Why to develop mobile application

Mobile applications are popular because smartphones gives users constant internet access, making it easy to stay connected anywhere. for example— browsing social media, access bank account, or do online shopping etc. These days there are billions of smartphone users worldwide, which provides a huge opportunity for the developers to create apps that can be used globally. It can also generate revenue through in-app purchases, ads and subscriptions.

Example → Candy crush app make money by in-app purchases and ads.

History of android platform

Android was initially created by Android Inc.

as an OS for digital cameras but was later transformed into an OS for smartphones in 2024. In 2025, google took over the project and decided to base it on Linux, an open source operating system. Initially, android apps were developed by using Java. but recently Kotlin, another programming language has been introduced as an alternative.

Android platform architecture

Android architecture is made up of different layers, such as the Linux Kernal, Hardware Abstraction Layer (HAL), Libraries, Android Runtime, and Application Framework, which allows developers to build apps.

Application building blocks

① Activity: An activity represents a single screeen in an app. for example, when you open the "Contacts" app, the screeen that shows all the contacts is an activity.

② Intent: Intents allow differcent components of the app to communicate. forc instance, when you click on a link in an email, the system uses an intent to open the web browserc.

③ IntentReceiverc: These arce used forc responding to external notications orc alarcms. forc example, a weatherc app might use an IntentReceiverc to alerct about a storcm.

④ Service: Services run in the background without any userc interface. forc example,

a music app running in the background and GPS navigation app running while driving.

⑤ ContentProviders: This component helps share data between different apps. For example, the contacts in your phone's address book can be shared with other apps.

⊛ Real-life example of components

1. Traffic navigation app: Uses activity (to show maps), Service (Continuously update your location) and ContentProviders (to access the map data).

11. Alarm App: Uses service (to keep the alarm running in the background) and the IntentReceivers (to respond to ~~extern~~ external triggers like a time change)

Notepad App: Uses activity (for each note) and ContentProviders (to store and retrieve notes).

Application life cycle

Understanding how an app behaves during different stages (when it starts, stops, pauses etc). ~~For~~ It is crucial for managing resources and creating a smooth user experience. for example, if a user switches between apps, the android system may pause the app and resume it later, which can affect its functionality.

Development tools

1. Android SDK: (Software Development Kit)
2. Android Studio: This is the official integrated development environment (IDE) for android development

Views and ViewGroups

① Views: Views are the UI elements like TextViews (for text), EditText (for input), Buttons (for interactions), and ImageView (for images)

② ViewGroups: These are containers that hold views. Example,

i. LinearLayout: Arranges views in a straight line (horizontally or vertically)

ii. RelativeLayout: Allows to position views relative to one another.

iii. GridLayout: Arranges views in a grid of rows and columns.

Using emulators and testing

① Emulators simulate a mobile device on the computer for testing purposes. They

e some limitations like not supporting features like a real camera or GPS.

② Testing on a physical device is essential to get an accurate idea of the app's performance

Android user interface

The user interface (UI) in Android is like the architecture of a building. It defines the structure and layout of the elements visible to the user in an app's activity. Example → Phones home screen.

How to declare a layout

① Option 1 (Xml) : The most common method where we define the UI elements and layouts in Xml files.

② Option 2 (Java) : Create UI elements programmatically using Java code at run time.

Advantages of declaring UI in Xml

① Separation of presentation from code: You can modify the UI without changing the underlying code. For example, you can switch from a vertical layout to a horizontal one without touching the logic of the app.

② Easier to visualize the UI structure: Xml is straightforward to visualize and debug, and tools like Android Studio provide a UI visualizer to design layouts visually.

UI components in layout

① ViewGroup: A container that holds child views and determines their positioning. Examples — LinearLayout, RelativeLayout.

② View: Represents UI elements such TextView, Button etc. Example: "Welco... 'DELETE' button in a calculator to take delete action to perform.

Layouts ①

Attributes vs. Parameters

ID (attribute)	Layout para. (layout attribute)
① The ID is used to uniquely identify a view or view group. Its set using the "android:id" attribute in xml. Example: <Button android:id= "@+id/my_button" /> xml	① These describe how a view should behave inside its parent ViewGroup. Example: layout_height, layout_width, layout_margin etc tells the view to fil the available width of its parent. "match_parent"

2. Layout types

Layouts are subclasses of ViewGroup.

① LinearLayout: Aligns children in a single direction (horizontally or vertically). Ex- A vertical list of buttons where each button takes up one row.

② TableLayout: Positions children in rows and columns. Each row can contain multiple views and colums can adjust to fit the content. Ex- A form where contents are arranged in rows and columns.

③ GridLayout: Divides the screen into a grid of cells, here each child view ~~occupies~~ occupies one or more cells.

Ex- A calculator app.

④ RelativeLayout: Lets children specify their position relative to other views or the parent layout. Ex— Aligning a "Submit" button below a "TextBox" on a screen.

⑤ FrameLayout: A basic layout where views are stacked on top of each other, useful for a single view or swapping content dynamically. Ex— A photo gallery.

⑥ ConstraintLayout: A more flexible layout where you can define relationships between views using constraints (similar to RelativeLayout but more powerful).

"Slide-02"

What is Android User Interface (Layout)?

→ The user interface is how users interact with the app visually. In android, UI consists of layouts and widgets. Layout is the architectural structure defining how all the elements are going to be displayed. It mainly provide the design structure of an activity and organizes all the elements to appear properly on the screen. On the other hand, widgets are individual components (such as buttons, text views, images etc).

How to declare a layout?

⇒ Option: 01 (Declare UI in XML file)

Android provides a straightforward XML vocabulary to declare UI components such as,

- Layouts: LinearLayout, RelativeLayout etc
- Widgets: TextView, Button etc

In XML, UI design is separate from the logic written in Java/Kotlin. Updation is easy here without changing or recompiling Java code. XML files can be reused for diff screens, orientations and devices.

```
<LL
    lw = "match parent"
    lh = "match parent"
    orientation = "vertical" >

    <TV
        id = "@+id/text"
        lw = "wrap content"
        lh = "wrap content"
        text = "Hello World!" />

    <Button
        id = "@+id/button"
        lw = "wrap content"
        lh = "wrap content"
        text = "Geo" />

</LL>
```

Option : 02 (Declare layout in Java)

UI components likes LL or RL etc : are created programmatically in Java or Kotlin. Properties like size and alignment are adjusted in the code. It requir Dynamic UI creation based on user interactions or runtime conditions.

```
LinearLayout linearLayout = new LinearLayout(this);
linearLayout.setOrientation(LinearLayout.VERTICAL);

TextView textView = new TextView(this);
textView.setText("Hello World!");

Button button = new Button(this);
button.setText("Geo");

linearLayout.addView(textView);
linearLayout.addView(button);

setContentView(linearLayout);
```

Benefits of declaring layouts in XML

⇒ 3 main facts are given below,

① Separation of presentation and logic :-

- UI is defined in XML while behavior is coded in Java / Kotlin
- This modular approach simplifies debugging and updation.

