FLUTTER
PARA INICIANTES

GUIA PRÁTICO
COM PASSO A PASSO

DESENVOLVIMENTO FLUTTER

TRABALHO DE CONCLUSÃO DE CURSO:

APOSTILA SOBRE APLICAÇÃO MOBILE EM FLUTTER

Cubatão - SP

2020

AUTORES

CTII 418

Camila de Souza Dantas Rodrigues	1790749
Elienai Ramos dos Santos	1790609
Gabriel Ribeiro Antunes	1790129
Geovanna da Silva Melone	1790161
João Vitor Souza Rocha	1790781
Pedro Malteze de Macedo	1790498

CTII 448

Caroline Ribeiro dos Santos	1790137
Emanuelle Costa Moura Jorge	1790641
Laura Moreira Cesar Souza Moço	1790013
Milena da Silva Matsuda Valência	1790111

Professor Orientador

Maurício Neves Asenjo

Disciplina

Projeto de Sistemas

ÍNDICE

DESENVOLVIMENTO FLUTTER

APRESENTAÇÃO DA APOSTILA: HISTÓRICO DA FRAMEWORK

INTRODUÇÃO

Flutter é um conjunto de métodos do Google, que utiliza a linguagem Dart, para desenvolvimento de interface de usuário. As aplicações são dinamicamente compiladas para mobile, web, desktop apoiado em um único código-base, ou seja, escrevendo o código apenas uma vez, o aplicativo pode ser rodado em mais de quatro plataformas diferentes.

OBJETIVO

Nesta relação iremos apresentar pontos consideráveis desse kit de desenvolvimento, incluindo seu histórico e suas principais características. Tendo como objetivo esclarecer sua origem e suas vantagens e desvantagens na área de programação.

ORIGEM

Em 2014, o projeto foi iniciado com o codinome "Sky", com o propósito de encontrar um melhor caminho de construir interfaces para o mobile. A proposta, em 2015, foi apresentada durante a Dart Developer Summit, onde foi possível ver o código Dart sendo executado em um aparelho Android. A partir desse momento, o programa foi exibido como Flutter. Após um ano (2016), no mesmo evento, Flutter foi descrito como "Uma melhor maneira de desenvolvimento mobile".
O Flutter teve sua estreia em um importante evento, no Google IO 2017, onde em uma sessão de "live coding", o aplicativo elaborado foi integrado, com acesso à câmera, ao Firebase. No Google IO 2018, o kit de desenvolvimento já teve um destaque maior, após ultrapassar as expectativas, contou com 3 palestras excepcionais, atendendo a tópicos como: gerenciamento de estado, uso do Material Design e novamente outra sessão "live coding" de um aplicativo, acessando o Firebase.
Em dezembro de 2018, foi lançada a versão 1.0 e o principal destaque foi para o projeto Hummingbird, que após um tempo tornou-se o Flutter para Web. Toda a configuração da versão 1.0, utilizando o Flutter, foi feita em um aplicativo nativo para MacOS.

PRINCIPAIS CARACTERÍSTICAS

- Aplicativos bem-apresentados. Com o framework é possível que o desenvolvedor tenha a autonomia para mudar cada pixel da tela. Para o iOS, o desenvolvedor pode usar os widgets (componentes de uma aplicação que permite o usuário interagir com as funcionalidades do aplicativo) da biblioteca Cupertino. Já no Android é possível o uso do Material de Design do Google, que tem diversos widgets disponíveis.

- É ágil. O mecanismo que é utilizado, o Skia 2D, tem o objetivo de focar na velocidade de hardware. Ele é mantido pelo Google, porém é open source, ou seja, pode ser utilizado por outros softwares, como por exemplo: Firefox e Firefox OS. A estrutura do Flutter foi criada com o objetivo de ter suporte aos gráficos jank-free na rapidez do aparelho. Os aplicativos criados com Flutter são desenvolvidos na linguagem Dart, os quais podem ser compilados no Android ou iOS com processadores ARM de 32 ou 64 bits, tornando-os assim o Flutter ágil.

- O Flutter é vantajoso. Ele possibilita que o aplicativo seja operado num smartphone ou no emulador enquanto o desenvolvedor o programa. O stateful hot reload nada mais é que a atualização automática e rápida no aparelho quando o desenvolvedor salva um arquivo no projeto, tornando assim o processo de desenvolvimento mais proficiente.

- É aberto. Como dito no tópico anterior, o Flutter é open source com licença BSD-style, que inclui colaborações mundiais de diversos desenvolvedores.

CONCLUSÃO

Conclui-se, portanto, que desde seu lançamento o Flutter vem se aprimorando para manter seu lema de ser a melhor maneira de desenvolvimento mobile, para isso ampliasse no conjunto de métodos do Google com a expansão da linguagem Dart, se tornando cada dia mais didático, diversificado, ágil e flexível tanto para o operador quanto para o programador.

IDES: PRINCIPAIS IDES QUE COMPACTUAM COM FLUTTER

Podemos construir aplicativos com Flutter usando qualquer editor de texto combinado com ferramentas de linha de comando. No entanto, é recomendado o uso de um dos seguintes plug-ins de editor para uma experiência ainda melhor, pois eles fornecem IntelliSense (completar automaticamente códigos), destaque de sintaxe, assistências de edição de widget, suporte para execução e depuração e muito mais.

São eles IntelliJ, Android Studio e VS Code, porém focaremos nos dois últimos principais no quesito desenvolvimento de aplicativos mobile.

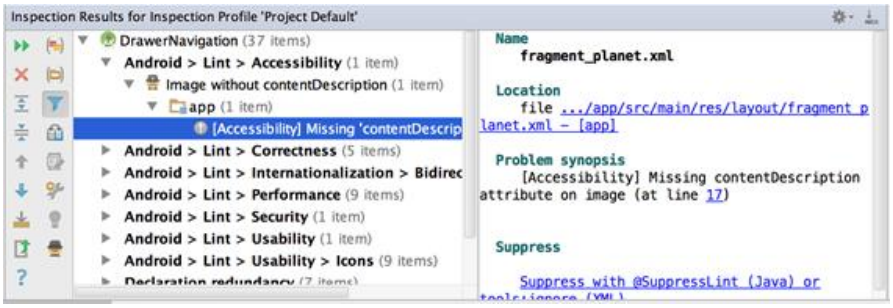
ANDROID STUDIO.

Baseado no software IntelliJ IDEA de JetBrains, o **Android Studio** foi feito especificamente para o desenvolvimento para Android. Está disponível para download em Windows, Mac OS X e Linux, e substituiu Eclipse Android Development Tools (ADT) como a IDE primária do Google de desenvolvimento nativo para Android. Funciona com as linguagens Java, Kotlin e C++.

Algumas características sobre o Android Studio:

1. Inspeções com Lint

Sempre que você compila um programa, o Android Studio executa automaticamente inspeções de lint configuradas e outras inspeções do ambiente de desenvolvimento integrado para ajudar a identificar e corrigir problemas na qualidade estrutural do código. A ferramenta de lint verifica os arquivos de origem do projeto Android para localizar possíveis bugs e melhorias de otimização em relação a critérios de precisão, segurança, desempenho, usabilidade, acessibilidade e internacionalização.

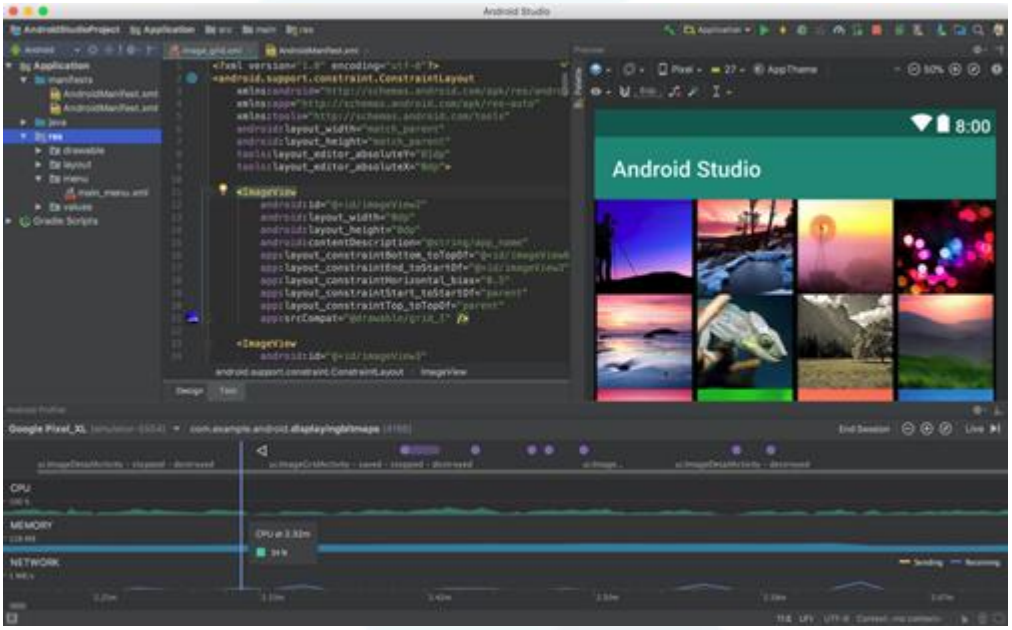


Recurso de inspeção Lint.

Fonte: site Developers Android, 27 de setembro de 2020.

2. Emulador

Um emulador rápido com inúmeros recursos e um ambiente unificado que possibilita o desenvolvimento para todos os dispositivos Android;



Captura de tela do Android Studio 1.2.1.1, junto com o emulador ativo.

Fonte: Wikipedia, 29 de setembro de 2020.

3. Preenchimento Automático de Código

O Android Studio tem três tipos de preenchimento automático de código, que podem ser acessados usando atalhos de teclado.

Tipo	Descrição	Windows e Linux	Mac
Preenchimento básico	Exibe sugestões básicas para variáveis, tipos, métodos, expressões, entre outros. Se você chamar o preenchimento básico duas vezes seguidas, verá mais resultados, incluindo membros privados e membros estáticos não importados.	Control + Espaço	Control + Espaço
Preenchimento inteligente	Exibe opções relevantes de acordo com o contexto. O preenchimento inteligente detecta os tipos e os fluxos de dados esperados. Se você chamar o preenchimento inteligente duas vezes seguidas, verá mais resultados, incluindo cadeias.	Control + Shift + Espaço	Control + Shift + Espaço
Preenchimento de declaração	Preenche a declaração atual, adicionando parênteses, colchetes e chaves ausentes, formatação etc.	Control + Shift + Enter	Shift + Command + Enter

Tipos de preenchimento automático de código.

Fonte: site Developers Android, 27 de setembro de 2020.

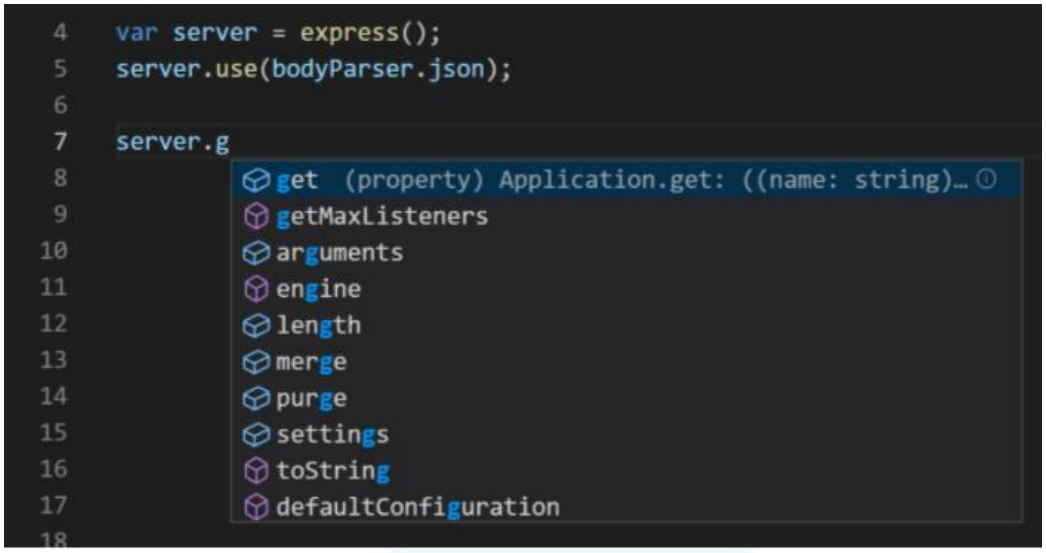
VISUAL STUDIO CODE

Visual Studio Code

O Visual Studio Code é um editor de código-fonte leve, mas poderoso, que é executado em sua área de trabalho disponível para Windows, macOS e Linux. Ele vem com suporte integrado para JavaScript, TypeScript e Node.js e tem um rico ecossistema de extensões para outras linguagens (como C ++, C #, Java, Python, PHP, Go e Flutter) e tempos de execução (como .NET e Unity).

Principais características:

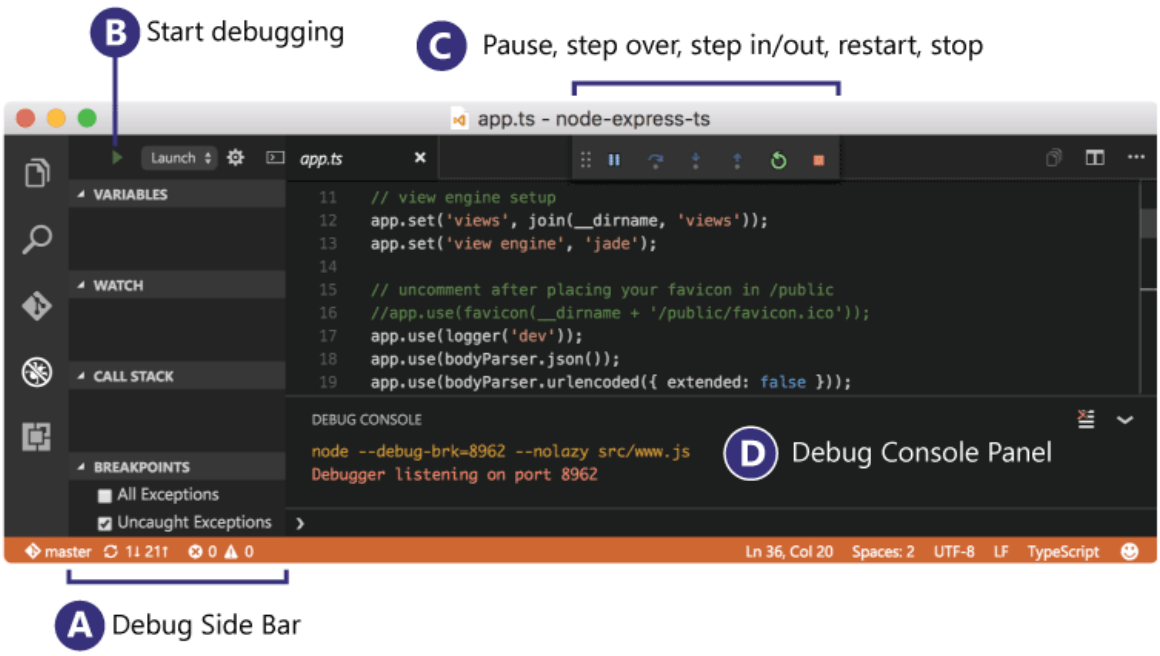
1. IntelliSense



Ferramenta de preenchimento automático IntelliSense.
Fonte: site Visual Studio Code, 27 de setembro de 2020.

O VS Code possui uma ferramenta chamada IntelliSense que permite ao usuário aprender mais sobre o código que está usando, a manter o acompanhamento dos parâmetros que está digitando e a adicionar chamadas a métodos e propriedades pressionando apenas algumas teclas.

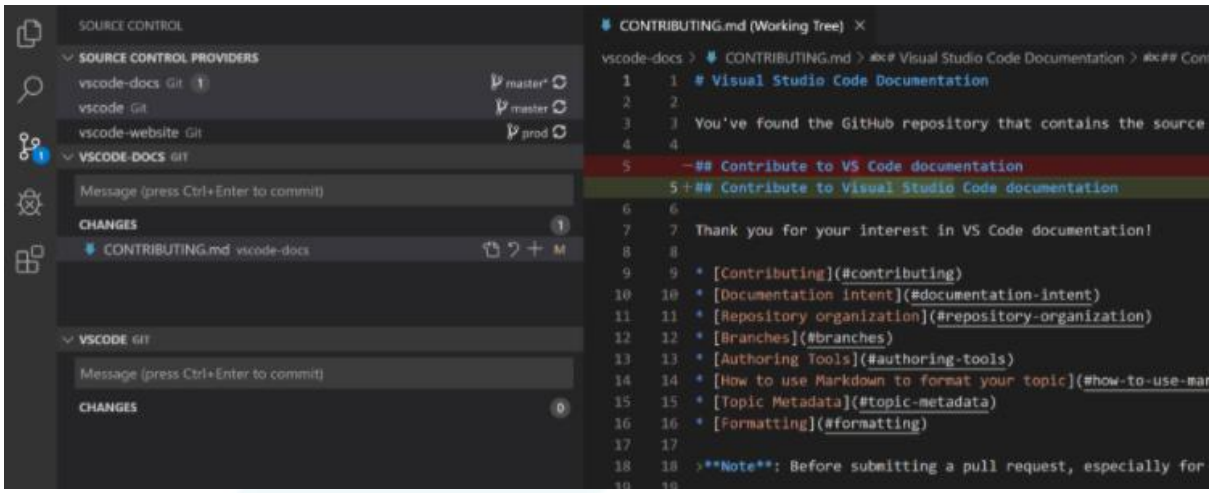
2. Debugging



Recurso Debugging no Visual Studio Code.
Fonte: site Visual Studio Code, 27 de setembro de 2020.

Um dos principais recursos do Visual Studio Code é seu excelente suporte à depuração. O depurador embutido do VS Code ajuda a acelerar seu loop de edição, compilação e depuração.

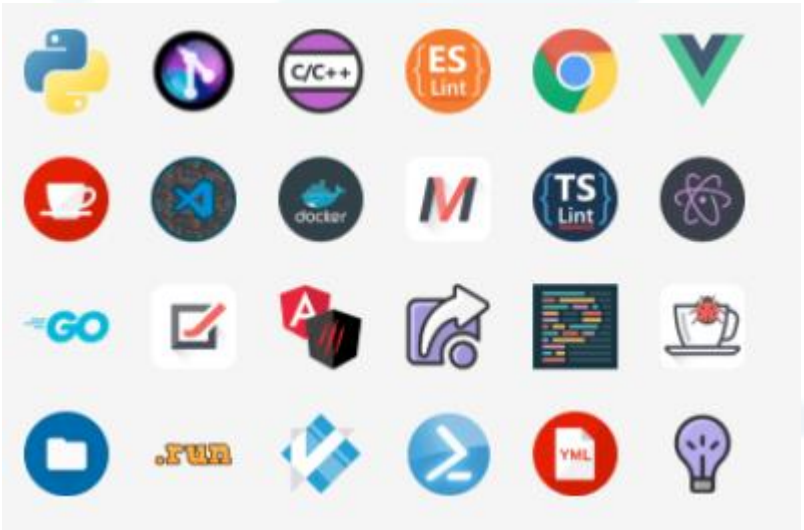
3. Comandos Git integrados



Comandos Git.
Fonte: site Visual Studio Code, 27 de setembro de 2020.

Trabalhar com Git e outros provedores de SCM nunca foi tão fácil. Com o VS Code podemos revisar diffs, prepare arquivos e faça commits direto do editor.

4. Extensível e personalizável



Fonte: site Visual Studio Code, 27 de setembro de 2020.

Quer ainda mais recursos? Instale extensões para adicionar novos idiomas, temas, depuradores e para se conectar a serviços adicionais. As extensões são executadas em processos separados, garantindo que não deixem seu editor lento

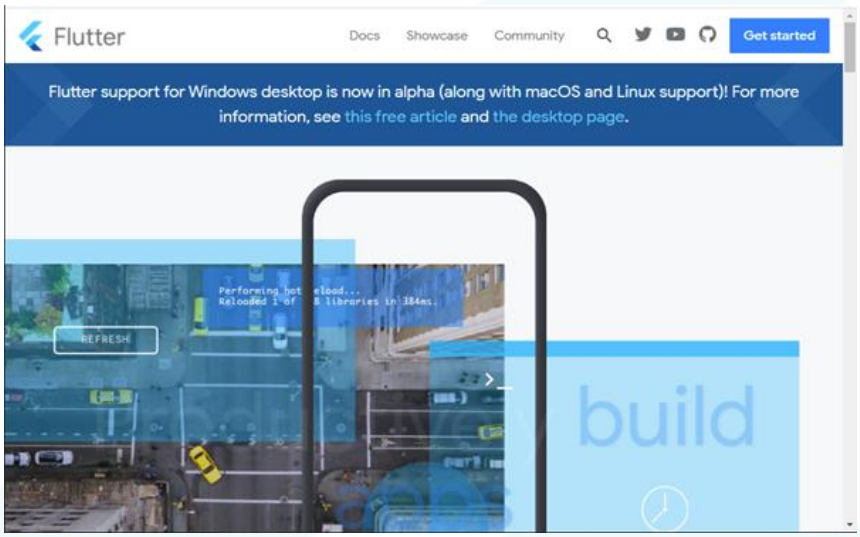
NOSSA DECISÃO

Pesquisando sobre essas duas IDEs mais famosas chegamos a conclusão de usar o Android Studio para a realização dos aplicativos.

ANDROID STUDIO E FLUTTER

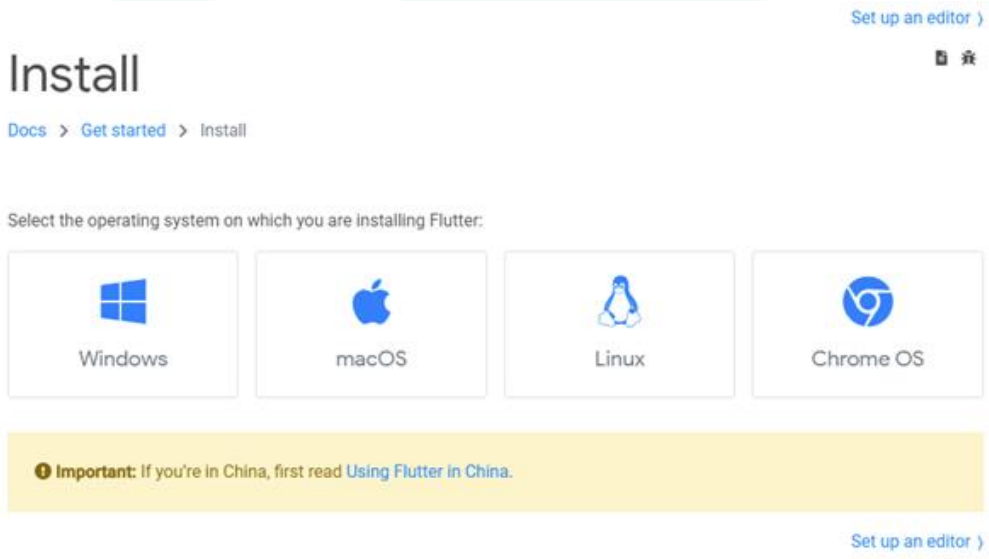
FAZENDO O DOWNLOAD DA IDE E DO FRAMEWORK

1. Visualização dos Requisitos de Instalação.



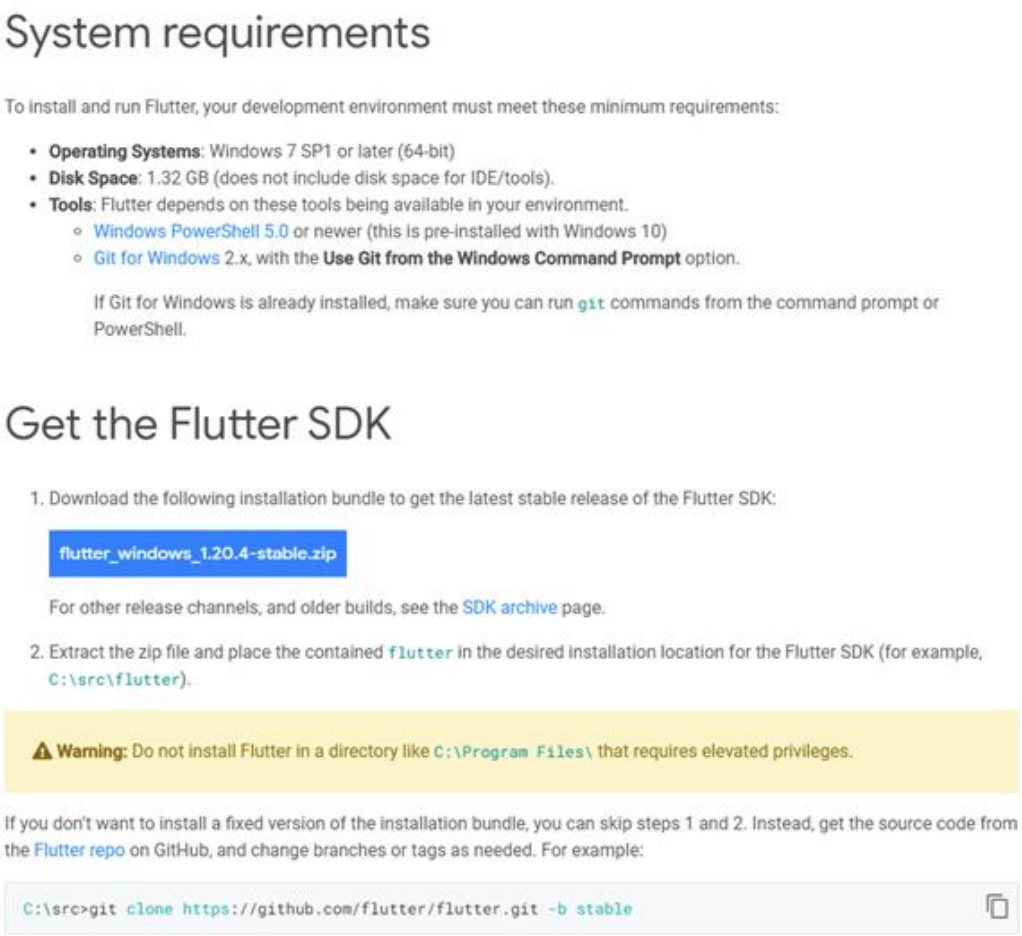
Printscreen de tela feito por Gabriel Ribeiro Antunes.

Para encontrar os links de instalação e visualizar os requisitos do Flutter, acesse “flutter.dev” e clique em “Get Started”.



Printscreen de tela feito por Gabriel Ribeiro Antunes.

Selecione o Sistema Operacional desejado.



System requirements

To install and run Flutter, your development environment must meet these minimum requirements:

- **Operating Systems**: Windows 7 SP1 or later (64-bit)
- **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.

Get the Flutter SDK

1. Download the following installation bundle to get the latest stable release of the Flutter SDK:

   flutter_windows_1.20.4-stable.zip

   For other release channels, and older builds, see the SDK archive page.

2. Extract the zip file and place the contained `flutter` in the desired installation location for the Flutter SDK (for example, `C:\src\flutter`).

⚠ **Warning:** Do not install Flutter in a directory like `C:\Program Files\` that requires elevated privileges.

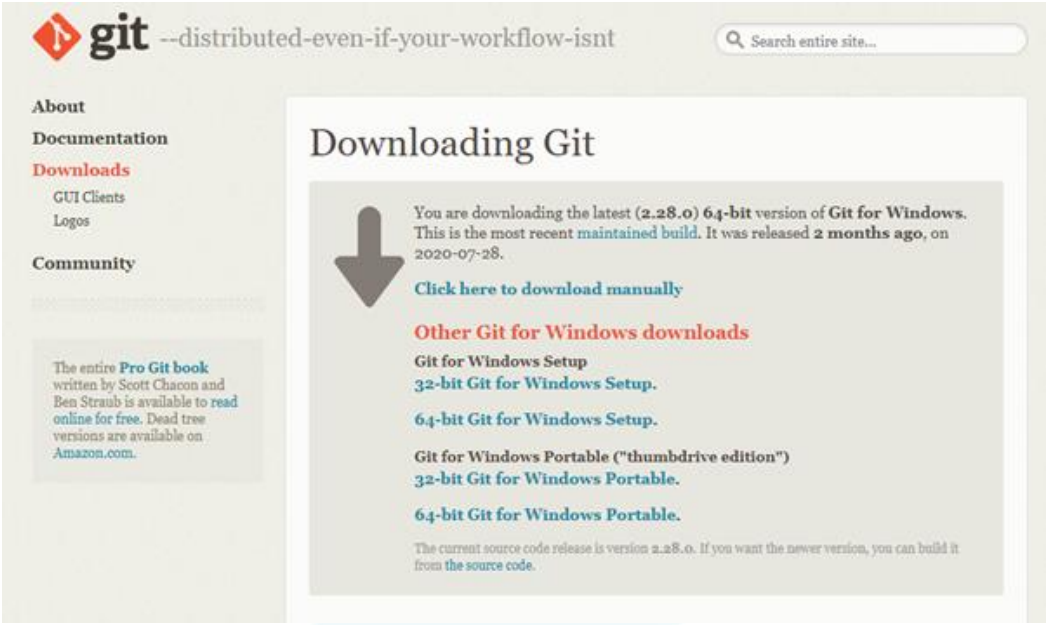
If you don't want to install a fixed version of the installation bundle, you can skip steps 1 and 2. Instead, get the source code from the Flutter repo on GitHub, and change branches or tags as needed. For example:

```
C:\src>git clone https://github.com/flutter/flutter.git -b stable
```

Printscreen de tela feito por Gabriel Ribeiro Antunes.

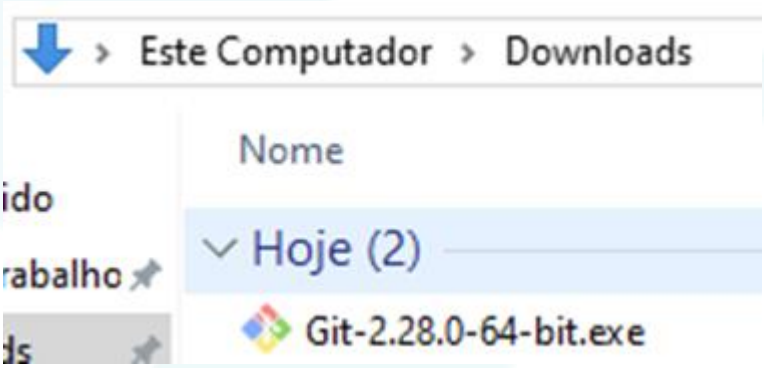
Para o funcionamento do Flutter, é necessário ter dois softwares: PowerShell e Git. O PowerShell não é necessário para instalação em máquinas com Windows 10. Para instalar o Git, clique em “Git for Windows” em azul.

2. Instalação do Git.



Printscreen de tela feito por Gabriel Ribeiro Antunes.

Ao entrar no site de downloads do Git, vá em “Click here to download manually” e aguarde o download terminar.

