UNIT II: ANDROID ARCHITECTURE

2.1 Android Stack

2.2 Android applications structure

2.3 Creating a project

2.4 Configuring the Android Manifest File

2.5 Understanding the Components or layouts of a Screen

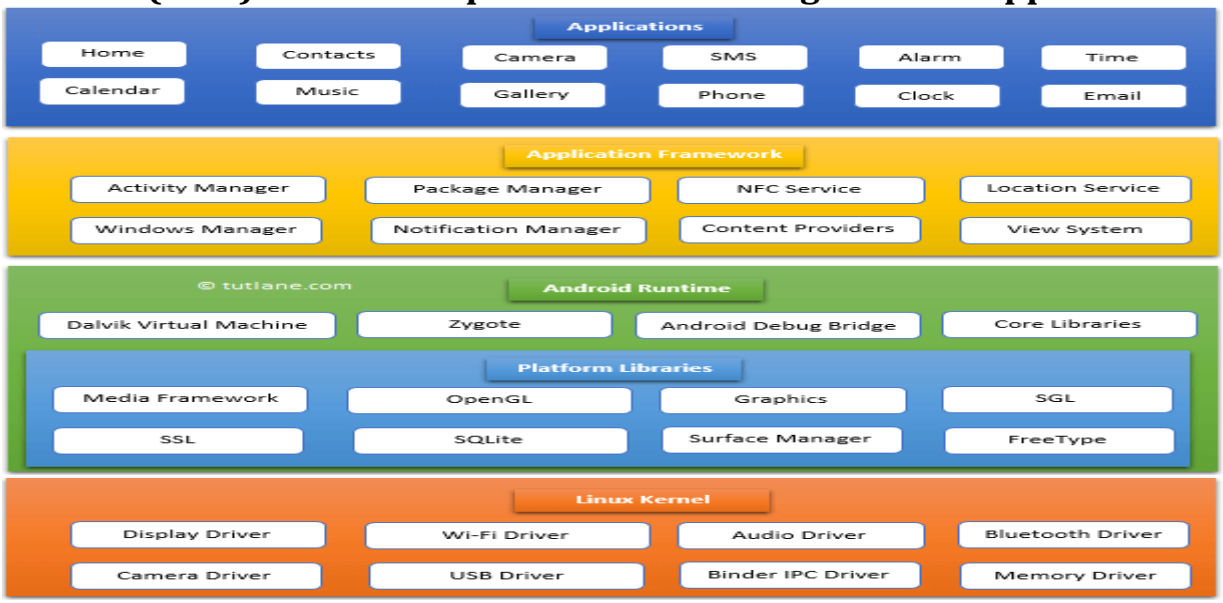
2.1 Android Stack

Android architecture **is a software stack of components to support mobile device needs. Android software stack contains a Linux Kernel, collection of c/c++ libraries which are exposed through an application framework services, runtime, and application.**

Following are main components of android architecture those are

1. Applications
2. Android Framework
3. Android Runtime
4. Platform Libraries
5. Linux Kernel

In these components, the Linux Kernel **is the main component in android to provide its operating system functions to mobile and** Dalvik Virtual Machine **(**DVM**) which is responsible for running a mobile application.**



Applications

The top layer of the android architecture is **Applications**. The native and third-party applications like contacts, email, music, gallery, clock, games, etc. whatever we will build those will be installed on this layer only.

Application Framework

The **Application Framework** provides the classes used to create Android applications. It also provides a generic abstraction for hardware access and manages the user interface and application resources. It basically provides the services through which we can create a particular class and make that class helpful for the Application creation.

Android Runtime

Android Runtime environment is an important part of Android rather than an internal part and it contains components like **core libraries** and the **Dalvik virtual machine**. The Android run time is the engine that powers our applications along with the libraries and it forms the basis for the application framework.

