Android Tutorial for Beginners

Published On: July 29, 2024

Android Tutorial for Beginners

Android is a whole suite of software for mobile devices, including set-top boxes, laptops, smartphones, tablets, and e-readers. In this Android tutorial, you will learn the fundamentals of how to develop apps for the Android platform.

Download Android Tutorial PDF

Introduction to Android

It includes middleware, essential mobile apps, and an operating system built on Linux. It functions similarly to a mobile operating system.

However, it isn't exclusive to mobile devices. These days, it's found in a lot of gadgets, including televisions, tablets, and cell phones.

In this Android tutorial, we'll cover the following:

- App Basics
- App Resources
- Localization
- App Architecture

Android Interview Questions

App Basics

Java, C++, and Kotlin are programming languages that can be used to create Android apps. Your code is assembled into an APK or Android App Bundle by the Android SDK tools, together with any necessary data and resource files.

Security Features of Android

Each Android program operates in its own security sandbox and is protected using the following Android security features:

- With multiple users for each program, the Android operating system is essentially a multi-user Linux system.
- The system automatically assigns each application a unique Linux user ID; the app is unaware of this ID and it is only used in the system.
- All of an application's files are granted rights by the system, limiting access to them to the user ID that is associated with that app.
- Because every process has a virtual machine (VM), the code of an application operates independently of other applications.
- Every application runs in a separate Linux process by default.
- When an app's components need to be run, the Android system launches the process; when it's no longer required or the system has to free up memory for other apps, it ends.

App Components

App components are the essential elements that make up an Android application's structure. Every part serves as a portal for users or the system to access your application. Certain elements rely on other elements.

Four categories of app components exist:

- Activities

ASP DOTNET Tutorial
Published On: July 31, 2024

ASP DOTNET Tutorial Microsoft created the web framework known as ASP.NET. It is employed in...

Artificial Intelligence Tutorial
Published On: July 30, 2024

Artificial Intelligence Tutorial Artificial intelligence (AI) is significant since it enhances many facets of society...

Appium Testing Tutorial
Published On: July 30, 2024

Appium Testing Tutorial Designed to make the UI automation of many app platforms easier, Appium...

- Services
- Broadcast receivers
- Content providers

Activities

The first step in communicating with a user is an activity. It depicts a solitary screen featuring an interface. An email app, for instance, might include three activities: one for creating emails, one for reading them, and one for displaying a list of recent emails.

An activity makes it easier for the system and app to connect in the following vital ways:

- maintaining track of the user's current focus—what's shown on the screen—so that the activity-hosting process continues to execute on the system.
- Understanding which halted activities in previously used processes the user could revisit and giving more priority to those processes in order to maintain their availability.
- assisting the application in handling the termination of its process so that the user can resume activities in their prior state.
- giving applications a means of implementing user flows amongst themselves and allowing the system to manage these flows. The main illustration of this is sharing.

Services

A service is a multifunctional entry point that allows an application to run in the background for many purposes. It is a background-running component that carries out recurring tasks or works with remote processes.

Started services and bound services are the two categories of services that instruct the system on how to handle an application.

- **Started services** instruct the system to

continue executing them until their task is finished. This might be used to play music even after the user exits the app or to sync some data in the background.
- **Bound services** operate because the system or another app has indicated that it wishes to use the service. The system is aware that there is a dependency between two processes when a bound service offers an API to another process.

Broadcast Receivers

To enable an app to react to system-wide broadcast announcements, the system can send events to it outside of normal user flow using a broadcast receiver.

Every broadcast is sent as an intent object, and broadcast receivers are implemented as subclasses of BroadcastReceiver.

Content Providers

You can store your app's data on the file system, in a SQLite database, online, or in any other permanent storage location that your app can access.

A content provider is responsible for managing this shared collection of data. Some apps may query or modify the data through it if permission is granted using the content source.

This enables the system to perform the following specific tasks while managing an app:

- URIs can survive the termination of the programs that own them since assigning a URI does not require the app to be operating.
- Additionally, these URIs offer a crucial, fine-grained security mechanism.

