#### Technology stack used

Flutter iOS, Firebase

#### Configure Flutter environment

Currently (October 9th) using the latest Flutter SDK (3.24.2), Intel chip SDK download link: <https://storage.flutter-io.cn/flutter_infra_release/releases/stable/macos/flutter_macos_3.24.2-stable.zip> , Apple chip SDK download link: <https://storage.flutter-io.cn/flutter_infra_release/releases/stable/macos/flutter_macos_arm64_3.24.2-stable.zip> ;

Create a folder for installing Flutter;

You may consider creating a directory in ~/development/;

Unzip the downloaded Flutter SDK compressed file (zip) to the ~/development/ directory;

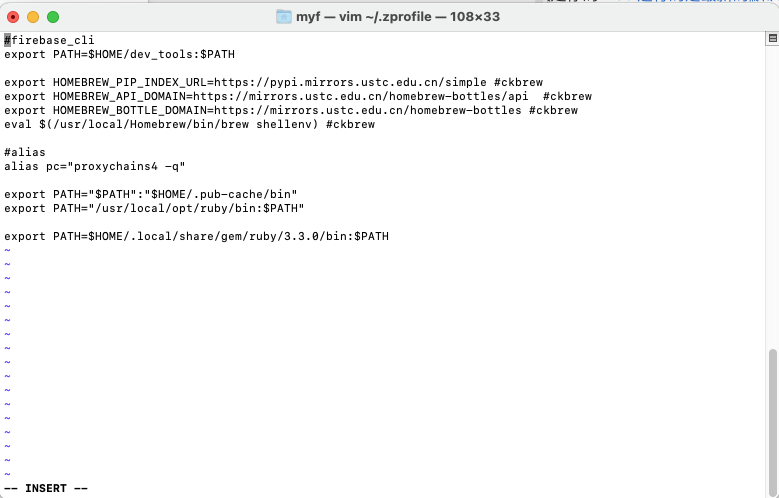
After completion, the Flutter SDK should be located in the ~/development/flutter directory.

Then add Flutter to PATH (the purpose is to use the flutter command directly in the terminal)

Currently, the default shell in macOS Terminal is Zsh, and you can use the ~/.zprofile file to handle environment variables.

Open the terminal:

Enter the command: vim ~/.zprofile, then press the i key to input text, at which point the bottom left corner will change to INSERT (ignore the content in the image below).



Move the cursor to the end to add a new line (you can paste with the right mouse button):

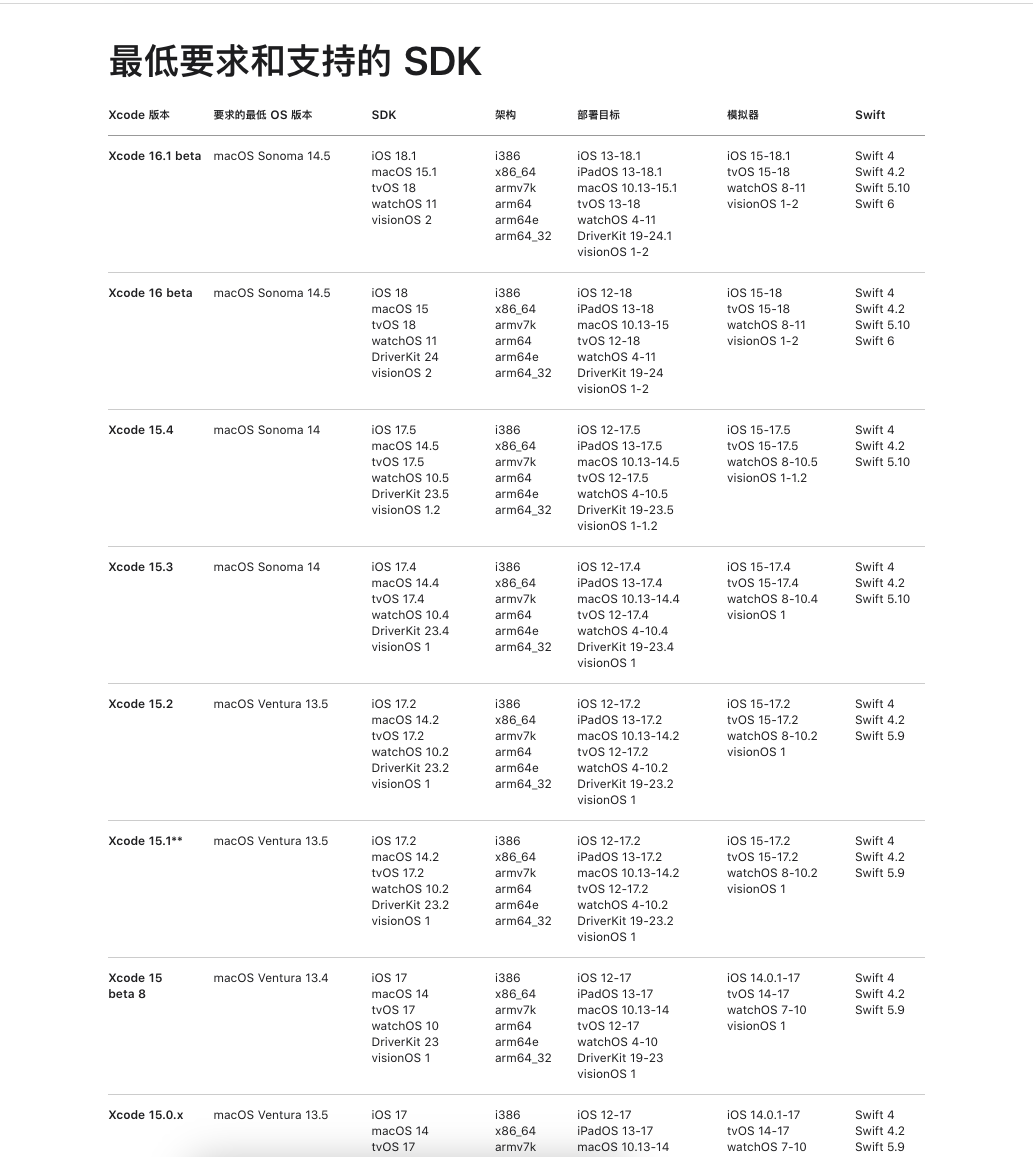
export PATH=$HOME/development/flutter/bin:$PATH

