Android Studio Setup & Folder Structure for Android projects

Android Studio is the official Integrated Development Environment (IDE) for Android app development. Based on the powerful code editor and developer tools from IntelliJ IDEA , 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**

How To Setup Android Studio

Here are the system requirements for Windows:

REQUIREMENT	MINIMUM	RECOMMENDED
OS	64-bit Microsoft Windows 8	Latest 64-bit version of Windows
RAM	8 GB RAM	16 GB RAM or more
CPU	x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor Framework.	Latest Intel Core processor
Disk space	8 GB (IDE and Android SDK and Emulator)	Solid state drive with 16 GB or more
Screen resolution	1280 x 800	1920 x 1080

- **Download and Install Android Studio:**
  - Visit the official Android Studio website: https://developer.android.com/studio
  - Download the latest stable version for your operating system (Windows, macOS, or Linux).
  - Follow the installation instructions provided on the setup dialog box.
  - Launch studio64.exe (for 64-bit machines) or studio.exe (for 32-bit machines).
  - Follow the Setup Wizard in Android Studio and install any recommended SDK packages.

https://developer.android.com/studio/install

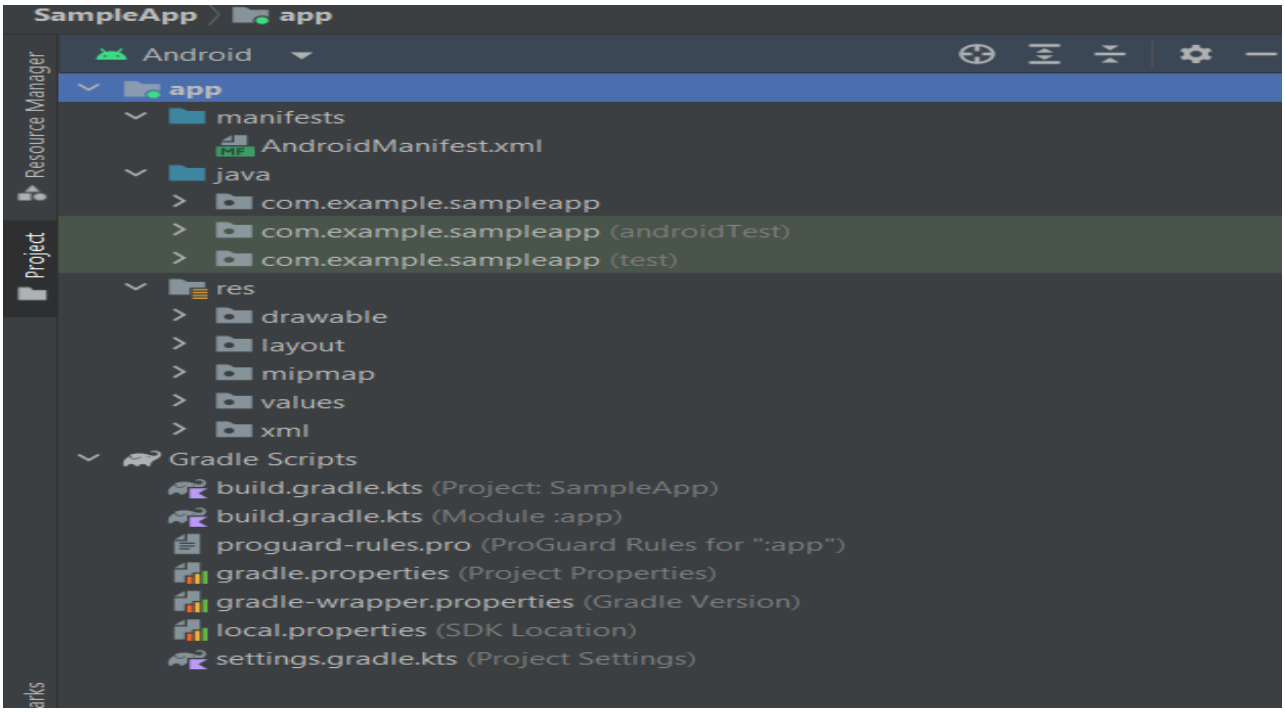
Android Project Folder Structure

Each project in Android Studio contains one or more modules with source code files and resource files. The types of modules include:

- **Android app modules**
- **Library modules**
- **Google App Engine modules**

By default, Android Studio displays your project files in the Android project view, as shown in below figure. This view is organized by modules to provide quick access to your project's key source files. All the build files are visible at the top level,underGradleScripts.