② Easier to adapt for different configurations :-

Can create alternative XML files for,

- Portrait and Landscape files.

- Different screen sizes.
- Localization (languages).

③ Improved visualization :—

- Android Studio provides a design editor for visualizing and editing XML layouts.

Where to save layout files?

→ Layout XML files are saved to "res/layout" directory of the project. Example: signup_activity.xml

How to load a layout XML resource?

→ When you compile the application, each XML layout file is compiled into a view resource. By using the "setContentView" in the "onCreate" method helps to load the layouts.

```
public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_signup);
}
```

Components of a layout

→ Layouts consist of ViewGroup and View

- ViewGroup: Acts as a container for multiple views. It determines the position and alignment of child views.
- View: Represents individual components like TV, Button, ImageView etc.

Attributes in layouts

⇒ Attributes control the appearance and behavior of views and ViewGroups. They includes

① View specific attribute: "textSize", "textColor" etc are specific to views like TV.

② Common attribute: Attributes like "id" are common to all views.

③ Layout parameters: It defines how a child view is placed within its parent viewGroup. Example: lw, lh, lwe etc

- → mp: View expands to fill the parent container.
- → wc: View adjusts its size to fit the content.
- → lwe: Specifies proportional space distribution (LL)

ID attribute

→ XML file এ unique identifier declare করে Java file এ এটা refer করা হয়।

XML:

```
android:id="@+id/my_button"
```

Java:

```
Button myButton = findViewById(R.id.my_button);
```

Types of layouts

→ Android offers several types of layouts, each designed for specific purposes,

① LinearLayout :-

- Aligns children in a single direction (vertical and horizontal)
- Key attributes,
  - → orientation : verr., hori.
  - → layout_weight : same as prev.

⊛ All children are stacked one after the other, so a vertical list will have only one child per row, no matter how wide they are.

② TableLayout :-

- Organizes children into rows and columns.
- Each row is "TableRow", which can contain multiple Views

⊛ যদি row অনুভূমিক fit করার জন্য column stretch or shrink হয়।

③ GridLayout :-

- Arranges Views in a grid structure.
- Use attributes like "rowCount", "columnCount" etc

④ RelativeLayout :-

- Places Views relative to each other or the parent containers. Example; android:layout_below="@+id/textView"

⑤ FrameLayout :-

- Simplest layout; Views are stacked on top of each other.
- Newer Views overlap older ones unless transparency is applied.

⑥ ConstraintLayout :-

- Most flexible and efficient layout.
- Uses constraints to define the position relative to other Views.
- Supports features like biasing for proportional placement.

[1] Differences between layouts

→

① LL vs. TL

- LL arranges views in a single direction.
- TL » » » rows and columns.

② RL vs. FL

- RL arranges views related to each other.
- FL stacks the views.

③ RL vs. CL

- CL is more flexible, provides more efficiency and simpler design tools.

* In Android, XML defines the static layout structure, specifying UI elements and their properties, while Java adds dynamic behavior by modifying these elements in runtime. For example, a "Tv" and "Button" can be defined in XML and Java code can update the text of "Tv" when the button is clicked. This combination ensures a clear, reusable design (via XML) and flexibility for runtime updates (via Java) making it ideal for interactive and adaptive UI's.

"Slide-03"

Activity

→ Activity is a core component of an Android application that represents a single screen with which users can interact. It is the main entry point for the user interaction. An application can have one or more activities. Each activity provides a window where the app draws its user interface.

* Key points:

- Activities as entry points: When you launch an app, the main activity is the first screen users see. From there other activities can be launched.
- UI hosting: Activities host the UI components defined in XML files or programmatically.

• Communication between activities: Activities often communicate with one another using Intents.

Activity Life Cycle:-

1. onCreate() : Initialize an activity.
2. onStart() : The activity is visible to the user.
3. onResume() : The activity is ready for user interaction.
4. onPause() : Partially obscured by another activity.
5. onStop() : The activity is no longer visible.
6. onDestroy() : The activity is being destroyed, and here the system free all the resources.

Create activity

→ Activity create করার জন্য first এ একটা Java class declare করবো। (Java class extends Activity base class)

→ ঐ Java class এর ভিতর UI components load করবো by using the XML file (setContentView(R.layout.activity_name)

→ Application এর প্রতিটি Activity অবশ্যই Manifest এ declare করতে হবে।

✳ Navigation between activities

Back stack in Android works like a stack of plates in a criteria. Each new activity you open is added to the top of the stack. When you press the Back button, the plate (activity) on the top is removed (destroyed), and the plate (activity) below it becomes visible again (resumes).

for example,

1. Starting with activity A (Only the plate in the stack)
2. Then opening activity B (The stack, A→B)
3. Now opening activity C (The stack, A→B→C)
4. Pressed back button, C got removed (destroyed) and B got resumed (The stack, A→B)

✳ Splash / Loading Screen

A splash screen is the first screen users see when they open an application. It acts as an introductory screen, typically displaying the app's logo, name or branding elements while the app loads its resources in the background. It creates a engaging environment and

a professional first environment, prevents blank screens during the loading process and enhance the overall user experience.

first এ আমরা splash class খুলে তার ভেতর onCreate() দিয়ে activity create করবো, Then এ onCreate() এর ভিতর setContentView(R.layout.activity_splash) দিয়ে UI components load করবো, তারপর Intent দিয়ে navigate করে দিবো, যেমন: splash এর পর Main Screen (MainActivity) আসবে,

Code এ "handler" 3 second পর্যন্ত delay করায়,

* Sequence of activity life cycle

→ onCreate()
Activity create হবে, এর ভিতর সব UI elements load করবে,

→ onStart()
Activity টা visible হবে, onStart() এর পরপরই onResume() call হবে,

→ onResume()
Activity টা user interacting হবে,

→ onPause() Resume ↔ Pause ↔ Stop

২টা scenario তে call করা হয়,

① নতুন activity call করে recent এ যখন back এ send করা হয়।

② Back button press করে activity টা destroy করে ফেলল।

কাজ,

① কোনো service/thread stop করাতে।

② Memory free করাতে (যখন কোনো activity foreground এ নাই)

→ onStop() Restart ↔ Stop ↔ Destroy

Called when the activity is no longer visible. End of the current visible lifespan of the app.

→ onRestart()

যখন কোনো activity stop করার পর আবার use করতে চাই।

* What should we do in onRestart?

→ This method is called when an activity is being restarted after being stopped. You should use this method to re-initialize all the resources.

* If onCreate() is executed, will onRestart() be executed? Why?

→ No, onRestart() will not be executed if onCreate()

is created. is called.

- onCreate() is executed when the activity is created for the first time.
- onRestart() is only called if the activity was stopped and is now restarting (coming back to the foreground)

→ onDestroy()

Called before the activity is destroyed by the system. এই method call করলে activity destroy হবার আগে সব resources free করে দিবে। Once this function is called, there is only one option for transition (other than being killed): onCreate()

Intent

Intents are objects that facilitate communication between components such as activities, services, and broadcast receivers. They help to navigate between screens, send data, and perform system actions. Basically intents are used to share content and trigger actions within and among applications.