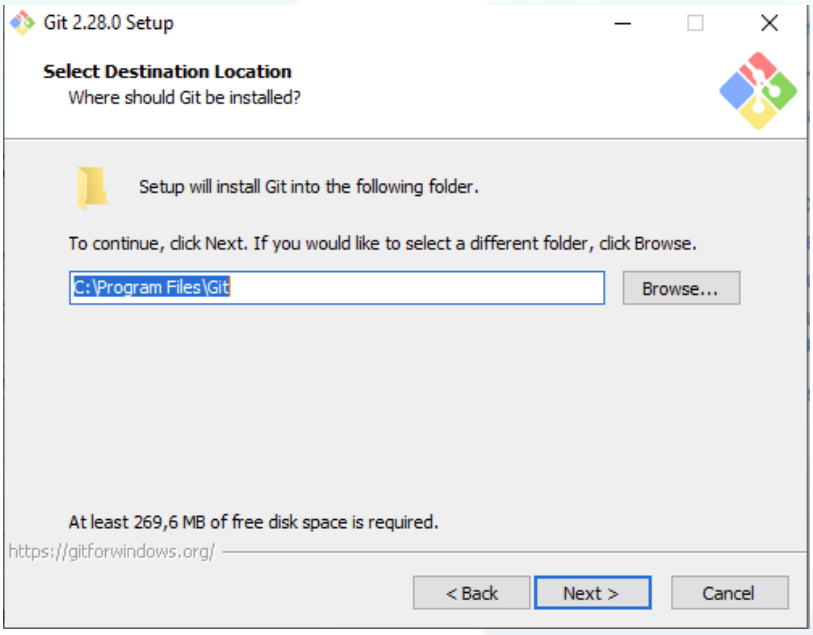


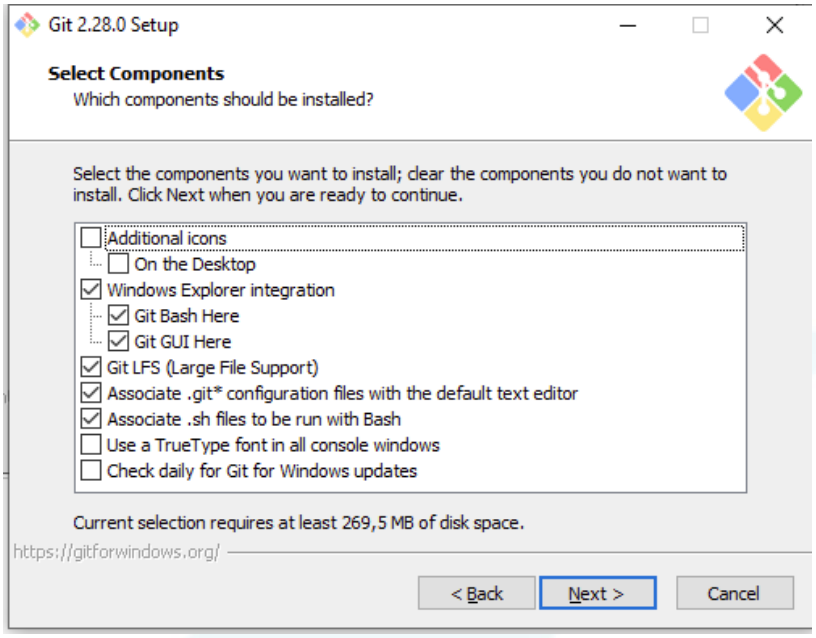
Printscreen de tela feito por Gabriel Ribeiro Antunes.

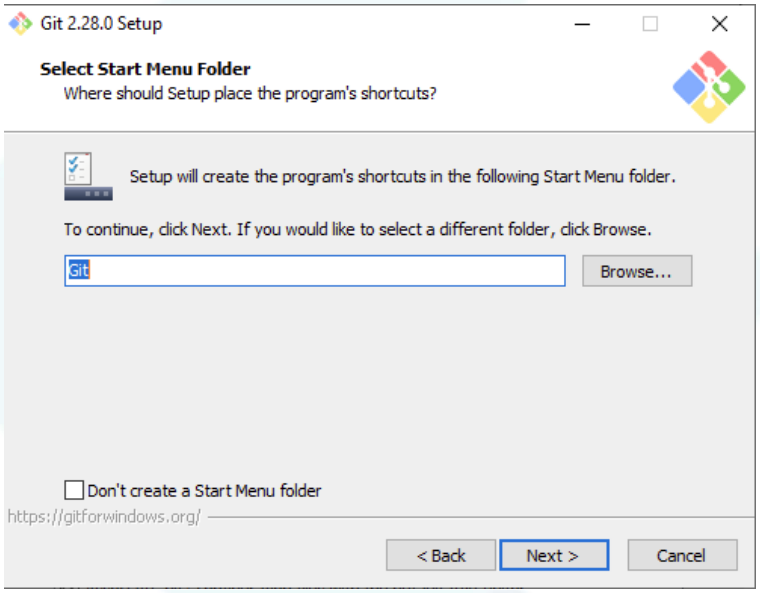
Ao finalizar o download, execute o arquivo em seu local baixado.

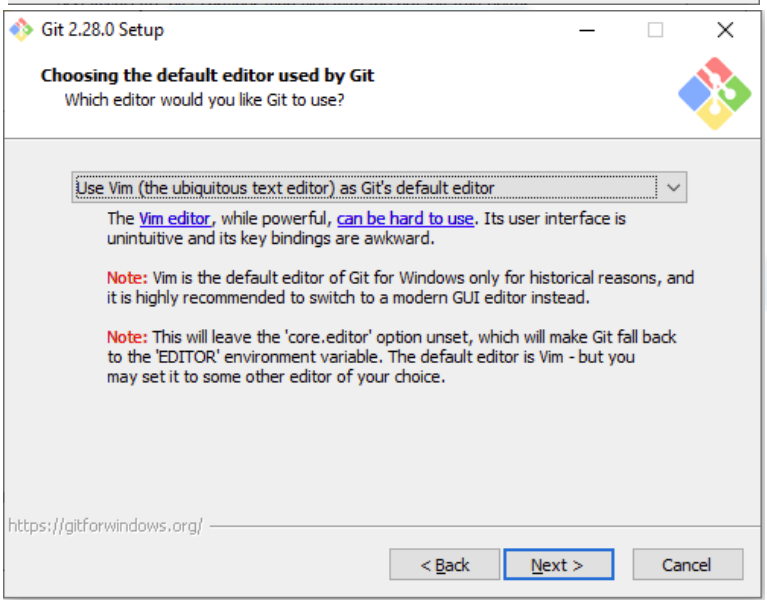
Printscreen de tela feito por Gabriel Ribeiro Antunes.

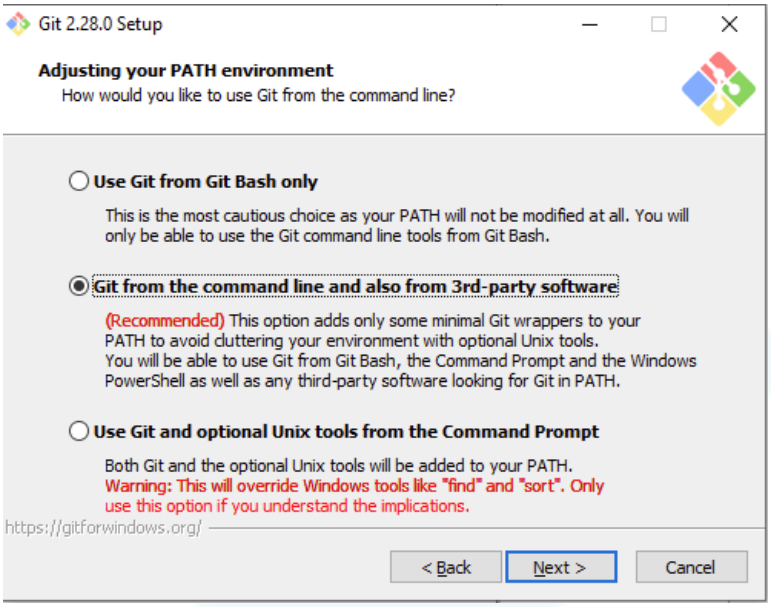
Para as próximas telas é recomendado clicar apenas em “Next >” a menos que o usuário visualize algo que deseje alterar.

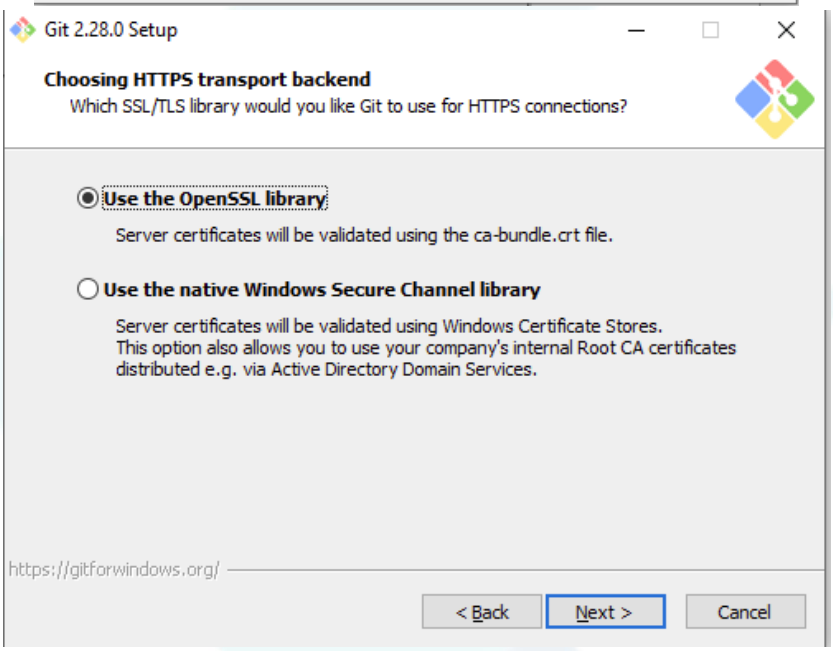


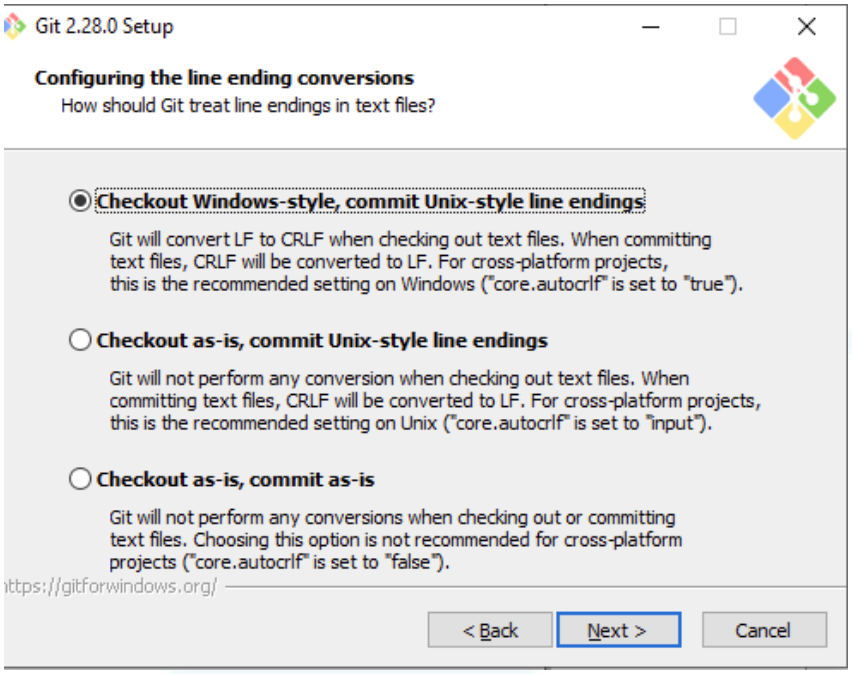

Git 2.28.0 Setup
Select Components
Which components should be installed?
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
Additional icons
On the Desktop
Windows Explorer integration
Git Bash Here
Git GUI Here
Git LFS (Large File Support)
Associate .git* configuration files with the default text editor
Associate .sh files to be run with Bash
Use a TrueType font in all console windows
Check daily for Git for Windows updates
Current selection requires at least 269,5 MB of disk space.
https://gitforwindows.org/
< Back
Next >
Cancel

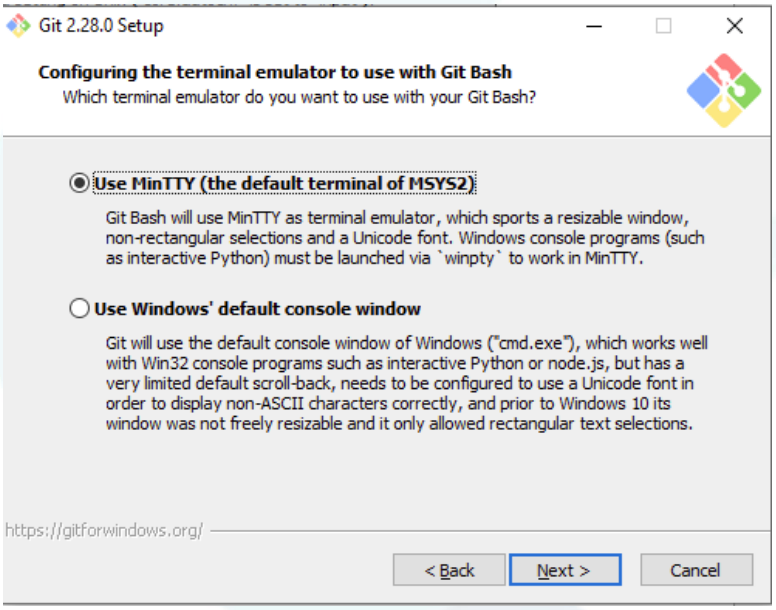

Git 2.28.0 Setup
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
Git
Browse...
Don't create a Start Menu folder
https://gitforwindows.org/
< Back
Next >
Cancel

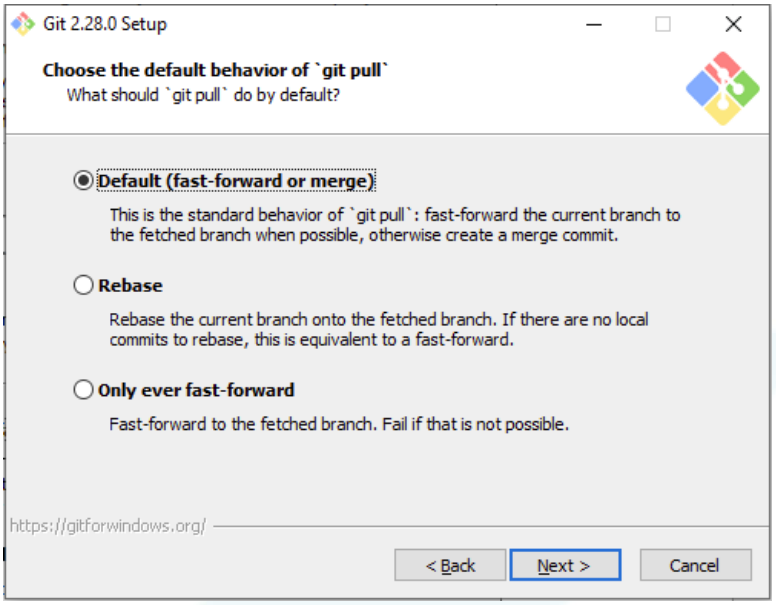

Git 2.28.0 Setup
Choosing the default editor used by Git
Which editor would you like Git to use?
Use Vim (the ubiquitous text editor) as Git's default editor
The Vim editor, while powerful, can be hard to use. Its user interface is unintuitive and its key bindings are awkward.
Note: Vim is the default editor of Git for Windows only for historical reasons, and it is highly recommended to switch to a modern GUI editor instead.
Note: This will leave the 'core.editor' option unset, which will make Git fall back to the 'EDITOR' environment variable. The default editor is Vim - but you may set it to some other editor of your choice.
https://gitforwindows.org/
< Back
Next >
Cancel

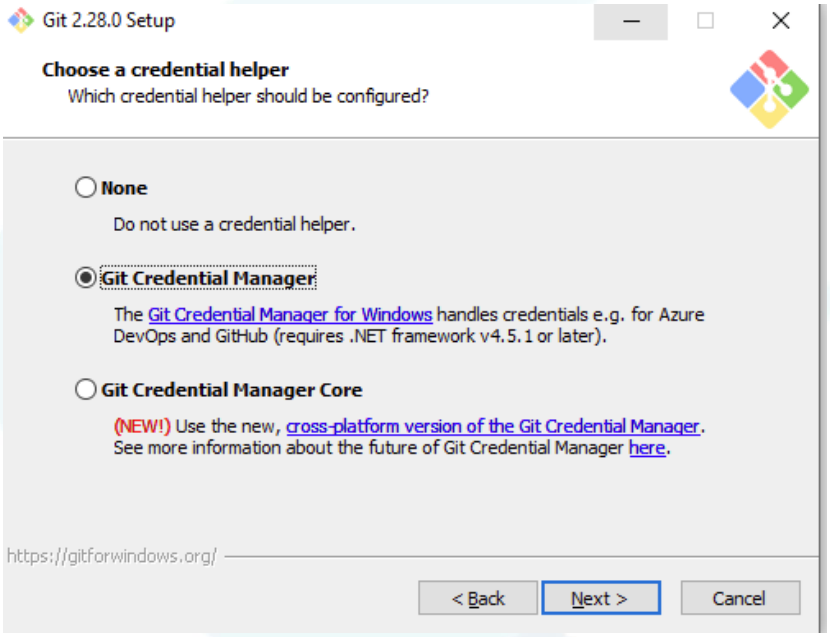

Git 2.28.0 Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.
Git from the command line and also from 3rd-party software
(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools.
You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.
Use Git and optional Unix tools from the Command Prompt
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.
https://gitforwindows.org/
< Back
Next >
Cancel

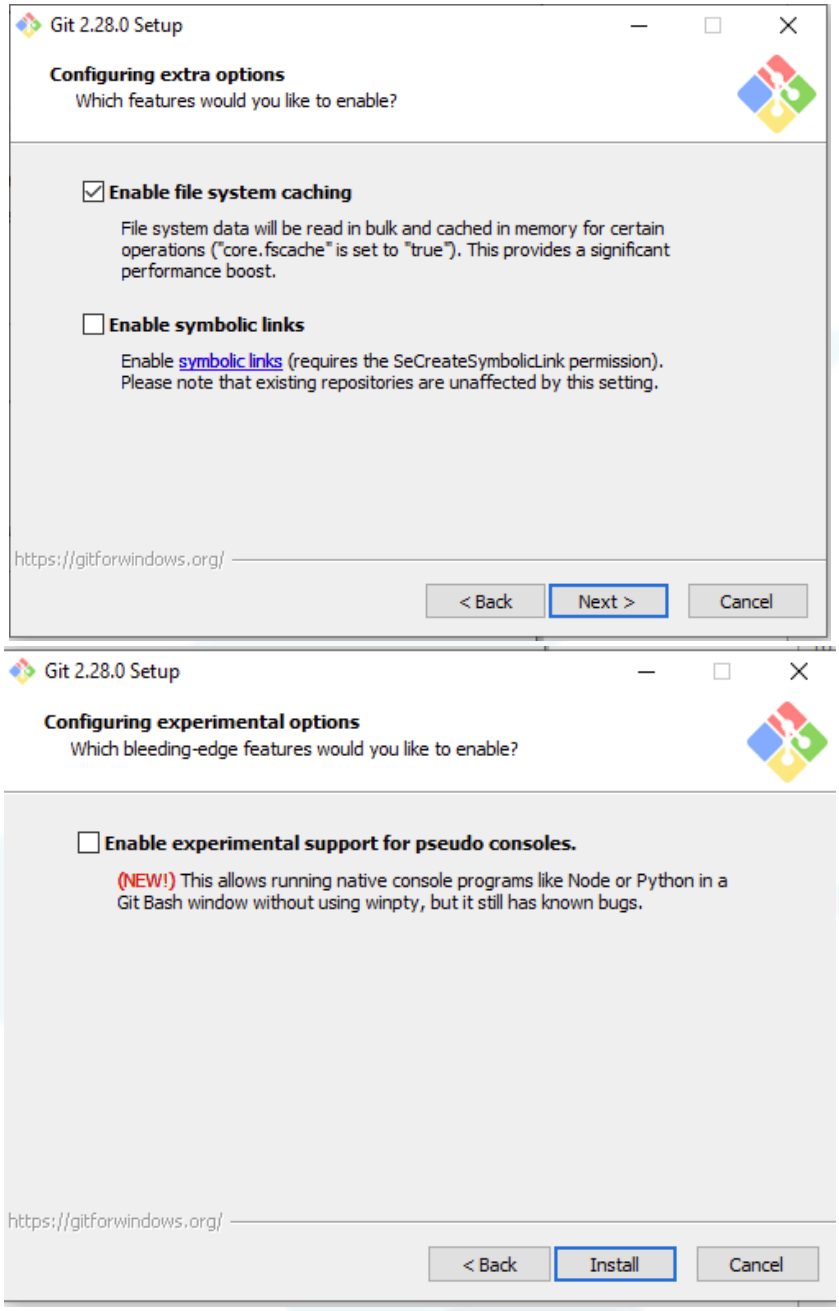

Git 2.28.0 Setup
Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?
Use the OpenSSL library
Server certificates will be validated using the ca-bundle.crt file.
Use the native Windows Secure Channel library
Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.
https://gitforwindows.org/
< Back
Next >
Cancel


Git 2.28.0 Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").
Checkout as-is, commit Unix-style line endings
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").
Checkout as-is, commit as-is
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").
https://gitforwindows.org/
< Back
Next >
Cancel


Git 2.28.0 Setup
Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?
Use MinTTY (the default terminal of MSYS2)
Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.
Use Windows' default console window
Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.
https://gitforwindows.org/
< Back
Next >
Cancel

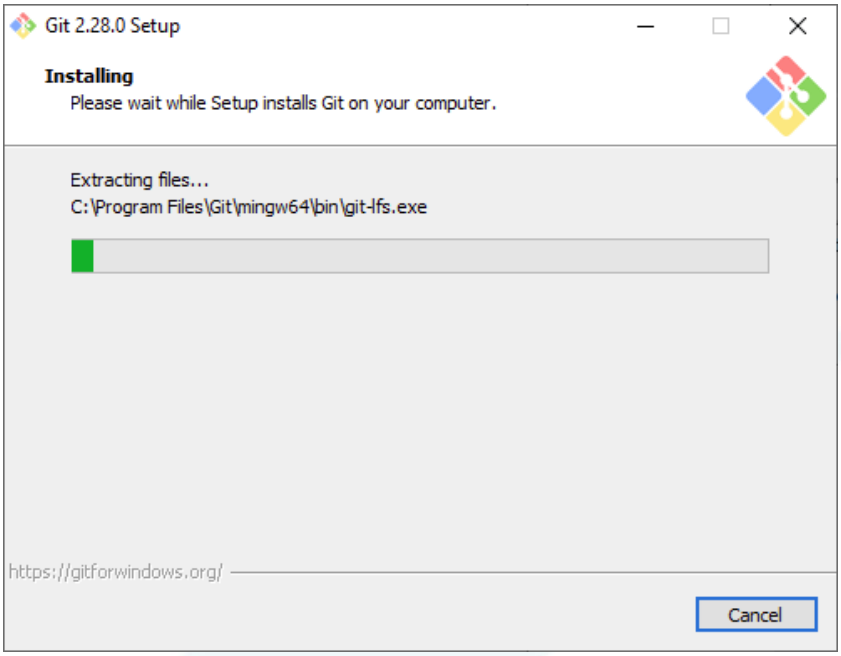

Git 2.28.0 Setup
Choose the default behavior of `git pull`
What should `git pull` do by default?
Default (fast-forward or merge)
This is the standard behavior of `git pull`: fast-forward the current branch to the fetched branch when possible, otherwise create a merge commit.
Rebase
Rebase the current branch onto the fetched branch. If there are no local commits to rebase, this is equivalent to a fast-forward.
Only ever fast-forward
Fast-forward to the fetched branch. Fail if that is not possible.
https://gitforwindows.org/
< Back
Next >
Cancel


Git 2.28.0 Setup
Choose a credential helper
Which credential helper should be configured?
None
Do not use a credential helper.
Git Credential Manager
The Git Credential Manager for Windows handles credentials e.g. for Azure DevOps and GitHub (requires .NET framework v4.5.1 or later).
Git Credential Manager Core
(NEW!) Use the new, cross-platform version of the Git Credential Manager.
See more information about the future of Git Credential Manager here.
https://gitforwindows.org/
< Back
Next >
Cancel



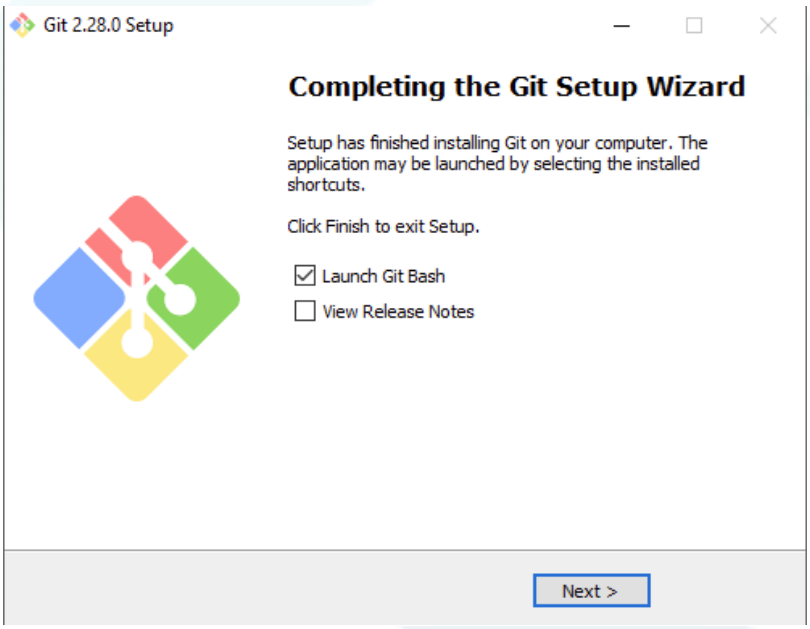
Printscreens de tela feito por Gabriel Ribeiro Antunes.

Nesta tela, clique em “Install”.



Printscreen de tela feito por Gabriel Ribeiro Antunes.

Aguarde o processo de instalação finalizar.



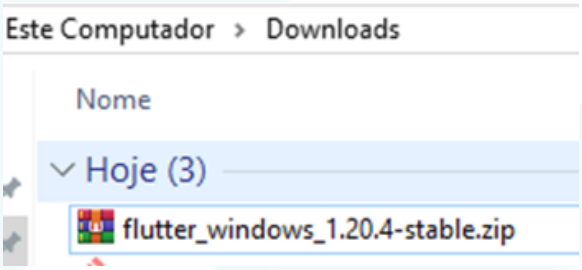
Printscreen de tela feito por Gabriel Ribeiro Antunes.

Está finalizada a instalação do Git. Vá em “Next >” para fechar a tela de instalação.

3. Instalação do Flutter SDK.

Printscreen de tela feito por Gabriel Ribeiro Antunes.

Retornando a página de requisitos de instalação do Flutter, clique em "flutter_windows_1.20.4-stable.zip" e aguarde o download terminar.

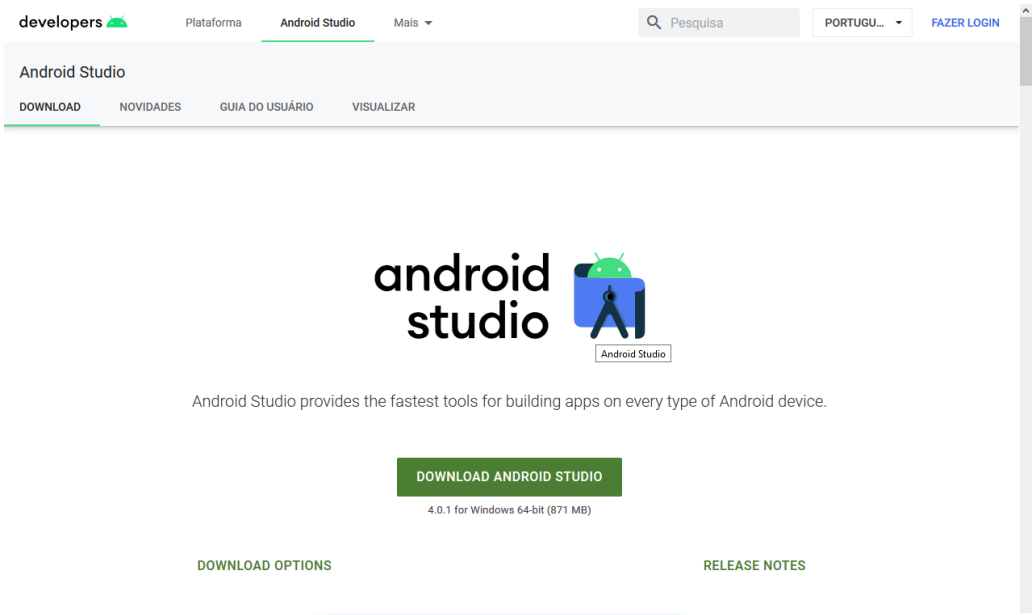


Printscreen de tela feito por Gabriel Ribeiro Antunes.

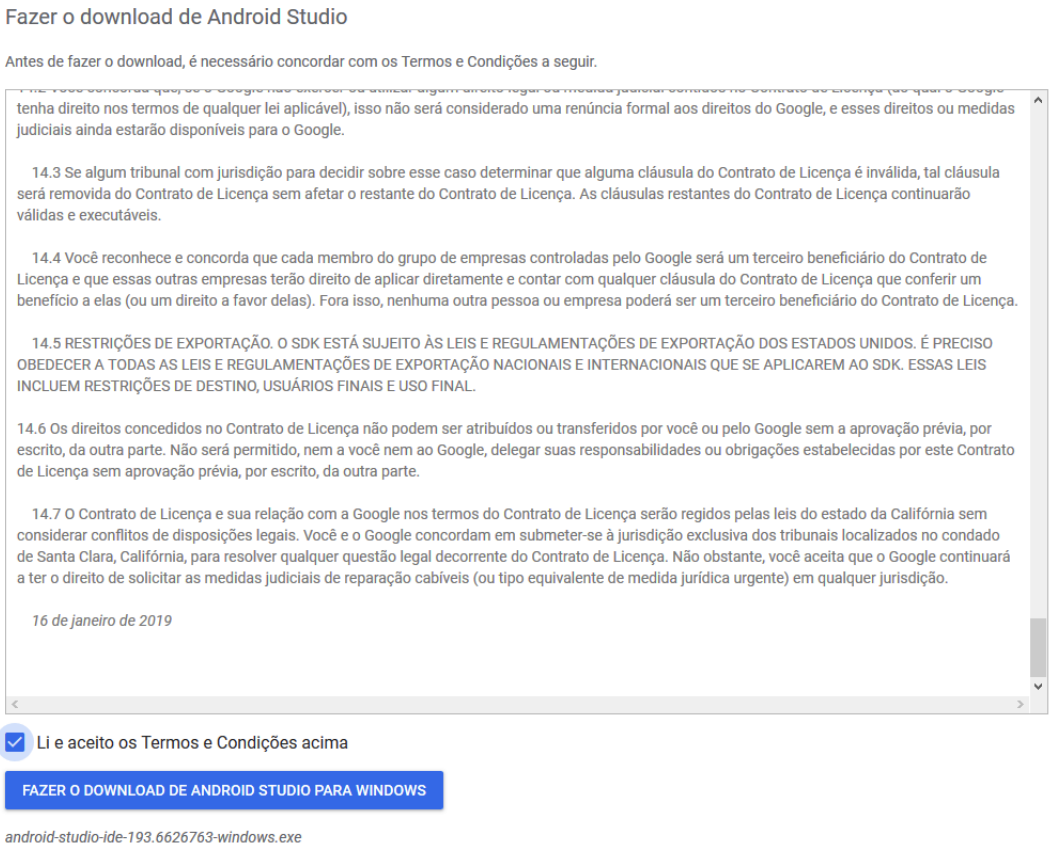
Após finalizar o download, extraia o arquivo em algum local desejado.

