envato

Manyvendor
eCommerce Customer App

Author: SoftTech-IT
Software Framework: Flutter
Application For: Manyvendor eCommerce & Multi-vendor CMS
Provided by: codecanyon

SoftTech-IT
Let's build your career

Manyvendor eCommerce CMS

Customer Flutter App

Documentation

Index

- ## Introduction
    - Welcome to the documentation of Manyvendor e-Commerce Customer Mobile application. The reader should pay a bit more attention while reading this documentation.

- ## Prerequisites
    - For running the application, Admin has to fill some prerequisites. Like:
        - Admin should have Manyvendor eCommerce & Multi-vendor CMS Web Application hosted on a live server.
        - The web application must be in the **latest version always**.
        - eCommerce Mode must be activated in the web application to run this Manyvendor eCommerce Customer Flutter App

- ## Syncing Manyvendor eCommerce Customer Flutter App with Manyvendor eCommerce & Multi-vendor CMS
    - If the admin does have the Manyvendor eCommerce & Multi-vendor CMS web application, he can sync the mobile app now.
    - To run this customer flutter app-admin must be activated eCommerce in Manyvendor eCommerce & Multi-vendor CMS web application.
    - Download the flutter app from code canyon.

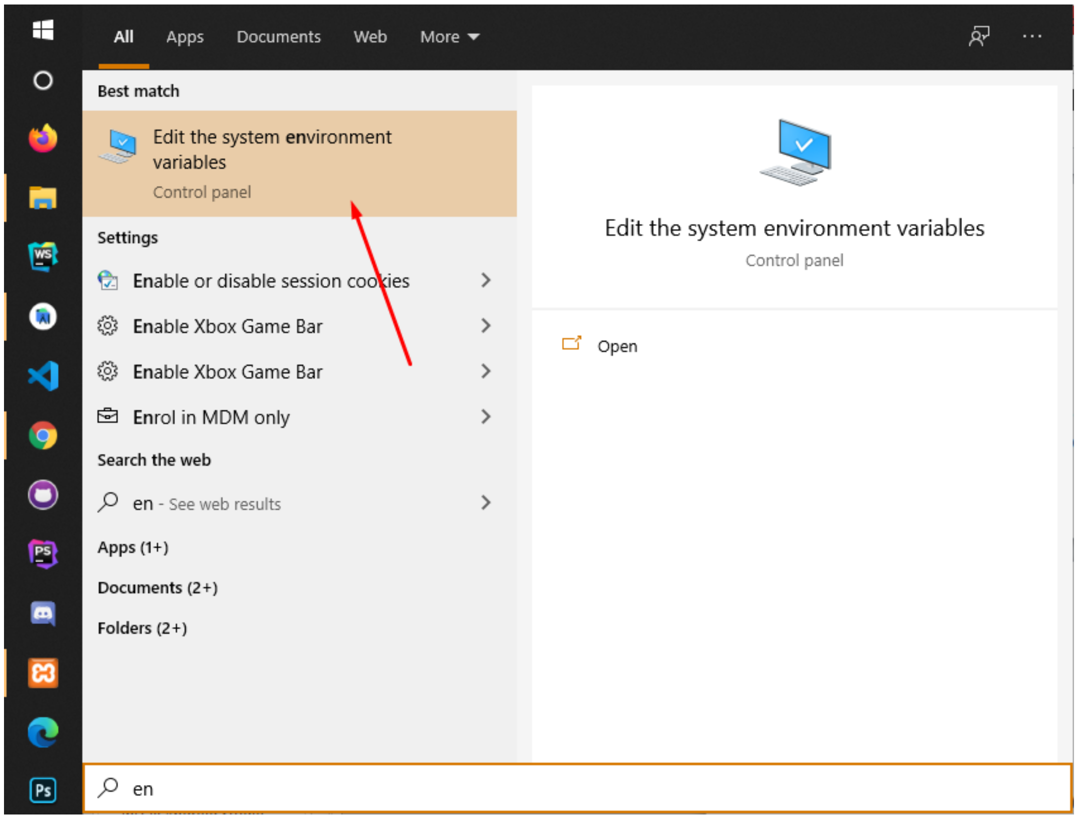
Install flutter following all instructions on your platform from this link - https://flutter.dev/docs/get-started/install

Or Follow this for windows,

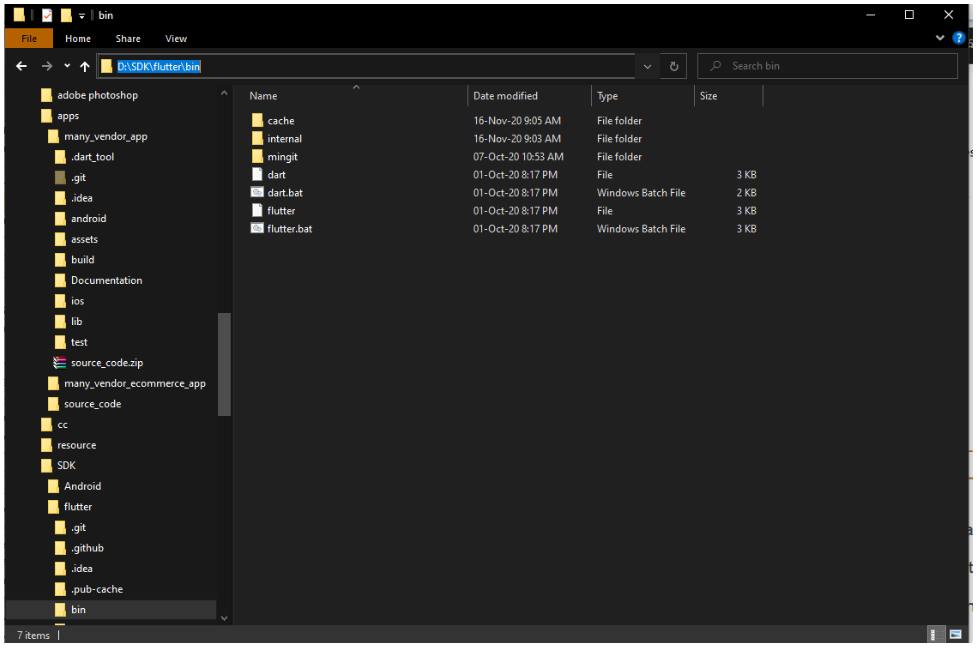
- To install and run Flutter, your development environment must meet these minimum requirements:
- **Operating Systems**: Windows 7 SP1 or later (64-bit), x86-64 based
- **Disk Space**: 1.32 GB (does not include disk space for IDE/tools).
- **Tools**: Flutter depends on these tools being available in your environment.
    - Windows PowerShell 5.0 or newer (this is pre-installed with Windows 10)
    - Git for Windows 2.x, with the **Use Git from the Windows Command Prompt** option.
        - If Git for Windows is already installed, make sure you can run git commands from the command prompt or PowerShell.
        - Download the following installation bundle to get the latest stable release of the Flutter SDK this link:

Update your path

- If you wish to run Flutter commands in the regular Windows console(CMD), take these steps to add Flutter to the PATH environment variable:
- From the Start search bar, enter ‘env’ and select **Edit environment variables for your account**.

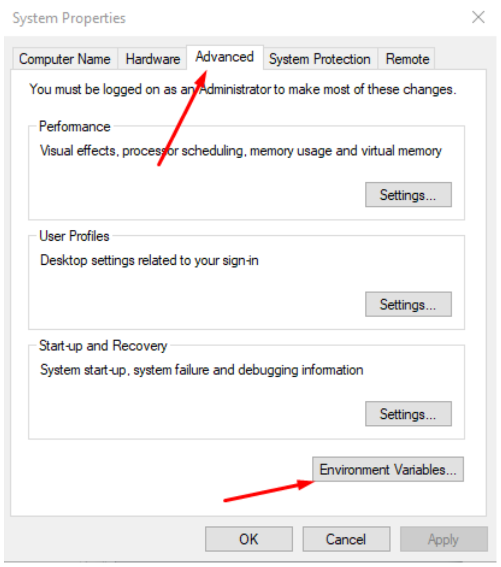


- Under **User variables** check if there is an entry called **Path**:
    - If the entry exists, append the full path to flutter\bin using; as a separator from existing values.



Name	Date modified	Type	Size
cache	16-Nov-20 9:05 AM	File folder	
internal	16-Nov-20 9:03 AM	File folder	
mingit	07-Oct-20 10:53 AM	File folder	
dart	01-Oct-20 8:17 PM	File	3 KB
dart.bat	01-Oct-20 8:17 PM	Windows Batch File	2 KB
flutter	01-Oct-20 8:17 PM	File	3 KB
flutter.bat	01-Oct-20 8:17 PM	Windows Batch File	3 KB

    - If the entry doesn't exist, create a new user variable named Path with the full path to flutter\bin as its value.
- You have to close and reopen any existing console windows for these changes to take effect.
- Follow this image

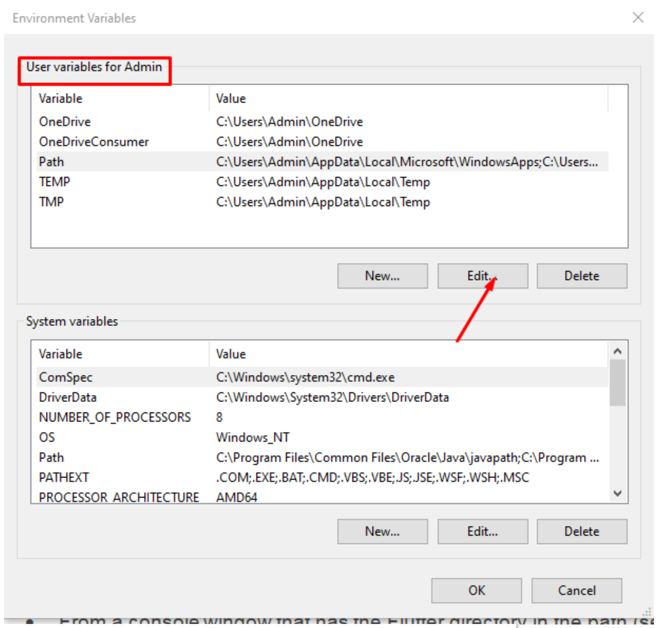


You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage and virtual memory

User Profiles
Desktop settings related to your sign-in

Start-up and Recovery
System start-up, system failure and debugging information

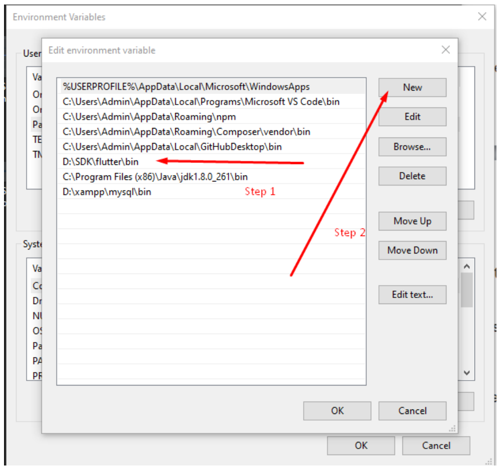


User variables for Admin

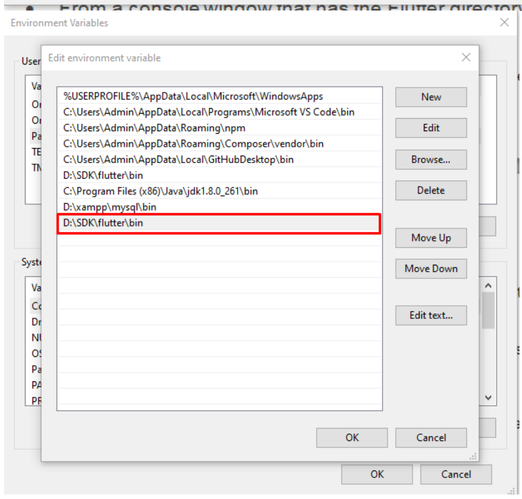
Variable	Value
OneDrive	C:\Users\Admin\OneDrive
OneDriveConsumer	C:\Users\Admin\OneDrive
Path	C:\Users\Admin\AppData\Local\Microsoft\WindowsApps;C:\Users...
TEMP	C:\Users\Admin\AppData\Local\Temp
TMP	C:\Users\Admin\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\Program Files\Common Files\Oracle\Java\javapath;C:\Program ...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64



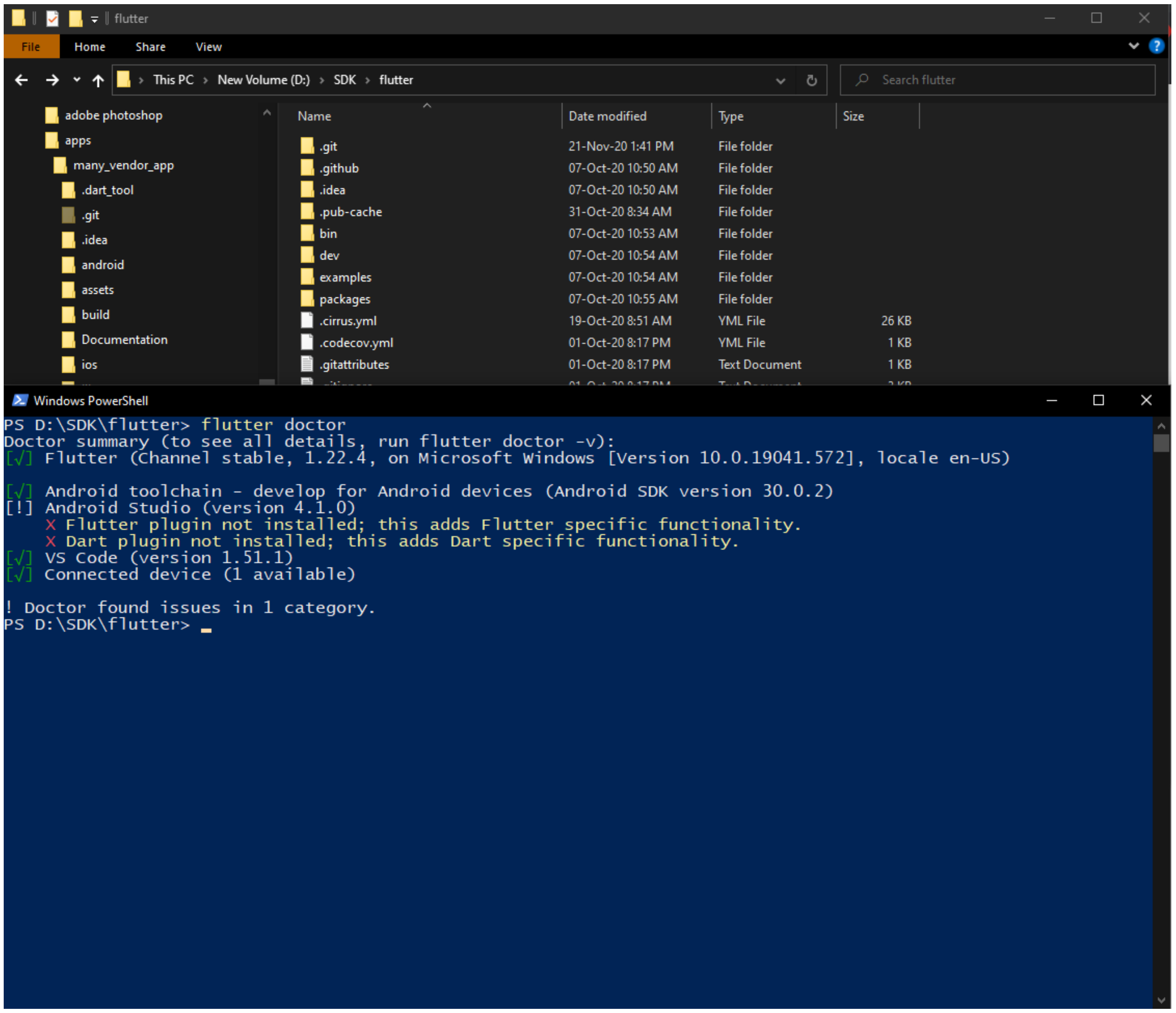
```
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\Admin\AppData\Local\Programs\Microsoft VS Code\bin
C:\Users\Admin\AppData\Roaming\npm
C:\Users\Admin\AppData\Roaming\Composer\vendor\bin
C:\Users\Admin\AppData\Local\GitHubDesktop\bin
D:\SDK\flutter\bin
C:\Program Files (x86)\Java\jdk1.8.0_261\bin
D:\xampp\mysql\bin
```



```
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\Admin\AppData\Local\Programs\Microsoft VS Code\bin
C:\Users\Admin\AppData\Roaming\npm
C:\Users\Admin\AppData\Roaming\Composer\vendor\bin
C:\Users\Admin\AppData\Local\GitHubDesktop\bin
D:\SDK\flutter\bin
C:\Program Files (x86)\Java\jdk1.8.0_261\bin
D:\xampp\mysql\bin
D:\SDK\flutter\bin
```

