a splash activity and make it launcher activity, add the animation on it and then launch the next activity after the animation is finished on splash activity.

Consuming REST API:

1. Add the dependency of retrofit to your application



1. Create an interface called as API interface that will keep all the required methods to call the existing API's
2. In the interface provide the methods with the proper annotation that will map to the existing backend API's



1. Create the Singleton class Retrofit client that will help us to get the object of this API interface.

public class RetrofitClient {

private static RetrofitClient retrofitClient;

private API api;

private RetrofitClient(){

api = new Retrofit.Builder()

.baseUrl(API.BASE\_URL)

.addConverterFactory(GsonConverterFactory.create())

.build()

.create(API.class);

}

public static RetrofitClient getInstance(){

if(retrofitClient!=null)

retrofitClient = new RetrofitClient();

return retrofitClient;

}

public API getApi() {

return api;

}

}

1. Start calling the api as per the requirement:

RetrofitClient.getInstance().getApi().registerUser(user).enqueue(new

Callback<JsonObject>() {

@Override

public void onResponse(Call<JsonObject> call, Response<JsonObject> response) {

Toast.makeText(RegisterActivity.this, "Registration is successful",

Toast.LENGTH\_SHORT).show();

}

@Override

public void onFailure(Call<JsonObject> call, Throwable t) {

Toast.makeText(RegisterActivity.this, "Something went wrong",

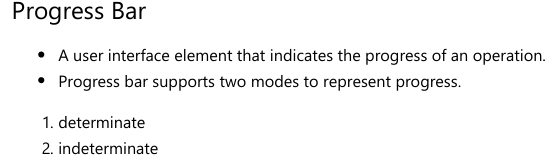
Toast.LENGTH\_SHORT).show();

}

});

1. To make the call and get the response successfully we need the permission to use the internet and the permission of usesCleartextTraffic. In manifest add both the permissions





<ProgressBar

android:id="@+id/progressBarInDeterminate"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<!-- Add the style in progress bar as horizontal and it will be determinate -->

<ProgressBar

style="@style/Widget.AppCompat.ProgressBar.Horizontal"

android:id="@+id/progressBarDeterminate"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

* Determinate has the ability to set the progress to it.
* We can set the max and min progress to it and also update it as per the requirement
* for Indeterminate we cannot set the progress.
* the views should not be manipulated inside threads as they can cause unexpected behaviour.
* to manipulate them we should always use the main UI Thread.
* to get the main UI Thread use the post() method on the views.

progressBarDeterminate.post(new Runnable() {

@Override

public void run() {

progressBarDeterminate.setVisibility(View.INVISIBLE);

}

});

progressBarInDeterminate.post(new Runnable() {

@Override

public void run() {

progressBarInDeterminate.setVisibility(View.INVISIBLE);

}

});

**Date and Time Picker Dialog:**

// For Date Picker Dialog

DatePickerDialog datePickerDialog = new DatePickerDialog(this);

datePickerDialog.setOnDateSetListener(new DatePickerDialog.OnDateSetListener() {

@Override

public void onDateSet(DatePicker view, int year, int month, int dayOfMonth) {

textViewDate.setText(dayOfMonth + "/" + month + "/" + year);

}

});

datePickerDialog.show();

// optional to get Time to be displayed inside timepickerdailog

Calendar calendar = Calendar.getInstance();

int hr = calendar.get(Calendar.HOUR);

int min = calendar.get(Calendar.MINUTE);

// For Time Picker Dialog

TimePickerDialog timePickerDialog = new TimePickerDialog(this, new

TimePickerDialog.OnTimeSetListener() {

@Override

public void onTimeSet(TimePicker view, int hourOfDay, int minute) {

String AM\_PM = hourOfDay < 12 ? " AM" : " PM";

textViewTime.setText(hourOfDay + " : " + minute + AM\_PM);

}

}, hr, min, false);

timePickerDialog.show();

**In App Notifications:**

To send notifications first add the permission in manifest and provide the permission to send the notification



To send the notification from app we require objects of

1. NotificationChannel

It creates a new channel for notification that needs to be declared to the android

2. NotificationManager

It is an object that is used to register our channel towards the NOtification Service in android

3. NotificationCompact

To create the new notification along with the required data to be shown inside it.

4. NotificationManagerCompact

It is used to display the notification in our status bar.

public void sendNotification(View view) {

// cretae a notification channel for your notification

NotificationChannel notificationChannel = new NotificationChannel("channel\_id",

"channel\_Name", NotificationManager.IMPORTANCE\_DEFAULT);

// register your notification channel with the Android Notification service

NotificationManager notificationManager =

getSystemService(NotificationManager.class);

notificationManager.createNotificationChannel(notificationChannel);

// Cretae the notification thet you want to display and pass the channel\_id

NotificationCompat.Builder builder = new NotificationCompat.Builder(this,

"channel\_id");

builder.setSmallIcon(R.drawable.watch);

builder.setContentTitle("New Notification");

builder.setContentText("This is a new Notification sent.");

//To display larger texts

builder.setStyle(new NotificationCompat.BigTextStyle().bigText("This is a new

Notification sent from your app to check. wether it works or no"));

//before displaying notification it requires the permission to check if post

notification permission is given or no

if (ActivityCompat.checkSelfPermission(this, android.Manifest.permission

POST\_NOTIFICATIONS) != PackageManager.PERMISSION\_GRANTED)

return;

// display the notification created immediately in your notifications

// the int value in notifiy determines wether to create new notification if

already one exists or to manipulate the same

NotificationManagerCompat.from(MainActivity.this).notify(1, builder.build());

}

**We can set the tap icon on the notification and launch the activity from our app.**

**For this we require object of PendingIntent**

**this should be done before we notify.**

//Handle the click event of notification

Intent intent = new Intent(this, SplashActivity.class);

//only set this flag if you want to start the activity when you are already

working with the app.

// however the activity launches only when the app is closed when this flag is not

set.

intent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

PendingIntent pendingIntent =PendingIntent.getActivity(this,0,intent,PendingIntent.FLAG\_IMMUTABLE);

builder.setContentIntent(pendingIntent);

builder.setAutoCancel(true);

**we can also set the progress to notifications.**

**it will be used when downloading needs to be shown in the notification.**

new Thread(new Runnable() {

@Override

public void run() {

int PROGRESS\_MAX = 100;

int PROGRESS\_CURRENT = 0;

for (int i = 1; i <= 10; i++) {

try {

Thread.sleep(1000);

PROGRESS\_CURRENT = 10 \* i;

builder.setProgress(PROGRESS\_MAX, PROGRESS\_CURRENT, false);

notificationManagerCompat.notify(3, builder.build());

}

catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

builder.setContentText("Downlaod Finished");

// set the pending intent when download completes

builder.setContentIntent(pendingIntent);

notificationManagerCompat.notify(3,builder.build());

}

}).start();

**Sensors:**

Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional device movement or positioning, or you want to monitor changes in the ambient environment near a device. For example, a game might track readings from a device's gravity sensor to infer complex user gestures and motions, such as tilt, shake, rotation, or swing. Likewise, a weather application might use a device's temperature sensor and humidity sensor to calculate and report the dewpoint, or a travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing.

The Android platform supports three broad categories of sensors:

Motion sensors

These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

Environmental sensors

These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

Position sensors

These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

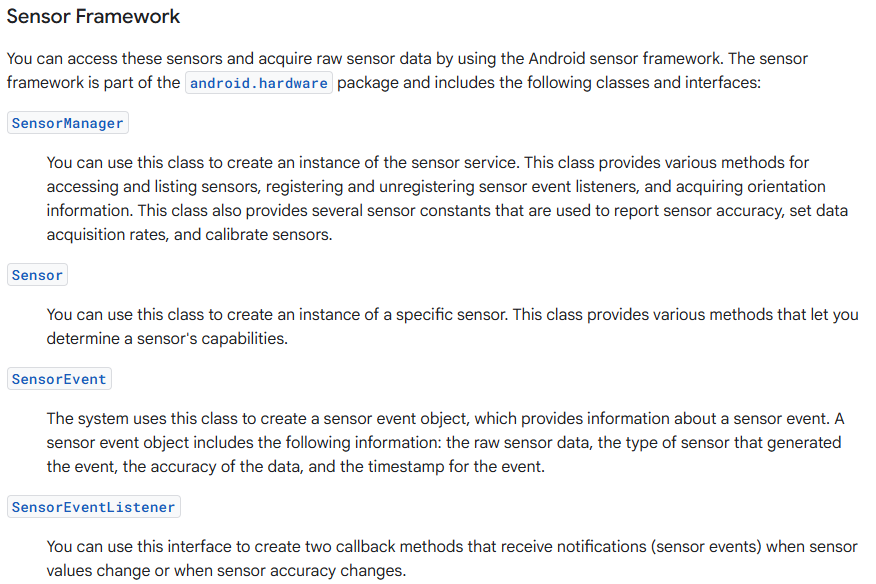
You can access sensors available on the device and acquire raw sensor data by using the Android sensor framework. The sensor framework provides several classes and interfaces that help you perform a wide variety of sensor-related tasks. For example, you can use the sensor framework to do the following:

Determine which sensors are available on a device.