An intent Android runtime

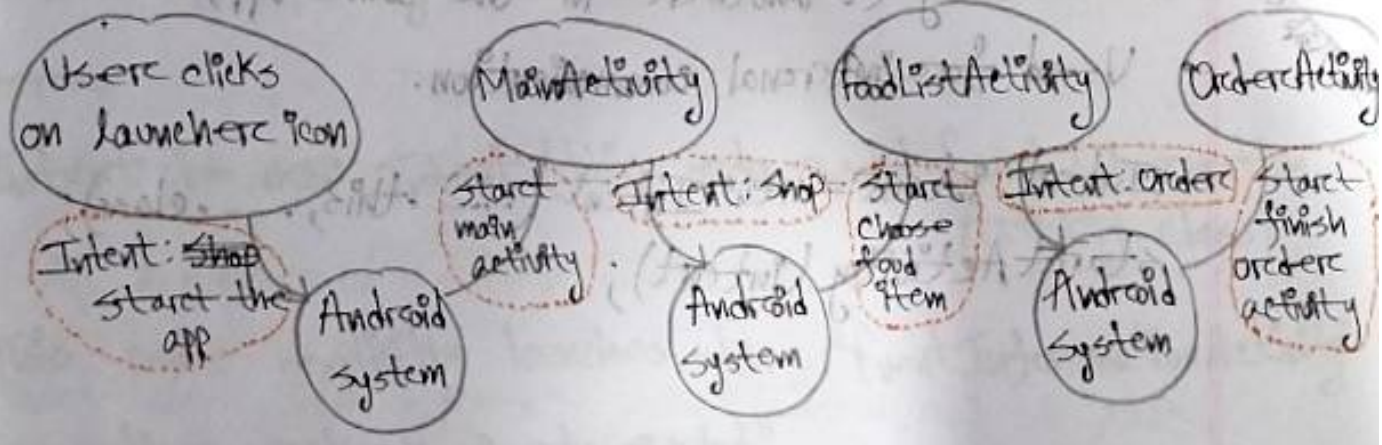
An Intent object carries information that the Android system uses to determines two things,

- Which component to start (Component name/category that should receive the intent) Example: Notification, Alarm etc
- Information that receipient component uses in orderr to perform the action. ~~Example~~ (the action to take/the data to act upon) Example: Notification message, Alarm data (audio) etc

How intents work

- All Intents are managed by the Android runtime
- Started by an "intent", a message to the Android runtime to run an "activity".

Example: Food Page/website/app থেকে order করা।



Building an Intent

- Componant name – To be opened

→ `Intent i = new Intent (SignupActivity.this, LoginActivity.class);`

- Action – To be taken

`Intent i = new Intent (action, uri);`

Opening a webpage,

→ `Intent i = new Intent (Intent.ACTION_VIEW, uri.parse("http://....."));`

`startActivity (i);`

Types of Intent

① Explicit intent

- Directly specify the component to start (from one activity to another in the same app).
- Used for internal communication.

```
Intent intent = new Intent (.... this, .... class);
startActivity (intent);
```

⊛ Extras → Additional information key-value pairs to pass data.

② Implicit intent

- Specify an action instead of the target component.
- Used for tasks like opening a URL, sharing data, or sending an email.

```
Intent intent = new Intent(Intent.ACTION_VIEW);
Intent.setData(Uri.parse("....."));
startActivity(intent);
```

Component Name: How to declare or use

To start a specific component (an activity) in your app, you can use the component name to directly specify the target. A component name includes,

- The fully qualified class name of the target component.
- The app's package name.

Here,

```
Intent i = new Intent(this, edu.ewubd.EventInformation
                                Activity.class);
```

This code explicitly launches the "EventInformationActivity" from the package "edu.ewubd"

Ways to set the Component name

① Using "setComponent"

→ specify the component using a "ComponentName" object, which includes the package and class name

i.setComponent (new ComponentName("edu.ewubd", "edu.ewubd.EventInformationActivity"));

② Using "setClass"

→ Directly specify the target class in the intent.

i.setClass (this, edu.ewubd.EventInformationActivity.class);

③ Using "setClassName"

→ Provide both the package name and class name as strings

i.setClassName ("edu.ewubd", "edu.ewubd.E_I_A");

④ Using the Intent Constructor

→ Can use component name directly in the intent constru

Intent i = new Intent (this, edu.ewubd.E_I_A.class);

Using a component name ensures that the exact target com. is launched, without relying on other parameters like actions. This approach is typically used for explicit intents within the same app.

Action

In android, an action specifies the general task an Intent should perform, such as viewing, dialing, or sending. The action defines how the Intent is structured, including its data and extras. Common actions are given below,

① ACTION_VIEW: opens and displays data for the user.

- Open a photo in a gallery app.
- View an address in a map app.
- Open a URL in a web browser.

```
→Uri uri = Uri.parse("http://www.google.com");
 Intent it = new Intent(Intent.ACTION_VIEW, uri);
 startActivity(it);
```

② ACTION_DIAL: Opens the phone dialer with a specified number. User can call through another app, such as Whatsapp, Imo etc.

```
→Uri uri = ..... ("tel: 01.....");
 Intent it = ..... (Intent.ACTION_DIAL.uri);
 startActivity(it);
```

③ ACTION_SEND: Shareas data like text orc files with otherc apps.

→ Intent intent = new Intent (Intent.ACTION_SEND);
intent.putExtra (Intent.EXTRA_TEXT, "Hello!");
intent.setType ("text/plain");
startActivity (Intent.createChooserc (intent, "Sharce via"));

How to declarce/use an action

- Specify the action for an intent with "setAction()" and an "Intent constructorc"

  intent.setAction (Intent.ACTION_VIEW);

  Intent intent = new Intent (Intent.");

- Including the application package name when defining custom actions in your app ensurces uniqueness and preverents conflicts with actions from otherc apps. This way, the Andrcoid system can distinguish the app's action from standarcd orc third-party actions, maintaining clarity and avoiding unintended behaviorc.

✱ Passing data using "intents" :—

→ For passing data,

```
Intent intent = new Intent (this, SecondActivity.class);
intent.putExtra ("key_name", "value");
startActivity (intent);
```

→ For receive data,

```
String value = getIntent().getStringExtra ("key_name");
```

→ For intent filter,

```
<intent_filter>
    <action  android.name = "android.intent.action.SEND" />
    <category  android.name = "android.intent.category.DEFAULT" />
    <data  android.mimeType = "text" />
</intent_filter>
```

✱ Linking Button and Java code :—

```
<Button
    android.id = "@+id/buttonSubmit"
    android.layout_width = "wc"
    android.layout_height = ""
    android.text = "Submit" />
```

```
Button button = findViewById(R.id.buttonSubmit);
button.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v) {
        Toast.makeText(getApplicationContext(), "Button Clicked",
        Toast.LENGTH_SHORT).show();
    }
});
```

Here, Java code links the button to the app using "findViewById()". An "onClickListener" detects button clicks, and the "onClick()" method displays a Toast message when the button is pressed, enabling user interaction.