Run flutter doctor

- From a console window(CMD) that has the Flutter directory in the path (see above), run the following command to see if there are any platform dependencies you need to complete the setup:



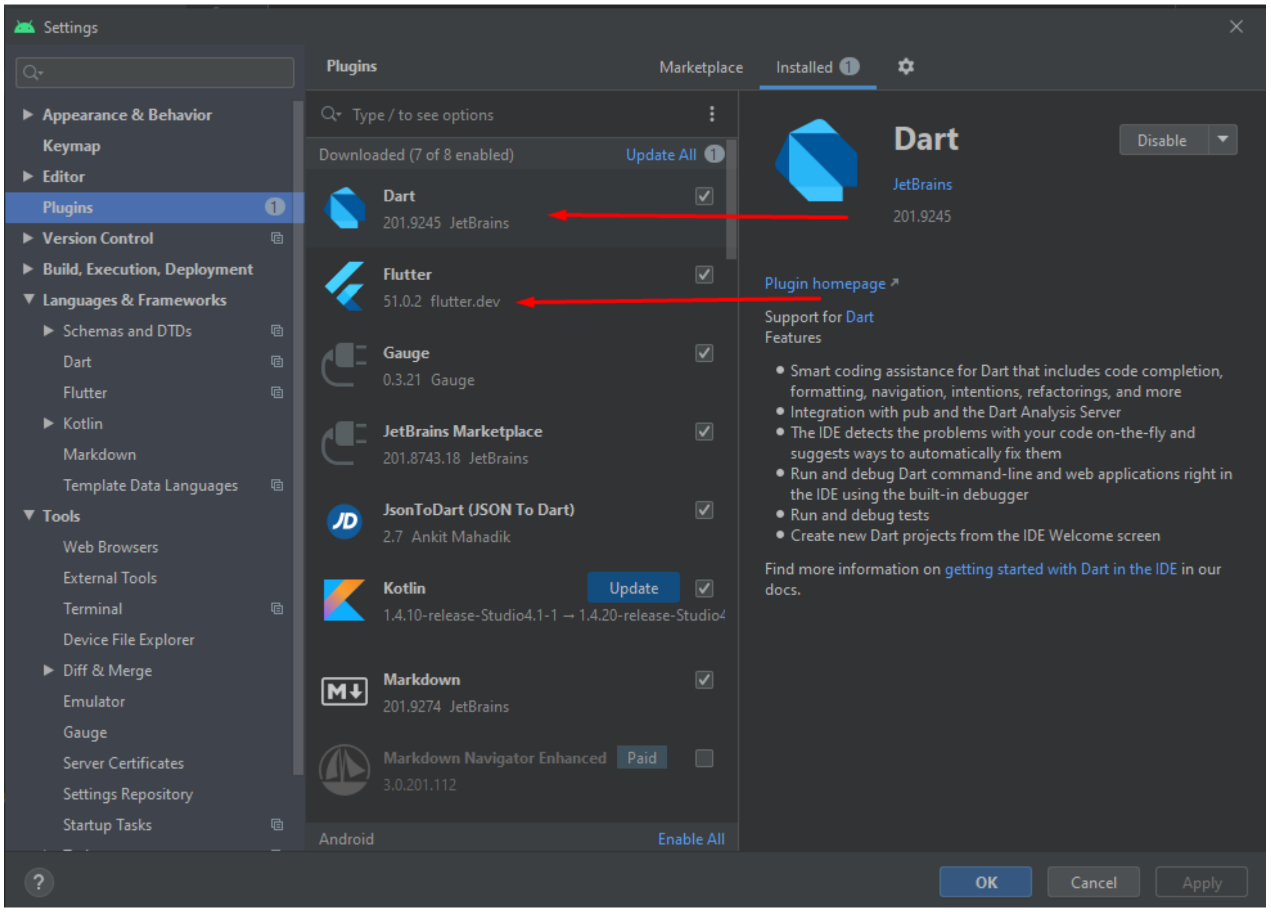
Name	Date modified	Type	Size
.git	21-Nov-20 1:41 PM	File folder	
.github	07-Oct-20 10:50 AM	File folder	
.idea	07-Oct-20 10:50 AM	File folder	
.pub-cache	31-Oct-20 8:34 AM	File folder	
bin	07-Oct-20 10:53 AM	File folder	
dev	07-Oct-20 10:54 AM	File folder	
examples	07-Oct-20 10:54 AM	File folder	
packages	07-Oct-20 10:55 AM	File folder	
.cirrus.yml	19-Oct-20 8:51 AM	YML File	26 KB
.codecov.yml	01-Oct-20 8:17 PM	YML File	1 KB
.gitattributes	01-Oct-20 8:17 PM	Text Document	1 KB

```
PS D:\SDK\flutter> flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 1.22.4, on Microsoft Windows [Version 10.0.19041.572], locale en-US)

[√] Android toolchain - develop for Android devices (Android SDK version 30.0.2)
[!] Android Studio (version 4.1.0)
    X Flutter plugin not installed; this adds Flutter specific functionality.
    X Dart plugin not installed; this adds Dart specific functionality.
[√] VS Code (version 1.51.1)
[√] Connected device (1 available)

! Doctor found issues in 1 category.
PS D:\SDK\flutter> _
```

- C:\src\flutter>flutter doctor
- This command checks your environment and displays a report of the status of your Flutter installation. Check the output carefully for other software you might need to install or further tasks to perform (shown in **bold** text).

Install Android Studio

- Download and install Android Studio.
- Start Android Studio, and go through the 'Android Studio Setup Wizard'. This installs the latest Android SDK, Android SDK Command-line Tools, and Android SDK Build-Tools, which are required by Flutter when developing for Android.
- After successfully install android studio you mast me install **Dart and Flutter** plugins in android studio belong this image



Downloaded (7 of 8 enabled)

Dart
201.9245 JetBrains

Flutter
51.0.2 flutter.dev

Gauge
0.3.21 Gauge

JetBrains Marketplace
201.8743.18 JetBrains

JsonToDart (JSON To Dart)
2.7 Ankit Mahadik

Kotlin
1.4.10-release-Studio4.1-1 → 1.4.20-release-Studio4

Markdown
201.9274 JetBrains

Markdown Navigator Enhanced
3.0.201.112

Dart
JetBrains
201.9245

Plugin homepage ↗

Support for Dart
Features

- Smart coding assistance for Dart that includes code completion, formatting, navigation, intentions, refactorings, and more
- Integration with pub and the Dart Analysis Server
- The IDE detects the problems with your code on-the-fly and suggests ways to automatically fix them
- Run and debug Dart command-line and web applications right in the IDE using the built-in debugger
- Run and debug tests
- Create new Dart projects from the IDE Welcome screen

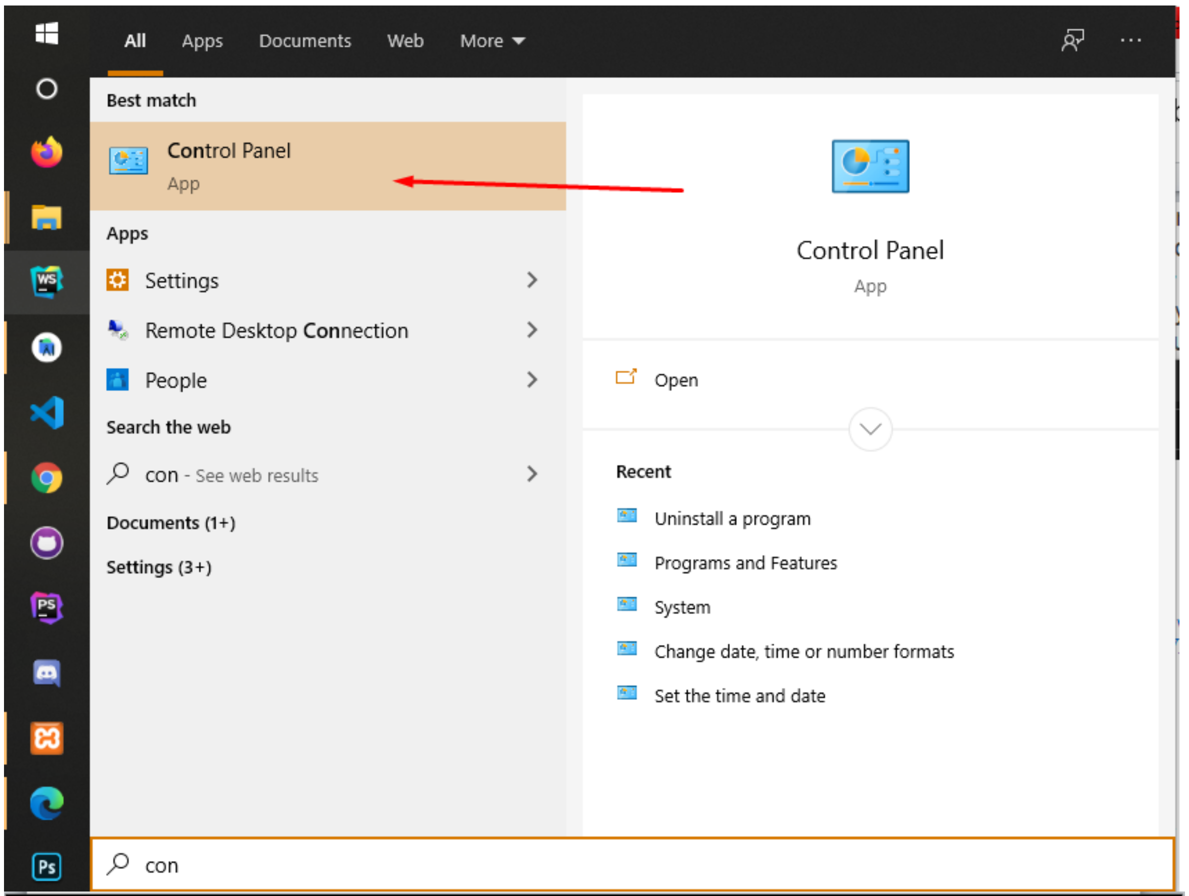
Find more information on getting started with Dart in the IDE in our docs.

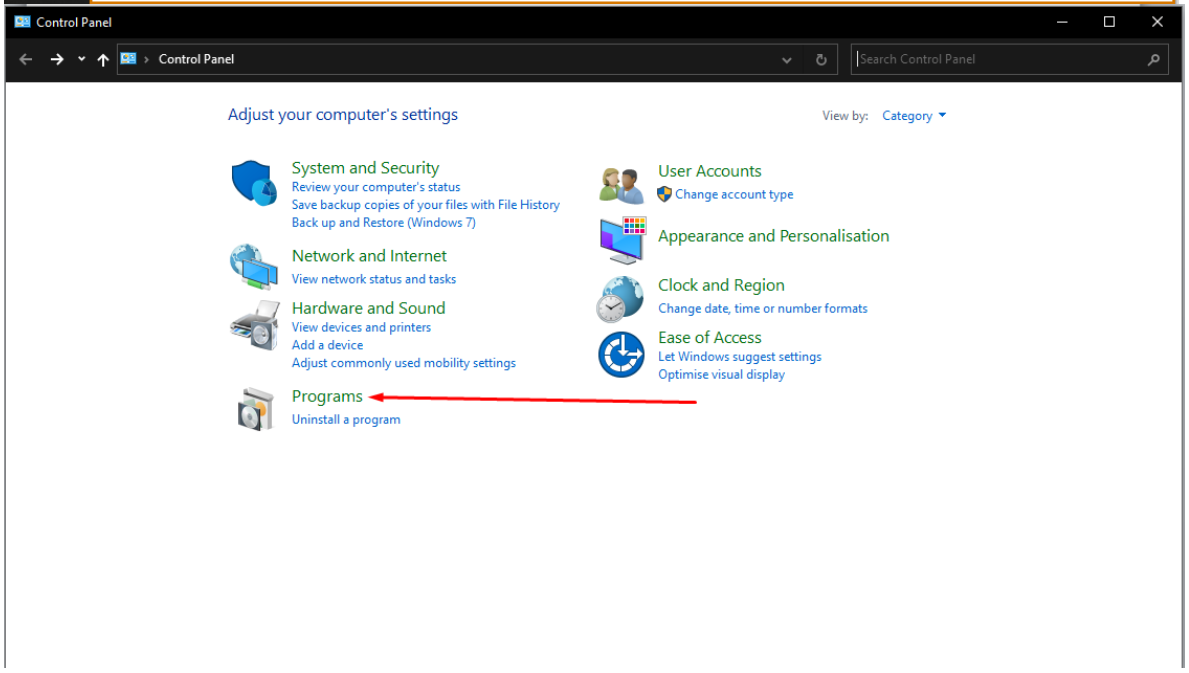
Set up your Android device

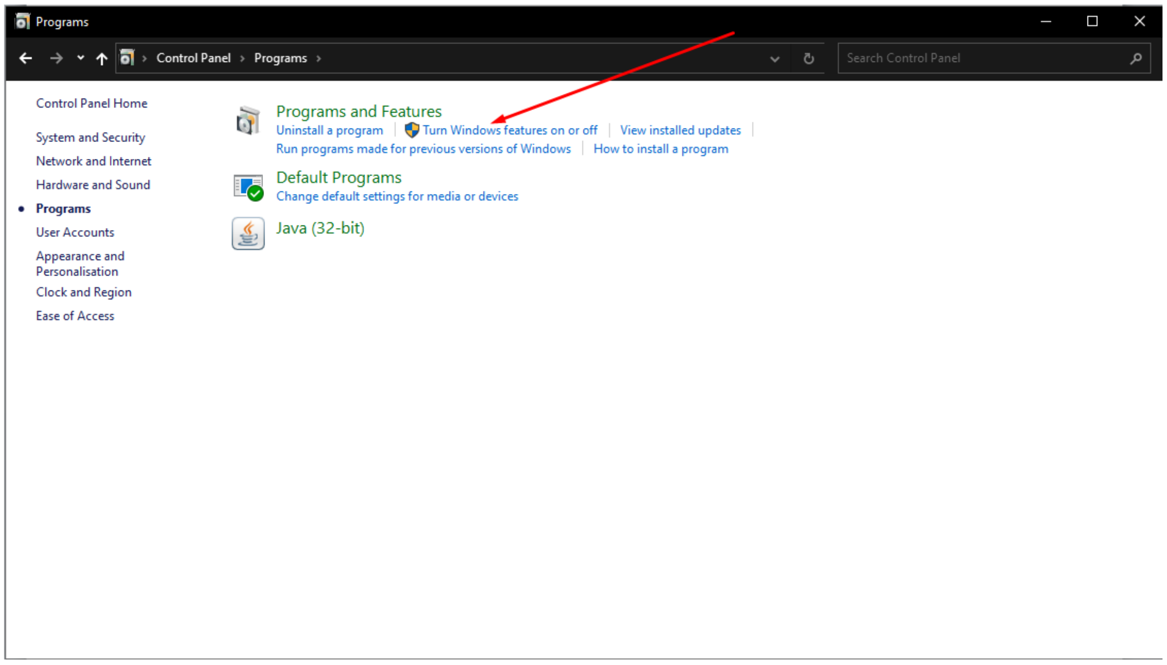
- To prepare to run and test your Flutter app on an Android device, you need an Android device running Android 4.1 (API level 16) or higher.
- Enable **Developer Options** and **USB debugging** on your device. Detailed instructions are available in the Android documentation.
- Windows-only: Install the Google USB Driver.
- Using a USB cable, plug your phone into your computer. If prompted on your device, authorize your computer to access your device.
- In the terminal, run the flutter devices command to verify that Flutter recognizes your connected Android device. By default, Flutter uses the version of the Android SDK where your ADB tool is based. If you want Flutter to use a different installation of the Android SDK, you must set the ANDROID_SDK_ROOT environment variable to that installation directory.

