

UNIT-1

Introduction of Mobile Applications

1.1 Introduction

Talking about the mobile applications, the first thing that comes to mind are the apps like WhatsApp, Instagram, swigy, etc that we use in our everyday life.

Ever thought about how these apps are made? Which technology is used? Let's discuss what technologies or frameworks can be used to develop a mobile application.

Mobile Apps are majorly developed for 3 Operating System:

- Android
- IOS
- Windows

There are 3 different ways to develop Mobile apps: –

1. **1st Party Native App development:** These types of apps normally run in the native devices, that is, it runs only in the OS that it is specifically designed for it. These apps cannot be used on different devices using a different OS. The apps that are developed for android are normally coded using Java or Kotlin languages. The IDE normally used for android app development is Android Studio which provides all features and the apps that are developed for IOS are generally coded in Swift language or Objective-C.

The IDE suggested for IOS App Development is XCode.

Example:

Here's an example of a 1st party native app:

A retail company wants to improve the in-store shopping experience for its customers. They develop a 1st party native app that allows customers to:

- Browse the store's inventory and product information
- Create a shopping list
- Scan barcodes to view product information and reviews
- Locate items in the store using an interactive map
- Pay for items directly through the app, without having to wait in line at the register
- The app is only available to the company's customers and can only be used in their physical stores. The app is designed to integrate with the company's existing systems, such as inventory management and point-of-sale systems.
- This app is developed by the retail company for their own use, to improve the in-store customer experience, increase sales and gain insights from the customer's behaviour.

In this example, the retail company is the 1st party, and the app is a native app, because it is developed for the specific platform (iOS or Android) and can take full advantage of the device's capabilities and features.

2. **Progressive web Application:** Progressive web apps are essentially a website which runs locally on your device. The technologies used are Microsoft Blazor, React, Angular JS, Native Script, Ionic. These technologies normally used for web development propose. The apps' UI is developed the same way as they are developed while developing the website. This category has many ups and downs let's start with the advantages of Progressive web apps.

Example:

Here's an example of a Progressive Web App:

A news website wants to provide its users with a better mobile experience. They develop a Progressive Web App that:

- Allows users to access the website offline by storing content on the user's device
- Sends push notifications to users to alert them of breaking news
- Can be installed on the user's home screen like a native app
- Provides a fast and smooth browsing experience
- Has a responsive design that adapts to different screen sizes
- Users can access the PWA by visiting the website on their mobile browser. They are prompted to install the PWA on their home screen, which allows them to access the website offline and receive push notifications.

In this example, the news website is the 1st party and the app is a Progressive web app, because it can be accessed through a web browser and can be installed on the user's device like a native app. It also allows users to access the content offline and have a fast and smooth experience.

3. **Cross-Platform Application:** These are frameworks that allow developing total native applications which have access to all the native features of IOS and Android but with the same code base. These apps run on both Android and IOS. So normally the development speeds of these apps are very fast and the maintenance cost is low. The performance speed is comparatively low to 1st party native apps but faster than PWA.

Xamarin is Microsoft cross-platform solution that uses the programming languages like .NET, C#, F#. The IDE preferred is Visual Studio. The UI/UX is totally native giving access to all features. This technology is having a wide community. And whenever an update is released by Android and IOS the same updates are released by Microsoft through Visual Studio.

React Native is Facebook's cross-platform solution which uses the language JavaScript And the preferred IDE is WebStrome & Visual Studio Code. Same like Xamarin React Native has totally native UI/UX and gives access to all features. And the updates are released the same day by Facebook as Android and IOS.

Flutter is Google's cross-platform solution which uses the language, Dart. The IDE preferred is Android Studio, IntelliJ IDE, and Visual Studio Code. The UI/UX is bespoke and Flutter has to come up with their new libraries whenever Android and IOS comes up with an update to mimic those update. The community is fast growing.

Example:

Here's an example of a cross-platform application:

A project management company wants to create a project management tool that can be used by teams on different platforms. They develop a cross-platform application that Can be used on Windows, Mac, iOS, and Android devices:

- Allows users to create and assign tasks, set deadlines, and track progress
- Integrates with popular tools such as Google Calendar and Trello
- Has a user-friendly interface that works seamlessly across all platforms
- The application can be downloaded from the company's website or from different app stores such as App Store, Google Play Store, Microsoft Store, and Mac App Store, depending on the platform.

This example illustrates how the company developed a project management tool that can be used by teams on different platforms, Windows, Mac, iOS and Android, which is a cross-platform application. It allows teams to collaborate and manage their projects seamlessly, regardless of the platform they use.

1.2 Mobile Development Framework

A mobile development framework is a software framework that is designed to support mobile app development. It is a software library that provides a fundamental structure to support the development of applications for a specific environment.

Frameworks can be in three categories: native frameworks for platform-specific development, mobile web app frameworks, and hybrid apps, which combine the features of both native and mobile web app frameworks.

Mobile App Development Framework is a library that offers the required fundamental structure to create mobile applications for a specific environment. In short, it acts as a layout to support mobile app development. There are various advantages of Mobile App Development frameworks such as *cost-effectiveness*, *efficiency*, and many more. Moreover, mobile application frameworks can be classified majorly into 3 categories: *Native Apps*, *Web Apps* & *Hybrid Apps*.

Before moving further, let's take a quick look at all three categories of mobile development apps.



- **Native Apps:** A Native App is an application specifically designed for a particular platform or device.

- **Web Apps:** A Web App is concerned with an application that is designed to deliver web pages on different web platforms for any device.
- **Hybrid Apps:** A Hybrid App is a combination of both native & web applications. It can be developed for any platform from a single code base.

1.2.1 React Native

React Native is one of the most recommended Mobile App Frameworks in the development industry. The framework, created by Facebook, is an open-source framework that offers you to develop mobile applications for Android & iOS platforms. The React Native framework is based on React and JavaScript that aims to develop native applications over hybrid applications that run on a web view. Moreover, it is a cross-platform development framework that uses a single code base for both Android & iOS applications. Some of the major benefits of React Native are mentioned below:

- Code Re-usability & Cost-Effective
- Compatible with third-party plugins
- Re-usable components for optimal performance
- Provides hot deployment features
- Ease of Maintenance

1.2.2 Flutter

Flutter, developed by **Google**, is a **UI toolkit** to build native applications for mobile apps, desktop & web platforms. Flutter is a **cross-platform mobile app development framework** that works on one code base to develop Android as well as iOS applications. The framework provides a large range of fully customizable widgets that helps to build native applications in a shorter span. Moreover, Flutter uses the **2D rendering engine called Skia** for developing visuals and its layered architecture ensures the effective functioning of components. Some of the major benefits of Flutter are mentioned below:

- Provides Full Native Performance
- Flexible User interface (UI)
- Provides Strong Widget Support
- Offers Built-in Material Design
- Fast Application Development

1.2.3 Ionic

Ionic, developed in **2013**, is an open-source framework that allows you to build cross-platform for mobile apps using web technologies like **HTML, CSS & JavaScript**. The application built through the Ionic framework can work on *Android, iOS & Windows* platforms. The framework offers numerous default UI components such as *forms, action sheets, filters, navigation menus*, and many more for attractive and worthwhile design. Moreover, Ionic has its **own command-line interface** and various other in-built features such as **Ionic Native, Cordova-Based App packages**, etc. Some of the major benefits of Ionic for mobile development apps are mentioned below:

- Faster Application Development.