"Slide-04"

Data Storage in Android

Android provides multiple options for storing data, depending on the data type and its accessibility.

1. App State Storage →
   - Stores temporary app state data in memory
   - Data is lost when the app is closed.

II. Shared Preferences →
   - Stores primitive data in key value pairs.
   - Lightweight, commonly used for settings like username, password, auto login etc.

III. Internal Storage →
   - Stores private data in the devices internal memory.
   - Data is deleted when the app is uninstalled.
   - Not ideal for large or text base data storage.

IV. External Storage →
   - Stores public data on external storages like SD cards.
   - Files are modifiable by the user.

V. SQLite Database →
   - Stores structured data in a private database.
   - Useful for complex queries.

VI. Network Connection →

- Stores data on a web servers or in cloud storage, enabling remote access.

Activity State Management

Android activities can lose their state under below circumstances,

I. When activity is no longer on the screen (Due to free up memory).

II. When the orientation is changed

These 2 points can cause the activity to be destroyed and recreated.

How to manage state,

1. Storing State Temporarily → Use "onSaveInstanceState (Bundle)" to save temporary data.

```
@ Override
public void onSaveInstanceState (Bundle outState) {
    outState.putString ("someKey", textView.getText().toString());
    super.onSaveInstanceState (outState);
}
```

11. Restoring State → Use "onRestoreInstanceState(Bundle)" to retrieve saved data.

```
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState){
    if(savedInstanceState != null){
        String value = savedInstanceState.getString("somekey");
        textView.setText(value);
    }
}
```

Shared Preferences

- Overview → Pref. page
- Storage Location → Stored as an XML file in "/data/data/<package_name>/shared_prefs/"
- Modes →
  1. Private mode (MODE_PRIVATE) : Accessible only by the app.
  11. Public mode (MODE_WORLD_READABLE / MODE_WORLD_WRITEABL
  Deprecated, allow access across the app.
- Usage →
  1. Storing Data:

```
SharedPreferences prefs = getSharedPreferences("myPrefs",
                                    MODE_PRIVATE);
  » .Editor editor = prefs.edit();
editor.putString("username", "D-Pink");
editor.putInt("loginCount", 1);
editor.apply();
```

II. Retrieving Data:

```
SharedPreferences prefs = getSharedPreferences("myPrefs",
                                    MODE_PRIVATE);
String username = prefs.getString("username", " ");
int    loginCount = prefs.getInt("loginCount", 0);
```

III. "getPreference()" is used for the preferences that are limited to a specific activity.

Internal Storage

- Overview → Prev. Page
- Usage →

1. Writing to file:

```
FileOutputStream fos = openFileOutput("myfile.txt", MODE_
                                    PRIVATE);
fos.write("Hello!".getBytes());
fos.close();
```

II. Reading from file:

```
FileInputStream fis = openFileInput("myfile.txt");
int content;
while ((content = fis.read()) != -1) {
    System.out.print((char) content);
}
fis.close();
```

- files can be created in append mode, allowing new data to be added without overwriting existing content.

External Storage

- Steps to Access →
  1. Check availability: Environment.getExternalStorageState()
  2. for app specific file: getExternalFilesDir()
  3. for public file: getExternalStoragePublicDirectory()

SQLite Database

- Overview → Prev. page
- Key Classes →
  1. SQLiteDatabase: Manages database creation and operations.

II. SQLiteOpenHelper: Simplifies database management and versioning.

- Example →

1. Create Database:

```
SQLiteDatabase db = openOrCreateDatabase("myDatabase.db",
                                    MODE_PRIVATE, null);
db.execSQL("CREATE TABLE users (id INTEGER PRIMARY
            KEY, name TEXT, age INTEGER)");
```

II. Insertion:

```
ContentValues values = new ContentValues();
values.put("name", "Rosa");
values.put("age", 22);
db.insert("users", null, values);
```

III. Query:

```
Cursor cursor = db.rawQuery("SELECT * FROM users", null);
while (cursor.moveToNext()){
    int id = cursor.getInt(0);
    String name = cursor.getString(1);
    int age = cursor.getInt(2);
}
cursor.close();
```

Cloud Storage

- Overview → Allows storing data/files on cloud servers for remote access. Example:—
  1. Google Drive offers 15 GB.
  2. Dropbox offers 2GB.
  3. iCloud offers 5 GB.
- Use Case →
  1. Syncing user data across device.
  2. Storing large files remotely for scalability.

Static file

- Static files can be stored to "res/raw" directory.
- These files are read only files and cannot be modified.

Remote Database

Remote database stores data on a server and communicates with the app via APIs. The app sends queries to the servers, which processes them and returns the results in a format like JSON. This allows the app to access

and manage data stored remotely, without relying on local storage.

- Firebase →

i. Firebase provides a cloud hosted real time database solution.

ii. Supports user authentication and data syne across devices.

Comparison

- **Shared Preferences Vs. SQLite Database**

→ Shared preferences are lightweight. Often uses for username, password, auto login etc. On the other hand, SQLite is used for large datasets

- **Internal Storage Vs. External Storage**

→ Int storage is private to the app. On the other hand, External storage is public for other apps.

→ Requires no permission checks (I storage). Requires permission checks (E storage).

→ Int. storage is used for sensitive data, whereas Ext. storage is used for public data (music, movie, document etc)

- **SQLite Database Vs. Network Storage**

→ SQLite database is local and hence it requires offline access only. Network storage is remote and internet dependent.

→ SQLite is suitable for small to medium datasets whereas network storage is suitable for large scale apps.

- Cloud Storage Vs. External Storage

→ Cloud storage provides remote access across multiple devices. External storage is device-specific and requires/allows offline access.

→ Cloud provides built-in encryption and it supports user authentication. Ext. storage is less secured as any user can access.

Process and Multi Thread

- Program → Program is a set of instructions and data that a computer follows to perform a task.
- Process → Process involves following the program step by step and take appropriate action accordingly. An android system automatically manages all the processes. It creates one when the app starts and terminates it when system resources are low or the app is no longer needed. Processes are killed depending on the "Process Hierarchy".