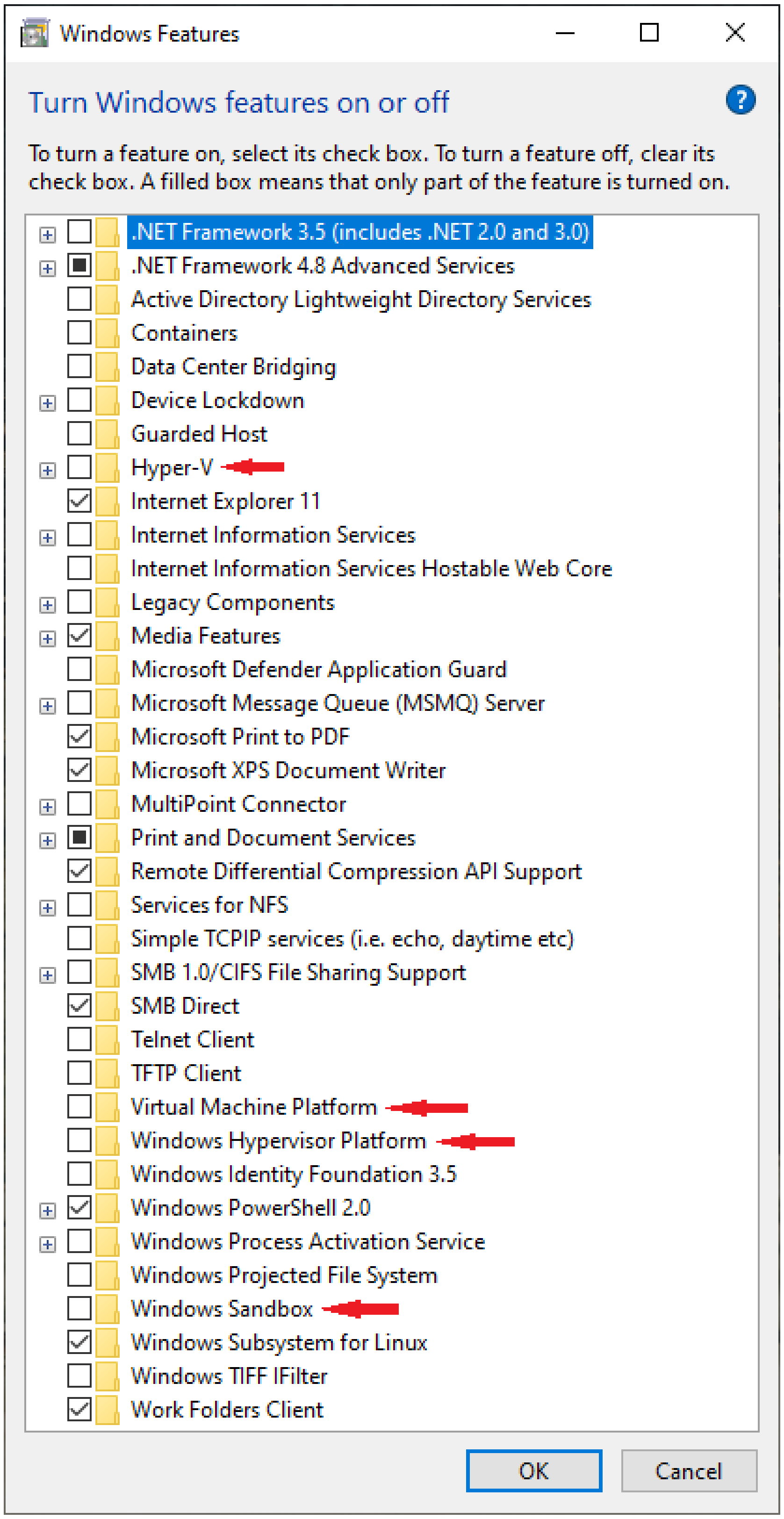
Set up the Android emulator

- To prepare to run and test your Flutter app on the Android emulator, follow these steps:
- Enable VM acceleration on your machine.
- Follow image like this


All
Apps
Documents
Web
More
Best match
Control Panel
App
Apps
Settings
Remote Desktop Connection
People
Search the web
con - See web results
Documents (1+)
Settings (3+)
Control Panel
App
Open
Recent
Uninstall a program
Programs and Features
System
Change date, time or number formats
Set the time and date
con


Control Panel
Control Panel
Search Control Panel
Adjust your computer's settings
View by: Category
System and Security
Review your computer's status
Save backup copies of your files with File History
Back up and Restore (Windows 7)
Network and Internet
View network status and tasks
Hardware and Sound
View devices and printers
Add a device
Adjust commonly used mobility settings
Programs
Uninstall a program
User Accounts
Change account type
Appearance and Personalisation
Clock and Region
Change date, time or number formats
Ease of Access
Let Windows suggest settings
Optimise visual display


Programs
Control Panel > Programs >
Search Control Panel
Control Panel Home
System and Security
Network and Internet
Hardware and Sound
Programs
User Accounts
Appearance and Personalisation
Clock and Region
Ease of Access
Programs and Features
Uninstall a program
Turn Windows features on or off
View installed updates
Run programs made for previous versions of Windows
How to install a program
Default Programs
Change default settings for media or devices
Java (32-bit)



Windows Features

Turn Windows features on or off

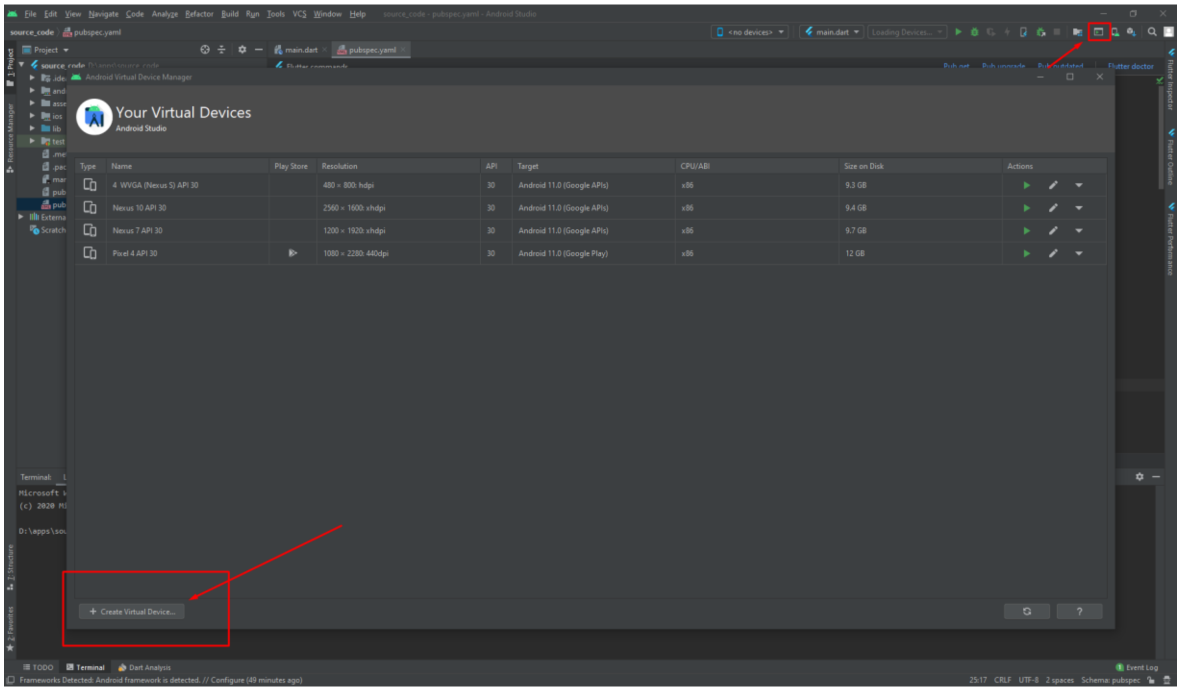
To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- [ ] .NET Framework 3.5 (includes .NET 2.0 and 3.0)
- [■] .NET Framework 4.8 Advanced Services
- [ ] Active Directory Lightweight Directory Services
- [ ] Containers
- [ ] Data Center Bridging
- [ ] Device Lockdown
- [ ] Guarded Host
- [ ] Hyper-V
- [✓] Internet Explorer 11
- [ ] Internet Information Services
- [ ] Internet Information Services Hostable Web Core
- [ ] Legacy Components
- [✓] Media Features
- [ ] Microsoft Defender Application Guard
- [ ] Microsoft Message Queue (MSMQ) Server
- [✓] Microsoft Print to PDF
- [✓] Microsoft XPS Document Writer
- [ ] MultiPoint Connector
- [■] Print and Document Services
- [✓] Remote Differential Compression API Support
- [ ] Services for NFS
- [ ] Simple TCPIP services (i.e. echo, daytime etc)
- [ ] SMB 1.0/CIFS File Sharing Support
- [✓] SMB Direct
- [ ] Telnet Client
- [ ] TFTP Client
- [ ] Virtual Machine Platform
- [ ] Windows Hypervisor Platform
- [ ] Windows Identity Foundation 3.5
- [✓] Windows PowerShell 2.0
- [ ] Windows Process Activation Service
- [ ] Windows Projected File System
- [ ] Windows Sandbox
- [✓] Windows Subsystem for Linux
- [ ] Windows TIFF IFilter
- [✓] Work Folders Client

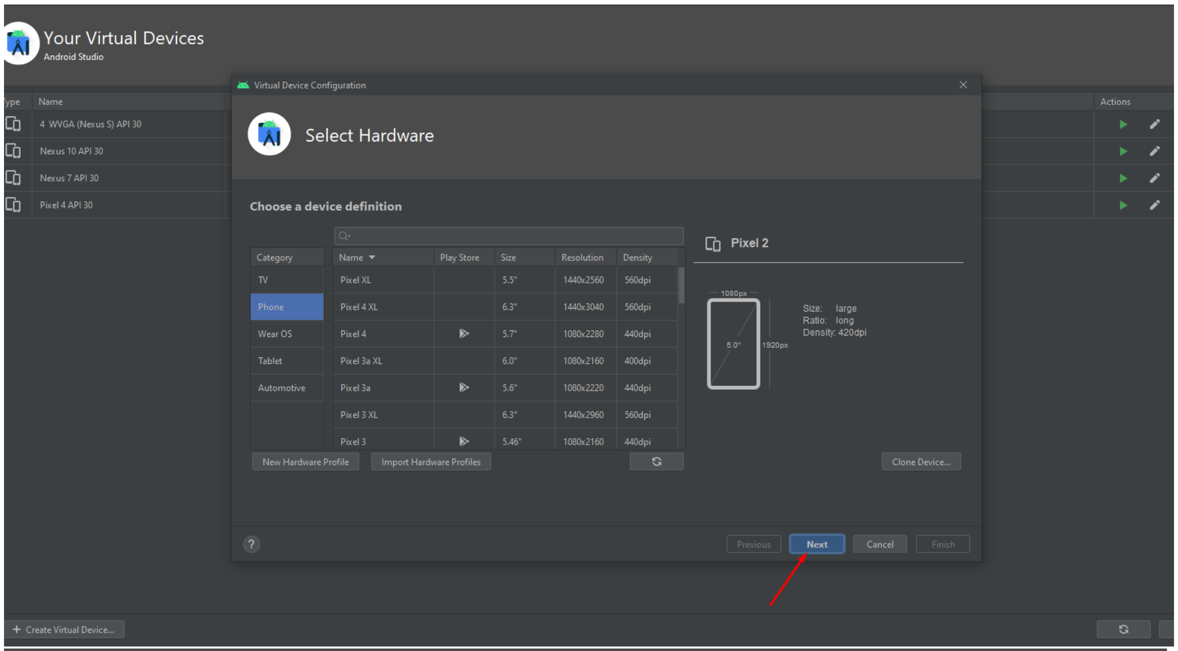
- Launch **Android Studio**, click the **AVD Manager** icon, and select **Create Virtual Device…**

    - In older versions of Android Studio, you should instead launch **Android Studio > Tools > Android > AVD Manager** and select **Create Virtual Device…**. (The **Android** submenu is only present when inside an Android project.)
    - If you do not have a project open, you can choose **Configure > AVD Manager** and select **Create Virtual Device…**
- Choose a device definition and select **Next**.
- Select one or more system images for the Android versions you want to emulate, and select **Next**. An *x86* or *x86_64* image is recommended.
- Under Emulated Performance, select **Hardware - GLES 2.0** to enable [hardware acceleration]().
- Verify the AVD configuration is correct, and select **Finish**.
    - For details on the above steps, see [Managing AVDs]().
- In Android Virtual Device Manager, click **Run** in the toolbar. The emulator starts up and displays the default canvas for your selected OS version and device.

Follow this images



Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	4 WVGA (Nexus S) API 30		480 × 800: hdpi	30	Android 11.0 (Google APIs)	x86	9.3 GB	
	Nexus 10 API 30		2560 × 1600: xhdpi	30	Android 11.0 (Google APIs)	x86	9.4 GB	
	Nexus 7 API 30		1200 × 1920: xhdpi	30	Android 11.0 (Google APIs)	x86	9.7 GB	
	Pixel 4 API 30	▶	1080 × 2280: 440dpi	30	Android 11.0 (Google Play)	x86	12 GB	



Your Virtual Devices
Android Studio

Type	Name	Actions
	4 WVGA (Nexus S) API 30	
	Nexus 10 API 30	
	Nexus 7 API 30	
	Pixel 4 API 30	

Virtual Device Configuration

Select Hardware

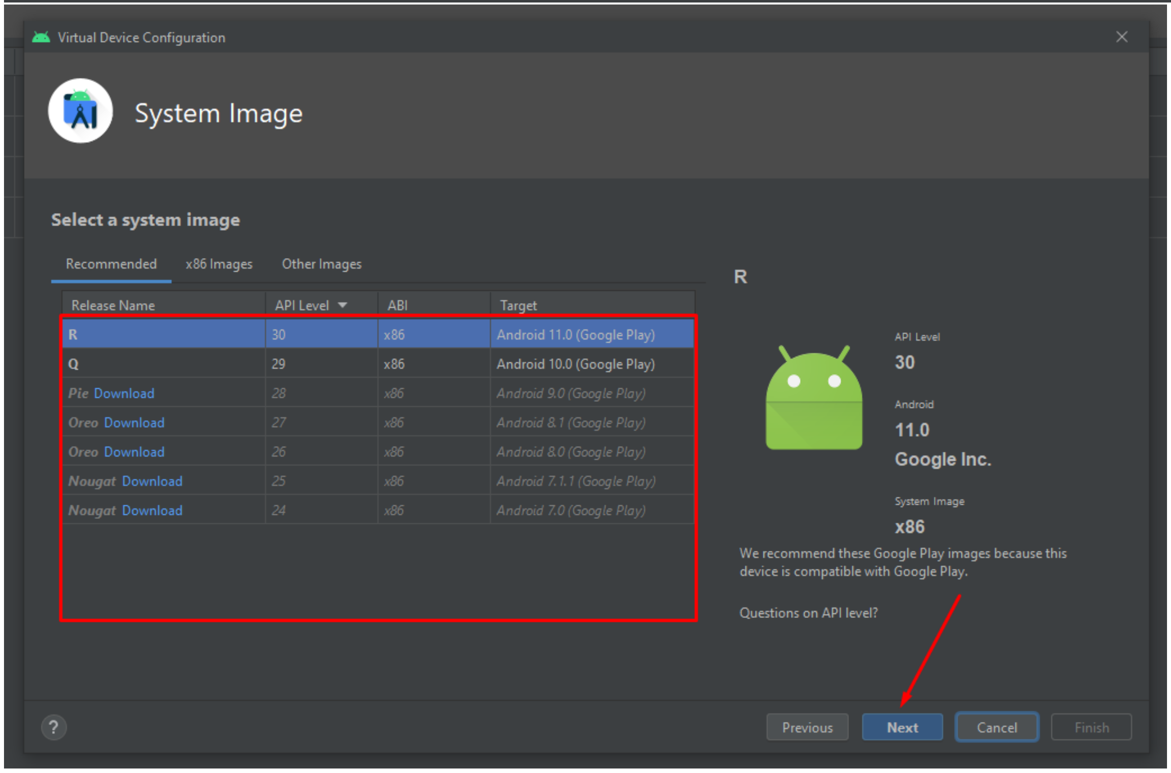
Choose a device definition

Category
TV
Phone
Wear OS
Tablet
Automotive

Name	Play Store	Size	Resolution	Density
Pixel XL		5.5"	1440x2560	560dpi
Pixel 4 XL		6.3"	1440x3040	560dpi
Pixel 4	▶	5.7"	1080x2280	440dpi
Pixel 3a XL		6.0"	1080x2160	400dpi
Pixel 3a	▶	5.6"	1080x2220	440dpi
Pixel 3 XL		6.3"	1440x2960	560dpi
Pixel 3	▶	5.46"	1080x2160	440dpi

