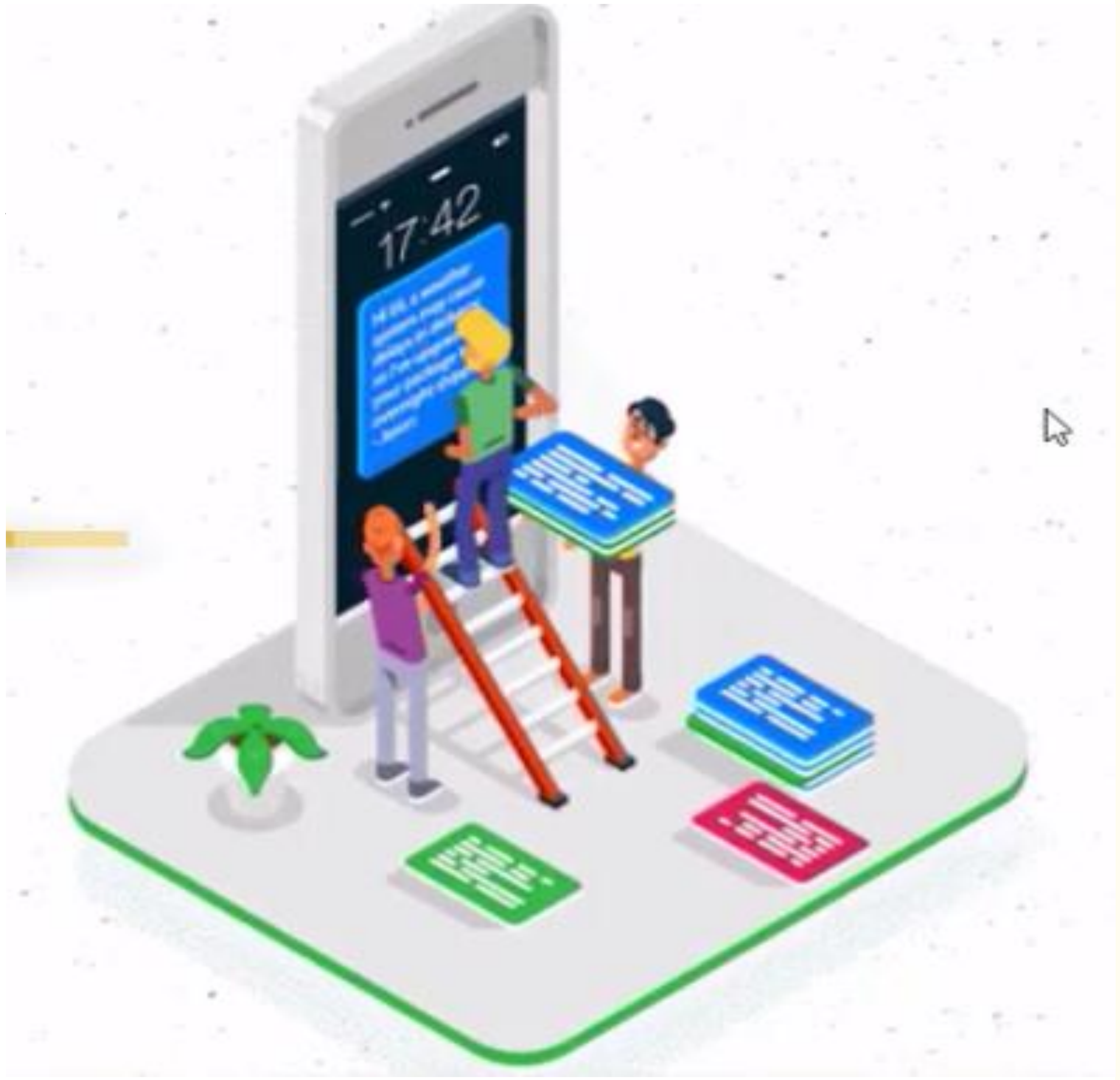
Course Name:-Mobile Application Development

Chapter one: Introduction to Flutter Development

compiled by Samuel. A

What are mobile applications?