Activate Components

Every kind of component can be activated using the

following different techniques:

- By giving an intent to ***startActivity() or startActivityForResult()*** when you want the activity to produce a result, you can launch an activity or assign it a new task.
- By supplying an intent to ***startService()***, you can launch a new service or update an existing one on older Android versions. By providing an intent to ***bindService()***, you can bind to the service.
- By giving an intent to functions like ***sendBroadcast() or sendOrderedBroadcast()***, you can start a broadcast.
- Calling ***query() on a ContentResolver*** will allow you to send a query to a content provider.

Android Syllabus PDF

The manifest file

Before the app can be started, the system needs to read the AndroidManifest.xml manifest file and know that an app component exists. This file, located in the app project directory's root, contains declarations for every component in your program.

In addition to declaring the components of the application, the manifest performs several other tasks, including the following:

- Identifies any user rights that the application needs, like read access to the user's contacts or internet access.
- Specifies the lowest API level that the application needs, depending on the APIs that it utilizes.
- Identifies the hardware and software components—like a camera, Bluetooth capabilities, or a multitouch screen—that are utilized or necessary for the program.
- Declares which API libraries, like the Google Maps library, the app must be linked against in

addition to the Android framework APIs.

Declare Components

The manifest's main job is to tell the system about the elements that make up the application. An activity can be declared in a manifest file.

Example

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

<manifest ... >

  <application android:icon="@drawable/app_icon.png" ... >

    <activity
android:name="com.example.project.ExampleActivity"

          android:label="@string/example_label" ... >

    </activity>

    ...

  </application>

</manifest>
```

The android:icon attribute in the element links to resources for an icon that represents the program.

The android:name property of the element specifies the fully qualified class name of the Activity subclass, and the android:label attribute specifies the string that will be displayed as the activity's user-visible label.

All app components must be declared using one of the following elements:

- <activity> elements for activities
- <service> elements for services
- <receiver> elements for broadcast receivers
- <provider> elements for content providers

Declare Component Capabilities

You can optionally include intent filters when declaring an activity in your app's manifest, which declare the activity's capabilities and enable it to

react to intents from other apps. To accomplish this, add an element as a child of the declaration element of the component.

> **Example**
>
> *<manifest ... >*
>
> *...*
>
> *<application ... >*
>
> *<activity android:name="com.example.project.ComposeEmailActivity">*
>
> *<intent-filter>*
>
> *<action android:name="android.intent.action.SEND" />*
>
> *<data android:type="*/*" />*
>
> *<category android:name="android.intent.category.DEFAULT" />*
>
> *</intent-filter>*
>
> *</activity>*
>
> *</application>*
>
> *</manifest>*

For the user to write and send an email, the system may launch your activity if another app develops an intent with the ACTION_SEND action and passes it to startActivity().

Declare App Requirements

Declaring device and software requirements in your manifest file will help you build a clear profile for the kinds of devices your app supports and prevent it from being installed on devices that don't have the capabilities you need.

These requirements must be disclosed. The *build.gradle* file in your app module contains the following values for *minSdkVersion and targetSdkVersion*:

> *android {*
>
> *...*

```
    defaultConfig {
      ...
      minSdkVersion 26
      targetSdkVersion 29
    }
}
```

The camera feature is declared in the manifest file of your application:

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera.any"
                  android:required="true" />
    ...
</manifest>
```

Android Training

App Resources

There's more to an Android app than just code. Images, audio files, and other materials related to the application's visual presentation are needed, but they are not part of the source code itself.

Example: XML files can be used to describe animations, menus, styles, colors, and the arrangement of activity user interfaces.

- It's simple to update different aspects of your app without changing the code by using app resources.
- You may optimize your app for a range of device settings, including different screen sizes and languages, by offering sets of alternative resources.

Group Resource Types

Put every kind of resource in its own subdirectory under the res/ directory of your project. Here is the file hierarchy for a basic project:

> **Example**
>
> *MyProject/*
>
> *src/*
>
> *MyActivity.java*
>
> *res/*
>
> *drawable/*
>
> *graphic.png*
>
> *layout/*
>
> *main.xml*
>
> *info.xml*
>
> *mipmap/*
>
> *icon.png*
>
> *values/*
>
> *strings.xml*

All of the resources in its subdirectories, including a string resource file, two layout resources, a mipmap/ directory for launcher icons, and an image resource, are contained in the res/ directory.