Then press the Esc key, followed by :wq and hit Enter to save and exit;

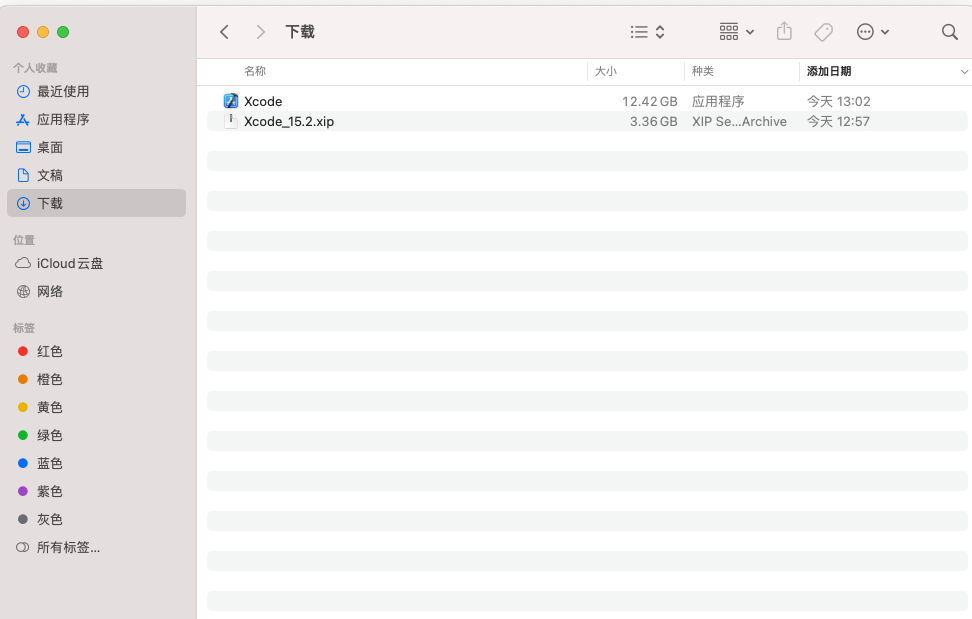
Restart all open terminal session windows to apply this change.

#### Configure Xcode

To develop for iOS, you must install Xcode, but the version installed varies by system, as shown in the image below:

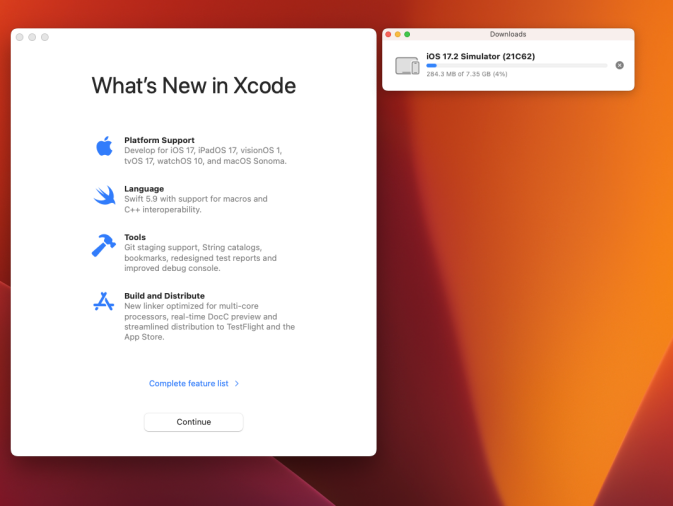


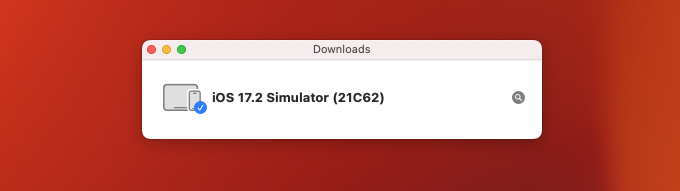
Open <https://xcodereleases.com/> find the corresponding version and click Download, then you may need to log in to your Apple account to download. After downloading, simply double-click to extract:



Drag the Xcode application from the above image to the Applications folder;

Now open Xcode, agree to the default prompts, automatically install tools, etc., and then Xcode will automatically install an iOS virtual device, which may take a long time;



After the virtual machine is installed, open it using the terminal with open -a Simulator (there are other methods to start it during the later development process);

Connect a real device later during development;