- ❖ In mobile apps: the client device is expected to be a pocket-fitting device.
  - Smart phones
  - palm-size or handheld computers...
- ❖ Those apps are targeted for devices of the macro- mobile environment.
  - Small size
  - limited bandwidth
  - low processing power.


17:42

3

Why are mobile applications needed?

- ❖ People are moving even more than before
  - ◦ They must still be able to manage contacts and events and access networked information.
- ❖ The web created new needs for availability of information.
  - ◦ Instant access to needed data.
  - ◦ Data must be available 24h a day.

Overview of Flutter

What is Flutter?

- ❑Flutter is a tool that allows developers to build native cross-platform apps with just one programming language and also one codebase. it creates a native app for both iOS and Android that can be published to the stores later.
- ❑The good news is that you can create this app just by using one programming language, rather than using separate ones to build an iOS app or an Android app. That way, you will have just one code base to worry about.
- ❑Flutter is an SDK (Software Development Kit) that allows you to compile your codebase into native machine code that runs on the platforms mentioned above.
- ❑Flutter acts as a framework by providing a collection of UI building blocks (widgets) like tabs, dropdowns, buttons, etc., some utility functions; and some packages that will get compiled by using the SDK.

Overview of Flutter

What is Dart?

- ❑ Dart is the programming language that Flutter uses. It is focused on building front-end user interfaces and front-end apps. By using it, you will be able to create either web apps or mobile apps.

- ❑ It is developed by Google and is a class-based, object-oriented, and strongly typed programming language. Dart's syntax is very similar to languages like Java or JavaScript, so if you have some experience with those, it will be easy for you to begin using it.

Flutter vs. Other Frameworks

Futter vs. Native Development (iOS and Android):

- **Development Time:** Flutter can potentially reduce development time by allowing developers to write code once for both platforms. Native development requires separate codebases for iOS and Android.
- **Performance:** Native apps typically offer the best performance as they directly leverage platform-specific APIs and components. Flutter apps have good performance but may have a slight overhead due to the Flutter engine.
- **UI Consistency:** Flutter provides a consistent UI experience across platforms since it uses its own rendering engine and widgets. Achieving the same level of consistency in native development may require additional effort.
- **Community and Ecosystem:** Native development has a mature ecosystem with extensive community support, libraries, and resources. Flutter's ecosystem is growing rapidly but may not be as extensive as native development yet.

Flutter vs. Other Frameworks

Flutter vs. React Native:

- **Programming Language:** Flutter uses Dart, while React Native uses JavaScript.
- **Performance:** Flutter tends to offer better performance due to its use of a compiled programming language (Dart) and its own rendering engine. React Native relies on a bridge to communicate with native components, which can introduce performance overhead.
- **UI Components:** Flutter provides a rich set of customizable UI components known as widgets, which are consistent across platforms. React Native uses native components, which can sometimes result in platform-specific UI differences.
- **Hot Reload:** Both Flutter and React Native offer hot reload functionality, allowing developers to see changes instantly during development.
- **Community Support:** React Native has a larger community and ecosystem due to its longer time in the market, resulting in more third-party libraries and resources.

Setting Up Development Environment
https://docs.flutter.dev/get-started/install/windows

- Download then install Flutter
- Update your Windows PATH variable
- Check your development setup
- Run Flutter doctor
  - Open PowerShell.
  - To verify your installation of all the components, run the following command.
    - flutter doctor
  - Troubleshoot Flutter doctor issues
    - flutter doctor -v

Setting Up Development Environment

https://docs.flutter.dev/get-started/install/windows

- **Set up an editor**

You can build apps with Flutter using any text editor or integrated development environment (IDE) combined with Flutter's command-line tools. The Flutter team recommends using an editor that supports a Flutter extension or plugin, like VS Code and Android Studio.