4. Instalação Do Android Studio

Entre no site "https://developer.android.com/studio" e clique no botão "Download Android Studio".



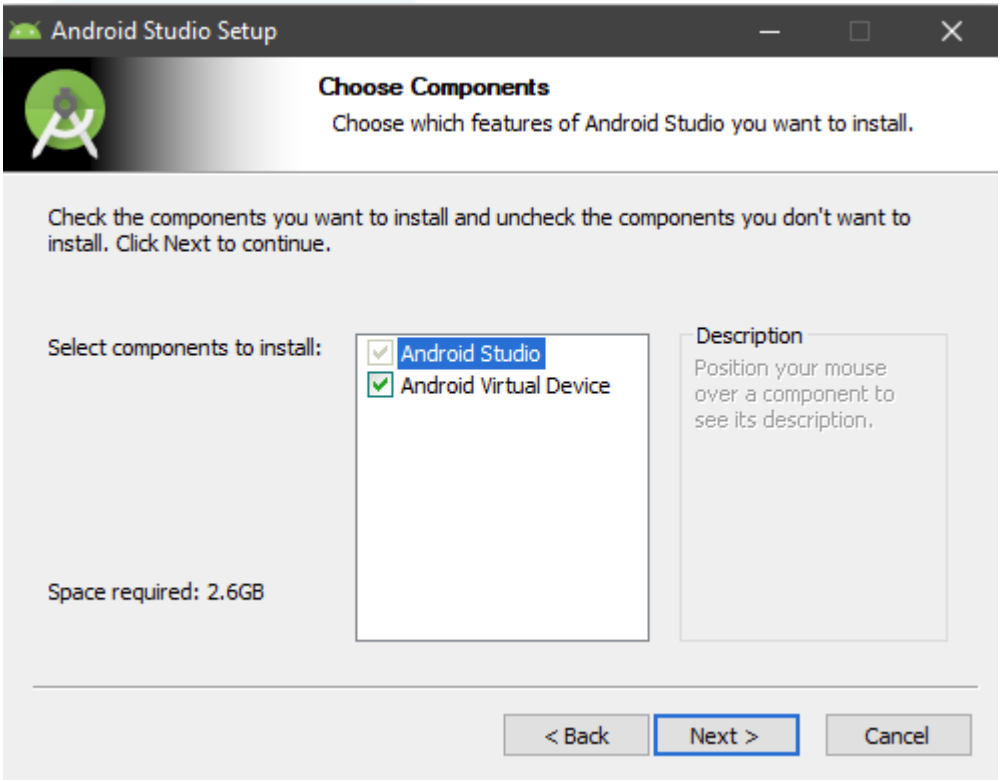
Após aparecer a tela a seguir, selecione “Li e aceito os Termos e Condições acima” e, depois, clique em “fazer o download de Android Studio para Windows”.



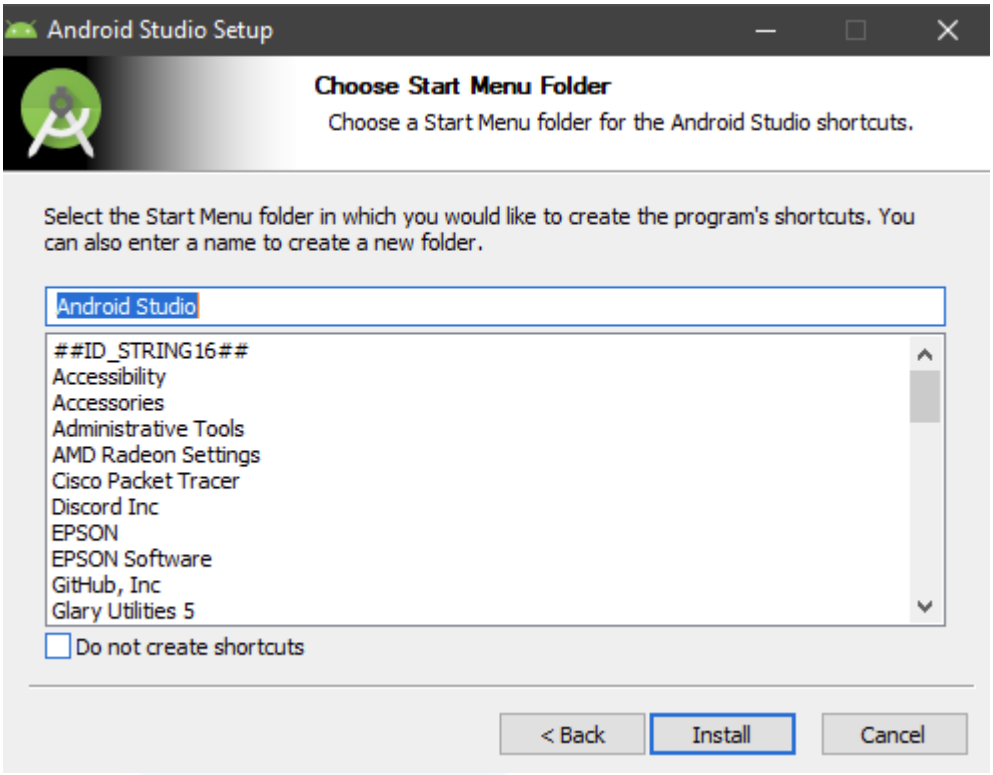
Salve o arquivo no lugar de sua preferência e espere o download terminar. Após finalizado, dê um duplo clique no arquivo .exe.

Clique em “Next >”.

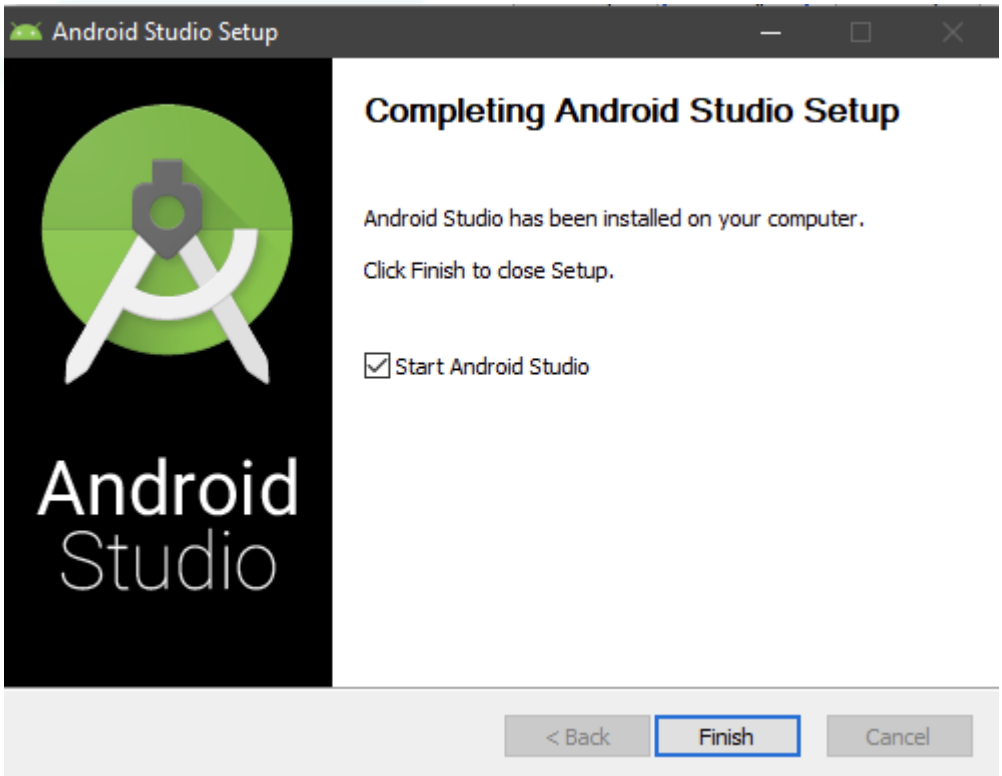
Deixe ambas as opções selecionadas e clique em “Next >”.

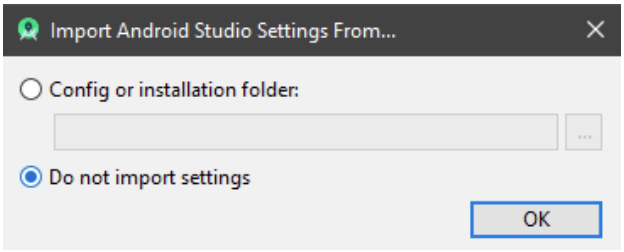


Clique em “Install”.

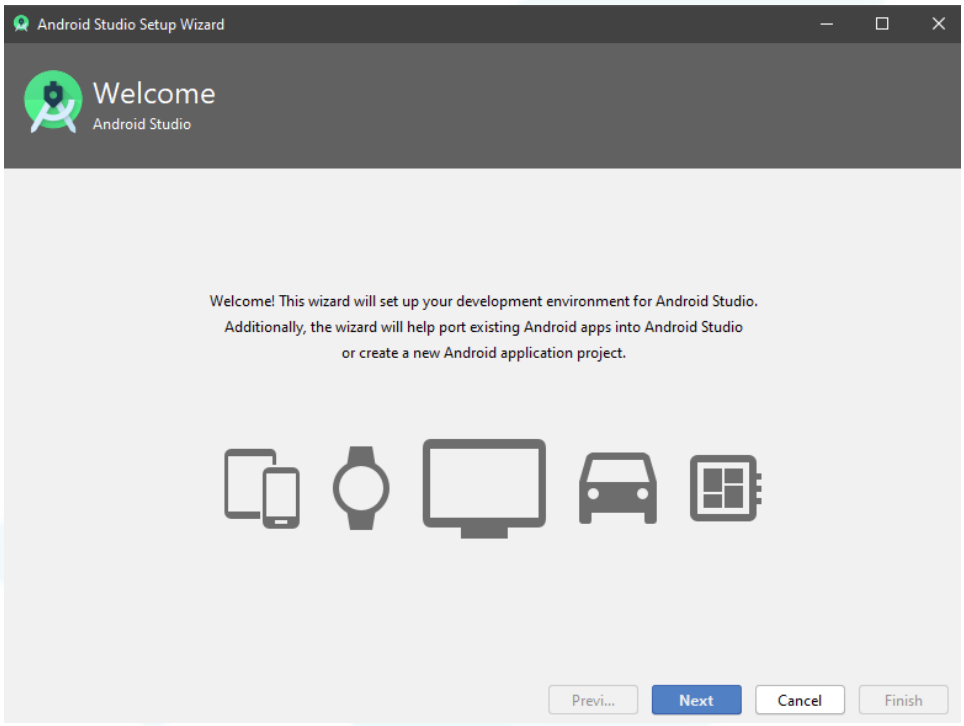


Download finalizado, clique em “Next >” e depois na opção “Finish” para iniciar o Android Studio.

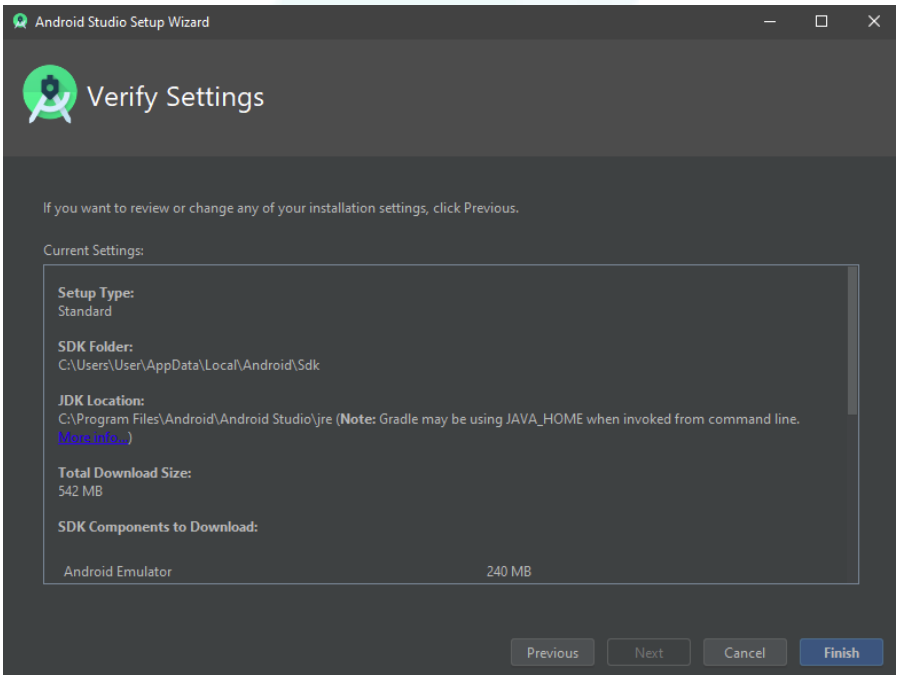




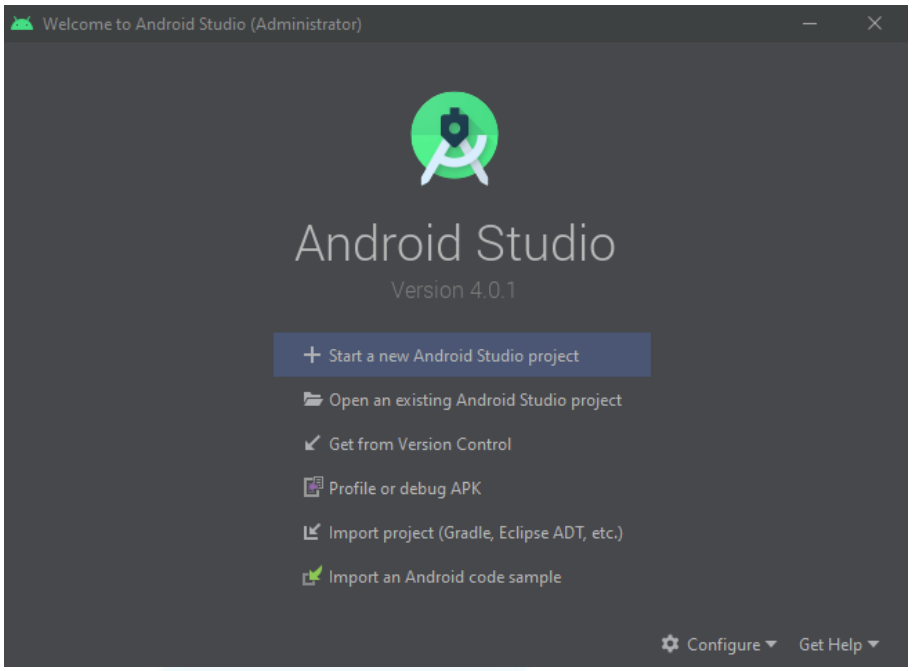
Aparecendo essa tela, clique em “Next >” e depois escolha o tipo de setup de sua preferência e clique em “Next >”. Após isso, escolha o tema de sua preferência e clique em “Next >” novamente.



Nessa página, clique em “Finish” e aguarde o download completo.



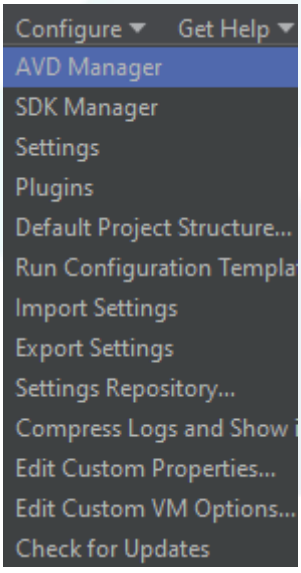
Aguarde aparecer a tela abaixo e, assim, o download foi finalizado.



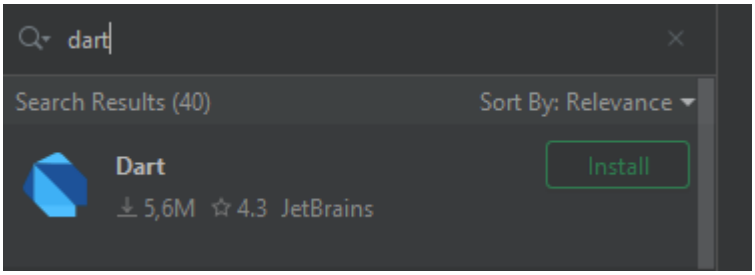
5. Integrando o Flutter ao Android Studio

Instalando as extensões Flutter e Dart

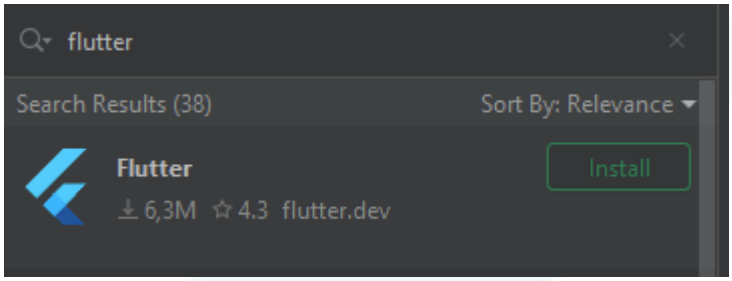
Vá em “Configure” e clique em “Plugins”.



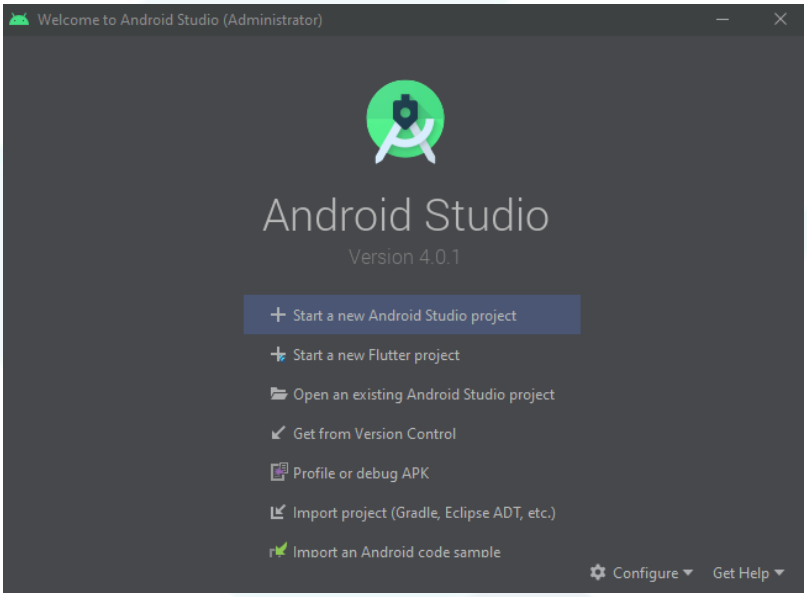
Pesquise por “Dart” e clique em “Install”. Espere finalizar.



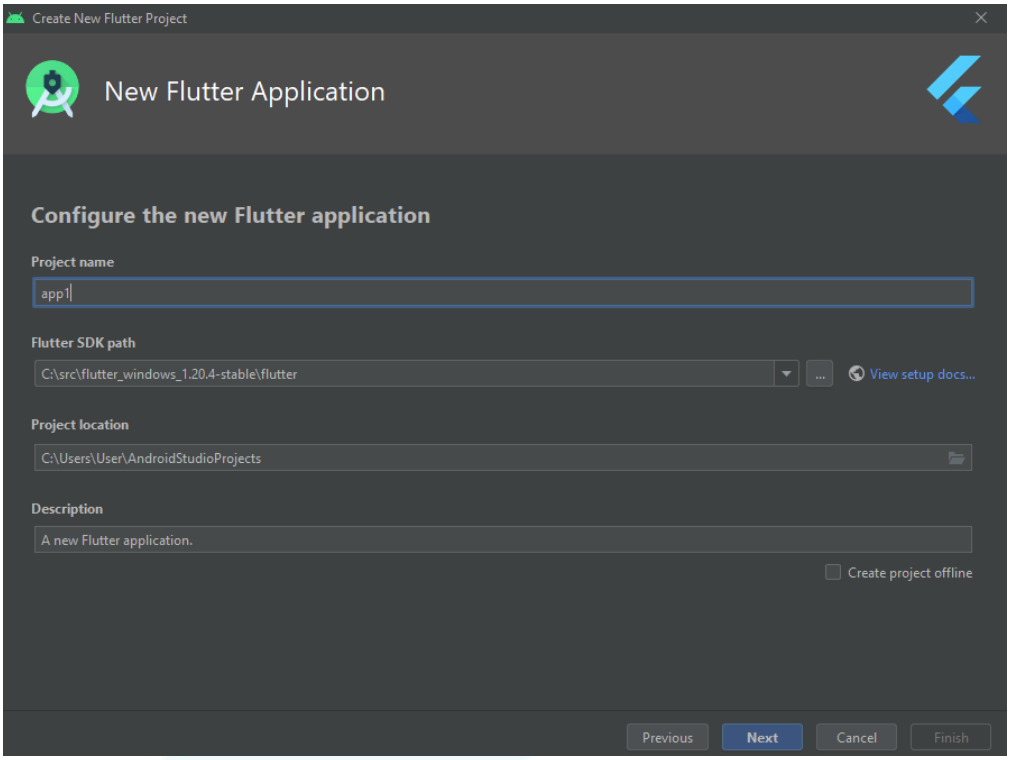
Depois, pesquise por “Flutter” e clique em “Install”. Espere finalizar.



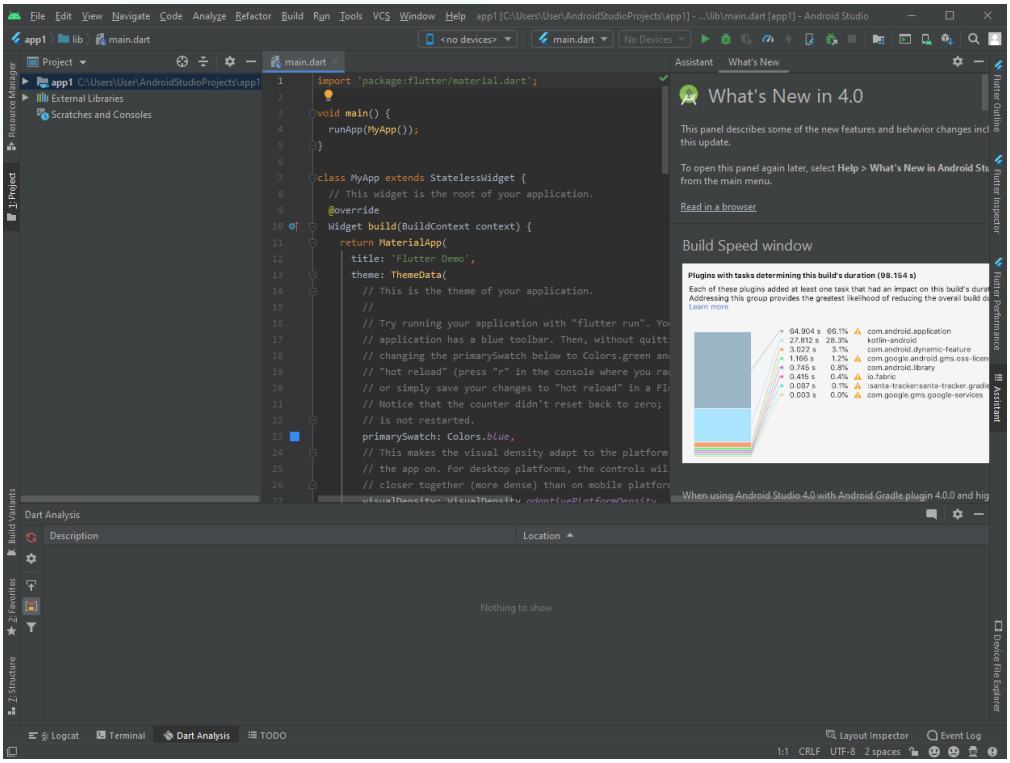
Após ambas instaladas, clique no botão “Restart IDE” da extensão Flutter. O aplicativo irá iniciar e deparar-se nessa página:



Clique em “Start a new Flutter project”. Nomeie a aplicação e depois clique em “Next >” até abrir a página home do projeto.

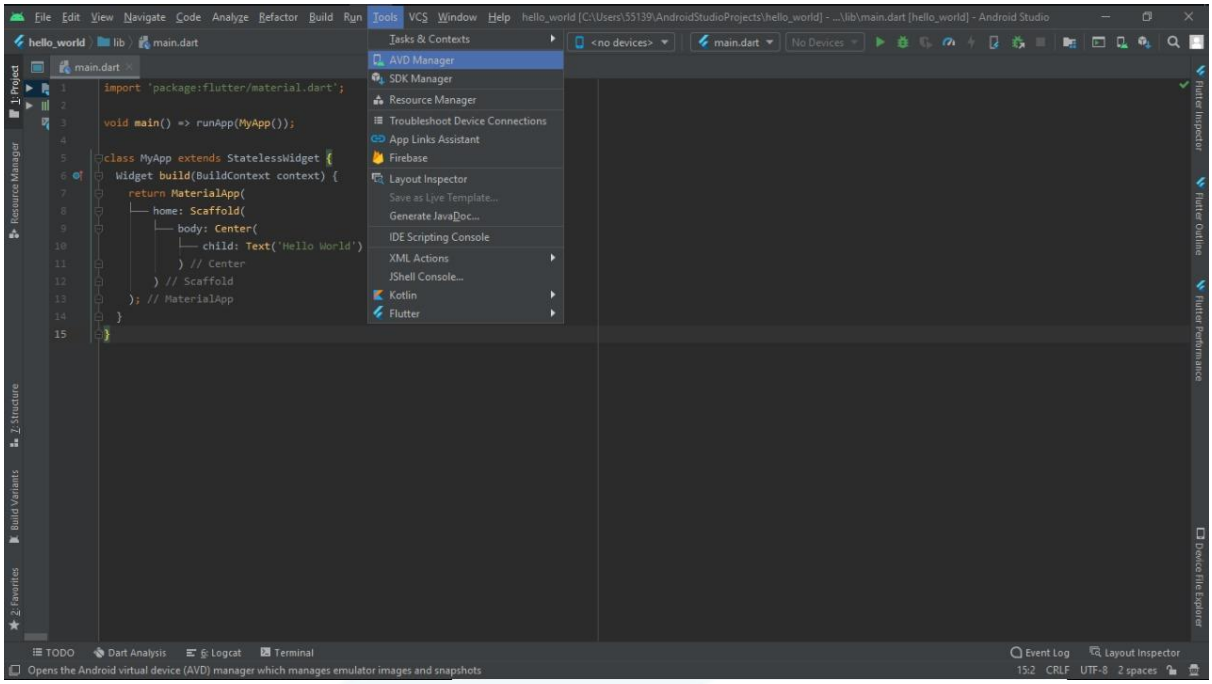


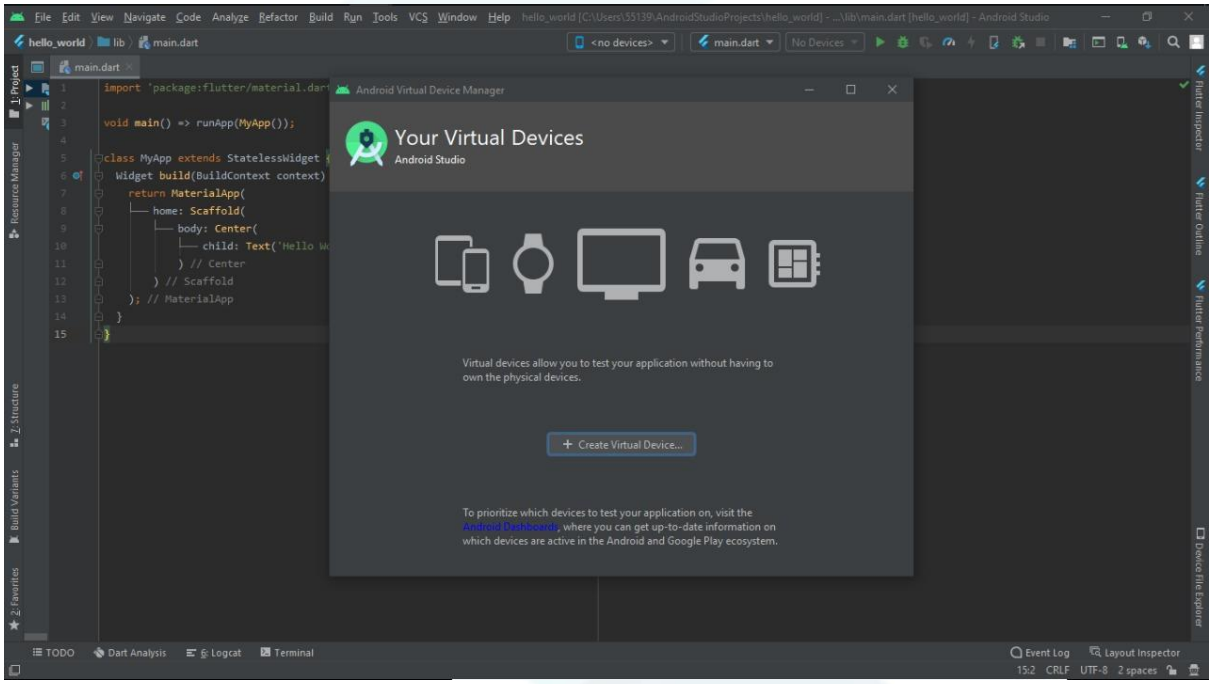
Essa é a página home do projeto. Instalado!