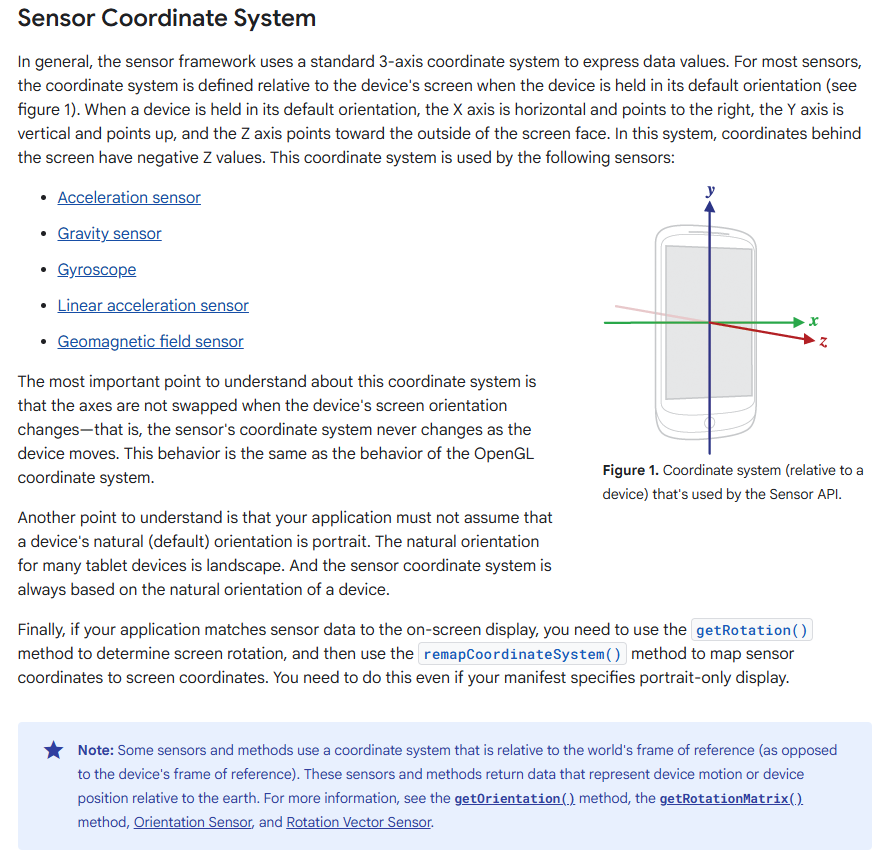
Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.

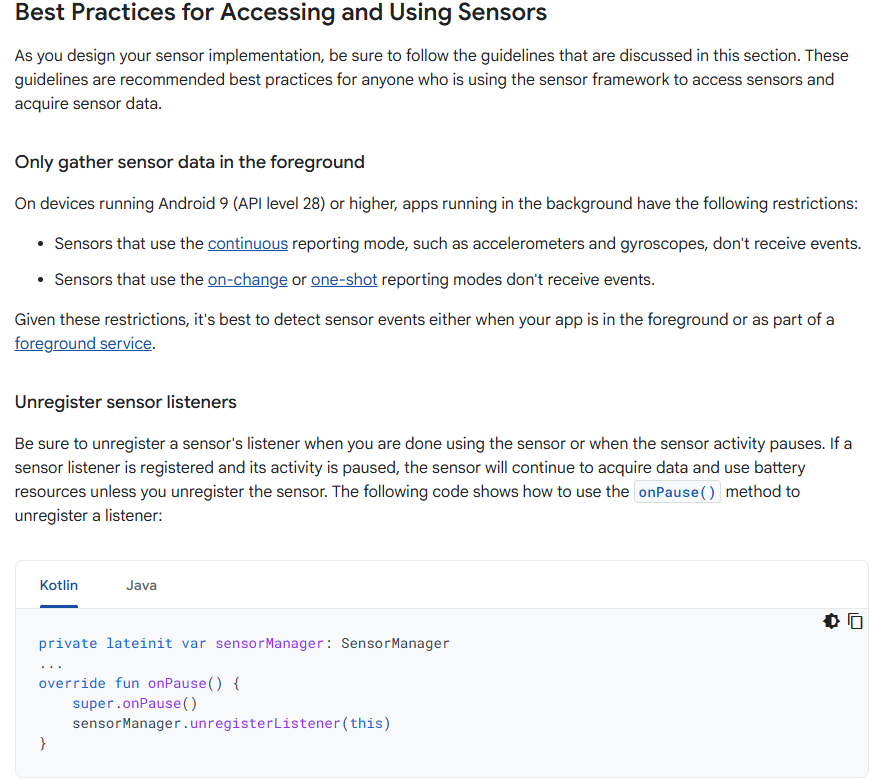
Acquire raw sensor data and define the minimum rate at which you acquire sensor data.

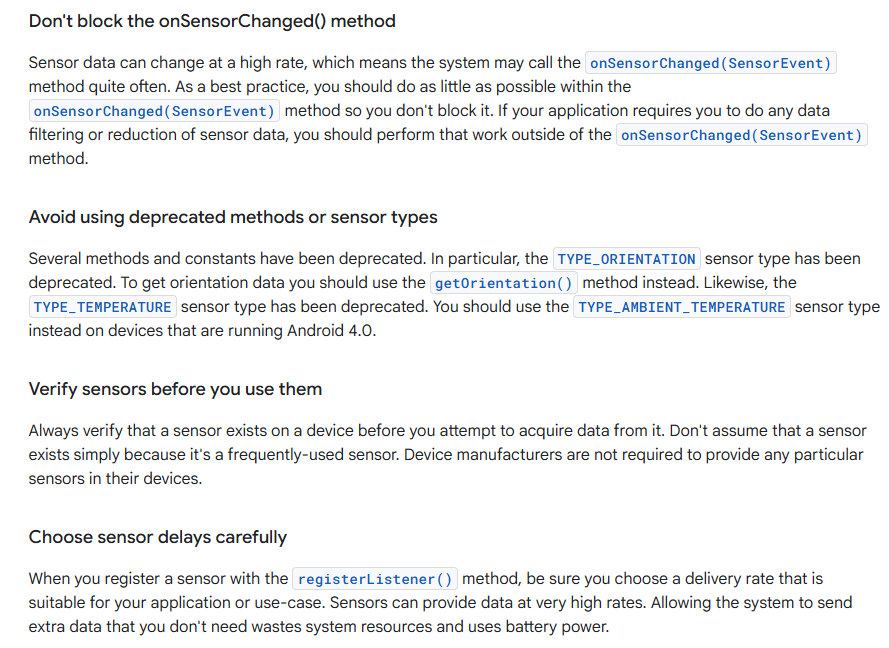
Register and unregister sensor event listeners that monitor sensor changes.



****







**Services:**

A Service is an application component that can perform long-running operations in the background. Service does not provide a user interface. Once started, a service might continue running for some time, even after the user switches to another application. 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. You should run any blocking operations on a separate thread within the service to avoid Application Not Responding (ANR) errors. There are the three different types of services:

1. Foreground:

* A foreground service performs some operation that is noticeable to the user.
* Foreground services must display a Notification.
* This notification cannot be dismissed unless the service is either stopped or removed from the foreground.
* Foreground services continue running even when the user isn't interacting with the app.
* For example, an audio app would use a foreground service to play an audio track.

1. Background:

* A background service performs an operation that isn't directly noticed by the user.
* For example, if an app used a service to compact its storage, that would usually be a background service.

