**1. Mobile application development architectures**

1.1. Introduction to Mobile Application technologies

Mobile application development involves creating software designed to run on mobile devices such as smartphones and tablets. Different platforms (Android, iOS, Windows, etc.) have their own development ecosystems, each with its distinct architecture, programming languages, and tools.

* **Native apps**: Built specifically for a single platform (Android, iOS, etc.) using platform-specific languages (e.g., Java/Kotlin for Android, Swift/Objective-C for iOS). These apps have direct access to device features and offer better performance.
* **Web apps**: Applications accessed via a web browser, using HTML, CSS, and JavaScript. They are cross-platform but don't have access to native device features.
* **Hybrid apps**: A combination of web and native apps. They are web apps wrapped in a native container, giving them access to native device features. Popular frameworks include React Native, Ionic, and Flutter.

1.2. Android Architecture

Android architecture is divided into 4 layers

1.Application Layer

2.Application Framework

3.Libraries and Android Runtime

4.Linux Kernel



Android's architecture is indeed organized into four main layers, each playing a crucial role in the functioning of the Android operating system. Here's an overview of each layer:

1. Application Layer

- Description: This is the topmost layer where all Android applications reside. These include the apps pre-installed with the device (like contacts, phone, browser, etc.) as well as user-installed apps.

- Components: The applications at this layer are written in Java or Kotlin and interact with the Application Framework to perform tasks.

- Key Points:

- Users interact directly with the application layer.

- Apps are installed and run within this layer, providing functionalities to the user.

2. Application Framework

- Description: This layer provides the building blocks or APIs (Application Programming Interfaces) that developers use to create applications. The framework is designed to simplify app development by offering pre-coded modules.

- Components:

- Activity Manager: Manages the lifecycle of applications.

- Content Providers: Enable apps to share data with other apps.

- Resource Manager: Manages application resources like strings, layouts, and graphics.

- Notification Manager: Handles notifications for the user.

- View System: Provides UI components like buttons, lists, and text fields.

- Key Points:

- The framework provides services and common tasks to simplify app development.

- It ensures apps can interact with each other and access hardware efficiently.

3. Libraries and Android Runtime

- Libraries:

- Description: This part of the architecture includes a set of C/C++ libraries used by various components of the Android system. These libraries are critical for functions such as data storage, multimedia, graphics rendering, and web browsing.

- Key Libraries:

- Surface Manager: Manages display and compositing windows.

- Media Framework: Handles audio and video playback and recording.

- SQLite: A lightweight database engine used for data storage.

- OpenGL ES: A library for rendering 2D and 3D graphics.

- WebKit: A browser engine to display web content.

- Android Runtime (ART):

- Description: ART is the managed runtime used by applications, starting from Android 5.0 (Lollipop). It replaces the older Dalvik Virtual Machine.

- Key Features:

- Ahead-of-Time (AOT) Compilation: Improves app performance by compiling bytecode into machine code before execution.

- Garbage Collection: Manages memory by reclaiming unused memory automatically.

- Core Libraries: Provide essential functionalities like data structures, networking, and I/O operations.

4. Linux Kernel

- Description: The Linux Kernel is the foundation of the Android architecture, providing a layer of abstraction between the hardware and the rest of the software stack. It ensures that Android can efficiently manage resources and hardware components like the camera, memory, and network.

- Components:

- Drivers: Interfaces for hardware components like display, camera, Bluetooth, and Wi-Fi.

- Memory Management: Handles RAM and processes to ensure efficient use of device memory.

- Process Management: Manages tasks, threading, and process scheduling.

- Security: Implements security features such as SELinux (Security-Enhanced Linux) to protect against unauthorized access.

- Key Points:

- The Linux Kernel provides core system services and hardware abstraction.

- It manages device drivers, memory, processes, and network interfaces, ensuring Android can operate smoothly on a wide variety of devices.

**Summary**

**- Application Layer: Where user-installed apps run.**

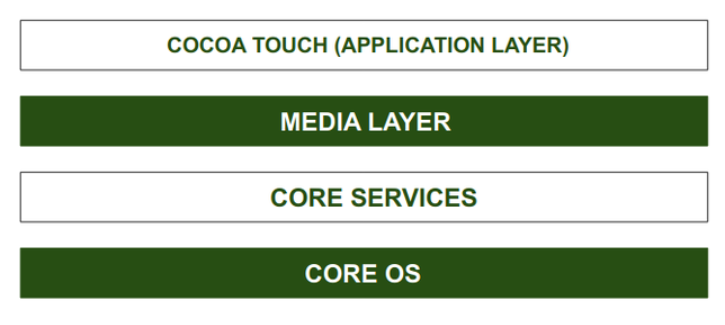
**- Application Framework: Provides APIs and system services for app development.**

**- Libraries and Android Runtime: Supports low-level system functions and provides runtime for apps.**

**- Linux Kernel: The base layer that manages hardware and system resources.**

**Each layer builds upon the one beneath it, creating a cohesive system that allows developers to create powerful, flexible applications for Android devices.**

1.3. IOS Architecture



1. Cocoa Touch Layer

- Description: This is the topmost layer and is responsible for the user interface and user interaction. It provides the frameworks necessary to create and manage the user interface and handle user inputs.

- Key Frameworks:

- UIKit: The primary framework for building and managing the app's UI, including buttons, labels, views, and controls.

- Foundation: Provides essential data types, collections, and utilities, such as strings, arrays, and dates.

- Core Motion: Accesses motion-related data from the device’s hardware, such as the accelerometer and gyroscope.

- MapKit: Enables the integration of maps and location services.

2. Media Layer

- Description: This layer is responsible for providing audio, video, animation, and graphics services. It ensures that iOS apps can deliver a rich multimedia experience.

- Key Frameworks:

- Core Graphics (Quartz): Provides 2D drawing, transparency layers, and vector-based rendering.