Pixel 2

Size: large
Ratio: long
Density: 420dpi



Virtual Device Configuration

System Image

Select a system image

Release Name	API Level	ABI	Target
R	30	x86	Android 11.0 (Google Play)
Q	29	x86	Android 10.0 (Google Play)
Pie Download	28	x86	Android 9.0 (Google Play)
Oreo Download	27	x86	Android 8.1 (Google Play)
Oreo Download	26	x86	Android 8.0 (Google Play)
Nougat Download	25	x86	Android 7.1.1 (Google Play)
Nougat Download	24	x86	Android 7.0 (Google Play)

R

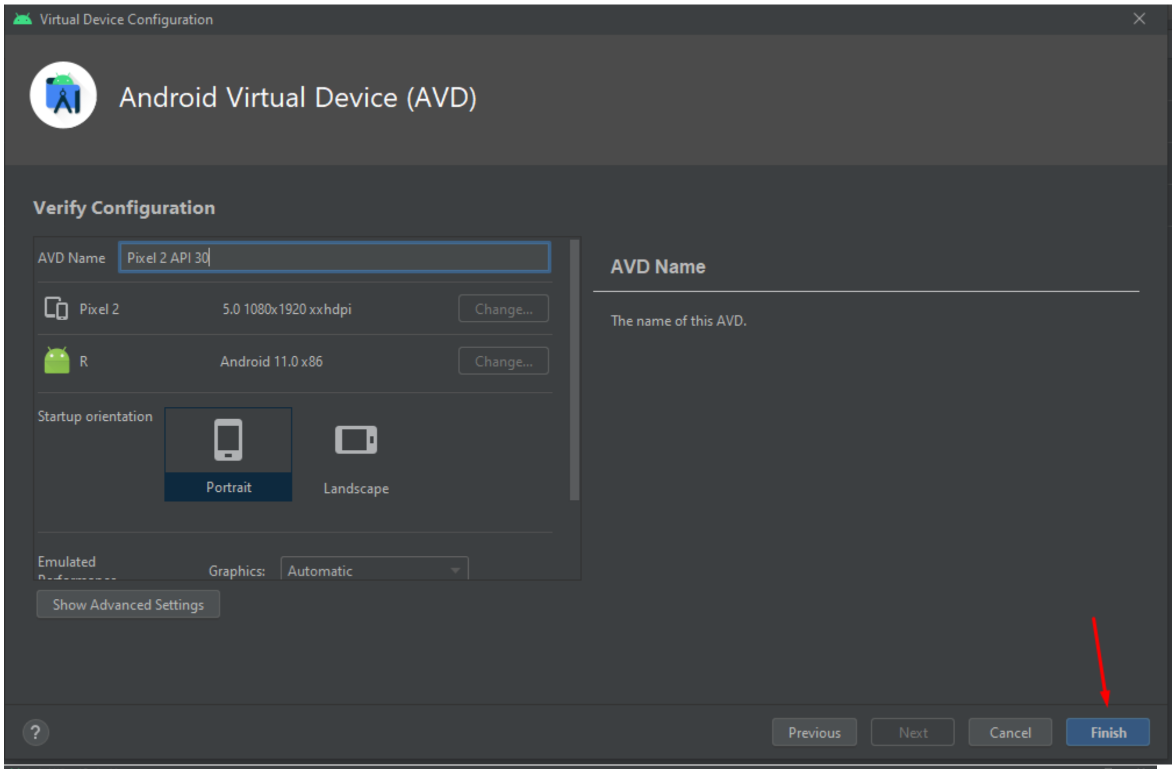
API Level
30

Android
11.0
Google Inc.

System Image
x86

We recommend these Google Play images because this device is compatible with Google Play.

Questions on API level?



Virtual Device Configuration

Android Virtual Device (AVD)

Verify Configuration

AVD Name: Pixel 2 API 30

Pixel 2	5.0 1080x1920 xxhdpi	Change...
R	Android 11.0 x86	Change...

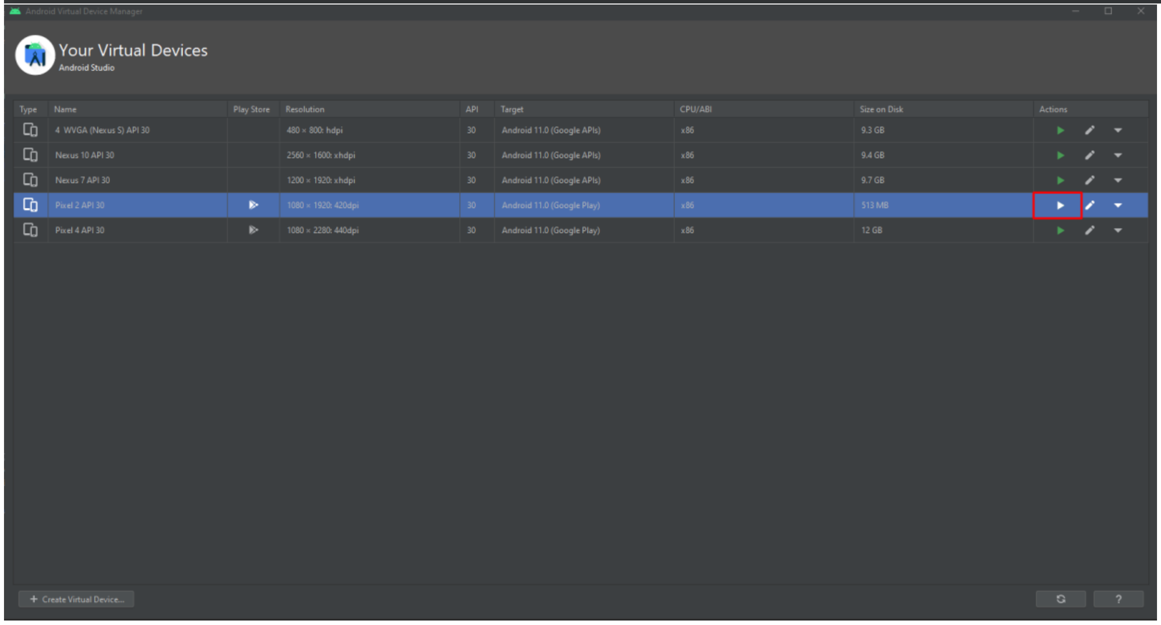
Startup orientation: [✓ Portrait] Landscape

Emulated Performance — Graphics: Automatic

Show Advanced Settings

AVD Name

The name of this AVD.

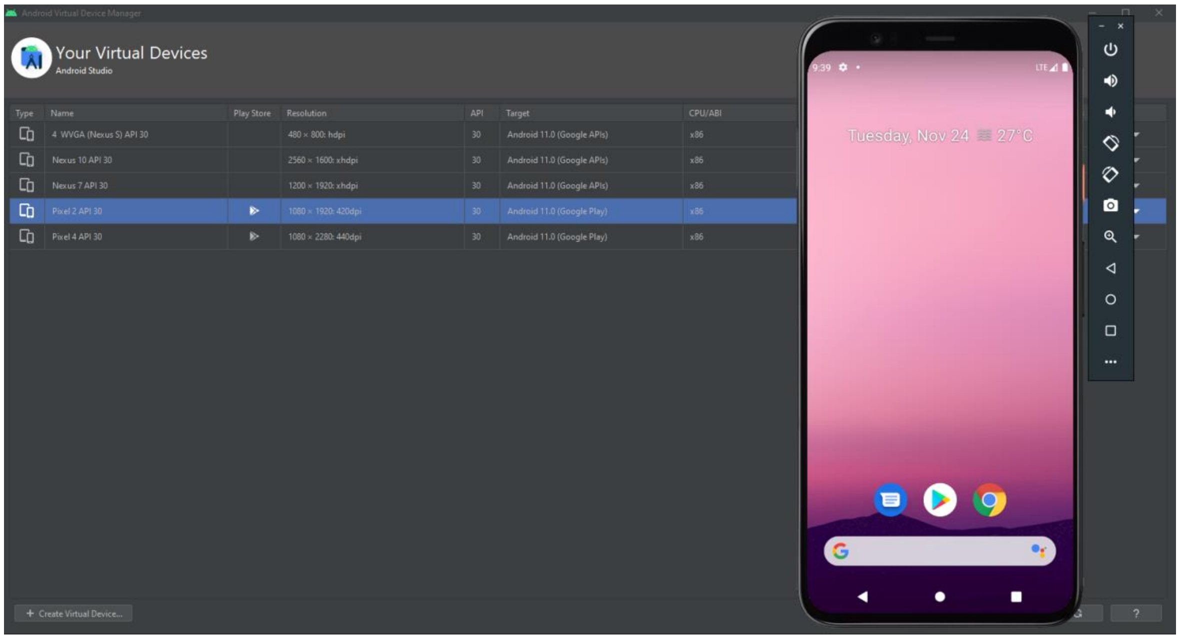


Android Virtual Device Manager

Your Virtual Devices

Android Studio

Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	4 WVGA (Nexus S) API 30		480 × 800: hdpi	30	Android 11.0 (Google APIs)	x86	9.3 GB	
	Nexus 10 API 30		2560 × 1600: xhdpi	30	Android 11.0 (Google APIs)	x86	9.4 GB	
	Nexus 7 API 30		1200 × 1920: xhdpi	30	Android 11.0 (Google APIs)	x86	9.7 GB	
	Pixel 2 API 30	▶	1080 × 1920: 420dpi	30	Android 11.0 (Google Play)	x86	513 MB	
	Pixel 4 API 30	▶	1080 × 2280: 440dpi	30	Android 11.0 (Google Play)	x86	12 GB	



Type	Name	Play Store	Resolution	API	Target	CPU/ABI
	4 WVGA (Nexus S) API 30		480 × 800: hdpi	30	Android 11.0 (Google APIs)	x86
	Nexus 10 API 30		2560 × 1600: xhdpi	30	Android 11.0 (Google APIs)	x86
	Nexus 7 API 30		1200 × 1920: xhdpi	30	Android 11.0 (Google APIs)	x86
	Pixel 2 API 30	▶	1080 × 1920: 420dpi	30	Android 11.0 (Google Play)	x86
	Pixel 4 API 30	▶	1080 × 2280: 440dpi	30	Android 11.0 (Google Play)	x86

- Or Follow this macOS

System requirements

- To install and run Flutter, your development environment must meet these minimum requirements:
- **Operating Systems**: macOS (64-bit)
- **Disk Space**: 2.8 GB (does not include disk space for IDE/tools).
- **Tools**: Flutter depends on these command-line tools being available in your environment.
  - bash
  - curl
  - git 2.x
  - mkdir
  - rm
  - unzip
  - which

Get the Flutter SDK

- Download the following installation bundle to get the latest stable release of the Flutter SDK link:
- For other release channels, and older builds, see the SDK releases page.

```
$ cd ~/development
$ unzip ~/Downloads/flutter_macos_1.22.4-stable.zip
```

  - If you don't want to install a fixed version of the installation bundle, you can skip steps 1 and 2. Instead, get the source code from the Flutter repo on GitHub with the following command:

```
$ git clone https://github.com/flutter/flutter.git
```

  - You can also change branches or tags as needed. For example, to get just the stable version:

```
$ git clone https://github.com/flutter/flutter.git -b stable --depth 1
```

- Add the flutter tool to your path:

  ```
  $ export PATH="$PATH:`pwd`/flutter/bin"
  ```

  - This command sets your PATH variable for the *current* terminal window only. To permanently add Flutter to your path, see [Update your path]().
- Optionally, pre-download development binaries:
  - The flutter tool downloads platform-specific development binaries as needed. For scenarios where pre-downloading these artifacts is preferable (for example, in hermetic build environments, or with intermittent network availability), iOS and Android binaries can be downloaded ahead of time by running:

  ```
  $ flutter precache
  ```

  - For additional download options, see flutter help precache.
- You are now ready to run Flutter commands!

Run flutter doctor

- Run the following command to see if there are any dependencies you need to install to complete the setup (for verbose output, add the `-v` flag):

```
$ flutter doctor
```

Update your path

- You can update your PATH variable for the current session at the command line, as shown in [Get the Flutter SDK](). You'll probably want to update this variable permanently, so you can run `flutter` commands in any terminal session.
- The steps for modifying this variable permanently for all terminal sessions are machine-specific. Typically you add a line to a file that is executed whenever you open a new window. For example:
- Determine the directory where you placed the Flutter SDK. You need this in Step 3.
- Open (or create) the `rc` file for your shell. Typing `echo $SHELL` in your Terminal tells you which shell you're using. If you're using Bash, edit `$HOME/.bash_profile` or `$HOME/.bashrc`. If you're using Z shell, edit `$HOME/.zshrc`. If you're using a different shell, the file path and filename will be different on your machine.
- Add the following line and change `[PATH_TO_FLUTTER_GIT_DIRECTORY]` to be the path where you cloned Flutter's git repo:

  ```
  $ export PATH="$PATH:[PATH_TO_FLUTTER_GIT_DIRECTORY]/flutter/bin"
  ```

- Run `source $HOME/.<rc file>` to refresh the current window, or open a new terminal window to automatically source the file.
- Verify that the `flutter/bin` directory is now in your PATH by running:

  ```
  $ echo $PATH
  ```

  - Verify that the `flutter` command is available by running:

  ```
  $ which flutter
  ```

Platform setup

- macOS supports developing Flutter apps in iOS, Android, and the web (technical preview release). Complete at least one of the platform setup steps now, to be able to build and run your first Flutter app.

iOS setup Install Xcode

- To develop Flutter apps for iOS, you need a Mac with Xcode installed.
- Install the latest stable version of Xcode (using web download or the Mac App Store).
- Configure the Xcode command-line tools to use the newly-installed version of Xcode by running the following from the command line:

```
$ sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer
$ sudo xcodebuild -runFirstLaunch
```

    - This is the correct path for most cases, when you want to use the latest version of Xcode. If you need to use a different version, specify that path instead.
- Make sure the Xcode license agreement is signed by either opening Xcode once and confirming or running `sudo xcodebuild -license` from the command line.
- Versions older than the latest stable version may still work, but are not recommended for Flutter development. Using old versions of Xcode to target bitcode is not supported, and is likely not to work.
- With Xcode, you'll be able to run Flutter apps on an iOS device or on the simulator.

Set up the iOS simulator

- To prepare to run and test your Flutter app on the iOS simulator, follow these steps:
- On your Mac, find the Simulator via Spotlight or by using the following command:

```
$ open -a Simulator
```

- Make sure your simulator is using a 64-bit device (iPhone 5s or later) by checking the settings in the simulator's **Hardware > Device** menu.
- Depending on your development machine's screen size, simulated high-screen-density iOS devices might overflow your screen. Grab the corner of the simulator and drag it to change the scale. You can also use the **Window > Physical Size** or **Window > Pixel Accurate** options if your computer's resolution is high enough.
    - If you are using a version of Xcode older than 9.1, you should instead set the device scale in the **Window > Scale** menu.

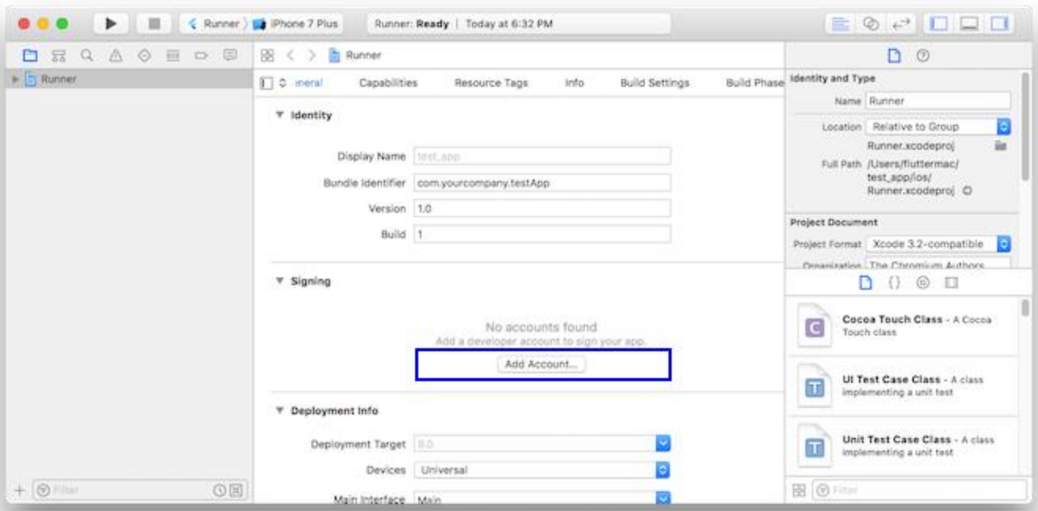
Deploy to iOS devices

- To deploy your Flutter app to a physical iOS device you'll need to set up physical device deployment in Xcode and an Apple Developer account. If your app is using Flutter plugins, you will also need the third-party CocoaPods dependency manager.
- You can skip this step if your apps do not depend on Flutter plugins with native iOS code. Install and set up CocoaPods by running the following commands:

```
$ sudo gem install cocoapods
```

    - **Note:** The default version of Ruby requires `sudo` to install the CocoaPods gem. If you are using a Ruby Version manager, you may need to run without `sudo`.
- **Follow the Xcode signing flow to provision your project:**
    - Open the default Xcode workspace in your project by running `open ios/Runner.xcworkspace` in a terminal window from your Flutter project directory.