1. 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
* 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.
* Service has its own lifecycle which is managed by the callback methods
* We can create our service in two ways depending on which the callback method

1. Bounded Service

onCreate()

onBind()

onUnbind()

onDestroy()

1. UnBounded / Started Service

onCreate()

onStartCommand()

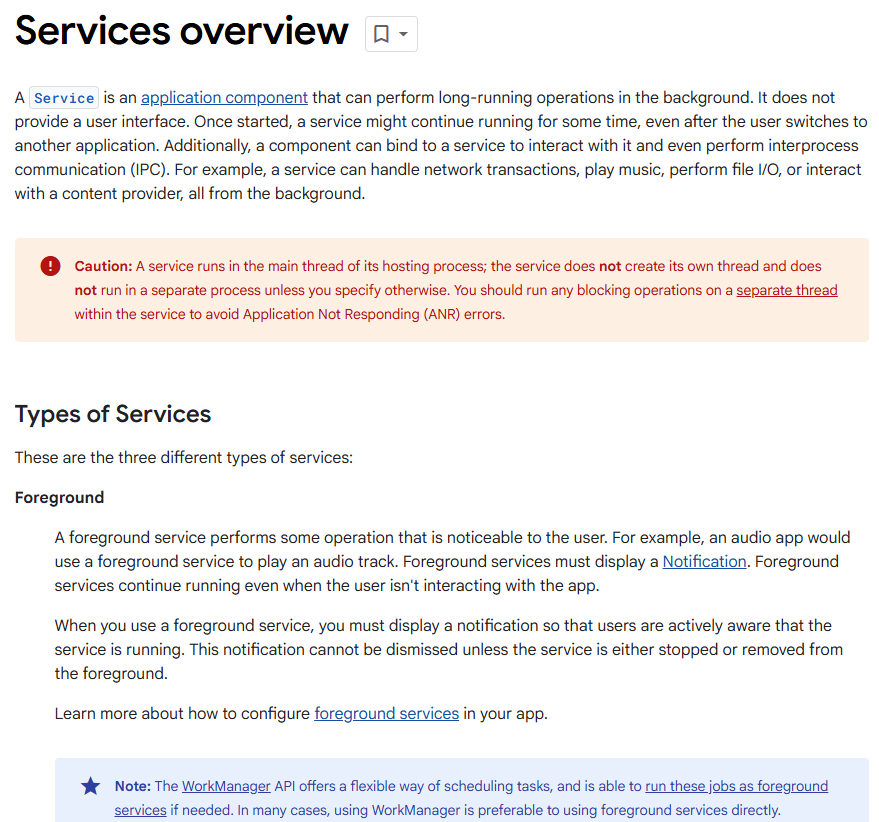
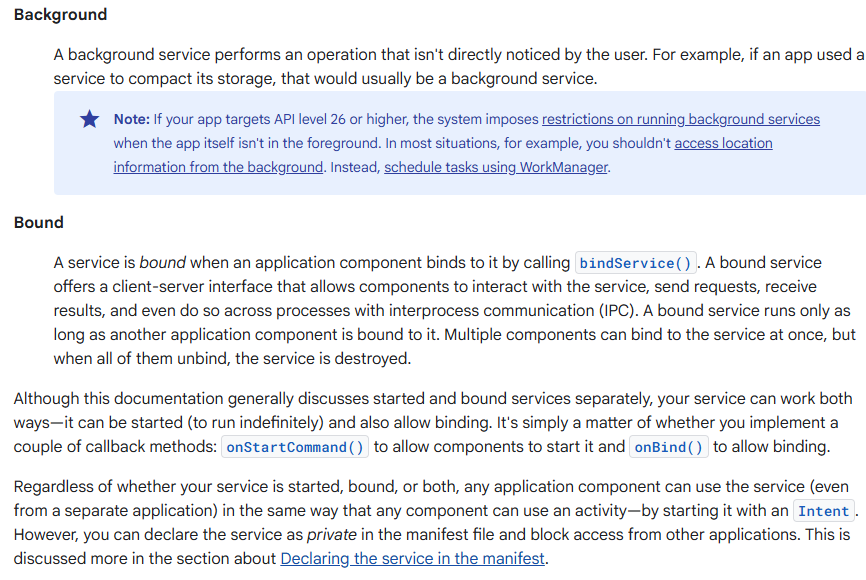
onDestroy()

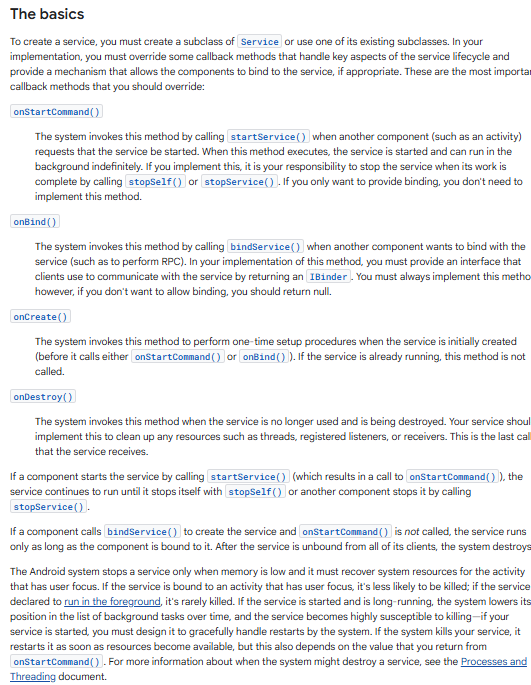
BoundedService:

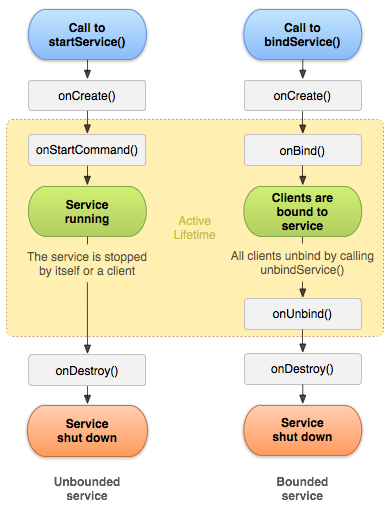
* Right click on your package and select Service -> Service
* Check the entry of Service into your maninfest
* Add all the Lifecycle callback methods of service in MyService class
* Add 2 buttons into MainActivity to start and stop the service and handle the click event
* To start the service in bounded mode you have to create Service Connection object
* To start service use the method onBind to start the service in bounded mode
* To stop Service use the method onUnBind.
* To perform a download task we can perform it only once into the onBind Method
* If we want to perform the task multiple times then we need the Service object that is binded with that component
* For this in MyService add a nested class MyBinder that extends from Binder which is responsible to
* return MyService class object.
* From onBind() return the new object of MyBinder
* This will call the onServiceConnected method from Service Connection in MainActivity
* Inside this method you can fetch the MyService object which you can perform the task kept inside the service.

UnBounded Service:

* In this the service is not bound to the component of android.
* we can stop the service automatically when the task is completed by calling the method stopself() after task is completed.

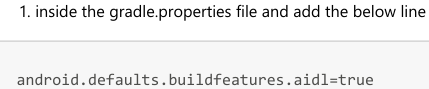
 





AIDL (Android Interface definition Language):

It lets you define the programming interface that both the client and service agree upon in order to communicate with each other using interprocess communication (IPC).



Inside the AIDL Provider add the aidl folder and an aidl interface

package aidl;

interface IRemoteService{

void downlaodFile();

}

Implement this interface inside your service class and return its stub/binder object from onBind method.

private IRemoteService.Stub stubBinder = new IRemoteService.Stub() {

@Override

public void downlaodFile() throws RemoteException {

new Thread(new Runnable() {

@Override

public void run() {

for (int i = 0; i < 10; i++) {

try {

Thread.sleep(1000);

Log.e(tag,"Counter - "+i);

}

}

}

}

}).start();

}

};

@Override

public IBinder onBind(Intent intent) {

// return the object of your aidl interface that is implemeted above

// it will be received inside main activity in service connection

Log.e(tag, "onBind()");

return stubBinder;

}

Inside MainActivity in Service connection object we can get the object of the aidl interface inside it

using the below method.

IRemoteService remoteService;

remoteService = IRemoteService.Stub.asInterface(service);

In second app that is AIDL Consumer we need to use the Provider service. In this inside manifest add the queries tag and add the intent.

<queries>

<intent>

<action android:name="myservice.AIDL"/>

</intent>

</queries>

Copy paste the aidl interface from the AIDL provider and add it inside Consumer app.

Inside the service connection get the obejct. Implement the startService onclick where we need to create the explicit intent from our implicit intent and start the service.