- Availability of Cordova Plugins
- Built-in UI components
- Platform Independent
- Based on AngularJS

Android Framework(Platform)

Android is an open source, Linux-based software stack created for a wide array of devices and form factors. Figure shows the major components of the Android Framework/Platform.

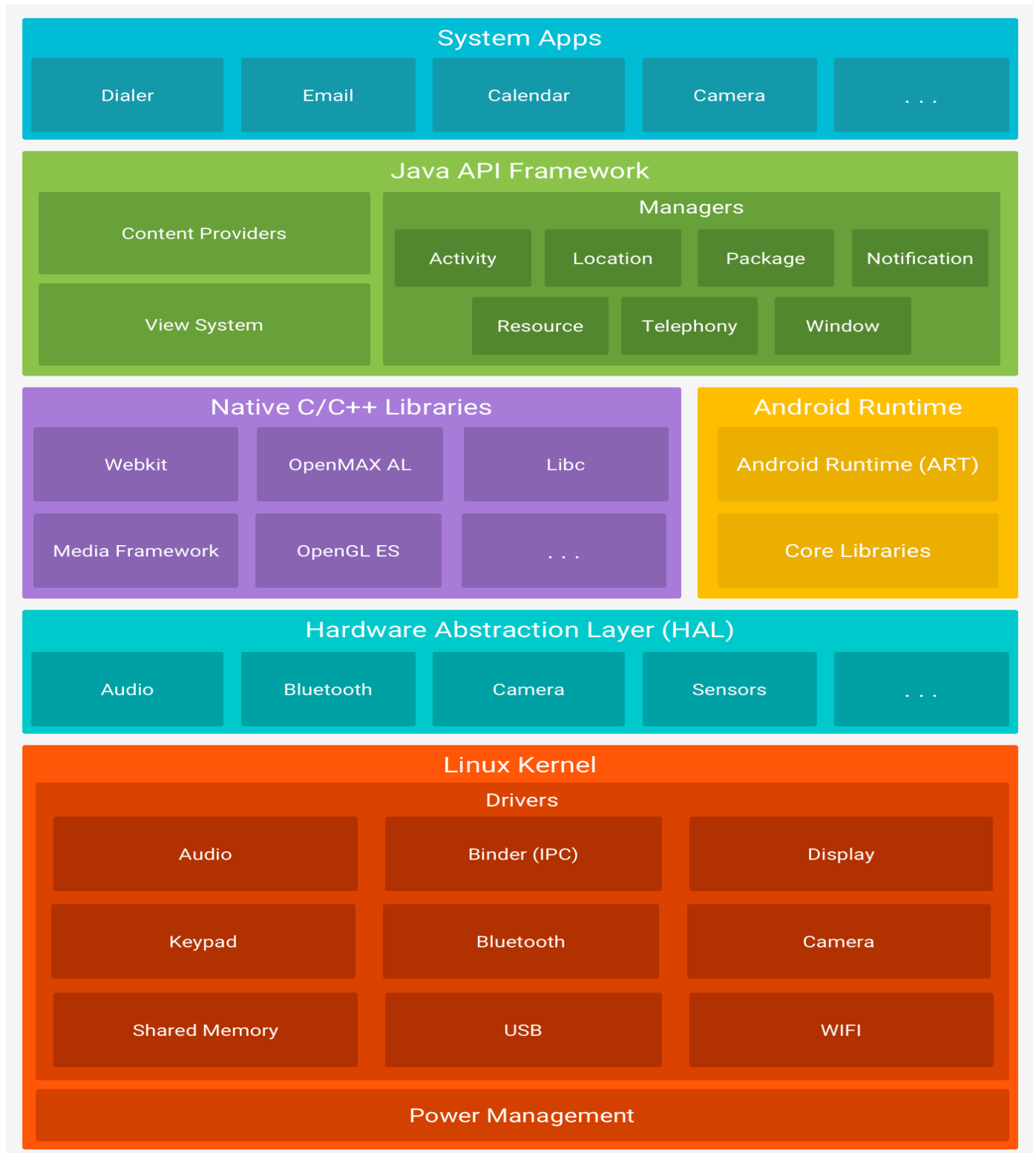


Figure. The Android software stack.

Linux kernel

The foundation of the Android platform is the Linux kernel. For example, the Android Runtime (ART) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management.

Using a Linux kernel lets Android take advantage of key security features and lets device manufacturers develop hardware drivers for a well-known kernel.

Hardware abstraction layer (HAL)

The hardware abstraction layer (HAL) provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or Bluetooth module. When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

Android runtime

For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART). ART is written to run multiple virtual machines on low-memory devices by executing Dalvik Executable format (DEX) files, a bytecode format designed specifically for Android that's optimized for a minimal memory footprint. Build tools, such as d8, compile Java sources into DEX bytecode, which can run on the Android platform.

Some of the major features of ART include the following:

- Ahead-of-time (AOT) and just-in-time (JIT) compilation
- Optimized garbage collection (GC)
- On Android 9 (API level 28) and higher, conversion of an app package's DEX files to more compact machine code
- Better debugging support, including a dedicated sampling profiler, detailed diagnostic exceptions and crash reporting, and the ability to set watchpoints to monitor specific fields

Prior to Android version 5.0 (API level 21), Dalvik was the Android runtime. If your app runs well on ART, then it can work on Dalvik as well, but the reverse might not be true.

Android also includes a set of core runtime libraries that provide most of the functionality of the Java programming language, including some Java 8 language features, that the Java API framework uses.

Native C/C++ libraries

Many core Android system components and services, such as ART and HAL, are built from native code that requires native libraries written in C and C++. The Android platform provides Java framework APIs to expose the functionality of some of these native libraries to apps. For example, you can access OpenGL ES through the Android framework's Java OpenGL API to add support for drawing and manipulating 2D and 3D graphics in your app.