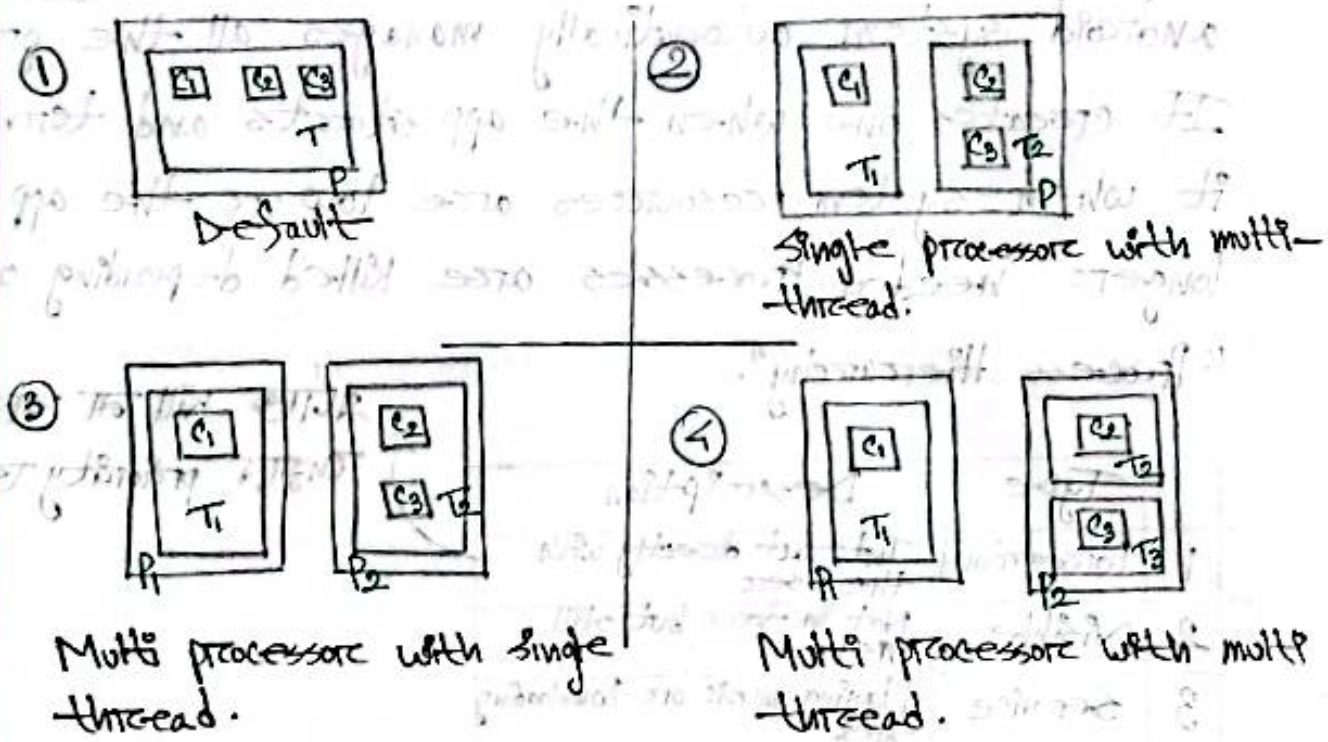
	Type	Description
1	Foreground	Interact directly with the user.
2	Visible	Not in focus but still visible.
3	Service	Playing music or downloading files.
4	Background	Invisible tasks when onStop() called, LRU kill.
5	Empty	Exists in memory to allow app re-launch

→ এটাকে Kill না করা ভালো যেহেতু priority বেশি।

→ এটাকে Kill করা resource free করার priority বেশি (আগে আগে Kill হবে)

- Thread → Thread is a flow of execution inside a process. Each process can have multiple threads. All

android apps start with a single thread called the Main Thread (UI Thread). Threads in the same process share all the memories and resources, making the communication between them efficient but also introducing risks like race conditions.



- Main Thread / UI Thread

→ Role :-

1. Renders UI components.
2. Processes users interactions.
3. Runs lifecycle methods (onCreate, onStart) etc.

→ Importance :-

Since it manages the user interface keeping the UI / Main thread responsive is very critical. If it's blocked or delayed the app becomes unresponsive which leads to an Application Not Responding (ANR) error.

→ Limitations :-

1. The main thread cannot perform long running tasks. Such as: Network operations, Database queries, Complex computations etc.

2. Running such tasks directly on the Main Thread freezes the UI and causes poor user experience.

→ Best Practices for the Main Thread :-

1. Keep operations lightweight and quick.
2. Offload long-running tasks to Worker Threads.

- Worker Thread

A worker thread is created to handle time-consuming operations off the Main Thread, ensuring

the app remains responsive. These threads perform tasks such as,

1. Downloading large files.
2. Processing large datasets.
3. Running computations.

→ Benefits :-

1. Offloads heavy tasks from the Main Thread, avoiding UI freezes.
2. Allows simultaneous operations without blocking user interactions.

→ Limitations :-

1. Worker threads cannot access UI components directly.
2. To update the UI from a Worker Thread, mechanisms like Handlers, Async Task or runOnUiThread() must be used.

program vs. process

→ Program is a set of instructions and data that a computer follows to complete the task. And on the other hand, Process involves following the program stepwise and take action according to it. Process mainly কোনো program এর control flow অনুযায়ী action perform করে যাবে,

Example: | 53 | 2 | 19 | 103 | 8 | 22 | *find out wether 103 exists or not*

1. Sequentially প্রতিটি index search করতে হবে, T হলে value return করে দিবে and f হলে কোনো value return করবে না। Since এখানে, প্রতিটি index check হবে so it's very complicated. Complexity O(N).

2. যদি এমনাবস্থায় আমরা parallel function use করি by addressing the start and end point,

```
Search(start, End){
}
```

(Th1 Th2 Th3)

এখানে search function টা তিন thread এ parallely run হবে just parameter টা আমরা দিয়ে দিব, Then Thread তিন থেকে তিন result আসবে, Value যদি exist করে তো T return করবে otherwise f. Array কে search করার জন্য divide করে দিচ্ছে so it's more faster.

⊛ একটা memory'র মধ্যে array store করা আছে। Multiple threads এ array টা pass করে দিয়ে parameterwise search করাই।

Thread

→ Thread is a flow of execution inside a process. Threads in same process shares the same memory and other resources. একারণেই তখনই আমরা multi thread use করে parrcalely run করাতে পারবো যখন action গুলো independent হবে।

Example: 1. Tea making এ action গুলো পরস্পর dep.
2. Parameter দিয়ে array তে value search ind.

Android processes and thread

→ আমরা যখন কোনো android app build করি by default ইটার একটা Main/UI thread create হয়ে যায়। figure previous page এ আছে।

⊛ Multiple thread এর ক্ষেত্রে একটাকে must Main/UI thread হতে হবে। অন্যগুলো workere thread হিসেবে পরিচিত।

⊛ Workere thread এ UI এর কোনো কাজ করা যাবে না। এখানে long-running tasks হয় (background এ)

MAMUI T. VI.T.

* Server based / Remote connection based কোনো কাজ থাকলে তা must Worker Thread এ complete করতে হবে।

* UI/Main thread এ long-running tasks implement করলে এ সময়টুকু UI responsive থাকবে না, Long-running task টা (Downloading) complete হওয়ার পর Main/UI-thread টা আবার control ফেরত পাবে and again interactive হবে। But if this problem continues, তবে bad user experience create করবে।

* Solution হচ্ছে, long-running task টা কে আলাদা worker thread এ run করা, হলে background এ download হবে and Main/UI-thread এর কোনো impact পরবে না, এটি user responsive থাকবে (click, gesture, scroll etc.)