Localization

Numerous devices across various geographies run Android. Make sure your program handles text, audio files, numbers, money, and visuals appropriately for the locations in which it is utilized to maximize its user base.

Resource switching in Android

Text strings, layouts, audio, pictures, and any other static data that your Android app need are all considered resources.

Multiple resource sets, each tailored for a certain device setup, can be included in an app.

Android chooses and loads the resources that are most appropriate for the device when the user launches the app.

Example

The Kotlin or Java code of an application relates to merely text_a and text_b, two strings.

Text_a and text_b are defined in English in a localized resource file (res/values-en/strings.xml) included with the application.

Additionally, the application comes with a default resource file (res/values/strings.xml) that defines text_a but not text_b.

- Because res/values-en/strings.xml contains both of the required text strings, this app may operate without issue when started on a device with the locale set to English.
- Nevertheless, the user encounters an error message and a Force Close button when they open this program on a device that is configured to speak a language other than English. There is no app load.

Android Developer Salary

Use Resources for Localization

Create default resources: The default resource set may also contain animations and other resource types.

The following directories include these resources:

- **res/drawable/:** need directory containing a minimum of one graphic file for the Google Play app's icon
- **res/layout/:** the necessary directory containing an XML file defining the standard layout
- **res/anim/:** necessary if any res/anim- folders exist
- **res/xml/:** necessary if any res/xml- directories exist.
- **res/raw/:** necessary if any res/raw- folders

exist

Create alternate resources: Applications can specify a variety of res// directories, each with its own set of requirements. A language qualifier or a language-region combination can be used to produce an alternate resource for a different locale.

Example:

Three strings.xml files are created and saved in separate locale-specific resource directories:

- **res/values/strings.xml:** Every string that the application uses, including the text for the title string, has English text included.
- **res/values-fr/strings.xml:** It includes the title and all other strings written in French.
- **res/values-ja/strings.xml:** have Japanese writing on every string but the title.

When your Java- or Kotlin-based code makes reference to R.string.title, the following occurs at runtime:

- Android loads titles from the res/values/strings.xml file if the device is configured in a language other than French.
- Android loads the title from the res/values-fr/strings.xml file if the device is set to French.

Android searches the res/values-ja/strings.xml file for the title if the device's language setting is set to Japanese.

However, Android reverts to its default and loads the English title from the res/values/strings.xml file since no such text is present in that file.

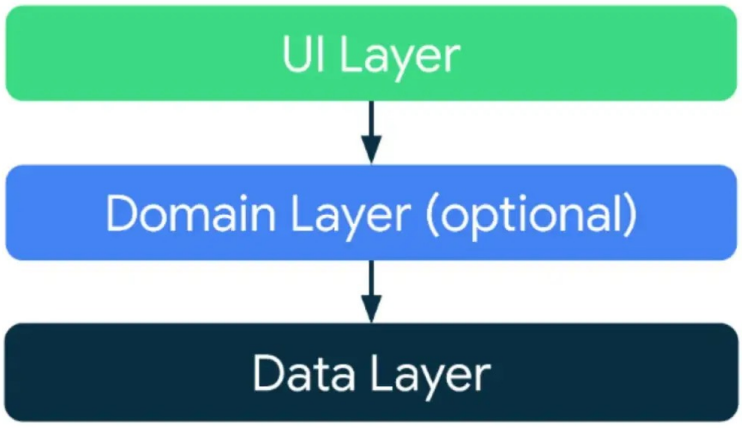
App Architecture

To make sure your apps are reliable, tested, and maintained, app architecture design is crucial. To assist you in building your app in accordance with best practices, Android offers a collection of libraries and components.

Every application needs to have a minimum of two layers:

- The **user interface layer** shows program data on the screen.
- The **data layer** exposes application data and houses your app's business logic.

To make the interactions between the UI and data levels simpler and more reusable, you can add a new layer called the domain layer.