- Select the device you intend to deploy to in the device drop-down menu next to the run button.
- Select the Runner project in the left navigation panel.
- In the Runner target settings page, make sure your Development Team is selected. The UI varies depending on your version of Xcode.
  - For Xcode 10, look under **General > Signing > Team**.
  - For Xcode 11 and newer, look under **Signing & Capabilities > Team**.
  - When you select a team, Xcode creates and downloads a Development Certificate, registers your device with your account, and creates and downloads a provisioning profile (if needed).
  - To start your first iOS development project, you might need to sign into Xcode with your Apple
    ID.


Runner
iPhone 7 Plus
Runner: Ready | Today at 6:32 PM
Runner
Runner
General
Capabilities
Resource Tags
Info
Build Settings
Build Phase
Identity and Type
Project Document
Cocoa Touch Class - A Cocoa Touch class
UI Test Case Class - A class implementing a unit test
Unit Test Case Class - A class implementing a unit test
Filter
Add Account...

Identity

Display Name: test_app
Bundle Identifier: com.yourcompany.testApp
Version: 1.0
Build: 1

Signing

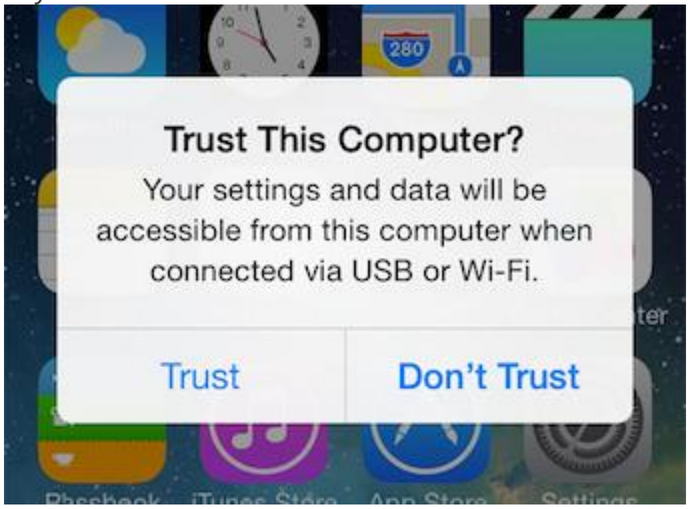
No accounts found
Add a developer account to sign your app.

Deployment Info

Deployment Target: 8.0
Devices: Universal
Main Interface: Main

Name: Runner
Location: Relative to Group
Runner.xcodeproj
Full Path: /Users/fluttermac/test_app/ios/Runner.xcodeproj
Project Format: Xcode 3.2-compatible
Organization: The Chromium Authors

 Development and testing is supported for any Apple ID. Enrolling in the Apple Developer Program is required to distribute your app to the App Store. For details about membership types, see [Choosing a Membership](#).
  - The first time you use an attached physical device for iOS development, you need to trust both your Mac and the Development Certificate on that device. Select Trust in the dialog prompt when first connecting the iOS device to your Mac.



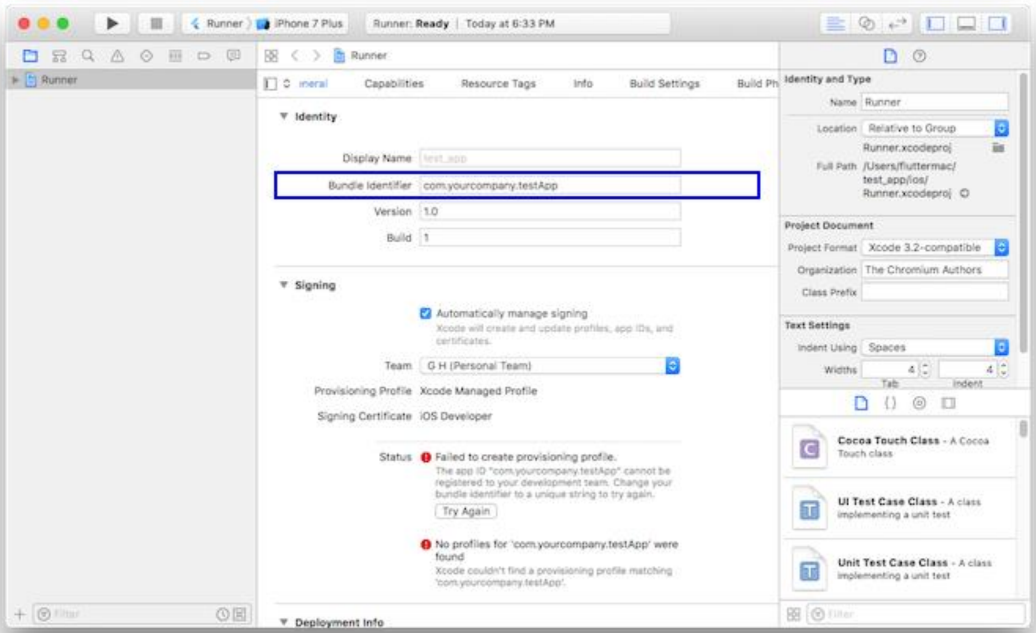
Trust This Computer?

Your settings and data will be accessible from this computer when connected via USB or Wi-Fi.

    -
    - Then, go to the Settings app on the iOS device, select **General > Device Management** and trust your Certificate. For first time users,

you may need to select **General > Profiles > Device Management** instead.

  - If automatic signing fails in Xcode, verify that the project's **General > Identity > Bundle Identifier** value is unique.


Runner
iPhone 7 Plus
Runner: Ready | Today at 6:33 PM
Runner
neral
Capabilities
Resource Tags
Info
Build Settings
Build Ph
Try Again
Cocoa Touch Class - A Cocoa Touch class
UI Test Case Class - A class implementing a unit test
Unit Test Case Class - A class implementing a unit test
Filter
Deployment Info

Identity

Display Name: test_app
Bundle Identifier: com.yourcompany.testApp
Version: 1.0
Build: 1

Signing

☑ Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Team: G H (Personal Team)
Provisioning Profile: Xcode Managed Profile
Signing Certificate: iOS Developer

Status: Failed to create provisioning profile.
The app ID "com.yourcompany.testApp" cannot be registered to your development team. Change your bundle identifier to a unique string to try again.

No profiles for 'com.yourcompany.testApp' were found
Xcode couldn't find a provisioning profile matching 'com.yourcompany.testApp'.

Identity and Type
Name: Runner
Location: Relative to Group
Runner.xcodeproj
Full Path: /Users/fluttermac/test_app/ios/Runner.xcodeproj

Project Document
Project Format: Xcode 3.2-compatible
Organization: The Chromium Authors
Class Prefix:

Text Settings
Indent Using: Spaces
Widths: 4 (Tab) 4 (Indent)

- Start your app by running `flutter run` or clicking the Run button in Xcode.

Android setup

- **Note:** Flutter relies on a full installation of Android Studio to supply its Android platform dependencies. However, you can write your Flutter apps in a number of editors; a later step discusses that.

Install Android Studio

- Download and install Android Studio.
- Start Android Studio, and go through the 'Android Studio Setup Wizard'. This installs the latest Android SDK, Android SDK Command-line Tools, and Android SDK Build-Tools, which are required by Flutter when developing for Android.

Set up your Android device

- To prepare to run and test your Flutter app on an Android device, you need an Android device running Android 4.1 (API level 16) or higher.
- Enable **Developer options** and **USB debugging** on your device. Detailed instructions are available in the Android documentation.
- Windows-only: Install the Google USB Driver.
- Using a USB cable, plug your phone into your computer. If prompted on your device, authorize your computer to access your device.

- In the terminal, run the flutter devices command to verify that Flutter recognizes your connected Android device. By default, Flutter uses the version of the Android SDK where your adb tool is based. If you want Flutter to use a different installation of the Android SDK, you must set the ANDROID_SDK_ROOT environment variable to that installation directory.

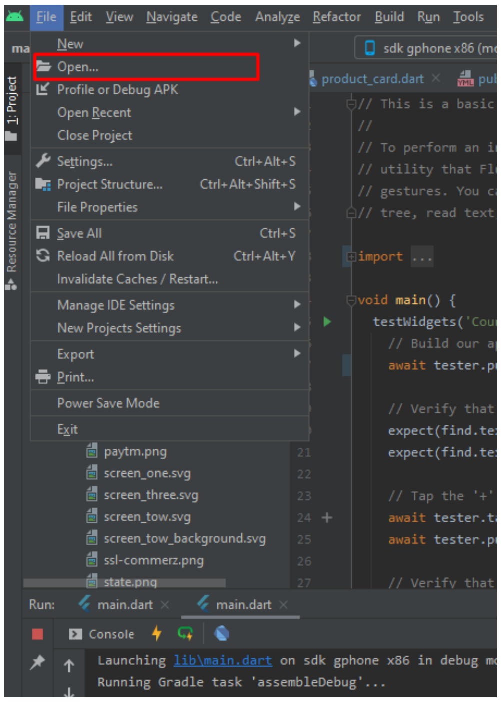
Set up the Android emulator

- To prepare to run and test your Flutter app on the Android emulator, follow these steps:
- Enable VM acceleration on your machine.
- Launch **Android Studio**, click the **AVD Manager** icon, and select **Create Virtual Device…**
    - In older versions of Android Studio, you should instead launch **Android Studio > Tools > Android > AVD Manager** and select **Create Virtual Device…**. (The **Android** submenu is only present when inside an Android project.)
    - If you do not have a project open, you can choose **Configure > AVD Manager** and select **Create Virtual Device…**
- Choose a device definition and select **Next**.
- Select one or more system images for the Android versions you want to emulate, and select **Next**. An *x86* or *x86_64* image is recommended.
- Under Emulated Performance, select **Hardware - GLES 2.0** to enable hardware acceleration.
- Verify the AVD configuration is correct, and select **Finish**.
    - For details on the above steps, see Managing AVDs.
- In Android Virtual Device Manager, click **Run** in the toolbar. The emulator starts up and displays the default canvas for your selected OS version and device.

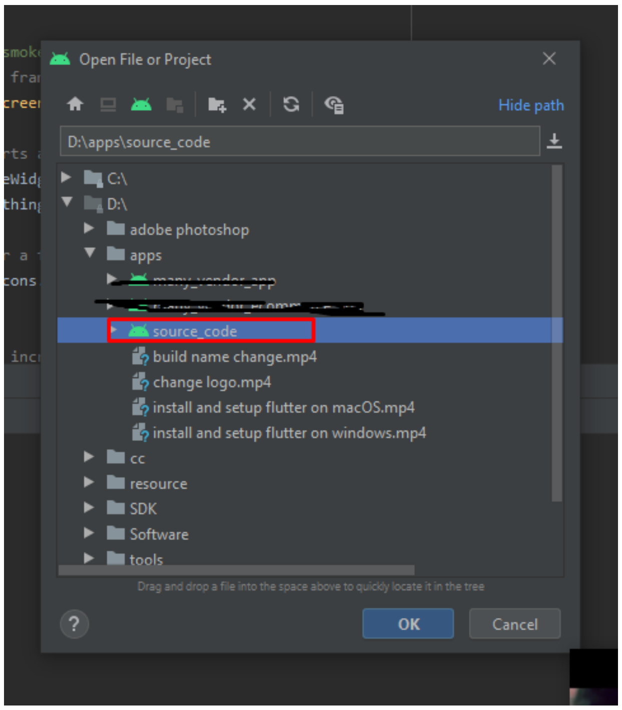
- **Download the Android Studio or vs code. You can follow this link** https://youtu.be/YPKYT1buIVU

- **Open the downloaded mobile app with Android Studio/VSCode ide.**

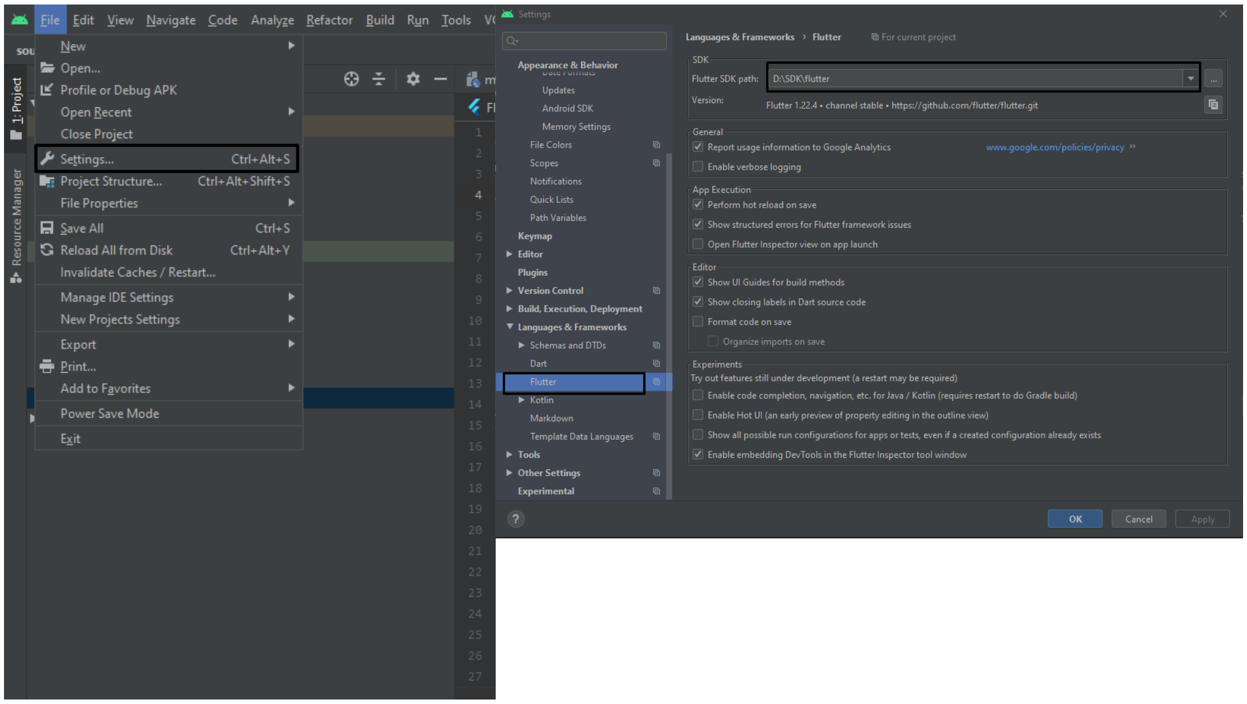
  Follow this images,



```
Launching lib\main.dart on sdk gphone x86 in debug mo
Running Gradle task 'assembleDebug'...
```



This way open **source_code** project



Languages & Frameworks › Flutter   For current project

SDK
Flutter SDK path: D:\SDK\flutter
Version: Flutter 1.22.4 • channel stable • https://github.com/flutter/flutter.git