- Core Animation: Manages animations and transitions for UI elements.

- AVFoundation: Handles audio and video playback, recording, and processing.

- Core Image: Provides powerful image processing and manipulation tools.

- Metal: A high-performance, low-level API for sophisticated graphics rendering and computation.

3. Core Services Layer

- Description: This layer contains fundamental system services that all apps need to use, including data management, networking, and device communication.

- Key Frameworks:

- Core Data: Manages object graphs and persistent storage, often used for managing complex data models.

- CloudKit: Provides cloud-based storage and syncing for app data.

- Core Location: Provides location services and geofencing.

- Networking: Frameworks like `NSURLSession` allow for robust networking capabilities.

- Address Book: Manages contacts and provides access to the user’s contacts database.

4. Core OS Layer

- Description: This is the lowest layer and provides the fundamental services upon which the other layers are built. It is responsible for memory management, file system handling, security, and networking.

- Key Components:

- Kernel: The heart of iOS, managing memory, file systems, processes, and network communication.

- Security Services: Provides encryption, authentication, and data protection features.

- Power Management: Efficiently manages battery life and system resources.

- Bluetooth: Manages Bluetooth communication.

- Libsystem: A collection of fundamental system libraries and utilities.

**Summary of iOS Architecture:**

**- Cocoa Touch Layer: Focuses on the user interface and interaction.**

**- Media Layer: Handles multimedia, graphics, and animations.**

**- Core Services Layer: Provides essential services like networking, data management, and location services.**

**- Core OS Layer: The foundation of the OS, managing hardware interactions, security, and system resources.**

1.4. Windows Architecture

Windows Mobile (now largely replaced by Windows 10 Mobile, but no longer actively developed) had its own mobile architecture for app development.

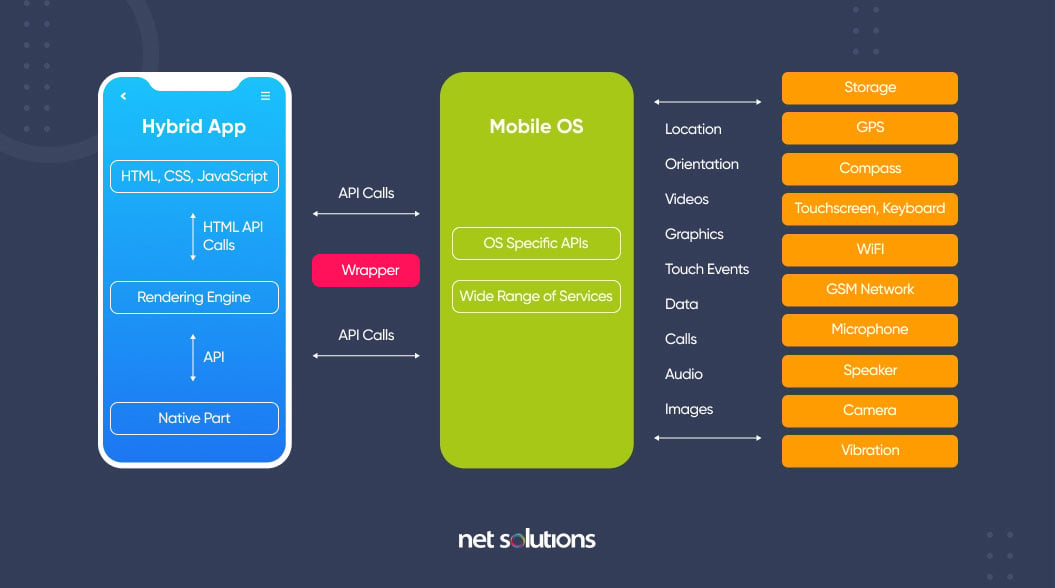
* **Kernel**: Windows Mobile used a stripped-down version of the Windows CE kernel, designed to run on small, low-power devices.
* **Core System Services**: This layer managed basic mobile services like messaging, phone calls, and power management. It also provided APIs for location services, Bluetooth, and network communication.
* **Compact Framework**: The Microsoft .NET Compact Framework was a key component, allowing developers to write mobile applications using C# and the .NET environment.
* **UI Framework (Silverlight)**: Similar to desktop apps, Windows Mobile provided Silverlight for creating the graphical user interface (UI) for mobile applications, offering tools to create visually rich UIs.

1.5. Hybrid Architecture

**Hybrid Mobile Application Architecture**

Hybrid mobile apps leverage both native and web solutions. Hybrid apps use native apps as “shells” for the back-end, but platform-neutral [JavaScript](https://www.netsolutions.com/javascript-development-company), HTML and CSS for the front-end. Hybrid apps use plugins such as Apache Cordova or Ionic Capacitor to access native platform features.

Hybrid mobile apps are among the fastest apps to create across a variety of platforms and easy to update, but are not appropriate for complex, interactive, or feature-rich applications.



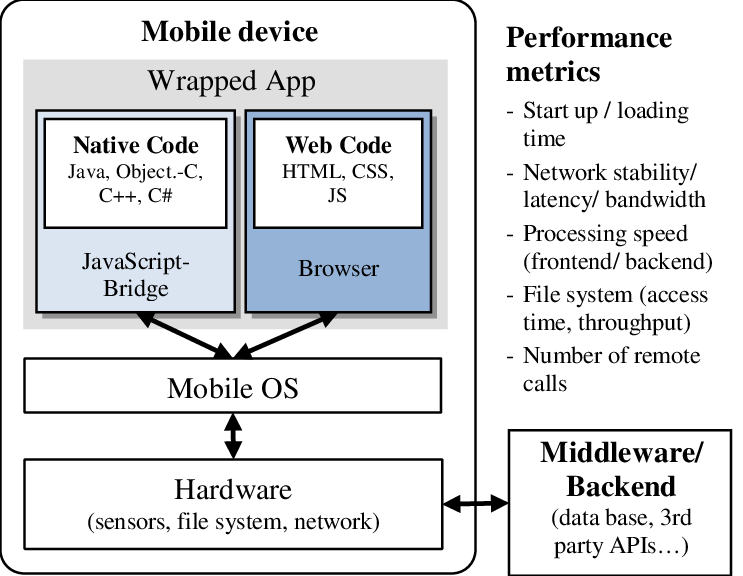
This solution is a blend, hence the name hybrid, of both native and web solutions. Where the core of the application is written using web technologies (HTML, CSS, and JavaScript), which are then encapsulated within a native application. Through the use of plugins, these applications can have full access to the mobile device’s features. To better understand this approach, let’s break down how it all fits together.