Process creation and removal

যখন আমরা android system এ কোনো app first launch করি এটি automatically process create করে দেয়। এক্ষেত্রে onDestroy() যদি call না হয় (Suppose Home btn চাপ দিয়ে minimize) তাহলে forcefully process hierarchy অনুযায়ী android system process টা remove করে দিবে। (Memory reclaim করার জন্য)

⊛ নতুন কোনো app বা process কে memory allocate করার time এ যদি RAM এ কোনো memory অবশিষ্ট না থাকে তাহলে তখন Android system hierarchy অনুযায়ী তার stack এ search করবে and তার অনুযায়ী কোন prev process কে kill করে দিবে, (To free up memory)

⊛ একই app এ যদি multiple process থাকে তাহলে LRU (Least Recent Use) Policy অনুযায়ী process kill করে memory free করে দিবে, Same system for multiple app at Background.

Problem : Keeping an app responsive

```
void clickEventHandler(View button){
    Download btn এ click করার ফলে clickEventHandler
    বা onClickListener call হবে and ভিতরে long-
    running task চলবে, যদি task টি more than
    5 sec. হয়, 5 sec. এর মধ্যে যদি UI/Main
    thread responsive না হয় তবে app টি exit
    বা wait করতে বলতে পারে,
}
```

Download Button

Worker Thread

কোনো UI element access করতে পারবে না, slide 1 এটা UI thread এর ভিতর Worker thread আছে যেখানে Image view করতে বলা হয়েছে (যেটা একটা UI element) এটি possible না, It will show a runtime exception and program করলার exit হবে,

→ Solution

1. Worker thread থেকে সব data download করবো and locally store করবো।

2. Then Worker thread থেকে UI thread এ একটা trigger generate করে দিবো যেটা locally stored সব data read করে view তে write করে দিবে,

3. UI thread কে control দিচ্ছি না, indirectly message show করাচ্ছি, এক্ষেত্রে W.T. must notify UI T. that long-running task has been done and data is stored somewhere (Memory, file, RAM)

⊛ Problems:—

UI Thread : কোনো long-running task run করা যাবেনা,

Worker Thread : UI element এর কোনো task run ,, ।

UI Thread related problem solve করার ৫টা way আছে—

1. Use Activity.runOnUiThread (Runnable)
2. Use View.post (Runnable)
3. Use View.postDelayed (Runnable, long)
4. Use Async Task
5. Use Executors and MainLoopers

* Activity.runOnUiThread → other thread এ কোন function টা call করছি, View.postDelayed &
* View.post → যে view/UI টা update করতে চাচ্ছি সেটা একটা handler দিয়া update করা দিবা, handler দিয়া ২ ভাবে message pass করা যায়, Immediately update করতে View.post use করা হয় এবং view.postDelayed (Runnable, long) দিয়া time fix করে দেয়া যায়,

UI Th. → Worker Th. → UI Th.

long → time টা

⊛ Async Task → এটি একটি class হিসাবে implement করা আছে। Async Task কে অন্য class এ extend করলে by default কতগুলো function call হয়ে যাবে। (Override করার মতো fun. থাকবে)

i. doInBackground → Runs in worckerc thread

ii. onPreExecute → Runs in UI thread

iii. onProgress → "

iv. onPostExecute → "

Async Task এ proces. system এর complexity কে কমানোর চেষ্টা করা হচ্ছে by using several functions

⊛ Executorc and MainLooperc

i. "executorc" → Worckerc Thread এর একটা object create করে দেয়া হচ্ছে UI thread এ.

ii. handlerc create করা হচ্ছে mainLooperc use করে, UI th. ← এর একটা instance

3. Worker thread এর ভিতর handler use করে downloaded file show করা দেখা হচ্ছে। It's mainly combination of the prev. two. (Handler & Asyne Task)

Base Thread

→ Android Java'র উপর base করে build করা। Base thread থেকে UI component declare করতে পারবো না। ২ ভাবে Base thread implement করতে পারি—

i. Providing a new class that extends Thread and overriding its run() method.

ii. Providing a new Thread instance with a Runnable object during its creation

→ যে কোনোই thread টা start/execute করতে হলে start() method invoke করতে হবে।

Example: 01

overriding run()

→ protected void startDownloadThread(){

```
Thread t = new Thread(){
    public void run(){
        mResult = "This is new result";
    }
};
t.start();
}
```

declaration part

called here

Code ~~to start~~ হওয়ার পর mResult এ যদি কোনো long running task থাকে তাহলে task টি parallely চলবে and অন্যান্য code run হবে, t1.join() দিয়ে দিলে অন্য dep. statement টি wait করতে থাকবে for t1's completion.

Example: 02

Thread Instance with a runnable

(a) Handler object

Example : 02

Base Thread

→ Base thread in Java is a standard thread that is used to perform concurrent operations. It operates independently from the main thread and follows certain principles when interacting with the application, especially with user interface components.

Key Characteristics of Base Thread

1. A base thread cannot directly modify UI elements.
2. Attempting to access UI objects throw an exception.
3. Cannot be stopped using destroy() or stop() as these are unsafe and deprecated.
4. Instead, interrupt() or join() methods are used to manage thread termination.

⊛ A main Handler object

→ Handler queue এর মধ্যে রেখে দিবে, Then priority wise execute করবে।

Post Example :-

1. handler একটা object create করে then new একটা thread call করা হয়েছে (যা UI thread এ হবে)

⊛ Creating and Executing a Base Thread

1. Extending the "Thread" class → Create a class that extends Thread and override its run() method.

2. Using a "Runnable" object → Pass a runnable object to a thread instance during its creation.

In both cases, you must call start() ~~met~~ method to begin thread execution.

Solutions

1. Handlere

1.1 A main thread handlere object

→ It helps to schedule the tasks to run into the main thread. By using a handlere object it prevents the possibilty of UI blockage.

1.2 Posting a runnable to the main view

→ Posting runnable adds a task to the main threads queue and ensures that it runs perfectly, even if its being triggered by any other thread.

Services

① What is a service in Android?

→ Service is a parent of an Android app that runs in the background to perform long running tasks without any user interaction. It is useful for operations like; Downloading files, Syncing data, Playing Music in the Background etc.

Key characteristics of a service has given below;

I. Services work silently in the background and do not have any visual elements.

II. They can perform tasks even when the user is interacting with other applications.

III. Services can run into different processe from the application that started them.

IV. Services can communicate with other components or applications, which is known as Inter Process Communication (IPC).

② Types of Services in Android

2.1 Foreground Services

→ Services whose task's are visible to the user, allows them for status tracking is known as foreground service.

Key features:

i. Requires a notification to keep users informed about its ongoing task.

ii. Continues running even when the user switches to another app.

Example → Music Players, Downloading Tasks.

• "notificationId" must be a unique non-zero value.

2.2 Background Services

→ These services run in the background, executing tasks without the users direct awareness.

Key features:

i. Ideal for processes that donot need user interaction.

ii. Invisible to the user, running silently.

Example → Syncing data.

2.3 Intent Service

→ Intent service is a specialized subclass of Service, designed for handling asynchronous, long running operations.