Intent intent = new Intent("myservice.AIDL");

//convert implicit intent to explicit intent

PackageManager packageManager = getPackageManager();

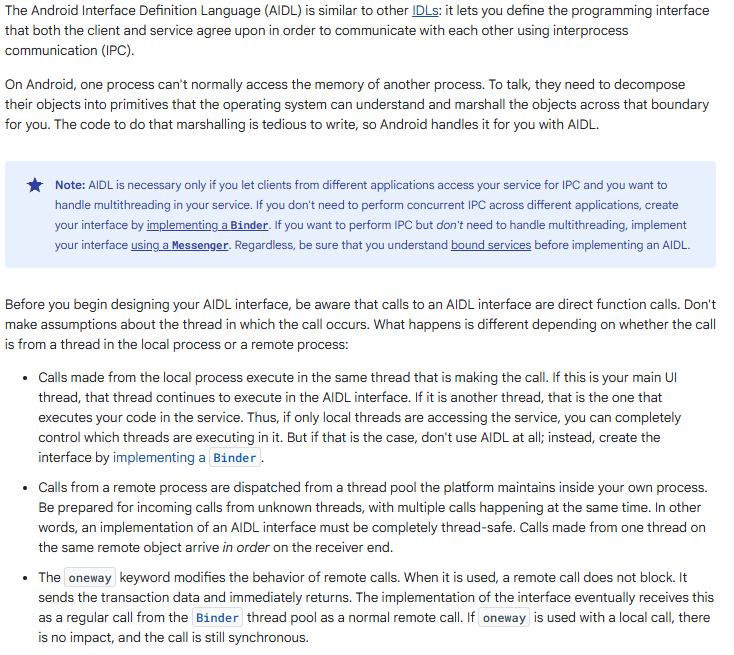
ResolveInfo resolveInfo = packageManager.resolveService(intent,0);

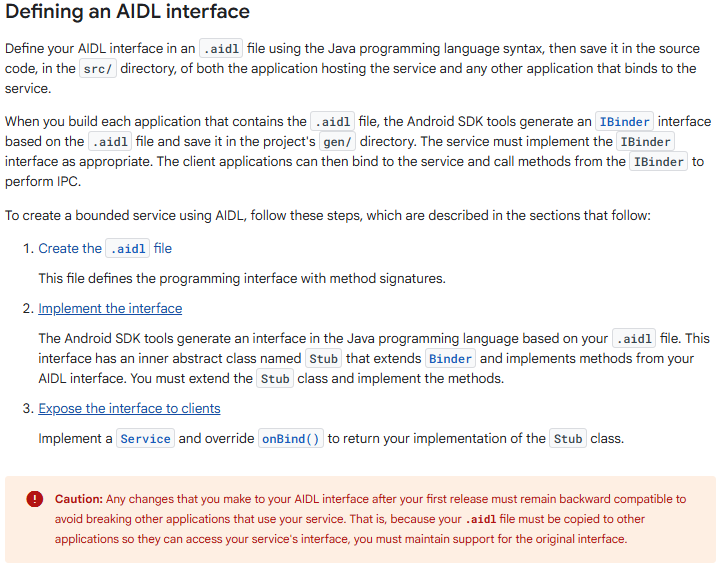
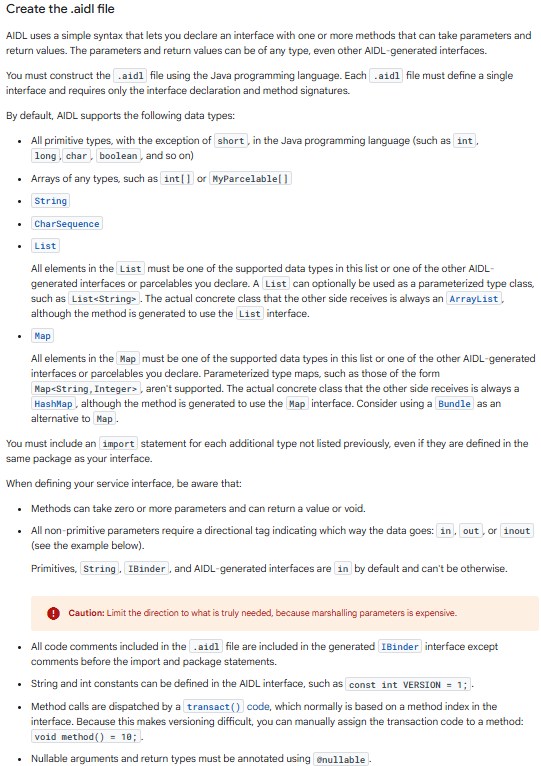
ComponentName componentName = new

ComponentName(resolveInfo.serviceInfo.packageName,resolveInfo.serviceInfo.name);

intent.setComponent(componentName);

bindService(intent, serviceConnection, BIND\_AUTO\_CREATE);



**BroadCastReceiver:**

It is the fundamental component of android Android apps send and receive broadcast messages from the Android system and other Android apps The system and apps typically send broadcasts when certain events occur. For example, the Android system sends broadcasts when various system events occur, such as system boot, device charging,Screen On Off,etc. Apps also send custom broadcasts, for example, to notify other apps of something that might interest them (for example, new data download). Apps can register to receive specific broadcasts. When a broadcast is sent, the system automatically routes broadcasts to apps that have subscribed to receive that particular type of broadcast. Broadcasts can be used as a messaging system across apps and outside of the normal user flow. However, you must be careful not to abuse the opportunity to respond to broadcasts and run jobs in the background that can contribute to a slow system performance. The system automatically sends broadcasts when various system events occur, such as when the system switches in and out of Airplane Mode. All subscribed apps receive these broadcasts. The Intent object wraps the broadcast message. The action string identifies the event that occurred, such as android.intent.action.AIRPLANE\_MODE.

The intent might also include additional information bundled into its extra field. For example, the Airplane Mode intent includes a boolean extra that indicates whether or not Airplane Mode is on. To get the state of Screen lock and unlock and to get the status of the bluetooth on and off.

For this add the permission of bluetooth in the maifest file:

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

The intent filter to be dynamically added in main activity.

MyReceiver receiver;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

receiver = new MyReceiver();

IntentFilter intentFilter = new IntentFilter();

intentFilter.addAction(Intent.ACTION\_SCREEN\_ON);

intentFilter.addAction(Intent.ACTION\_SCREEN\_OFF);

intentFilter.addAction(BluetoothAdapter.ACTION\_STATE\_CHANGED);

registerReceiver(receiver,intentFilter);

}

**Android Permissions:**

to ask for multiple permissions at same time use the below code:

if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ\_PHONE\_STATE)

==

PackageManager.PERMISSION\_GRANTED && ContextCompat.checkSelfPermission(this,

Manifest.permission.READ\_CALL\_LOG)== PackageManager.PERMISSION\_GRANTED){

//do the task

}

else{

requestPermissions(new String[]{Manifest.permission.READ\_PHONE\_STATE,

Manifest.permission.READ\_CALL\_LOG}, 1);

}

@Override

public void onRequestPermissionsResult(int requestCode, @NonNull String[]

permissions, @NonNull int[] grantResults) {

super.onRequestPermissionsResult(requestCode, permissions, grantResults);

if (requestCode == 1)

if (grantResults.length > 0 && grantResults[0] ==

PackageManager.PERMISSION\_DENIED)

Toast.makeText(this, "both permissions are required",

Toast.LENGTH\_SHORT).show();

else

//do the task

}

**Location Based Services:**

Use the location services to display the current location It will be displayed as latitude and longitude in the logcat display the latitude and longitude in textviews and add a button that opens the map Permissions required are:

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

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"/>

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION"/>

To get the location, internet shoud be on, location should be on. use the location manager to get the location service, add a location listner and register this to listen to location changes.