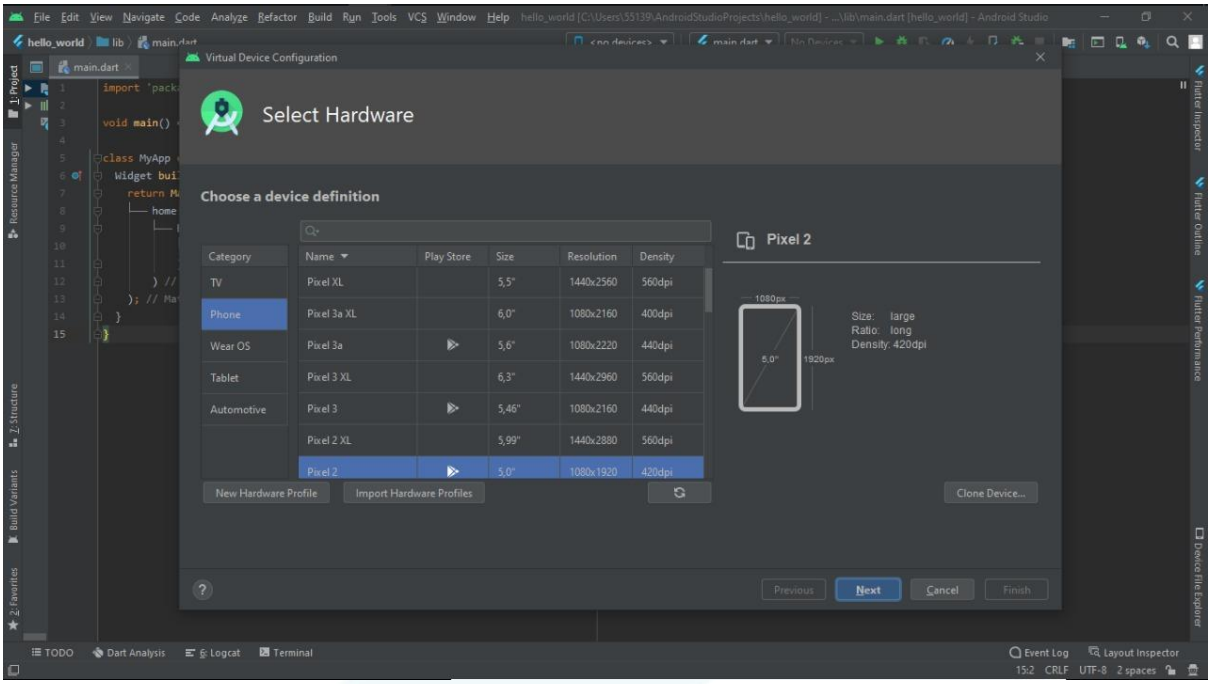


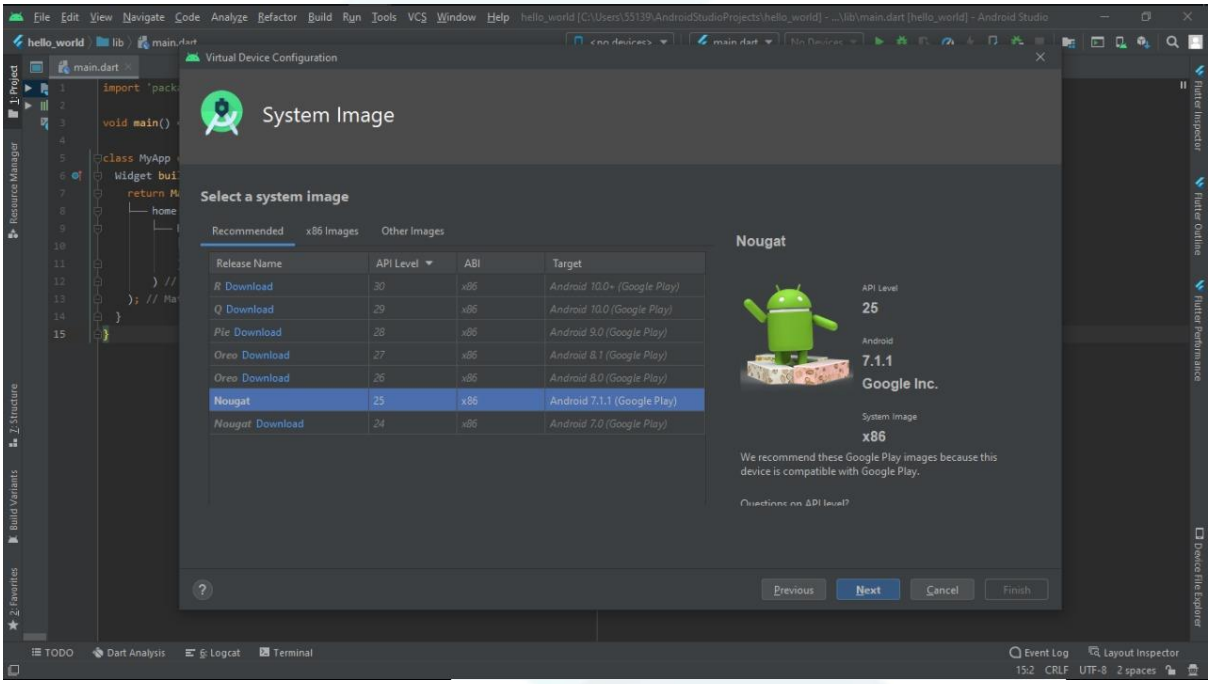
6. Criando o emulador

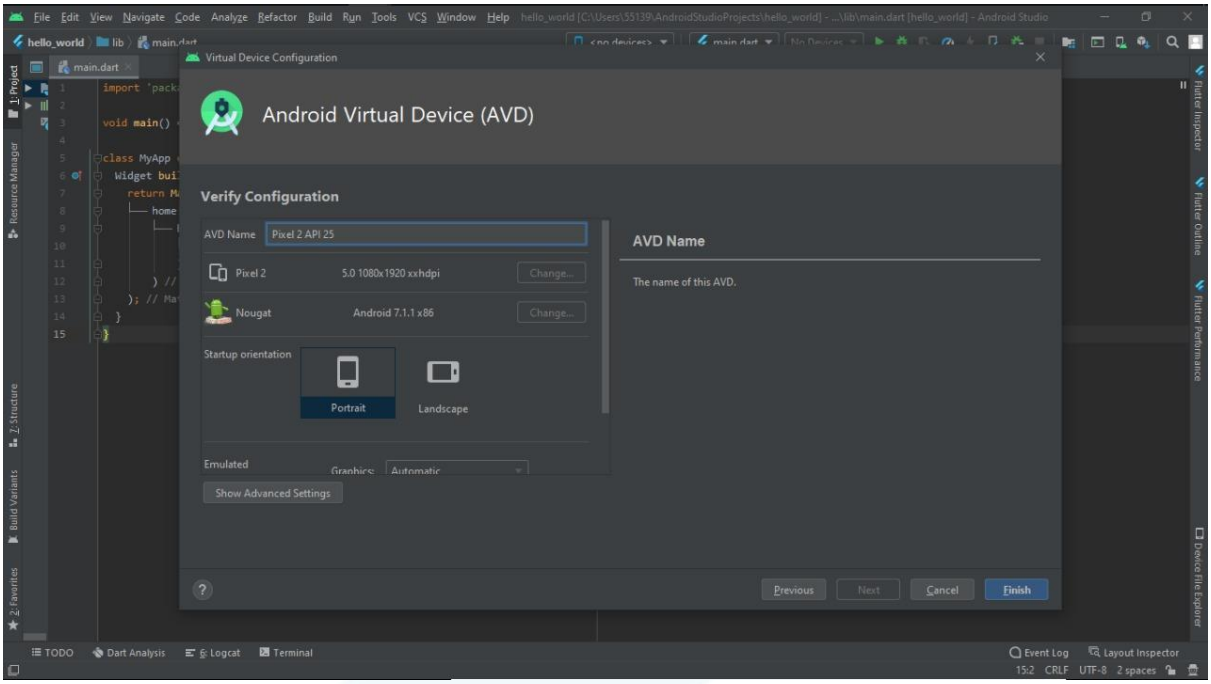
Vá em Tools e em AVD Manager. Clique em “Create Virtual Device”.
Escolha um modelo de ‘Phone” e baixe uma “system image”.

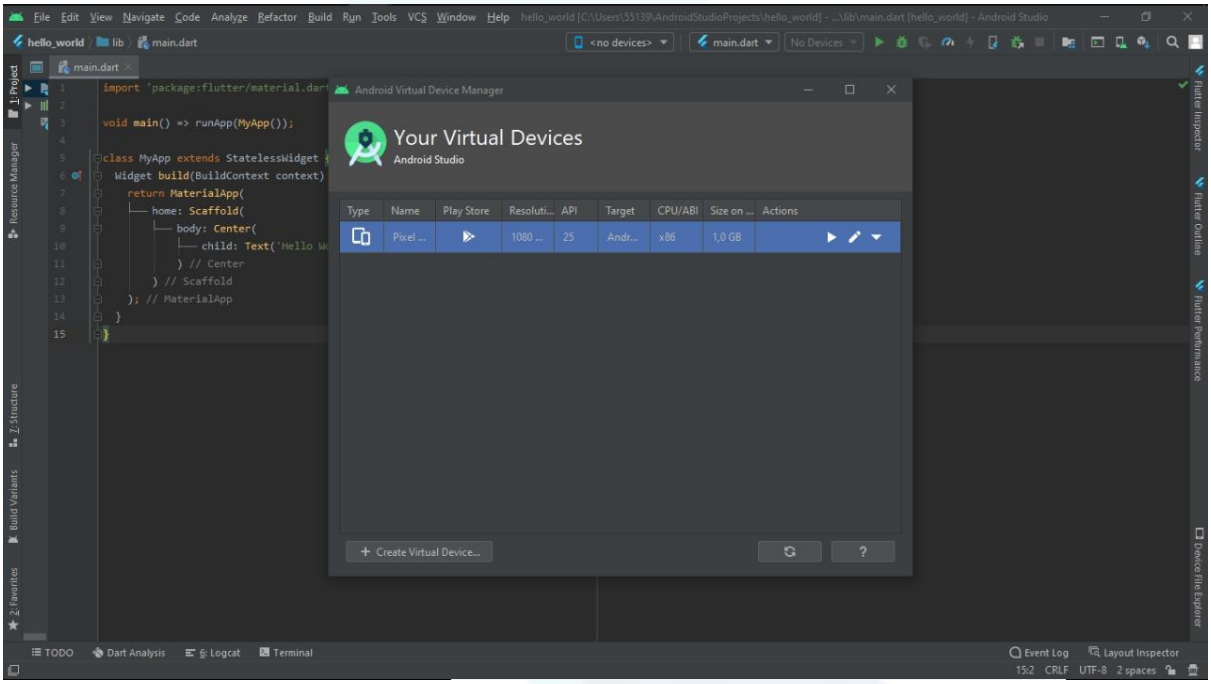

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Tasks & Contexts
AVD Manager
SDK Manager
Resource Manager
Troubleshoot Device Connections
App Links Assistant
Firebase
Layout Inspector
Save as Live Template...
Generate JavaDoc...
IDE Scripting Console
XML Actions
JShell Console...
Kotlin
Flutter
Opens the Android virtual device (AVD) manager which manages emulator images and snapshots

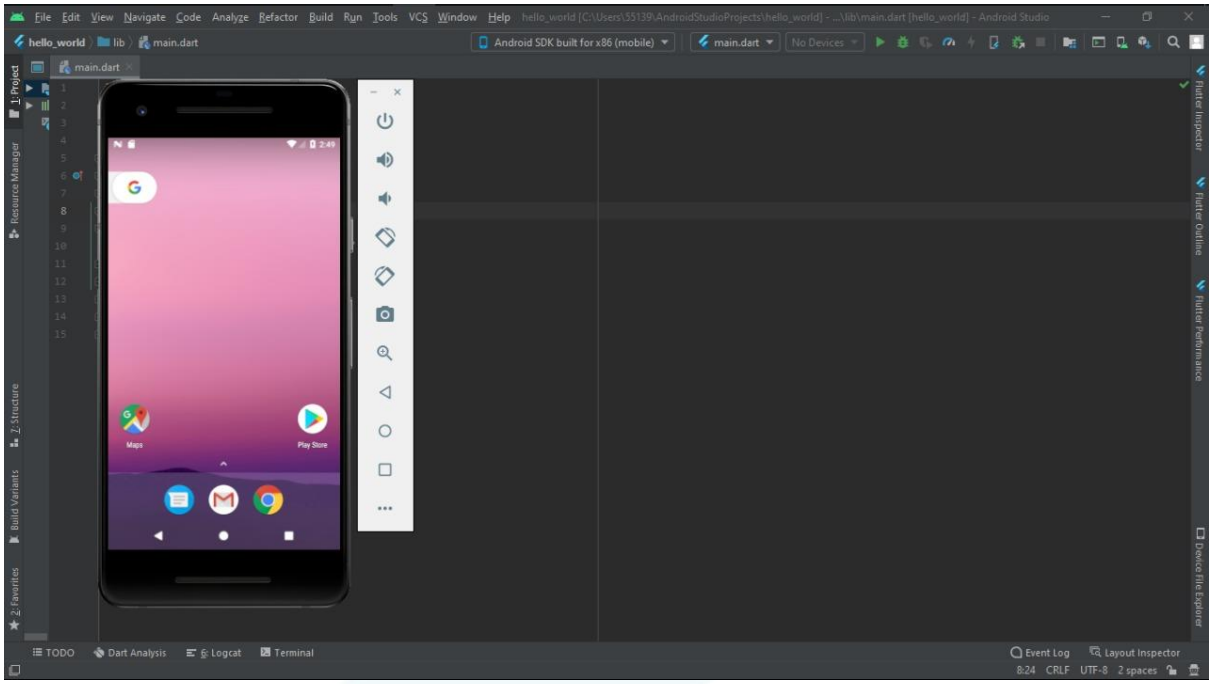

Android Virtual Device Manager
Your Virtual Devices
Android Studio
Virtual devices allow you to test your application without having to own the physical devices.
Create Virtual Device...
To prioritize which devices to test your application on, visit the Android Dashboard, where you can get up-to-date information on which devices are active in the Android and Google Play ecosystem.


Virtual Device Configuration
Select Hardware
Choose a device definition
Category
Name
Play Store
Size
Resolution
Density
TV
Phone
Wear OS
Tablet
Automotive
Pixel XL
Pixel 3a XL
Pixel 3a
Pixel 3 XL
Pixel 3
Pixel 2 XL
Pixel 2
Pixel 2
Size: large
Ratio: long
Density: 420dpi
New Hardware Profile
Import Hardware Profiles
Clone Device...
Previous
Next
Cancel
Finish


Virtual Device Configuration
System Image
Select a system image
Recommended
x86 Images
Other Images
Release Name
API Level
ABI
Target
R Download
Q Download
Pie Download
Oreo Download
Oreo Download
Nougat
Nougat Download
Nougat
API Level
25
Android
7.1.1
Google Inc.
System Image
x86
We recommend these Google Play images because this device is compatible with Google Play.
Previous
Next
Cancel
Finish


Virtual Device Configuration
Android Virtual Device (AVD)
Verify Configuration
AVD Name
Pixel 2 API 25
Pixel 2
5.0 1080x1920 xxhdpi
Change...
Nougat
Android 7.1.1 x86
Change...
Startup orientation
Portrait
Landscape
Emulated
Graphics:
Automatic
Show Advanced Settings
AVD Name
The name of this AVD.
Previous
Next
Cancel
Finish


Android Virtual Device Manager
Your Virtual Devices
Android Studio
Type
Name
Play Store
Resoluti...
API
Target
CPU/ABI
Size on ...
Actions
Create Virtual Device...

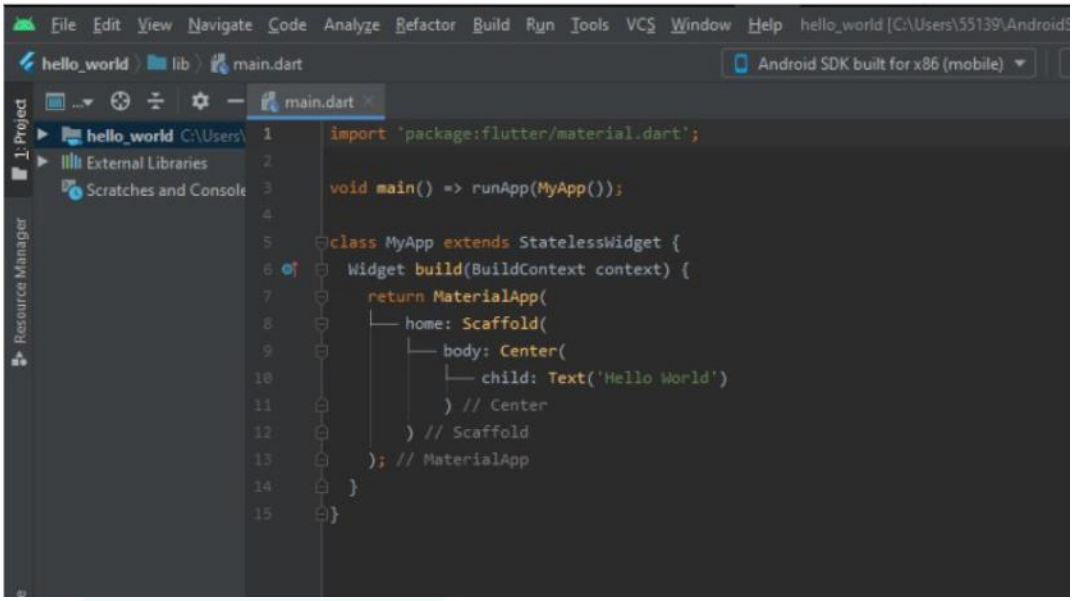


Após isso, o emulador aparecerá aberto!

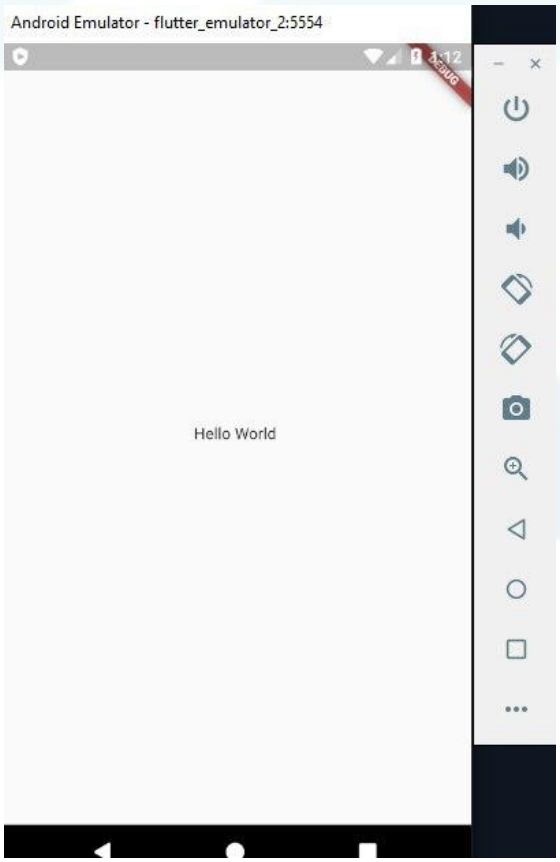
APLICAÇÃO DE TESTE

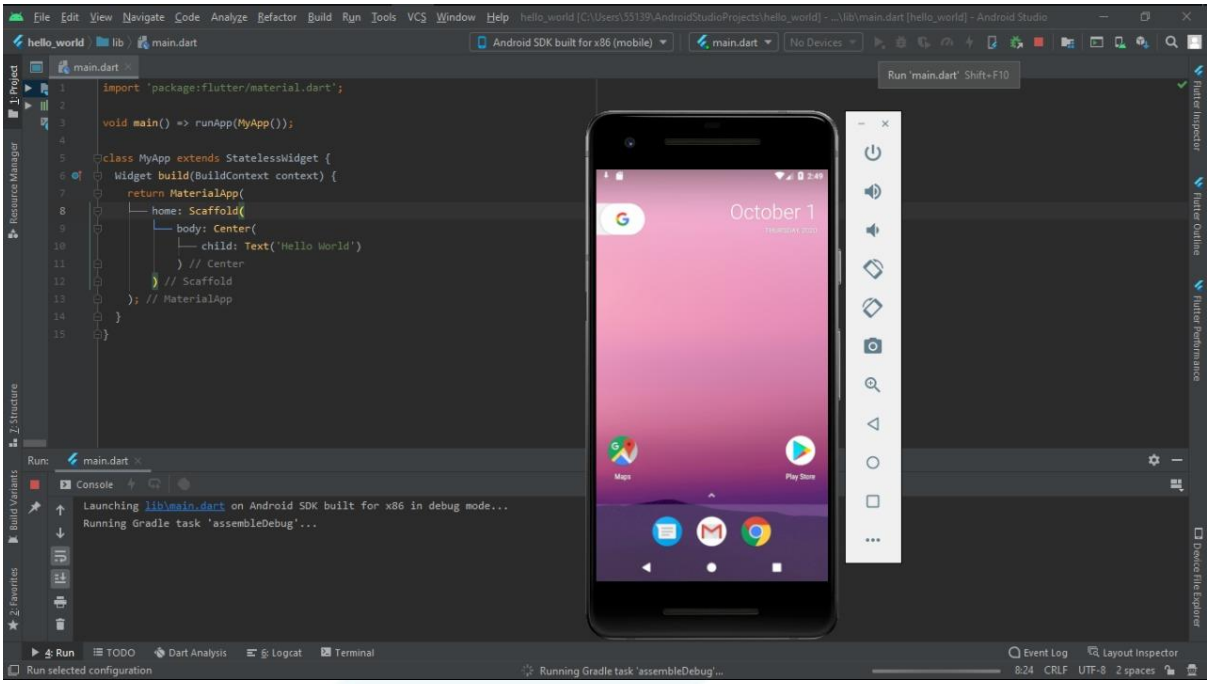
CRIANDO UMA APP “HELLO WORLD”

1. No canto esquerdo superior da página inicial, clique em File -> New Flutter Project e ao abrir, escreva o código abaixo:



2. O resultado do código aparecerá no emulador ao lado:




File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
hello_world
main.dart
Android SDK built for x86 (mobile)
No Devices
Run 'main.dart' Shift+F10
import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
Widget build(BuildContext context) {
return MaterialApp(
home: Scaffold(
body: Center(
child: Text('Hello World')
) // Center
) // Scaffold
); // MaterialApp
}
}
October 1
Maps
Play Store
Run:
Console
Launching lib\main.dart on Android SDK built for x86 in debug mode...
Running Gradle task 'assembleDebug'...
Run
TODO
Dart Analysis
Logcat
Terminal
Event Log
Layout Inspector
Run selected configuration
Running Gradle task 'assembleDebug'...
8:24 CRLF UTF-8 2 spaces

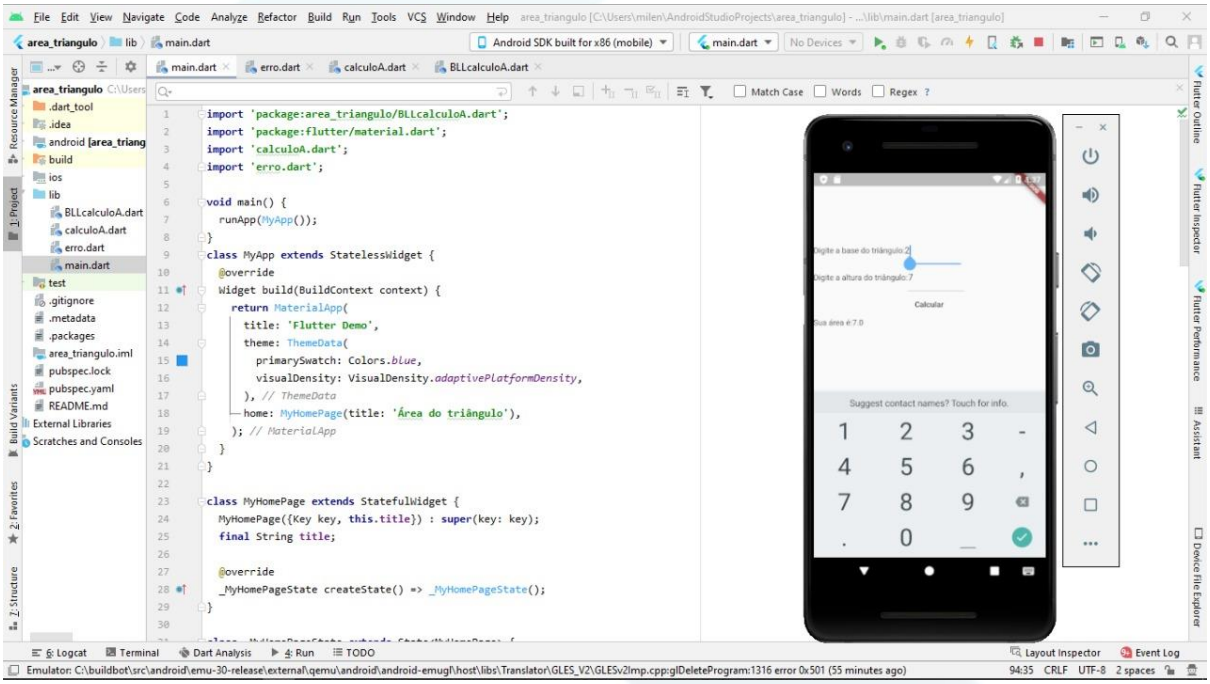
APLICAÇÃO 01 - ÁREA DO TRIÂNGULO

Levando em conta que já ensinamos, no início desta apostila, a como criar um projeto e como inserir o emulador mobile nele, iremos partir para o próximo passo: criar um exemplo de aplicação mobile rodável.

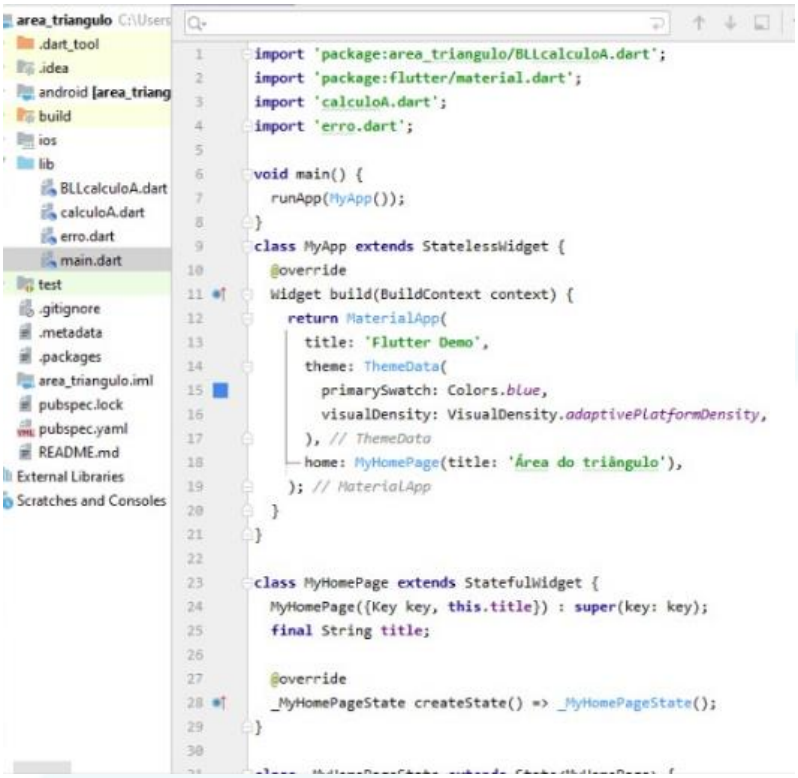
Escolhemos, então, uma aplicação para calcular a área de um triângulo, cuja base e altura deverão ser digitadas pelo usuário. Leia o passo-a-passo abaixo:

1. Primeira aplicação mobile: a app para calcular a área do triângulo cuja base e altura devem ser digitadas pelo usuário.

Layout geral da página Home



No print screen acima, temos o código do layout geral da página única de nossa aplicação, onde consta seu título *title* e seu tema *theme*, no qual este último está inserido a escolha de cor *Colors* do detalhe do layout, que, neste caso, é azul *blue*.



Código principal da aplicação mobile

Esse bloco é responsável para importar as classe que vamos criar futuramente e importar outros packages necessários para o design do app, como o *materal.dart.*

```
import 'package:area_triangulo/BLLcalculoA.dart' ;
import 'package:flutter/material.dart';
import 'calculoA.dart';
import 'erro.dart';
```

Esse bloco é criado automaticamente quando criamos um projeto do flutter, necessário para rodar o app.

```
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
```

```
    title: 'Flutter Demo',
    theme: ThemeData(
      primarySwatch: Colors.blue,
      visualDensity: VisualDensity.adaptivePlatformDensity,
    ),
    home: MyHomePage(title: 'Área do triângulo'),
  );
 }
}
class MyHomePage extends StatefulWidget {
 MyHomePage({Key key, this.title}) : super(key: key);
 final String title;
 @override
 _MyHomePageState createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
```

Temos, primeiramente, a declaração das variáveis assim como controleB e controleH que servem para resgatar valores dos TextField para adquirir os valores da base e altura.

```
 String resposta, area;
 TextEditingController controleB = TextEditingController();
 TextEditingController controleH = TextEditingController();

 void calcularA() {
  setState(() {
    calculoA calc = calculoA();
    calc.valorB = controleB.text;
    calc.valorH = controleH.text;
    BLLcalculoA.validaDados(calc);
    if(Erro.error){
      resposta = Erro.msg;
    } else {
      area = calc.area;
      return resposta = area;
```

```
    }
  }
  );
}
```

Código do layout da aplicação

Algumas considerações sobre o código da aplicação:

- os itens de layout como textbox, label e button, são criados e instanciados no código em questão;
- a cor **vinho** indica os itens de layout;
- a cor **verde musgo** indica a explicação dos itens e sua necessidade.