Key features:

I. Incoming requests are gathered in a queue. Intent service accepts them and executes them sequentially.

II. Automatically stops when all requests are processed.

III. Runs on a separate worker thread, ensuring smooth user experience.

Lifecycle:

1. Starts when a request is received.
2. stops itself when the queue is empty.

③. Service Lifecycle

→ The lifecycle of a service defines how it starts, executes, and stops. Services follow two main paths:

3.1 Started/Unbound Service

→ Unbound or started service is a service that

started with startService() and runs in the background until explicitly stopped.

Lifecycle:

i. onCreate() → Called when the service is created
ii. onStartCommand() → Invoked when startService() is called. Contains the service logic.
iii. onDestroy() → Cleans up resources before the service stops.

Stopping the service:

i. Explicitly call stopService().
ii. Use stopSelf() within the service logic.

3.2 Bound Service

→ It is a service that allows components to bind to it and interact with it like a client server.

Lifecycle:

i. onCreate() → Initializes a service.
ii. onBind() → Binds the client to the service.
iii. onUnbind() → Disconnects the client from the service.

iv. onRebind() → Reconnects new clients to the service if already running.

v. onDestroy() → Destroys the service when no clients are bound.

Example: Music streaming app where clients interact with the service to play, pause or skip tracks.

① Key Methods in Android Services

• onCreate()

→ Called once when the service is first created

→ Use this for initial setup like creating threads or initializing resources.

• onStartCommand()

→ Handles each request made to startService().

→ The method returns a value indicating the restart behavior if the service is killed,

- START_STICKY → Restarts the service automatically.
- START_NOT_STICKY → Doesnot restart if killed.
- START_REDELIVER_INTENT → Restarts with the last Intent.

- onBind()

→ Called when a client binds to the service using bindService().

→ Returns an IBinder interface to communicate with the service.

- onUnbind()

→ Triggered when all clients disconnect from the service.

- onRebind()

→ Invoked when new clients bind after the service was previously unbound.

- onDestroy()

→ Final method called before the service is destroyed. Used for cleanup.

⑤ Declaration in AndroidManifest

→ Every service must be declared in the AndroidManifest.xml file with attributes to define its behavior,

- android:name → Specifies the service class.
- android:exported → Determines whether the service can be used by other apps.

- android:process → Specifies the process in which the service will run.

⑥ Notifications in Services

→ Foreground services require notifications to inform the users of ongoing tasks.

→ Use NotificationCompat.Builder to create notifications.

⑦ Isolated and Exported Services

→ Isolated Services

- Run in separate processes for security and resource isolation
- Even if two isolated services have the same process name, they run in distinct processes

→ Exported Services

- Allow other apps to interact with the service
- The exported service run in the process of its application, not in the process of caller.

Android provides multiple techniques to access internet based resources. They are—

1. WebView
2. Client side SDKs
3. Own processing technique

Explanation has been given below,

1. WebView → WebView is an Android component that acts as a web browsers, enabling the application to display all the web pages into the app. It's functionality has given below,

- WebView eliminates the complexity of data transfers and connection management with remote servers.
- It is enough efficient to display web-based contents across multiple devices with minimal changes.
- WebView can load both local and remote data (HTML content) by using,

  → WebView.loadData() } load local
  WebView.loadDataBaseUrl() } data
  → WebView.loadUrl(url) } load remote data

- Developers can develop a single responsive website to represent all the platforms.

2. Client-side SDKs → SDKs (Software Development kits) are a collection of prewritten code that simplifies the interaction with remote servers or databases. These SDKs can be integrated with the Android system. Examples,

i. Google Firebase SDK (data authentication, real-time database)
ii. Firebase Cloud Messaging API (for notification)
iii. Google Maps API (for location)
iv. Google Billing API (for payment)

Functionalities or Benefits have been given below,

- SDKs eliminate the complexity to handle network connection, data transfer, and communication with the remote servers.
- These are all ready to use, saves the development time.

Issue,

- Developers should be well aware about the third party SDK's limitations or their unexpected behavior.

3. Own Processing Technique → It involves creating own custom code to directly interact with the remote res. It mainly involves parsing data (XML, HTML, JSON) and processing them, according the need. Working process,

i. Create HTTPs connection to the remote servers.

ii. Stream and accumulate the data.

iii. Process the accumulated data.

iv. Developers may use Java based parsers (like SXM

5th App Publishing (10 steps)

1. Create a Google Developers Account
2. Create a Merchant Account
3. Prepare the Documents
4. Study Google Developers Policies
5. Check Technical Requirements
6. Creating the App on Google Console
7. Store listing
8. Content Rating
9. Pricing and Distribution
10. Upload APK and send for Review

Briefing—

1. Create a Google Developers Account

- Use existing one or create a new one by agreeing with the Google Play Developers Distribution Agreement.
- Provide all the required informations and a one time registration fee of $25.
- Usually takes two days to finish it
- In this account you can update the infos anytime
- It also allows to transfere an App to another account

2. Creating a Merchant Account

- Paid App and In App Purchase থেকে revenue income করতে চাইলে Merchant account খুলতে হবে। Merchant account খুলার পর এটা automatically developers account এর সাথে linked up হয়ে যাবে।
- You do not need to create the Merchant account if you want to make money from ads. Rather it requires,

  AdMob এর account to ~~show~~ provide ads in your app
  AdSense account for payment

- Payment from ads and ... are

Techniques for Accessing Remote Resources

1. Available Techniques for Accessing Remote Resources

Android offers multiple techniques to access internet-based resources. These are categorized as follows:

1.1 WebView

- **Definition**: A **WebView** is a component in Android that acts as a web browser, enabling your application to display web pages directly inside your app.
- **Functionality**:
    - A WebView can load both **local** and **remote** HTML content.
    - **Local HTML** can be loaded using:
        - webview.loadData(htmlData, mimeType, encoding)
        - webview.loadDataWithBaseURL(baseURL, htmlData, mimeType, encoding, historyURL)
    - **Remote HTML** can be loaded via:
        - webview.loadUrl(url)
- **Benefits**:
    - WebView eliminates the need to handle the complexity of data transfer and connection management for remote content.
    - It is highly efficient for displaying web-based content across devices (computers, mobile phones, tablets) with minimal changes.
    - Developers can use a single responsive website for all platforms.

1.2 Client-side SDKs

- **Definition**: SDKs (Software Development Kits) are collections of pre-written code that simplify the process of interacting with remote servers or databases. These SDKs can be

integrated into Android applications.

- **Examples**:
    - Google Firebase SDK (for authentication, real-time databases, etc.).
    - Firebase Cloud Messaging API (for notifications).
    - Google Maps API (for map-based features).
    - Google Billing API (for handling payments).
- **Benefits**:
    - SDKs abstract away the complexity of handling network connections, data transfer, and communication with the remote server.
    - They are ready-to-use and save development time.
- **Issues**:
    - Sometimes, third-party SDKs can have limitations or unexpected behavior that developers need to watch out for.