Android Tutorial for Beginners

Among other things, this modern app architecture recommends employing the following methods:

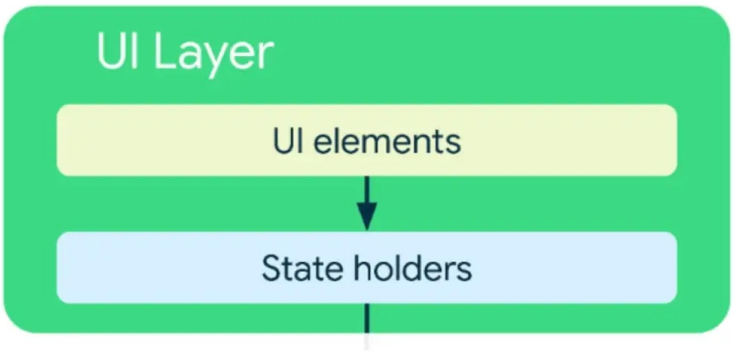
- A tiered, responsive architecture.
- All application levels provide unidirectional data flow or UDF.
- A UI layer that uses stakeholders to control the UI's complexity.
- Flows and coroutines.
- Dependency injection.

UI Layer

The user interface (UI) layer, sometimes referred to

as the presentation layer, displays the application data on the screen.

The user interface (UI) should be updated to reflect any changes made to the data, whether as a result of human engagement (e.g., pushing a button) or external input (e.g., a network response).



Android Tutorial for Beginners

There are two components to the UI layer:

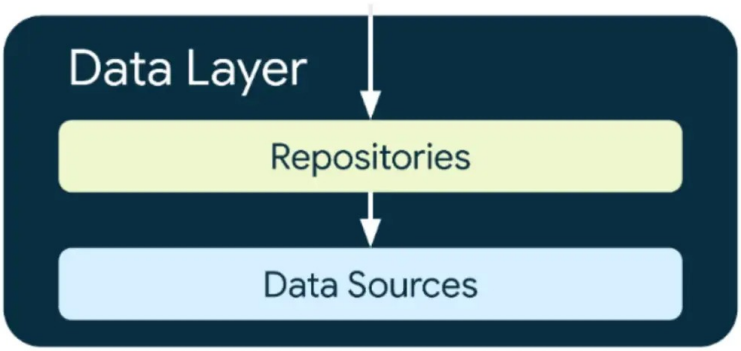
- **UI elements** that display the data on the display. These elements are constructed with the use of Jetpack Compose functions or views.
- **State holders** that manage logic, store data, and present it to the user interface (such as ViewModel classes).

Data Layer

The business logic of an application is stored in the data layer. Your app's value comes from its business logic, which is composed of rules that specify how it should create, save, and modify data.

The repositories that make up the data layer may contain one or more data sources. It is necessary to develop a repository class for every kind of data that your application handles.

Example: You may make a MoviesRepository class to hold information about movies, or a PaymentsRepository class to hold information about payments.



Android Tutorial for Beginners

The following are the responsibilities of repository classes:

- Exposing information to the app's other features.
- Centralized updating of the data.
- Resolving disagreements arising from several data sources.
- Separating data sources from the rest of the application.
- Having commercial sense in it.

Domain Layer

The domain layer holds shared business logic, either straightforward or complex, for several

"ViewModels."

Since not all apps will meet these requirements, this layer is optional. It should only be used in specific situations, such as when handling complexity or promoting reusability.

Conclusion

We hope this Android tutorial will be helpful for beginners to understand the core concepts of building apps on the Android platform. Learn to build your first app by enrolling in our **Android training in Chennai** at SLA.

Share on your Social Media

Softlogic Academy

Softlogic Systems

KK Nagar [Corporate Office]

No.10, PT Rajan Salai, K.K. Nagar, Chennai – 600 078.
Landmark: Karnataka Bank Building
Phone: +91 86818 84318
Email: enquiry@softlogicsys.in
Map: Google Maps Link

OMR

No. E1-A10, RTS Food Street
92, Rajiv Gandhi Salai (OMR),
Navalur, Chennai - 600 130.

Landmark: Adj. to AGS Cinemas
Phone: +91 89256 88858
Email: info@softlogicsys.in
Map: Google Maps Link

Courses

- Python
- Software Testing
- Full Stack Developer
- Java
- Power BI
- Clinical SAS
- Data Science
- Embedded
- Cloud Computing
- Hardware and Networking
- VBA Macros
- Mobile App Development
- DevOps

Social Media Links

Review Sources

- Google
- Trustpilot
- Glassdoor
- Mouthshut
- Sulekha
- Justdial
- Ambitionbox
- Indeed
- Software Suggest
- Sitejabber