If you are developing an app that requires C or C++ code, you can use the Android NDK to access some of these native platform libraries directly from your native code.

Java API framework

The entire feature-set of the Android OS is available to you through APIs written in the Java language. These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components and services, which include the following:

- A rich and extensible view system you can use to build an app's UI, including lists, grids, text boxes, buttons, and even an embeddable web browser
- A resource manager, providing access to non-code resources such as localized strings, graphics, and layout files
- A notification manager that enables all apps to display custom alerts in the status bar
- An activity manager that manages the lifecycle of apps and provides a common navigation back stack
- Content providers that enable apps to access data from other apps, such as the Contacts app, or to share their own data

Developers have full access to the same framework APIs that Android system apps use.

System apps

Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more. Apps included with the platform have no special status among the apps the user chooses to install. So, a third-party app can become the user's default web browser, SMS messenger, or even the default keyboard. Some exceptions apply, such as the system's Settings app.

The system apps function both as apps for users and to provide key capabilities that developers can access from their own app. For example, if you want your app to deliver SMS messages, you don't need to build that functionality yourself. You can instead invoke whichever SMS app is already installed to deliver a message to the recipient you specify.

1.3 Components of Android Application

There are some necessary building blocks that an Android application consists of. These loosely coupled components are bound by the application manifest file which contains the description of each component and how they interact. The manifest file also contains the app's metadata, its hardware configuration, and platform requirements, external libraries, and required permissions. There are the following main components of an android app:

1. Activities:

Activities are said to be the presentation layer of our applications. The UI of our application is built around one or more extensions of the Activity class. By using Fragments and Views, activities set the layout and display the output and also respond to the user's actions. An activity is implemented as a subclass of class Activity.

```
class MainActivity : AppCompatActivity() {  
}
```

2. Services:

Services are like invisible workers of our app. These components run at the backend, updating your data sources and Activities, triggering Notification, and also broadcast Intents. They also perform some tasks when applications are not active. A service can be used as a subclass of class Service:

```
class ServiceName : Service() {  
  
}
```

3. Content Providers:

It is used to manage and persist the application data also typically interacts with the SQL database. They are also responsible for sharing the data beyond the application boundaries. The Content Providers of a particular application can be configured to allow access from other applications, and the Content Providers exposed by other applications can also be configured.

A content provider should be a sub-class of the class ContentProvider.

```
class contentProviderName : ContentProvider() {  
  
    override fun onCreate(): Boolean {}  
  
}
```

4. Broadcast Receivers:

They are known to be intent listeners as they enable your application to listen to the Intents that satisfy the matching criteria specified by us. Broadcast Receivers make our application react to any received Intent thereby making them perfect for creating event-driven applications.

5. Intents:

It is a powerful inter-application message-passing framework. They are extensively used throughout Android. Intents can be used to start and stop Activities and Services, to broadcast messages system-wide or to an explicit Activity, Service or Broadcast Receiver or to request action be performed on a particular piece of data.

6. Widgets:

These are the small visual application components that you can find on the home screen of the devices. They are a special variation of Broadcast Receivers that allow us to create dynamic, interactive application components for users to embed on their Home Screen.

7. Notifications:

Notifications are the application alerts that are used to draw the user's attention to some particular app event without stealing focus or interrupting the current activity of the user. They are generally used to grab user's attention when the application is not visible or active, particularly from within a Service or Broadcast Receiver. Examples: E-mail popups, Messenger popups, etc.

1.4 Android Manifest File in Android

Every project in Android includes a Manifest XML file, which is `AndroidManifest.xml`, located in the root directory of its project hierarchy. The manifest file is an important part of our app because it defines the structure and metadata of our application, its components, and its requirements. This file includes nodes for each of the Activities, Services, Content Providers, and Broadcast Receivers that make the application, and using Intent Filters and Permissions determines how they coordinate with each other and other applications.

The manifest file also specifies the application metadata, which includes its icon, version number, themes, etc., and additional top-level nodes can specify any required permissions, and unit tests, and define hardware, screen, or platform requirements. The manifest comprises a root manifest tag with a package attribute set to the project's package. It should also include an `xmlns:android` attribute that will supply several system attributes used within the file. We use the `versionCode` attribute is used to define the current application version in the form of an integer that increments itself with the iteration of the version due to update. Also, the `versionName` attribute is used to specify a public version that will be displayed to the users.

We can also specify whether our app should install on an SD card of the internal memory using the `installLocation` attribute. A typical manifest file looks as:

Example: XML

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.application"
    android:versionCode="1"
    android:versionName="1.0"
    android:installLocation="preferExternal">

    <uses-sdk
        android:minSdkVersion="18"
        android:targetSdkVersion="27" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
```

```
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/Theme.MyApplication"
tools:targetApi="31">
<activity>
    android:name=".MainActivity"
    android:exported="true">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

A manifest file includes the nodes that define the application components, security settings, test classes, and requirements that make up the application. Some of the manifest sub-node tags that are mainly used are:

1. manifest

The main component of the AndroidManifest.xml file is known as manifest. Additionally, the packaging field describes the activity class's package name. It must contain an <application> element with the xmlns:android and package attribute specified.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.application">
    <!-- manifest nodes -->
    <application>
    </application>
</manifest>
```

2. uses-sdk:

It is used to define a minimum and maximum SDK version by means of an API Level integer that must be available on a device so that our application functions properly, and the target SDK for which it has been designed using a combination of minSdkVersion, maxSdkVersion, and targetSdkVersion attributes, respectively. It is contained within the <manifest> element.

```
<uses-sdk
```

```
    android:minSdkVersion="18"
```

```
    android:targetSdkVersion="27" />
```

3. uses-permission

It outlines a system permission that must be granted by the user for the app to function properly and is contained within the `<manifest>` element. When an application is installed (on Android 5.1 and lower devices or Android 6.0 and higher), the user must grant the application permissions.

```
<uses-permission
```

```
    android:name="android.permission.CAMERA"
```

```
    android:maxSdkVersion="18" />
```

4. application

A manifest can contain only one application node. It uses attributes to specify the metadata for your application (including its title, icon, and theme). During development, we should include a debuggable attribute set to true to enable debugging, then be sure to disable it for your release builds. The application node also acts as a container for the Activity, Service, Content Provider, and Broadcast Receiver nodes that specify the application components. The name of our custom application class can be specified using the `android:name` attribute.