Once you create a new Android project, you will find a standard folder structure:



1. app:

- **manifests: Contains the AndroidManifest.xml file, it Describes essential information about your app, like activities, permissions, etc.**
- **java: Contains the Kotlin and Java source code files, including JUnit test code.**

- **res:** Contains all non-code resources such as UI strings, layout files, drawables, values and bitmap images.
    - **drawable:** It is used to store various types of drawable resources for your app. Drawables are visual assets such as vector images, icons, and shapes that are used to enhance the user interface of your application.
    - **layout:** It is used to store XML layout files. These layout files define the structure and appearance of the user interface for your app. The layout directory can contain different layouts for various screens and orientations, allowing your app to adapt to different device configurations.
    - **mipmap:** The mipmap directory in an Android project is used to store launcher icons for different densities. Launcher icons are the images that represent your app on the device's home screen and in the app drawer. The mipmap directory is similar to the drawable directory, but it is specifically used for launcher icons to ensure that the appropriate icon is used on different screen densities.
    - **values:** It is used to store various XML files that define values, such as strings, colors, dimensions, styles, and other resources.These resources are then referenced in your app's layout files, manifest, and code. The values directory helps you centralize and manage these resources, making it easier to maintain a consistent design and behavior across your app.
    - **xml:** xml directory used to organize XML files related to various configurations, data, or settings.

2. Gradle Scripts:

- **build.gradle.kts (Project):** The build.gradle.kts file for the project is a Kotlin DSL (Domain-Specific Language) script used to configure the build settings for the entire Android project and it is used to specify settings that apply to the entire project, such as the build tools version, repositories, and global dependencies.
- **build.gradle.kts (Module):** The build.gradle.kts file for a module in an Android project is used to configure settings specific to that module and it

includes information about the module's dependencies, build types, product flavors, and other settings.

- **proguard-rules.pro:** The proguard-rules.pro file in an Android project is used to specify rules for ProGuard, a code shrinker, optimizer, and obfuscator. ProGuard is often used in Android development to reduce the size of the APK, improve runtime performance, and obfuscate the code to make reverse engineering more difficult.It's important to carefully configure ProGuard to strike a balance between code size reduction and preserving the necessary information for your app to function correctly.
- **gradle.properties :** The gradle.properties file in an Android project is used to define properties that can be used in the Gradle build scripts. It allows you to externalize configuration values, such as API keys, version numbers, or other settings, making it easier to manage and share these values across the project..
- **gradle-wrapper.properties:** The gradle-wrapper.properties file is a configuration file used by the Gradle Wrapper, which is a tool provided by Gradle to ensure that a specific version of Gradle is used for building a project. The Gradle Wrapper simplifies the process of managing Gradle versions and helps ensure consistency across different environments and development teams.
- **local.properties:** The local.properties file is used to store local configuration settings for an Android project, primarily related to the SDK location. It is an autogenerated file that should not be manually modified in most cases. The Android Studio IDE typically creates and manages this file automatically based on your local development environment.
- **settings.gradle.kts:** The settings.gradle.kts file in an Android project is a Kotlin script that is used to configure settings for the entire project, including defining which modules (sub-projects) are part of the project and helps organize the structure of a multi-module Android project.

What is Gradle

Gradle is an advanced build automation system that is used for managing and building projects. It is an open-source build tool that combines the flexibility of Apache Ant with the convention-over-configuration and opinionated nature of Apache Maven. Gradle is often used in Java, Kotlin, and Android development, but it is not limited to these languages and can be employed for building projects in various programming languages.

Key features of the Gradle build system include:

1. Declarative Build Scripts: Gradle build scripts are written in either Groovy or Kotlin and are designed to be concise, readable, and expressive. They use a declarative syntax that allows developers to describe the desired state of their projects rather than focusing on procedural details.

2. Plugin System: Gradle has a powerful plugin system that allows developers to extend and customize the build process. There are many plugins available for common tasks, and developers can also create their own plugins to meet specific project requirements.

3. Incremental Builds: Gradle is designed to perform incremental builds, which means that it only rebuilds parts of the project that have changed since the last build. This feature speeds up the build process and is especially useful in large projects.

4. Dependency Management: Gradle handles dependency management by allowing developers to declare dependencies on external libraries and frameworks. It can automatically download dependencies from repositories such as Maven Central and manage their versions.

5. Multi-Project Builds: Gradle supports multi-project builds, allowing developers to manage and build multiple projects within a single build. This is particularly useful for complex projects with interdependent modules.

6. Task-Based Build Process: The build process in Gradle is organized into tasks. Developers can define custom tasks and configure existing ones. Tasks can be executed individually or as part of a larger build process.

7. Integration with IDEs: Gradle integrates well with popular Integrated Development Environments (IDEs) such as IntelliJ IDEA and Android Studio. Developers can import Gradle projects into their IDEs and leverage advanced build features directly from the IDE.

8. Extensibility: Gradle is highly extensible and can be customized to suit the specific needs of a project. Developers can create custom plugins, tasks, and even extend the build process itself.

9. Community and Ecosystem: Gradle has a vibrant community, and its ecosystem includes a wide range of plugins, documentation, and community support. This makes it easy for developers to find solutions and share best practices.

Gradle is widely used in the Java and Android development communities, and its adoption continues to grow across different domains due to its flexibility, scalability, and powerful features.

https://gradle.org/