LocationManager locationManager;

locationManager = getSystemService(LocationManager.class);

LocationListener locationListener = new LocationListener() {

@Override

public void onLocationChanged(@NonNull Location location) {

latitude = location.getLatitude();

longitude = location.getLongitude();

Log.e("MainActivity", latitude + "," + longitude);

textLatitude.setText("Latitude - "+latitude);

textLongitude.setText("Lobgitude - "+longitude);

if (ActivityCompat.checkSelfPermission(this,

android.Manifest.permission.ACCESS\_FINE\_LOCATION) !=

PackageManager.PERMISSION\_GRANTED && ActivityCompat.checkSelfPermission(this, android.Manifest.permission.ACCESS\_COARSE\_LOCATION) !=

PackageManager.PERMISSION\_GRANTED) {

requestPermissions(new String[]{Manifest.permission.ACCESS\_FINE\_LOCATION,

Manifest.permission.ACCESS\_COARSE\_LOCATION}, 1);

return;

}

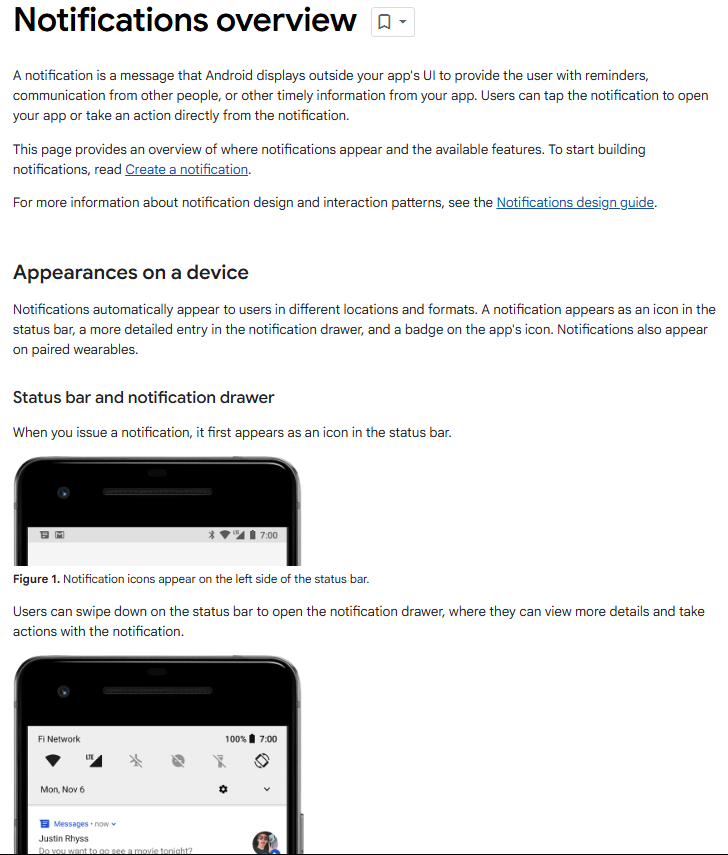
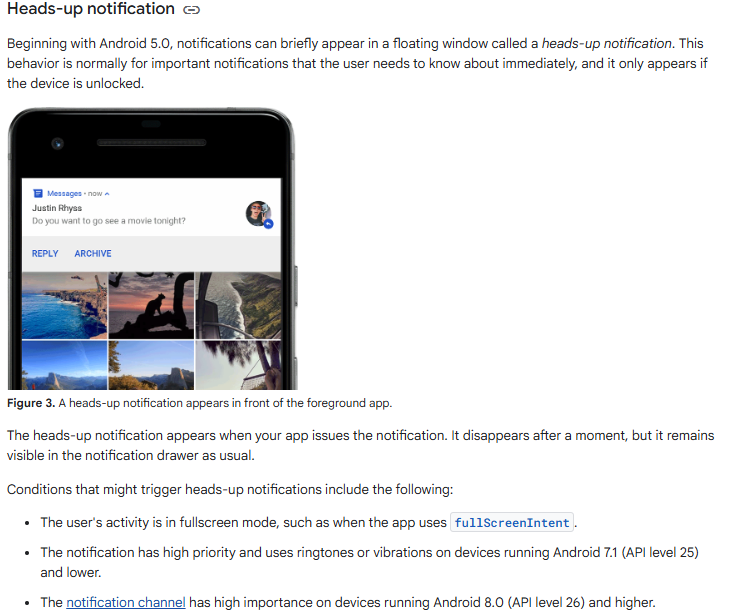
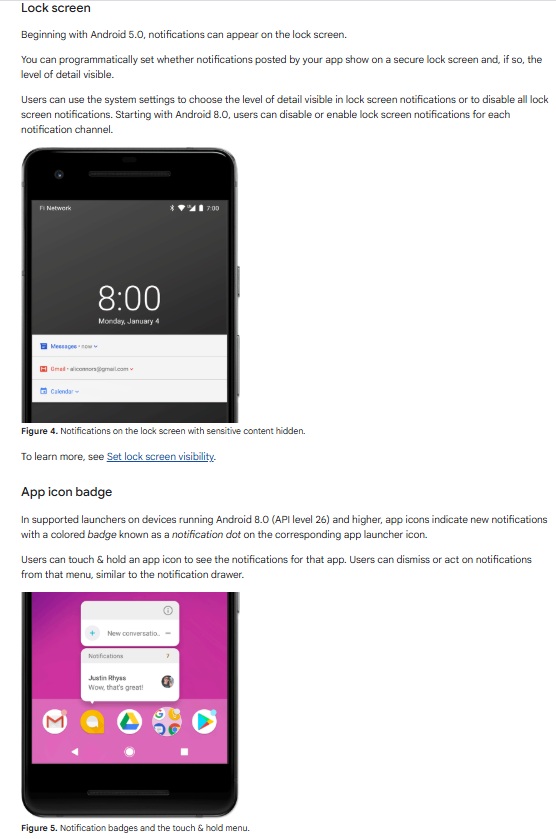
locationManager.requestLocationUpdates(LocationManager.NETWORK\_PROVIDER, 0, 0, locationListener);