```
<application
```

```
    android:name=".Application"
```

```
    android:allowBackup="true"
```

```
    android:dataExtractionRules="@xml/data_extraction_rules"
```

```
    android:fullBackupContent="@xml/backup_rules"
```

```
    android:icon="@drawable/gfgIcon"
```

```
    android:label="@string/app_name"
```

```
    android:roundIcon="@mipmap/ic_launcher_round"
```

```
    android:supportsRtl="true"
```

```
    android:theme="@android:style/Theme.Light"
```

```
    android:debuggable="true"
```

```
    tools:targetApi="31">
```

```
    <!-- application nodes -->
```

```
</application>
```

5. uses-library

It defines a shared library against which the application must be linked. This element instructs the system to add the library's code to the package's class loader. It is contained within the `<application>` element.

```
<uses-library
```

```
android:name="android.test.runner"
```

```
android:required="true" />
```

6. activity

The Activity sub-element of an application refers to an activity that needs to be specified in the AndroidManifest.xml file. It has various characteristics, like label, name, theme, launchMode, and others. In the manifest file, all elements must be represented by <activity>. Any activity that is not declared there won't run and won't be visible to the system. It is contained within the <application> element.

```
<activity
```

```
    android:name=".MainActivity"
```

```
    android:exported="true">
```

```
</activity>.
```

7. intent-filter

It is the sub-element of activity that specifies the type of intent to which the activity, service, or broadcast receiver can send a response. It allows the component to receive intents of a certain type while filtering out those that are not useful for the component. The intent filter must contain at least one <action> element.

```
<intent-filter>
```

```
    <action android:name="android.intent.action.MAIN" />
```

```
    <category android:name="android.intent.category.LAUNCHER" />
```

```
</intent-filter>.
```

8. action

It adds an action for the intent-filter. It is contained within the <intent-filter> element.

```
<action android:name="android.intent.action.MAIN" />
```

9. category

It adds a category name to an intent-filter. It is contained within the <intent-filter> element.

```
<category android:name="android.intent.category.LAUNCHER" />
```

.

10. uses-configuration

The uses-configuration components are used to specify the combination of input mechanisms that are supported by our application. It is useful for games that require particular input controls.

```
<uses-configuration
```

```
    android:reqTouchScreen="finger"
```

```
    android:reqNavigation="trackball"
```

```
    android:reqHardKeyboard="true"
```

```
android:reqKeyboardType="qwerty"/>
```

```
<uses-configuration
```

```
    android:reqTouchScreen="finger"
```

```
    android:reqNavigation="trackball"
```

```
    android:reqHardKeyboard="true"
```

```
    android:reqKeyboardType="twelvekey"/>
```

11. uses-features

It is used to specify which hardware features your application requires. This will prevent our application from being installed on a device that does not include a required piece of hardware such as NFC hardware, as follows

```
<uses-feature android:name="android.hardware.nfc" />
```

12. permission

It is used to create permissions to restrict access to shared application components. We can also use the existing platform permissions for this purpose or define your own permissions in the manifest.

```
<permission
```

```
    android:name="com.paad.DETONATE_DEVICE"
```

```
    android:protectionLevel="dangerous"
```

```
    android:label="Self Destruct"
```

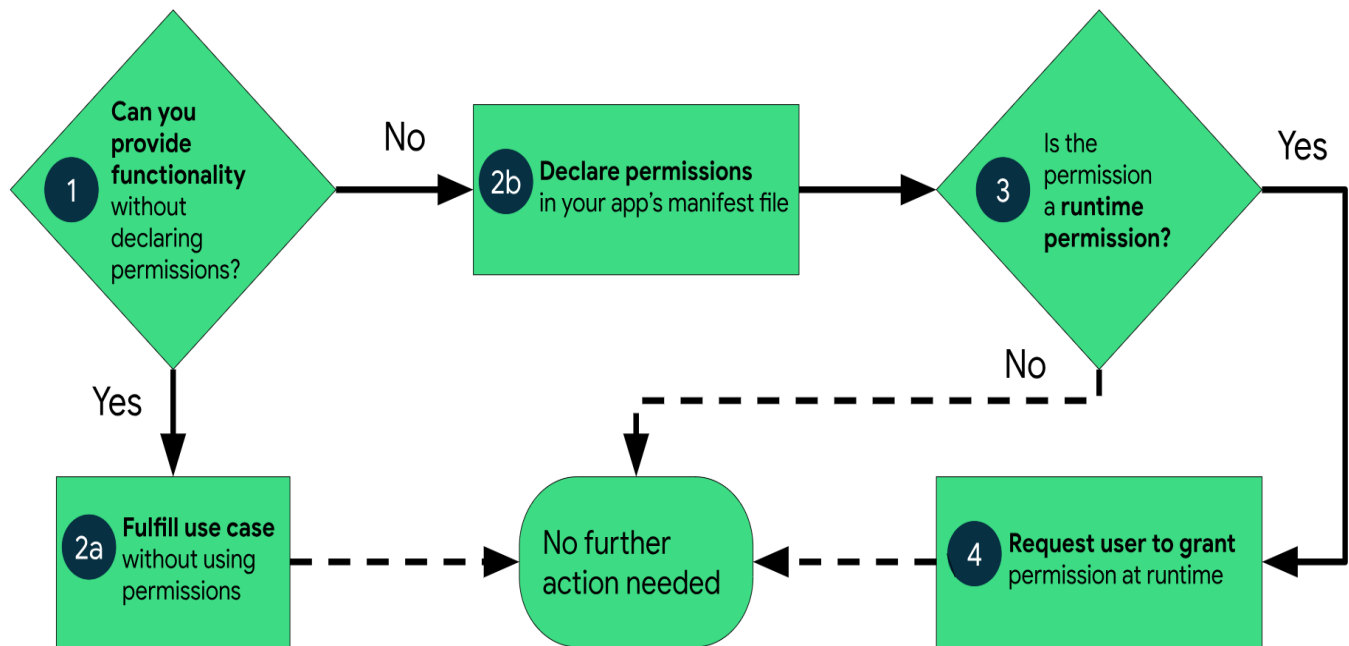
```
    android:description="@string/detonate_description">
```