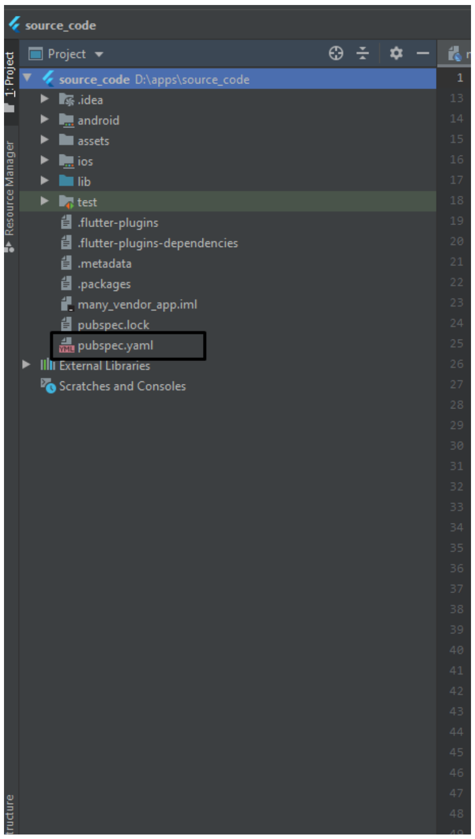
General
- [✓] Report usage information to Google Analytics   www.google.com/policies/privacy
- [ ] Enable verbose logging

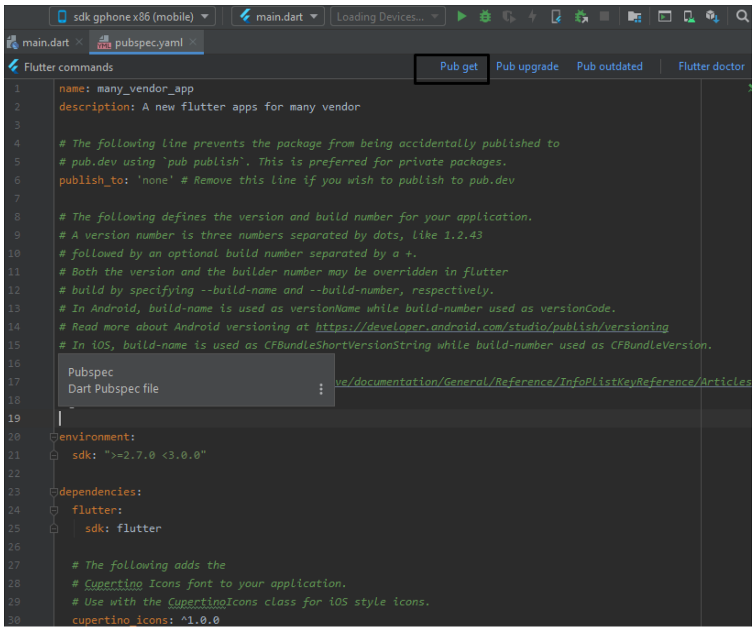
App Execution
- [✓] Perform hot reload on save
- [✓] Show structured errors for Flutter framework issues
- [ ] Open Flutter Inspector view on app launch

Editor
- [✓] Show UI Guides for build methods
- [✓] Show closing labels in Dart source code
- [ ] Format code on save
- [ ] Organize imports on save

Experiments
Try out features still under development (a restart may be required)
- [ ] Enable code completion, navigation, etc. for Java / Kotlin (requires restart to do Gradle build)
- [ ] Enable Hot UI (an early preview of property editing in the outline view)
- [ ] Show all possible run configurations for apps or tests, even if a created configuration already exists
- [✓] Enable embedding DevTools in the Flutter Inspector tool window

Goto **File > setting > Languages & Frameworks > Flutter** save the flutter SDK path properly. If every thing is ok click the **ok** button

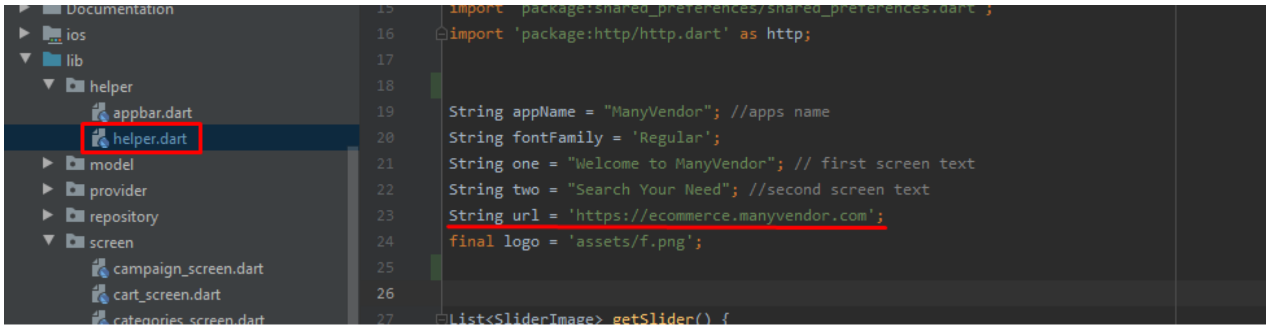




```
name: many_vendor_app
description: A new flutter apps for many vendor

# The following line prevents the package from being accidentally published to
# pub.dev using `pub publish`. This is preferred for private packages.
publish_to: 'none' # Remove this line if you wish to publish to pub.dev

# The following defines the version and build number for your application.
# A version number is three numbers separated by dots, like 1.2.43
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as versionCode.
# Read more about Android versioning at https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-number used as CFBundleVersion.

                           ve/documentation/General/Reference/InfoPlistKeyReference/Articles


environment:
  sdk: ">=2.7.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter

  # The following adds the
  # Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.0
```

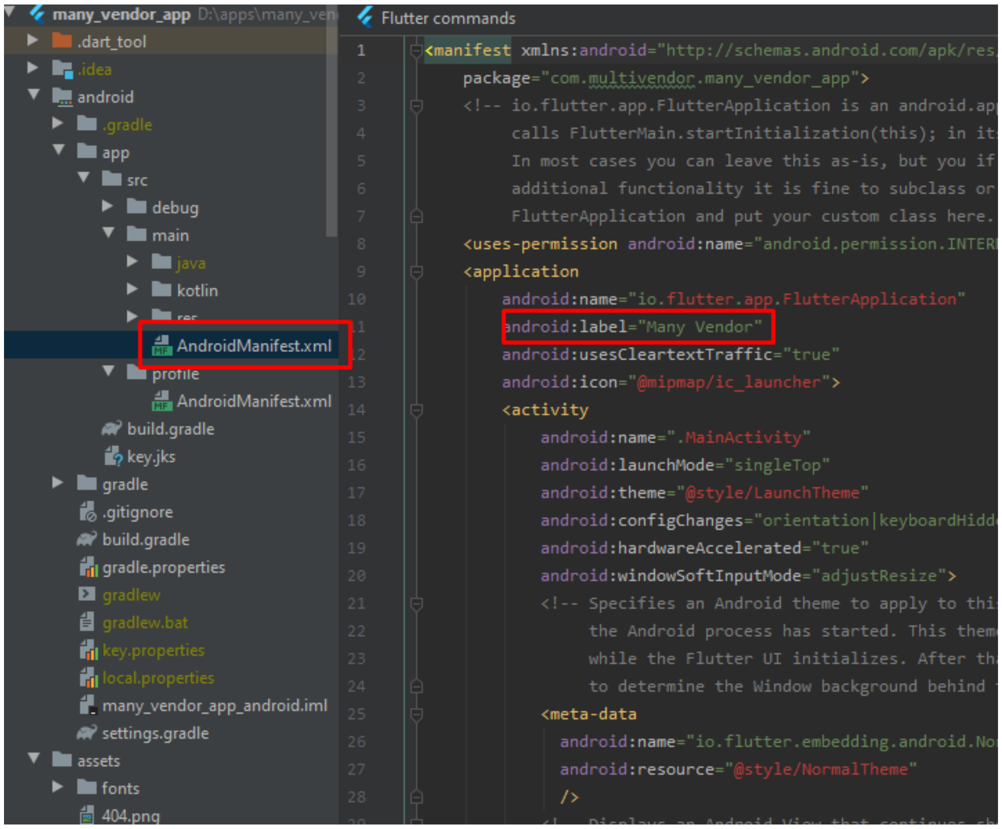
Open the **pubspec.yaml** file - Click the **Pub get**, after click **pubget** update all package and sync the project or packages

- **Open the file name "helper.dart". Which is located at "<u>lib/helper</u>".**
- **Change appName from Line 19**
- **Go to line number 23.**
- **You will find a variable declared called "URL"**
- **Provide your hosted Manyvendor eCommerce & Multi-Vendor CMS application on the placeholder "Follow the picture".**



```
15  import 'package:shared_preferences/shared_preferences.dart';
16  import 'package:http/http.dart' as http;
17
18
19  String appName = "ManyVendor"; //apps name
20  String fontFamily = 'Regular';
21  String one = "Welcome to ManyVendor"; // first screen text
22  String two = "Search Your Need"; //second screen text
23  String url = 'https://ecommerce.manyvendor.com';
24  final logo = 'assets/f.png';
25
26
27  List<SliderImage> getSlider() {
```

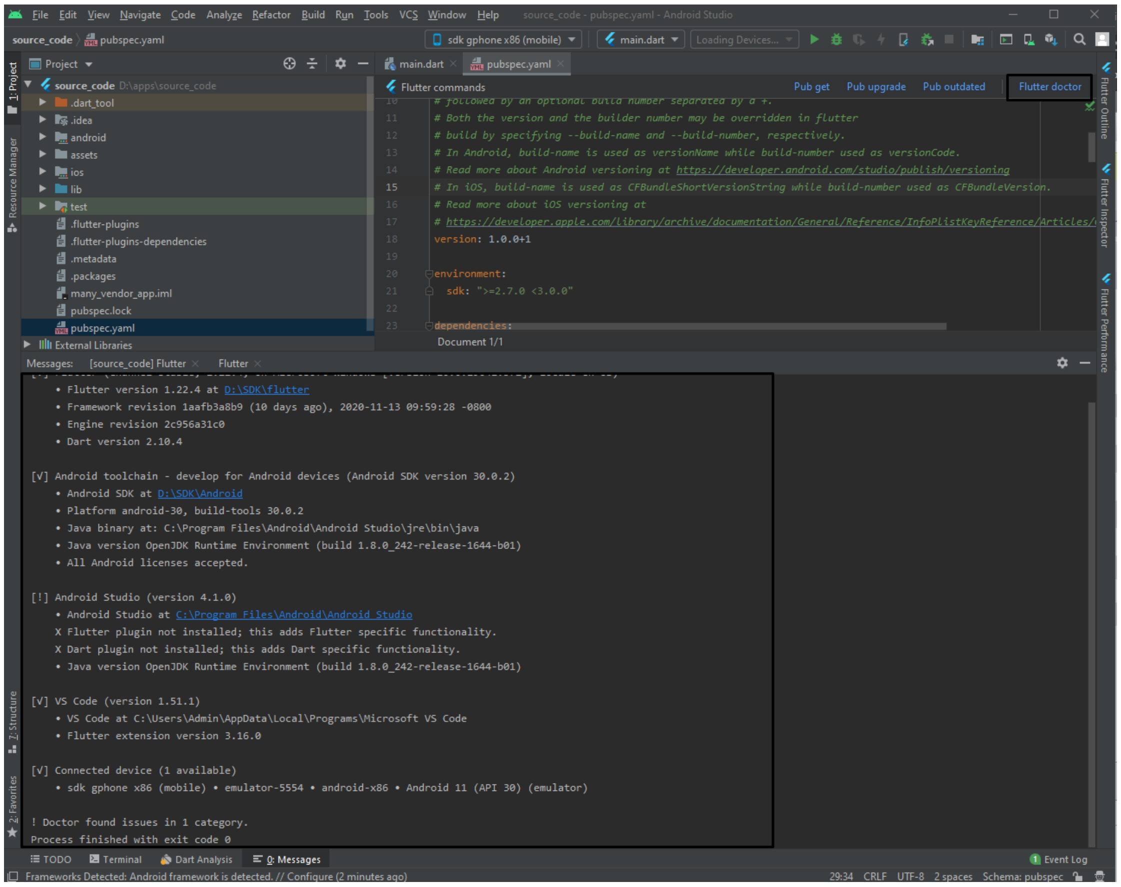
Also, Change the app name from here android: label follows this picture.



```
<manifest xmlns:android="http://schemas.android.com/apk/res
    package="com.multivendor.many_vendor_app">
    <!-- io.flutter.app.FlutterApplication is an android.ap
         calls FlutterMain.startInitialization(this); in it
         In most cases you can leave this as-is, but you if
         additional functionality it is fine to subclass or
         FlutterApplication and put your custom class here.
    <uses-permission android:name="android.permission.INTER
    <application
        android:name="io.flutter.app.FlutterApplication"
        android:label="Many Vendor"
        android:usesCleartextTraffic="true"
        android:icon="@mipmap/ic_launcher">
        <activity
            android:name=".MainActivity"
            android:launchMode="singleTop"
            android:theme="@style/LaunchTheme"
            android:configChanges="orientation|keyboardHidd
            android:hardwareAccelerated="true"
            android:windowSoftInputMode="adjustResize">
            <!-- Specifies an Android theme to apply to thi
                 the Android process has started. This them
                 while the Flutter UI initializes. After th
                 to determine the Window background behind 
            <meta-data
              android:name="io.flutter.embedding.android.No
              android:resource="@style/NormalTheme"
              />
```

Your app name and android label name must be the same.

- Check if everything is right.

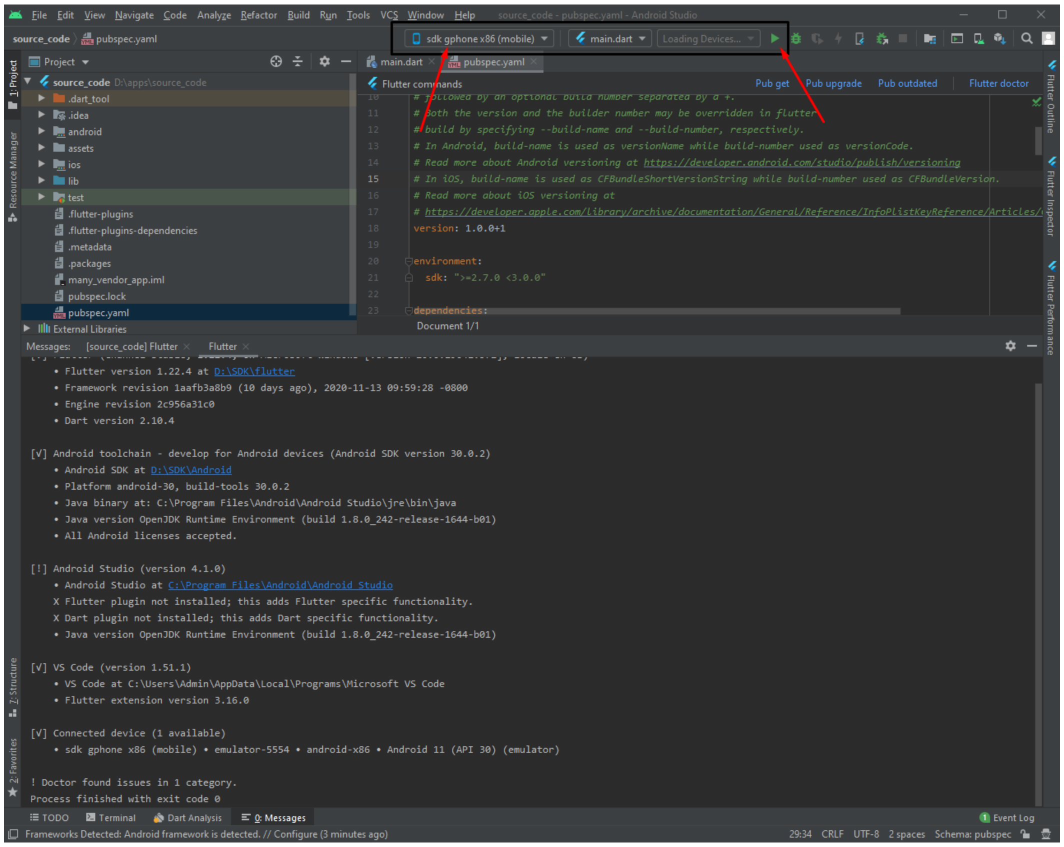


```
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as versionCode.
# Read more about Android versioning at https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-number used as CFBundleVersion.
# Read more about iOS versioning at
# https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/
version: 1.0.0+1

environment:
  sdk: ">=2.7.0 <3.0.0"

dependencies:
```

```
    • Flutter version 1.22.4 at D:\SDK\flutter
    • Framework revision 1aafb3a8b9 (10 days ago), 2020-11-13 09:59:28 -0800
    • Engine revision 2c956a31c0
    • Dart version 2.10.4

[√] Android toolchain - develop for Android devices (Android SDK version 30.0.2)
    • Android SDK at D:\SDK\Android
    • Platform android-30, build-tools 30.0.2
    • Java binary at: C:\Program Files\Android\Android Studio\jre\bin\java
    • Java version OpenJDK Runtime Environment (build 1.8.0_242-release-1644-b01)
    • All Android licenses accepted.

[!] Android Studio (version 4.1.0)
    • Android Studio at C:\Program Files\Android\Android Studio
    X Flutter plugin not installed; this adds Flutter specific functionality.
    X Dart plugin not installed; this adds Dart specific functionality.
    • Java version OpenJDK Runtime Environment (build 1.8.0_242-release-1644-b01)

[√] VS Code (version 1.51.1)
    • VS Code at C:\Users\Admin\AppData\Local\Programs\Microsoft VS Code
    • Flutter extension version 3.16.0

[√] Connected device (1 available)
    • sdk gphone x86 (mobile) • emulator-5554 • android-x86 • Android 11 (API 30) (emulator)

! Doctor found issues in 1 category.
Process finished with exit code 0
```