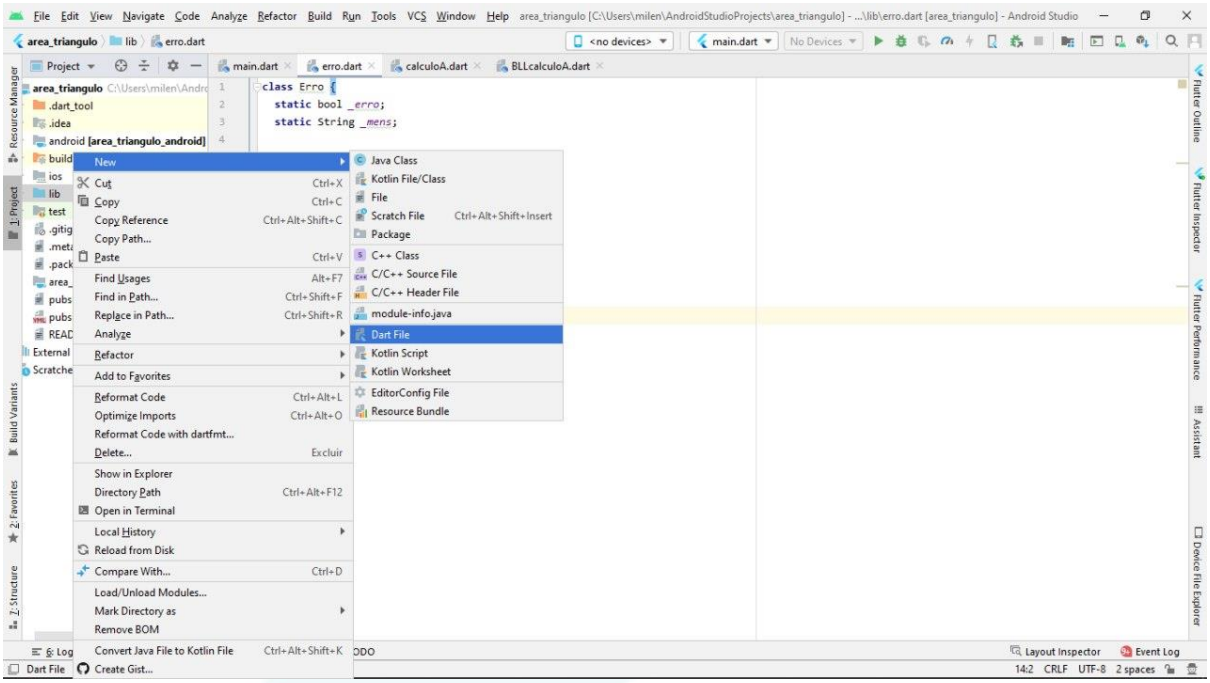
Código da aplicação:

```
@override
  Widget build(BuildContext context) {
return Scaffold(
        //Widget necessário para o layout do flutter, contendo o conteúdo que será mostrado na tela
        body: Center(
        // o widget Center ocupa toda a área possível
        child: Column(
        // adicionamos Column para organizar os filhos (children) um abaixo do outro
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
        Row(
        // adicionamos Row para organizar os filhos (children) um ao lado do outro
        children: [
                Text(
                // adicionamos Text para orientar o usuário o que escrever no TextField
                'Digite a base do triângulo:',
                ),
                Container(
```

```
                //criamos um Container para declarar um TextField que seria filho (child) com um tamanho (width) de
                // 100px, porque os TextField precisam de um tamanho
                width: 100,
                child: TextField(
                //TextField para o usuário inserir valores
                controller: controleB,
                ),
                ),
        ],
        ),
        Row(
        // adicionamos Row para organizar os filhos (children) um ao lado do outro
        children: [
                Text(
                // adicionamos Text para orientar o usuário o que escrever no TextField
                'Digite a altura do triângulo:',
                ),
                Container(
                //criamos um Container para declarar um TextField que seria filho (child) com um tamanho (width) de
                // 100px, porque os TextField precisam de um tamanho
                width: 100,
                child: TextField(
                //TextField para o usuário inserir valores
                controller: controleH,
                keyboardType: TextInputType.number,
                ),
                ),
        ],
        ),

        FlatButton(
        // Esse Widget seria um botão, que quando apertado, chamaria a função calcularA (para calcular a área)
        // e tem como filho (child) um Text, que seria o texto que aparece no botão
```

```
            onPressed: calcularA,
            child: Text('Calcular')
        ),

        Row(
        // adicionamos Row para organizar os filhos (children) um ao lado do outro
        children: [
            Text(
            //indicamos o que está escrito ao lado
            'Sua área é:',
            ),
            Text('$resposta'  //mostramos a área por meio desse Text
        ),
        ],
        ),
        ],
        ),
        ),
        );
  }
}
```

Como criar classes no Android Studio



Criando a classe Erro

Essa é uma classe para retornar uma mensagem quando erro for verdadeiro, pode se ver a demonstração e aplicação na função calculoA.

```
class Erro {
 static bool _erro;
 static String _mens;

 static set error(bool erro) {
   _erro = erro;
 }
 static set msg(String mens) {
   _erro = true;
   _mens = mens;
 }
 static get error => _erro;
 static get msg => _mens;
}
```

Criando a classe calculoA

Essa classe é responsável para receber os valores da altura e base e efetuar o cálculo da área e retorná-la. Podemos ver sua aplicação na função calculoA, onde essa classe é chamada.

```
class calculoA {
 String _h, _b;

 String get h => _h;
 String get b => _b;
 String get area => (double.parse(_b) * double.parse(_h) / 2).toString();
 set valorH(h) {
   return _h = h;
 }
 set valorB(b) {
   return _b = b;
 }

}
```

Criando a BLLcalculoA

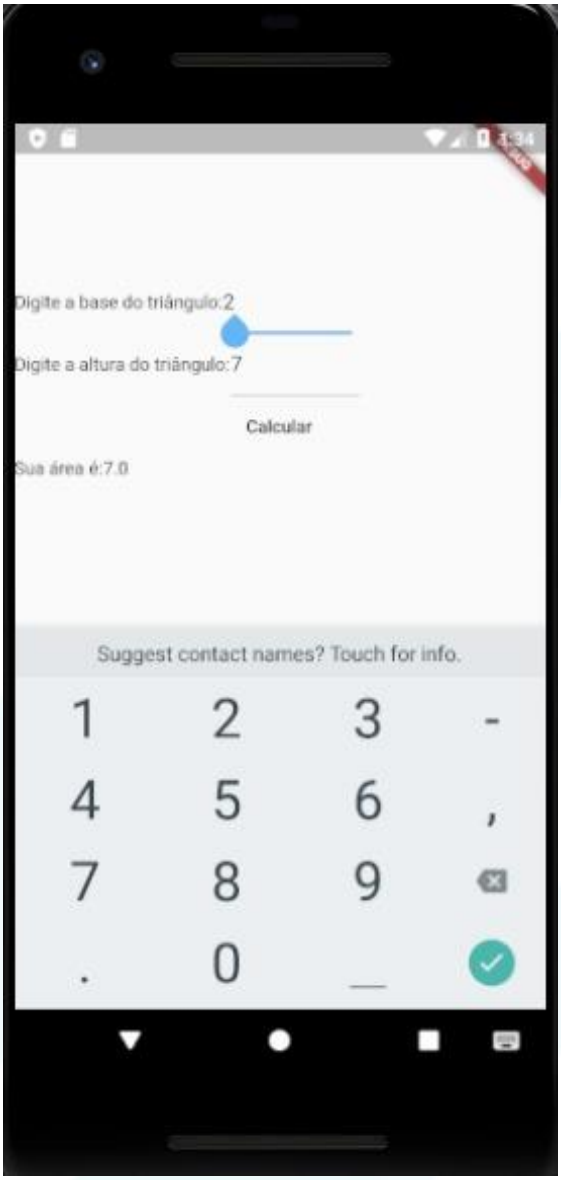
Essa é uma camada para verificação dos dados, para ver se o usuário digitou um número do tipo double.

```
import 'erro.dart';
import 'calculoA.dart';

class BLLcalculoA{
 static validaDados(calculoA calc){
   Erro.error = false;
   if(calc.h.length == 0){
     Erro.msg = "O campo da altura é de preenchimento obrigatório";
   }else{
     try{
```

```
      double.parse(calc.h);
    }
    catch(id){
      Erro.msg = "O campo de ALTURA deve ser numérico";
      return;
    }
  }
  if(calc.b.length == 0){
    Erro.msg = "O campo de BASE é de preenchimento obrigatório";
  }else{
    try{
      double.parse(calc.b);
    }
    catch(id){
      Erro.msg = "O campo BASE deve ser numérico";
      return;
    }
  }
 }
}
```

Emulador

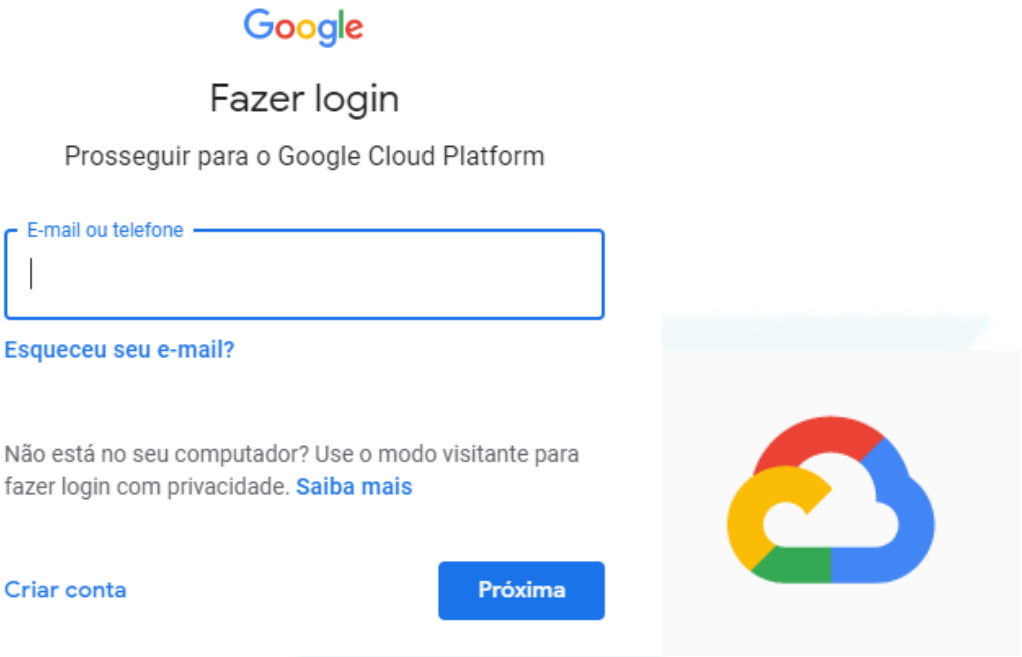
Após finalizarmos tudo isso, a nossa aplicação mobile rodará no emulador desta forma aqui apresentada:



APLICAÇÃO 02 - GPS E LOCALIZAÇÃO DO CELULAR

Instalando as API's necessárias no Google Cloud

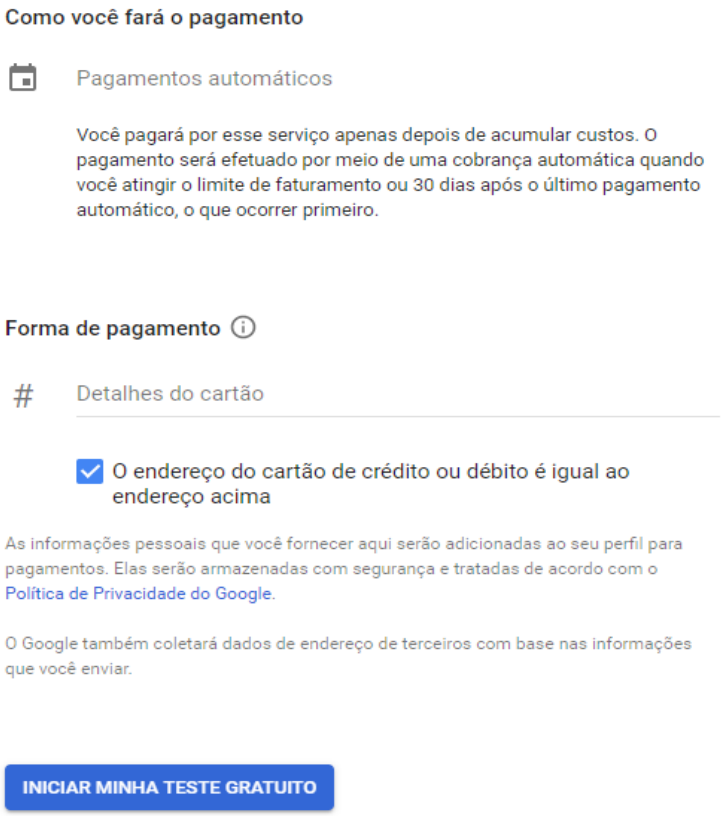
1. O primeiro passo a se realizar para uma aplicação com Google Maps é acessar a plataforma da Google Cloud através do endereço console.cloud.google.com e fazer o login com a sua conta da Google.



2. Ao se conectar, informe seu país, concorde com os termos de uso e clique em concordar e continuar.

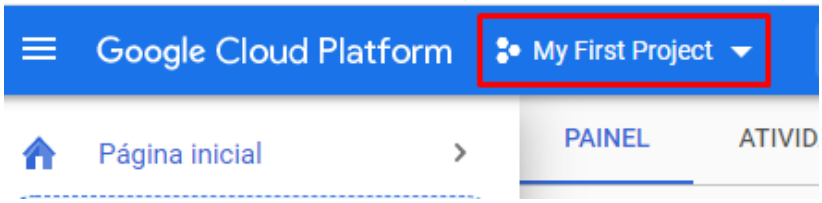
3. Ao adentrar a página principal do Google Cloud, o usuário será informado de que os serviços oferecidos são pagos. Porém, é disponibilizado um vale de U$300 *(trezentos dólares)* durante o período de 90 dias.

4. Após o preenchimento de dados pessoais é necessário inserir os dados de um cartão de débito ou crédito. Mas fique tranquilo, eles explicam que nenhuma cobrança automática será feita após o término do período de teste gratuito e solicitem seu cartão de crédito para ter certeza de que você não é um robô. Você não será cobrado, a menos que atualize manualmente para uma conta paga.

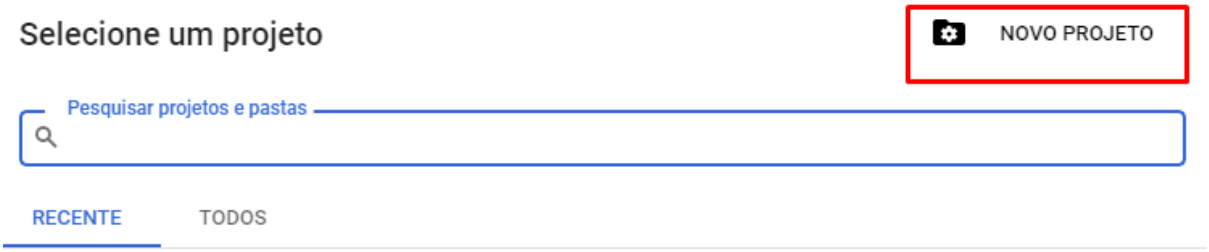


5. Para criar o projeto devemos clicar em representado na imagem abaixo, posteriormente em “novo projeto” e por último preencher as informações requeridas

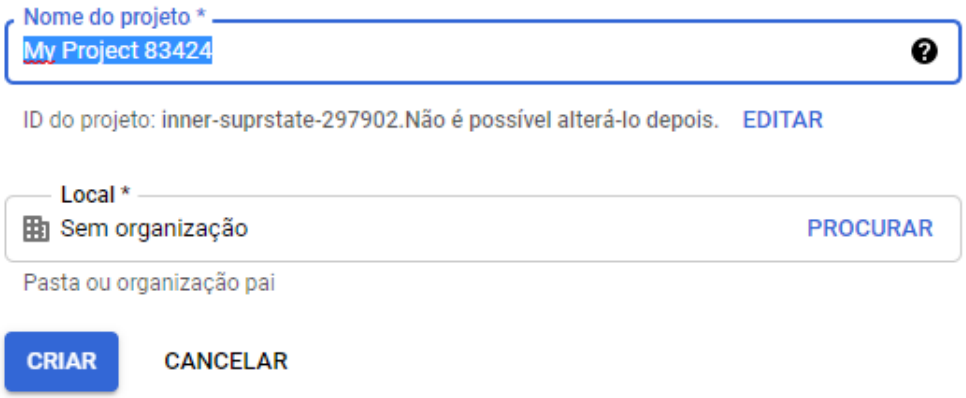
5.1



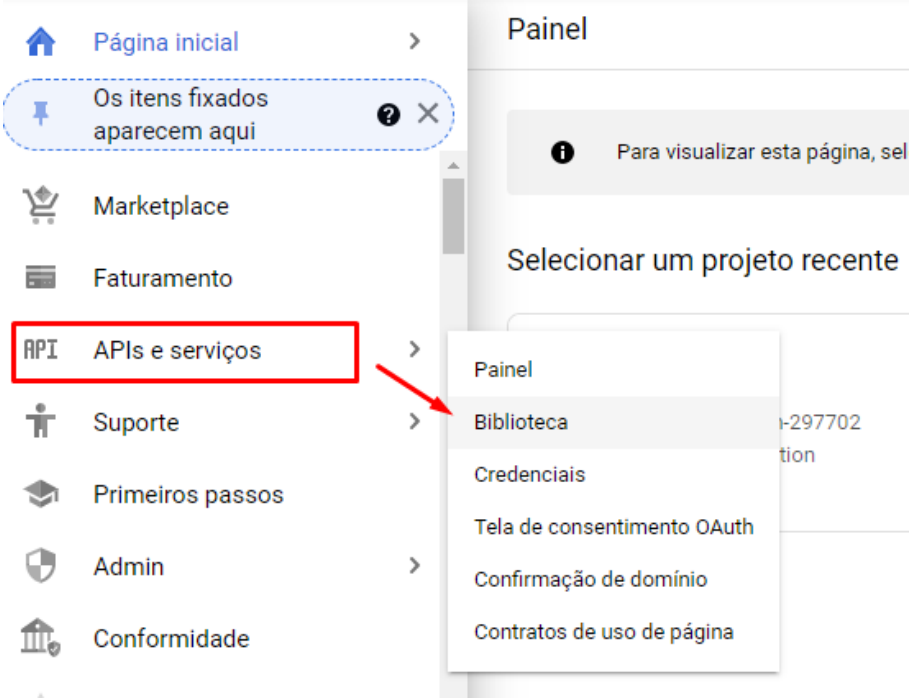
5.2



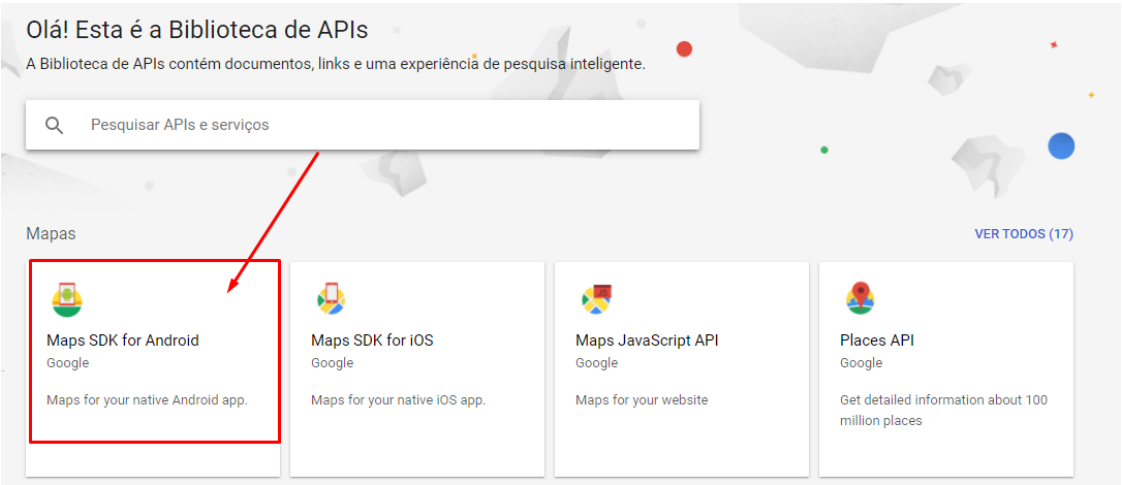
5.3



6. Após a criação e seleção do projeto vá em *APIs e serviços > Bibliotecas.*



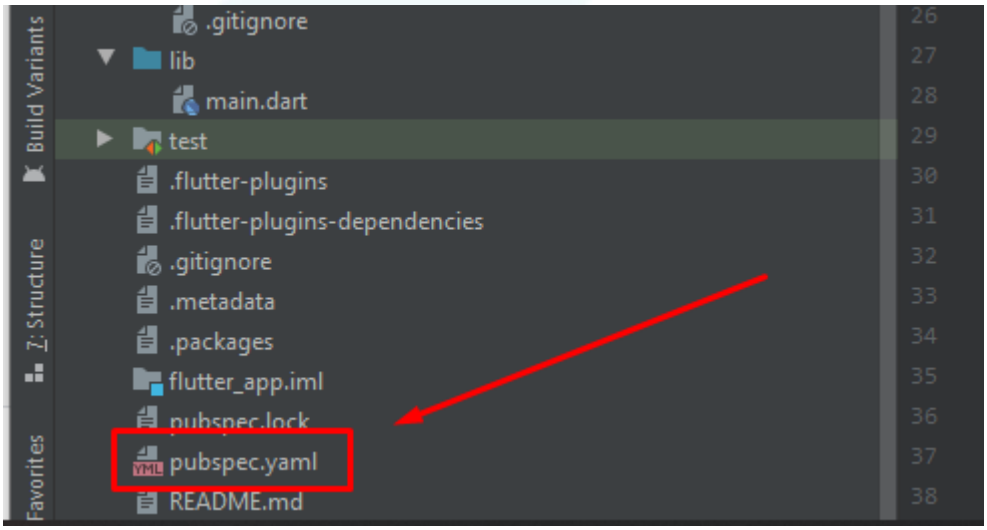
7. Agora devemos habilitar os mapas, como estamos criando uma aplicação para android, vamos habilitar a biblioteca para android em "*Maps SDK para Android".*



8. Outro recurso necessário é o *"Geolocation API".* Devemos pesquisá-lo da mesma forma que o anterior e ativá-lo.

Criando o código no Android Studio

1. Após a instalação das API's devemos abrir uma aplicação e o emulador e ir até o aplicativo *pubspac.yaml*

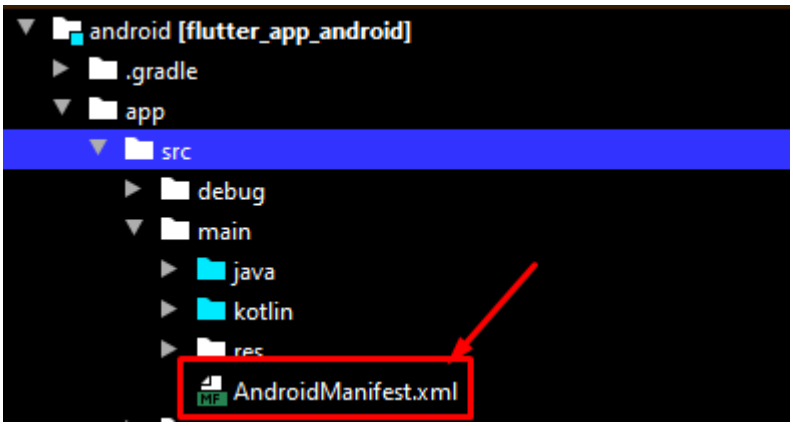


2. Nas dependências digitamos o código abaixo:

Obs: tome cuidado com a identação, não deixe de alinhar o *"google_maps_flutter"* da forma descrita na imagem.

```
dependencies:
  flutter:
    sdk: flutter
  google_maps_flutter: ^0.5.25+1
```

3.Agora iremos adicionar algumas configurações de mapa para o Android. Para isso iremos em *Android > App > Src > Main e* a abrir o arquivo *AndroidManifest.xml.*

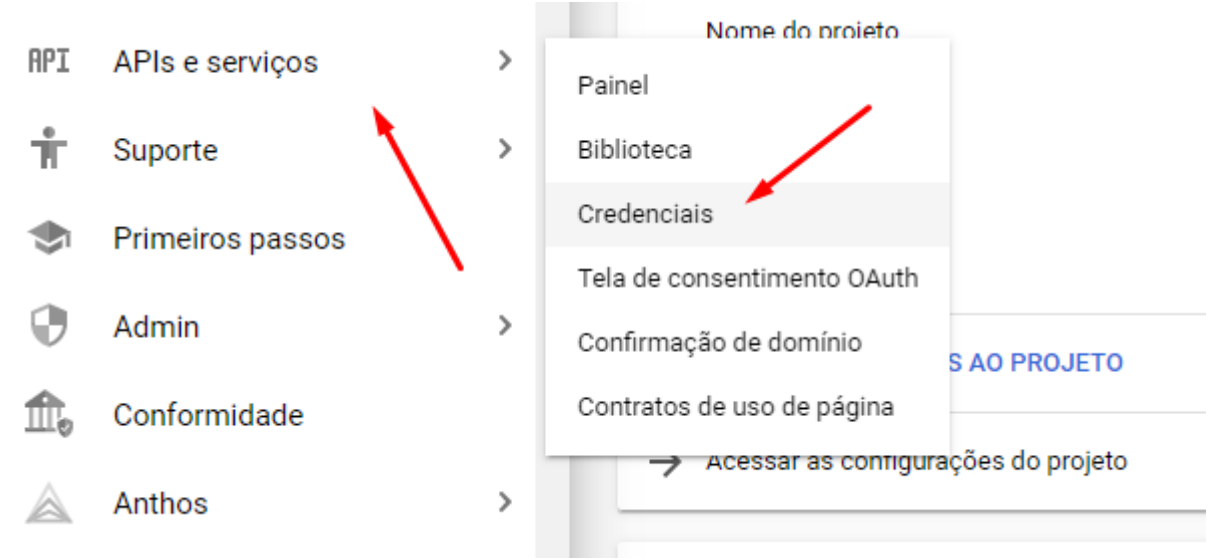


4. Em *AndroidManifest.xml.* antes de Activity vamos adicionar o seguinte meta:

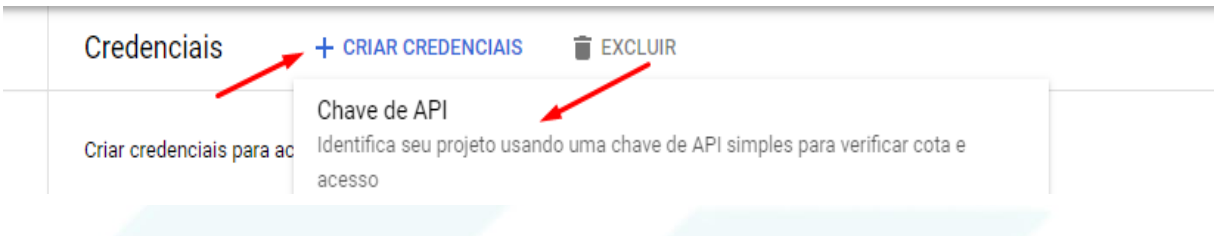
```
<application
    android:name="io.flutter.app.FlutterApplication"
    android:label="googlemaps_aula14"
    android:icon="@mipmap/ic_launcher">
    <meta-data android:name="com.google.android.geo.API_KEY"
        android:value=" CHAVE DA SUA API "/>

    <activity
        android:name=".MainActivity"
        android:launchMode="singleTop"
```

Note que em “Value” devemos adicionar a chave de nossa API.

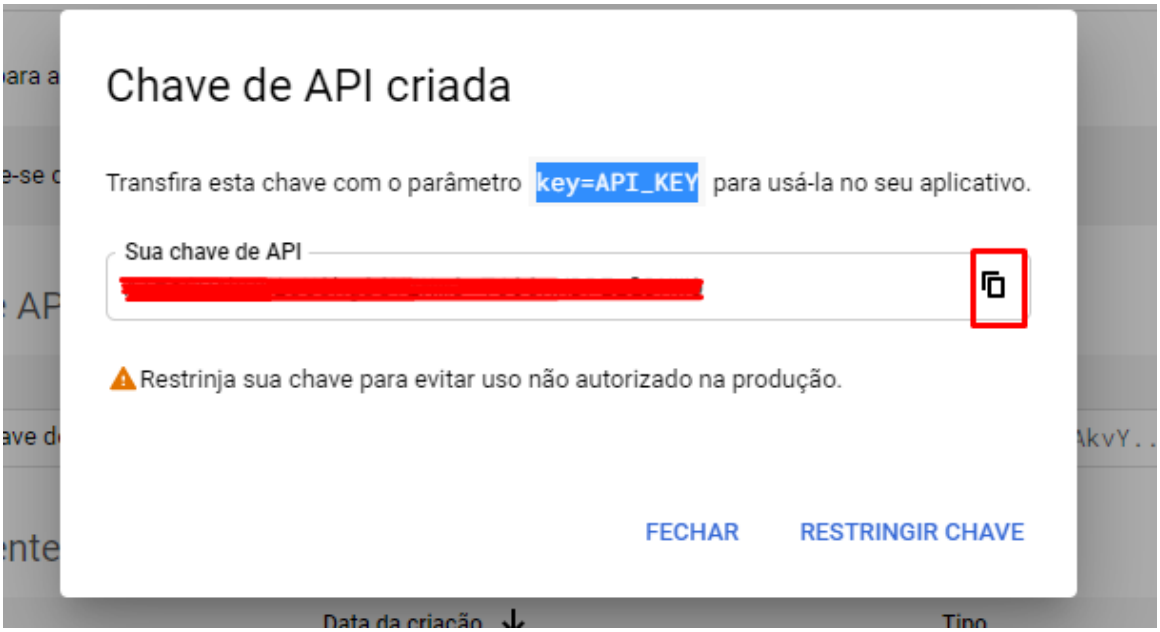
5. Para obter uma chave devemos voltar ao Google Cloud e seguir por *API’s e Serviços> Credenciais.*



5.1. E posteriormente em *Criar Credenciais* > *Chave de API*



5.2. Copie a chave de API gerada



5.3. E cole no local descrito.

```
android:value=" CHAVE DA SUA API "/>
```

6. Agora que configuramos vamos criar um novo arquivo em lib denominado google.Maps.dar que será nossa página de mapas. E nele adicionaremos um *StatefulWidget.*

6.1. Primeiramente fizemos um import do pacote do google maps

```
3  //Importação do Mapa
4  import 'package:google_maps_flutter/google_maps_flutter.dart';
5
```

7. Para que seja trabalhado um mapa, onde consigamos dar zoom e trocar localizações, adicionamos um controller “mapcontroller”.

```
11  //Deslocamento do mapa
12  class _MapPageState extends State<MapPage> {
13    GoogleMapController mapController;
```

7.1.criamos um marcador e as variáveis longitude e latitude utilizadas no mapa

```
15  //Inserção do Marcador
16  Set<Marker> markers = new Set<Marker>();
17
18  //Variáveis
19  double lat = 45.521563;
20  double long = -122.677433;
21
```

- instanciamento do marcador:

```
markers: markers,
```

- informações fixas de exibição do marcador para as determinadas coordenadas:

```
50
51          //Inserção e informações do marcador
52          final Marker marker = Marker(
53            markerId: new MarkerId("123456"),
54            position: position,
55            infoWindow: InfoWindow(
56              title: "Casa",
57              snippet: 'Cubatão/SP',
58            ), // infoWindow
```

7.2. Após a criação de um marcador e do controle fazemos a ligação do controle com o mapa na linha a seguir:

```
22  //Interligação do mapa com o controle
23  void _onMapCreated (GoogleMapController controller) {
24    mapController = controller;
25  }
```

8. Após isso, no body instanciamos o GoogleMap:

```
66      body: Stack(
67        children: <Widget>[
68          Container(
69              child: GoogleMap(
70          onMapCreated: _onMapCreated,
71
```

9. Para que seja feito um teste atribuímos valores fixos a longitude e latitude, lembrando que foi criado uma caixa de texto onde o usuário digitaria o seu local de pesquisa assim o “text field” (a caixa de texto) atribuiria os respectivos valores daquele local pesquisando as latitudes e longitudes no google, porém fora atribuído valores as essas variáveis para a simplificação. .

- atribuição/instanciamento de valores

```
26  //Latitude e Longitude
27  void CEP(){
28    setState(() {
29      lat = -23.9050876;
30      long = -46.4296952;
31    });
32  }
```

- simplificação no text field com valores fixos

```
lib > map_page.dart > MapPageState > build
39      //Inserção da caixa de texto na barra superior do aplicativo
40      title: TextField(
41        onSubmitted: (val) {
42          lat = -23.9050876;
43          long = -46.4296952;
44
```

9.1.Para que a câmera fosse deslocada para as coordenadas dadas, criamos um position (posição) e encaminhamos a mesma para a nova posição com o “mapcontroller.moveCamera”:

```
45        //Determinada posição no mapa
46        LatLng position = LatLng(lat, long);
47
48        //Encaminhamento da câmera para a nova posição
49        mapController.moveCamera(CameraUpdate.newLatLng(position));
50
51        //Inserção e informações do marcador
```

10. Para que toda vez que o usuário toque na tela ou movimente a câmera recebessemos os dados, utilizamos os seguintes dados:

- toque na tela

```
76
77        //Recebimento de informações quando a tela for tocada
78        onTap: (position) {
79          print(position);
80        },
81
```

- movimentação da câmera.