The heart of a hybrid-mobile application is still just an application that is written with HTML, CSS, and JavaScript. However, instead of the app being shown within the user’s browser, it is run from within a native application and its own embedded browser, which is essentially invisible to the user. For example, an iOS application would use the WKWebView to display our application, while on Android it would use the WebView element to do the same function.

This code is then embedded into a native application wrapper using a solution like Apache Cordova (also known as PhoneGap) or [Ionic’s Capacitor](https://capacitor.ionicframework.com/?__hstc=13779304.ff773afa2806bd7f052bf65a96c2684f.1675071220943.1675071220943.1675071220943.1&__hssc=13779304.2.1675071220944&__hsfp=3756997744" \t "_blank). These solutions create a native shell application that is just the platform’s webview component in which it will load your web application. This gives you the ability to create and publish true native applications that can be submitted to each of the platform’s app stores for sale.

Additionally, both Cordova and Capacitor have a plugin system that allows you to extend beyond the limitations of the ‘browser’ and access the full suite of capabilities of a user’s mobile device. So, if you wanted to use TouchID on an iOS device as a login option, or wanted to connect to a Bluetooth device, this can be easily done by installing a plugin. These plugins are created by a wide range of developers and many are actively supported. Ionic even offers a complete ecosystem of supported plugins as part of its [Enterprise](https://ionicframework.com/enterprise?__hstc=13779304.ff773afa2806bd7f052bf65a96c2684f.1675071220943.1675071220943.1675071220943.1&__hssc=13779304.2.1675071220944&__hsfp=3756997744) solution. So, the limitations of a web-only application are easily overcome, allowing your application to have parity with native applications in their features.

However, there are some drawbacks with this option. Similarly to the web-only application solution, the UI library has to be recreated. Here is where solutions like Ionic, NativeScript, Xamarin, React Native, and others step in. These options all provide robust UI components that look and feel like their native counterparts, giving you a full suite of building blocks for your hybrid mobile app.



**Features of hybrid applications**

Hybrid applications features include the following:

* the ability to function whether the device is connected or not;
* integration with the mobile device's file system;
* integration with web-based services; and
* an embedded browser to improve access to dynamic online content.

**How hybrid applications work**

Hybrid apps work in the same manner as web apps, but are downloaded to the device like native apps. Similar to web apps, developers typically write hybrid apps in [HTML5](https://www.techtarget.com/whatis/definition/HTML5), CSS and JavaScript. Hybrid apps run code inside a container. The device's browser engine renders HTML, JavaScript and native [APIs](https://www.techtarget.com/searchapparchitecture/definition/application-program-interface-API) to access device-specific hardware.

Although a hybrid app will typically share similar navigation elements as a web app, whether the application can work offline depends on its functionalities. If an application does not need support from a database, developers can make it function offline.

**Hybrid application pros and cons**

Pros of hybrid apps include the following:

* will operate on different platforms;
* faster build time compared to native apps;
* cheaper to develop compared to building two versions of a native app for two different platforms;
* easier to launch patches and updates; and
* can work online and offline.

**Some cons, however, include the following:**

* Variations due to leaning development on one platform may occur -- for example, if a development team leans their work on one platform, another supported platform may lack in quality or suffer from bugs.
* The appearance of an application may vary from platform to platform.
* Developers need to test the application on a range of devices to ensure proper operation.
* User experience ([UX](https://www.techtarget.com/searchcio/definition/UX-user-experience)) may fail if the user interface ([UI](https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI)) isn't similar to and well enough designed to what browsers the user is used to.

**2. Creating Android Application**

2.1. Creating an Android Project

To create an Android project in Android Studio:

Open Android Studio and select New Project.

Choose a Project Template (like Empty Activity, Navigation Drawer, etc.).

Configure your project:

Name: Choose a name for your app.

Package name: This is the unique identifier for your app (e.g., com.example.myapp).

Save location: Choose where to save your project files.

Language: Select between Java or Kotlin.

Minimum API Level: The lowest version of Android your app will support.

Click Finish. Android Studio will create the project structure automatically.

2.2. Project Structure

A typical Android project consists of the following directories:

app: Contains all the source code and resources for your app.

manifests: The AndroidManifest.xml file defines app components and permissions.

java: Contains your Java (or Kotlin) code files, including activities, fragments, and business logic.

res: Contains app resources such as layouts, images, strings, etc.

layout: XML files that define the UI (like activity\_main.xml).

drawable: Stores images and shapes.

values: Stores string, color, and dimension resources.

Gradle Scripts: Files that handle the build process, dependencies, and configuration (e.g., build.gradle).

2.3. Activity and Activity Life Cycle

An Activity is a single screen in your app, equivalent to a "page" on a website. Activities hold the UI components and handle user interaction.

Activity Life Cycle: Android manages the life cycle of an activity through the following stages:

onCreate(): Called when the activity is first created. Used to initialize components (e.g., setContentView).

onStart(): Called when the activity becomes visible to the user.

onResume(): Called when the activity starts interacting with the user.

onPause(): Called when the activity is partially obscured (e.g., when a dialog appears).

onStop(): Called when the activity is no longer visible.

onDestroy(): Called when the activity is destroyed and removed from memory.

onRestart(): Called when the activity is restarted after stopping.