Setting Up Development Environment https://docs.flutter.dev/get-started/install/windows

- ❑ **Install VS Code**

VS Code is a code editor to build and debug apps. With the Flutter extension installed, you can compile, deploy, and debug Flutter apps.

- ❑ **Install the VS Code Flutter extension**

Start **VS Code**.

Open a browser and go to the Flutter extension page on the Visual Studio Marketplace.

Click **Install**. Installing the Flutter extension also installs the Dart extension.

Setting Up Development Environment

https://docs.flutter.dev/get-started/install/windows

- ❑Validate your VS Code setup

Go to View > Command Palette….

- ❑You can also press Ctrl / Cmd + Shift + P.
- ❑Type doctor.
- ❑Select the Flutter: Run Flutter Doctor.

Once you select this command, VS Code does the following.

- ❑ Opens the Output panel.
- ❑ Displays flutter (flutter) in the dropdown on the upper right of this panel
- ❑Displays the output of Flutter Doctor command.

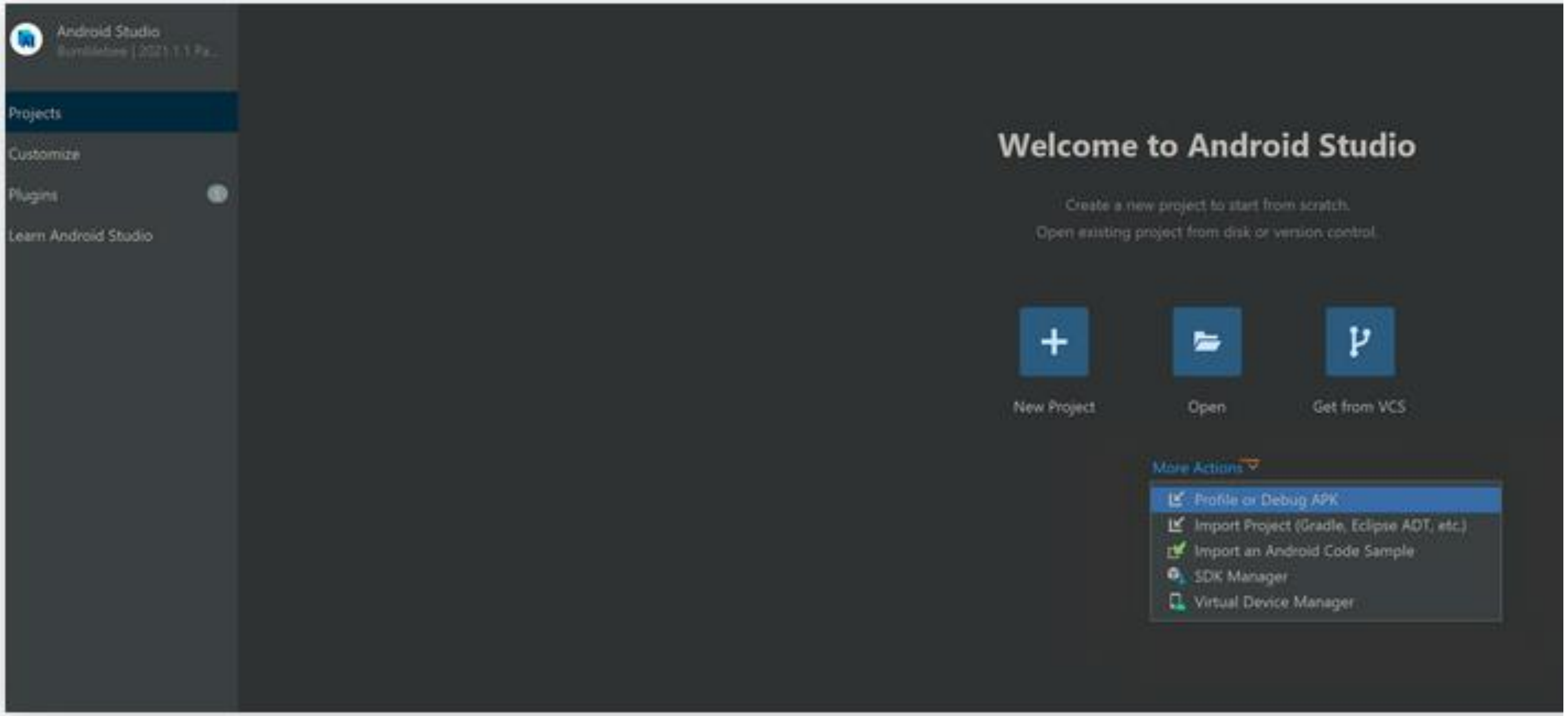
Installing the Android Debug Bridge (ADB) Tool