Dalvik Virtual Machine **(**DVM**) is a register-based virtual machine like Java Virtual Machine (JVM). It is specially designed and optimized for android to ensure that a device can run multiple instances efficiently. It relies on the Linux kernel for threading and low-level memory management.**

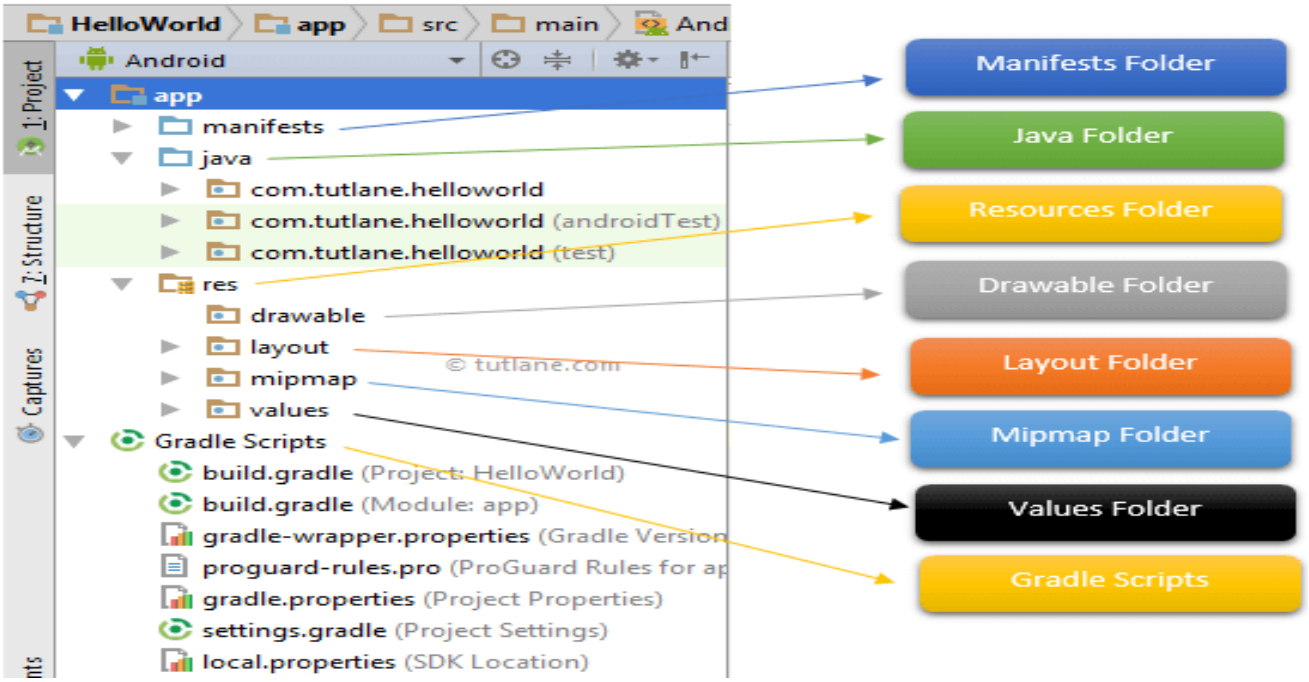
Platform Libraries

The **Platform Libraries** includes various C/C++ core libraries and Java-based libraries such as SSL, libc, Graphics, SQLite, Webkit, Media, Surface Manger, OpenGL, etc. to provide support for Android development.

Linux Kernel

Linux Kernel is a bottom layer and heart of the android architecture. It manages all the drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc.

2.2 Android applications structure



Manifests Folder

This folder will contain a manifest file (**AndroidManifest.xml**) for our android application. This manifest file will contain information about our application such as android version, access permissions, metadata, etc. of our application and its components. The manifest file will act as an intermediate between android OS and our application.

Java Folder

This folder will contain all the java source code (**.java**) files which we'll create during the application development, including JUnit test code. Whenever we create any new project/application, by default the class file **MainActivity.java** will create automatically under the package name

res (Resources) Folder

It's an important folder that will contain all non-code resources, such as bitmap images, UI strings, XML layouts
The res (**Resources**) will contain a different type of folders that are

Drawable Folder (res/drawable)

It will contain the different types of images as per the requirement of application. It's a best practice to add all the images in a **drawable** folder other than app/launcher icons for the application development.

Layout Folder (res/layout)

This folder will contain all XML layout files which we used to define the user interface of our application. Following is the structure of the **layout** folder in the android application.

Mipmap Folder (res/mipmap)

This folder will contain app / launcher icons that are used to show on the home screen. It will contain different density type of icons such as hdpi, mdpi, xhdpi, xxhdpi, xxxhdpi, to use different icons based on the size of the device.

Values Folder (res/values)

This folder will contain various XML files, such as strings, colors, style definitions and a static array of strings or integers.