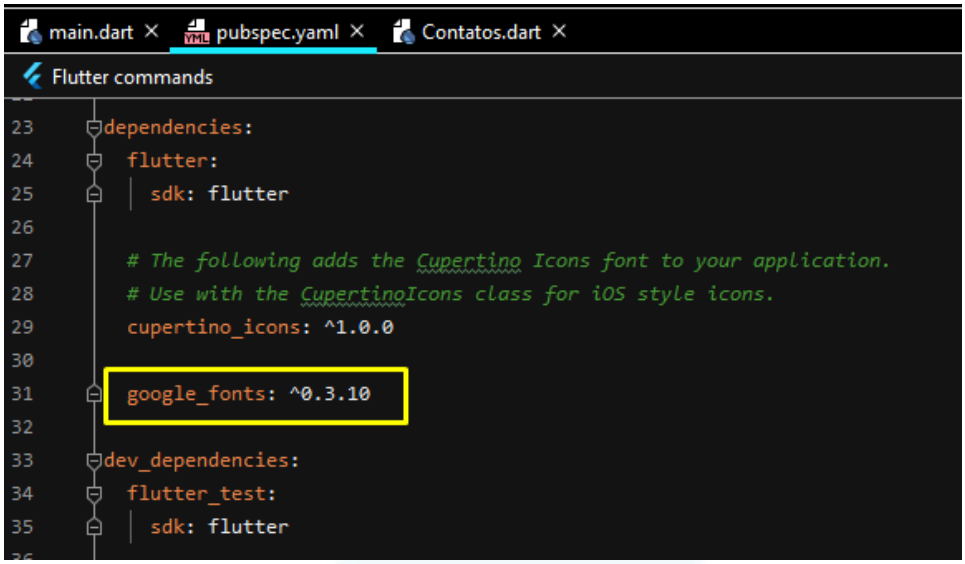
APLICAÇÃO 03 - LISTA DE CONTATOS

PASSO A PASSO

Agora vamos iniciar a nossa terceira aplicação de Flutter e nela iremos abordar uma lista de contatos de um dispositivo mobile. O objetivo desse app é gerar uma lista de contatos no celular por meio de POO e, após isso, criar um mecanismo de pesquisa para localizar algum contato dessa lista. Vamos para o passo a passo:

1. Adicionar Google Fonts no arquivo pubspec.yaml:

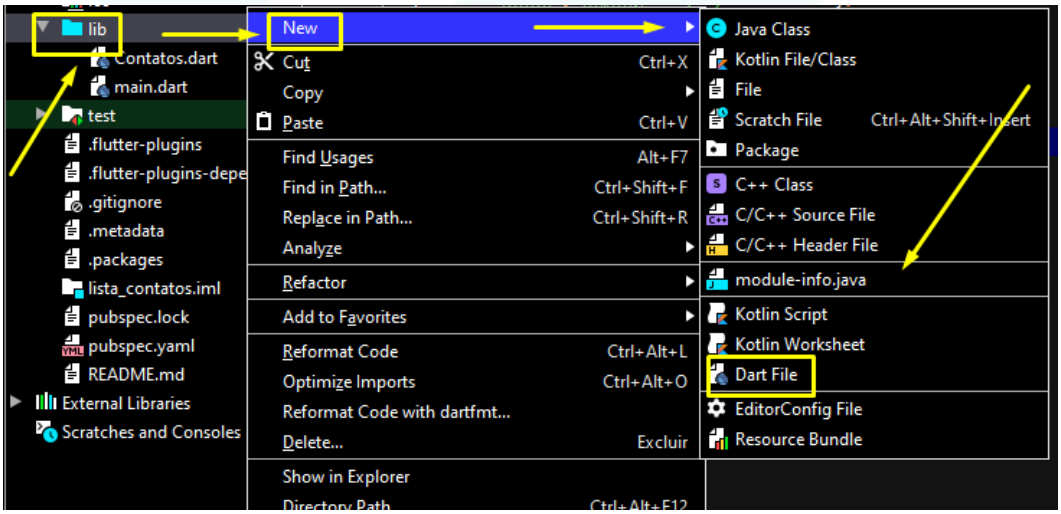
Coloque a linha de código abaixo no arquivo pubspec.yaml, de preferência abaixo da linha “cupertino_icons”, como indicado no print screen abaixo:



2. Criar página .dart para a classe Contatos

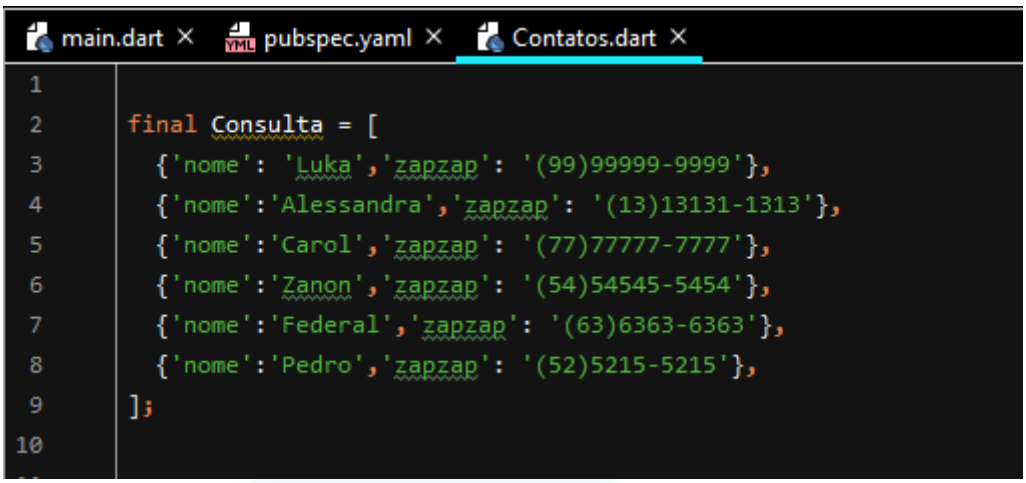
2.1. Criando o File .dart:

Na biblioteca ‘lib’, clique com o botão direito do mouse. Vá em ‘New’ e em ‘Dart file’. Depois nomeie o arquivo com o mesmo nome da classe que será utilizada.



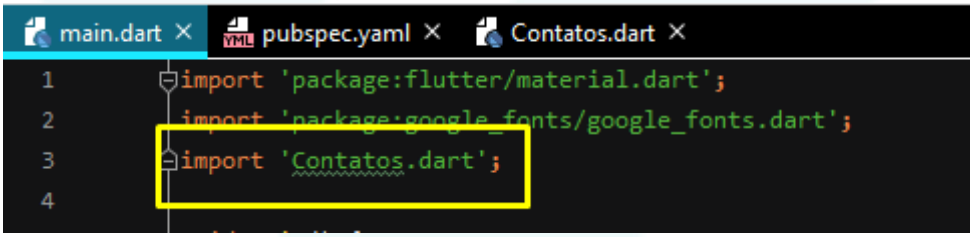
2.2. Criando a classe “Contatos”:

Crie a classe utilizando o termo “final”, indicando o nome da variável que acomodam todos os contatos da lista, bem como seus nomes. Deste modo abaixo:



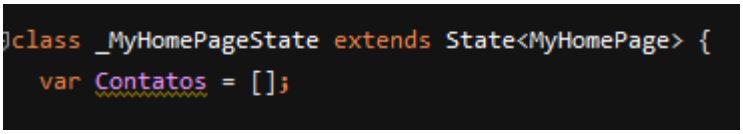
3. Importar a classe Contatos.dart

Para importar a classe Contatos no arquivo principal, digite a linha abaixo no arquivo “main.dart”:

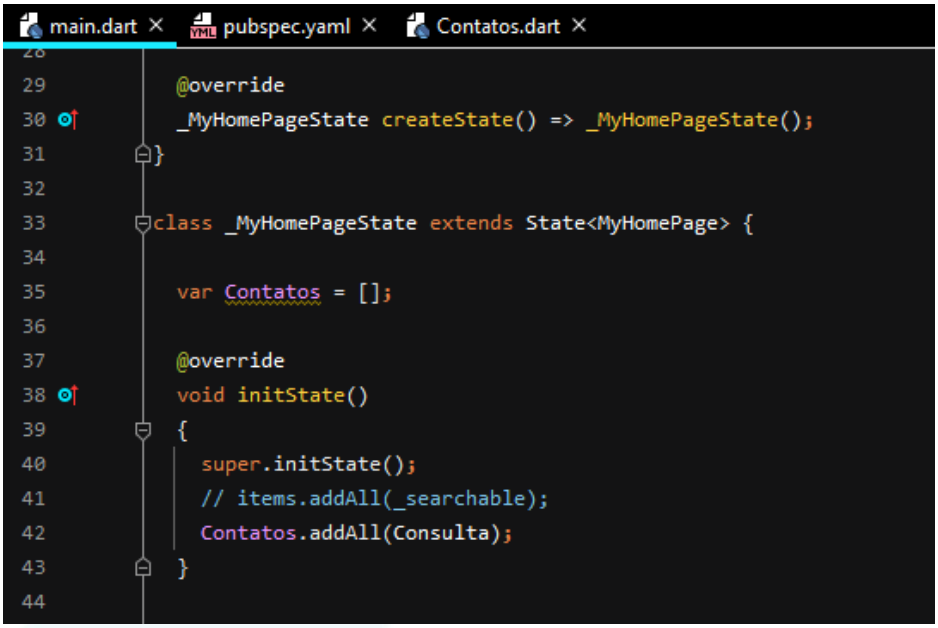


4. Função para instanciar variável dos contatos para o celular

Primeiramente, instanciamos a variável responsável por armazenar todos os contatos do dispositivo, desta forma:



Após isso, é necessário inserir todos os dados no armazenamento da variável. Para isso, a conectamos com a classe Contatos que fora criada, a partir do método initState e o atributo addAll:

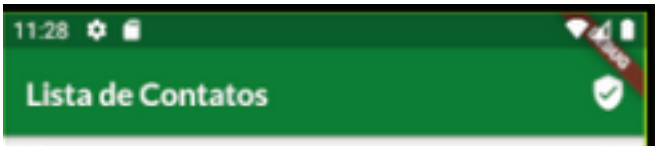


5. Configuração do Scaffold (layout) da lista de contatos

O Scaffold é o lugar onde iremos setar e configurar todo o layout (design, parte visual) da nossa lista de contatos. É claro: grande parte desses atributos são opcionais! Porém, quanto melhor visualmente, melhor o resultado gerado! Vamos lá:

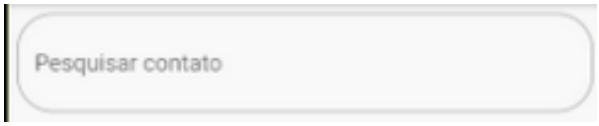
5.1. APP BAR

É a barra da lista de contatos que se encontra no topo da página inicial. Ela que nos dá o título da aba da lista:



5.2. CONTAINER

É a barra de pesquisa, onde iremos utilizar para pesquisar os contatos da lista:



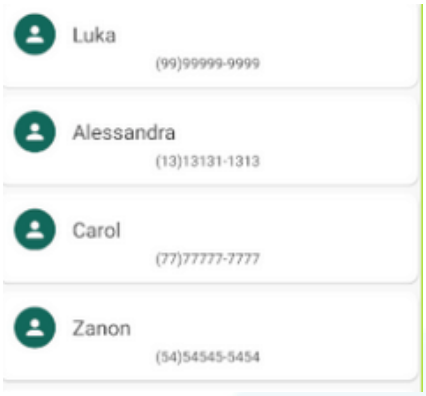
Veja abaixo como foram configuradas o layout de exemplo:

main.dart × pubspec.yaml × Contatos.dart ×

```
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(

      centerTitle: false,
      backgroundColor: Color(0xff007E33),
      actions: <Widget>[
      IconButton(icon: Icon(Icons.verified_user, color: Colors.white), onPressed: null,)
      ],  // <Widget>[]
      title: Text(widget.title, style: GoogleFonts.lato(
          textStyle: TextStyle(
              color: Colors.white, fontSize: 20, fontWeight: FontWeight.w600
          )  // TextStyle
      ),),  // Text
    ),  // AppBar

    body: Center(

      child: Column(

        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
        new Container(
          padding: EdgeInsets.all(2.0),
          margin: EdgeInsets.all(5.0),
          child: TextField(
            decoration: InputDecoration(
                border: OutlineInputBorder(
                    borderRadius: BorderRadius.all(Radius.circular(25.0))
```

5.3. PADDING

São os contatos, os itens contidos na nossa lista. Foram configurados com o icon de usuário ao lado esquerdo, simulando uma lista de contatos comum dos dispositivos:



5.4.CIRCLE AVATAR

É o ícone da lista de contatos, que simboliza um conjunto de pessoas.

Veja a codificação dos atributos e características da lista nos print screens abaixo:

```
74          ),  // OutlineInputBorder
75          prefix: Icon(Icons.person_search_outlined),
76          labelText: 'Pesquisar contato'
77        ),  // InputDecoration
78        onChanged: (value){
79          filterContact(value.toLowerCase());
80        },
81      ),  // TextField
82    ),  // Container
83    new Expanded(
84      child: ListView.builder(
85        itemBuilder: (context, int index){
86          return new Card(
87            elevation: 1.0,
88            shape: RoundedRectangleBorder(
89                side: BorderSide(color: Colors.white70,width: 1),
90                borderRadius: BorderRadius.circular(10)
91            ),  // RoundedRectangleBorder
92            child: new Container(
93              margin: EdgeInsets.all(9.0),
94              padding: EdgeInsets.all(6.0),
95              child: Column(
96                  children: <Widget>[
97                Row(
98                  children: <Widget>[
99                  new CircleAvatar(
100                   child: Icon(Icons.person),
101                   backgroundColor: Color(0xff00695c),
102                   foregroundColor: Colors.white,
```

```
103                                    ),  // CircleAvatar
104                                    new Padding(padding: EdgeInsets.all(8.0)),
105                                    Text(
106                                      '${Contatos[index]['nome']}',
107                                      style: TextStyle(fontSize: 20.0),
108                                    ),  // Text
109                                  ],  // <Widget>[]
110                                ),  // Row
111                                Text('${Contatos[index]['zapzap']}'
112                                ),  // Text
113                              ]  // <Widget>[]
114                            ),  // Column
115                          ),  // Container
116                        );  // Card
117                    },
118                    itemCount: Contatos.length,
119                  )  // ListView.builder
120              ),  // Expanded
121            ],  // <Widget>[]
122          ),  // Column
123        ),  // Center
124      );  // Scaffold
125    }
```

6. Função para pesquisar contatos na lista

Agora, vamos para a parte não menos importante do projeto: a função de pesquisar os contatos! Iremos utilizar uma estrutura if: else: para chegar se a barra de pesquisa se encontra não vazia para, aí sim, o processo de pesquisa ser realizado.

A lógica usada é a seguinte: será pesquisado para cada elemento da lista o conteúdo digitado na barra de pesquisa, e apenas retornará se tiver o contato digitado.

```
void filterContact(String pesquisaContato)
{
  var Search = [];
  Search.addAll(Consulta);
  if(pesquisaContato.isNotEmpty)
  {
    List<Map<String, dynamic>> tmpList = List<Map<String, dynamic>>();
    Search.forEach((element) {
      if (element['nome'].toLowerCase().contains(pesquisaContato.trim())
          || element['zapzap'].contains(pesquisaContato.trim())){

        tmpList.add(element);
      }
    });
```

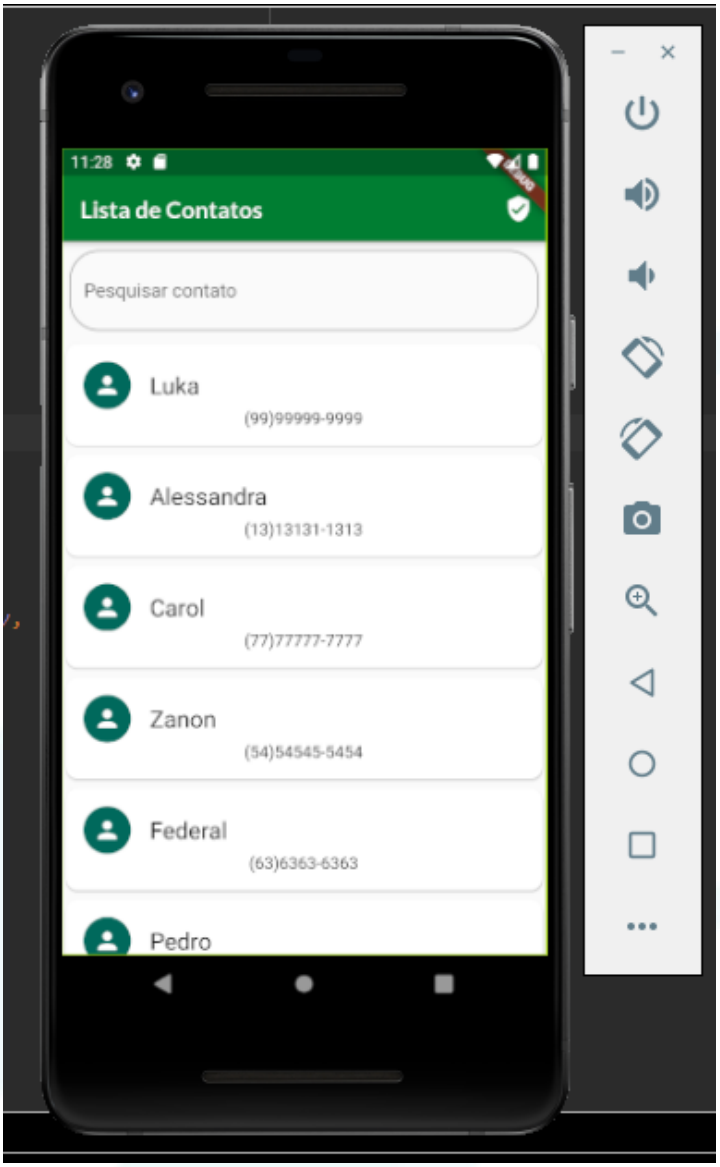
Se por acaso não houver contato correspondente ao conteúdo digitado, o curso se direciona para o else que retornará uma lista vazia como forma de avisar que nada fora encontrado.

```
      setState(() {
        Contatos.clear();
        Contatos.addAll(tmpList);
      });
      return;

    }
    else
    {
      setState(() {
        Contatos.clear();
        Contatos.addAll(Consulta);
      });
    }
  }
}
```

7. Resultado final e funcionamento

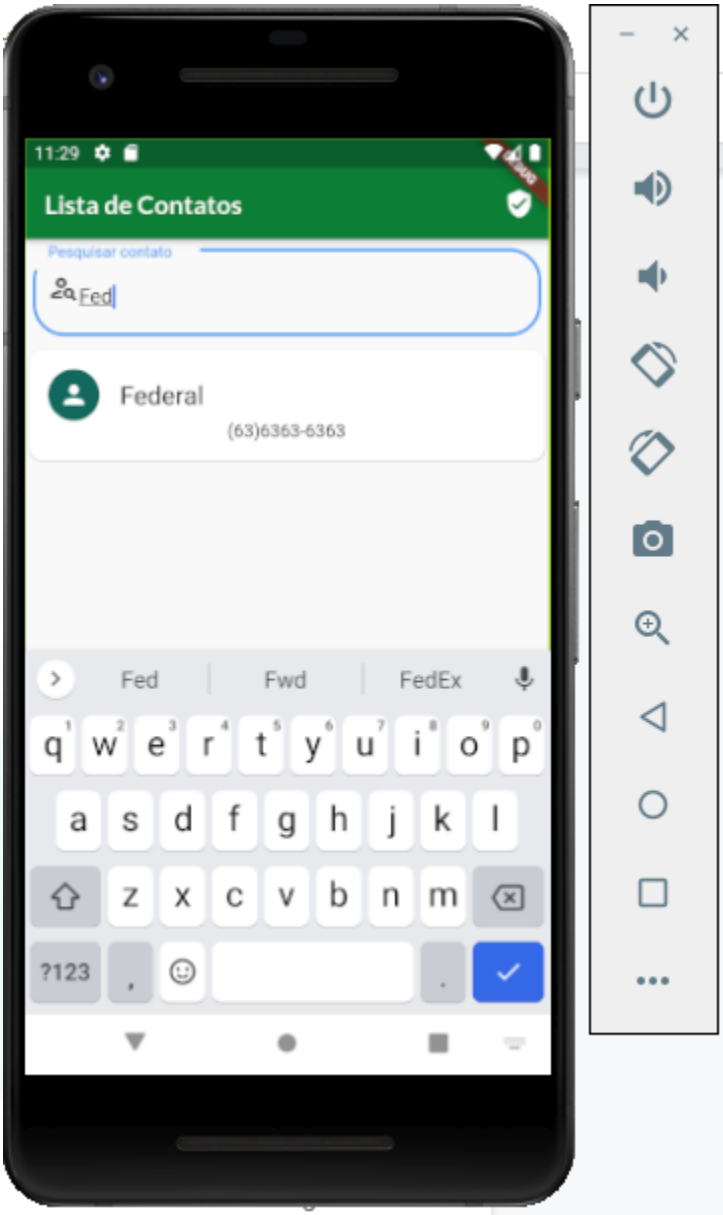
7.1.PÁGINA INICIAL

Após todo o processo de codificação, você pode se conectar a um device para emular a aplicação. Abaixo você vê a tela esperada ao rodar:



7.2.PESQUISAR CONTATO

Na barra de pesquisa que foi feita, você pode testar a aplicação enquanto ela roda, digitando os nomes dos contatos e verificando se eles existem na lista cadastrada ou não, desta forma abaixo:


11:29
Lista de Contatos
Pesquisar contato
Fed
Federal
(63)6363-6363
Fed
Fwd
FedEx
?123

A seguir, você pode observar mais dos códigos utilizados na página inicial geral da aplicação da lista de contatos:

main.dart × pubspec.yaml × Contatos.dart ×

```dart
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'Contatos.dart';

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Lista de Contatos',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),  // ThemeData
      home: MyHomePage(title: 'Lista de Contatos'),
    );  // MaterialApp
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;
```

main.dart × pubspec.yaml × Contatos.dart ×

```dart
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {

  var Contatos = [];

  @override
  void initState()
  {
    super.initState();
    // items.addAll(_searchable);
    Contatos.addAll(Consulta);
  }

```

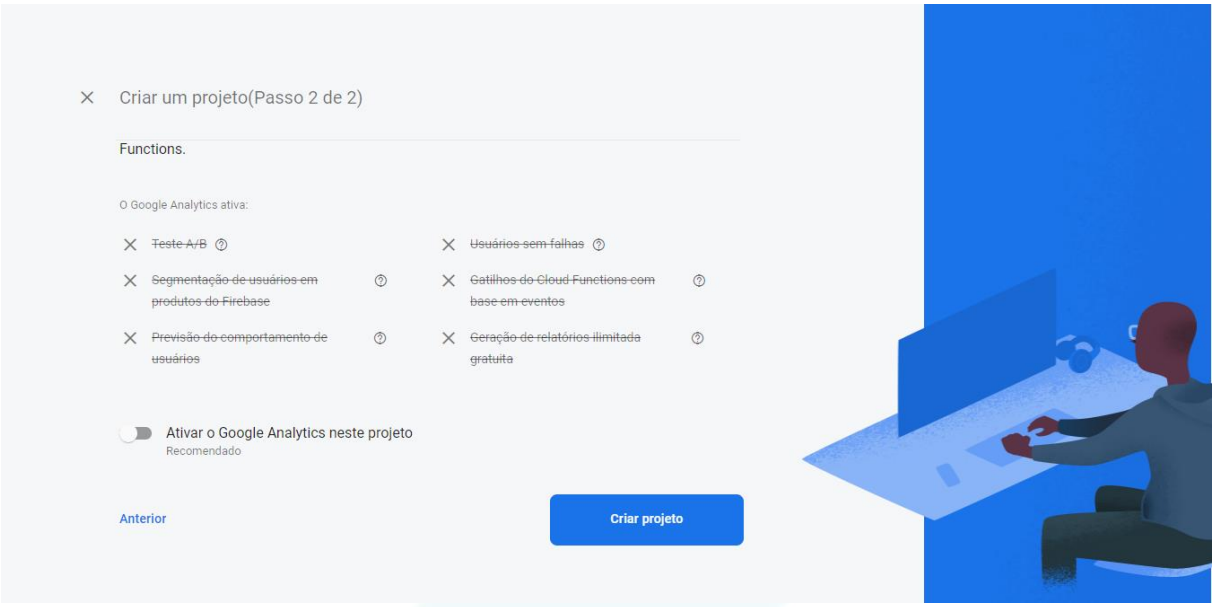
APLICAÇÃO 04 - BANCO DE DADOS E CRUD SIMPLES

Ao desenvolver a aplicação 4 do projeto em um todo, o grupo elaborou um CRUD que nos permite salvar, deletar, consultar e alterar dados em um banco de dados. Os dados manipulados trata-se da tabela de livros usada ao longo do curso, e o serviço utilizado foi o Firebase da Google, onde abaixo será mostrada sua performance passo a passo.

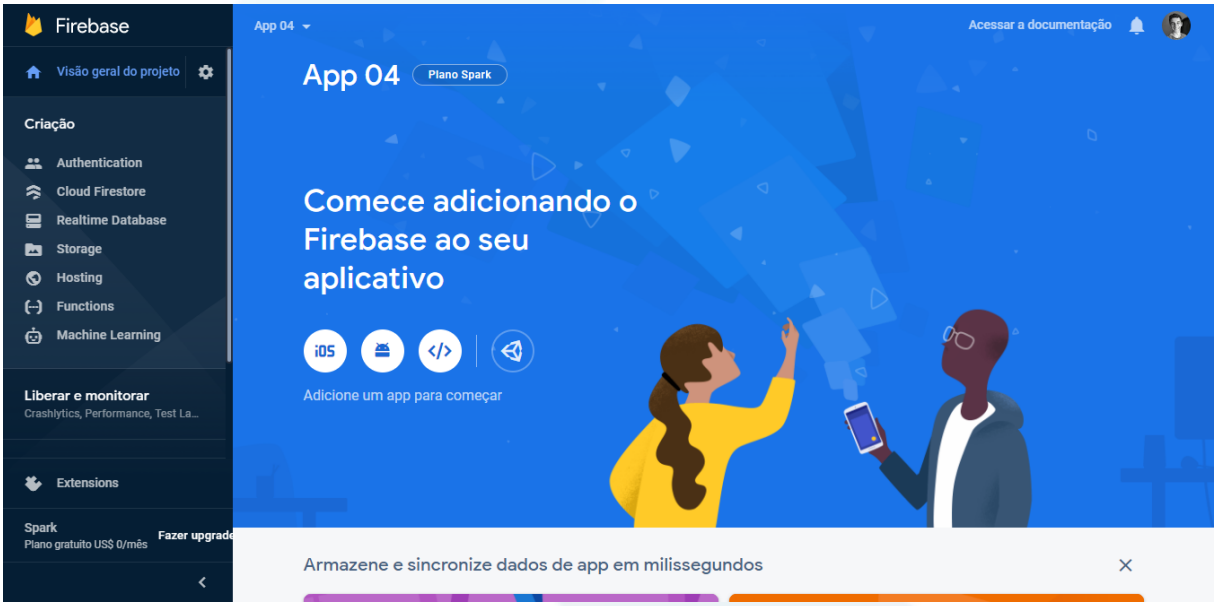
1. Quarta aplicação: CRUD para salvar, deletar, consultar e alterar dados em um banco de dados, referente a tabela de livros usada ao longo do curso:

1.1. Acessar o Firebase da Google e clicar na opção de criar um projeto.

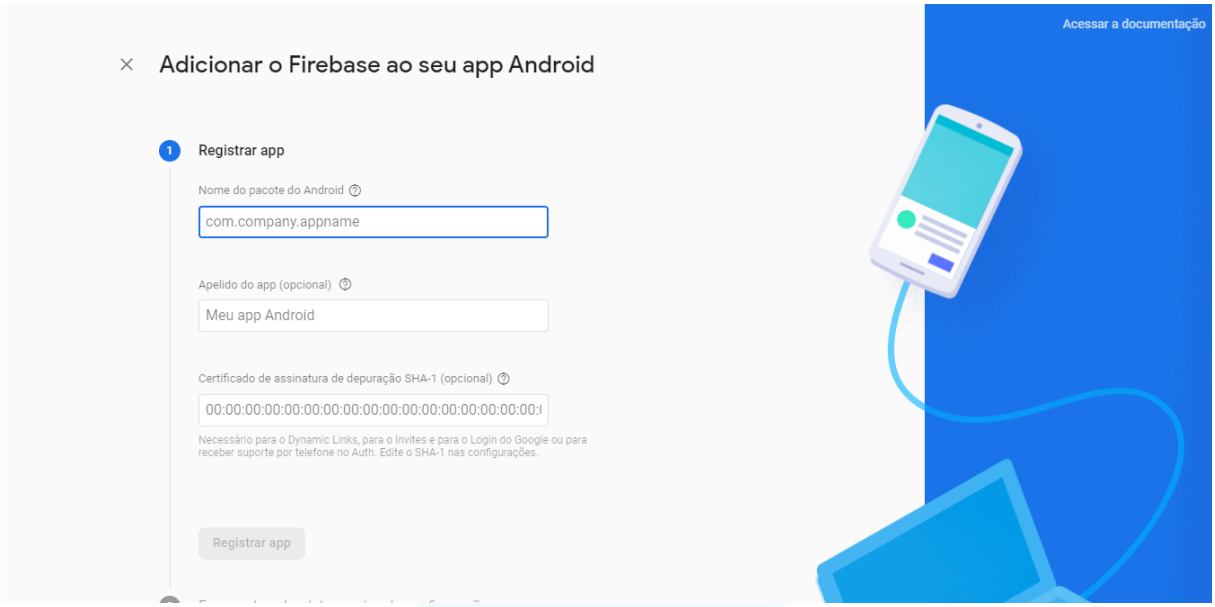
1.2. Desativar o Google Analytics, que é um sistema gratuito de monitoramento de tráfego.



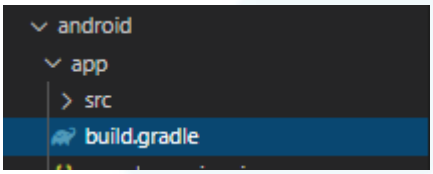
1.3. Clicar no ícone Android, que se encontra ao lado do IOS, dentre as opções.