2.4. Fragment and Fragment Life Cycle

A Fragment is a modular section of an activity, used to create more flexible and reusable UI components. A single activity can contain multiple fragments.

Fragment Life Cycle:

onCreate(): Initialize components related to the fragment.

onCreateView(): Inflate the layout for the fragment.

onActivityCreated(): Called when the activity's onCreate() is done.

onStart(): Fragment is visible.

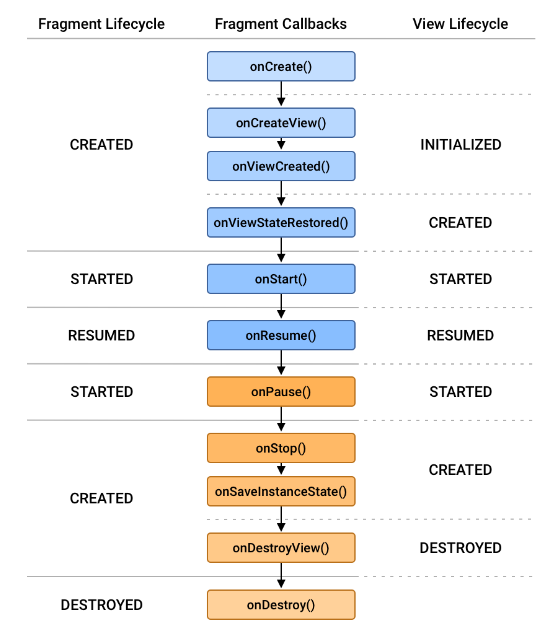
onResume(): Fragment is active and interacting with the user.

onPause(): Fragment is no longer active.

onStop(): Fragment is no longer visible.

onDestroyView(): Clean up the fragment's UI.

onDestroy(): The fragment is destroyed.



2.5. Views and ViewGroups

View: A single UI component (e.g., TextView, Button, EditText) that handles rendering and user interaction.

ViewGroup: A container that holds other views or ViewGroups. Examples:

LinearLayout: Aligns children in a single direction, either vertically or horizontally.

RelativeLayout: Positions children relative to each other.

ConstraintLayout: A flexible layout that positions elements based on constraints to other views or the parent container.

**3. Interactivity Tools**

3.1. Intents and Filters

Intents are messaging objects used to request an action from another app component. They allow communication between activities, services, and broadcast receivers.

Types of Intents:

Explicit Intent: Used to start a specific component (e.g., opening another activity within the same app).

java

Intent intent = new Intent(CurrentActivity.this, TargetActivity.class);

startActivity(intent);

Implicit Intent: Used when you don't specify the component, and the system determines which apps can handle the request (e.g., sharing content, opening a web browser).

java

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

intent.setData(Uri.parse("http://www.example.com"));

startActivity(intent);

Intent Filters: An intent filter allows an app to declare its capabilities. For example, if an app can open web pages, it can declare this using an intent filter in its `AndroidManifest.xml`.

xml

<intent-filter>

<action android:name="android.intent.action.VIEW" />

<category android:name="android.intent.category.DEFAULT" />

<data android:scheme="http" android:host="www.example.com" />

</intent-filter>

3.2. Adapters

An Adapter bridges a data source and a view (like a list or grid). It provides access to the data items and is responsible for creating views for each item.

Common Adapter Types:

ArrayAdapter: Maps an array or list to views.

RecyclerView.Adapter: More powerful, used with RecyclerView for efficient lists.

CursorAdapter: Used when working with data from a database.

Example of an ArrayAdapter for a ListView:

java

ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, data);

listView.setAdapter(adapter);

3.3. Dialogs

Dialogs are small windows that prompt the user to make a decision or enter additional information. Android provides several types of dialogs:

AlertDialog: Displays a simple alert dialog with buttons.

java

AlertDialog.Builder builder = new AlertDialog.Builder(this);

builder.setMessage("Do you want to continue?")

.setPositiveButton("Yes", (dialog, id) -> {

// Action for "Yes" button

})

.setNegativeButton("No", (dialog, id) -> {

// Action for "No" button

});

AlertDialog dialog = builder.create();

dialog.show();

DatePickerDialog: Allows users to select a date.

TimePickerDialog: Allows users to select a time.

Custom Dialog: You can create a custom layout and show it in a dialog.

3.4. Menus

Menus in Android are used to provide options to users. There are three types of menus:

Options Menu: Appears in the app bar or action bar.

- Define the menu items in `res/menu/menu\_main.xml`:

xml

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

<item android:id="@+id/action\_settings"

android:title="Settings"

android:orderInCategory="100"

android:showAsAction="never" />

</menu>

- Inflate the menu in your activity:

java

@Override

public boolean onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.menu\_main, menu);

return true;

}

Context Menu: Appears when the user long-presses on a view.

- Register the view for a context menu:

java

registerForContextMenu(view);

- Create the context menu:

java

@Override

public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) {

super.onCreateContextMenu(menu, v, menuInfo);

getMenuInflater().inflate(R.menu.context\_menu, menu);

}

Popup Menu: A small, modal menu that anchors itself to a view.

java

PopupMenu popup = new PopupMenu(this, view);

popup.getMenuInflater().inflate(R.menu.popup\_menu, popup.getMenu());

popup.show();

3.5. Notifications

Notifications are messages displayed outside of your app's UI, used to alert users about events. Notifications appear in the status bar, and when expanded, show more detailed information.

To create a notification:

1. Set up a Notification Channel (required for Android 8.0+).

java

NotificationChannel channel = new NotificationChannel("my\_channel\_id", "My Channel", NotificationManager.IMPORTANCE\_DEFAULT);

NotificationManager manager = getSystemService(NotificationManager.class);

manager.createNotificationChannel(channel);

2. Build the Notification:

java

NotificationCompat.Builder builder = new NotificationCompat.Builder(this, "my\_channel\_id")

.setSmallIcon(R.drawable.notification\_icon)

.setContentTitle("Notification Title")

.setContentText("This is a notification message.")