- ❑ Android Debug Bridge (ADB) is a command line tool that lets you communicate with a device, enabling you to access certain features of the Android platform that are not otherwise accessible.
- ❑ There are two ways to use adb: over a USB or Wi-Fi.
- ❑ **Installing ADB on Windows**

Step 1: Download and install the latest Android Studio release.

Step 2: Once installed, click the **More Actions** button, and select **SDK Manager** from the dropdown.

Installing the Android Debug Bridge (ADB) Tool



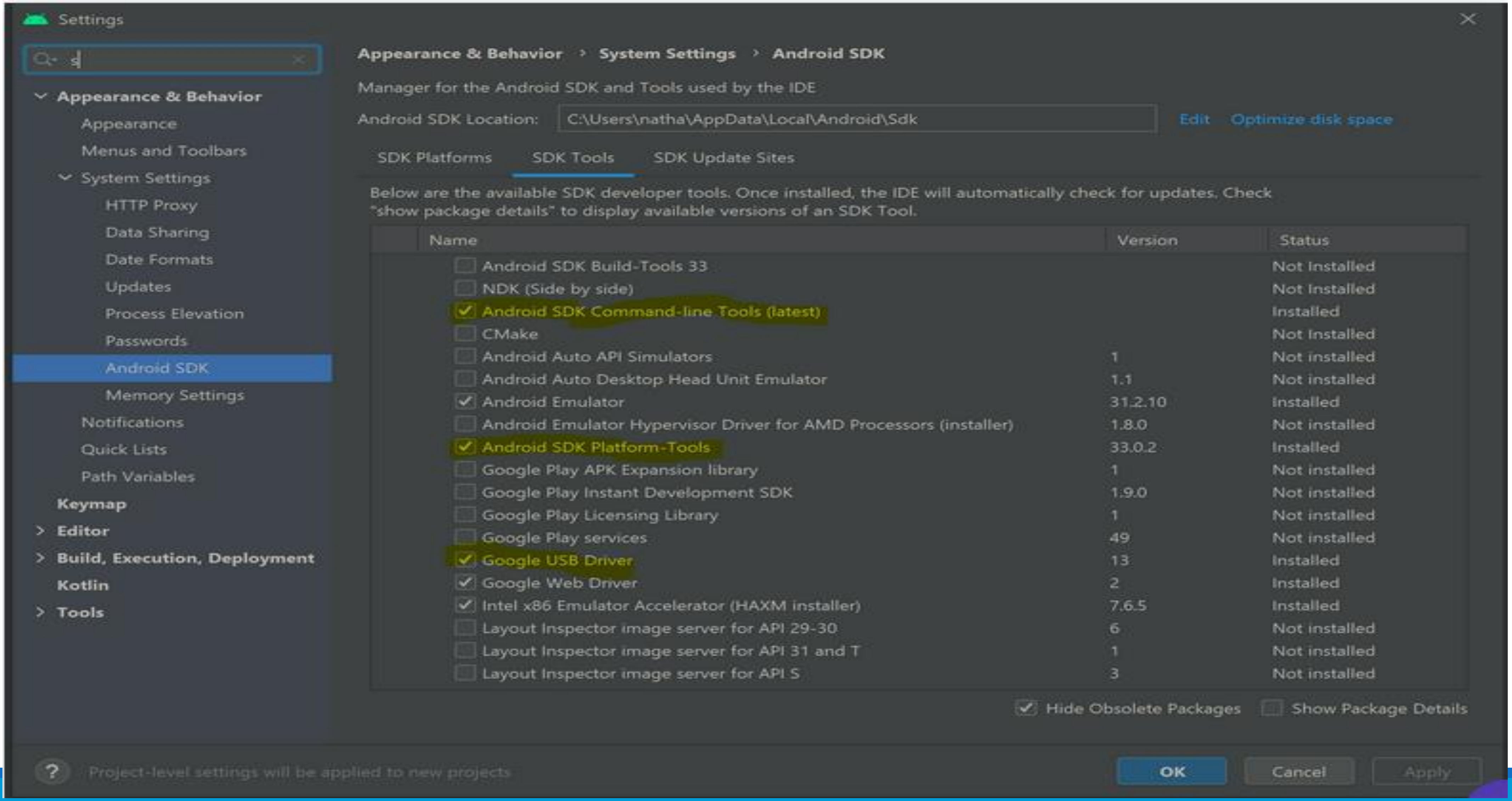
Welcome to Android Studio

Create a new project to start from scratch.
Open existing project from disk or version control.

Installing the Android Debug Bridge (ADB) Tool

- Step 3: In the SDK manager, click **SDK Tools** and select the following for installation:
  - Android SDK Command-Line tools
  - Android SDK Platform-Tools
  - Google USB Driver

Installing the Android Debug Bridge (ADB) Tool



Appearance & Behavior › System Settings › Android SDK

Manager for the Android SDK and Tools used by the IDE

Android SDK Location: C:\Users\natha\AppData\Local\Android\Sdk

Below are the available SDK developer tools. Once installed, the IDE will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

	Name	Version	Status
☐	Android SDK Build-Tools 33		Not Installed
☐	NDK (Side by side)		Not Installed
✓	Android SDK Command-line Tools (latest)		Installed
☐	CMake		Not Installed
☐	Android Auto API Simulators	1	Not installed
☐	Android Auto Desktop Head Unit Emulator	1.1	Not installed
✓	Android Emulator	31.2.10	Installed
☐	Android Emulator Hypervisor Driver for AMD Processors (installer)	1.8.0	Not installed
✓	Android SDK Platform-Tools	33.0.2	Installed