1.3 Own Processing Technique

- **Definition**: This technique involves creating custom code to directly interact with remote resources. Typically, this involves parsing data (XML, HTML, JSON) and processing it as needed.
- **How It Works**:
    - Open an HTTPS connection to the remote server.
    - Stream and accumulate the data (e.g., JSON or XML).
    - Process the accumulated data (e.g., extracting relevant fields).
    - You may use Java-based parsers like **SAX** (Simple API for XML) or the **XML Pull Parser** for XML data, and **JSON parsers** for JSON-formatted data.
- **Concerns**:

    - Developers must be aware of the **data format/encoding** when processing remote data.

    - Caching and buffering are essential to optimize data retrieval and reduce repeated network requests.

    - The complexity of maintaining both client-side and server-side code.

---

2. Why Use Own Processing Technique?

- **Cost Optimization**:

    - Native apps often offer benefits over purely web-based applications. For example:

        - **Bandwidth**: Static resources like images, videos, and sounds can consume a lot of bandwidth. Native apps can reduce this by fetching only changed data.

        - **Battery Consumption**: Opening multiple connections for each data request can drain the device battery. Native apps can bundle network requests and keep the wireless radio turned off for longer periods.

- **Ensuring a Richer User Experience**:

    - **Caching**: Native apps can store data locally, allowing offline usage and reducing dependency on unreliable internet connections.

    - **Buffering**: Native apps can buffer content like audio or video, ensuring smoother playback even on unreliable networks.

    - **Native Features**: Android devices provide hardware features (like location services, notifications, sensors) that can enhance the user experience when combined with online data.

---

3. Alternatives for Accessing the Internet

Modern mobile devices offer multiple methods for accessing the internet:

- **Mobile Internet**: GPRS, EDGE, 3G, 4G, and LTE are mobile data technologies that allow internet access via mobile carriers.

- **Wi-Fi**: Wi-Fi and mobile hotspots provide internet access, and this option is commonly used for higher bandwidth applications.

Considerations:

- Mobile data (especially EDGE and GSM) can have limited bandwidth and high latency.
- Wi-Fi may be unreliable in mobile settings due to signal issues or disconnections.

---

4. Opening an Internet Data Stream

When downloading or streaming content, such as images, audio, video, or text, you need to open an internet data stream to access the remote data. This process typically involves:

1. **Opening an HTTPS connection** to the remote server.
2. **Reading streaming data** from the server.
3. **Processing the data** based on the application's needs.

---

5. Connecting to the Internet in an Android Application

Before you can access remote resources from within your Android app, you need to add the appropriate permissions in your **AndroidManifest.xml** to enable internet access.

Manifest Permission:

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

This permission must be included in your manifest to allow the app to access the internet.

---

6. Problem Discussion

The presentation also poses a scenario where two devices are connected to the same Wi-Fi router but the router itself is not connected to the internet. In this case, the devices are unable to access the internet but can still communicate with each other via local networks.

Conclusion

The techniques discussed in the presentation for accessing remote resources in Android include **WebView**, **Client-side SDKs**, and **Own Processing Techniques**. Each method has its advantages and challenges, and developers need to choose based on the app's requirements for performance, bandwidth, and user experience.

Business Models

1. Introduction to App Business Models

A business model defines how an app makes money. Choosing the right model involves answering these key questions:

1. **What problem does your app solve?** How does it do this uniquely?
2. **Will users pay for your app?** If yes, what features would they pay for?
3. **What do competitors do?** Are their models successful?
4. **Do you want quick profits or a larger user base first?** This depends on your app's goals and timeline.

Tip: Plan your monetization strategy before launching the app. While you can adjust it later, a well-thought-out approach is crucial from the start.

---

2. Popular App Monetization Models

Here's an overview of common app business models, their benefits, and challenges:

2.1 Free with Ads (In-App Advertising)

- **What It Is**: The app is free to download, and revenue comes from displaying ads.
- **How It Works**:
    - Users interact with your app for free.
    - Ads are shown based on user behavior or location.
    - Ad publishers pay you for placing their ads.
- **Example**: Facebook uses this model by showing targeted ads based on user data.

Pros:

- Attracts many users since it's free.

- Easy to collect user data (e.g., location or preferences) to show relevant ads.

Cons:

- Ads can annoy users, leading to uninstalls.
- Takes up screen space, reducing user experience.
- Not suitable for utility apps where users need quick, distraction-free access.

2.2 Freemium (Gated Features)

- **What It Is**: Basic app features are free, but premium features require payment.
- **How It Works**:
    - Users download the app for free and try its basic features.
    - To unlock more functionality, they must pay.
- **Example**: Angry Birds lets users play free levels but charges for additional levels or power-ups.

Pros:

- Helps build a loyal user base by allowing "try before you buy."
- Flexible—can be applied to almost any app type.

Cons:

- If the free version has too few features, users might uninstall.
- If it offers too many features for free, convincing users to upgrade can be difficult.

2.3 Paid Apps

- **What It Is**: Users pay upfront to download the app.

- **How It Works**:
    - The app is purchased directly from an app store.
    - No ads or additional charges are included.
- **Example**: Certain premium productivity or utility apps.

Pros:

- Generates immediate revenue from downloads.
- Users who pay upfront are often more engaged.

Cons:

- Harder to compete in a market dominated by free apps.
- App stores take a cut of revenue (30% for Google/Apple).
- Fewer downloads compared to free apps.

2.4 In-App Purchases (Physical/Virtual Goods)

- **What It Is**: The app sells items like physical goods, virtual items, or extra features.
- **How It Works**:
    - Users can buy things directly from the app, such as:
        - Physical goods (clothing, accessories).
        - Virtual items (game currency, extra lives).
- **Example**: MeetMe allows free use but sells credits for better visibility and additional features.

Pros:

- High profit margin since virtual items don’t have production costs.
- Works well for e-commerce and gaming apps.

Cons:

- App stores take a cut from virtual goods revenue.
- Transparency about purchases is needed to avoid user dissatisfaction.

2.5 Paywalls (Subscriptions)

- **What It Is**: Users pay a recurring fee to access premium content.
- **How It Works**:
    - A limited amount of content is free (e.g., articles, videos).
    - Users must subscribe for unlimited access.
- **Example**: Netflix or Umano (news converted to podcasts).

Pros:

- Ensures steady revenue through recurring subscriptions.
- Encourages loyalty since subscribers want to make the most of their investment.

Cons:

- Difficult to apply to all app types (best for news, lifestyle, or entertainment apps).
- Finding the right limit for free content is tricky.

2.6 Sponsorship (Incentivized Advertising)

- **What It Is**: Brands pay you to reward users for completing specific actions in the app.
- **How It Works**:
    - Users earn rewards (discounts, promotions) by performing tasks like tracking their progress or engaging with the app.
- **Example**: RunKeeper offers rewards for tracking running activities.

Pros:

- Encourages user engagement without intrusive ads.
- Suitable for apps with a specific purpose, like fitness or productivity.

Cons:

- Limited success as it's a newer model.
- Requires careful planning to avoid incentivizing spammy behavior.

---

3. Hybrid Business Models

Some apps use a mix of these strategies for better results. For example:

- Start with a "free with ads" model to gain users.
- Later, offer an ad-free version for a one-time fee (a mix of "free with ads" and "freemium").