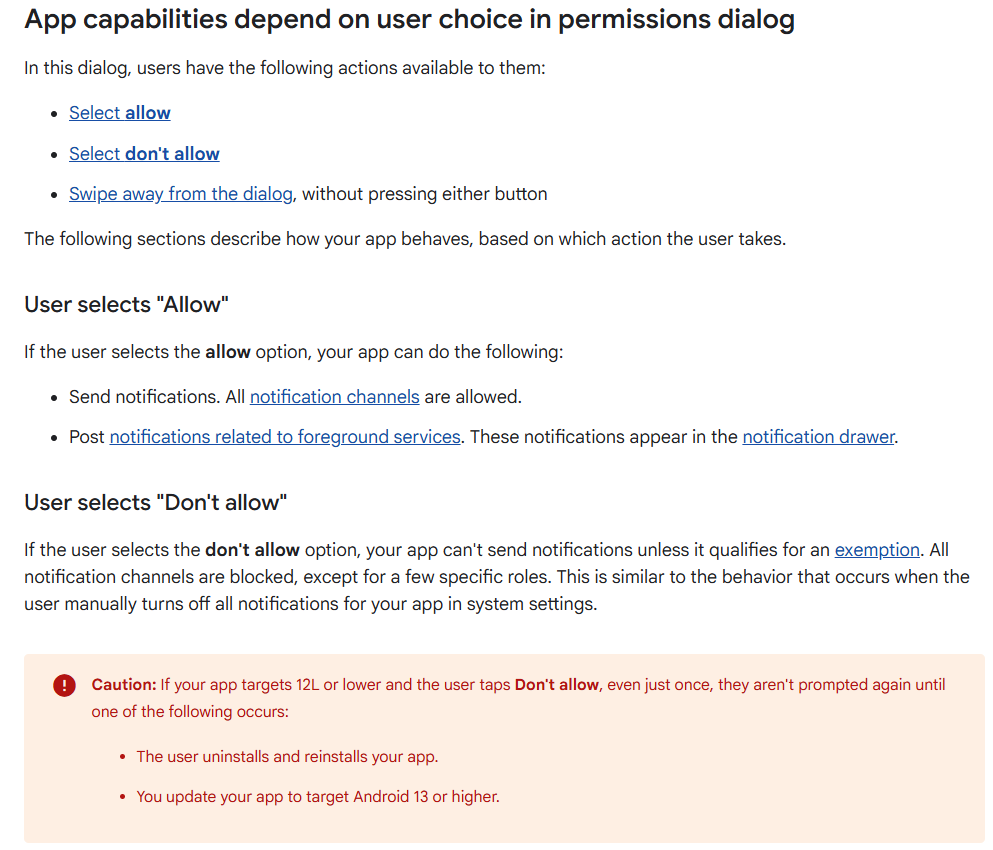
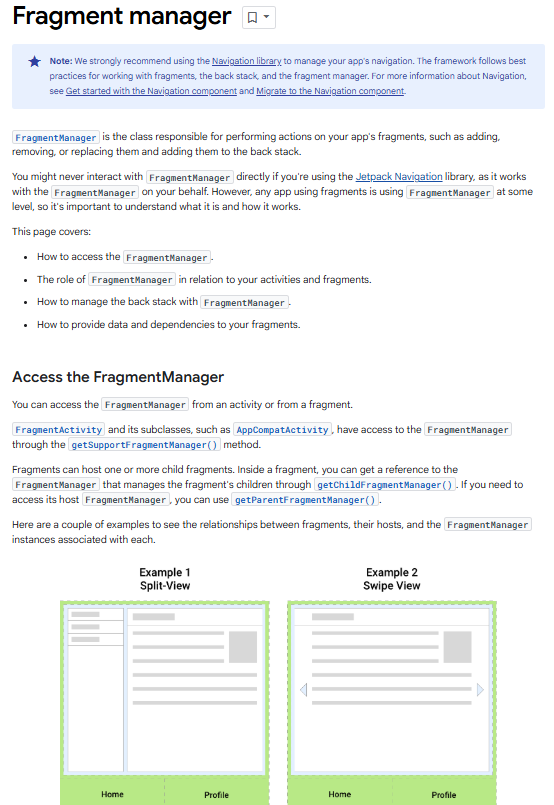
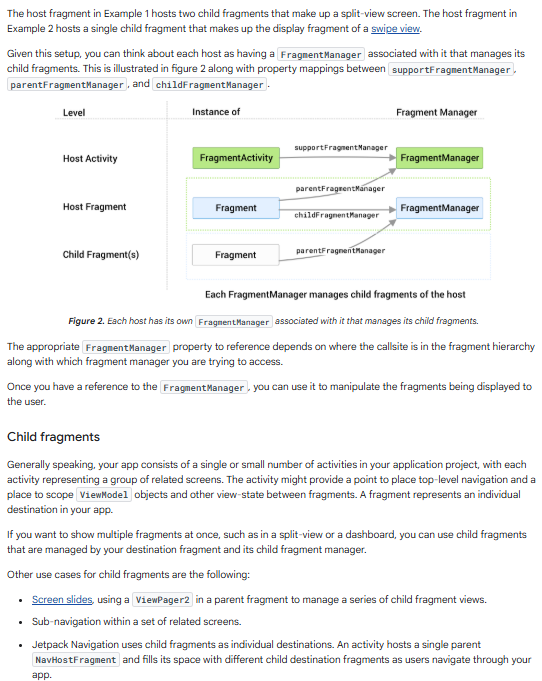
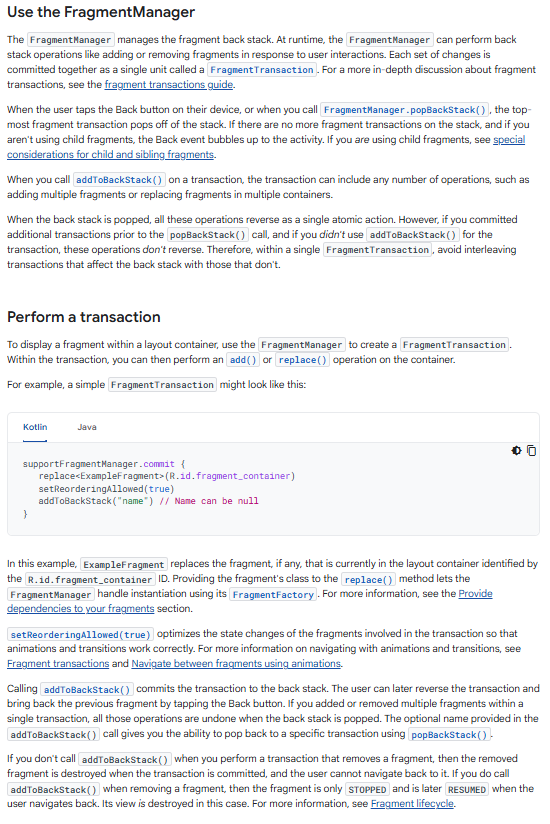
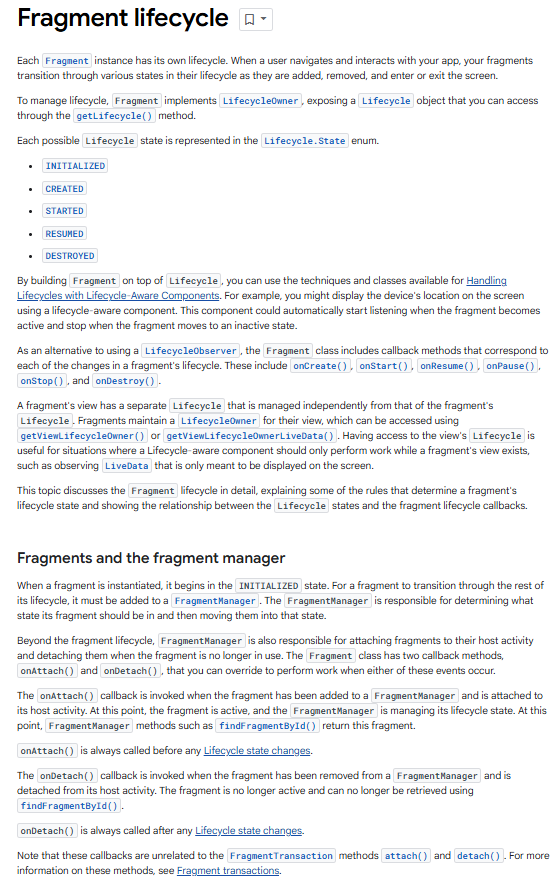
Add a button to open the maps application. create an intent, add the uri and start the activity

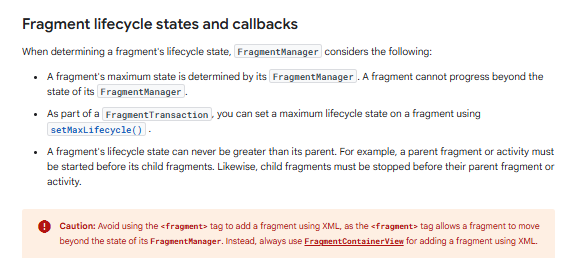
Uri uri = Uri.parse("https://www.google.com/maps/place/"+latitude+","+longitude);

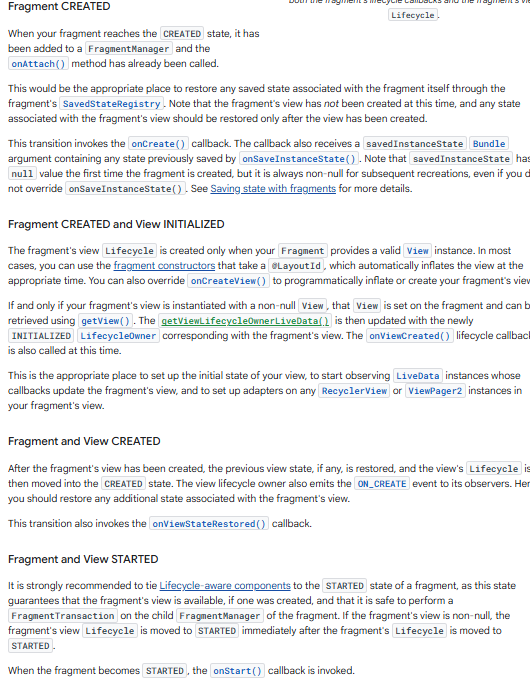
Intent intent = new Intent(Intent.ACTION\_VIEW,uri);

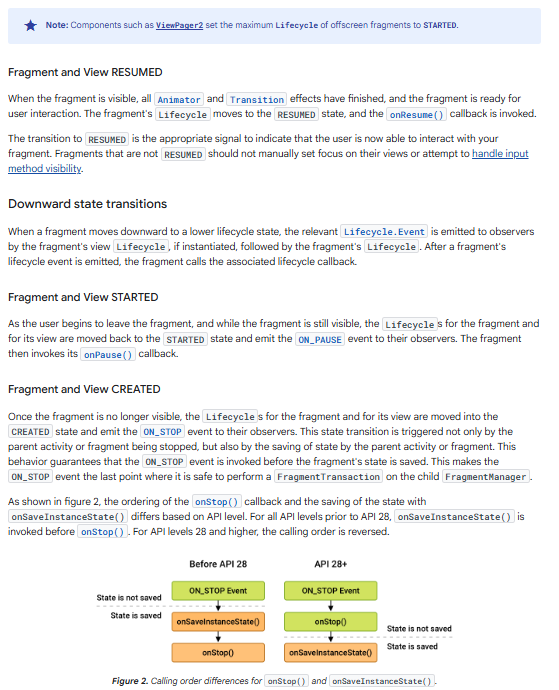
startActivity(intent);