☐	Google Play APK Expansion library	1	Not installed
☐	Google Play Instant Development SDK	1.9.0	Not installed
☐	Google Play Licensing Library	1	Not installed
☐	Google Play services	49	Not installed
✓	Google USB Driver	13	Installed
✓	Google Web Driver	2	Installed
✓	Intel x86 Emulator Accelerator (HAXM installer)	7.6.5	Installed
☐	Layout Inspector image server for API 29-30	6	Not installed
☐	Layout Inspector image server for API 31 and T	1	Not installed
☐	Layout Inspector image server for API S	3	Not installed

✓ Hide Obsolete Packages ☐ Show Package Details

Project-level settings will be applied to new projects

Installing the Android Debug Bridge (ADB) Tool

- Step 4: Once installed, set the platform-tools file into the path.
    - In your search bar, type environment and click “Edit the system environment variables”.
    - Click “Environment Variables”.
    - Under “User variables,” click “New”.
    - Set the variable name to “Android”.
    - For the variable value, you need to find where your platform tools are located on your hard drive. In general, when installed with Android Studio, it will live here: C:\Users{Your username}\AppData\Local\Android\Sdk\platform-tools
    - **Note:** You need to turn on the view hidden files function to access app data.
    - Once you have verified the location of your platform tools, click “OK”.

Installing the Android Debug Bridge (ADB) Tool

- ❑Step 5: Now, verify that the path is working correctly:

  run command line > Windows key + R > cmd > enter

- ❑Step 6: Type “adb devices”, and then press enter. If pathed correctly, you will see a list of connected devices. You are now ready to use adb to troubleshoot your devices.