.setPriority(NotificationCompat.PRIORITY\_DEFAULT);

NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);

notificationManager.notify(1, builder.build());

3. Add Actions or Intents to your notification to allow users to interact with it (e.g., opening an activity when tapped).

**4. Interaction with Database**

**What is SQLite Database? Explain CRUD operations.**

An SQL database is SQLite. Tables are used in SQL databases to hold data. Data is stored in tables, which are made up of rows and columns. An Android database is a type of permanent data storage used by apps for Android-powered devices. In order for the app to remain accessible even if the device loses connectivity, it frequently comprises of on-device, local storage. The most popular database technology used with Android applications is SQLite, an open-source relational database, as a result of its inclusion in the Android Software Development Kit (SDK). The most popular combination of SQLite with Room, a framework for controlling the lifespan of objects, is for Android

**CRUD Operations**

The four fundamental actions that each software programme should be able to carry out are referred to as CRUD:

1. Create
2. Read
3. Update
4. Delete

Users must be able to create data, access it in the UI by reading it, update or edit it, and delete it in these apps.

**Create a SQLiteOpenHelper class**

**DBHandler** class is our database handler class which extends **SQLiteOpenHelper**class to manage database operations. First time on constructor (DBHandler) call we will create our database. In onCreate() method we will create any table e.g. Student.

**Creating db in sqlite**

public DBHandler( Context context, String name, SQLiteDatabase.CursorFactory factory, int version)

{  
 super(context, "TEST.db", factory, version);  
}

**Create Table in SQLite**

**E.g. Students**

@Override  
public void onCreate(SQLiteDatabase db) {  
 db.execSQL("CREATE TABLE STUDENTS(ID INTEGER PRIMARY KEY AUTOINCREMENT, FIRSTNAME TEXT UNIQUE, LASTNAME TEXT);");  
  
  
}

**Insert New Record**

public void InsertStudent(String SName, String SAddress, String ContactNo, String Emailid) {

        // on below line we are creating a variable for

        // our sqlite database and calling writable method

        // as we are writing data in our database.

        SQLiteDatabase db = this.getWritableDatabase();

        // on below line we are creating a

        // variable for content values.

        ContentValues values = new ContentValues();

        // on below line we are passing all values

        // along with its key and value pair.

        values.put(NAME\_COL, SName);

        values.put(ADDRESS\_COL, SAddress);

        values.put(CONTACTNUMBER\_COL, ContactNo);

        values.put(EMAIL\_COL, Emailid);

        // after adding all values we are passing

        // content values to our table.

        db.insert(TABLE\_NAME, null, values);

        // at last we are closing our

        // database after adding database.

        db.close();

    }

**Content Values** creates an empty set of values using the given initial size. We’ll discuss the other instance values when we jump into the coding part.

**When you need to modify a subset of your database values, use the**[**update()**](https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase#update(java.lang.String,%20android.content.ContentValues,%20java.lang.String,%20java.lang.String[]))**method. Updating the table combines the [ContentValues](https://developer.android.com/reference/android/content/ContentValues) syntax of**[**insert()**](https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase#insert(java.lang.String,%20java.lang.String,%20android.content.ContentValues))**with the WHERE syntax of**[**delete()**](https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase#delete(java.lang.String,%20java.lang.String,%20java.lang.String[]))**.**

**Updating Table**

public int update(long\_id, String Sname, String SAddress) {

ContentValues contentValues = new ContentValues();

contentValues.put(DatabaseHelper.Sudent, name);

contentValues.put(DatabaseHelper.SName, SAddress);

int i = database.update(DatabaseHelper.TABLE\_NAME, contentValues, DatabaseHelper.\_ID + " = " + \_id, null);

return i;

}

**Delete Record**

 Just need to pass the id of the record to be deleted as shown below.

public void delete(long \_id) {

database.delete(DatabaseHelper.TABLE\_NAME, DatabaseHelper.\_ID + "=" + \_id, null);

}

**Read Records**

A Cursor represents the entire result set of the query. Once the query is fetched a call to cursor.moveToFirst() is made. Calling moveToFirst() does two things:

* It allows us to test whether the query returned an empty set (by testing the return value)
* It moves the cursor to the first result (when the set is not empty)

The following code is used to fetch all records:

public Cursor fetch() {

String[] columns = new String[] { DatabaseHelper.\_ID, DatabaseHelper.Sname, DatabaseHelper.SAddress };

Cursor cursor = database.query(DatabaseHelper.TABLE\_NAME, columns, null, null, null, null, null);

if (cursor != null) {

cursor.moveToFirst();

}

return cursor;

**}**

Another way to use a Cursor is to wrap it in a CursorAdapter. Just as ArrayAdapter adapts arrays, CursorAdapter adapts Cursor objects, making their data available to an AdapterView like a ListView. Let’s jump to our project that uses SQLite to store some meaningful data.

**What is Firebase?**

A real-time database that saves and updates data in the cloud was created by Google and is called Firebase. On a variety of platforms, including Android, iOS, and the web, we can use Firebase. For the purpose of showing users data, Firebase supports the JSON format. Using the Firebase database platform, a developer may carry out fundamental CRUD (Create, Read, Update, and Delete) activities. **Firebase** is a Backend-as-a-Service (Baas). It provides developers with a variety of tools and services to help them develop quality apps, grow their user base, and earn profit. Firebase is a product of Google which helps developers to build, manage, and grow their apps easily. It helps developers to build their apps faster and in a more secure way. No programming is required on the firebase side which makes it easy to use its features more efficiently. It provides services to android, ios, web, and unity. It provides cloud storage. It uses NoSQL for the database for the storage of data. Although Firebase works as a database, it also has various other usages, as shown in the illustration below.