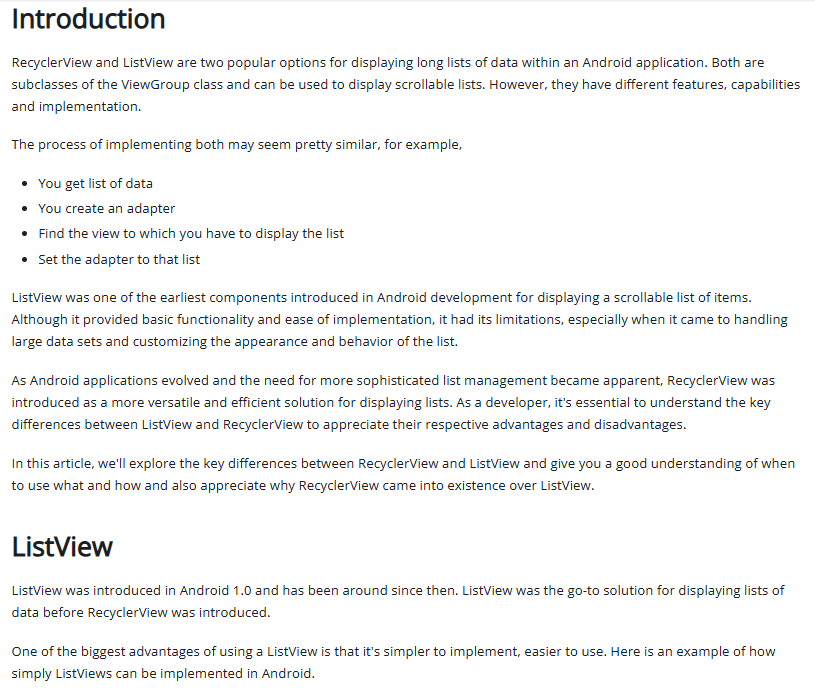
**** ****    

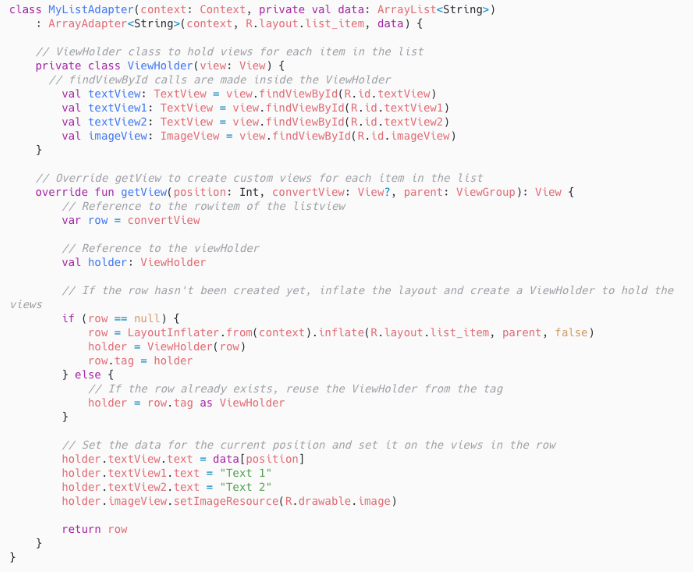
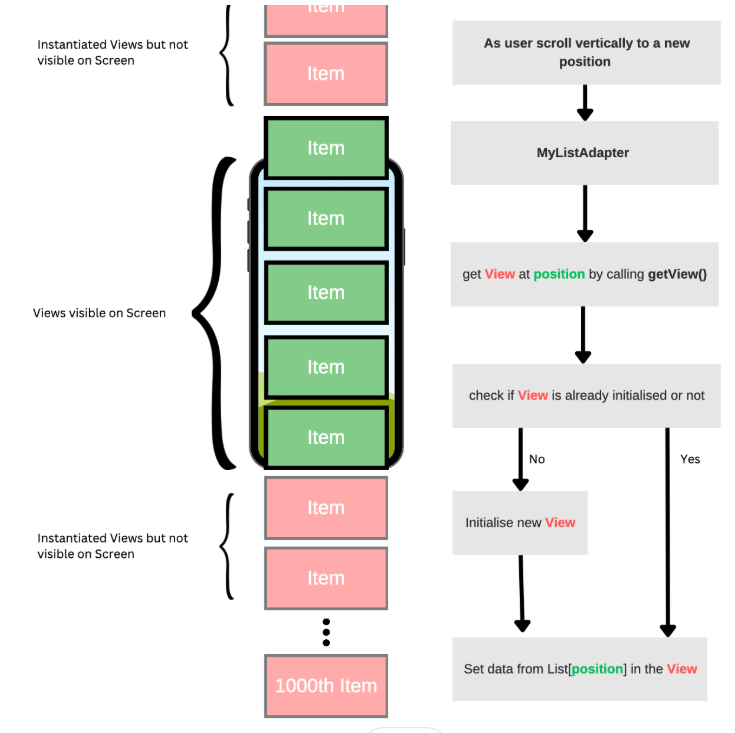
**** fragment lifecycle states and their relation both the fragment's
            lifecycle callbacks and the fragment's view lifecycle