1.4. Para preencher o primeiro text, devemos seguir o 5º passo.



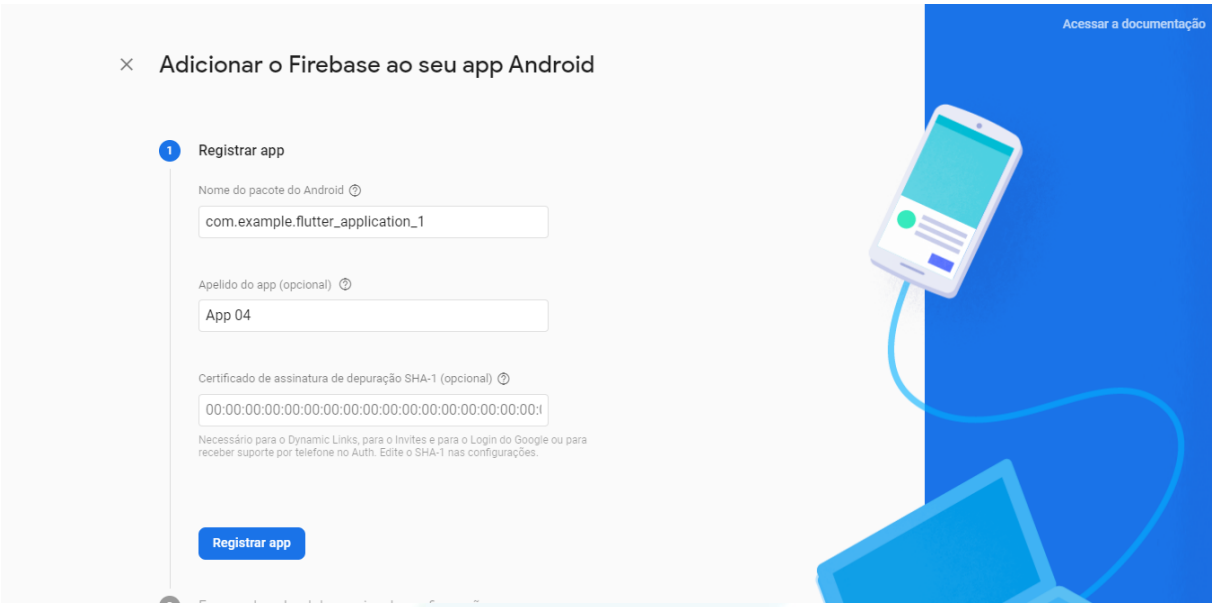
1.5. No projeto, ir em android>app>src>build.gradle



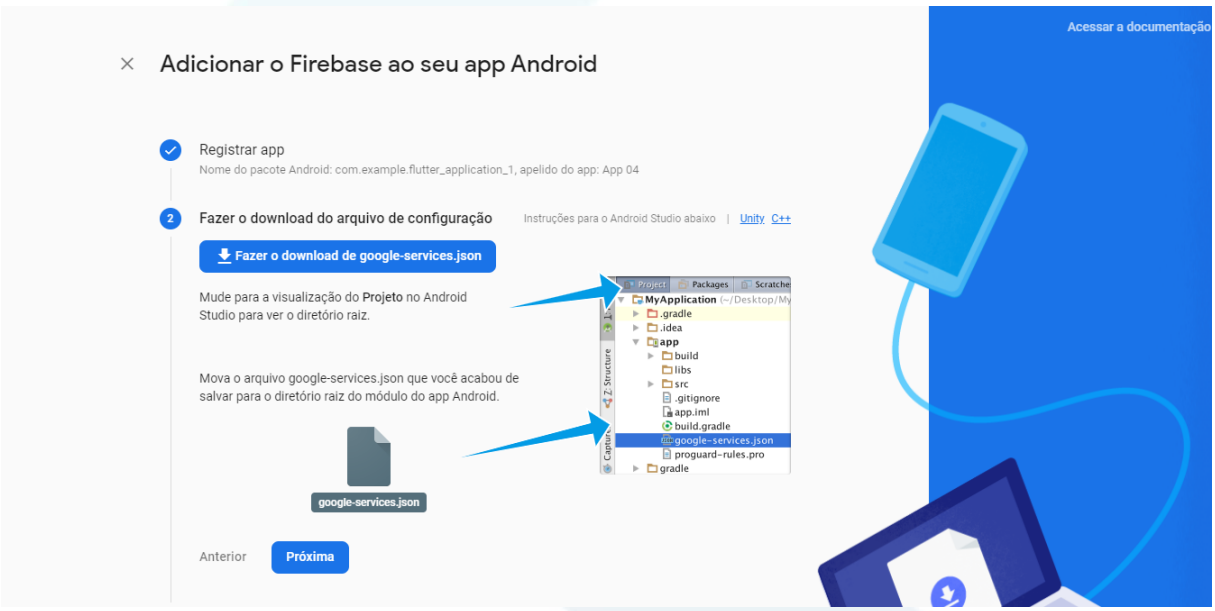
1.6. Copiar o que estiver entre aspas, na applicationId

```
applicationId "com.example.flutter_application_1"
```

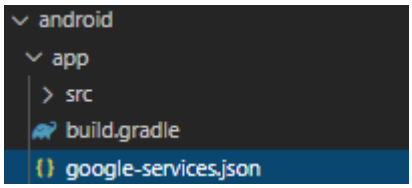
1.7. Colar no primeiro text o que estive na applicationId, e depois é necessário dar um apelido ao app, no caso, escolheu-se “App 04”. Não se precisa preencher o último espaço.



1.8. Após registrar o aplicativo, deve-se fazer o download do google-services.json, que um auxiliar de início rápido para integrar rapidamente os serviços do Google ao seu aplicativo.



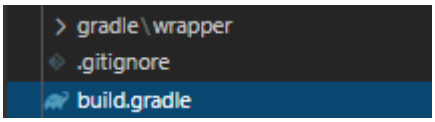
1.9. Arrastar o arquivo dentro do projeto, em android>app>src



1.10. Copiar a classpath

```
    dependencies {
      ...
      // Add this line
      classpath 'com.google.gms:google-services:4.3.5'
    }
}
```

1.11. Se direcionar ao android>build.gradle novamente



1.12. Colar a linha do 10º passo na área dependencies

```
dependencies {
    classpath 'com.android.tools.build:gradle:3.5.0'
    classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    classpath 'com.google.gms:google-services:4.3.5'
```

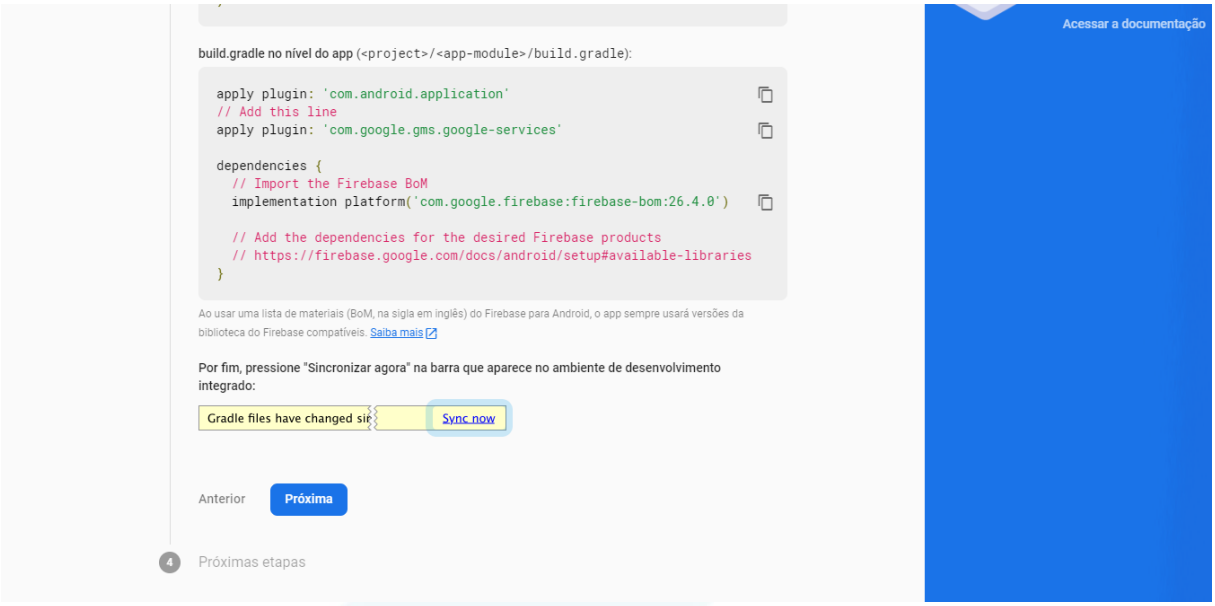
1.13. Copie o apply plugin.

```
// Add this line
apply plugin: 'com.google.gms.google-services'
```

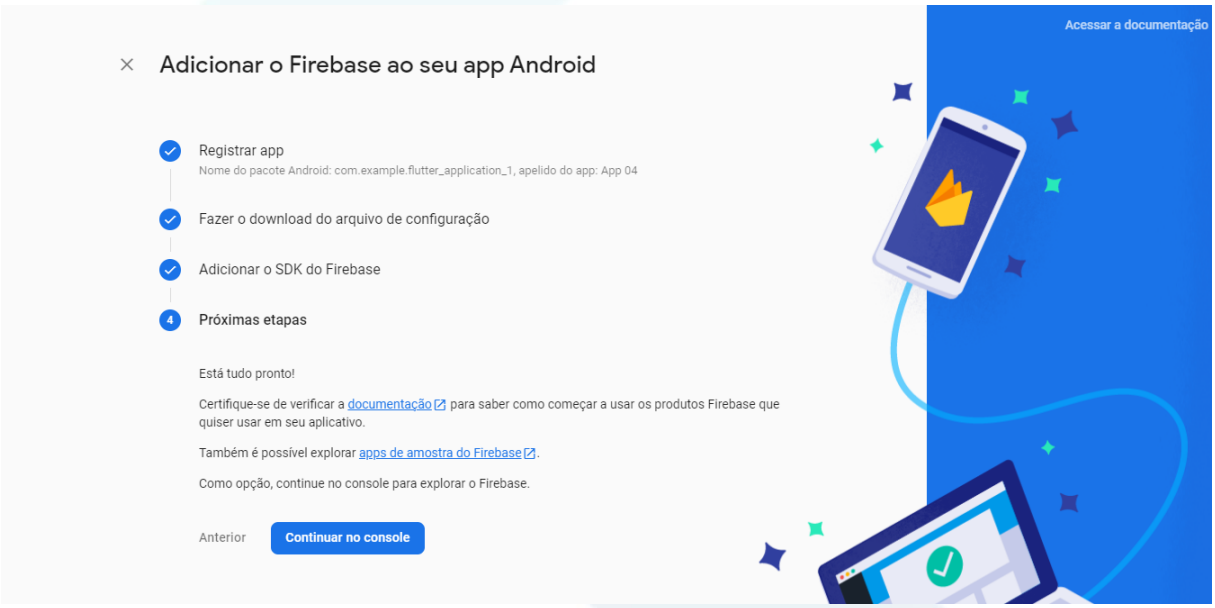
1.14. Cole o apply em android>app>src>build.gradle

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply from: "$flutterRoot/packages/flutter_tools/gradle/flutter.gradle"
apply plugin: 'com.google.gms.google-services'
```

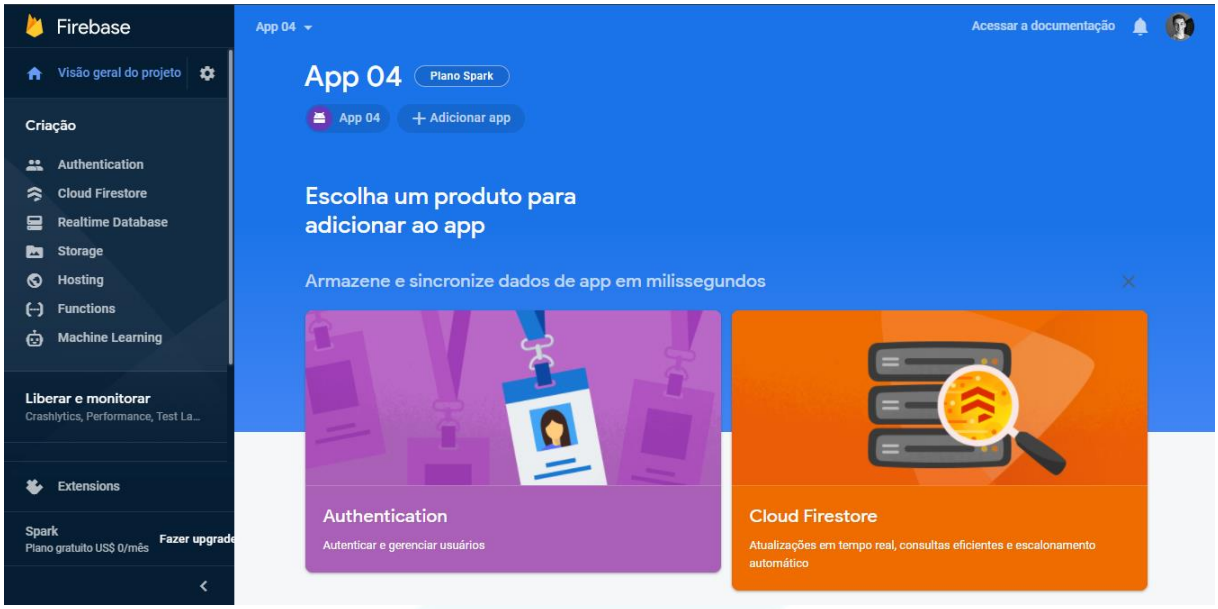
1.15. Siga clicando em "Próximas".



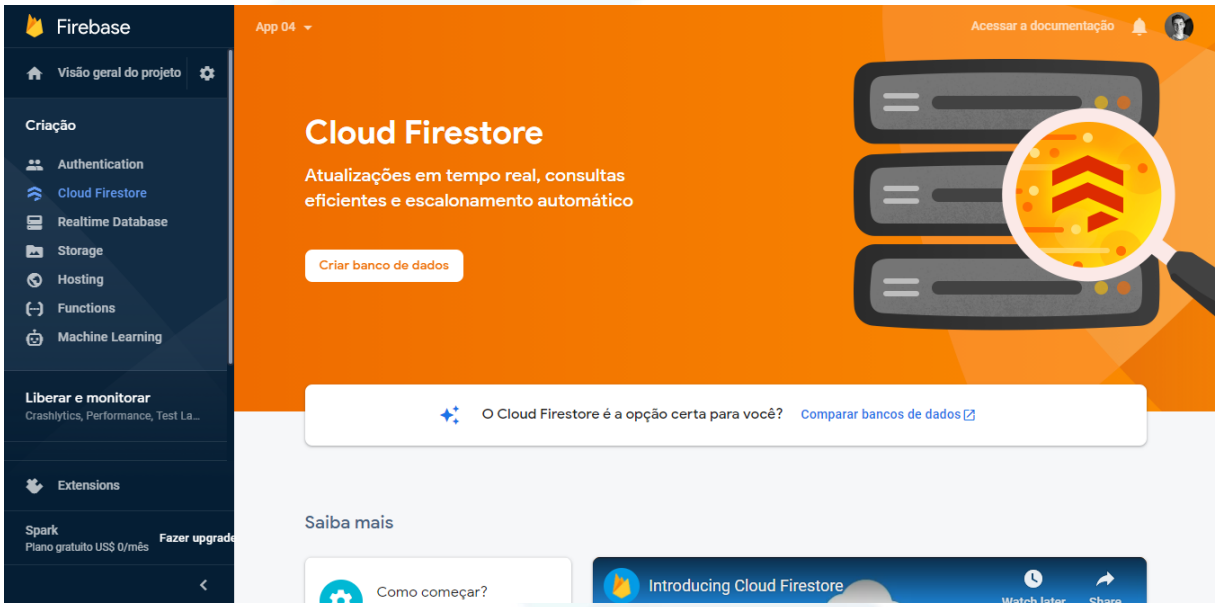
1.16. Clicar em "Continuar no Console"



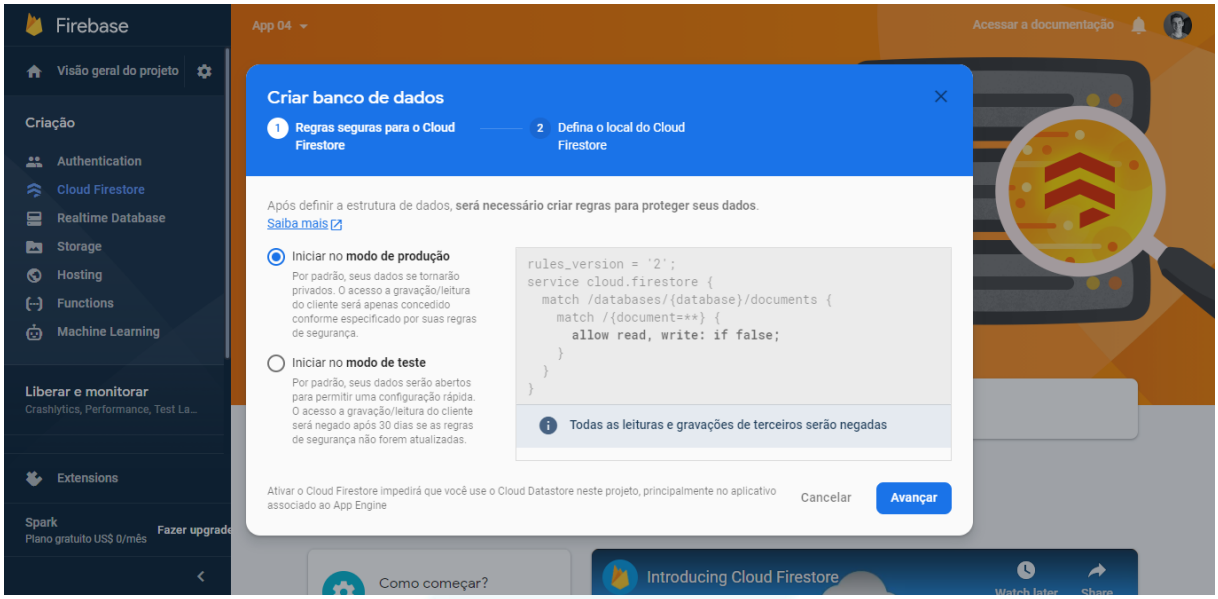
1.17. Após isso, clique em "Cloud Firestore".



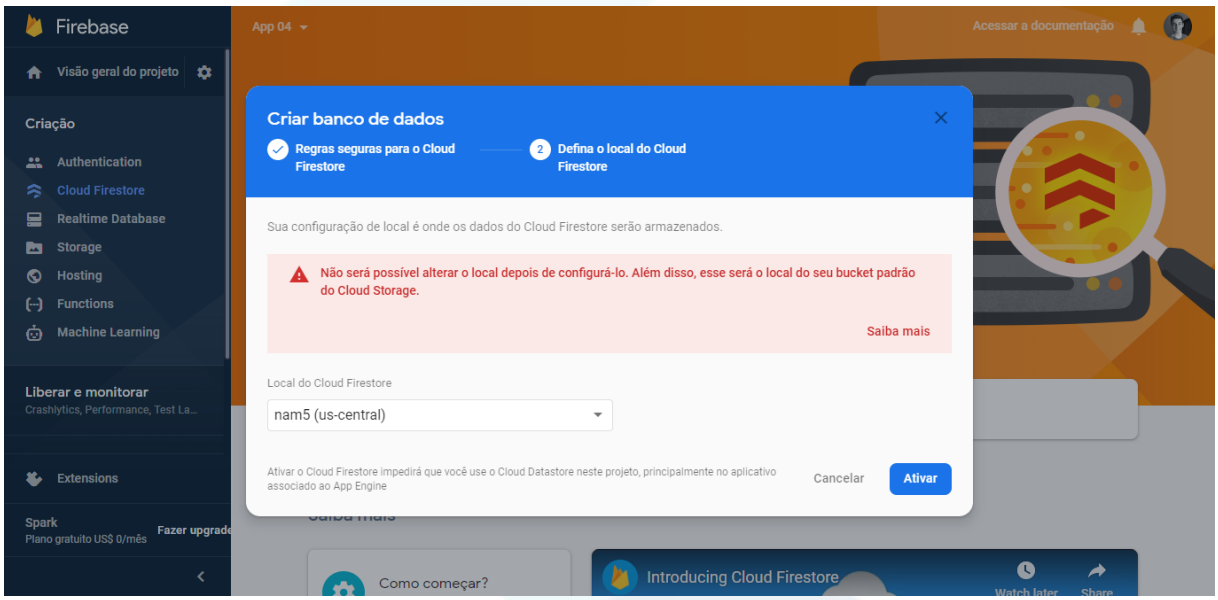
1.18. Clique em "Criar banco de dados".



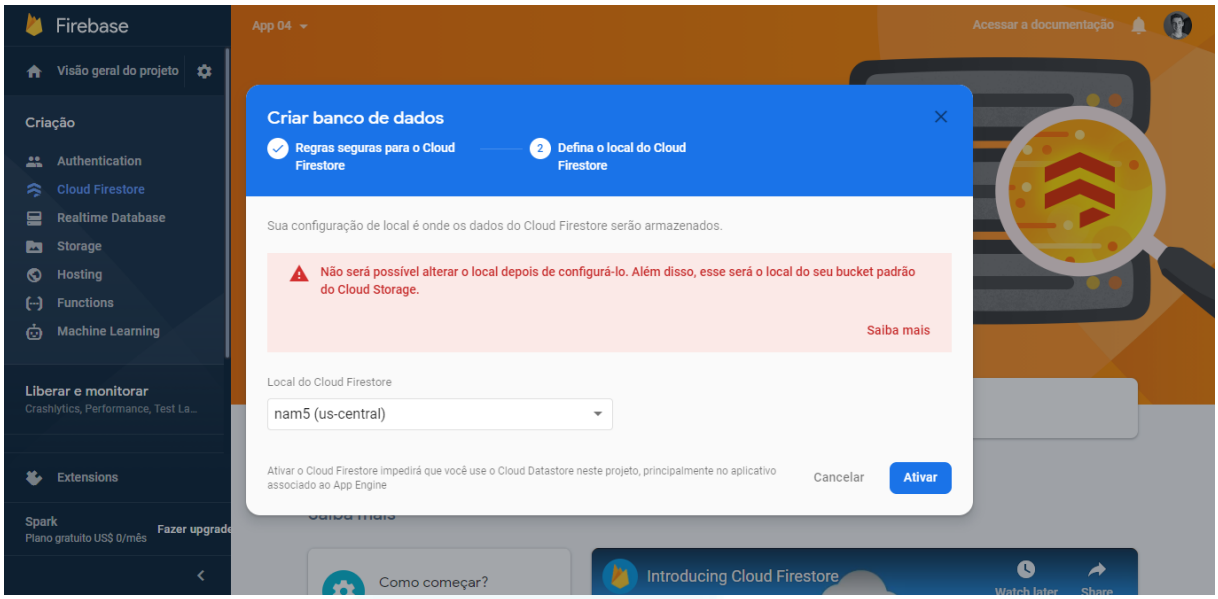
1.19. Clique em "Avançar"



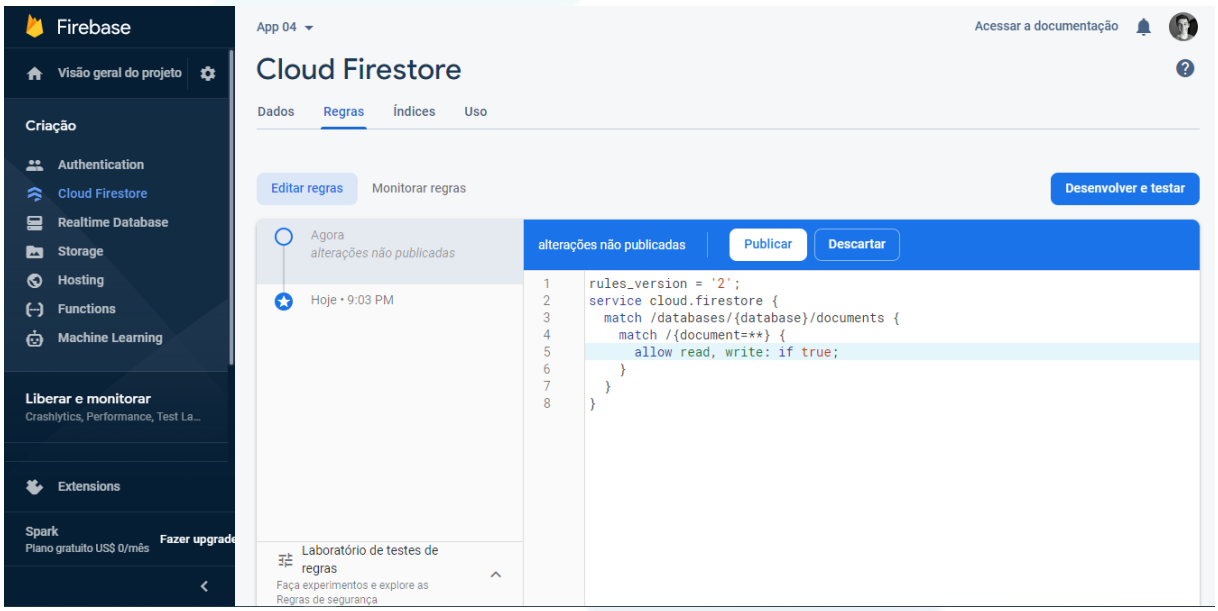
1.20. Clique em "Ativar"



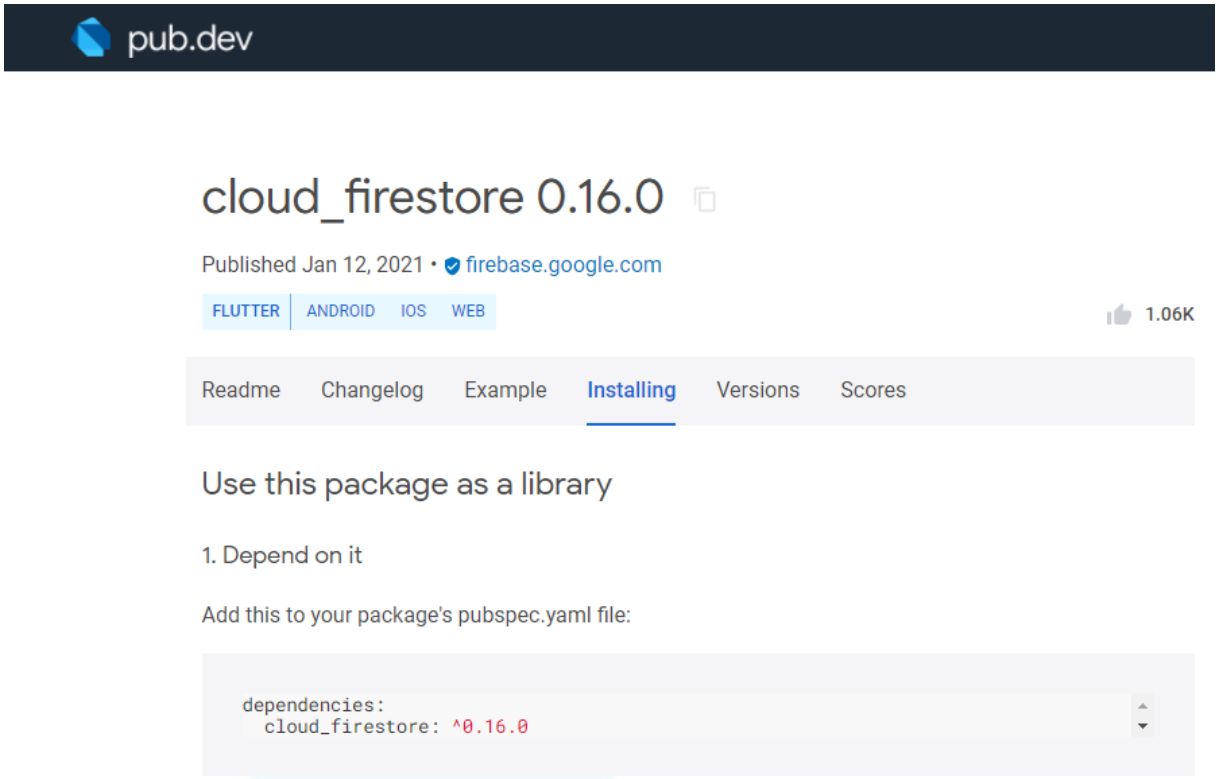
1.21. Clique em "Regras", mais acima na tela.



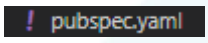
1.22. Na linha allow "read, write: if false;", mudar para "true;".



1.23. No site "pub.dev", procure por "Cloud Firestore", clique na primeira opção e vá para Installing, e em seguida copie a linha "cloud_firestore: ^0.16.0"



1.24. No projeto, vá em test>pubspec.yaml.



1.25. Cole em "dependencies:"