1. User Authentication
2. Real-time Database
3. Remote Configuration
4. Analytics
5. Building micro services
6. Game Development

Firebase is Back-end as a service owned by Google which provides server-less back-end to the app developers. It makes app developers to concentrate on front-end by managing back-end itself for developers. You can many features and options like analytics, notifications to apps using FCM, cloud functions, authentication, real-time databases, storage. For testing purposes we can use testlab and crash reporting. Firebase made my development very easy and convince too.

firebase is not an SQL database and it does not stores data in tabular format. It uses JSON tree structure. So for firebase realtime database the structure for above database will be.

**Brief History of Firebase:**

Firebase initially was an online chat service provider to various websites through API and ran with the name Envolve. It got popular as developers used it to exchange application data like a game state in real time across their users more than the chats. This resulted in the separation of the Envolve architecture and it’s chat system. The Envolve architecture was further evolved by it’s founders James Tamplin and Andrew Lee,to what modern day Firebase is in the year 2012.

**Advantages of Firebase**

Firebase comes with many advantages, as it reduces workload and saves costs for a developer. We mainly use this database because of the following benefits:

1. Real-time updating of data
2. Offline availability
3. Centralized database for all connected devices
4. Easy understanding
5. Cloud messaging

**How to connect Firebase with Android Studio**

**Step 1.** Go to Android Studio. If we don’t already have it, we must download and install it first. Create a new project.

**Step 2.** Once our empty project is fully loaded and Gradle files are synced properly, go to tools -> Firebase. Once a new window opens, locate and select Save and retrieve under the Realtime Database section.

**Step 3.** Connect our Firebase project and move to set our dependencies.

Now CRUD operations can be able to done by using Firebase.

**CRUD Operation in Android using Firebase Database.**

The Android SDK that we are going to use here is the set of development tools to develop applications for the android platform. The database we are using is Firebase which is a cloud-hosted database. So, let us start with the development. Crud Stands for **Create, Read, Update and Delete** operation in Database.

### **Getting Database Reference**

* First we need to get the Firebase Database Reference. You can use DatabaseReference to get the reference.

private DatabaseReference mDatabase;

mDatabase = FirebaseDatabase.getInstance().getReference("path"); //Dont pass any path if you want root of the tree

* The data is stored in the JSON Tree form so you need to get the reference of an specified path. Like in the above database we can get all the Artists by passing “Artists”. If you want to access everything don’t pass anything and it will create a reference of the root of the tree.

### **Write Operation**

* **setValue() –** This method will take a model java class object that will hold all the variables to be stored in the reference. The same method will be used to update the values at it overwrites the data of the specified reference.
* Suppose we have to store an Artist to our reference then we will create a model class as below.

public class Artist {

private String artistId;

private String artistName;

private String artistGenre;

public Artist(){

}

public Artist(String artistId, String artistName, String artistGenre) {

this.artistId = artistId;

this.artistName = artistName;

this.artistGenre = artistGenre;

}

public String getArtistId() {

return artistId;

}

public String getArtistName() {

return artistName;

}

public String getArtistGenre() {

return artistGenre;

}

}

* Now to save the artist we will use setValue() method in MainActivity.java.

Artist artist = new Artist(id, name, genre);

databaseReference.child(id).setValue(artist);

* The update operation will also be done in the same way.

### **Read Operation**

* We will attache a **ValueEventListener** to the reference to read the data.

databaseReference.addValueEventListener(new ValueEventListener() {

@Override

public void onDataChange(DataSnapshot dataSnapshot) {

}

@Override

public void onCancelled(DatabaseError databaseError) {

}

});

* Whenever you will change something in the Database the method **onDataChange()** will be executed. It contains all the data inside the specified path in the reference. We can use the **DataSnapshot** object to read all the data inside the reference.  If some error occurres **onCancelled()** method will be called.
* **onDataChange()** method will also called once after the app launch and hence you can read the data at starting as well.

### **Delete Operation**

* **removeValue()**can be used to delete the data.

**5. Web Services and Web View**

**5.1. Introduction to Web Services**

A **web service** is a method of communication between two electronic devices over a network. In Android development, web services allow apps to request and exchange data over the internet. Web services are typically RESTful, which means they use standard HTTP methods (GET, POST, PUT, DELETE) to perform CRUD operations.

* **REST (Representational State Transfer)**: A lightweight web service that uses HTTP and is commonly used for APIs.
* **SOAP (Simple Object Access Protocol)**: A more complex protocol that uses XML for messaging and is less common in modern mobile apps.

Web services often return data in formats like **JSON** (JavaScript Object Notation) or **XML** (eXtensible Markup Language), which the app can then parse and display.

**5.2. Receiving HTTP Response (JSON, XML)**

To communicate with a web service and receive a response, you typically use an HTTP client library. In Android, libraries like **Retrofit**, **Volley**, or **OkHttp** are popular choices.

* **Retrofit**: Retrofit is a type-safe HTTP client for Android, used for making requests to web services. Here's how to receive a JSON or XML response using Retrofit:
  1. Add Retrofit dependency to your build.gradle file:

groovy

Copy code

implementation 'com.squareup.retrofit2:retrofit:2.9.0'

implementation 'com.squareup.retrofit2:converter-gson:2.9.0' // For JSON

implementation 'com.squareup.retrofit2:converter-simplexml:2.9.0' // For XML

* 1. Define an API interface:

java

Copy code

public interface ApiService {

@GET("users")

Call<List<User>> getUsers(); // For JSON

@GET("users.xml")

Call<List<User>> getUsersXML(); // For XML

}

* 1. Create a Retrofit instance and make the network call:

java

Copy code

Retrofit retrofit = new Retrofit.Builder()

.baseUrl("https://api.example.com/")

.addConverterFactory(GsonConverterFactory.create()) // For JSON

.build();

ApiService service = retrofit.create(ApiService.class);

Call<List<User>> call = service.getUsers();

**5.3. Parsing JSON and XML**

Once you receive the response from a web service, you need to parse the data to use it in your app.