Frameworks Detected: Android framework is detected. // Configure (2 minutes ago)

If all setup is perfect, press **Flutter doctor** in top. If all is ok in flutter setup on your pc show the console success belong this image



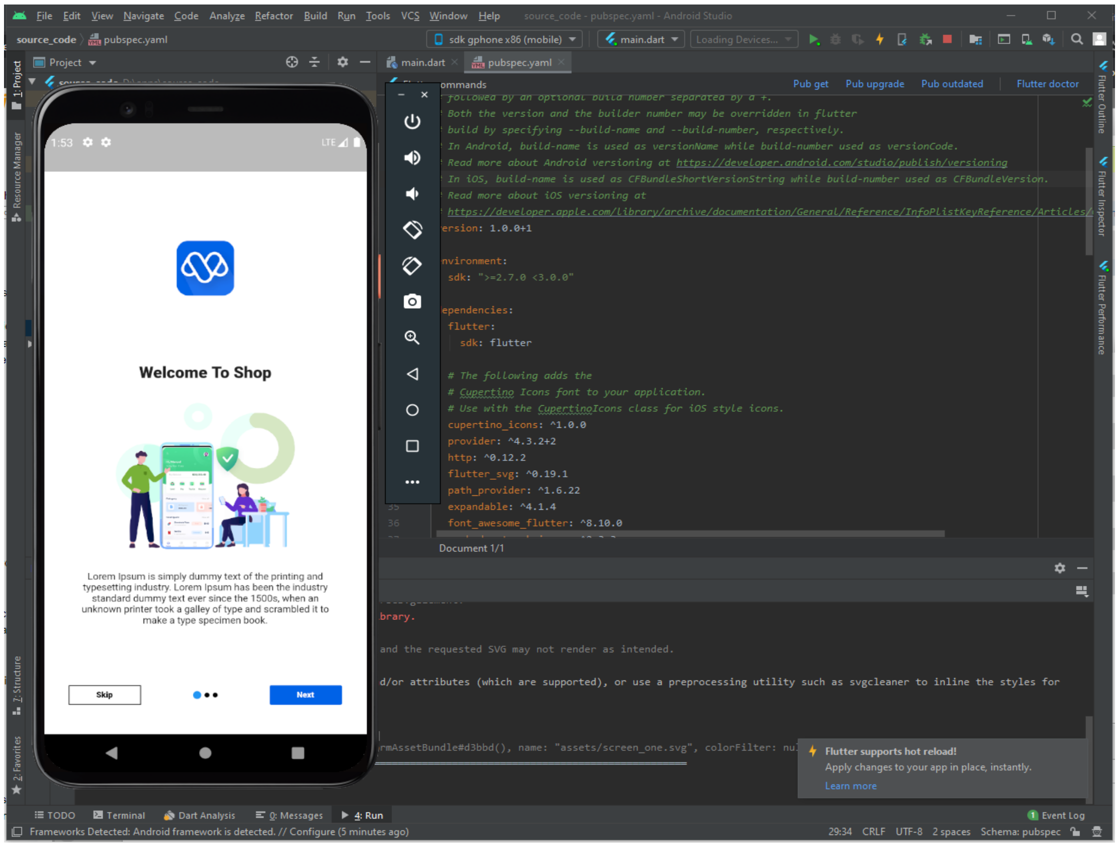
```
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as versionCode.
# Read more about Android versioning at https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-number used as CFBundleVersion.
# Read more about iOS versioning at
# https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/
version: 1.0.0+1

environment:
  sdk: ">=2.7.0 <3.0.0"

dependencies:
```

```
    • Flutter version 1.22.4 at D:\SDK\flutter
    • Framework revision 1aafb3a8b9 (10 days ago), 2020-11-13 09:59:28 -0800
    • Engine revision 2c956a31c0
    • Dart version 2.10.4

[√] Android toolchain - develop for Android devices (Android SDK version 30.0.2)
    • Android SDK at D:\SDK\Android
    • Platform android-30, build-tools 30.0.2
    • Java binary at: C:\Program Files\Android\Android Studio\jre\bin\java
    • Java version OpenJDK Runtime Environment (build 1.8.0_242-release-1644-b01)
    • All Android licenses accepted.

[!] Android Studio (version 4.1.0)
    • Android Studio at C:\Program Files\Android\Android Studio
    X Flutter plugin not installed; this adds Flutter specific functionality.
    X Dart plugin not installed; this adds Dart specific functionality.
    • Java version OpenJDK Runtime Environment (build 1.8.0_242-release-1644-b01)

[√] VS Code (version 1.51.1)
    • VS Code at C:\Users\Admin\AppData\Local\Programs\Microsoft VS Code
    • Flutter extension version 3.16.0

[√] Connected device (1 available)
    • sdk gphone x86 (mobile) • emulator-5554 • android-x86 • Android 11 (API 30) (emulator)

! Doctor found issues in 1 category.
Process finished with exit code 0
```

Frameworks Detected: Android framework is detected. // Configure (3 minutes ago)

Open the android **Emulator** and click the **play icon** for run the project in your pc.

If All is ok project run in your **emulator.**



```
followed by an optional build number separated by a +.
Both the version and the builder number may be overridden in flutter
build by specifying --build-name and --build-number, respectively.
In Android, build-name is used as versionName while build-number used as versionCode.
Read more about Android versioning at https://developer.android.com/studio/publish/versioning
In iOS, build-name is used as CFBundleShortVersionString while build-number used as CFBundleVersion.
Read more about iOS versioning at
https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/
ersion: 1.0.0+1

nvironment:
  sdk: ">=2.7.0 <3.0.0"

ependencies:
  flutter:
    sdk: flutter

  # The following adds the
  # Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.0
  provider: ^4.3.2+2
  http: ^0.12.2
  flutter_svg: ^0.19.1
  path_provider: ^1.6.22
  expandable: ^4.1.4
  font_awesome_flutter: ^8.10.0
```

```
brary.

and the requested SVG may not render as intended.

d/or attributes (which are supported), or use a preprocessing utility such as svgcleaner to inline the styles for

rmAssetBundle#d3bbd(), name: "assets/screen_one.svg", colorFilter: nu
```

Flutter supports hot reload!
Apply changes to your app in place, instantly.
Learn more

Frameworks Detected: Android framework is detected. // Configure (5 minutes ago)

Welcome To Shop

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

No Enjoy You successfully done to run your Flutter Application,

Build the application For Release:

- Change Android Package Name Follow this link. Or see this video https://youtu.be/BhpmHlN2Kjg
- Change Application Launcher Icon, Follow this link.
- Build the app follow this for the
  - **Android link,**

Build and release an Android app

- During a typical development cycle, you test an app using `flutter run` at the command line, or by using the **Run** and **Debug** options in your IDE. By default, Flutter builds a *debug* version of your app.
- When you’re ready to prepare a *release* version of your app, for example, to publish to the Google Play Store, this page can help. Before publishing, you might want to put some finishing touches on your app. This page covers the following topics:
- Adding a launcher icon
- Signing the app
- Shrinking your code with R8

Adding a launcher icon

- When a new Flutter app is created, it has a default launcher icon. To customize this icon, you might want to check out the flutter_launcher_icons package.
- Alternatively, you can do it manually using the following steps:
- Review the Material Design product icons guidelines for icon design.
- In the `<app dir>/android/app/src/main/res/` directory, place your icon files in folders named using configuration qualifiers. The default `mipmap-` folders demonstrate the correct naming convention.
- In `AndroidManifest.xml`, update the `application` tag’s `android:icon` attribute to reference icons from the previous step (for example, `<application android:icon="@mipmap/ic_launcher" ...`).
- To verify that the icon has been replaced, run your app, and inspect the app icon in the Launcher.

Signing the app

- To publish on the Play Store, you need to give your app a digital signature. Use the following instructions to sign your app.

Create a Keystore

- If you have an existing keystore, skip to the next step. If not, create one by running the following at the command line:
- **On Mac/Linux, use the following command:**

```
keytool -genkey -v -keystore ~/key.jks -keyalg RSA -keysize 2048 -
validity 10000 -alias key
```

- **On Windows, use the following command:**

```
keytool -genkey -v -keystore c:\Users\USER_NAME\key.jks -storetype JKS -
keyalg RSA -keysize 2048 -validity 10000 -alias key
```

- This command stores the `key.jks` file in your home directory. If you want to store it elsewhere, change the argument you pass to the `-keystore` parameter. **However, keep the `keystore` file private; don’t check it into public source control!**

Reference the keystore from the app

- Create a file named `<app dir>/android/key.properties` that contains a reference to your keystore:

```
storePassword=<password from previous step>
keyPassword=<password from previous step>
keyAlias=key
storeFile=<location of the key store file, such as /Users/<user
name>/key.jks>
```

- **Warning:** Keep the `key.properties` file private; don't check it into public source control.

Configure signing in Gradle

- Configure signing for your app by editing the `<app dir>/android/app/build.gradle` file.
- Add code before `android` block:

```
android {
...
}
```

  With the keystore information from your properties file:

```
def keystoreProperties = new Properties()
def keystorePropertiesFile = rootProject.file('key.properties')
if (keystorePropertiesFile.exists()) {
keystoreProperties.load(new
FileInputStream(keystorePropertiesFile))
}

android {
...
}
```

  - Load the `key.properties` file into the `keystoreProperties` object.
- Add code before `buildTypes` block:

```
buildTypes {
release {
// TODO: Add your own signing config for the release build.
// Signing with the debug keys for now,
// so `flutter run --release` works.
signingConfig signingConfigs.debug
}
}
```

  With the signing configuration info:

```
signingConfigs {
release {
keyAlias keystoreProperties['keyAlias']
keyPassword keystoreProperties['keyPassword']
storeFile keystoreProperties['storeFile']
file(keystoreProperties['storeFile']) : null
storePassword keystoreProperties['storePassword']
}
}
buildTypes {
release {
signingConfig signingConfigs.release
}
}
```

  - Configure the `signingConfigs` block in your module's `build.gradle` file.
- Release builds of your app will now be signed automatically.
- **Note:** You may need to run `flutter clean` after changing the gradle file. This prevents cached builds from affecting the signing process.
- For more information on signing your app, see Sign your app on developer.android.com.

Shrinking your code with R8

- R8 is the new code shrinker from Google, and it's enabled by default when you build a release APK or AAB. To disable R8, pass the `--no-shrink` flag to `flutter build apk` or `flutter build appbundle`.
- **Note:** Obfuscation and minification can considerably extend compile time of the Android application.

Reviewing the app manifest

- Review the default App Manifest file, `AndroidManifest.xml`, located in `<app dir>/android/app/src/main` and verify that the values are correct, especially the following:
- `application`

  Edit the `android:label` in the `application` tag to reflect the final name of the app.

- `uses-permission`

  Add the `android.permission.INTERNET` permission if your application code needs Internet access. The standard template does not include this tag but allows Internet access during development to enable communication between Flutter tools and a running app.

Reviewing the build configuration

- Review the default Gradle build file, `build.gradle`, located in `<app dir>/android/app` and verify the values are correct, especially the following values in the `defaultConfig` block:
- `applicationId` Specify the final, unique (Application Id)appid
- `versionCode` & `versionName`

  Specify the internal app version number, and the version number display string. You can do this by setting the `version` property in the pubspec.yaml file. Consult the version information guidance in the versions documentation.

- `minSdkVersion`, `compilesdkVersion`, & `targetSdkVersion`

  Specify the minimum API level, the API level on which the app was compiled, and the maximum API level on which the app is designed to run. Consult the API level section in the versions documentation for details. `buildToolsVersion`

  Specify the version of Android SDK Build Tools that your app uses. Alternatively, you can use the [Android Gradle Plugin] in Android Studio, which will automatically import the minimum required Build Tools for your app without the need for this property.

Building the app for release

- You have two possible release formats when publishing to the Play Store.
- App bundle (preferred)
- APK
- **Note:** The Google Play Store prefers the app bundle format. For more information, see Android App Bundle and About Android App Bundles.
- **Warning:** Recently, the Flutter team has received several reports from developers indicating they are experiencing app crashes on certain devices on Android 6.0. If you are targeting Android 6.0, use the following steps:

- If you build an App Bundle Edit `android/gradle.properties` and add the flag: `android.bundle.enableUncompressedNativeLibs=false`.
- If you build an APK Make sure `android/app/src/AndroidManifest.xml` doesn't set `android:extractNativeLibs=false` in the `<application>` tag.
- For more information, see the public issue.

Build an app bundle

- This section describes how to build a release app bundle. If you completed the signing steps, the app bundle will be signed. At this point, you might consider obfuscating your Dart code to make it more difficult to reverse engineer. Obfuscating your code involves adding a couple flags to your build command, and maintaining additional files to de-obfuscate stack traces.
- From the command line:
- Enter `cd <app dir>`
  (Replace `<app dir>` with your application's directory.)
- Run `flutter build appbundle`
  (Running `flutter build` defaults to a release build.)
- The release bundle for your app is created at `<app dir>/build/app/outputs/bundle/release/app.aab`.
- By default, the app bundle contains your Dart code and the Flutter runtime compiled for armeabi-v7a (ARM 32-bit), arm64-v8a (ARM 64-bit), and x86-64 (x86 64-bit).

Test the app bundle

- An app bundle can be tested in multiple ways—this section describes two.

Offline using the bundle tool

- If you haven't done so already, download `bundletool` from the GitHub repository.
- Generate a set of APKs from your app bundle.
- Deploy the APKs to connected devices.

Online using Google Play

- Upload your bundle to Google Play to test it. You can use the internal test track, or the alpha or beta channels to test the bundle before releasing it in production.
- Follow these steps to upload your bundle to the Play Store.

Build an APK

- Although app bundles are preferred over APKs, there are stores that don't yet support app bundles. In this case, build a release APK for each target ABI (Application Binary Interface).
- If you completed the signing steps, the APK will be signed. At this point, you might consider obfuscating your Dart code to make it more difficult to reverse engineer. Obfuscating your code involves adding a couple flags to your build command.
- From the command line:

- Enter `cd <app dir>`
  (Replace `<app dir>` with your application's directory.)
- Run `flutter build apk --split-per-abi`
  (The `flutter build` command defaults to `--release`.)
- This command results in three APK files:
- `<app dir>/build/app/outputs/apk/release/app-armeabi-v7a-release.apk`
- `<app dir>/build/app/outputs/apk/release/app-arm64-v8a-release.apk`
- `<app dir>/build/app/outputs/apk/release/app-x86_64-release.apk`
- Removing the `--split-per-abi` flag results in a fat APK that contains your code compiled for *all* the target ABIs. Such APKs are larger in size than their split counterparts, causing the user to download native binaries that are not applicable to their device's architecture.

Install an APK on a device

- Follow these steps to install the APK on a connected Android device.
- From the command line:
- Connect your Android device to your computer with a USB cable.
- Enter `cd <app dir>` where `<app dir>` is your application directory.
- Run `flutter install`.

Publishing to the Google Play Store

- For detailed instructions on publishing your app to the Google Play Store, see the Google Play launch documentation.

Updating the app's version number

- The default version number of the app is `1.0.0`. To update it, navigate to the `pubspec.yaml` file and update the following line:
- `version: 1.0.0+1`
- The version number is three numbers separated by dots, such as `1.0.0` in the example above, followed by an optional build number such as `1` in the example above, separated by a `+`.
- Both the version and the build number may be overridden in Flutter's build by specifying `--build-name` and `--build-number`, respectively.
- In Android, `build-name` is used as `versionName` while `build-number` used as `versionCode`. For more information, see Version your app in the Android documentation.
- After updating the version number in the pubspec file, run `flutter pub get` from the top of the project, or use the **Pub get** button in your IDE. This updates the `versionName` and `versionCode` in the `local.properties` file, which are later updated in the `build.gradle` file when you rebuild the Flutter app.