Gradle Scripts

In android, Gradle means automated build system and by using this we can define a build configuration that applies to all modules in our application. In Gradle **build.gradle (Project)**, and **build.gradle (Module)** files are useful to build configurations that apply to all our app modules or specific to one app module.

2.3. Creating a project

Step 1: Android Studio makes it easy to create Android apps for various form factors, such as handsets, tablets, TV, and Wear devices.

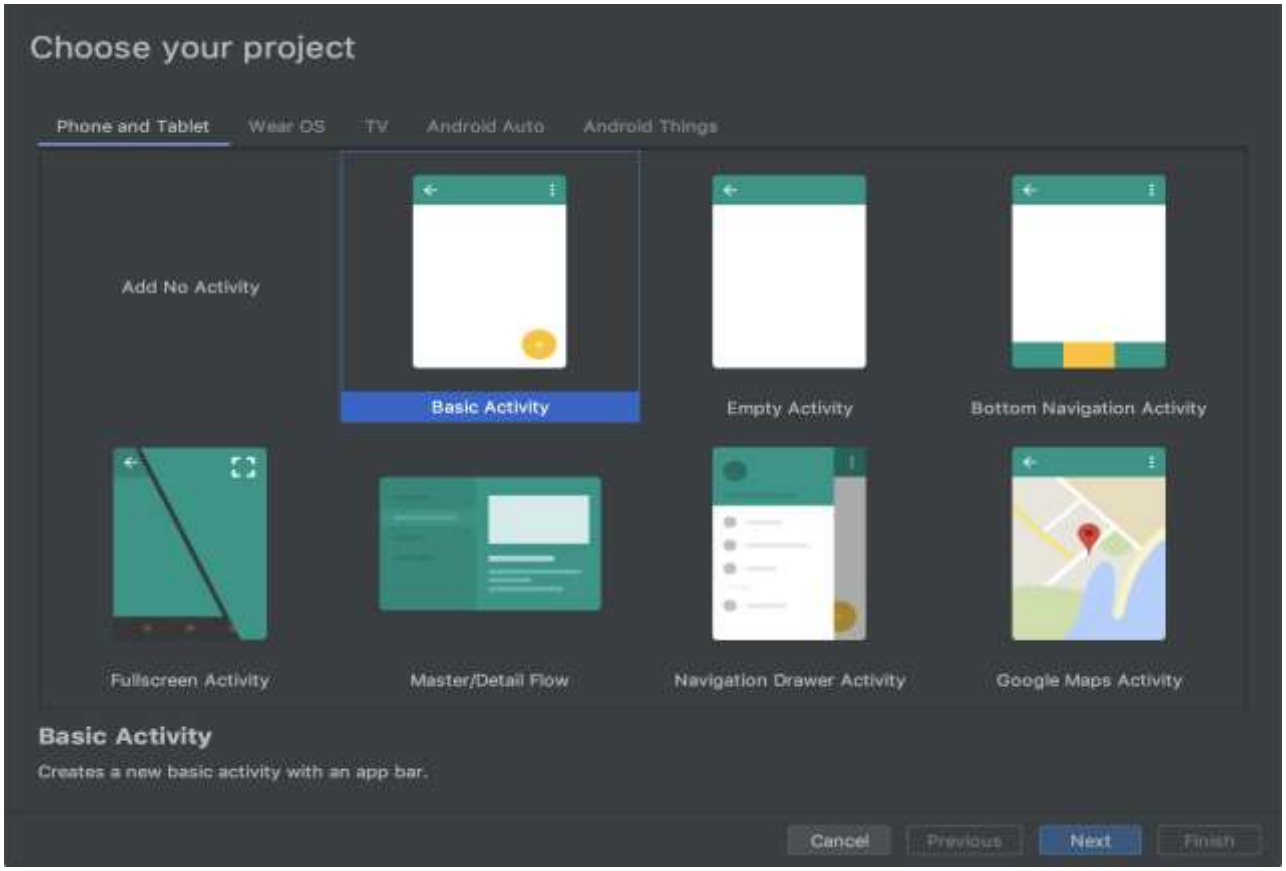
If you don't have a project opened, Android Studio shows the Welcome screen, where you can create a new project by clicking **Start a new Android Studio project**.

If you do have a project opened, create a new project by selecting **File > New > New Project** from the main menu.

You then see the **Create New Project** wizard, which lets you choose the type of project you want to create and populates with code and resources to get you started. This page guides you through creating a new project using the **Create New Project** wizard.