* **Parsing JSON**: If you're using Retrofit with **GsonConverterFactory**, JSON parsing happens automatically. However, you can manually parse JSON using the **org.json** package.

Example of manually parsing JSON:

java

Copy code

JSONObject jsonObject = new JSONObject(responseString);

String name = jsonObject.getString("name");

int age = jsonObject.getInt("age");

* **Parsing XML**: To manually parse XML in Android, you can use **XmlPullParser**. If you're using Retrofit with **SimpleXmlConverterFactory**, the XML will be parsed automatically.

Example of manual XML parsing:

java

Copy code

XmlPullParser parser = Xml.newPullParser();

parser.setInput(new StringReader(xmlResponse));

int eventType = parser.getEventType();

while (eventType != XmlPullParser.END\_DOCUMENT) {

if (eventType == XmlPullParser.START\_TAG && parser.getName().equals("name")) {

String name = parser.nextText();

}

eventType = parser.next();

}

**5.4. Introduction to Web View**

A **WebView** is a view that displays web pages inside your Android app, essentially acting as a mini browser. It can be used to load static HTML content, display web-based content, or even open external URLs.

* To use a WebView, first, add the WebView component to your XML layout:

xml

Copy code

<WebView

android:id="@+id/webview"

android:layout\_width="match\_parent"

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

* In your activity or fragment, you can load a URL or HTML content:

java

Copy code

WebView webView = findViewById(R.id.webview);

webView.getSettings().setJavaScriptEnabled(true); // Enable JavaScript if needed

webView.loadUrl("https://www.example.com"); // Load a URL

* You can also load static HTML content:

**6. React Native**

**6.1. Introduction**

**React Native** is a popular open-source framework for building mobile applications using JavaScript and React. It allows developers to create native apps for both Android and iOS with a single codebase, leveraging the power of React's component-based architecture. The key advantage of React Native is that it provides real native components, rather than web views, offering better performance and a more native feel.

* **Key Features**:
  + Cross-platform development (Android, iOS).
  + Uses JavaScript and React for building UI.
  + Supports a wide range of native device features.
  + Hot Reloading, allowing real-time updates to the app during development.

**6.2. Environment Setup**

To develop React Native apps, you need to set up your development environment. Here's how to do it:

1. **Install Node.js and npm** (Node Package Manager):
   * Download and install Node.js from the [official website](https://nodejs.org/).
   * npm is included with Node.js and is used to install packages.
2. **Install React Native CLI**:
   * You can install React Native using either the React Native CLI or Expo.
   * For React Native CLI:

bash

Copy code

npm install -g react-native-cli

1. **Install Android Studio and Xcode**:
   * For Android development, install Android Studio and set up the Android SDK.
   * For iOS development, install Xcode (only available on macOS).
2. **Set up a Virtual Device or Emulator**:
   * In Android Studio, create an Android Virtual Device (AVD) for testing.
   * For iOS, use Xcode's built-in iOS Simulator.
3. **Create a new React Native project**:

bash

Copy code

npx react-native init MyReactNativeApp

**6.3. JavaScript ES6 Overview**

React Native is built with JavaScript, particularly **ES6** (ECMAScript 2015) and beyond. Some of the important features of ES6 include:

* **Arrow Functions**: Shorter syntax for writing functions.

javascript

Copy code

const add = (a, b) => a + b;

* **Template Literals**: String interpolation using backticks (`).

javascript

Copy code

const name = "React";

console.log(`Hello, ${name}!`);

* **Classes**: A cleaner syntax for creating objects and inheritance.

javascript

Copy code

class Person {

constructor(name) {

this.name = name;

}

sayHello() {

return `Hello, my name is ${this.name}`;

}

}

* **Destructuring**: Extracting values from arrays or objects.

javascript

Copy code

const { name, age } = user;

* **Promises**: Used for asynchronous operations, often with fetch() for API calls.

javascript

Copy code

fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => console.log(data));

**6.4. Create React Native App**

You can create a new React Native app using **Expo CLI** or **React Native CLI**.

* **Using Expo** (simpler setup, especially for beginners):
  1. Install Expo CLI:

bash

Copy code

npm install -g expo-cli

* 1. Create a new project:

bash

Copy code

expo init MyNewProject

* 1. Start the development server:

bash

Copy code

cd MyNewProject

expo start

* **Using React Native CLI** (more control, needed for accessing native modules):
  1. Initialize the project:

bash

Copy code

npx react-native init MyReactNativeApp

* 1. Start the Metro bundler:

bash

Copy code

npx react-native start

* 1. Run the app on Android or iOS:

bash

Copy code

npx react-native run-android # For Android

npx react-native run-ios # For iOS

**6.5. React Native Alert API**

The **Alert API** in React Native provides a way to display alerts (similar to JavaScript alert()).

* Example of using the Alert API:

javascript

Copy code

import { Alert, Button } from 'react-native';

const showAlert = () => {

Alert.alert(

"Alert Title",

"My Alert Message",

[

{

text: "Cancel",

onPress: () => console.log("Cancel Pressed"),

style: "cancel"

},

{ text: "OK", onPress: () => console.log("OK Pressed") }

]

);

};

<Button title="Show Alert" onPress={showAlert} />

You can customize the button actions and appearance based on the use case.

**6.6. React Native Geolocation API**

The **Geolocation API** allows you to get the current location of the device. React Native provides this functionality via the built-in **Geolocation** module.