For Build iOS Follow this link:

Build and release an iOS app

- This guide provides a step-by-step walkthrough of releasing a Flutter app to the App Store and TestFlight.
    - Preliminaries
- Before beginning the process of releasing your app, ensure that it meets Apple's App Review Guidelines.
- In order to publish your app to the App Store, you must first enroll in the Apple Developer Program. You can read more about the various membership options in Apple's Choosing a Membership guide.

Register your app on App Store Connect

- Manage your app's life cycle on App Store Connect (formerly iTunes Connect). You define your app name and description, add screenshots, set pricing, and manage releases to the App Store and TestFlight.
- Registering your app involves two steps: registering a unique Bundle ID, and creating an application record on App Store Connect.
- For a detailed overview of App Store Connect, see the App Store Connect guide.

Register a Bundle ID

- Every iOS application is associated with a Bundle ID, a unique identifier registered with Apple. To register a Bundle ID for your app, follow these steps:
- Open the App IDs page of your developer account.
- Click **+** to create a new Bundle ID.
- Enter an app name, select **Explicit App ID**, and enter an ID.
- Select the services your app uses, then click **Continue**.
- On the next page, confirm the details and click **Register** to register your Bundle ID.

Create an application record on App Store Connect

- **Register your app on App Store Connect**:
- Open App Store Connect in your browser.
- On the App Store Connect landing page, click **My Apps**.
- Click **+** in the top-left corner of the My Apps page, then select **New App**.
- Fill in your app details in the form that appears. In the Platforms section, ensure that iOS is checked. Since Flutter does not currently support tvOS, leave that checkbox unchecked. Click **Create**.
- Navigate to the application details for your app and select **App Information** from the sidebar.
- In the General Information section, select the Bundle ID you registered in the preceding step.
- For a detailed overview, see Add an app to your account.

Review Xcode project settings

- This step covers reviewing the most important settings in the Xcode workspace. For detailed procedures and descriptions, see Prepare for app distribution.
- **Navigate to your target's settings in Xcode:**
- In Xcode, open Runner.xcworkspace in your app's ios folder.
- To view your app's settings, select the **Runner** project in the Xcode project navigator. Then, in the main view sidebar, select the **Runner** target.
- Select the **General** tab.
- Verify the most important settings.

- In the **Identity** section:
- Display Name

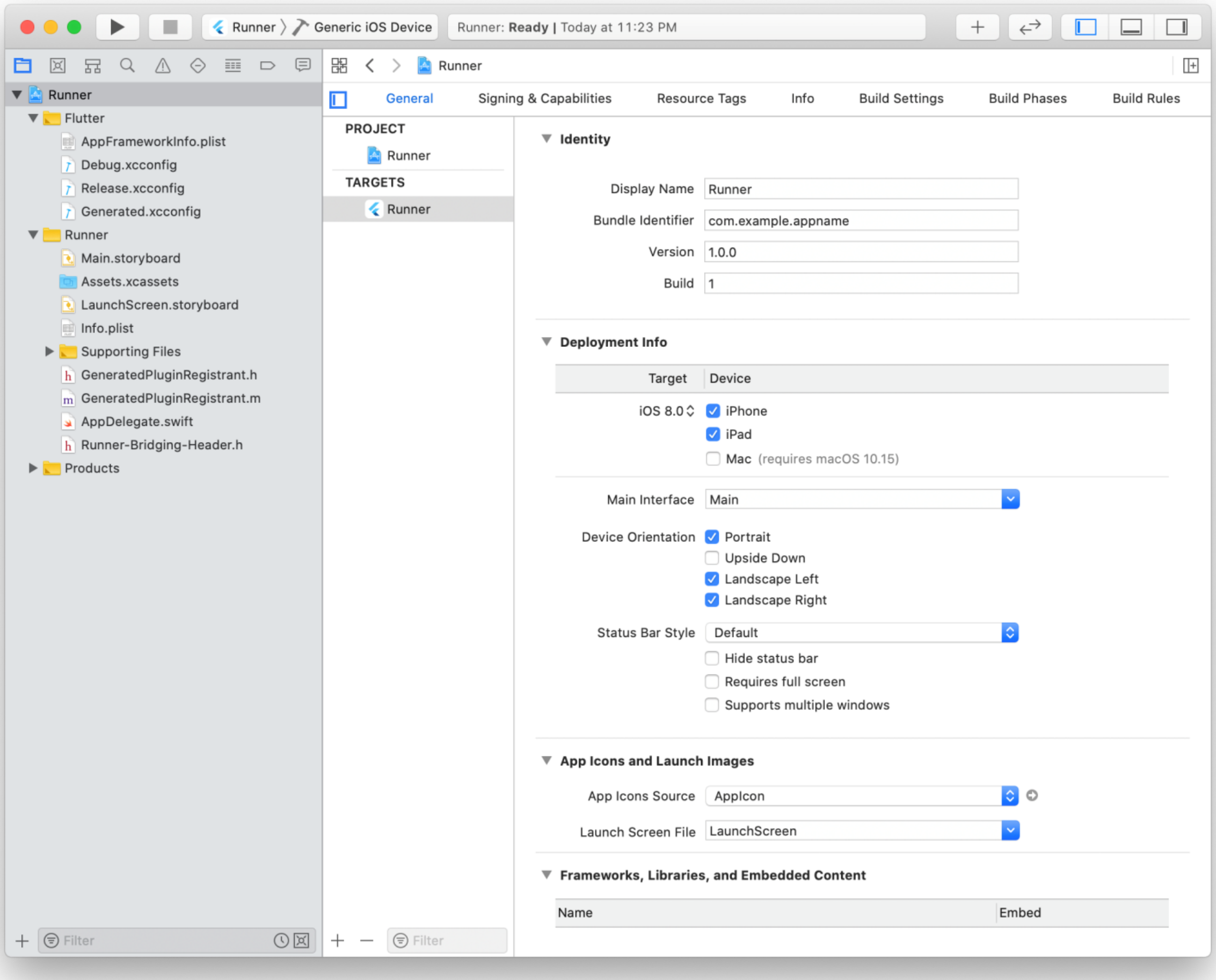
  The display name of your app.
- Bundle Identifier

  The App ID you registered on App Store Connect.
- In the **Signing & Capabilities** section:
- Automatically manage signing

  Whether Xcode should automatically manage app signing and provisioning. This is set true by default, which should be sufficient for most apps. For more complex scenarios, see the Code Signing Guide.
- Team

  Select the team associated with your registered Apple Developer account. If required, select **Add Account…**, then update this setting.
- In the **Build Settings** section:
- iOS Deployment Target

  The minimum iOS version that your app supports. Flutter supports iOS 8.0 and later. If your app includes Objective-C or Swift code that makes use of APIs that were unavailable in iOS 8, update this setting appropriately.
- The **General** tab of your project settings should resemble the following:



Identity

Display Name: Runner
Bundle Identifier: com.example.appname
Version: 1.0.0
Build: 1

Deployment Info

Target	Device
iOS 8.0	✓ iPhone
	✓ iPad
	☐ Mac (requires macOS 10.15)

Main Interface: Main

Device Orientation: ✓ Portrait, ☐ Upside Down, ✓ Landscape Left, ✓ Landscape Right

Status Bar Style: Default
☐ Hide status bar
☐ Requires full screen
☐ Supports multiple windows

App Icons and Launch Images

App Icons Source: AppIcon
Launch Screen File: LaunchScreen

Frameworks, Libraries, and Embedded Content

Name	Embed

- For a detailed overview of app signing, see Create, export, and delete signing certificates.

Updating the app’s deployment version

- If you changed Deployment Target in your Xcode project, open ios/Flutter/AppframeworkInfo.plist in your Flutter app and update the MinimumOSVersion value to match.

Updating the app’s version number

- The default version number of the app is 1.0.0. To update it, navigate to the pubspec.yaml file and update the following line:
- version: 1.0.0+1
- The version number is three numbers separated by dots, such as 1.0.0 in the example above, followed by an optional build number such as 1 in the example above, separated by a +.
- Both the version and the build number may be overridden in Flutter’s build by specifying --build-name and --build-number, respectively.
- In iOS, build-name uses CFBundleShortVersionString while build-number uses CFBundleVersion. Read more about iOS versioning at Core Foundation Keys on the Apple Developer’s site.

Add an app icon

- When opening these apps a placeholder icon set is created. This step covers replacing these placeholder icons with your app's icons:
- Review the iOS App Icon guidelines.
- In the Xcode project navigator, select Assets.xcassets in the Runner folder. Update the placeholder icons with your own app icons.
- Verify the icon has been replaced by running your app using flutter run.

Create a build archive

- This step covers creating a build archive and uploading your build to App Store Connect.
- During development, you've been building, debugging, and testing with *debug* builds. When you're ready to ship your app to users on the App Store or TestFlight, you need to prepare a *release* build. At this point, you might consider obfuscating your Dart code to make it more difficult to reverse engineer. Obfuscating your code involves adding a couple flags to your build command.
- On the command line, follow these steps in your application directory:
- Run flutter build ios to create a release build (flutter build defaults to --release).
- To ensure that Xcode refreshes the release mode configuration, close and re-open your Xcode workspace. For Xcode 8.3 and later, this step is not required.
- **In Xcode, configure the app version and build:**
- In Xcode, open Runner.xcworkspace in your app's ios folder.
- Select **Product > Scheme > Runner**.
- Select **Product > Destination > Any iOS Device**.
- Select **Runner** in the Xcode project navigator, then select the **Runner** target in the settings view sidebar.
- In the Identity section, update the **Version** to the user-facing version number you wish to publish.
- In the Identity section, update the **Build** identifier to a unique build number used to track this build on App Store Connect. Each upload requires a unique build number.
- Finally, create a build archive and upload it to App Store Connect:
- Select **Product > Archive** to produce a build archive.
- In the sidebar of the Xcode Organizer window, select your iOS app, then select the build archive you just produced.
- Click the **Validate App** button. If any issues are reported, address them and produce another build. You can reuse the same build ID until you upload an archive.
- After the archive has been successfully validated, click **Distribute App**. You can follow the status of your build in the Activities tab of your app's details page on App Store Connect.
- You should receive an email within 30 minutes notifying you that your build has been validated and is available to release to testers on TestFlight. At this point you can choose whether to release on TestFlight, or go ahead and release your app to the App Store.
- For more details, see Upload an app to App Store Connect.

Release your app on TestFlight

- TestFlight allows developers to push their apps to internal and external testers. This optional step covers releasing your build on TestFlight.
- Navigate to the TestFlight tab of your app's application details page on App Store Connect.
- Select **Internal Testing** in the sidebar.
- Select the build to publish to testers, then click **Save**.

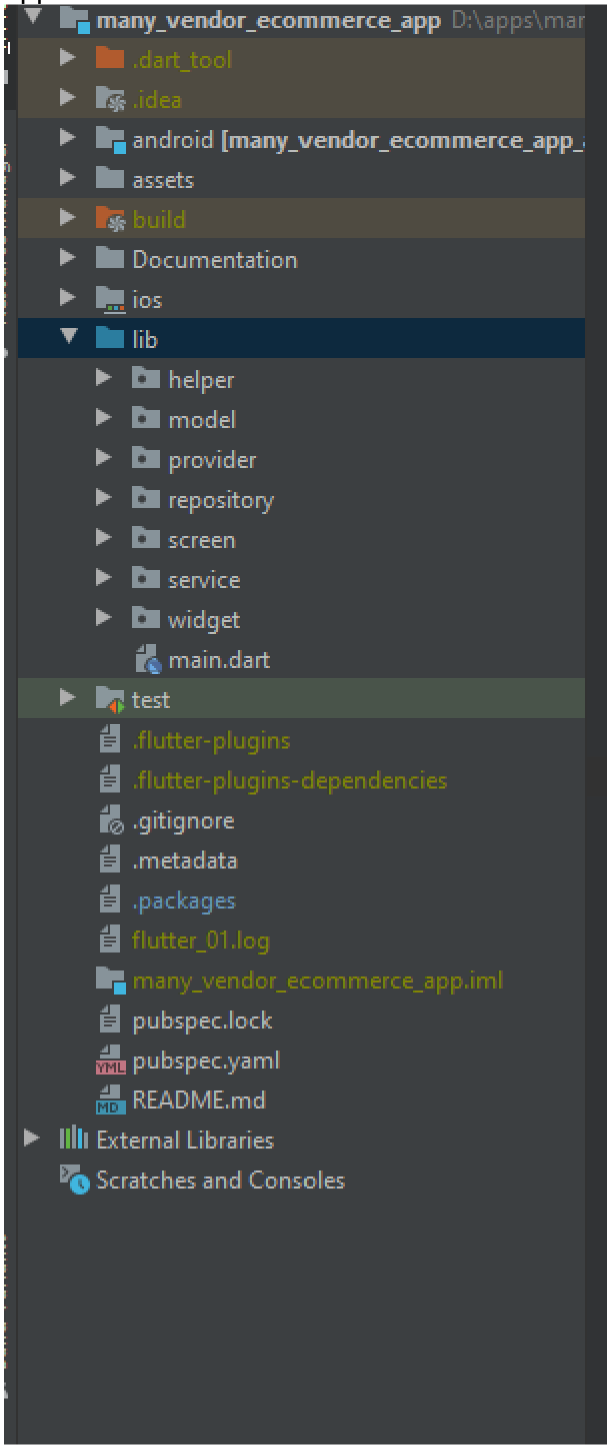
- Add the email addresses of any internal testers. You can add additional internal users in the **Users and Roles** page of App Store Connect, available from the dropdown menu at the top of the page.
- For more details, see Distribute an app using TestFlight.

Release your app to the App Store

- When you're ready to release your app to the world, follow these steps to submit your app for review and release to the App Store:
- Select **Pricing and Availability** from the sidebar of your app's application details page on App Store Connect and complete the required information.
- Select the status from the sidebar. If this is the first release of this app, its status is **1.0 Prepare for Submission**. Complete all required fields.
- Click **Submit for Review**.
- Apple notifies you when its app review process is complete. Your app is released according to the instructions you specified in the **Version-Release** section.
- For more details, see Distribute an app through the App Store.
    - If you have done this good so far, You will find the build option on the top navigation menu.
        - Go to the terminal. Run flutter build apk from the terminal. It will build a release apk.

- Distribution
    - Manual Distribution
        - If you have built it successfully, you will find the apk file inside: "**build/app/outputs/flutter-apk/app- release.apk**". You can distribute this application manually by hosting it on your server or somewhere else.
    - Google Playstore
        - You can host the application on Google Playstore as well. You will find tons of supporting videos and blogs on the internet like this https://www.youtube.com/watch?v=dR04ArAhxd4&ab_channel=GoogleDevelopers
            - Follow whichever you feel easier.
    - App Store iOS
        - You can host the application on Apple Store as well. You will find tons of supporting videos and blogs on the internet like this https://www.youtube.com/watch?v=MxejThYFDdY&ab_channel=DarranKelinske
            - Follow whichever you feel easier.

- Source code Structure
  - Here is the source code structure of the Manyvendor eCommerce customer Mobile Application.



```
▼ many_vendor_ecommerce_app D:\apps\mar
    ▶ .dart_tool
    ▶ .idea
    ▶ android [many_vendor_ecommerce_app_
    ▶ assets
    ▶ build
    ▶ Documentation
    ▶ ios
    ▼ lib
        ▶ helper
        ▶ model
        ▶ provider
        ▶ repository
        ▶ screen
        ▶ service
        ▶ widget
          main.dart
    ▶ test
      .flutter-plugins
      .flutter-plugins-dependencies
      .gitignore
      .metadata
      .packages
      flutter_01.log
      many_vendor_ecommerce_app.iml
      pubspec.lock
      pubspec.yaml
      README.md
▶ External Libraries
  Scratches and Consoles
```

All the screens we've used is in the screens folder. All the providers are inside the provider's folder. All the widgets we've used are inside provider folder are holding all provider classes, repository folder contain database operations, the screen folder contains all screens, service folder contains all service classes. widgets folder contains all custom widgets, a model folder contain all the data models we’ve used here. main.dart file contains all the Splash Screen UI. Android and ios folders contain android and ios related files respectfully. You can open the contents of the ios folder from xCode to change app icons and splash screens. Related android content in the Android folder.