Running a Flutter App on your phone wirelessly

- **Step 1:** Connect Your Phone via USB Before setting up a wireless connection, make sure you have previously connected your phone via USB for initial setup and debugging. This step is necessary to install the necessary Flutter and Dart packages on your device.
- **Step 2:** Check ADB Installation Ensure that you have the Android Debug Bridge (ADB) tool installed on your computer. You can check this by running the following command in your terminal:

```
adb --version
```

- **If ADB is not installed, you can download it from the Android Studio or the Android Command Line Tools.**

Running a Flutter App on your phone wirelessly

- ❑ **Step 3:** Find Your Phone’s IP Address You’ll need your phone’s IP address to connect wirelessly. You can usually find this information in your phone’s network settings or Wi-Fi settings. Note down the IP address.

- ❑**Step 4:** Start ADB Over TCP/IP On your computer, run the following command to start ADB in TCP/IP mode:

```
adb tcpip 5555
```

- ❑Step 5: Connect to Your Phone Now, connect your phone to your computer wirelessly using the IP address you noted earlier. Replace your_phone_ip with your phone's IP address:

```
adb connect your_phone_ip:5555
```

Running a Flutter App on your phone wirelessly

- **Step 6:** Verify the Connection To verify that your phone is connected wirelessly, run the following command:

```
adb devices
```

- You should see your device listed as connected over the network.
- Step 7: Run Your Flutter App With the wireless connection established, you can now run your Flutter app on your phone as if it were an emulator. Navigate to your Flutter project’s directory and run the app using:

```
flutter run
```

Running a Flutter App on your phone wirelessly

- ❑ Flutter will build and install the app on your connected phone.
- ❑ Step 8: Debug Wirelessly You can now debug your Flutter app wirelessly. Any print statements or error messages will appear in the terminal, just like when running the app on an emulator or a physically connected device.
- ❑ Step 9: Disconnect Wirelessly (Optional) If you want to disconnect your phone from the wireless debugging, you can use the following command:

```
adb disconnect your_phone_ip:5555
```