Choose your project

In the **Choose your project** screen that appears, you can select the type of project you want to create from categories of device form factors, which are shown as tabs near the top of the window.



After you make a selection, click **Next**.

1. Specify the **Name** of your project.
2. Specify the **Package name**. By default, this package name also becomes your application ID, which you can change later.
3. Specify the **Save location** where you want to locally store your project.
4. Select the **Language** you want Android Studio to use when creating sample code for your new project. Keep in mind, you are *not* limited to using only that language in the project.
5. Select the **Minimum API level** you want your app to support. When you select a lower API level, your app can't use as many modern Android APIs. However, a larger percentage of Android devices are able to run your app. The opposite is true when selecting a higher API level. If you want to see more data to help you decide, click **Finish Button**.

2.4 Configuring the Android Manifest File

The **AndroidManifest.xml file** *contains information of your package*, including components of the application such as activities, services, broadcast receivers, content providers etc.

- It is **responsible to protect the application** to access any protected parts by providing the permissions.
- It also **declares the android api** that the application is going to use.
- It **lists the instrumentation classes**. The instrumentation classes provides profiling and other informations. These informations are removed just before the application is published etc.

This is the required xml file for all the android application and located inside the root directory.

Elements of the AndroidManifest.xml file

The elements used in the above xml file are described below.

1. <manifest>

manifest is the root element of the AndroidManifest.xml file. It has **package** attribute that describes the package name of the activity class.

2. <application>

application is the subelement of the manifest. It includes the namespace declaration. This element contains several subelements that declares the application component such as activity etc.

The commonly used attributes are of this element are **icon**, **label**, **theme** etc.

android:icon represents the icon for all the android application components.

android:label works as the default label for all the application components.

android:theme represents a common theme for all the android activities.

3. <activity>

activity is the subelement of application and represents an activity that must be defined in the AndroidManifest.xml file. It has many attributes such as label, name, theme, launchMode etc.

android:label represents a label i.e. displayed on the screen.

android:name represents a name for the activity class. It is required attribute.

4. <intent-filter>

intent-filter is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.

4.1 <action>

It adds an action for the intent-filter. The intent-filter must have at least one action element.

4.2 <category>

It adds a category name to an intent-filter.

```
<manifest>
  <application>
<activity android:name=".MainActivity" >
    </activity>
  </application>
</manifest>
```

5. <uses-permission>: This element specifies the Android Manifest permissions that are requested for the purpose of security.

2.5 Understanding the Components or layouts of a Screen

1. Linear Layout

LinearLayout is the most basic layout in android studio, that aligns all the children sequentially either in a horizontal manner or a vertical manner by specifying the **android:orientation** attribute.

If one applies **android:orientation="vertical"** then elements will be arranged one after another in a vertical manner and

If you apply **android:orientation="horizontal"** then elements will be arranged one after another in a horizontal manner.