#### Install CocoaPods

You can open the terminal and type pod to see if there is any output. If there is, it means it is already installed by default; if not, continue reading:

macOS comes with Ruby by default, so you can directly use gem. Enter the following command

gem install cocoapods --user-install

This command will install CocoaPods to the user's home directory, and then configure the environment variable:

cd ~

Open ~/.zprofile, then press the i key to enter text, at this point the bottom left corner changes to INSERT

export GEM\_HOME=$HOME/.gem

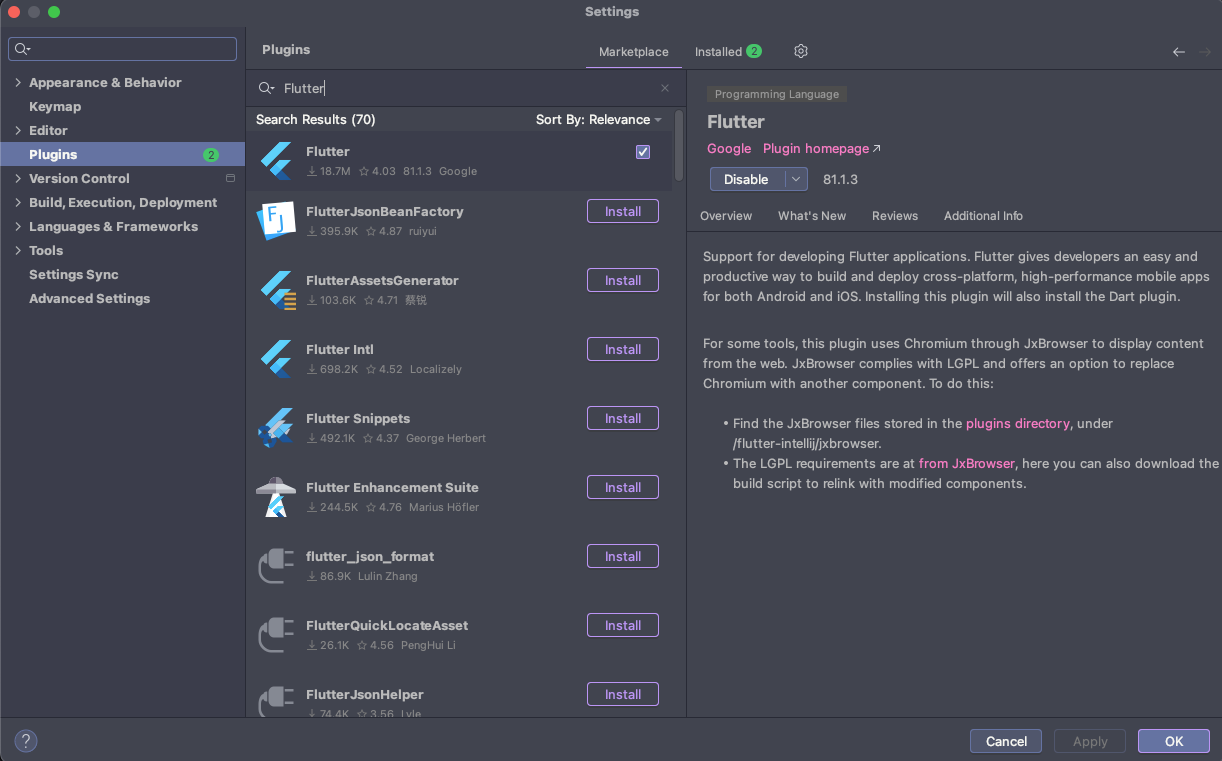
export PATH=$GEM\_HOME/bin:$PATH

Paste the above two lines at the end, then press the Esc key, followed by :wq and hit Enter to save and exit

Restart the terminal, then CocoaPods configuration is complete.

#### Download IntelliJ IDEA

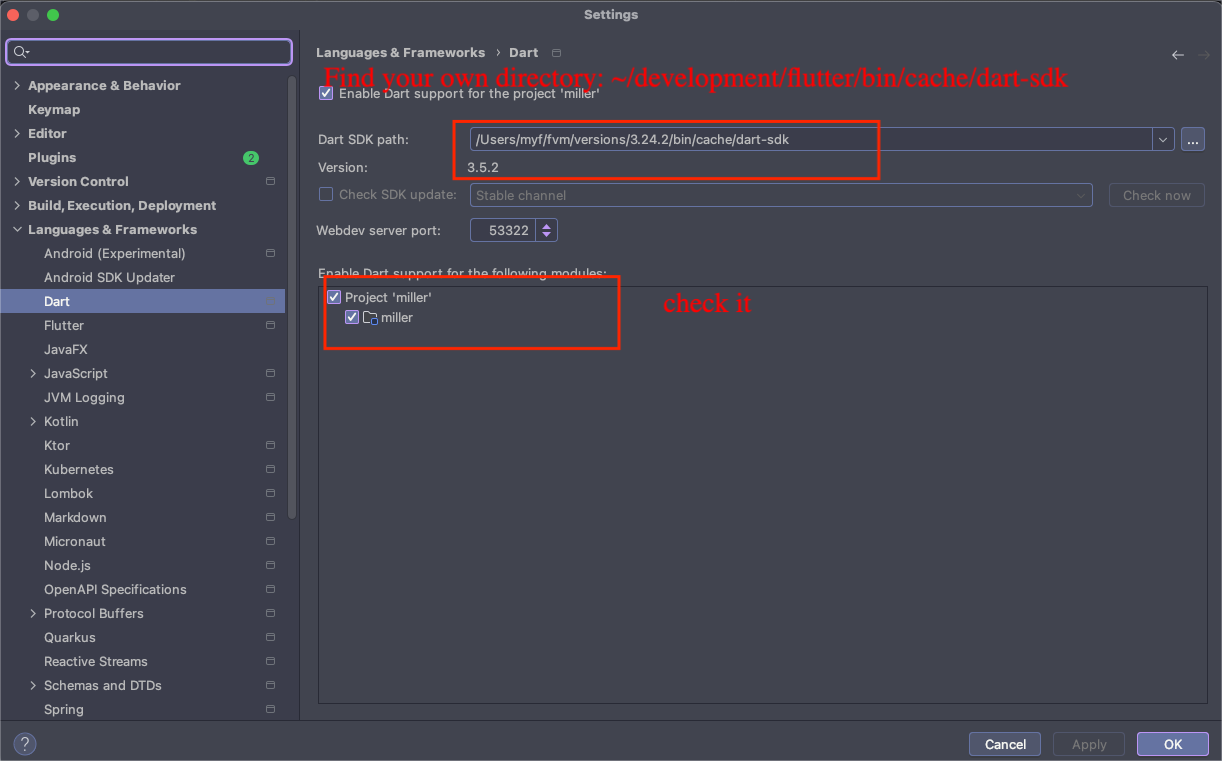
Search online and download IntelliJ IDEA, both Community and Professional editions are fine, then open IDEA and install the Flutter plugin as shown in the image below:



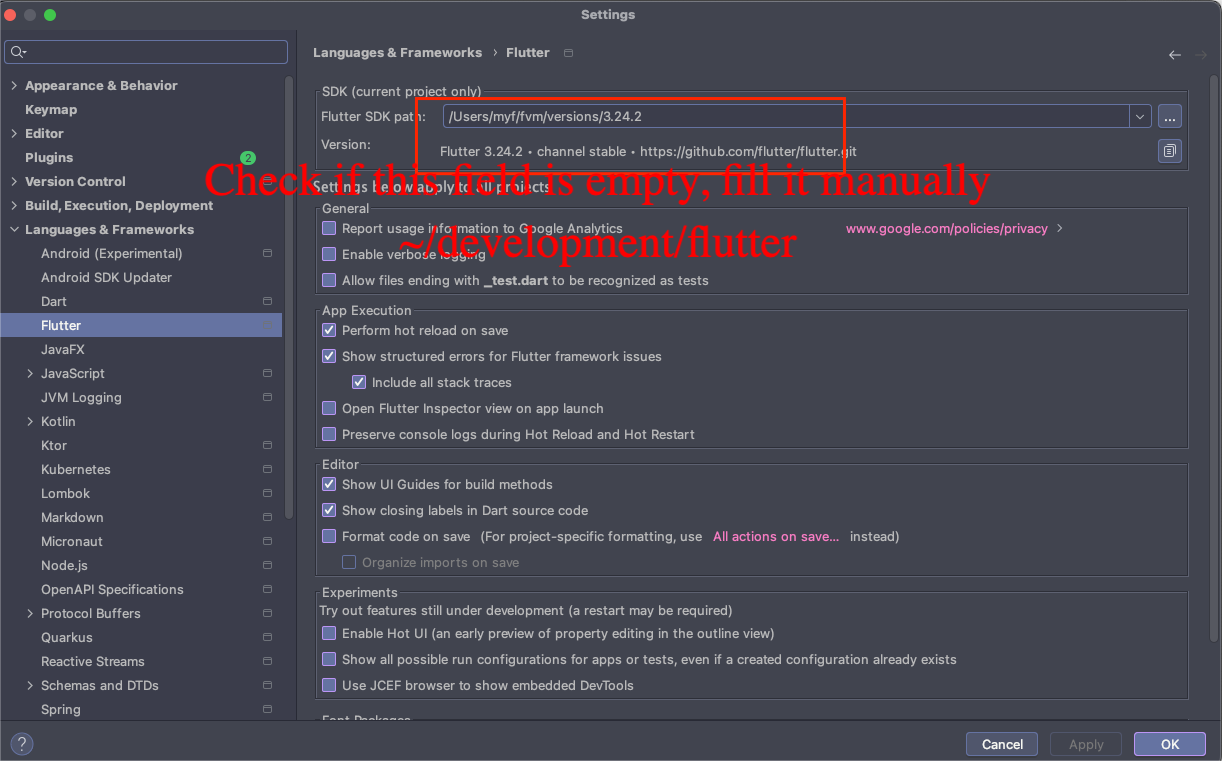
#### Start using Flutter to develop iOS applications on macOS

Open the code package, place it in a location, and open it with IDEA:

Open IDEA settings,

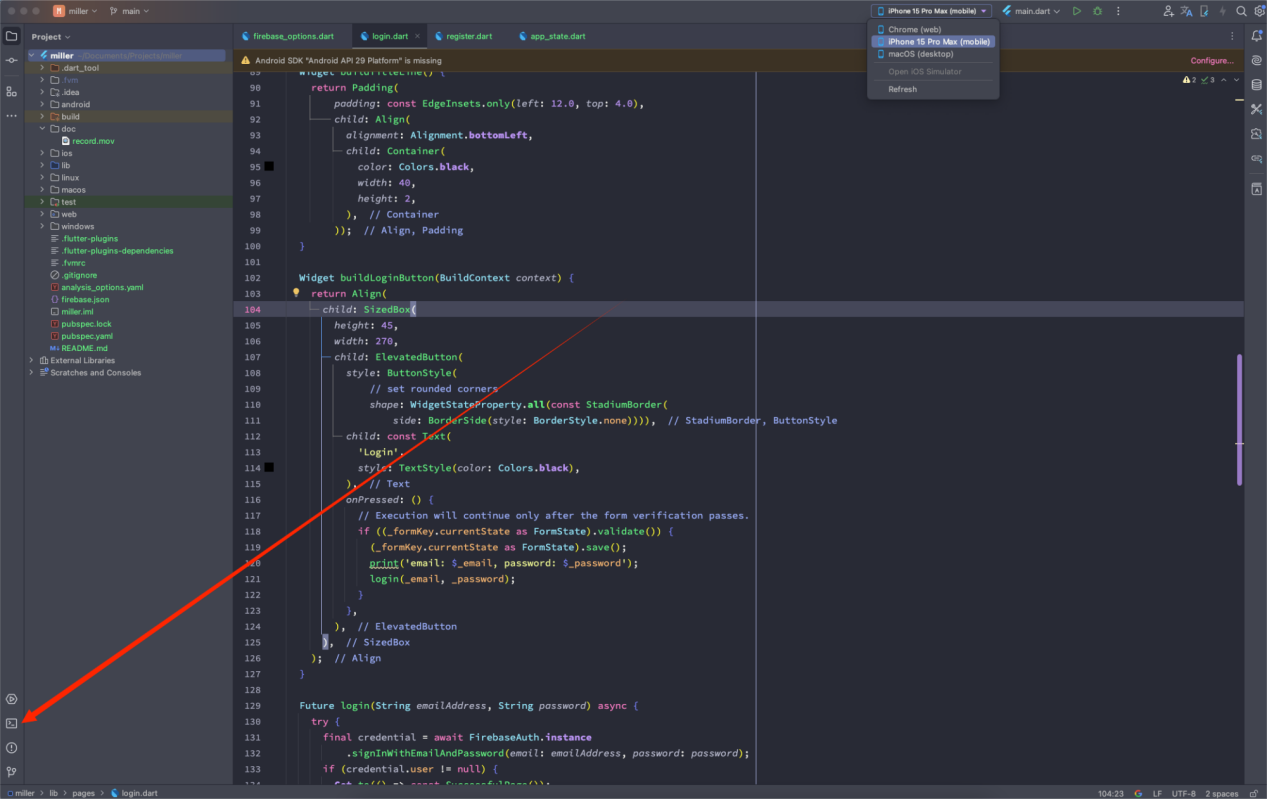


Check Flutter:



Close the settings;

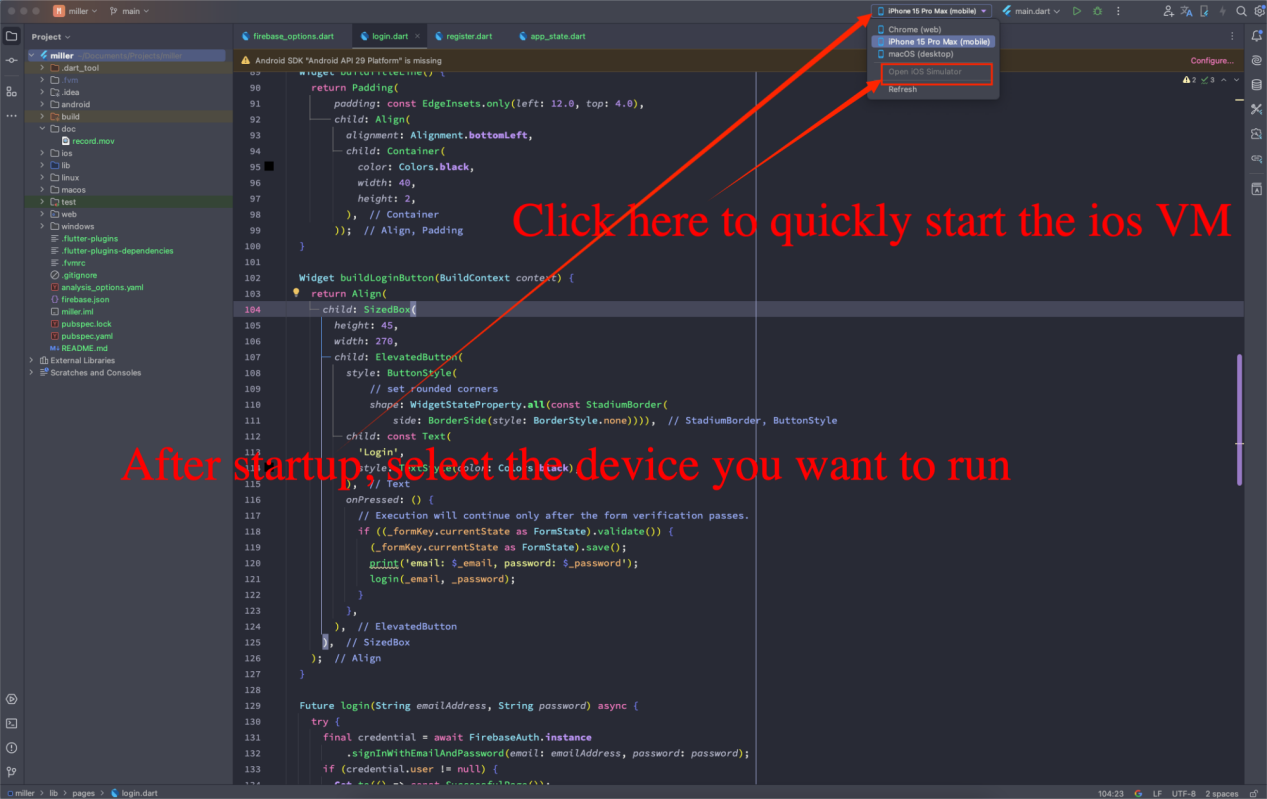
Open the terminal in IDEA:



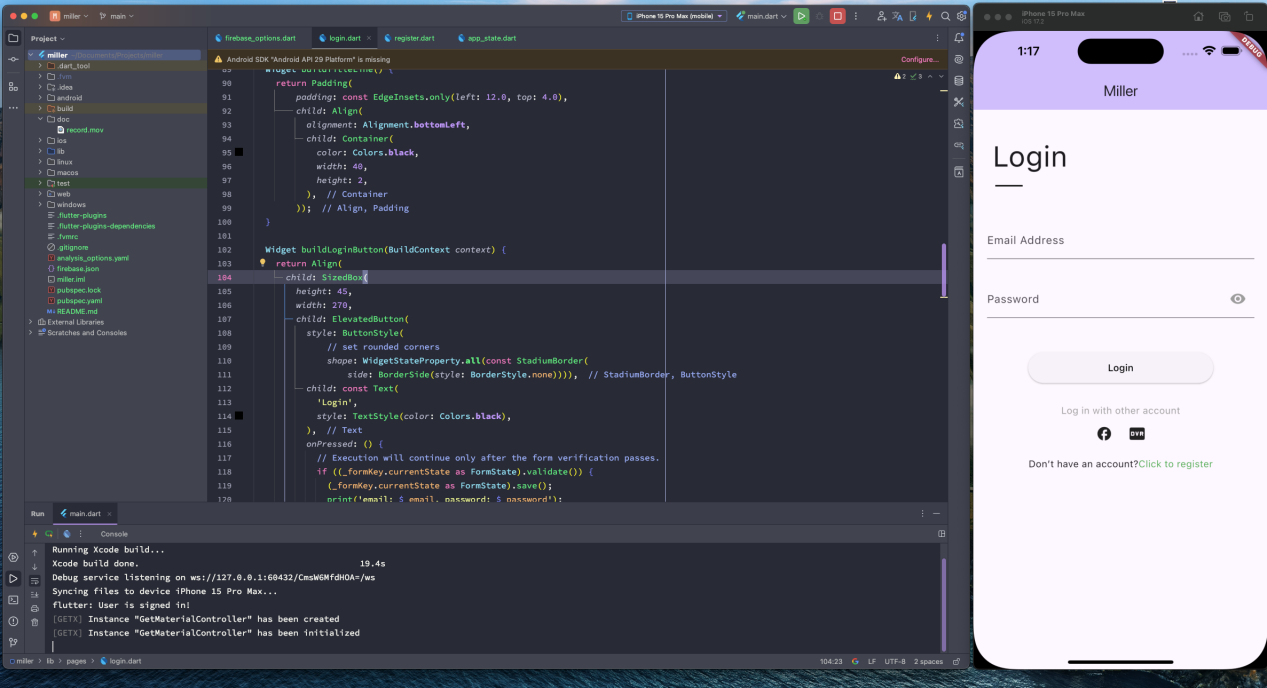
Enter flutter pub get

After completion, close the terminal;

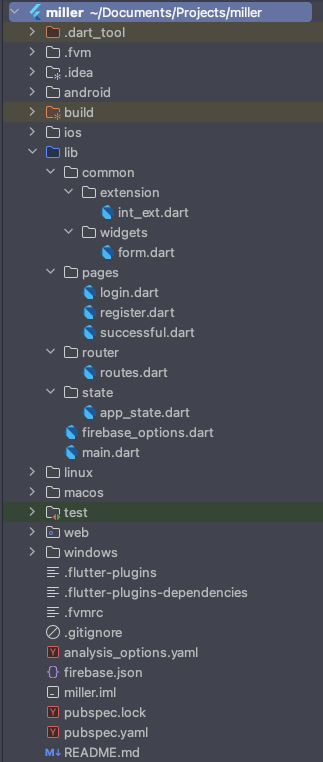
Then as shown in the image:



It will start up in a while



#### Introduction to the code directory

The following directories are unrelated to the project code and can be ignored:

.dart\_tool

.fvm

.idea

android

build

linux

macos

web

windows

Mainly introducing the lib folder:

All source code should be written in lib,

where the common folder contains reusable methods for all components,

extension is for Dart language extensions,

widgets are reusable components,

int\_ext.dart is an extension for the int type,

form.dart is a wrapper for form item components,

The pages folder contains routing page components, mainly for writing the front-end interface and logic,

login.dart is the login page,

register.dart is the registration page,

successful.dart is the login success page,

The router folder and routes.dart currently have no content, belonging to forward planning files,

The app\_state.dart in the state folder is the application state file for Firebase,

firebase\_options.dart is the Firebase configuration file, which is essential for Firebase authentication,

main.dart is the entry point of the program,

the pubspec.yaml below is the project dependency file, and the other files are automatically generated and can be ignored.

The test folder contains test code, which is not currently used in the project.