```
</permission>
```

1.5. Permission Model of Android

If your app offers functionality that might require access to restricted data or restricted actions, determine whether you can get the information or perform the actions without needing to declare permissions. You can fulfill many use cases in your app, such as taking photos, pausing media playback, and displaying relevant ads, without needing to declare any permissions.

If you decide that your app must access restricted data or perform restricted actions to fulfill a use case, declare the appropriate permissions. Some permissions, known as install-time permissions, are automatically granted when your app is installed. Other permissions, known as runtime permissions, require your app to go a step further and request the permission at runtime.



Types of permission:

1. Install-time permissions

Install-time permissions give your app limited access to restricted data or let your app perform restricted actions that minimally affect the system or other apps. When you declare install-time permissions in your app, an app store presents an install-time permission notice to the user when they view an app's details page, as shown in figure 2. The system automatically grants your app the permissions when the user installs your app.

2. Normal permissions

These permissions allow access to data and actions that extend beyond your app's sandbox but present very little risk to the user's privacy and the operation of other apps.

3. Signature permissions

The system grants a signature permission to an app only when the app is signed by the same certificate as the app or the OS that defines the permission.

Applications that implement privileged services, such as autofill or VPN services, also make use of signature permissions. These apps require service-binding signature permissions so that only the system can bind to the services.

4. Runtime permissions