OR

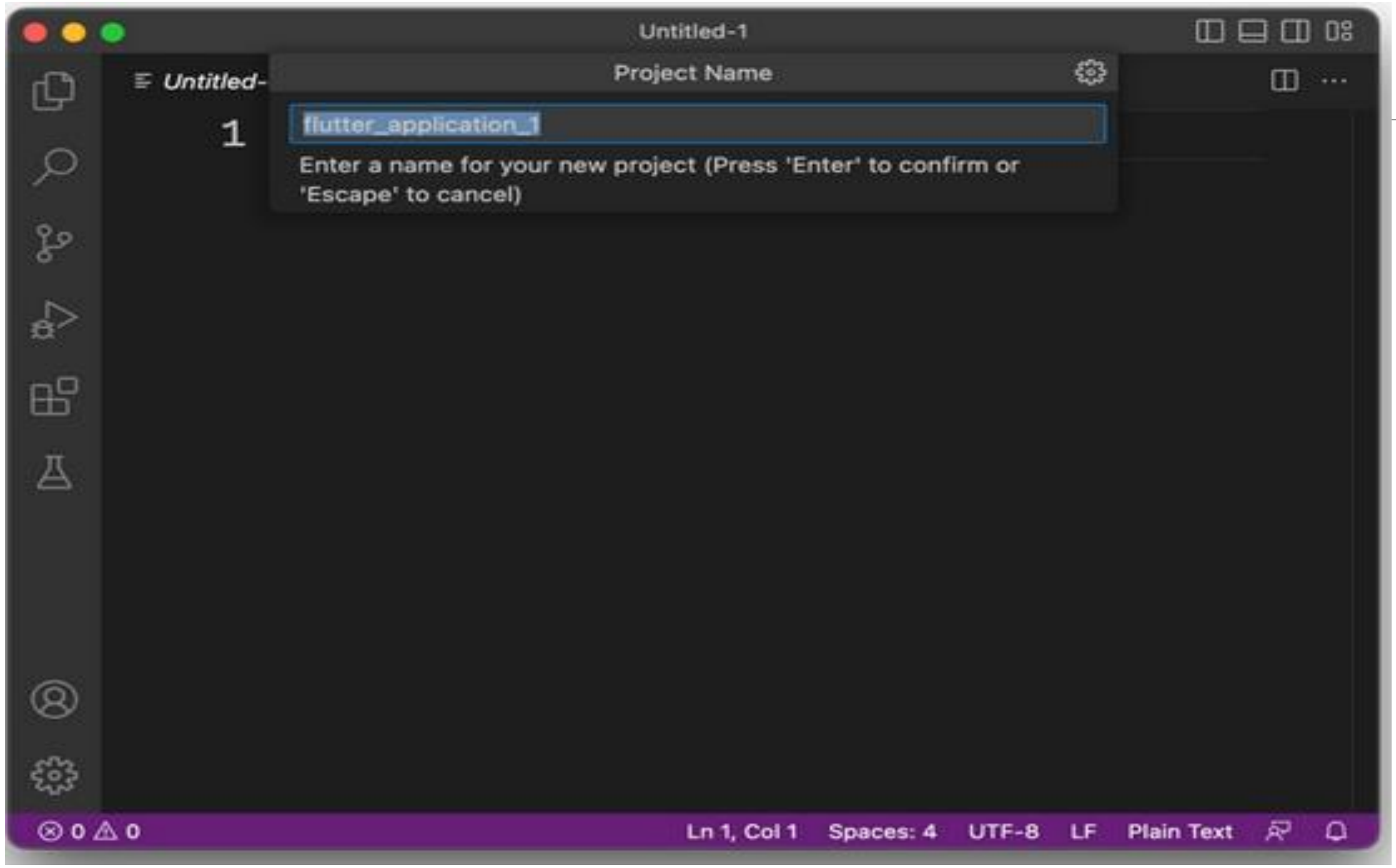
```
adb disconnect
```

Creating first Flutter project

- ❑ Launch Visual Studio Code and open the command palette (with F1 or Ctrl+Shift+P or Shift+Cmd+P). Start typing "flutter new". Select the Flutter: New Project command.
- ❑Next, select Application and then a folder in which to create your project. This could be your home directory, or something like C:\src\.
- ❑Once you’re done with that, it's time to create your first Flutter app by using the command line:

flutter create your_app_name

Creating first Flutter project



Project Name

flutter_application_1

Enter a name for your new project (Press 'Enter' to confirm or 'Escape' to cancel)

❑ Flutter now creates your project folder and VS Code opens it.