* Example of using Geolocation:

javascript

Copy code

import React, { useEffect, useState } from 'react';

import { Text, View } from 'react-native';

import Geolocation from '@react-native-community/geolocation';

const App = () => {

const [location, setLocation] = useState(null);

useEffect(() => {

Geolocation.getCurrentPosition(

(position) => {

setLocation(position.coords);

},

(error) => {

console.log(error);

},

{ enableHighAccuracy: true, timeout: 20000, maximumAge: 1000 }

);

}, []);

return (

<View>

<Text>Latitude: {location?.latitude}</Text>

<Text>Longitude: {location?.longitude}</Text>

</View>

);

};

export default App;

For more advanced features like continuously tracking location or using background geolocation, you might need third-party libraries like **react-native-geolocation-service**.

**6.7. Third-Party Libraries**

React Native has a large ecosystem of third-party libraries to extend functionality.

* **React Navigation**: For handling navigation in your app.

bash

Copy code

npm install @react-navigation/native

npm install react-native-screens react-native-safe-area-context

* **Redux**: For managing global state in large applications.

bash

Copy code

npm install redux react-redux

* **Axios**: For making HTTP requests to APIs (alternative to fetch()).

bash

Copy code

npm install axios

* **Lottie**: For adding animations to your app.

bash

Copy code

npm install lottie-react-native

* **React Native Paper**: Provides Material Design components for React Native apps.

bash

Copy code

npm install react-native-paper

These libraries and APIs greatly enhance React Native's capabilities, allowing you to build robust, feature-rich mobile applications.

**7. Introduction to Flutter**

**7.1. Overview of Flutter**

**Flutter** is an open-source UI software development kit (SDK) created by Google, used to build cross-platform applications for Android, iOS, web, and desktop from a single codebase. Flutter apps are written in the **Dart** programming language and can be compiled to native machine code for high performance.

* **Key Features**:
  + **Fast Development**: Flutter's "hot reload" feature allows for real-time updates during development.
  + **Expressive UI**: Flutter offers a rich set of customizable widgets to build beautiful interfaces.
  + **Cross-Platform**: Single codebase for Android, iOS, web, and desktop apps.
  + **Native Performance**: Flutter compiles to native ARM machine code for mobile platforms, ensuring high performance.

**7.2. Installation of Flutter**

To start developing with Flutter, you need to install the SDK and set up your environment.

1. **Install Flutter SDK**:
   * Download the latest Flutter SDK from the official website.
   * Extract the downloaded file and add the Flutter SDK to your system's PATH.

bash

Copy code

export PATH="$PATH:/path-to-flutter-sdk/flutter/bin"

1. **Set up IDE**:
   * Install **Android Studio** for Android development.
   * Install the **Flutter plugin** in Android Studio or **VS Code** for a more lightweight editor.
2. **Set up an Emulator**:
   * For Android, set up an Android Virtual Device (AVD) using Android Studio.
   * For iOS, use Xcode's built-in iOS Simulator (available on macOS).
3. **Verify the Installation**: After installation, run the following command to ensure everything is set up correctly:

bash

Copy code

flutter doctor

This command checks your environment and displays any missing dependencies.

**7.3. Architecture of Flutter**

Flutter follows a unique architecture based on the **Widget** tree model.

* **Widgets**: Everything in Flutter is a widget, whether it's a button, a layout, or even the entire screen. Flutter offers two types of widgets:
  + **Stateless Widgets**: These widgets are immutable and do not change over time.
  + **Stateful Widgets**: These widgets can change based on user interaction or data changes.
* **Flutter Engine**: The core of Flutter, responsible for rendering UI and handling input/output. It is built using **Skia**, a 2D graphics engine, and compiles Dart code to native ARM code.
* **Framework**: The high-level, reactive programming model that provides various built-in widgets and tools for UI, animation, and gestures.
* **Dart Language**: Flutter uses Dart as its programming language, and Dart's Ahead-Of-Time (AOT) compilation ensures the app runs with near-native performance.

**7.4. Introduction to Dart Programming**

**Dart** is the language used to develop Flutter apps. It is a modern, object-oriented language developed by Google.

* **Key Features**:
  + **Strong typing**: Dart supports both static and dynamic typing.
  + **Asynchronous programming**: Dart has built-in support for asynchronous operations using **Futures** and **async/await**.
  + **Garbage collection**: Dart automatically handles memory allocation and deallocation.
* **Basic Syntax**:
  + **Variables**:

dart

Copy code

var name = 'Flutter';

int age = 5;

* + **Functions**:

dart

Copy code

void greet(String name) {

print('Hello, $name');

}

* + **Classes and Objects**:

dart

Copy code

class Person {

String name;

int age;

Person(this.name, this.age);

void introduce() {

print('I am $name and I am $age years old.');

}

}

* + **Asynchronous Programming**:

dart

Copy code

Future<String> fetchData() async {

return 'Data fetched';

}

void main() async {

String data = await fetchData();

print(data);

}

**7.5. Demonstration of Simple Application**

Here's how to create a simple "Hello World" Flutter application.

1. **Create a new Flutter project**:

bash

Copy code

flutter create hello\_world\_app

cd hello\_world\_app

1. **Basic App Structure**: In Flutter, the main.dart file is the entry point of the app. Open lib/main.dart and modify the code:

dart

Copy code

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Hello World App',

home: Scaffold(

appBar: AppBar(

title: Text('Hello World'),

),

body: Center(

child: Text('Hello, Flutter!'),

),

),

);

}

}

* + The MyApp widget is the root of the application.
  + The MaterialApp widget is the top-level app container that provides Material Design widgets.
  + The Scaffold widget provides a basic structure, including the app bar and body.
  + The Text widget is used to display "Hello, Flutter!" in the center of the screen.

1. **Run the App**: You can now run the app on an Android emulator, iOS simulator, or a physical device using the following command:

bash

Copy code

flutter run

This will launch the "Hello, Flutter!" app on your device or emulator.