Runtime permissions, also known as dangerous permissions, give your app additional access to restricted data or let your app perform restricted actions that more substantially affect the system and other apps. Therefore, you need to request runtime permissions in your app before you can access the restricted data or perform restricted actions. Don't assume that these permissions have been previously granted—check them and, if needed, request them before each access.

When your app requests a runtime permission, the system presents a runtime permission prompt.

Many runtime permissions access *private user data*, a special type of restricted data that includes potentially sensitive information. Examples of private user data include location and contact information.

5.Special permissions

Special permissions correspond to particular app operations. Only the platform and OEMs can define special permissions. Additionally, the platform and OEMs usually define special permissions when they want to protect access to particularly powerful actions, such as drawing over other apps.

The Special app access page in system settings contains a set of user-toggable operations. Many of these operations are implemented as special permissions.

6. Permission groups

Permissions can belong to permission groups. Permission groups consist of a set of logically related permissions. For example, permissions to send and receive SMS messages might belong to the same group, as they both relate to the application's interaction with SMS.

Permission groups help the system minimize the number of system dialogs that are presented to the user when an app requests closely related permissions. When a user is presented with a prompt to grant permissions for an application, permissions belonging to the same group are presented in the same interface. However, permissions can change groups without notice, so don't assume that a particular permission is grouped with any other permission.

1.6. Downloading and Installing SDK

Install the SDK

1. Click Tools > SDK Manager.
2. In the SDK Platforms tab, select Android UpsideDownCake Preview.
3. In the SDK Tools tab, select Android SDK Build-Tools 29.
4. Click OK to install the SDK.

1.7. Exploring the Development Environment

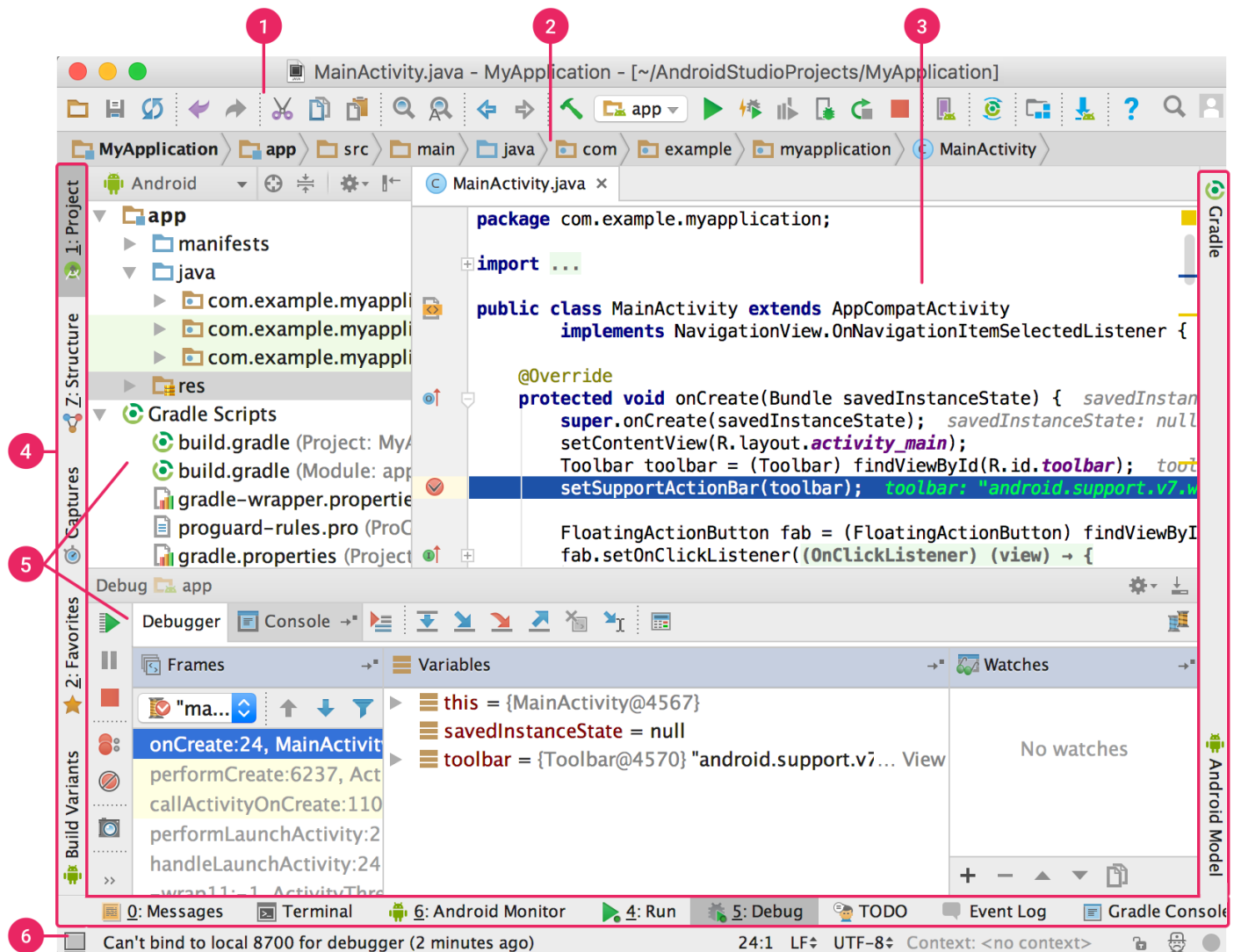