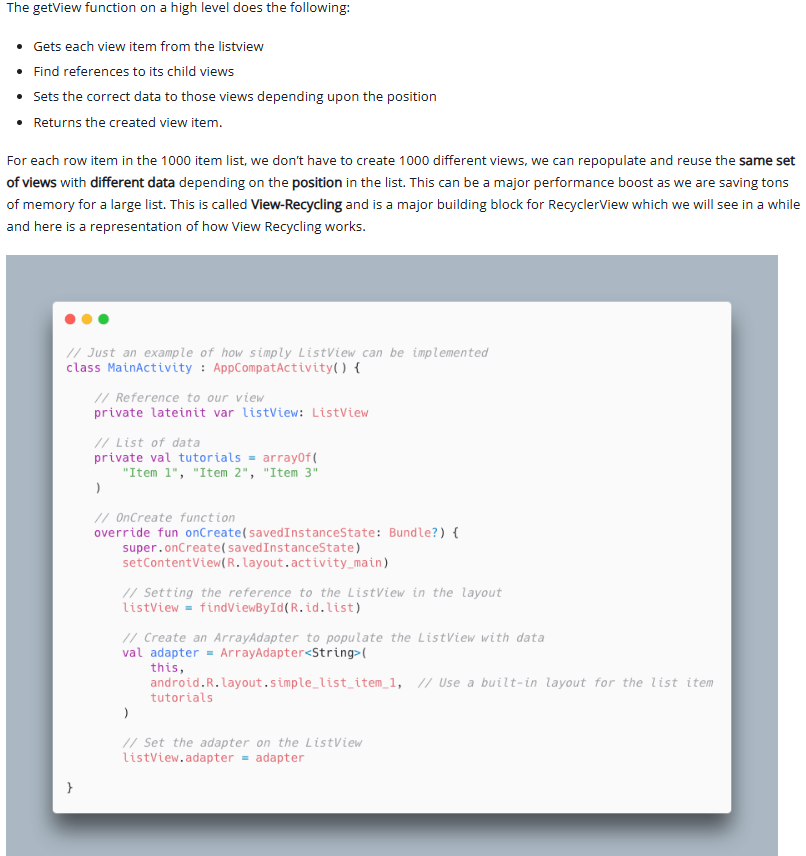
****

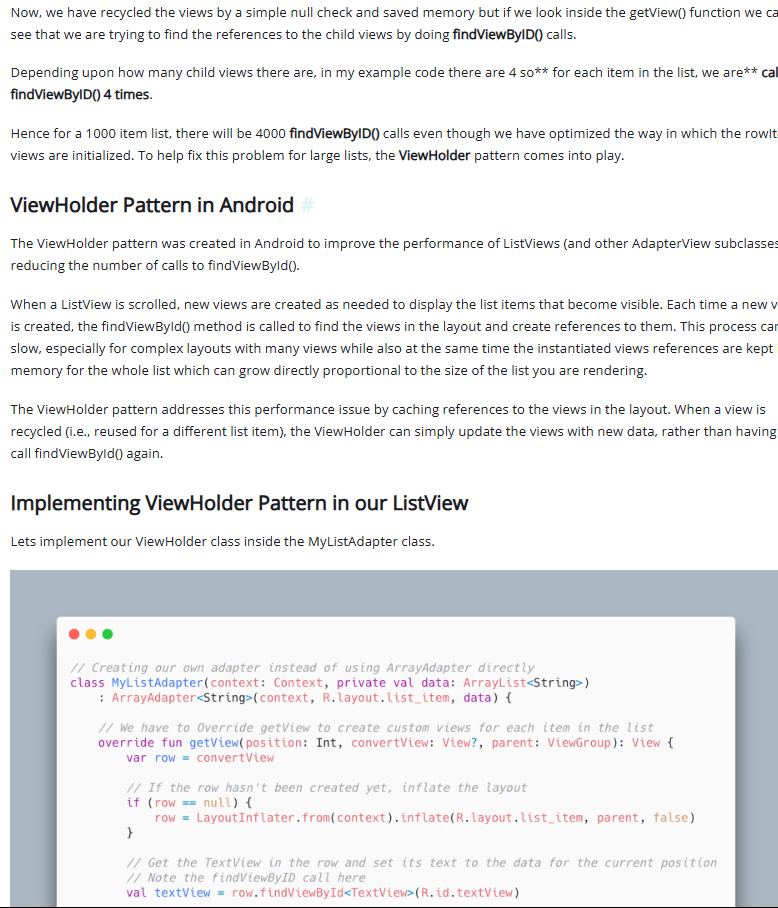
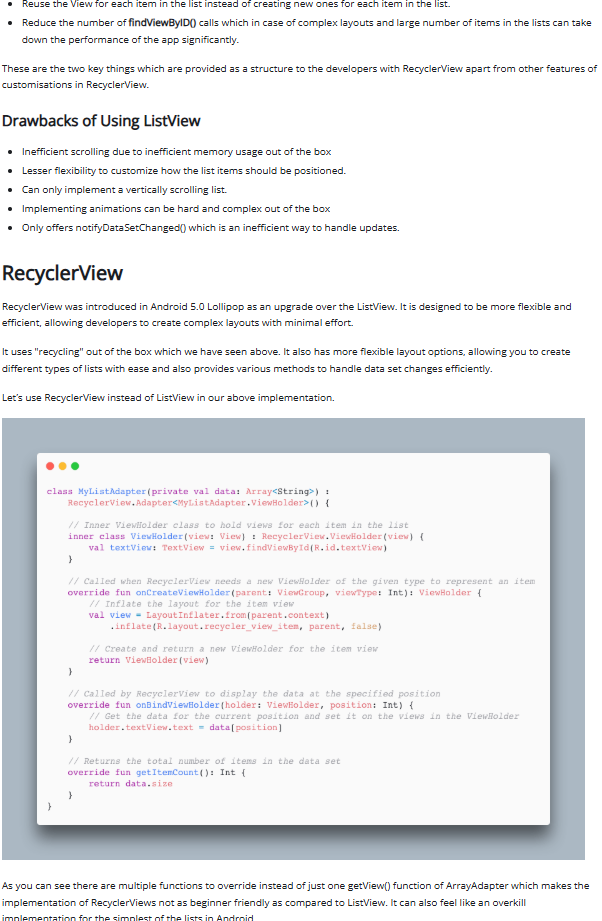
****

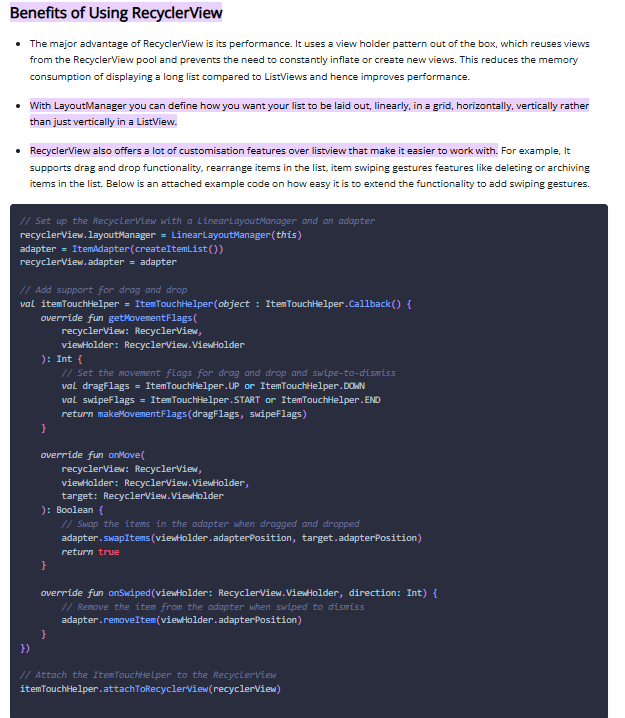




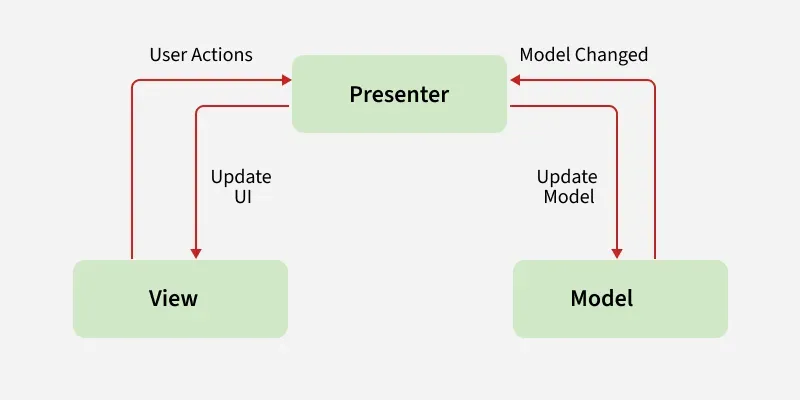
 

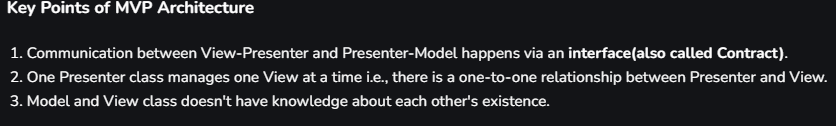


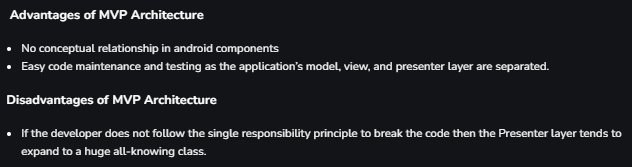
 

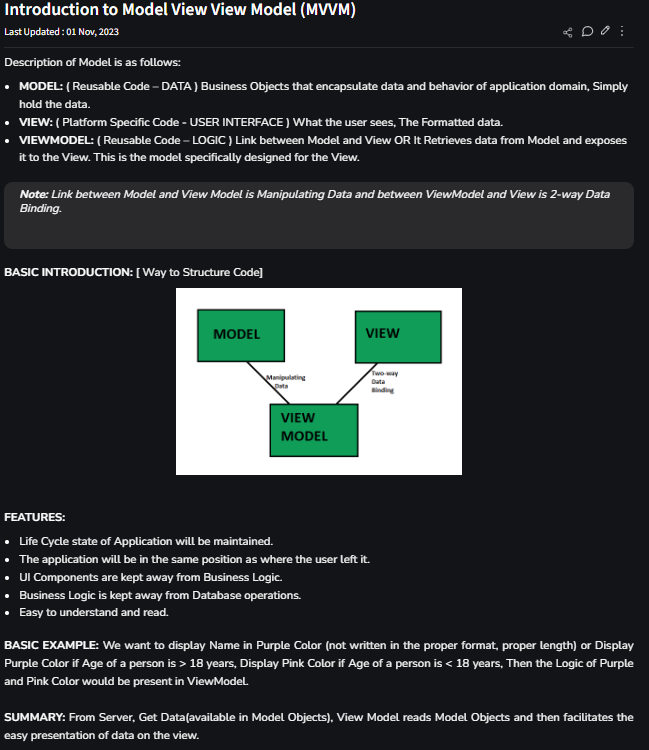


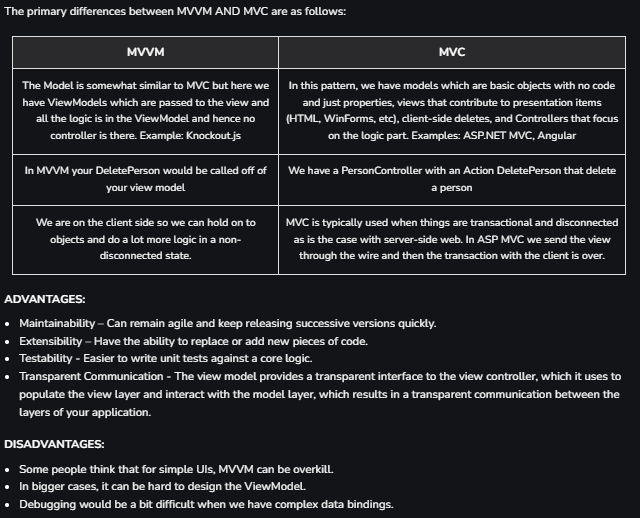












<https://www.geeksforgeeks.org/how-to-publish-your-android-app-on-google-play-store/>