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

</LinearLayout>
```

android:id

This is the ID which uniquely identifies the layout.

android:Layout_Gravity

This specifies how a **control** should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center

android:gravity

This specifies how a **text** should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.

android:orientation

This specifies the direction of arrangement and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.

android:weightSum

Sum up of child weight

android:LayoutWeight

It is use for Divide the weight sum according to the design.

2. Relative Layout

Android RelativeLayout enables you to specify how child views are positioned relative to each other. The position of each view can be specified as relative to sibling elements or relative to the parent.

1. **android:layout_alignParentBottom.**

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

2. **android:layout_alignParentRight**

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

3. **android:layout_centerHorizontal**

If true, centers this child horizontally within its parent. Must be a boolean value, either "true" or "false".

4. **android:layout_centerVertical**

If true, centers this child vertically within its parent. Must be a boolean value, either "true" or "false".

5. **android:layout_centerInParent**

If true, centers this child horizontally and vertically within its parent. Must be a boolean value, either "true" or "false".

6. **android:layout_above**

Positions the bottom edge of this view above the given anchor view ID and must be a reference to another resource, in the form

android:layout_above=”@id/text”

7. **android:layout_below**

Positions the top edge of this view below the given anchor view ID and must be a reference to another resource

android:layout_below=”@id/text”

8. **android:layout_toLeftOf**

Positions the right edge of this view to the left of the given anchor view ID and must be a reference to another resource

android:layout_ toLeftOf=”@id/text”

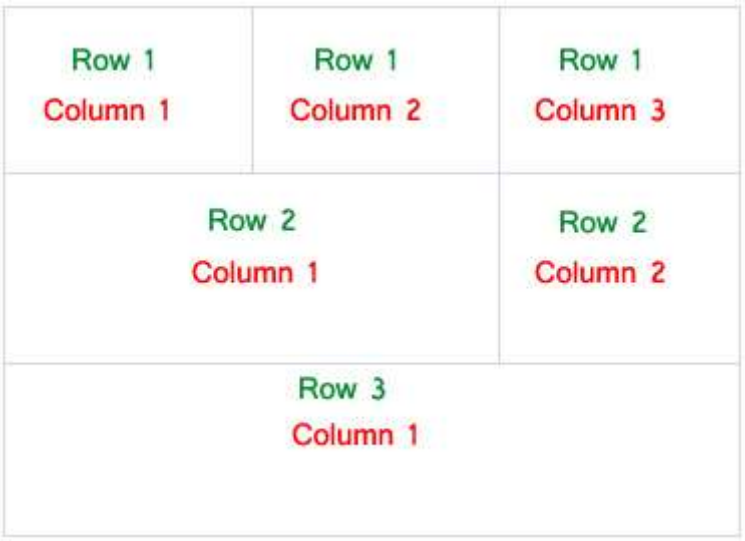
9. android:layout_toRightOf

Positions the left edge of this view to the right of the given anchor view ID and must be a reference to another resource

android:layout_ toRightOf=”@id/text”

3. Table Layout

Android Table Layout going to be arranged groups of views into rows and columns. You will use the <TableRow> element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.



android:id

This is the ID which uniquely identifies the layout.

android:collapseColumns

Collapse columns attribute is used to collapse or invisible the columns of a table layout. These columns are the part of the table information but are invisible.

If the values is 0 then the first column appears collapsed, i.e. it is the part of table but it is invisible.

android:shrinkColumns

Shrink column attribute is used to shrink or reduce the width of the column‘s. We can specify either a single column or a comma delimited list of column numbers for this attribute. The content in the specified columns word-wraps to reduce their width.

If the value is 0 then the first column's width shrinks or reduces by word wrapping its content.

If the value is 0, 1 then both first and second columns are shrinks or reduced by word wrapping its content.

If the value is '*' then the content of all columns is word wrapped to shrink their widths.

android:stretchColumns

Stretch column attribute is used in Table Layout to change the default width of a column which is set equal to the width of the widest column but we can also stretch the columns to take up available free space by using this attribute. The value that assigned to this attribute can be a single column number or a comma delimited list of column numbers (1, 2, and 3…n).

If the value is 1 then the second column is stretched to take up any available space in the row, because of the column numbers are started from 0.

If the value is 0, 1 then both the first and second columns of table are stretched to take up the available space in the row.

4. Constraint Layout

Advantages of using Constraint Layout in Android

- Constraint Layout provides you the ability to completely design your UI with the drag and drop feature provided by the Android Studio design editor.
- It helps to improve the UI performance over other layouts.
- With the help of Constraint Layout, we can control the group of widgets through a single line of code.
- With the help of Constraint Layout, we can easily add animations to the UI components which we used in our app.

Disadvantages of using Constraint Layout

- When we use the Constraint Layout in our app, the XML code generated becomes a bit difficult to understand.
- In most of the cases, the result obtain will not be the same as we got to see in the design editor.
- Sometimes we have to create a separate layout file for handling the UI for the landscape mode.

android:id

This is used to give a unique id to the layout.

app:layout_constraintBottom_toBottomOf

This is used to constrain the view with respect to the bottom position.

app:layout_constraintLeft_toLeftOf

This attribute is used to constrain the view with respect to the left position.

app:layout_constraintRight_toRightOf

This attribute is used to constrain the view with respect to the right position.

app:layout_constraintTop_toTopOf

This attribute is used to constrain the view with respect to the top position.

5. Frame Layout

Framelayout is a **ViewGroup** subclass that is used to specify the position of **View** instances it contains on the top of each other to display only single **View** inside the FrameLayout.

In simple manner, we can say FrameLayout is designed to block out an area on the screen to display a single item.
FrameLayout will act as a placeholder on the screen and it is used to hold a single child view.

In FrameLayout, the child views are added in a stack and the most recently added child will show on the top. We can add multiple children views to FrameLayout and control their position by using gravity attributes in FrameLayout.

```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">



</FrameLayout>
```

The End