Android Studio is the official Integrated Development Environment (IDE) for Android app development.

Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Live Edit to update composables in emulators and physical devices in real time

- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine.

Get to know the Android Studio UI



1. **Toolbar:** Carry out a wide range of actions, including running your app and launching Android tools.
2. **Navigation bar:** Navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.
3. **Editor window:** Create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
4. **Tool window bar:** Use the buttons on the outside of the IDE window to expand or collapse individual tool windows.

5. **Tool windows:** Access specific tasks like project management, search, version control, and more. You can expand them and collapse them.
6. **Status bar:** Display the status of your project and the IDE itself, as well as any warnings or messages.

Tool windows

To expand or collapse a tool window, click the tool's name in the tool window bar. You can also drag, pin, unpin, attach, and detach tool windows.

To return to the default layout of the current tool window, click Window > Restore Default Layout. To customize your default layout, click Window > Store Current Layout as Default.

To show or hide the entire tool window bar, click the window  in the bottom left-hand corner of the Android Studio window.

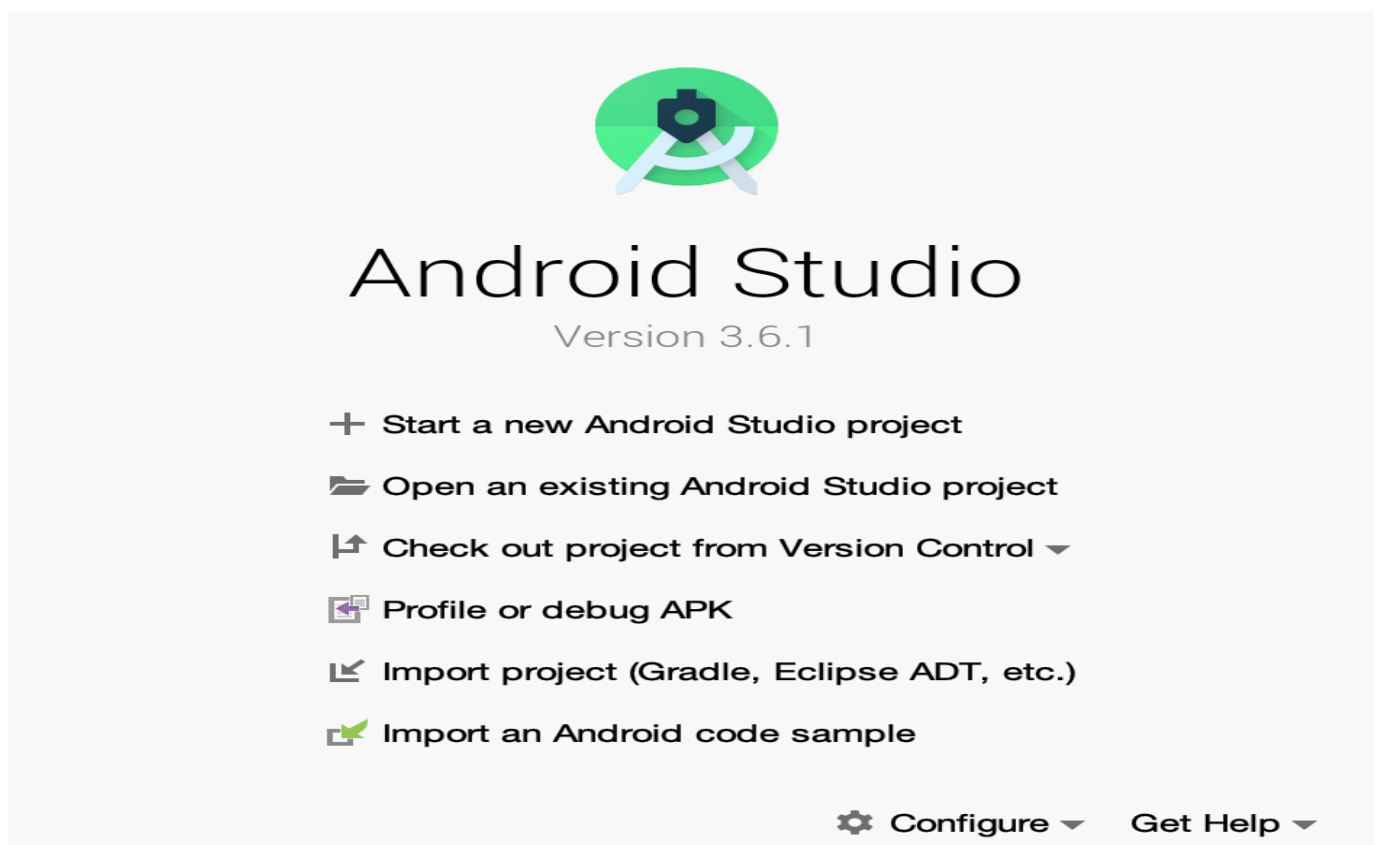
To locate a specific tool window, hover over the window icon and select the tool window from the menu.

1.8 Build Your First Android App in Kotlin

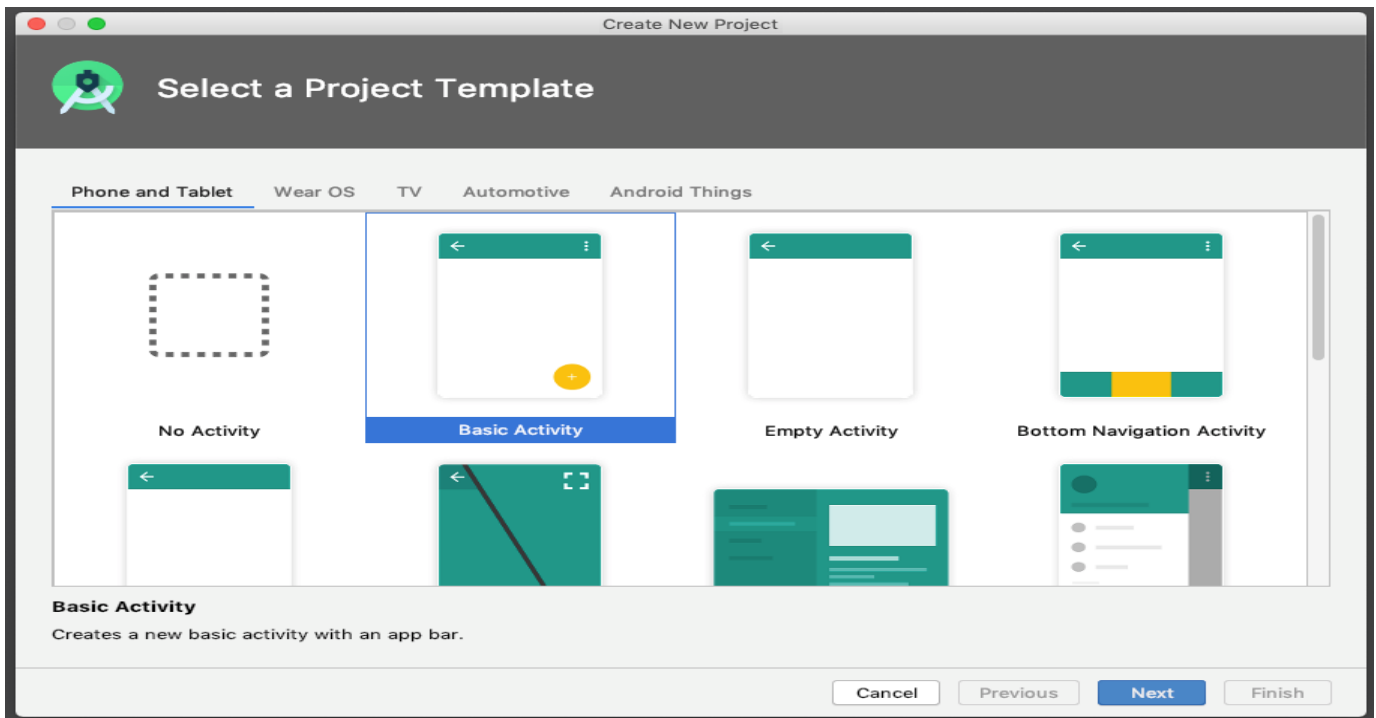
1.Install Android Studio

2.Create your first project

click Start a new Android Studio project.



Select Basic Activity (not the default). Click Next.



Give your application a name, such as My First App.

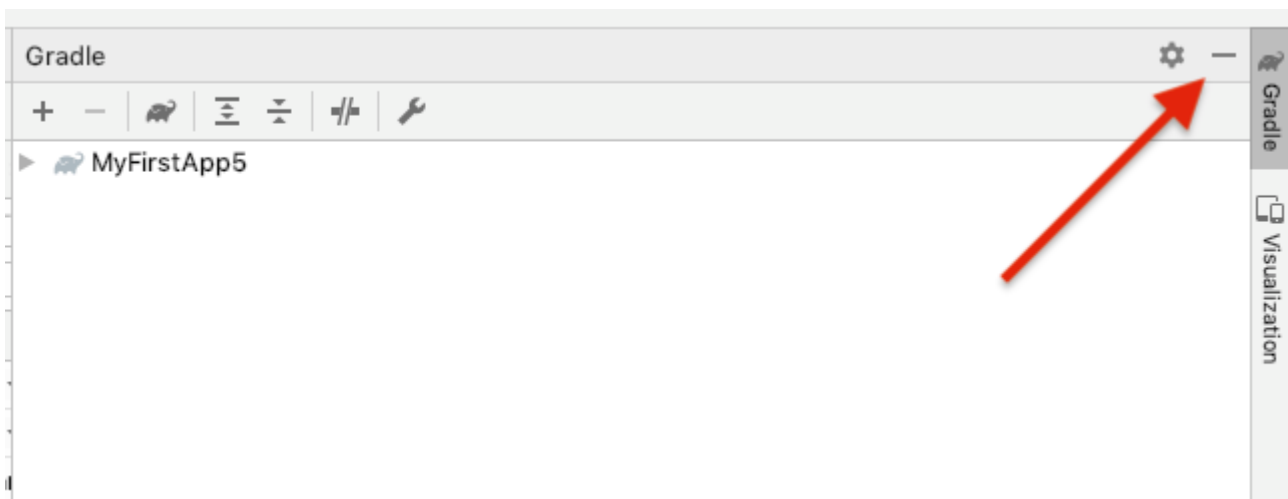
Make sure the Language is set to Kotlin.

Leave the defaults for the other fields.

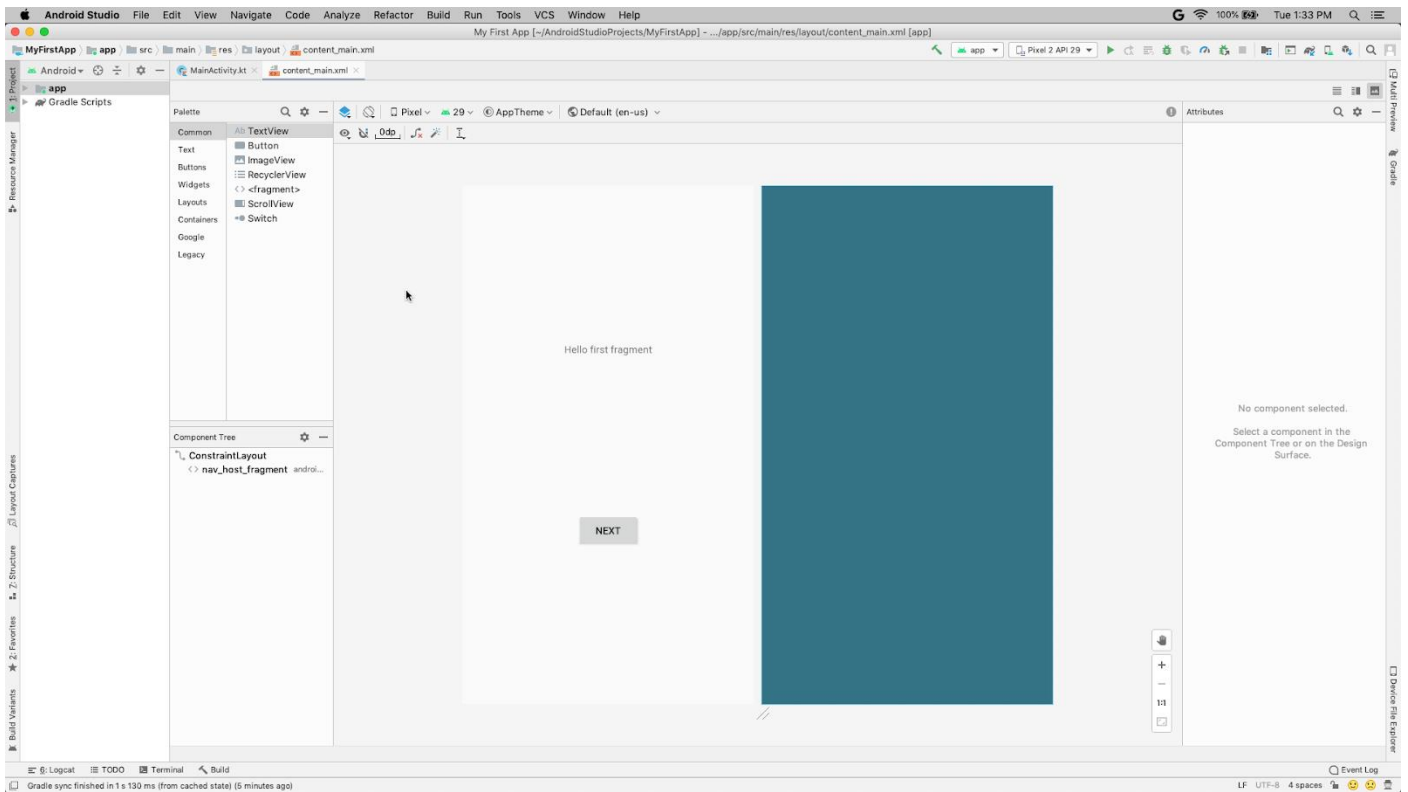
Click Finish.

Get your screen set up

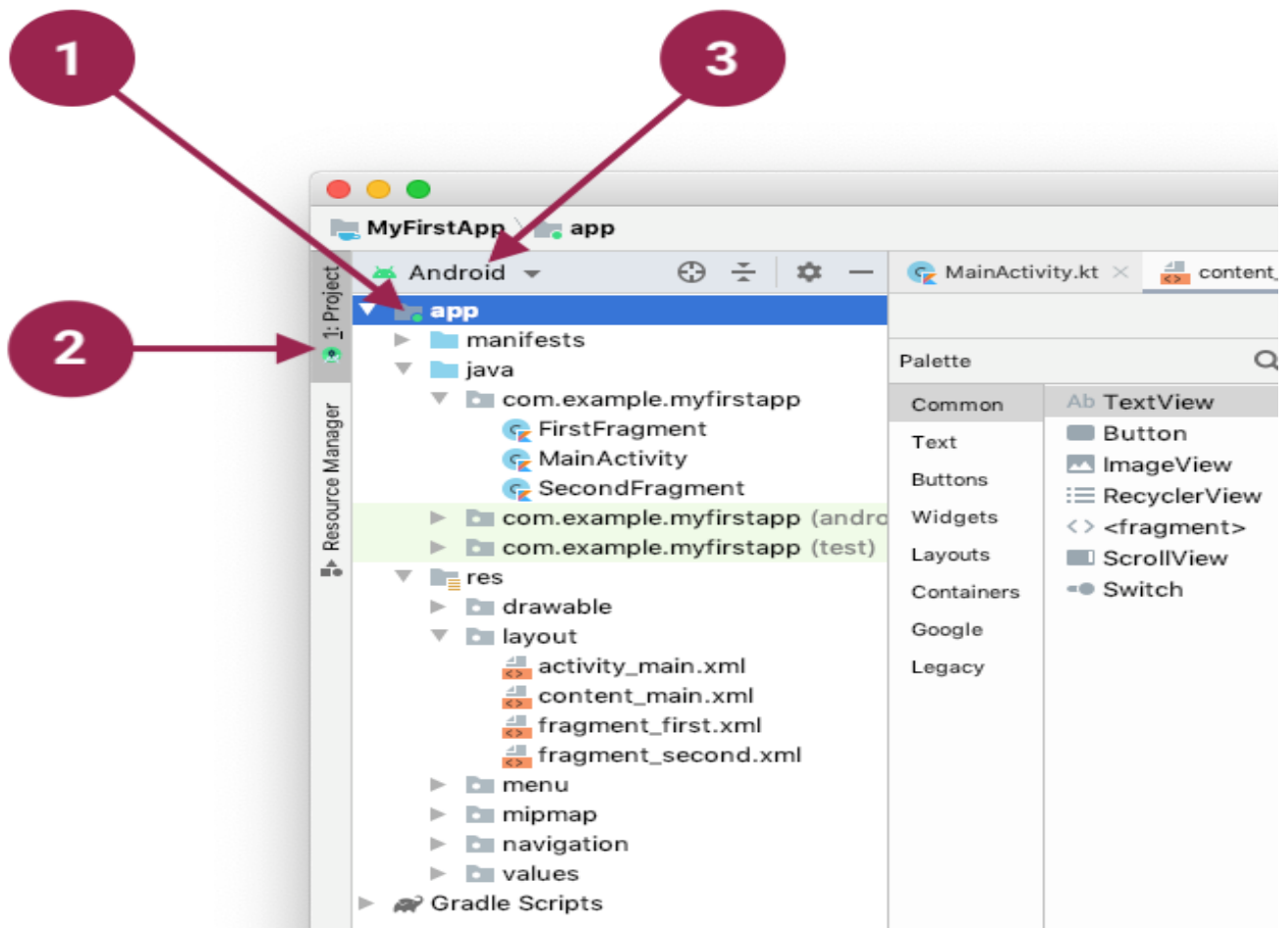
If there's a **Gradle** window open on the right side, click on the minimize button (—) in the upper right corner.