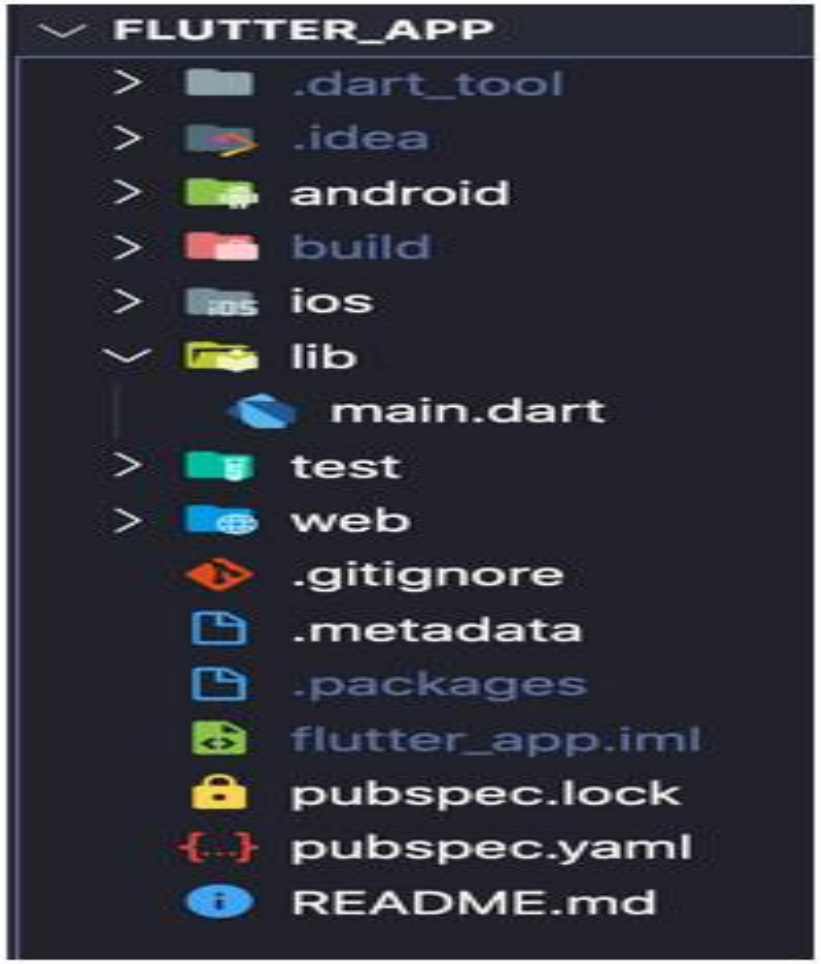
Creating first Flutter project

- The app’s name must include underscores to separate each word.
  - cd your_app_name
- Go inside your folder’s app.
  - flutter run
- Then execute this command and you should see the default Flutter app running on your device or your emulator.

Creating first Flutter project

- Caveat: In this app, you will find that by default, Flutter is using the material design system, but you will not be attached to this. Flutter also provides iOS-styled widgets (Cupertino Widgets) and both of them are highly customizable, so you can even have your own styled app

Project Structure



```
FLUTTER_APP
  > .dart_tool
  > .idea
  > android
  > build
  > ios
  v lib
      main.dart
  > test
  > web
    .gitignore
    .metadata
    .packages
    flutter_app.iml
    pubspec.lock
    pubspec.yaml
    README.md
```

Creating first Flutter project

- ❑ The majority of the files and folders you will find are intended for configuration. Let's take a look at the most important ones:
  - ❑ Android (folder): Contains a complete Android project, and it will be the one in which your Flutter app will be “merged” when compiling to native code. Inside this folder, you will find important files like build.gradle and AndroidManifex.xml.
  - ❑ build (folder): Maintains the output of your Flutter app and will be managed by Flutter’s SDK.
  - ❑ ios (folder): Contains a complete iOS project and will be the one in which your Flutter app will be “merged” when compiling to native code.

Creating first Flutter project

- ❑Lib (folder): The folder where you will probably be working the most. It contains all the .dart files to create the Flutter application and contains the main.dart file, which is by default the entry point of the app.
- ❑ test (folder): Holds all the automated tests that will check the functionality of the code.
- ❑ pubspec.yaml (file): Contains some metadata about the flutter application, but most importantly it will be the palace to manage the project's dependencies, so this is the place to configure the external packages that will be used by your application.

Creating first Flutter project

First lines of code

```
❑ void main() {
  runApp(MyApp);
    }


class MyApp extends StatelessWidget {
  Widget build(BuildContext context)  {
      return MaterialApp(home: Text('Hello!'))
   }
}
```

Creating first Flutter project
First lines of code

- ❑ In the code above, we have one main class, MyApp, that is extending a Flutter class (widget) called StatelessWidget.
- ❑ This class needs to define a build method that must return a widget that at the end will be the element rendered in the screen. For that purpose, we use the MaterialApp class and inside its home property, we make use of another widget, Text, to render a string.
- ❑ Finally, all this code will run inside the main function (the first function to be called when running the application), which in turn will also run another function runApp to mount its given widget as the root widget and attach it to the screen.
- ❑ This is a tiny sample, but as you can see we used many widgets above and it can be cumbersome initially, but that's actually the way Flutter works — the user interface is composed by mixing widgets together.

Thank you