```
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
  cloud_firestore: ^0.16.0
  firebase_core: ^0.7.0
```

2. A seguir, mostraremos algumas imagens da parte do código, com seus devidos comentários em cada bloco:

Imagem Classe “database_services”: Classe referente ao banco de dados, efetivamente.

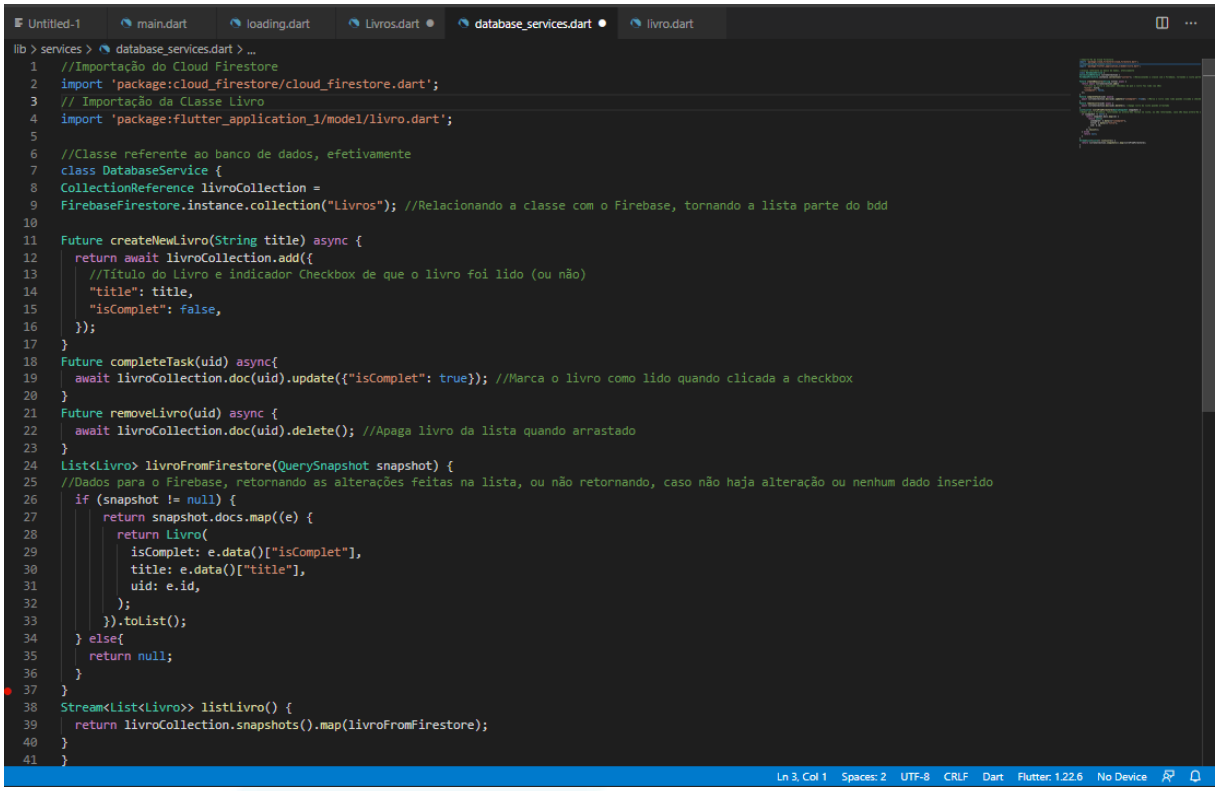
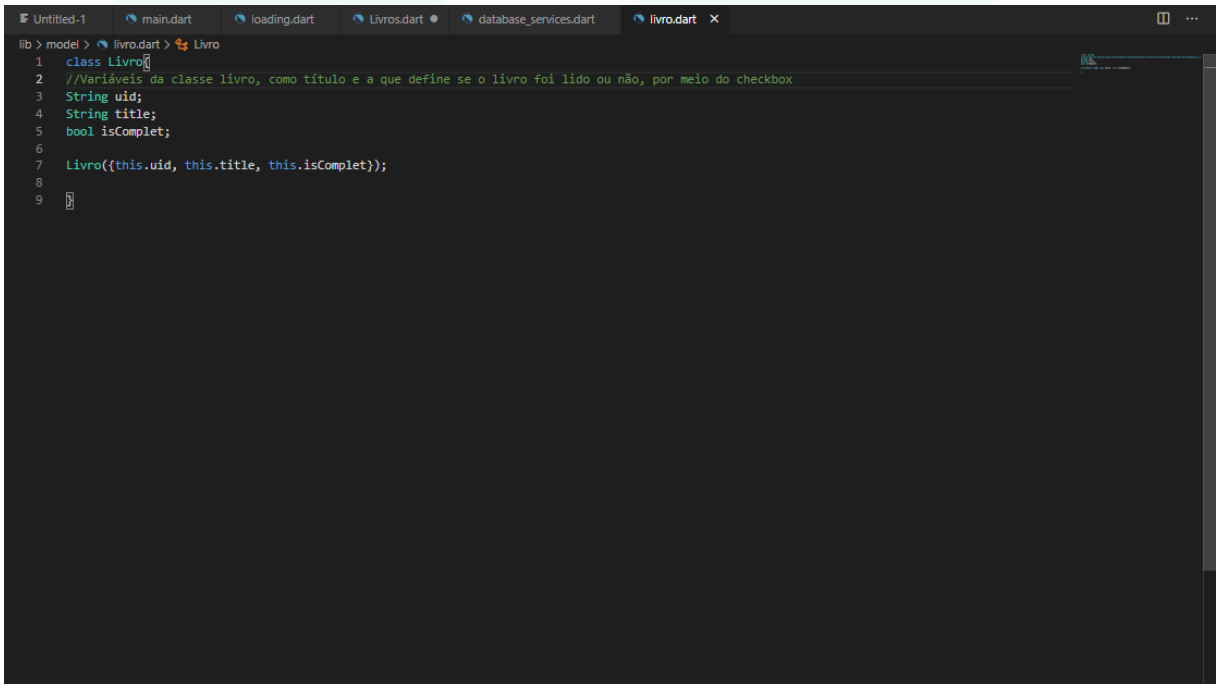
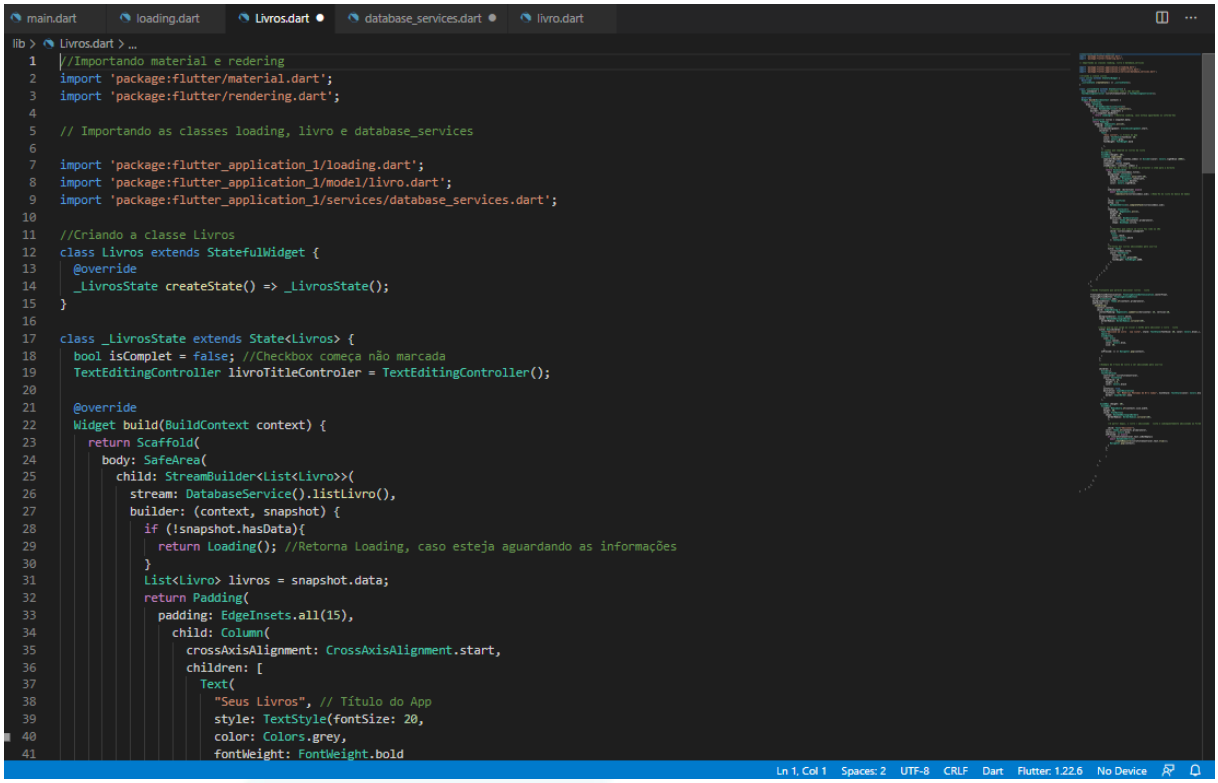
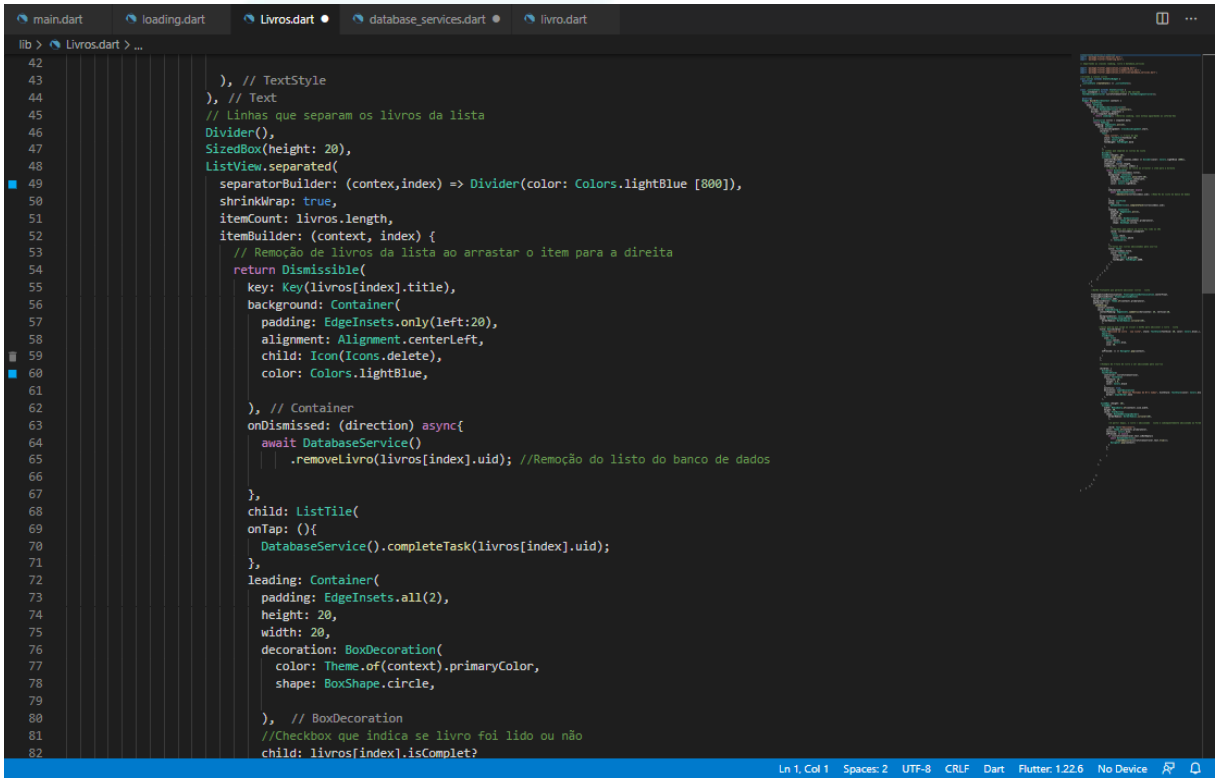


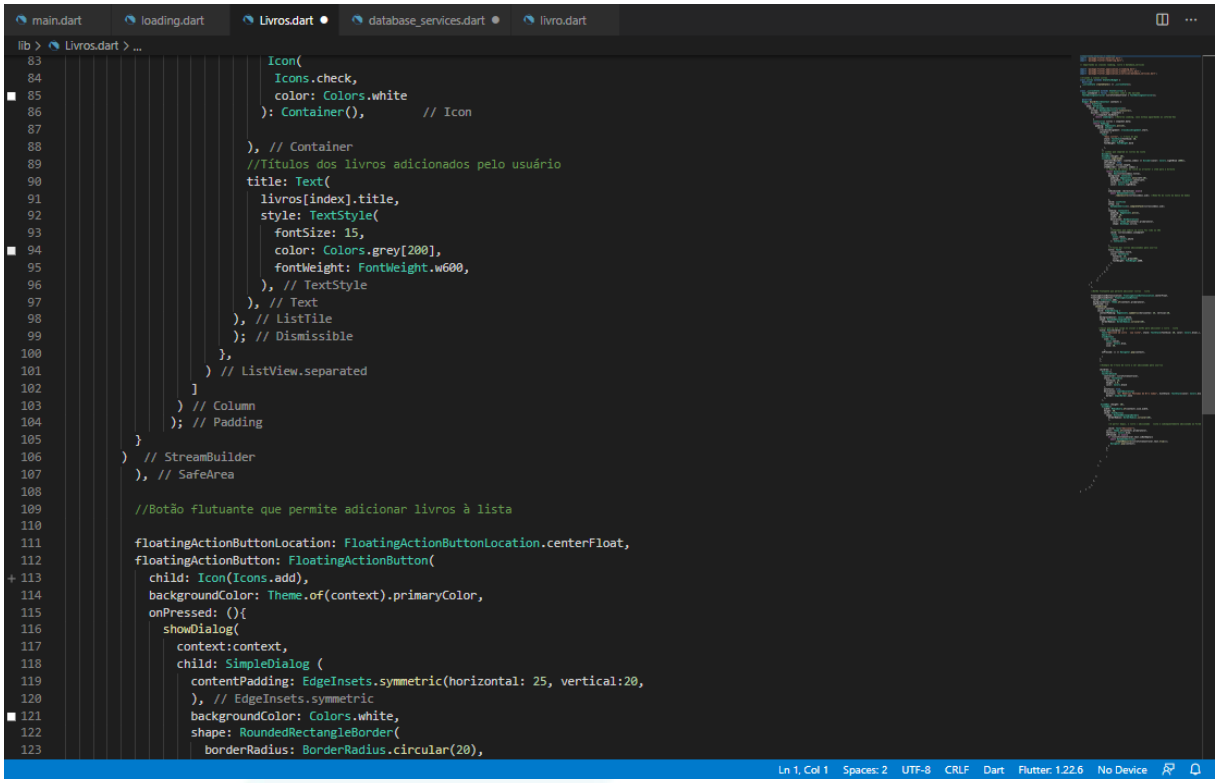
Imagem Classe “livro: Classe que define se o livro foi lido ou não.

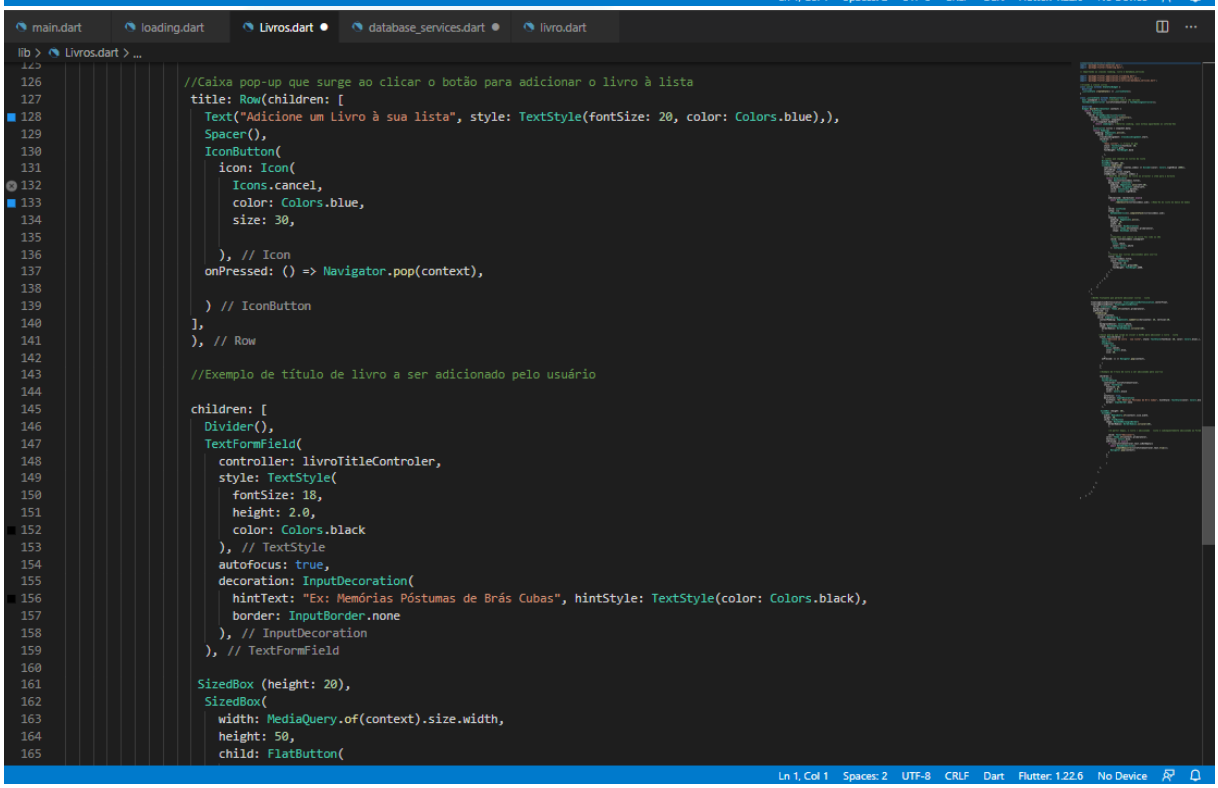


Imagens da Classe “Livros”: Classe de comandos do CRUD.


main.dart
loading.dart
Livros.dart
database_services.dart
livro.dart
lib > Livros.dart > ...
//Importando material e redering
import 'package:flutter/material.dart';
import 'package:flutter/rendering.dart';
// Importando as classes loading, livro e database_services
import 'package:flutter_application_1/loading.dart';
import 'package:flutter_application_1/model/livro.dart';
import 'package:flutter_application_1/services/database_services.dart';
//Criando a classe Livros
class Livros extends StatefulWidget {
@override
_LivrosState createState() => _LivrosState();
}
class _LivrosState extends State<Livros> {
bool isComplet = false; //Checkbox começa não marcada
TextEditingController livroTitleControler = TextEditingController();
@override
Widget build(BuildContext context) {
return Scaffold(
body: SafeArea(
child: StreamBuilder<List<Livro>>(
stream: DatabaseService().listLivro(),
builder: (context, snapshot) {
if (!snapshot.hasData){
return Loading(); //Retorna Loading, caso esteja aguardando as informações
}
List<Livro> livros = snapshot.data;
return Padding(
padding: EdgeInsets.all(15),
child: Column(
crossAxisAlignment: CrossAxisAlignment.start,
children: [
Text(
"Seus Livros", // Título do App
style: TextStyle(fontSize: 20,
color: Colors.grey,
fontWeight: FontWeight.bold
Ln 1, Col 1 Spaces: 2 UTF-8 CRLF Dart Flutter: 1.22.6 No Device


main.dart
loading.dart
Livros.dart
database_services.dart
livro.dart
lib > Livros.dart > ...
), // TextStyle
), // Text
// Linhas que separam os livros da lista
Divider(),
SizedBox(height: 20),
ListView.separated(
separatorBuilder: (contex,index) => Divider(color: Colors.lightBlue [800]),
shrinkWrap: true,
itemCount: livros.length,
itemBuilder: (context, index) {
// Remoção de livros da lista ao arrastar o item para a direita
return Dismissible(
key: Key(livros[index].title),
background: Container(
padding: EdgeInsets.only(left:20),
alignment: Alignment.centerLeft,
child: Icon(Icons.delete),
color: Colors.lightBlue,
), // Container
onDismissed: (direction) async{
await DatabaseService()
.removeLivro(livros[index].uid); //Remoção do listo do banco de dados
},
child: ListTile(
onTap: (){
DatabaseService().completeTask(livros[index].uid);
},
leading: Container(
padding: EdgeInsets.all(2),
height: 20,
width: 20,
decoration: BoxDecoration(
color: Theme.of(context).primaryColor,
shape: BoxShape.circle,
), // BoxDecoration
//Checkbox que indica se livro foi lido ou não
child: livros[index].isComplet?
Ln 1, Col 1 Spaces: 2 UTF-8 CRLF Dart Flutter: 1.22.6 No Device


main.dart
loading.dart
Livros.dart
database_services.dart
livro.dart
lib > Livros.dart > ...
Icon(
Icons.check,
color: Colors.white
): Container(), // Icon
), // Container
//Títulos dos livros adicionados pelo usuário
title: Text(
livros[index].title,
style: TextStyle(
fontSize: 15,
color: Colors.grey[200],
fontWeight: FontWeight.w600,
), // TextStyle
), // Text
), // ListTile
); // Dismissible
},
) // ListView.separated
]
) // Column
); // Padding
}
) // StreamBuilder
), // SafeArea
//Botão flutuante que permite adicionar livros à lista
floatingActionButtonLocation: FloatingActionButtonLocation.centerFloat,
floatingActionButton: FloatingActionButton(
child: Icon(Icons.add),
backgroundColor: Theme.of(context).primaryColor,
onPressed: (){
showDialog(
context:context,
child: SimpleDialog (
contentPadding: EdgeInsets.symmetric(horizontal: 25, vertical:20,
), // EdgeInsets.symmetric
backgroundColor: Colors.white,
shape: RoundedRectangleBorder(
borderRadius: BorderRadius.circular(20),
Ln 1, Col 1 Spaces: 2 UTF-8 CRLF Dart Flutter: 1.22.6 No Device


main.dart
loading.dart
Livros.dart
database_services.dart
livro.dart
lib > Livros.dart > ...
//Caixa pop-up que surge ao clicar o botão para adicionar o livro à lista
title: Row(children: [
Text("Adicione um Livro à sua lista", style: TextStyle(fontSize: 20, color: Colors.blue),),
Spacer(),
IconButton(
icon: Icon(
Icons.cancel,
color: Colors.blue,
size: 30,
), // Icon
onPressed: () => Navigator.pop(context),
) // IconButton
],
), // Row
//Exemplo de título de livro a ser adicionado pelo usuário
children: [
Divider(),
TextFormField(
controller: livroTitleControler,
style: TextStyle(
fontSize: 18,
height: 2.0,
color: Colors.black
), // TextStyle
autofocus: true,
decoration: InputDecoration(
hintText: "Ex: Memórias Póstumas de Brás Cubas", hintStyle: TextStyle(color: Colors.black),
border: InputBorder.none
), // InputDecoration
), // TextFormField
SizedBox (height: 20),
SizedBox(
width: MediaQuery.of(context).size.width,
height: 50,
child: FlatButton(
Ln 1, Col 1 Spaces: 2 UTF-8 CRLF Dart Flutter: 1.22.6 No Device

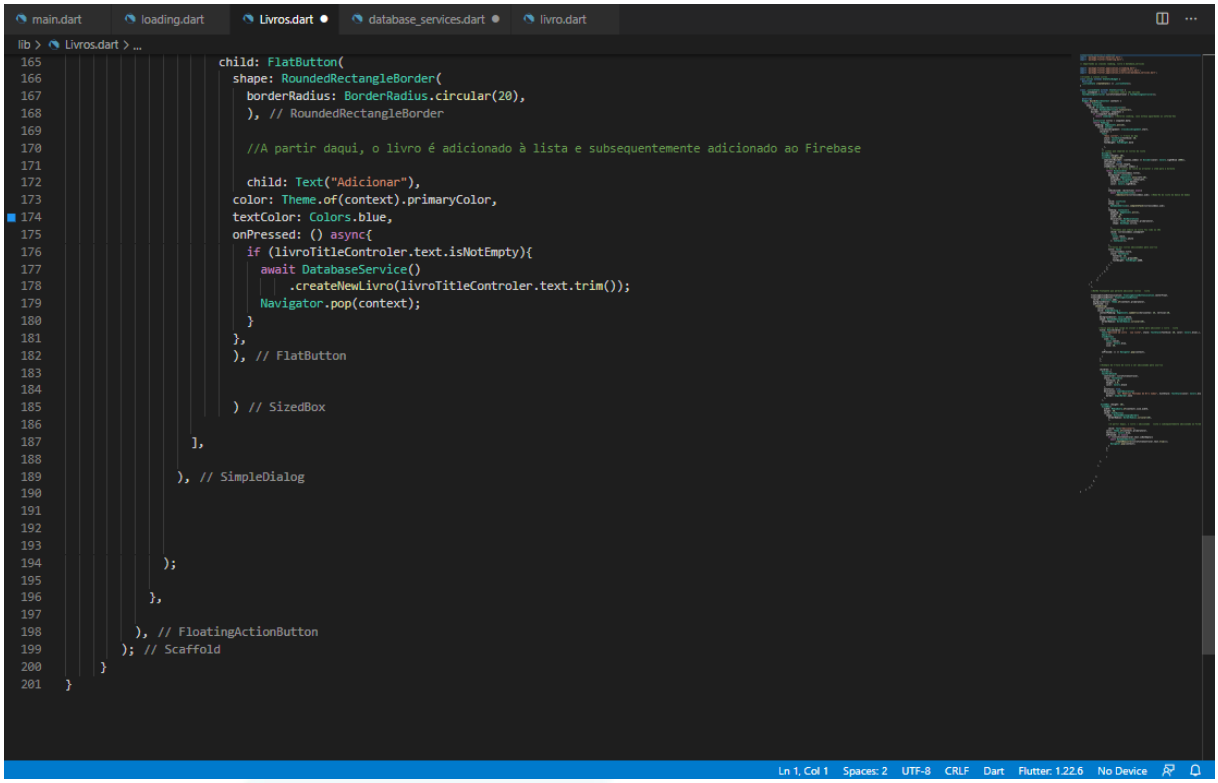


Imagem Classe “loading”: Classe que define que os dados estão chegando à tela, utilizada em main.

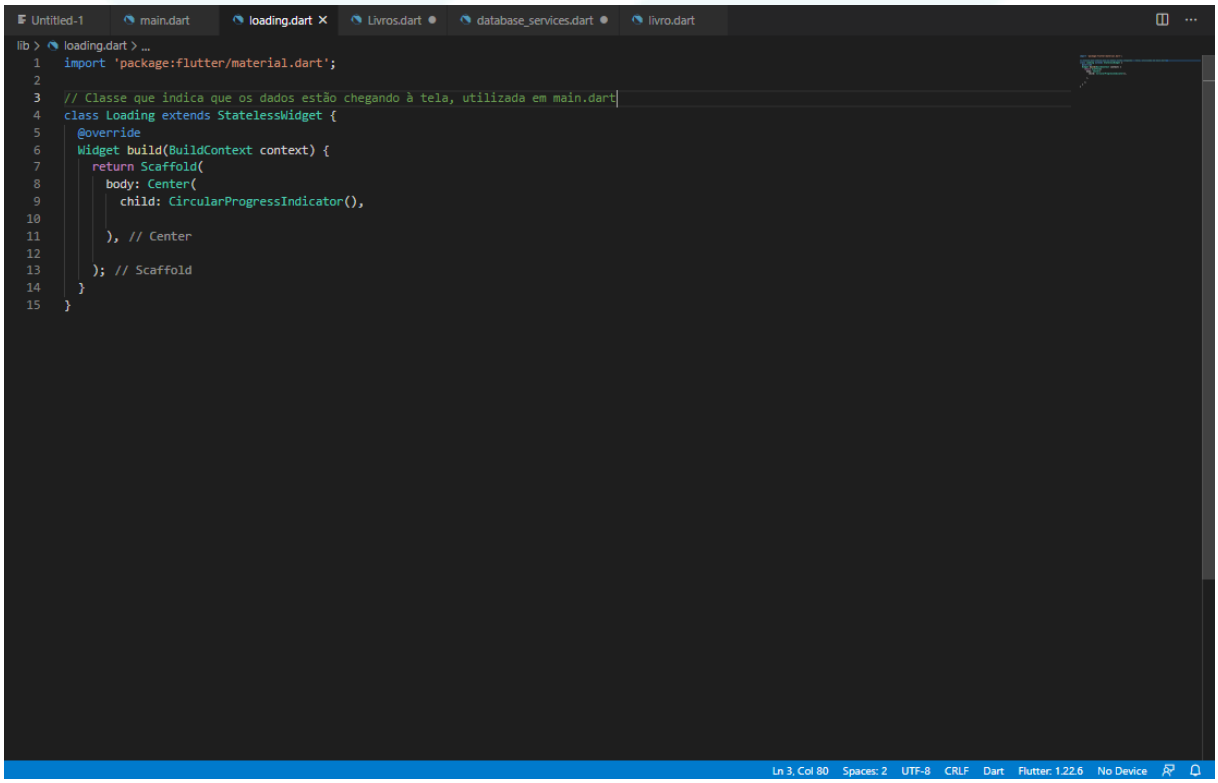
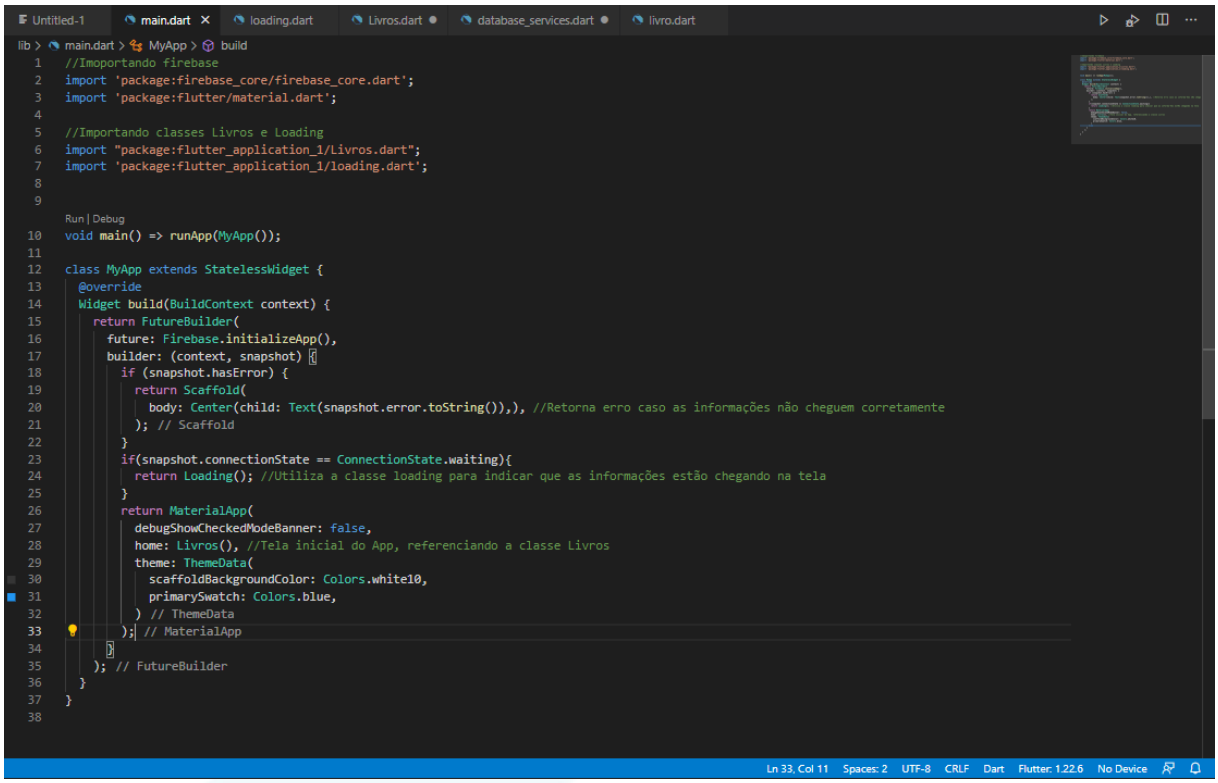


Imagem Classe “main”: Utiliza das classes criadas para o funcionamento do aplicativo.


Untitled-1
main.dart
loading.dart
Livros.dart
database_services.dart
livro.dart
lib > main.dart > MyApp > build
//Imoportando firebase
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
//Importando classes Livros e Loading
import "package:flutter_application_1/Livros.dart";
import 'package:flutter_application_1/loading.dart';
Run | Debug
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return FutureBuilder(
future: Firebase.initializeApp(),
builder: (context, snapshot) {
if (snapshot.hasError) {
return Scaffold(
body: Center(child: Text(snapshot.error.toString())), //Retorna erro caso as informações não cheguem corretamente
); // Scaffold
}
if(snapshot.connectionState == ConnectionState.waiting){
return Loading(); //Utiliza a classe loading para indicar que as informações estão chegando na tela
}
return MaterialApp(
debugShowCheckedModeBanner: false,
home: Livros(), //Tela inicial do App, referenciando a classe Livros
theme: ThemeData(
scaffoldBackgroundColor: Colors.white10,
primarySwatch: Colors.blue,
) // ThemeData
); // MaterialApp
}
); // FutureBuilder
}
}
Ln 33, Col 11
Spaces: 2
UTF-8
CRLF
Dart
Flutter: 1.22.6
No Device

CONSIDERAÇÕES FINAIS

Ufa! Finalmente finalizamos todo o conteúdo da nossa apostila de Flutter! Esperamos, com toda a felicidade do mundo, que você tenha adquirido um rico conhecimento nessa framework da Google tão promissora!

Esse projeto foi titulado como um trabalho de conclusão de curso (TCC) a ser apresentado na nossa formatura do curso técnico integrado de Informática, no campus do IFSP Cubatão. Então, além da extrema responsabilidade de reunir os nossos aprendizados e transmiti-los nesse documento, também ficamos cientes do compromisso em espalhar todo o conhecimento de Flutter e aplicação mobile para os não veteranos nesse assunto.

Nós, do grupo Desenvolvimento Flutter, estamos deveras orgulhosos de você e da sua caminhada para chegar até aqui. Na verdade, da *nossa* caminhada, pois evoluímos a apostila inteira juntinhos: nós, na aprendizagem e montagem da apostila; você, na leitura, prática e aprendizado dos conhecimentos aqui descritos.

Desejamos todo o sucesso para a sua vida e agradecemos a atenção e o apoio que você empenhou nessa jornada! Até uma próxima aventura de programação.

Com todo o carinho do mundo,

Caroline, Camila, Elienai, Emanuelle, Gabriel, Geovanna, João, Laura, Milena e Pedro.
Grupo Flutter, CTII 448.

REFERÊNCIAS BIBLIOGRÁFICAS

Developers Android. Disponível em: https://developer.android.com/studio. Acesso em 27 de setembro de 2020.

Wikipédia. Disponível em: https://pt.wikipedia.org/wiki/Android_Studio. Acesso em 27 de setembro de 2020.

Visual Studio Code. Disponível em: https://code.visualstudio.com/docs. Acesso em 27 de setembro de 2020.

Médium. Disponível em: https://medium.com. Acesso em 27 de setembro de 2020.

Flutter. Disponível em: https://flutter.dev/docs/get-started. Acesso em 27 de setembro de 2020.

ALENCAR, Jayr. "O que é Flutter?" Clube dos Geeks. Disponível em: http://clubedosgeeks.com.br/programacao/o-que-e-o-flutter. Acesso em 27 de setembro de 2020.

MELO, Rubens de. "O que é Flutter?" Flutter para iniciantes. Disponível em: https://www.flutterparainiciantes.com.br/o-que-e-flutter. Acesso em 27 de setembro de 2020.

ANDRADE, Ana Paula de. "O que é Flutter?"; Blog da TreinaWeb. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-flutter/. Acesso em 27 de setembro de 2020.

Canal “Code like Ice” no Youtube. “*Flutter - Contact List App - Part 1*”. Disponível em: https://www.youtube.com/watch?v=cAQVNn6pCq8. Acesso feito em 09 de janeiro de 2021.

FLUTTER
PARA INICIANTES

GUIA PRÁTICO
COM PASSO A PASSO