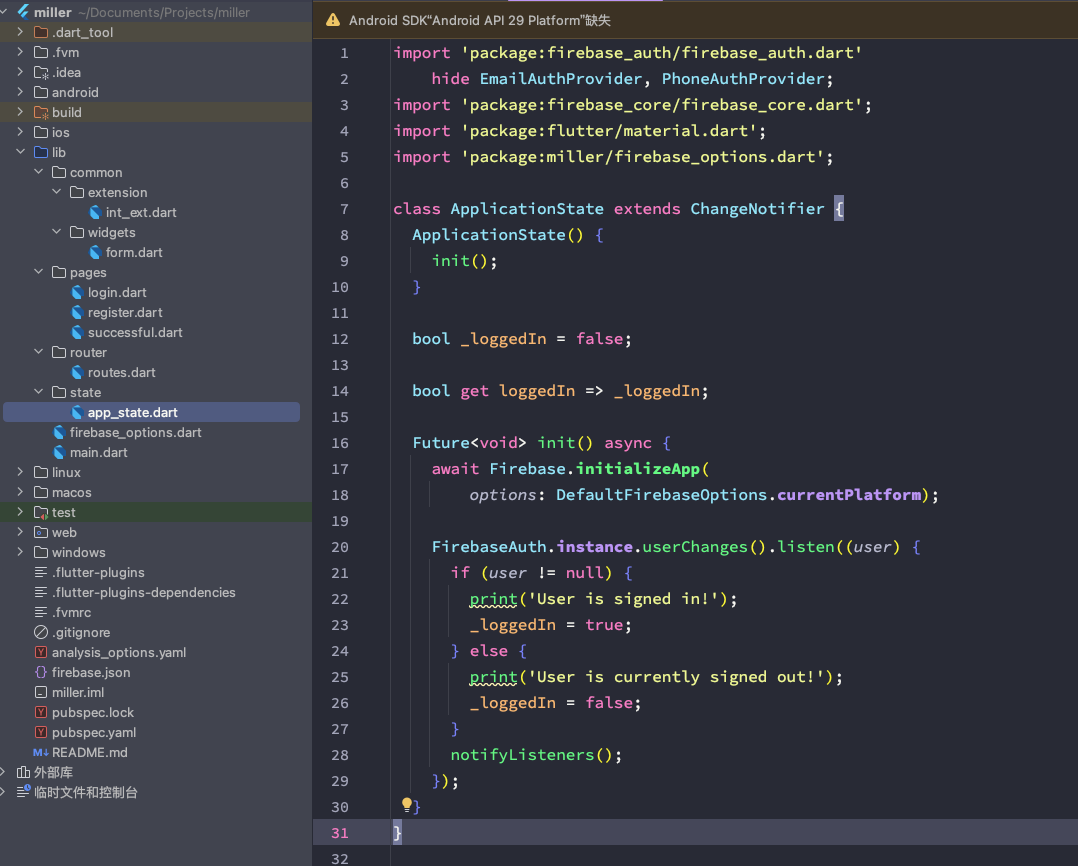
#### Technical challenges encountered

The project uses GetX, which provides rich features such as state management, dependency injection, and route management. For those who are not very familiar with it, it may take some time to master. Although GetX simplifies state management, correctly organizing and managing state in complex applications can still be quite troublesome. For page routing, GetX provides convenient routing management features, but in large applications, the configuration and transitions of routes can become complex. Managing route transitions is also a significant undertaking.

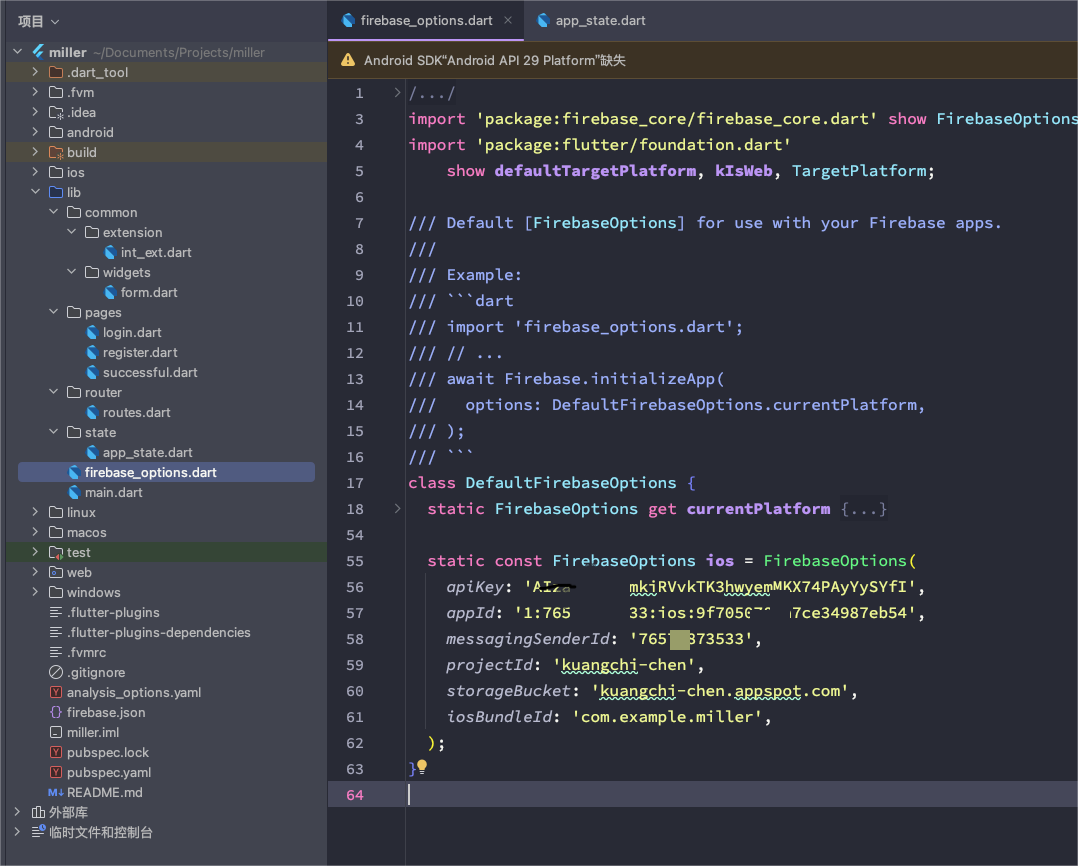
Firebase is a comprehensive mobile and web development platform provided by Google, offering a range of tools and services such as real-time databases, authentication, analytics, storage, and cloud functions. Integrating Firebase requires configuration of multiple services within existing projects. Correctly configuring the Firebase project, adding necessary dependencies, and initializing services will take considerable time. Firebase offers various authentication options, including email/password and social media logins. Properly implementing and configuring these authentication methods, as well as handling user sessions and security issues, can present some challenges.