Depending on the size of your screen, consider resizing the pane on the left showing the project folders to take up less space.



Explore the project structure and layout



Double-click the app (1) folder to expand the hierarchy of app files. (See (1) in the screenshot.)

If you click Project (2), you can hide or show the Project view.

The current Project view selection is Project > Android.

Create a virtual device (emulator)

In Android Studio, select Tools > AVD Manager, or click the AVD Manager icon in the toolbar.
1ef215721ed1bd47.png

Click +Create Virtual Device. (If you have created a virtual device before, the window shows all of your existing devices and the +Create Virtual Device button is at the bottom.) The Select Hardware window shows a list of pre-configured hardware device definitions.

Choose a device definition, such as Pixel 2, and click Next. (For this codelab, it really doesn't matter which device definition you pick).

In the System Image dialog, from the Recommended tab, choose the latest release. (This does matter.)



If a Download link is visible next to a latest release, it is not installed yet, and you need to download it first. If necessary, click the link to start the download, and click Next when it's done. This may take a while depending on your connection speed.

In the next dialog box, accept the defaults, and click Finish.

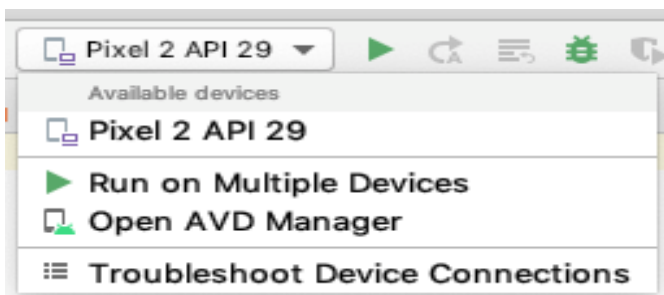
The AVD Manager now shows the virtual device you added.

If the Your Virtual Devices AVD Manager window is still open, go ahead and close it.

Run your app on your new emulator]

In Android Studio, select Run > Run 'app', or click the Run icon in the toolbar.  The icon changes once your app is running 

In Run > Select Device, under Available devices, select the virtual device that you just configured. A dropdown menu also appears in the toolbar.



Once your app builds and the emulator is ready, Android Studio uploads the app to the emulator and runs it. You should see your app as shown in the following screenshot.

11:37 



My First App



Hello first fragment

NEXT