#### Firebase authentication authentication code

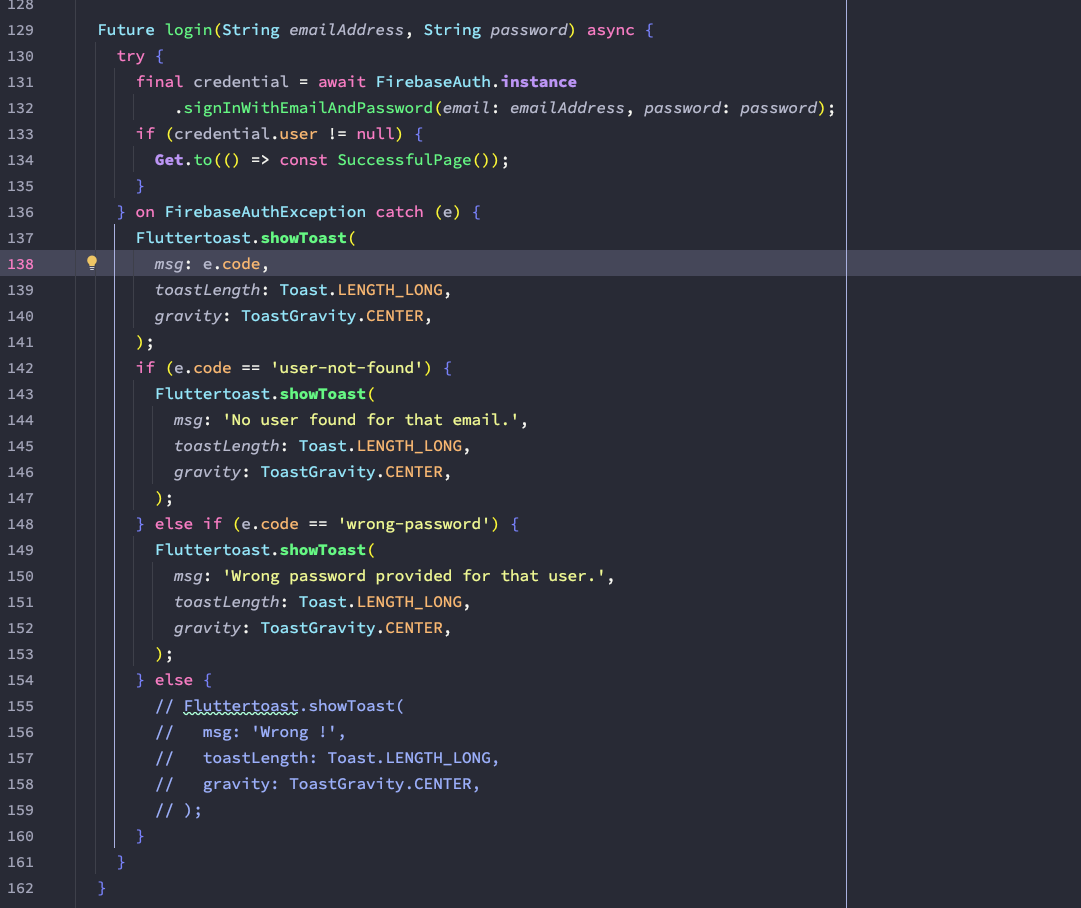
Code to initialize Firebase services:



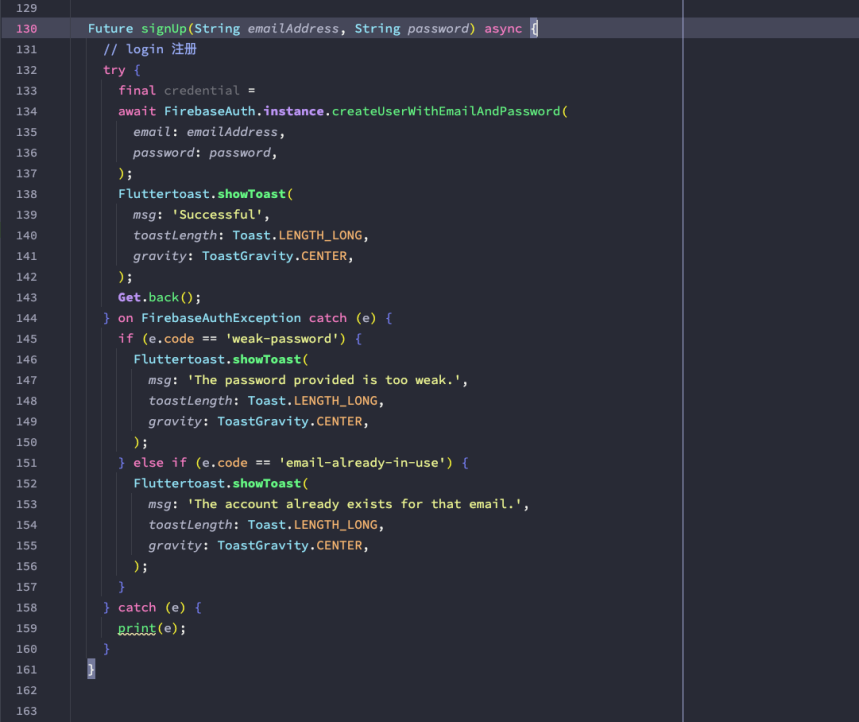
Program connecting to Google Firebase configuration:



Using Firebase login method:



Using Firebase registration method:



#### Run on a real device

<https://docs.flutter.cn/get-started/install/macos/mobile-ios>